UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA PETROLERA

DESARROLLO DE UN SOFTWARE PARA ADMINISTRAR LA INFORMACIÓN REQUERIDA EN EL DESPACHO DE GNL

PROYECTO DE GRADO PARA OBTENER EL TÍTULO DE LICENCIATURA EN INGENIERÍA EN GAS, PETRÓLEO Y PROCESOS

PRESENTADO POR:

UNIV. EVELYN CLAUDIA QUINTEROS SORIA

TUTOR: ING. JIMMY GUILLEN ESPINOZA

LA PAZ - BOLIVIA

2022

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERIA

LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) Visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) Copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) Copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la cita o referencia correspondiente en apego a las normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

DEDICATORIA.

A Dios y a mis padres.

A Dios porque me acompaña en cada paso que doy, cuidándome y dándome fortaleza para continuar, a mis padres, quienes a lo largo de mi vida me apoyaron en cada uno de mis sueños velando por mi bienestar y educación. Depositando su entera confianza en cada reto que se me presentaba sin dudar ni un solo momento en mi inteligencia y capacidad.

AGRADECIMIENTO.

A Dios por estar siempre presente en mi vida, por no dejarme caer nunca y mantenerme firme en el camino hacia mis metas.

A mis padres y hermanas por el apoyo incondicional, la paciencia y el amor durante todos estos años.

A mis amigos de la universidad y a la Facultad de Ingeniería de la UMSA por haberme acogido durante todos los años de mi formación académica.

RESUMEN

La constante migración a nuevas tecnologías y la automatización de los procesos de manejo de datos, que garanticen a los operadores el copiado correcto y la toma de decisiones más apropiadas para sus tareas, ha influenciado cada vez más a las grandes empresas. Es por tanto que se conoce que uno de los mayores beneficios de utilizar un software es la facilidad de tomar decisiones debido a que se cuenta con los datos de la institución en el orden requerido. Con la disponibilidad de la información se tiene la facilidad de elaboración de informes y el ahorro de tiempo en cuanto al manejo de la amplia base de datos que se presenta.

El presente Proyecto de Grado se desarrolla para la Agencia Nacional de Hidrocarburos, quién tiene la misión de Regular, Supervisar, Controlar y Fiscalizar con eficacia todas las actividades, siendo una de ellas la de distribución de GNL por el territorio Nacional, para ello se encararon las necesidades y requisitos para diseñar el programa propuesto.

El proyecto encara el problema de copiado manual que realizan, automatizando este proceso para su administración y la facilidad en cuanto la visualización para la toma de decisiones para la generación de reportes que realizan como institución, teniendo presente toda la base de datos y los filtrados de fácil manejo para manipulación de la misma. Se implementa también un estudio en cuanto al comportamiento que presentan las Estaciones de Regasificación en su consumo, para determinar mediante redes neuronales las predicciones futuras, generando así una mejor coordinación en cuanto a la toma de decisiones para envío de cisternas a las distintas localidades, ayudando al usuario a crear una lista de prioridades para el envío, tanto mediante la visualización de los datos en el software y mediante las predicciones presentadas como ayuda.

El programa está compuesto por un entorno gráfico agradable al usuario basado en la biblioteca Tkinter utilizando el lenguaje de programación Python, que permite la interacción con las diferentes funcionalidades del software.

ABSTRACT.

The constant migration to new technologies and the automation of data management processes, which guarantee operators the correct copying and making the most appropriate decisions for their tasks, has increasingly influenced large companies. It is therefore known that one of the greatest benefits of using software is the ease of making decisions due to the fact that the institution's data is available in the required order. With the availability of the information, there is the ease of preparing reports and saving time in terms of managing the extensive database that is presented.

This Degree Project is developed for the National Hydrocarbons Agency (ANH), which has the mission to effectively Regulate, Supervise, Control and Supervise all activities, one of them being the distribution of LNG throughout the National territory. the needs and requirements to design the proposed program.

The project addresses the problem of manual copying that they carry out, automating this process for its administration and the ease in terms of visualization for decision-making and generation of reports that they carry out as an institution, keeping in mind the entire database and easily filtered management for manipulation of information. A study is also implemented regarding the behavior that Regasification Stations present in their consumption, to determine future predictions through neural networks, thus generating better coordination in terms of decision-making to send cisterns to different locations, helping Create a priority list for shipping, both by viewing the data in the software and by using the predictions presented as an aid.

The program is composed of a user-friendly graphical environment based on the Tkinter library using the Python programming language, which allows interaction with the different functionalities of the software.

GLOSARIO DE SIGLAS Y ABREVIATURAS

ANH	Agencia Nacional de Hidrocarburos
BDD	Base de Datos
DMC	Data Mining Consulting
ER	Estación de Regasificación
ESR	Estación Satélite de Regasificación
GASYRG	Gasoducto Yacuiba – Río Grande
GLP	Gas Licuado de Petróleo
GNL	Gas Natural Licuado
GUI	Graphical User Interface
LSTM	Long Short Term Memory
MH	Ministerio de Hidrocarburos
MMpcd	Millones de pies cúbicos día
PGNL	Planta de Gas Natural Licuado
POO	Programación Orientada a Objetos
PSL	Planta Separadora de Líquidos
RN	Redes Neuronales
RNN	Red Neuronal Recurrente
Sm^3	Metros Cúbicos Estándar

VSC	Visual Studio Code
YPFB	Yacimientos Petrolíferos Fiscales Bolivianos

ÍNDICE

CAPÍTULO 1 INTRODUCCIÓN

CAPÍTULO 2 MARCO TEÓRICO

CAPÍTULO 7 BIBLIOGRAFÍA

CAPÍTULO 8 ANEXOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

INTRODUCCIÓN

1 Introducción

El presente proyecto pretende crear una herramienta útil que apoye la coordinación de la programación de entregas del producto GNL a cada ESR (Estación satelital de Regasificación), las mismas abastecen la demanda de cada población (27 en la actualidad), mediante el desarrollo de un software que facilite al operador la toma de decisiones para el envío de cada unidad de transporte a las Plantas Regasificadoras.

Se pretende lograr el objetivo mediante el uso de librerías en el lenguaje de programación Python, el mismo que es un lenguaje paradigmático, multiplataforma de código libre y abierto, cualidades que previo estudio y selección son propicias para lograr el objetivo [González Duque, 2000].

Actualmente el manejo de la logística comienza con el envío rutinario de un archivo Excel como “Planilla de Reporte (ESR)” a la Planta de GNL, se tomará ese nodo inicial como punto de partida para el proyecto, el mismo que registrará los siguientes datos que son recogidos de las Plantas Regasificadoras:

- Fecha
- Usuarios
- Saldo Inicial (Sm^3)
- Recibido de Planta (Sm^3)
- Entrega de gas (Sm3)
- Saldo Final (Sm^3)

En ese contexto, lo que se pretende realizar es el uso de formularios que el operador de cada planta ESR registrará, para posteriormente ser enviados a la Planta de GNL y descargado al software, para que este realice una lectura ordenada y facilite la administración de los datos.

1.1 Antecedentes

Según el reporte de SENER (Grupo Privado de Ingeniería y Tecnología) [SENER, 2016], a principios de 2016 fue inaugurado el gasoducto virtual de Bolivia, en un acto en Santa Cruz de la Sierra, en la región de Rio Grande. SENER ha llevado a cabo el diseño, construcción y puesta en marcha el gasoducto virtual, se trata de un sistema que comprende una planta de gas natural licuado (GNL), una flota de cisternas criogénicas y estaciones satélites de re- gasificación. La construcción y la puesta en marcha de la Planta de GNL en Bolivia marcó sin duda el desarrollo del país, facilitando el transporte de Gas Natural sin importar las distancias que existan entre los consumidores y los productores, impulsando el desarrollo de las poblaciones donde no se cuentan con gasoductos convencionales, promoviendo el uso del gas natural y logrando sustituir el consumo de diésel, GLP y gasolina. La planta de gas natural licuado produce 210 toneladas de GNL al día y cuenta con una capacidad de almacenamiento de 3000 m^3, una parte es distribuido por el momento a 27 poblaciones repartidas por distintos puntos del país. Desarrollando un sistema de distribución a todas las poblaciones para garantizar el abastecimiento a lugares alejados.

De acuerdo a Guido Van Rossum creador de Python a finales de los 80s y principio de los 90s, desarrolla uno de los lenguajes de programación dinámico más populares en la actualidad. Utilizado para la programación de interfaces gráficas y base de datos, que permiten la automatización de procesos, para poder trabajar con grandes volúmenes de datos, tratando con un lenguaje sencillo, de fácil lectura y con un estilo elegante.

1.2 Planteamiento del problema

1.2.1 Identificación del problema

La Planta de GNL actualmente distribuye su producto a 27 poblaciones a nivel nacional, con posible ampliación a 33 poblaciones y con negociaciones para la exportación a la ciudad de Puno-Perú y Paraguay por gasoductos virtuales [YPFB, 2014]. Lo anterior generará la manipulación y descarga de información constante, en la cual la instancia que coordina la programación de entregas de producto en cada población, recibe información diaria de cada una de las plantas de regasificación, sin tener la opción de gestionar la misma. Manipular la información y además llenar los datos manualmente, del número amplio de estaciones que se tiene, ha dificultado su manejo y genera problemas al personal encargado.

Es por tanto que se desea apoyar a la administración de los datos, para el transporte de GNL a las distintas estaciones. En la actualidad el Ciclo de Suministro de GNL, se maneja de la siguiente manera:

Tabla 1.1 Etapas para distribución de GNL

N	ETAPAS	LUGAR	SITUACIÓN ACTUAL
Etapa 1	Carguío de GNL	Planta Río grande	Excel (email)
Etapa 2	Transporte de GNL	Rutas de Bolivia	Llamada
Etapa 3	Descarguío de GNL	ESR	Llamada
Etapa 4	Remisión de datos	ESR	Excel

Fuente: Elaboración propia

1.2.2 Formulación del Problema

La actual gestión de información utilizada en el flujo de transporte de GNL es administrada de manera manual lo cual aumenta la dificultad, complejidad y tiempo del procesamiento de la información remitida por las estaciones Regasificadoras a la planta de GNL.

1.3 Objetivos

1.3.1 Objetivo General

Administrar la información remitida por los operadores a la Agencia Nacional de Hidrocarburos, automatizando el manejo de la misma para consolidar una base de información histórica que serán utilizados para la toma de decisiones.

1.3.2 Objetivos Específicos

- Aportar a la toma de decisiones de envío de cisternas de GNL a las distintas localidades, para fortalecer la coordinación de abastecimiento a las estaciones de regasificación.
- Analizar las necesidades y definir los requisitos que deberá cumplir el software.
- Registrar el historial de datos necesarios para su integración al programa.
- Realizar una interfaz amigable entre usuario y sistema.
- Estimar modelos de predicción en series de tiempo con redes neuronales que se adecuen al comportamiento de consumo de cada localidad.

1.4 Justificación

1.4.1 Justificación Técnica

Cada día se observa un constante cambio hacia los paquetes de software que están orientados al software libre, como es el caso de distintas librerías con las que cuenta el lenguaje de programación Python, y esto debido a que poseen las características de ser gratuitos, de fuente abierta, potentes y que brindan ciertas libertades que no se encuentran en paquetes de software de tipo propietario o privativo, cualidades que previo estudio y selección se encontró muy propicias para lograr el objetivo.

En este sentido el desarrollo de este proyecto apoyará en la gestión del transporte de gas virtual sin pérdidas o retrasos y en condiciones que garantice el estado inicial de carga de la cisterna criogénica, además de dar un aporte en la mejora de la gestión logística para despacho.

Se busca apoyar la cadena de suministro de GNL, que es el ciclo para el transporte:

Figura 1.1 Ciclo para el transporte GNL

Fuente: Elaboración propia

El diseño de la logística de transporte dependerá de la dinámica de la demanda al momento, es decir si bien se tenía una proyección esta varió de manera considerable, es así que lo que se pretende generar con el presente proyecto es identificar el modelo que se adecue al consumo de cada localidad, siendo que la logística es un proceso dinámico en función del consumo.

En cumplimiento con la misión de la ANH de Regular, Supervisar, Controlar y Fiscalizar con eficacia, calidad y transparencia, el presente proyecto pretende aportar en el manejo de información para controlar de manera eficaz y sencilla toda la base de datos recabada para su análisis y toma de decisiones.

1.4.2 Justificación Social

Como derecho fundamental de las personas, y objetivo principal del proyecto de sistema de Gas Natural virtual, se debe fortalecer la distribución continua a todas las poblaciones beneficiadas. La ventaja de mayor impacto del desarrollo de un software para atender la gestión logística, es la automatización de los procesos involucrados para cumplir con lo establecido en la Ley de Hidrocarburos 3058 y el Decreto Supremo 2159.

En esta Ley de Hidrocarburos se reconoce el valor del Gas Natural y se establece una garantía para el abastecimiento de hidrocarburos al mercado interno, incentivando la expansión a zonas donde no es posible la construcción de gasoductos. En el D.S. 2159 que promulga el "Reglamento Técnico para el Diseño, Construcción, Operación, Mantenimiento y Abandono de Plantas de Gas Natural Licuado-GNL y Estaciones de Regasificación, 23 de octubre de 2014", indica en el artículo 49 los Reportes mensuales de volúmenes, donde el Licenciatario deberá presentar al Ente Regulador reportes mensuales de los volúmenes comercializados de GNL en la Planta de GNL, es por tanto que en conformidad con el artículo, el presente proyecto pretende aportar un mejor manejo de la base de datos, para su análisis e interpretación.

1.4.3 Justificación Económica

Siendo un derecho fundamental de las personas el acceso al gas, el proyecto de sistema virtual de Gas Natural pretende beneficiar al 100 por ciento del mercado interno, de tal manera que alrededor de 60 ciudades se beneficien con este servicio, esto se lo va logrando mediante el proyecto del sistema virtual de Gas Natural Licuado. Es por tanto que el presente proyecto pretende apoyar al sistema de distribución de GNL, que coadyuve al manejo de datos para la toma de decisiones, generando una disminución en el tiempo y gasto económico debido a la coordinación y la mejora de programación de viajes.

CAPÍTULO 2. MARCO TEÓRICO

2 GNL boliviano

El gas natural licuado se presenta como una alternativa al transporte de gas natural por cañerías de alta presión o gasoductos, teniendo en cuenta que Bolivia tiene una geografía que a medida que aumenta la distancia a la población donde el gas debe llegar, este presenta lugares donde puede ocurrir deslizamientos, inundaciones, y otros factores que podrían dañar el ducto. En efecto, se crea la construcción de un sistema virtual de GNL que cumple con la demanda de la población en todos los rincones del país, para poder garantizar el abastecimiento de gas natural.

2.1 Planta de licuefacción de Gas Natural

El GNL es gas natural que ha sido enfriado hasta el punto que se condensa a líquido, para lo cual requiere altas presiones o muy bajas temperaturas a costes muy elevados. Según la Memoria Descriptiva de implementación del Proyecto de la Planta de Licuefacción de Gas Natural en la planta de Río Grande, indica que la cadena integrada del gas natural licuado se compone por tres eslabones:

1) La licuefacción del gas, que contempla filtrado, secado y generalmente se realiza en una zona cercana al pozo o planta de compresión.

2) El transporte, mediante tanques criogénicos trasladados por cisternas o buques metaneros (cuando se trata de sitios marinos).

3) La regasificación, donde se vuelve a convertir el líquido en gas y es introducido a la red de transporte del mercado de consumo.

1. LICUEFACCIÓN

El proceso de licuefacción del gas natural usa los mismos principios de un refrigerador, sufriendo un cambio de estado mediante refrigerantes. El refrigerante es vaporizado en uno o más intercambiadores de calor criogénicos, en donde el gas natural es enfriado. Luego

este es comprimido a alta presión y temperatura, para ser enfriado mediante agua o aire y, posteriormente, enfriado por expansión. En esta etapa de licuefacción, puede decirse que el rendimiento medio del proceso es del 90 % lo que significa que el restante es utilizado como fuente de energía para el proceso o se pierde como quema. [Comisión Nacional Para El Uso Eficiente De La Energía CONUEE, 2017].

2. TRANSPORTE

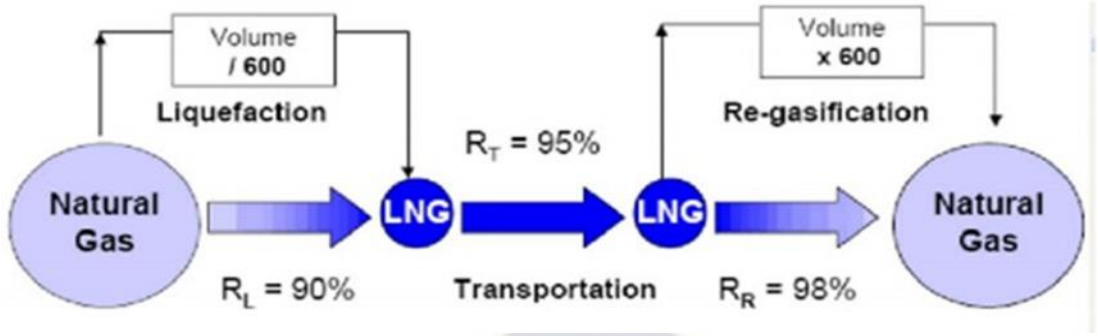
El transporte de GNL se lo realiza mediante cisternas criogénicas. Los mismos deben ser construidos de acero o aleación de aluminio no expuestos a la rotura frágil a la temperatura mínima y máxima de servicio, teniendo una fijación sobre el vagón tal que evite un enfriamiento de las partes portantes frágiles. Deberán contemplar en su diseño y construcción todos los elementos de seguridad y cumplir con el aislamiento térmico [Reg.,].

En el caso del transporte de GNL se tiene un rendimiento del 95 %, teniendo una perdida condicionada a la distancia. Actualmente se cuenta con 29 cisternas criogénicas GNL-7bar y 3 cisternas Twistlock GNL- 7bar de 23 m^3, 12 Tracto Camiones y 3 Camiones tipo Chasis. El flujo de transporte de GNL se maneja con este número de unidades de transporte, las cuales cuentan con la Licencia de Operación para este servicio.

3. REGASIFICACIÓN

Consiste en llevar el gas natural nuevamente a su estado gaseoso, devolviéndole el calor removido en el proceso de refrigeración, donde se elimina el calor sensible y latente del gas natural. El sistema de Gas Virtual cuenta con 27 Plantas o Estaciones Satélites de Regasificación, distribuidos en distintos puntos del país situados en zonas no abastecidas por redes de gas natural canalizado.

Figura 2.1 Esquema de la cadena integrada de GNL



Fuente: Memoria Descriptiva de la implementación del Proyecto – Planta de Licuefacción de Gas Natural (GNRGD)

2.1.1 Demanda

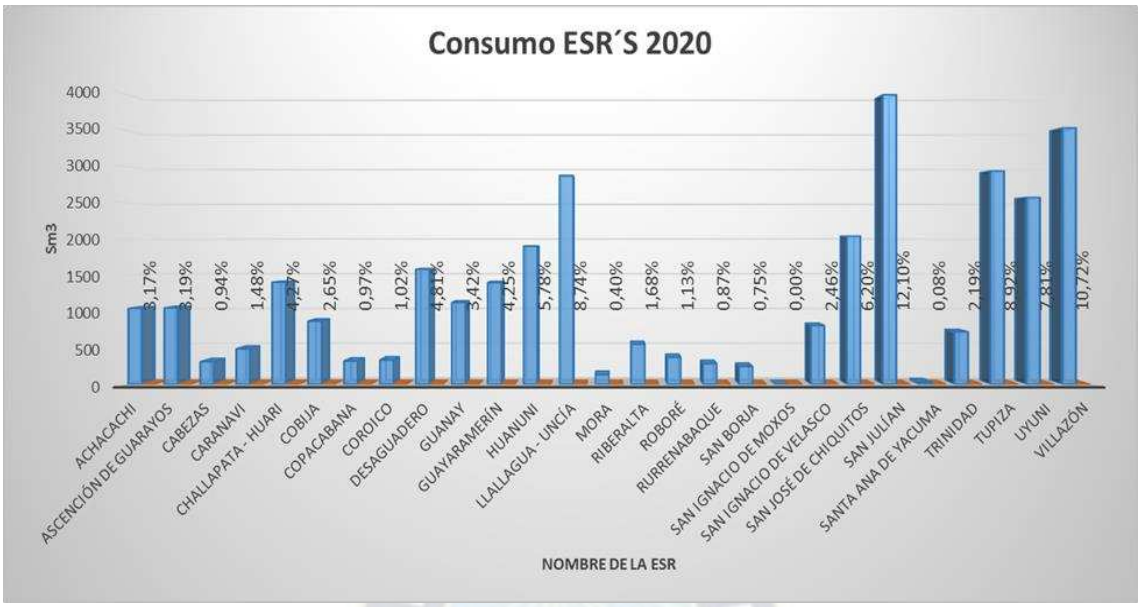
A la fecha se cuenta con 27 Estaciones Satelitales de Regasificación (ESR) construidas en el marco de proyecto construcción de la Planta Gas Natural Licuado y 5 ESR en el marco del proyecto Ampliación de gas virtual GNC a nivel nacional [Hidrocarburos and De, 2021].

Figura 2.2 Industrialización: Estaciones Satelitales de Regasificación

FUENTE: Noticias: Comienza a operar oficialmente el gasoducto virtual de Bolivia, Noticias SENER

Mediante las planillas de reporte obtenidas de las distintas estaciones, se tiene los consumos del año 2020:

Figura 2.3 Consumo por ESR's 2020



Fuente: Elaboración Propia, DATOS ANH

Figura 2.4 Consumo de GNL por Estación de Regasificación, año 2020

N°	Nombre ESR	Consumo Promedio	Porcentaje
1	ACHACACHI	1043,91934	3,17%
2	ASCENCIÓN DE GUARAYOS	1051,301342	3,19%
3	CABEZAS	310,322273	0,94%
4	CARANAVI	488,2142571	1,48%
5	CHALLAPATA - HUARI	1404,593483	4,27%
6	COBIJA	870,7314587	2,65%
7	COPACABANA	319,0529006	0,97%
8	COROICO	334,1833332	1,02%
9	DESAGUADERO	1582,834882	4,81%
10	GUANAY	1125,808577	3,42%
11	GUAYARAMERÍN	1400,188246	4,25%
12	HUANUNI	1902,124057	5,78%
13	LLALLAGUA - UNCÍA	2875,738476	8,74%
14	MORA	131,3029654	0,40%
15	RIBERALTA	553,2487487	1,68%
16	ROBORÉ	372,4562309	1,13%
17	RURRENABAQUE	285,4303318	0,87%
18	SAN BORJA	245,3437545	0,75%
19	SAN IGNACIO DE MOXOS	0	0,00%
20	SAN IGNACIO DE VELASCO	810,1039686	2,46%
21	SAN JOSÉ DE CHIQUITOS	2039,239062	6,20%
22	SAN JULÍAN	3984,63877	12,10%
23	SANTA ANA DE YACUMA	26,59730987	0,08%
24	TRINIDAD	721,7773228	2,19%
25	TUPIZA	2937,11732	8,92%
26	UYUNI	2571,918484	7,81%
27	VILLAZÓN	3529,278713	10,72%

Fuente: Elaboración Propia, en base a datos de la ANH

2.1.2 Oferta

Según la ficha técnica de la Planta de Gas Natural Licuado, este recibe el suministro de materia prima de dos fuentes, una es la Planta Separadora de Líquidos (PSL) Río Grande y la otra del gasoducto Yacuiba – Río Grande (GASYRG), ambos ductos de 4 pulgadas. Pro- cesa 9,5 millones de pies cúbicos diarios (MMpcd) de gas natural y tiene la capacidad de producir hasta 210 TMD, por lo que se necesita paradas establecidas en la producción,

debido a que la oferta es mucho mayor a la demanda que se tiene en el sistema interno de GNL.

Figura 2.5 Paradas y Producción de GNL, 2018

FUENTE: Elaboración Propia, datos ANH

En la Figura 5 se observa periodos establecidos para la producción de GNL y periodos establecidos para las paradas programadas de la Planta en el año 2018.

2.2 Estudio Técnico

La selección técnica de una planta de licuefacción pasa por un conjunto de requerimientos que inicialmente requieren datos como la cromatografía, presión de entrada y temperatura del gas natural a ser procesado, condiciones del sitio de ubicación de la planta, condiciones atmosféricas y meteorológicas y por supuesto las necesidades en cuanto a la demanda ya antes determinadas, de acuerdo al crecimiento de las poblaciones en las distintas localidades.

2.2.1 Planta de GNL

Bolivia inauguró su primera planta de gas natural licuado (GNL), que costó US 145,8 millo- nes, en la región de Río Grande en el departamento de Santa Cruz, para dotar de suministro energético a varias poblaciones rurales.

Las paradas programas en la planta se da debido a que si opera a un porcentaje menor al 100% de su capacidad consume casi la misma cantidad de energía que si funcionará al 100%, por lo tanto, es mejor realizar un trabajo periódico, estableciendo el tiempo en el que lo almacenado en los tanques es distribuido, para luego ser reiniciado el funcionamiento y se realice la repetición del ciclo, hasta alcanzar la capacidad máxima de la planta en función al crecimiento de la demanda [YPFB,].

Figura 2.6 Planta de Gas Natural Licuado

Fuente: ANH, www.anh.gob.bo

2.2.2 Distribución de cisternas criogénicas

La distribución del GNL se realizará a través de cisternas criogénicas, la capacidad máxima útil de recipiente es del noventa y cinco por ciento, quedando prohibido cargarlos por encima del llenado máximo admisible. Durante la evaluación del grado inicial de llenado se debe tener en cuenta el tiempo previsto necesario para el transporte, así como los retrasos que podrían producirse [Reg.,].

Estos camiones deben cumplir todas las consideraciones de seguridad, contando con válvulas, conducciones y dispositivos de carga y descarga, con bombas criogénicas

ubicadas en las estaciones de carga (licuefacción) y descarga (regasificación). Los requerimientos que se utilizaron para poder determinar la posibilidad de distribuir GNL mediante cisternas, fue el estudio de datos para observar la existencia de vías de acceso terrestre y las condiciones de las mismas.

Se distribuye el producto en camiones cisterna de un volumen total de 55 m^3 del que, por motivos de seguridad, durante el llenado no debe exceder del 83 %, a una temperatura de -161°C y a una presión preferiblemente, próxima a la atmosférica. El presión y temperatura del GNL de las cisternas a su llegada a las ESR's dependerá de las condiciones ambientales y de la ruta pudiendo alcanzar hasta los 7 barg.

Es por tanto que el Director Ejecutivo de la Agencia Nacional de Hidrocarburos, a nombre del Estado Boliviano y en ejercicio a sus atribuciones conferidas por la Ley N°1600 del Sistema de Regulación Sectorial; la Ley de Hidrocarburos 3058 de 17 de mayo de 2005; el Reglamento Técnico y de Seguridad para el servicio de Transporte de Gas Natural Licuado - GNL aprobado mediante Decreto Supremo N°2571 de 28 de octubre de 2015 y demás normas legales sectoriales aplicables, otorgó a Yacimientos Petrolíferos Fiscales Bolivianos (YPFB) la Licencia de Operación para el Servicio de Transporte de Gas Natural Licuado a:

Tabla 2.1 Descripción de Equipamiento de cisternas

Descripción de equipamiento	Cantidad	Código
Cisterna Twistlock GNL 7 bar de 23 m3	3	CG-15337;CG-15338
Estación Satelital de Regasificación Móvil	2	CG-15359;CG-15360
Cisterna Criogénica GNL-7 bar	29	CG-15308-AlK A

Fuente: YPFB

Tabla 2.2 Unidades de Tracto-camión

Descripción de Unidades de Transporte	Cantidad
Camión Chasis 6 x 4	3
Tracto Camión 6 x 2	8
Tracto Camión 6 x 4	4

Fuente: YPFB

2.2.3 Sistemas de ESR's que conforman el sistema virtual de GNL

La primera actividad del proceso de regasificación de GNL es la descarga de las cisternas mediante bomba instalada en las mismas al depósito criogénico de GNL, mediante manguera flexible de 2 pulgadas para el trasvase de GNL al tanque. La selección técnica de los puntos de recepción y distribución fue desarrollada mediante un conjunto de requerimientos como las necesidades de la distribución presentes y futuras, la presión de salida, el almacenaje, las necesidades de compresión del gas natural, el tiempo de almacenamiento, condiciones del sitio de ubicación de la planta, condiciones atmosféricas y meteorológicas y por supuesto las necesidades en cuanto a la demanda que fueron determinadas antes de realizar la construcción de las mismas.

Las actividades necesarias para la descarga son: la puesta a tierra de la cisterna y conexión de la manguera, posterior a esto todo el proceso se realiza de manera automática, contando con un panel local de comando con las botoneras de inicio y paro de la descarga de acuerdo a la toma inferior del tanque o por las duchas superiores en función de la presión y temperatura del mismo.

Las plantas satélites de regasificación pueden ser de dos tipos: la primera es la planta que consta de un sistema de regasificación forzada, es decir que el calentamiento del gas natural licuado depende del intercambio de calor con un fluido caliente, el cual es generalmente agua que es calentado en un caldero cuyo calor es proporcionado por la combustión en parte del gas resultante de la regasificación producida por el efecto boil off y en mayor proporción resultante de la regasificación final posterior al regasificador forzado. El segundo tipo consta de un sistema de regasificación atmosférico, es decir que el gas es calentado por simple intercambio de calor con el medio ambiente, en este caso el licuado a temperatura que oscilan los -160 °C absorbe el calor del medio ambiente y como resultado de este intercambio la humedad existente en el aire toma este frío y se congela alrededor de las tuberías que transportan el licuado a enfriar, este hielo a medida que pasa el tiempo terminaría congelando todo el sistema de tuberías del regasificador lo cual produciría un inadecuado calentamiento del GNL, resultando en un gas natural frío que provocaría

problemas en las tuberías de polietileno, las cuales no están diseñadas para soportar estas condiciones de temperatura.

En ambos tipos de regasificación se tendrá posterior a estos un sistema de regulación de presión, el cual permitirá reducir la presión a la necesaria para la red de gas natural que pretendamos instalar en la población en cuestión, y posterior a este sistema de regulación de presión se tendrá instalado un dosificador de odorizante por arrastre el cual proporcionará el olor característico del gas natural que es añadido por cuestiones de seguridad.

Para cada tipo de las ER's según consumo, se han diseñado configuraciones de depósitos horizontales criogénicos, que están conformados por diferentes capacidades, tales como:

- Configuración A1, A2: 1 depósito de 80 m3
- Configuración B1, B2, TB1: 2 depósitos de 80 m3
- Configuración C1, TC1: 3 depósitos de 80 m3

Las características del tanque de almacenaje de GNL son tales que puede soportar temperaturas extremas, contando con un diseño interno y externo que funciona como capa aislante. El depósito interior es de acero inoxidable y el externo cumple la función de proteger el depósito interior, entre estos se encuentra el material aislante para prevenir el paso del calor, y evitar la evaporación al someterse a temperaturas superiores a su punto de ebullición generando el llamado Boil-off Gas.

La presión del depósito se mantendrá dentro de los límites establecidos, en base a dos sistemas que incorpora los propios tanques:

- Circuito PPR (Puesta en Presión Rápida)
- Circuito Economizador

El PRR está formado por un vaporizador ambiental horizontal, el circuito empieza a través del paso en el regulador mecánico, que es una válvula reguladora de presión con tag number, es decir su función es la de mantener constante la presión en todo el sistema de

alimentación, ajustado previamente para dar paso al GNL hacia el vaporizador E-1100, cuando la presión del depósito descienda por debajo de un valor determinado. Los vapores generados a la salida del vaporizador retornan al depósito aumentando la presión del mismo [Del and ESR's, 2015]. En cambio, se tiene el circuito economizador que cumple la función de conseguir el efecto contrario, es decir disminuir la presión en el interior del depósito. Para ello también utiliza un regulador mecánico, capaz de aliviar parte de la fase de gas a la tubería de salida de consumo.

De acuerdo a las fichas técnicas presentadas por la página oficial de la ANH, se tienen las siguientes especificaciones, para cada Estación Satélite de Regasificación:

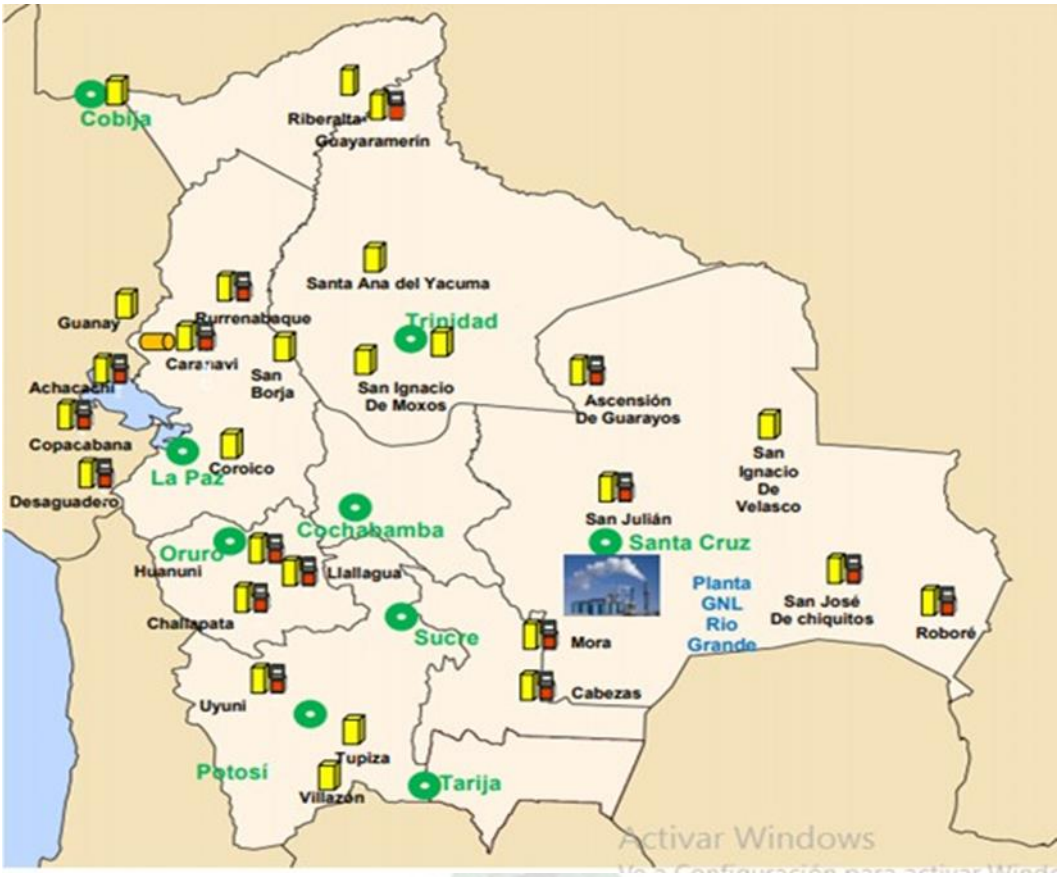
Tabla 2.3 Especificaciones Técnicas de las 27 Estaciones Satélite de Regasificación

ESR	Distancia ER a PGNL (Km)	Tiempo de viaje ida/vuelta Planta de GNL-ER (hr)	Unidades de Almacenaje	Capacidad Total (m3)	Capacidad de almacenaje (m3)
ACHACACHI	1129	56	2	160	80(Alta)
					80(Baja)
CABEZAS	20	3	1	80	80(Baja)
CARANAVI	1233	68	3	240	80(Baja)
					80(Baja)
					80(Alta)
CHALLAPATA	977	50	2	160	80(Alta)
					80(Baja)
COBIJA	448	26	2	160	80(Alta)
					80(Baja)
COPACABANA	1183	60	2	100	80(baja)
					20(Baja)
COROICO	1167	60	1	80	80(Baja)
DESAGUADERO	1143	57	2	100	80(baja)
					20(Alta)
GUANAY	1303	76	1	80	80(Baja)
GUARAYOS	411	19	1	80	80(Baja)
GUAYANAMERIN	700	554	3	240	80(Baja)
					80(Baja)
					80(Alta)
HUANUNI	905	46	1	80	80(Baja)
LLALLAGUA	962	49	2	160	80(Alta)
					80(Baja)
MORA	18	2	1	80	80(Baja)
RIBERALTA	86	14	3	240	80(Baja)
					80(Baja)
					80(Alta)
ROBORE	477	21	1	80	80(Baja)
RURRENABAQUE	1080	58	1	80	80(Baja)
SAN BORJA	930	44	1	80	80(Baja)
SAN IGNACIO DE MOXOS	792	34	1	80	80(Baja)
SAN IGNACIO DE VELASCO	149,5	3	1	80	80(Baja)
SAN JOSE DE CHIQUITOS	345	15	1	80	80(Baja)
SAN JULIAN	240	10	1	80	80(Baja)
SANTA ANA DEL YACUMA	792	34	2	160	80(Alta)
					80(Baja)
TRINIDAD	700	26	3	240	80(Baja)
					80(Baja)
					80(Alta)
TUPIZA	1373	88	2	160	80(Alta)
					80(Baja)
UYUNI	1176	74	2	100	80(baja)
					20(Baja)
VILLAZON	1465	96	1	80	80(Baja)

Fuente: Elaboración propia, datos ANH. www.anh.gob.bo

Al momento se cuentan con 15 Estaciones de Servicio de GNV, de las cuales 4 se encuentran en funcionamiento en las poblaciones de Ascensión de Guarayos, Huanuni, San José de Chiquitos y San Julián.

Figura 2.7 Mapa de distribución de las ESR's



Fuente: ANH www.anh.gob.bo

Se tiene una estación madre en la ESR de Caranavi, la cual cuenta con un regasificador móvil con cisterna convencional, circuito de regasificación ambiental, bomba criogénica, manifold para la Regulación, Medición y Odorización. Este conjunto, requiere al igual que la cisterna, de una unidad tractora (tracto camión).

2.2.4 Sistema Logístico de transporte para distribución de GNL

El sistema logístico de transporte depende principalmente de las condiciones de acceso a los sitios de regasificación y la demanda que presenta cada población a la cual llegará la

distribución con gas natural. El diseño de la logística dependerá también de la dinámica de la demanda al momento, es decir, si bien se tiene una proyección de la demanda esta puede variar en función al comportamiento del consumo de cada localidad.

El proceso logístico de transporte, por lo tanto, tendrá su inicio en las paradas de carga de gas natural licuado ubicados en la planta central de GNL, en los cuales se realizará el trasvase de licuado desde los tanques de almacenaje hasta las cisternas, el tiempo de carga de acuerdo a un informe de la Gerencia Nacional de Redes de Gas y Ductos es de aproximadamente 2 horas. Posteriormente el tracto camión que lleve la cisterna tomará la ruta más adecuada hacia la localidad de destino, cuyo tiempo estaría en función a la distancia que se deberá recorrer, la velocidad máxima a la cual puede transitar el vehículo (por seguridad es no más de 50 km/h) y el estado de la vía pudiendo ser esta asfaltada, ripiada o de tierra.

Una vez llegado el tracto camión a la localidad de destino este realizara la descarga el cual sería un procedimiento similar al del carguío y tardara nuevamente 2 hora. En caso de que la cisterna no se vacíe completamente el tracto camión llevara este a otra población a la cual podamos transvasar el líquido sobrante, minimizando de esta manera el tiempo de reabastecimiento de esta segunda población. Finalmente, el tracto camión retorna a la planta de GNL con la cisterna vacía para luego repetir el ciclo. Es importante tomar en cuenta que el tiempo de retorno pueda ser menor al tiempo de ida debido a que la cisterna ya descargo el líquido que transportaba, pero es necesario tomar previsiones y tomar el mismo tiempo de ida por seguridad y posibles contratiempos.

2.3 Software libre

Es el año 2013, la industria informática ha tomado como relevancia el fenómeno que se conoce como software libre. En este, se vio la oportunidad del acceso al código fuente de un programa, lo que admite ser libre de uso, ejecución, distribución y modificación. Es decir, el nuevo software creado bajo este concepto podría emplearse para cualquier fin, ejecutarse en cualquier ambiente, distribuirse a discreción del propio usuario y modificarse de ser necesario. Lo que maro el comienzo fue el lanzamiento de la primera versión del núcleo Linux por Linus Torvalds en 1991, y en ese mismo año Guido van Rossum libera la

primera versión del lenguaje de programación Python [Challenger-Pérez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, 2014].

Es el año 2013, la industria informática ha tomado como relevancia el fenómeno que se conoce como software libre. En este, se vio la oportunidad del acceso al código fuente de un programa, lo que admite ser libre de uso, ejecución, distribución y modificación. Es decir, el nuevo software creado bajo este concepto podría emplearse para cualquier fin, ejecutarse en cualquier ambiente, distribuirse a discreción del propio usuario y modificarse de ser necesario. Lo que maro el comienzo fue el lanzamiento de la primera versión del núcleo Linux por Linus Torvalds en 1991, y en ese mismo año Guido van Rossum libera la primera versión del lenguaje de programación Python [Challenger-Pérez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, 2014].

Los lenguajes de programación son la herramienta básica de construcción de programas. En nuestra vida diaria estamos rodeados de computadoras, desde equipos portátiles (laptops) hasta teléfonos móviles (celulares). Podemos pensar en esas computadoras como nuestros “asistentes personales”, que pueden ocuparse de muchas tareas por nosotros.

Crear programas útiles, elegantes e inteligentes para que los usen otros, es una actividad muy creativa. Tu computadora normalmente contiene muchos programas diferentes pertenecientes a distintos grupos de programadores, cada uno de ellos compitiendo por tu atención e interés [González Duque, 2000].

Un programa es una secuencia simple de operaciones que permite la ejecución automática de escritos digitales en un dispositivo informático. Y en un software, puede existir varios programas. La secuencia de instrucciones que especifica un cálculo, puede ser matemático o simbólico, como buscar y reemplazar texto en un documento o compilar un programa [Soediono, 1989].

Los detalles se ven diferentes en diferentes idiomas, pero algunas instrucciones básicas aparecen en solo sobre todos los idiomas:

- input: Obtenga datos del teclado, un archivo o algún otro dispositivo.

- output: Muestra datos en la pantalla o envía datos a un archivo u otro dispositivo.
- math: Realiza operaciones matemáticas básicas como sumas y multiplicaciones.
- conditional execution: Verifique ciertas condiciones y ejecute el código apropiado.
- repetition: Realiza alguna acción repetidamente, generalmente con alguna variación.

Todos los programas realizados, sin importar lo complicados que sean, se componen de instrucciones como estas. Por lo tanto, se puede pensar en la programación como el proceso de dividir una tarea grande y compleja en sub tareas cada vez más pequeñas hasta que estas sean lo demasiado básicas, para poder realizarlas.

2.3.1 Python

Python fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba, los desarrolladores de Python han creado sus propias formas de escribir código.

De acuerdo a la página oficial de Python “citepython”, existe una lista de principios de diseño que se deberían seguir cuando se escribe código en Python:

- Hermoso es mejor que feo
- Explícito es mejor que implícito
- Simple es mejor que complejo
- Plano es mejor que anidado
- Disperso es mejor que denso
- El código legible cuenta
- Los errores no deben pasar nunca desapercibidos, a menos que se especifique este comportamiento

Interpretando la lista, nos da a entender que mientras más sencilla y clara se mantenga e implemente las ideas, mejores serán estas.

La característica principal de Python es su sencillez, tanto en sus instrucciones, en su sintaxis y en su estructura. Sin afectar estas características a su potencia, siendo una programación multiplataforma, que permite la expansión de sus funciones mediante una infinidad de librerías. La sintaxis de Python es tan sencilla y cercana al lenguaje natural que los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar.

Se definen como sus principales propiedades de Python, las siguientes:

- Lenguaje de programación de alto nivel.
- Diseñado para ser fácil de leer y simple de implementar.
- Es código abierto (de uso libre).
- Es de propósito general y se puede utilizar también para desarrollar aplicación web y contenido web dinámico.
- Los scripts de Python tienen la extensión de archivo .py.
- Permite grabar programas compilados con extensión de archivo. pyc, los cuales suelen ser usados como módulo que pueden ser referenciados por otros programas Python. [pyt,]

Python es definido como un lenguaje de alto nivel, siendo sus ventajas:

- Primero: la facilidad de programar en un alto nivel, es decir toma menos tiempo para escribir, son más cortos y más fáciles de leer y es más probables que sean correctos.
- Segundo: Los lenguajes de alto nivel son portable, lo que significa que pueden ejecutarse en diferentes tipos de computadoras con pocas o ninguna modificación.

Un detalle muy importante a tomar, para comenzar la programación y desarrollo de software, es saber que es propensa a errores. Pueden ocurrir tres tipos de errores en un programa: errores de sintaxis, errores de tiempo de ejecución y errores semánticos. Es útil distinguirlos para localizarlos más rápidamente.

- ✓ Error de sintaxis:

Python solo puede ejecutar un programa si la sintaxis es correcta, de lo contrario, lo interpreta como error. La sintaxis se refiere a la estructura de un programa y las reglas sobre estas, si Python encuentra un error, no podrá ejecutar el programa.

- ✓ Error de tiempo:

Este es el segundo tipo de error, llamado así porque no aparece hasta después de que el programa haya comenzado a ejecutarse, por lo general indican que sucedió algo erróneo con la ejecución.

- ✓ Error semántico:

El tercer tipo, es el error semántico en el programa, este ejecutara normalmente pero no realizara lo correcto, es decir lo que tu pretendías que realice. Por lo que se volverá a replantear las líneas de código.

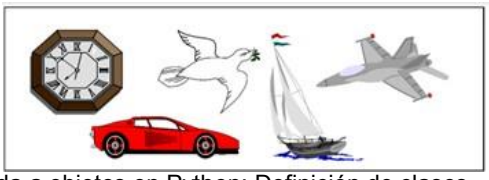
2.3.2 Programación Orientada a Objetos (POO)

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos. La programación orientada a objetos (POO), organiza el diseño de software entorno a datos u objetos, siendo este un modelo de programación informática [González Duque, 2000].

Las principales definiciones son:

Objeto: Campo de datos que tiene atributos y comportamientos únicos. Un objeto es simplemente un trozo de código más estructuras de datos, que conforman un pequeño programa completo. Ej.: Un ser humano que se describe por propiedades como nombre y dirección, hasta pequeños programas informáticos, como widgets.

Figura 2.8 ¿Qué es un objeto, en programación?



Fuente: Programación orientada a objetos en Python: Definición de clases — Solución de problemas con algoritmos

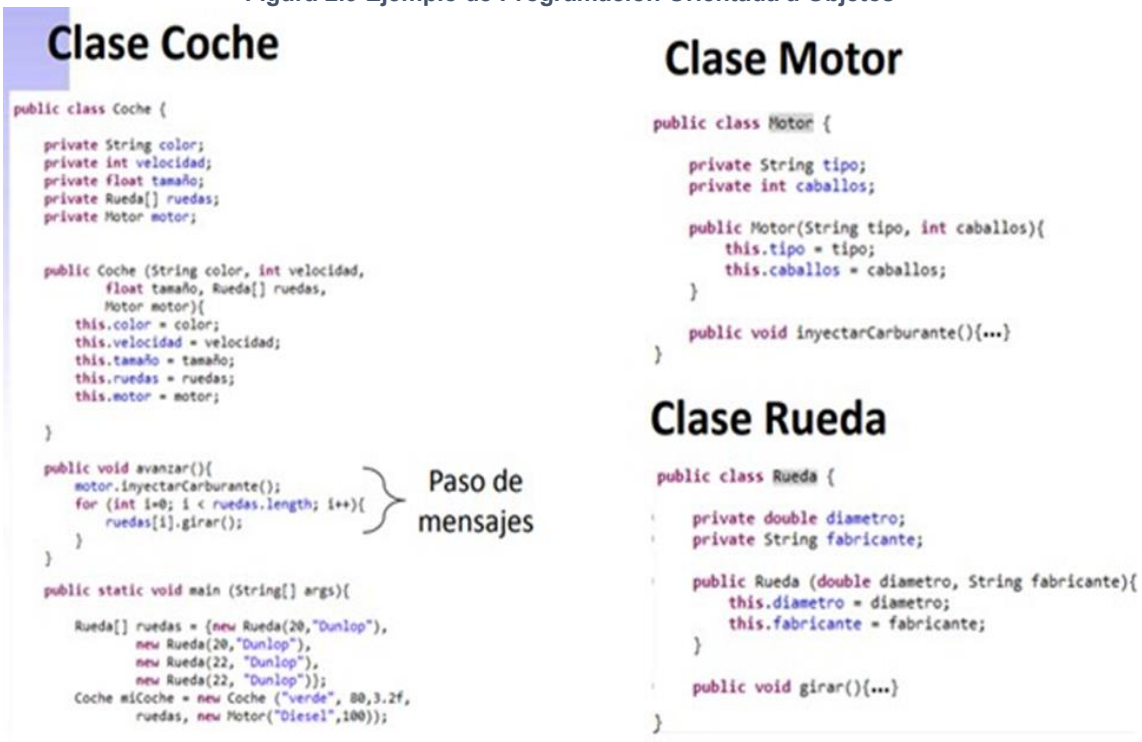
- Los objetos son representaciones (simples/complejas) (reales/imaginarias) de cosas: reloj, avión, coche.
- No todo puede ser considerado como un objeto, algunas cosas son simplemente características o atributos de los objetos: color, velocidad, nombre.

Un objeto es una forma de agrupar un conjunto de datos (estado) y de funcionalidad (comportamiento) en un mismo bloque de código que luego puede ser referenciado desde otras partes de un programa [Villena, 2019].

Clase: Conjunto de objetos con estados y comportamientos similares, es un tipo de clasificación de datos, define el comportamiento y atributos de un grupo de estructura y comportamiento similar. Usamos la palabra clave “class” para definir los datos y el código que compondrán cada objeto. La clase a la que pertenece el objeto puede considerarse como un nuevo tipo de datos.

Una clase es una entidad abstracta, es un tipo de clasificación de datos y define el comportamiento y atributos de un grupo de estructura y comportamiento similar.

Figura 2.9 Ejemplo de Programación Orientada a Objetos

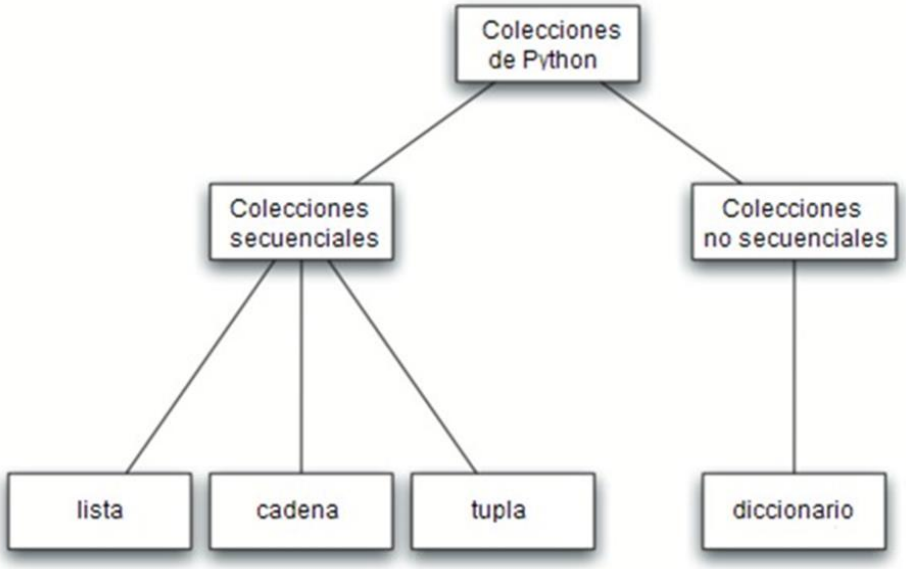


Fuente: Programación orientada a objetos en Python: Definición de clases — Solución de problemas con algoritmos

La clase es como un molde para galletas y los objetos creados usándola, son las galletas. El POO es más un proceso de modelado, donde se identifican las entidades que intervienen en el problema, y su comportamiento, y se definen clases que modelizan esas entidades.

Una de las características más importantes en un lenguaje de programación orientado a objetos es la capacidad de permitir a un programador (solucionador de problemas) crear nuevas clases que modelen los datos necesarios para resolver el problema. Un aspecto importante de este proceso es la habilidad de herencia que tiene, para que una clase esté relacionada con otra clase de la misma manera que las personas pueden estar relacionadas entre sí. Del mismo modo, las clases hija en Python pueden heredar datos y comportamientos característicos de una clase madre. Estas clases se denominan a menudo subclases y superclases, respectivamente.

Figura 2.10 Jerarquía de herencias para las colecciones en Python



Fuente: Programación de Sistemas Orientada a Objetos (Villena, 2019).

Las listas, las tuplas y las cadenas son todas tipos de colecciones secuenciales. Todas heredan organización de datos y operaciones comunes. Sin embargo, cada una de ellas es distinta según los datos sean o no homogéneos y si la colección es inmutable. Al organizar las clases de esta manera jerárquica, los lenguajes de programación orientados a objetos permiten que el código previamente escrito se extienda para satisfacer las necesidades de una nueva situación. Además, al organizar los datos de esta manera jerárquica, podemos comprender mejor las relaciones que existen entre ellos. Podemos ser más eficientes en la construcción de nuestras representaciones abstractas [Pro,].

2.3.3 Librerías de Python

Una de las fortalezas de Python, son las librerías con las que cuenta. Tiene una variedad de módulos que cubren la mayoría de las necesidades básicas de un programador [Challenger-Pérez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, 2014].

Se tiene tópicos como:

- Cadenas

- Estructura de datos
- Funciones numéricas y matemáticas
- Comprensión de datos
- Formatos de archivo

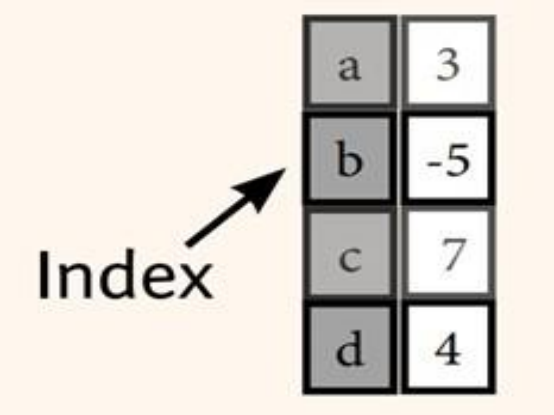
Se definirá las librerías utilizadas en el presente proyecto:

PANDAS

Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que el trabajo con datos relacionales o etiquetados sea fácil e intuitivo. Pretende ser el elemento fundamental de alto nivel para realizar análisis de da- tos prácticos y del mundo real en Python. La librería de Pandas está construida en base a otra, llamada Numpy y provee un fácil uso de las herramientas que conforman la estructura de datos y análisis de datos para la programación en el lenguaje de Python. Las principales estructuras de datos son:

- Series

Figura 2.11 Series

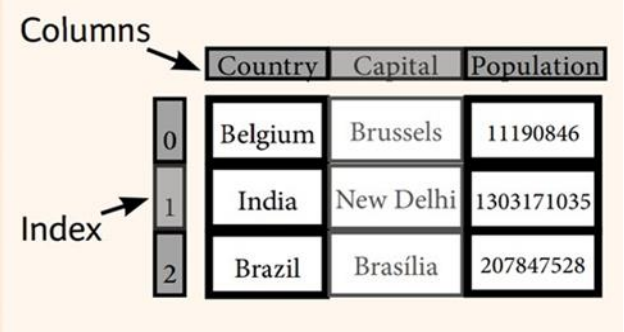


Fuente: Elaboración propia con conceptos www.python.gob.bo

Una serie es una matriz unidimensional capaz de contener cualquier tipo de datos.

- Dataframe

Figura 2.12 Dataframe



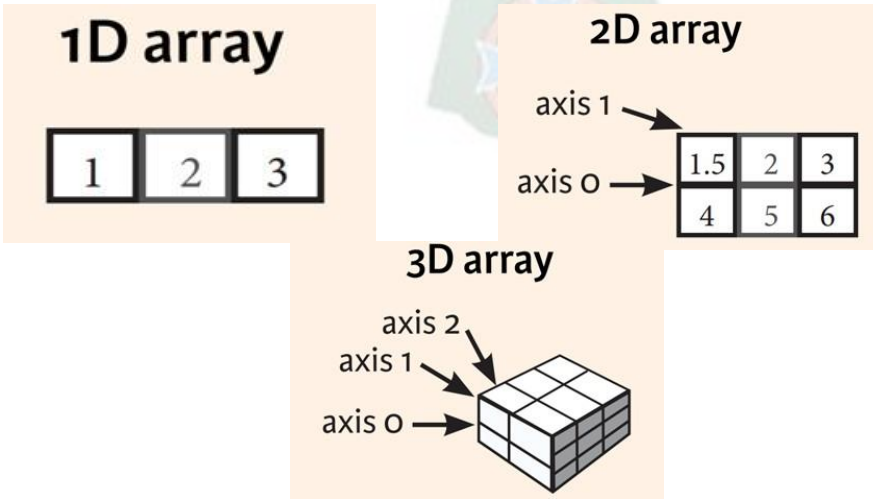
Fuente: Elaboración propia, mediante con www.python.gob.bo

Un dataframe es una estructura de datos etiquetada bidimensional con columnas de tipos potencialmente diferentes.

NUMPY

La librería de Numpy es la librería central para la computación científica en Python, valga la redundancia. Proporciona una matriz multidimensional de alto rendimiento, objeto y herramientas para trabajar con estas matrices. Los “arrays” de Numpy es una colección de N elementos, siendo que los multidimensionales nos permiten almacenar datos de manera estructurada y las funciones universales nos permiten operar con esos datos de manera eficiente.

Figura 2.13 1D array, 2D array y 3D array

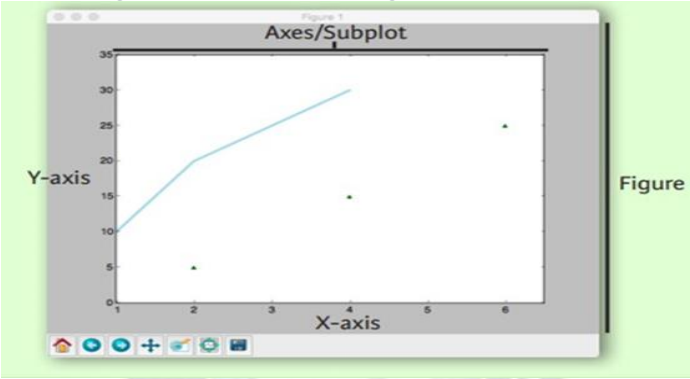


Fuente: Elaboración propia, mediante con www.python.gob.bo

Es importante saber que si Numpy no entiende el tipo de datos o construimos un array con argumentos incorrectos devolverá un array con “dtype” o “object”. Estos arrays rara vez son útiles y su aparición suele ser signo de que algo falla en nuestro programa.

MATHPLOTLIB

Figura 2.14 Trazado de la gráfica Mathplotlib



Fuente: Elaboración propia, mediante con www.python.gob.bo

Es una librería de trazado de Python 2D que produce figuras con calidad de publicación en una variedad de formatos y entornos interactivos en plataformas. [Hunter Jhon, Dale Darren, Firing Eric, 2021].

DATETIME

Para la utilización de fechas en la programación, se utiliza la librería “datetime” que incorpora los tipos de datos “date”, “time” y “datetime” para representar todo lo que contenga fechas y sus funciones para manejarlas [dat, 2020].

Algunas de las operaciones más habituales que permite son:

- Acceder a los distintos componentes de una fecha, dependiendo del formato del mismo (año, mes, día, hora, minuto, segundos y microsegundos).
- Convertir cadenas con formato de fecha en los tipos date, time o datetime.
- Convertir fechas de los tipos date, time o datetime en cadenas formateadas de acuerdo a diferentes formatos de fechas.

- Hacer aritmética de fechas.
- Comparar fechas.

Horas:

Estos valores se representan con la clase time, tiene atributos para “hour”, “minute”, “second”, y “microsecond” y también incluye información de zona horaria. Esta instancia solo tiene valores de tiempo, y no una fecha asociada con el tiempo.

Fechas:

Los valores de la fecha que vemos en nuestro calendario, son representados por la clase date, tiene atributos para “year”, “month” y “day”. El método más importante para el manejo de nuestros datos, está representado por el método de clase today (), el cual representa la fecha actual.

Os:

Este módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo. [os,]

Notas sobre la disponibilidad de estas funciones:

- El diseño de todos los módulos incorporados en Python dependientes del sistema operativo es tal que, mientras la funcionalidad esté disponible, usará la misma interfaz.
- Todas las funciones que aceptan rutas o nombres de archivos aceptan bytes o cadenas de texto, y el resultado es un objeto del mismo tipo, siempre que se retorna una ruta o un archivo.

2.3.4 Estructura y elementos del lenguaje

A diferencia de la mayoría de los lenguajes de programación, Python nos provee de reglas de estilos, a fin de poder escribir código fuente más legible y de manera estandarizada.

- ✓ Variables: Una variable es un espacio para almacenar datos modificables, en la memoria de un ordenador.

- ✓ Función: Es una secuencia de sentencias con un nombre que realizan alguna operación útil.
- ✓ Funciones con argumento: Las funciones internas necesitan argumentos.
- ✓ Bucles definidos: Se repite un bucle a través de un conjunto de cosas, como una lista de palabras, las líneas de un archivo, o una lista de números. Cuando se tiene una lista de cosas para recorrer, se puede construir un bucle definido usando una sentencia “for”. Se la denomina de esta manera, porque se repite hasta que cumpla cierta condición.
- ✓ Método: Es similar a una función (esta toma argumentos y devuelve un valor) pero la sintaxis es diferente. Llamamos a un método uniendo el nombre del método al de la variable, usando un punto como delimitador.
- ✓ Operador de formato: El operador de formato % nos permite construir cadenas, reemplazando partes de las cadenas con datos almacenados en variables. Cuando los aplicamos a enteros, es el operador módulo, pero cuando es aplicado a una cadena, es el operador de formato.
- ✓ Cadenas: Las cadenas o strings son un tipo de datos por secuencias de caracteres que representan texto.
- ✓ Listas: A diferencia de las cadenas, las listas son mutables porque pueden cambiar el orden de los elementos en una lista o reasignar un elemento en una lista. La forma más común de recorreré los elementos de una lista es con un bucle for.
- ✓ Tuplas: Una tupla es una secuencia de calores similares a una lista. Los valores guardados en un tupla pueden ser de cualquier tipo, y son indexados por números enteros. La principal diferencia es que las tuplas son inmutables.
- ✓ Diccionarios: Asociación de un conjunto de índices y un conjunto de valores. Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor.

En un programa de Python se deberá tener en cuenta dos valores, muy diferenciados: números y letras. El valor introducido entre comillas corresponderá al tipo “Sting”, mientras que un valor numérico podrá tener diferentes tipos de datos, enteros y “float”, de manera que depende si nuestros datos los manejamos con decimales o no.

2.3.5 Interfaz gráfica de usuario

Las interfaces gráficas de usuario, que tiene como acrónimo (GUI) en inglés Graphical User Interface, permite a los usuarios de un sistema interactuar con él a través de indicadores visuales o íconos gráficos. La mayoría utilizan ideas como las ventanas, en donde se tiene distintos elementos, como ser: botones, hipervínculos, que conforman el lenguaje visual de las interfaces gráficas de usuario. Los GUI's ofrecen un gestor de ventanas que facilita las interacciones entre las mismas, las aplicaciones y el sistema operativo.

2.4 Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, resaltando la sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias [vis,].

2.5 Series Temporales

Una serie de tiempo es un conjunto secuencial de puntos de datos, medidos típicamente en tiempos sucesivos. Se define matemáticamente como un conjunto de vectores x(t), t =..., 2, 1, 0, donde t representa el tiempo transcurrido. La variable x(t) se trata como una variable aleatoria. Las mediciones tomadas durante un evento en una serie de tiempo se organizan en un orden cronológico adecuado. Una serie de tiempo que contiene registros de una sola variable se denomina univariante. Pero si se consideran los registros de más de una variable, se denomina multivariante. La serie puede ser continua o discreta. En una serie de tiempo continua, las observaciones se miden en cada instancia de tiempo, mientras que una serie de tiempo discreta contiene observaciones medidas en puntos de tiempo discretos. Por ejemplo, lecturas de temperatura, flujo de un río, concentración de un proceso químico etc. se pueden registrar como una serie de tiempo continua. Por otro lado, la población de una ciudad en particular, las producciones de una empresa pueden representar una serie de tiempo discreta. Por lo general, en una serie de tiempo discreta,

las observaciones consecutivas se registran en intervalos de tiempo igualmente espaciados, como separaciones horarias, diarias, semanales, mensuales o anuales [Adhikari and Agrawal, 2013].

De una manera más específica se denomina serie de tiempo a un conjunto de valores observados durante una sucesión de periodos temporales secuencialmente ordenada, mediante su análisis se pueden extraer parámetros relevantes o características que pueden ser utilizados para generar un modelo matemático que la describa y ser utilizado para realizar predicciones.

2.5.1 Análisis de una serie de tiempo

En el análisis se ve distintos aspectos, en los que se debe conocer, la definición de:

Figura 2.15 Análisis gráfico de una serie de tiempo

Fuente: Curso de Análisis de Forecast con R y Python, DCM.

- Estacionalidad. - La variación estacional representa un movimiento periódico de la serie de tiempo, es decir sus propiedades estadísticas no dependen del tiempo al momento en que la serie es observada.

  Se realiza una distinción entre componentes cíclicos y estacionarios, estas últimas ocurren con periodos identificables, en cambio el otro termino es atribuible a alguna causa que refiere a ciclos grandes.

- Tendencia. - La tendencia representa el comportamiento predominante de la serie, esta puede estar definida como el cambio de la media a lo largo de un extenso periodo de tiempo.

  Consiste en la evolución a largo plazo de la serie. Es usual encontrarse con series temporales que presentan un movimiento sostenido en la misma dirección durante un amplio período de tiempo, con independencia de pequeñas oscilaciones al alza o a la baja. La tendencia suele ser representada mediante curvas “suaves”, siendo habitual que se represente mediante funciones lineales, dando lugar a las rectas de tendencia (Guías,).

  Las tendencias y estacionalidades pueden darse simultáneamente.

- Variaciones aleatorias. - Los movimientos irregulares representan todos los tipos de movimientos de una serie de tiempo que no tenga tendencia, variaciones estacionales y fluctuaciones cíclicas.

- Outliers. - Son puntos de la serie que escapan de lo normal, si se sospecha que una observación es un outlier, se debe reunir información adicional sobre posibles factores que afectaron el proceso.

Una manera de determinar de forma cuantitativa si una serie temporal es estacionaria es utilizando un test estadístico, llamado “raíces unitarias”. Permiten determinar si es necesario aplicarle transformaciones a la serie. Entre estos están los siguientes test:

- El test de Dickey-Fuller Aumentado (Augmented Dickey-Fuller Test) (ADF), elimina la auto correlación e indica si una serie es estacionaria o no, esta da un número negativo, cuanto más negativo es, más fuerte es el rechazo de la hipótesis nula de que existe una raíz unitaria para un cierto nivel de confianza.

Ecuación 1 Dickey-Fuller

$$DF_{\gamma} = \frac{\gamma}{SE(\gamma)}$$

Ho: La serie tiene raíces unitarias, entonces no es estacionaria.

H1: La serie no tiene raíces unitarias, entonces es estacionaria. Este determina el test de hipótesis: p < 0,05

- El test de Phillips-Perron (PP). - Es una modificación de test de Dickey-Fuller, donde corrige la auto correlación y heterocedasticidad en los errores.
  Aborda la cuestión de que el proceso de generación de datos podría tener un orden superior de auto correlación que es admitido en la ecuación de prueba.

Ecuación 2 Phillips-Perron

$$\gamma_t = p\gamma_{t-1} + \mu_t$$

Ho: No es estacionaria
H1: Es estacionaria
Este determina el test de hipótesis: p > 0,05

2.6 Métodos predictivos

2.6.1 Método predictivo estadístico

Estos modelos se pueden utilizar si nuestra serie temporal presenta tendencia y/o estacionalidad, determinando esto primeramente de manera visual y posterior a esto realizando los test presentados anteriormente para determinar estos componentes en nuestra serie.

Modelo Autorregresivo

El modelo predictivo Autorregresivo (AR), se basa en la idea de calcular un valor en función a los valores de la misma serie en un tiempo pasado. Los modelos Autorregresivos son muy flexibles en cuanto al manejo de un amplio abanico de patrones de series temporales diferente, pero requieren que la serie temporal sea estacionaria. El modelo más simple, se describe en la siguiente ecuación:

Ecuación 3 Modelo Autorregresivo

$$X_t = \alpha_1 + \beta_1 X_{t-1} + s_t$$

Donde:

t = el valor del tiempo en ese momento

α_1 = valor constante

β_1= valor constante

s_t= error que varía con el tiempo, se asume que dicho error posee varianza constante y media 0 (cero).

Para la creación de un modelo Autorregresivo que depende de los ρ valores más recientes AR(ρ) se describe de la siguiente manera:

Ecuación 4 Modelo Autorregresivo

$$X_t = \alpha_1 + \beta_1 X_{t-1} + \alpha_2 + \beta_2 X_{t-2} + \cdots + \alpha_\rho + \beta_\rho X_{t-\rho} + s_t$$

Modelo Vector Autorregresivo

De igual manera, requiere que la serie temporal sea estacionaria, realizando un modelo predictivo estadístico para el caso de una serie de múltiples variables, donde cada variable ejerce una influencia sobre el resto [Fierro, 2020]. Considerado un modelo Autorregresivo porque cada variable es modelada en función de valores pesados, se tendrá un sistema de ecuaciones con una ecuación variable. Por ejemplo, el sistema de ecuaciones para un modelo VAR (1) de dos variables X1 y X2 sería el siguiente:

Ecuación 5 Sistema de ecuaciones

$$X_{1,t} = \alpha_1 + \beta_{11,1} X_{1,t-1} + \beta_{12,1} X_{2,t-1} + s_{1,t}$$

$$X_{2,t} = \alpha_2 + \beta_{21,1} X_{1,t-1} + \beta_{22,1} X_{2,t-1} + s_{2,t}$$

2.6.2 Métodos predictivos de aprendizaje automático

Si la serie no es estacionaria se toman logaritmos, diferencias o transformaciones. El mecanismo de las redes neuronales se inspira en la estructura de una red neuronal biológica, construida a través de neuronas artificiales simples conectadas entre sí, como los cerebros biológicos, para calcular las entradas recibidas y producir los resultados de una suma ponderada de entradas El resultado luego es alimentado a una función no lineal, llamada función de activación, para generar el resultado final.

Dado que se presentan varias series de tiempo que presentan una naturaleza no lineal, se hace necesario utilizar otras técnicas fuera de las clásicas para realizar la predicción de éstas y así obtener modelos más eficientes. En los últimos años las Redes Neuronales Artificiales (RNA) han sido exitosamente aplicadas como herramienta en la predicción de series de tiempo en un amplio rango de problemas en áreas de comercio, industria y ciencia. Las RNA son útiles en la predicción de series de tiempo dado que, a diferencia de los métodos estadísticos clásicos, son capaces de capturar las relaciones lineales y no lineales entre los datos debido a su estructura no lineal que permite un modelo con más grados de libertad.

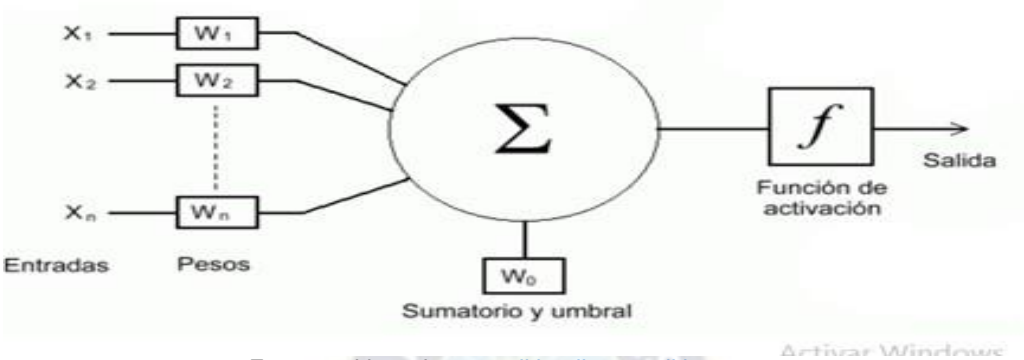
Perceptrón

Una neurona es determinada como un procesador elemental tal que, a partir de un vector de entrada, procedente del exterior o de otras neuronas, proporciona una única respuesta o salida, creando algoritmos que puedan modelar distintos problemas. La facultad de las redes neuronales proviene de su capacidad para aprender con los datos de entrenamiento y relacionarlos con la variable a predecir, creando un mapeo matemático, esta estructura puede aprender a representar variables en diferentes escalas y combinarlas en variables más complejas.

Las neuronas biológicas son células nerviosas que constituyen los elementos primordiales del sistema nervioso o central. Una neurona es capaz de recibir información desde miles de otras neuronas, procesarla y luego generar una nueva información que

enviará a otras con las que está conectada. Es así que se crea la definición de una neurona artificial o procesador elemental que opera como una unidad de procesamiento de información que es fundamental para la operación de una red neuronal.

Figura 2.16 Redes Neuronales



Fuente: sitio web www.wikipedia.org.wiki.perceptron

Un Perceptrón es un modelo de una sola neurona artificial, precursor de una red neuronal que consiste en un conjunto de entradas numéricas, pesos sinápticos asociados a las entradas, una entrada extra denominada “bias”, una regla de propagación que es la suma ponderada, una función de activación, y por último una salida numérica [Fierro, 2020].

Las entradas reciben los datos o parámetros que le permiten decidir a la neurona si estará activa o no, normalmente se presentan como x_1, x_2, ..., xn.

Entre la entrada y el núcleo se tienen los pesos (w_1, w_2, ..., wn) que presentan la memoria de la red, estos pesos sinápticos son coeficientes que parametrizan las capas, y definen la intensidad de la interacción de las entradas sobre las salidas. En el núcleo se realizan todas las operaciones necesarias para determinar la salida de la neurona; el proceso que se realiza en el núcleo varía dependiendo de la red neuronal que se esté trabajando. Teniendo una función de activación que mapea la suma de las señales ponderadas a la señal de salida. Sin una función de activación la salida del Perceptrón consistiría únicamente de dos operaciones lineales, luego este sería capaz de aprender únicamente transformaciones lineales o afines.

El término arquitectura de la red está relacionado con el diseño estructural de la red y busca determinar los siguientes elementos: la cantidad de entradas y salidas, la cantidad

de nodos ocultos, la función de red y de activación asociada a cada nodo, la forma en que los nodos están interconectados, la dirección que sigue la información, y la selección de un conjunto de datos adecuado, para realizar el entrenamiento y la validación del modelo obtenido.

Función de activación

Hay varios tipos de funciones de activación y su utilización depende del objetivo de la red y de la capa en la misma. La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1). Se buscan funciones que las derivadas sean simples, para minimizar con ello el coste computacional.

Tipos de función de activación:

- Sigmoide: La función Sigmoide transforma los valores introducidos a una escala (0,1), donde los valores altos tienden de manera sintónica a 1 y los valores muy bajos tienden de manera asintótica a 0.

Ecuación 6 Función Sigmoide

$$f(x) = \frac{1}{1 - e^{-x}}$$

Sus características:

- Satura y mata el gradiente.
- Lenta convergencia.
- No está centrada en el cero.
- Está acotada entre o y 1.
- Buen rendimiento en la última capa.

- Tanh - Tangent Hyperbolic- Tangente hiperbólica: Transforma los valores introducidos a una escala (-1,1), donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1.

Ecuación 7 Función Tangente hiperbólica

$$f(x) = \frac{1}{1+e^{-2x}} - 1$$

Sus características:

- Muy similar a la Sigmoide.
- Satura y mata el gradiente.
- Lenta convergencia.
- Centrada en 0.
- Está acotada entre -1 y 1.
- Se utiliza para decidir entre una opción y la contraria.
- Buen desempeño en redes recurrentes.

- ReLu – Rectified Lineal Unit: La función ReLu transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.

Ecuación 8 Función ReLu

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

Sus características:

- No está acotada.
- Se pueden morir demasiadas neuronas.
- Se comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

- Leaky ReLu - Rectified Lineal Unit: La función Leaky ReLu transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos según entran.

Ecuación 9 Función Leaky ReLu

$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ ax \; for \; x \geq 0 \end{cases}$$

Sus características:

- Similar a la función ReLu.
- Penalizada los negativos mediante un coeficiente rectificador.
- No está acotada.
- Se comporta bien con imágenes.

- Buen desempeño con redes convolucionales.

- Softmax – Rectified Lineal Unit

  La función Softmax transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1.

Ecuación 10 Función Rectified Lineal Unit

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}$$

Sus características:

- Se utiliza cuando queremos tener una representación en forma de probabilidades.
- Está acotada entre 0 y 1.
- Muy diferenciable.
- Se utiliza para normalizar tipo multiclase.
- Buen rendimiento en las últimas capas.

La capa de entrada simplemente pasa los valores de entrada a la próxima capa. Las capas siguientes a la capa de entrada son llamadas capas ocultas por no estar directamente expuestas a los valores de entrada. A la última capa se llama capa de salida, es la responsable de devolver el o los resultados calculados por la red.

Función de pérdida

Se mide la diferencia entre el valor esperado y el valor calculado por la red, este es el trabajo de la función de pérdida, toma las predicciones de la red y los valores verdaderos de la variable, luego calcula la diferencia entre sí, utilizando el valor calculado como realimentación para ajustar los pesos, de forma que se reduzcan las diferencias.

Algoritmo de optimización

Se utiliza para minimizar o maximizar una función objetivo. Determina la actualización minimizando la función de pérdida, basándose en la técnica" Descenso de Gradiente".

Entrenamiento

Entrenamiento Para el entrenamiento de una red neuronal mínimamente se requiere de las capas de la red, los datos de entrada, los objetivos de salida, la función de pérdida y el algo- ritmo de optimización. En este esquema, la red compuesta por capas conectadas, mapea los datos de entrada a las predicciones obtenidas. Luego, la función de pérdida compara dichas predicciones con los objetivos de salida esperados, obteniendo así una medida del error. El algoritmo de optimización utiliza dicha medida para actualizar los pesos sinápticos de la red neuronal. Inicialmente a los pesos de la red se le asignan valores aleatorios, pero a partir de cada muestra de datos que la red procesa la función de pérdida se minimiza, luego la diferencia entre los valores de salida esperados y calculados se minimiza, y los pesos sinápticos se ajustan. Es decir, que los pesos de la red se actualizan a partir de los errores calculados para cada muestra. Esto representa en términos generales el proceso de entrenamiento. El resultado final, luego de suficientes iteraciones es un modelo que representa o generaliza los datos involucrados [Fierro, 2020].

El proceso de aprendizaje se puede dividir en tres grandes grupos de acuerdo a sus características [Isasi Viñuela y Galván León, 2004]:

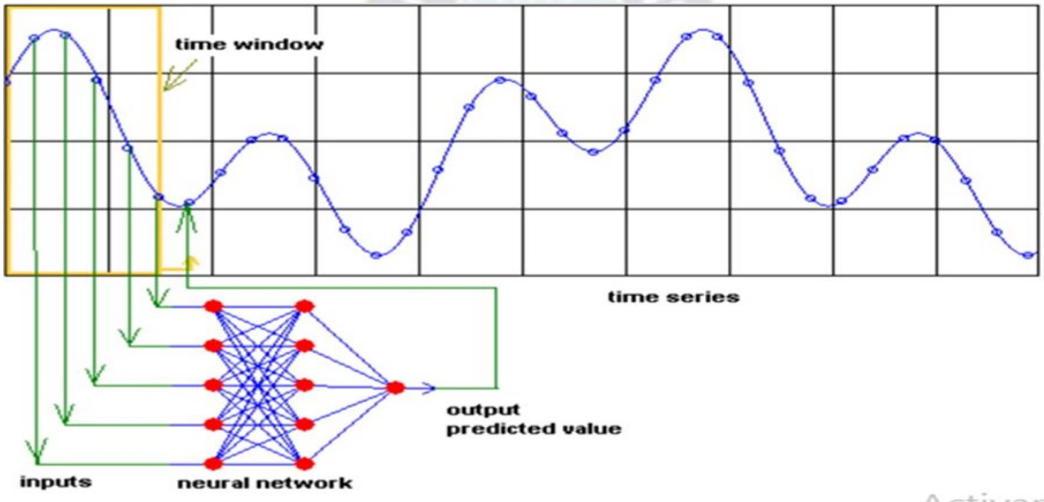
- Aprendizaje supervisado: Se presenta a la red un conjunto de patrones de entrada junto con la salida esperada. Los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada.
- Aprendizaje no supervisado: Se presenta a la red un conjunto de patrones de entrada. No hay información disponible sobre la salida esperada. El proceso de entrenamiento en este caso debería ajustar sus pesos en base a la correlación existente entre los datos de entrada.
- Aprendizaje por refuerzo: Este tipo de aprendizaje se ubica entre los dos anteriores. Se le presenta a la red un conjunto de patrones de entrada y se le indica a la red si la salida obtenida es o no correcta. Sin embargo, no se le proporciona el valor de la

salida esperada. Es útil en casos donde no se conoce la salida exacta que debe proporcionar la red.

2.7 Redes neuronales aplicadas a series de tiempo

La idea base reside en que los valores futuros son dependientes de los valores pasados, es decir, buscando en los valores pasados de una serie temporal se puede predecir su comportamiento en el futuro.

Figura 2.17 Redes neuronales aplicadas a series temporales



Fuente: Predicción de Series Temporales con Redes Neuronales (Fierro, 2020)

Perceptrón Multicapa

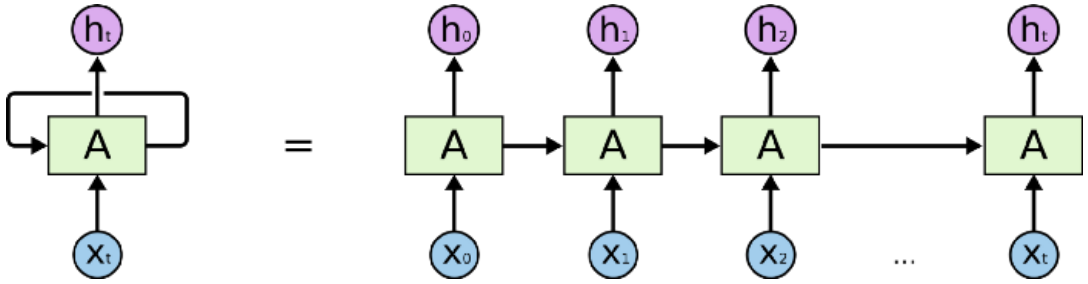
Los perceptrones multicapa (MLP) son las formas más básicas de redes neuronales. Un MLP consta de tres componentes: una capa de entrada, un montón de capas ocultas y una capa de salida.

Redes neuronales recurrentes

Una red neuronal recurrente (RNN) es un tipo de red muy adecuada para datos de series temporales. Las RNN procesan una serie temporal paso a paso, manteniendo un estado interno que resume la información.

La RNN cuenta con una recursión o recurrencia debido a que este tipo de redes realizan la misma operación sobre todos los elementos de la secuencia compartiendo información obtenida del elemento anterior. Puede ser representada de una manera compacta como se muestra a continuación.

Figura 2.18 Representación básica de una Red Recurrente



Fuente: Predicción de Series Temporales con Redes Neuronales (Fierro, 2020)

Nos referimos a las RNN como que operan en una secuencia que contiene vectores **x(t)** con el índice de paso de tiempo **t** que va de **1** a **t**. Por lo general, también un vector de estado oculto **h(t)** para cada paso de tiempo **t**.

Normalmente la red se suele desdoblar en una secuencia finita, intentando predecir el siguiente elemento de la secuencia mirando únicamente N pasos atrás, de manera que sea viable entrenar y obtener resultados de esta [Jianqiang, 2016].

LSTM (Long Short Term Memory)

Son un tipo especial de red neuronal recurrente capaces de aprender dependencias temporales largas, ya que conservan el error y además lo pueden propagar a través del tiempo y las capas. Las neuronas de las LSTM's contienen información fuera del flujo normal de la red recurrente en una celda cerrada.

2.7.1 Librerías para aprendizaje automático

En esta sección se presentará las librerías utilizadas en *Python* para realizar un aprendizaje automático:

Scikit-learn

Scikit-learn, es una librería software libre creada por David Cournepeau en Junio de 2007. En esta se implementan funcionalidades ´útiles para el aprendizaje automático tales como métodos de pre procesamiento de datos, de particionado de los mismos o incluso distintos clasificadores como vecinos próximos, random forest o máquinas de vectores de soporte. Nos da también una implementación del Perceptrón multicapa [Scikit.org].

Tensorflow

Tensorflow es una librería para aprendizaje automático implementada en un principio para *Python* y ahora también disponible para JavaScript. Fue creada por Google y en 2015 se cambió su licencia a código abierto [Cade M, 2015].

Esta librería implementa múltiples funcionalidades de aprendizaje automático y es muy reconocida por su utilidad en la implementación de modelos de redes neuronales. Su funcionamiento se basa en operaciones con tensores. Según la definición de tensor, es un vector o matriz de n-dimensiones que representa todos los tipos de datos. Los valores de un tensor contienen un tipo de datos idéntico con una forma conocida (o parcialmente conocida). La forma de los datos es la dimensionalidad de la matriz.

Su uso se ha popularizado gracias a la versatilidad y facilidad que da a la hora de desarrollar modelos de redes neuronales que se ejecuten sobre GPUs. Esto es muy importante sobre todo a la hora de crear modelos complejos y de entrenar con grandes cantidades de datos ya que reduce en gran medida los tiempos de entrenamiento. Otra de las ventajas a destacar es que el usuario solo tiene que definir el grafo de la red, Tensorflow realizará el algoritmo de retro propagación de manera automática [Roman. 2018].

Keras

Keras es una librería de código abierto de redes neuronales, que se implementa sobre otros *frameworks* de aprendizaje automático como Tensorflow [keras, 2018]. La definición de framework en sistemas informáticos es referido a una estructura en capas que indica qué tipo de programas pueden o deben ser construidos y cómo se interrelacionan.

CAPÍTULO 4. MARCO PRÁCTICO

3 Desarrollo de la mejora del sistema de información

De acuerdo al Decreto Supremo N° 2159 [Dec,], que aprueba el Reglamento Técnico para el Diseño, Construcción, Operación, Mantenimiento y Abandono de Plantas de Gas Natural Licuado-GNL y Estaciones de Regasificación [Bol,],en el Artículo 49 de Reportes mensuales de volúmenes, indica que el Licenciatario, quién es la persona jurídica a la que el Ente Regulador es decir la ANH otorga una Licencia de Operación de una Planta de GNL y/o Estación de Regasificación, debería presentar reportes mensuales de los volúmenes comercializados de GNL en la Planta de GNL conforme a Resolución Administrativa emitida por el Ente Regulador. También se considera falta leve, el incumplimiento en el envío de la información de movimiento de los volúmenes comercializados, en este tipo de infracciones el Ente Regulador emitirá una intimación de cumplimiento al Licenciatario. Es, por tanto, que, para el cumplimiento del presente reglamento, las estaciones de regasificación presentan un reporte diario, a la Gerencia de Industrialización de YPFB y la misma deriva el reporte a la Agencia Nacional de Hidrocarburos.

Figura 3.1 Planilla de Reporte de las ESR's

YPFB
La fuerza que transforma Bolivia

REPORTE DE ESR

GIND

FECHA: 19/4/2021

N°	ESR'S	USUARIOS	SALDO INICIAL (Sm3)	RECIBIDO DE PLANTA GNL Sm3 (GNL)	ENTREGA DE GAS DOMICILIARIO (Sm3)	ENTREGA DE GAS GNV (Sm3)	SALDO FINAL (Sm3)	FECHA DE DESPACHO CISTERNA	VOLUMEN DESPACHADO (Sm3)	PLACA /SERIE	FECHA ESTIMADA ENTREGA	COMENTARIO
1	ACHACACHI	2.475	11.291	-	1.500	-	9.791					
2	ASCENCIÓN DE GUARAYOS	527	9.407	-	158	1.005	8.244					
3	CABEZAS	268	11.210	-	200	-	11.010					
4	CARANAVI	1.756	13.788	-	624	-	13.164					
5	CHALLAPATA - HUARI	4.474	3.403	-	537	-	2.866					
6	COBIJA	700	5.154	-	210	-	4.944					
7	COPACABANA	1.080	4.205	-	299	-	3.906					
8	COROICO	629	3.916	-	164	-	3.752					
9	DESAGUADERO	543	21.650	-	127	150	21.374					
10	GUANAY	268	4.917	-	300	-	4.617					
11	GUAYARAMERIN	495	8.613	-	248	-	8.565					
12	HUANUNI	3.557	15.506	-	1.688	132	13.686					
13	LLALLAGUA - UNCÍA	6.295	16.864	-	3.148	-	13.717					
14	MOIRA	219	15.311	-	110	-	15.202					
15	RIBERALTA	1.328	9.167	-	664	-	8.503					
16	ROBORÉ	745	10.505	-	373	-	10.133					
17	RURRENABAQUE	590	496	-	41	-	455					
18	SAN BORJA	693	2.757	-	80	-	2.677					

Fuente: ANH

Con los reportes obtenidos, se creó un historial de datos con el objetivo de generar mediante este, una visualización completa de los datos, para simplificar de la mayor manera el trabajo del usuario.

3.1 Obtención de los datos de la Planilla de Reporte Diario

Para manejar los datos del historial, se debe entender de donde vienen los mismos y como se realiza el manejo de las unidades.

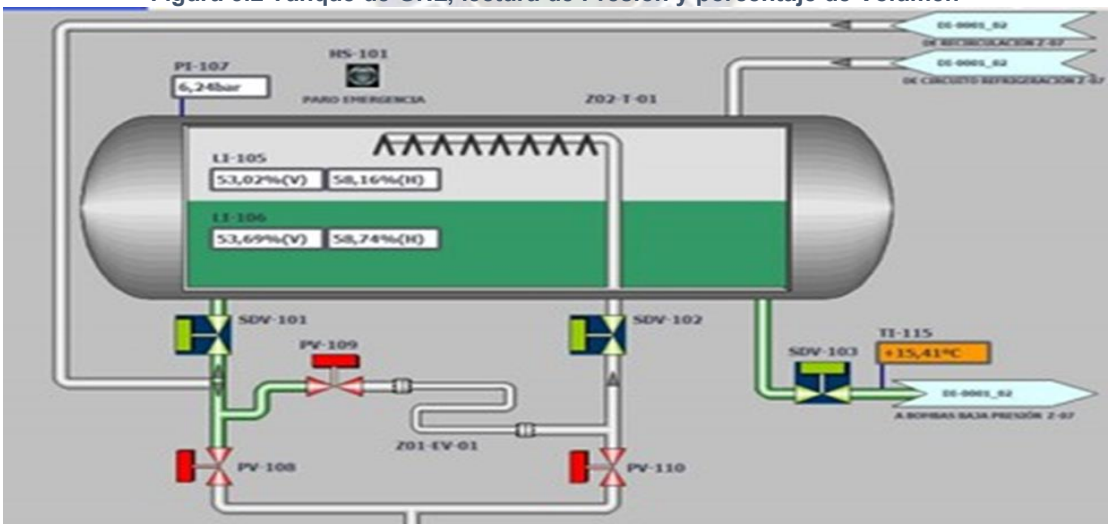
El tanque de almacenamiento tiene un transmisor de presión y un transmisor indicador de nivel. La presión del depósito, se mantiene dentro de unos límites de trabajo establecidos en base a dos sistemas que incorpora los propios tanques.

- Circuito PPR (Puesta en Presión Rápida)
- Circuito Economizador

Los mismos que son explicados y mencionados en el capítulo 2 del presente proyecto.

3.1.1 Presión y porcentaje de volumen del depósito criogénico de GNL

Figura 3.2 Tanque de GNL, lectura de Presión y porcentaje de Volumen



Fuente: ANH

- Estimación de volumen de la fase líquida y gas

Tomando como ejemplo lo calculado por el simulador Hysys, es decir los datos de la presión y el porcentual de volumen, tenemos:

Ecuación 11 Estimación de volumen de la fase líquida y gas

$$\text{Volumen fase liquida} = (\%\text{Volumétrico}) * (\text{Volumen del tanque})$$

Sabemos que los tanques que presentan las Estaciones son de una capacidad de 20, 80, 100 m^3, dependiendo de la localidad en la cual se está realizando el cálculo, se tiene que tomar el valor respectivo. En este caso, tomaremos como valor 80m3, para el ejemplo.

Volumen fase líquida = 53 % $*$ 80m^3 = 42, 4m^3 (GNL)
Volumen fase gas = (100 %- %Volumétrico) *(Volumen del Tanque)
Volumen fase gas = (100 %-53 %)*80m^3 = 37, 6m^3(gas frío)

- Cálculo de la densidad de la fase líquida y gas

Para calcular la densidad del gas se hará uso del simulador Hysys y se empleará la composición del certificado (que se deja con la descarga de GNL) y se estimará un flujo másico de 1 kg/h.

Figura 3.3 Cromatografía del Gas Natural

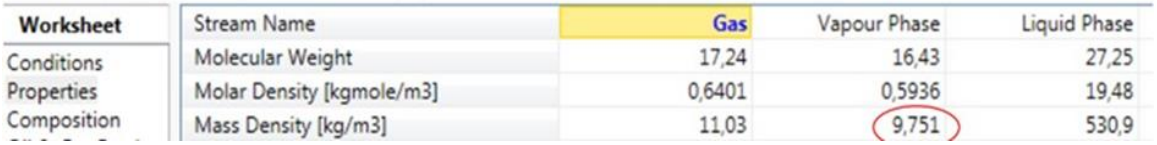
N2	CO2	C1	C2	C3	iC4	nC4	iC5	nC5	C6	Total
0,2151	0,0000	93,6618	6,1021	0,0202	0,0004	0,0006	0,0000	0,0000	0,0000	100,000

Fuente: ANH

Para la densidad de la fase líquida se tomaría el valor del certificado de GNL, en el cual nos indica el valor de la densidad 440, 50 m3, con un poder calorífico de 1038,5 BTU/PC.

En la obtención de la densidad fase gas, para fines prácticos, la empresa estatal trabaja con una temperatura constante que es igual a -100 °C, en la cual la obtención de la densidad dependerá de la presión obtenida por el transmisor de presión. Se toma datos de 6,24 barg y se estima esta temperatura y un flujo másico de 1 kg/h.

Figura 3.4 Densidad Másica



Worksheet	Stream Name	Gas	Vapour Phase	Liquid Phase
Conditions	Molecular Weight	17,24	16,43	27,25
Properties	Molar Density [kgmole/m3]	0,6401	0,5936	19,48
Composition	Mass Density [kg/m3]	11,03	9,751	530,9

Fuente: ANH, Hysis

Cálculo de volumen estándar en el tanque de almacenamiento

Ecuación 12 Determinación del volumen estándar

Volumen Std en la fase líquida = (volumen del líquido) * (densidad del líquido) / (densidad Std del gas)

Para la densidad estándar de gas se estimará una densidad de 0,715 Std kg/m3.
Volumen Std en la fase líquida:

$$42{,}4m^3 \frac{440{,}25\ kg/m^3}{0{,}71kg/m^3} = 26{,}291\ Stdm^3$$

Volumen Std en la fase gas = (volumen del gas) *(densidad del gas) / (densidad Std del gas)

Volumen Std en la fase gas:

$$37{,}6m^3 \frac{9{,}751\ kg/m^3}{0{,}71kg/m^3} = 516\ Stdm^3$$

Volumen Std Total = (Volumen Std en la fase gas) +(Volumen Std en la fase líquida)
Volumen Std Total = 26, 291$Stdm^3$ + 516$Stdm^3$ = 26, 807$Stdm^3$

- Determinación de los días de autonomía

El cálculo de los días de autonomía, lo realizan de la siguiente manera:

Después de determinar el volumen total que se tiene, restan el volumen de gas natural consumido, que se encuentra en la variable de. “Entrega de Gas Domiciliario” más el de

“Entrega de GNV”, esto para obtener un factor de seguridad, posterior a esto se realiza la división con el consumo total teniendo, así como producto los días de autonomía de la ESR.

El consumo de la ESR variará de acuerdo a la población, pero para fines de cálculo se estimará un consumo de 445 $Stdm^3$.

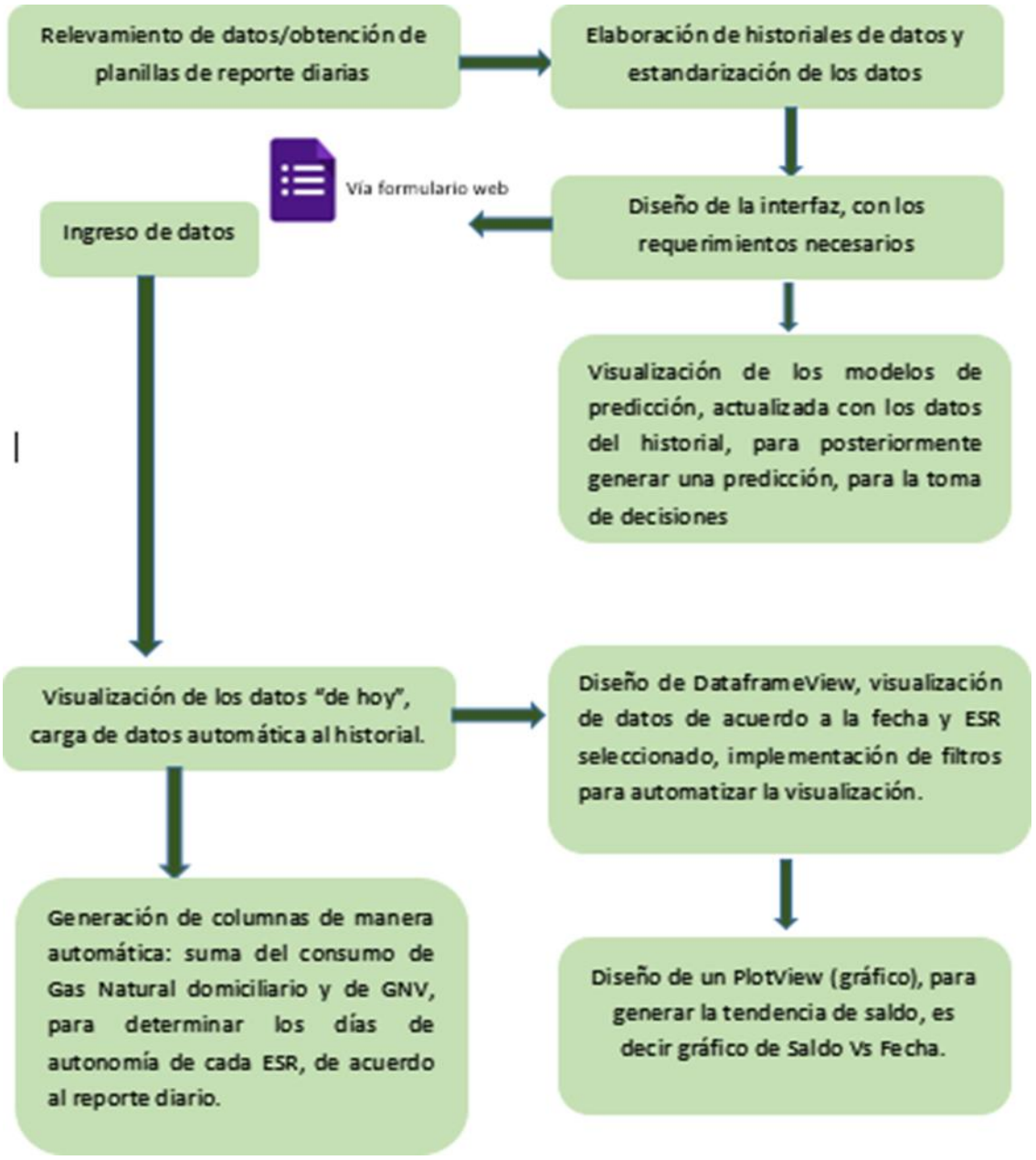
Ecuación 13 Determinación de los días de autonomía

Días de autonomía= (Volumen Std Total-consumo diario) / (consumo diario)

Días de autonomía= (26,807 $Stdm^3$ - 600 $Stdm^3$) / (600 $Stdm^3$) = 43 días

3.2 Esquema General del Proyecto

Figura 3.5 Esquema de Trabajo



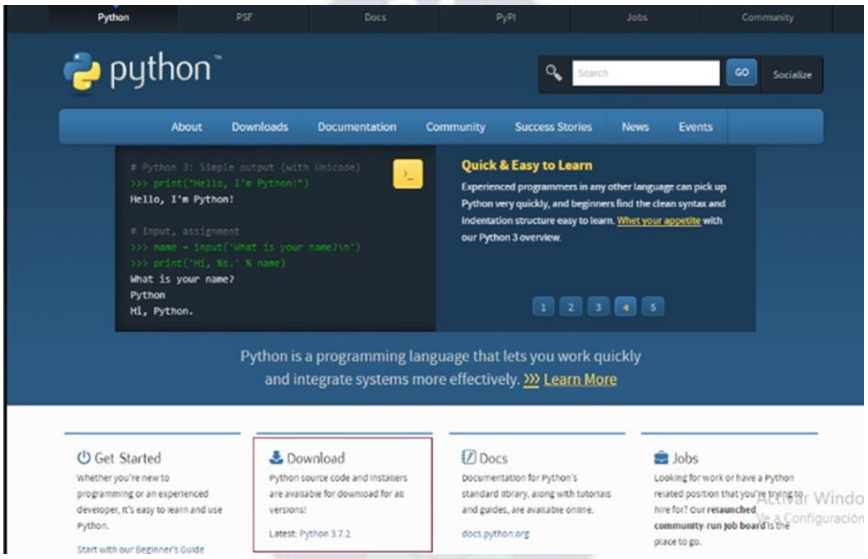
Fuente: Elaboración propia

3.3 Especificaciones de Requerimientos

3.3.1 Instalación del lenguaje de programación Python

Se debe ingresar a la página oficial de Python: https://www.python.org/ y dar clic en descargar Python 3.8.5 que se encuentra en la parte inferior de la pantalla, la cual es la última versión a descargar.

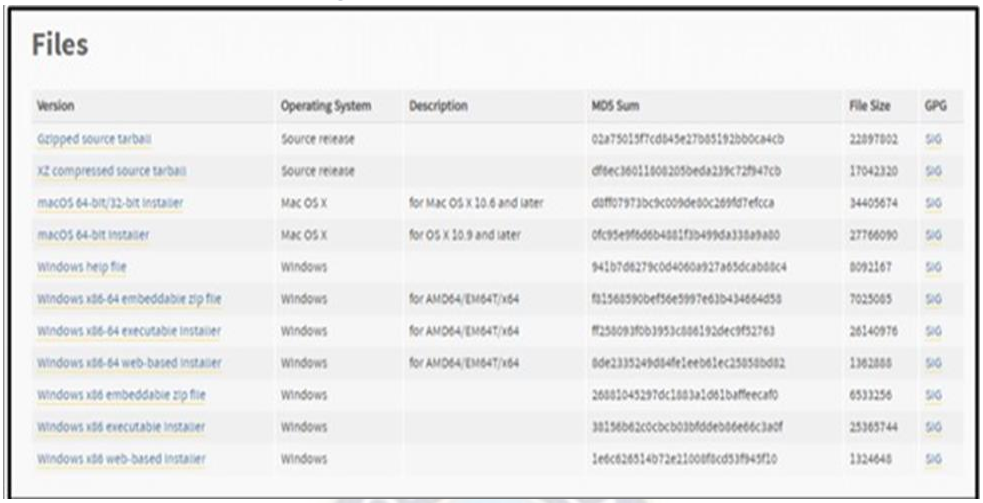
Figura 3.6 Ventana emergente, instalación



Fuente: www.python.org

La ventana emergente muestra la lista de opciones de descarga para los diferentes sistemas operativos:

Figura 3.7 Sistemas Operativos



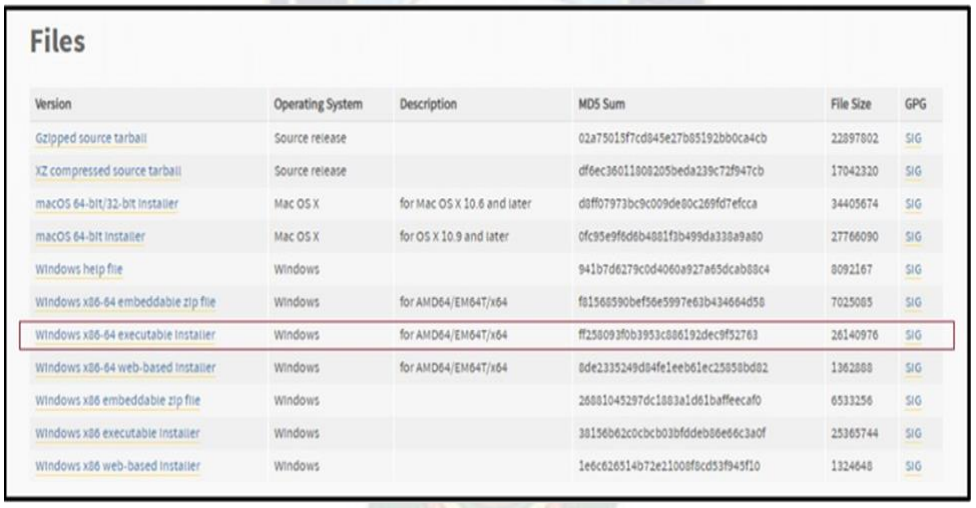
Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		02a75015f7cd845e27b85192bb0ca4cb	22897802	SIG
XZ compressed source tarball	Source release		df6ec36011808205beda239c72f947cb	17042320	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	d8ff07973bc9c009de80c269fd7efcca	34405674	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	0fc95e9f6d6b4881f3b499da338a9a80	27766090	SIG
Windows help file	Windows		941b7d6279c0d4060a927a65dcab88c4	8092167	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f81568590bef56e5997e63b434664d58	7025085	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	ff258093f0b3953c886192dec9f52763	26140976	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	8de2335249d84fe1eeb61ec25858bd82	1362888	SIG
Windows x86 embeddable zip file	Windows		26881045297dc1883a1d61baffeecaf0	6533256	SIG
Windows x86 executable installer	Windows		38156b62c0cbcb03bfddeb86e66c3a0f	25365744	SIG
Windows x86 web-based installer	Windows		1e6c626514b72e21008f8cd53f945f10	1324648	SIG

Fuente: www.python.org

La instalación de Python para Windows 7/8/10 deberá escoger “Windows executable Installer”.

Figura 3.8 Instalación Windows executable Installer



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		02a75015f7cd845e27b85192bb0ca4cb	22897802	SIG
XZ compressed source tarball	Source release		df6ec36011808205beda239c72f947cb	17042320	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	d8ff07973bc9c009de80c269fd7efcca	34405674	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	0fc95e9f6d6b4881f3b499da338a9a80	27766090	SIG
Windows help file	Windows		941b7d6279c0d4060a927a65dcab88c4	8092167	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f81568590bef56e5997e63b434664d58	7025085	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	ff258093f0b3953c886192dec9f52763	26140976	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	8de2335249d84fe1eeb61ec25858bd82	1362888	SIG
Windows x86 embeddable zip file	Windows		26881045297dc1883a1d61baffeecaf0	6533256	SIG
Windows x86 executable installer	Windows		38156b62c0cbcb03bfddeb86e66c3a0f	25365744	SIG
Windows x86 web-based installer	Windows		1e6c626514b72e21008f8cd53f945f10	1324648	SIG

Fuente: www.python.org

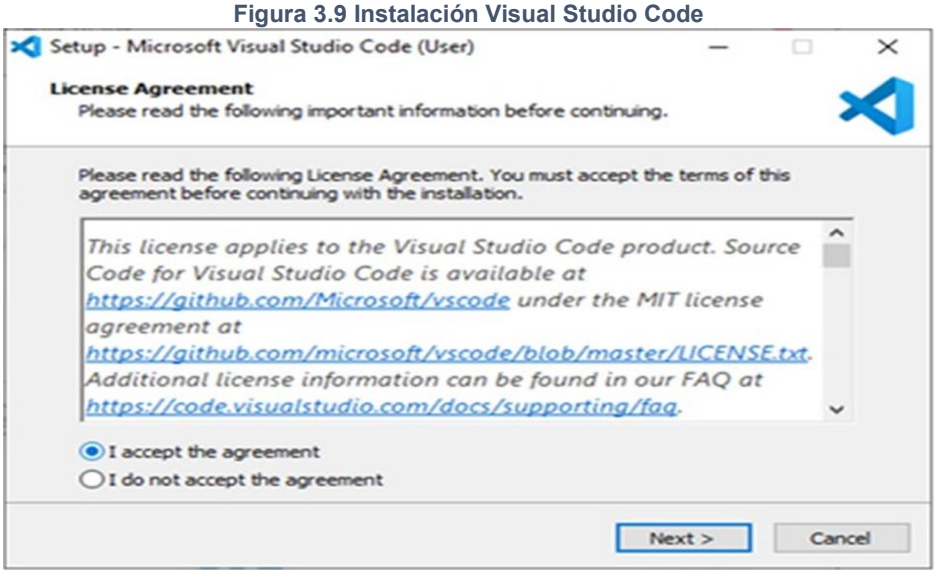
Completa la descarga, ejecutar el archivo Python-3.8.5-amd64.exe el cual contiene la última versión del programa Python.

3.3.2 Instalación Microsoft Visual Studio Code

Ingresar a la página de Microsoft Visual Studio Code en Academic Software y hacer clic en el botón verde “Descargar Visual Studio Code”, para descargar el instalador. Abrir el

archivo de instalación *.exe* en tu carpeta de descargas para iniciar la instalación. Lee y acepta el acuerdo de licencia.

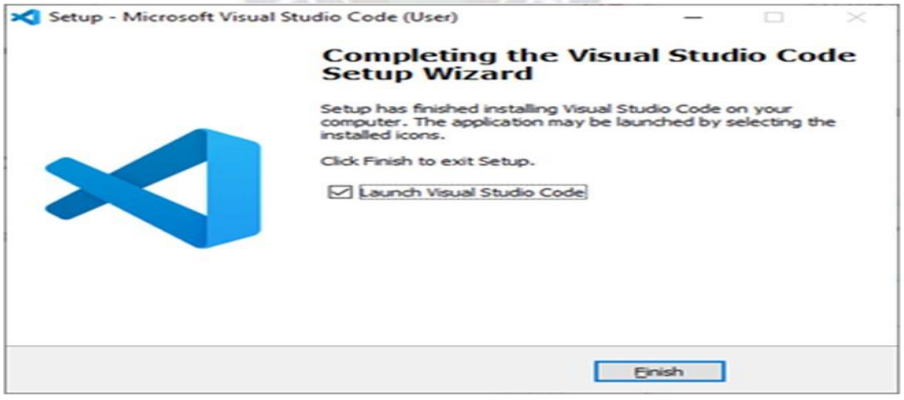
Figura 3.9 Instalación Visual Studio Code



Fuente: www.code.visualstudio

Se puede cambiar la ubicación de la carpeta de instalación o mantener la configuración predeterminada. Continuar con el procedimiento de instalación.

Figura 3.10 Visual Studio Code



Fuente: www.code.visualstudio

3.3.3 Diseño y Desarrollo del software

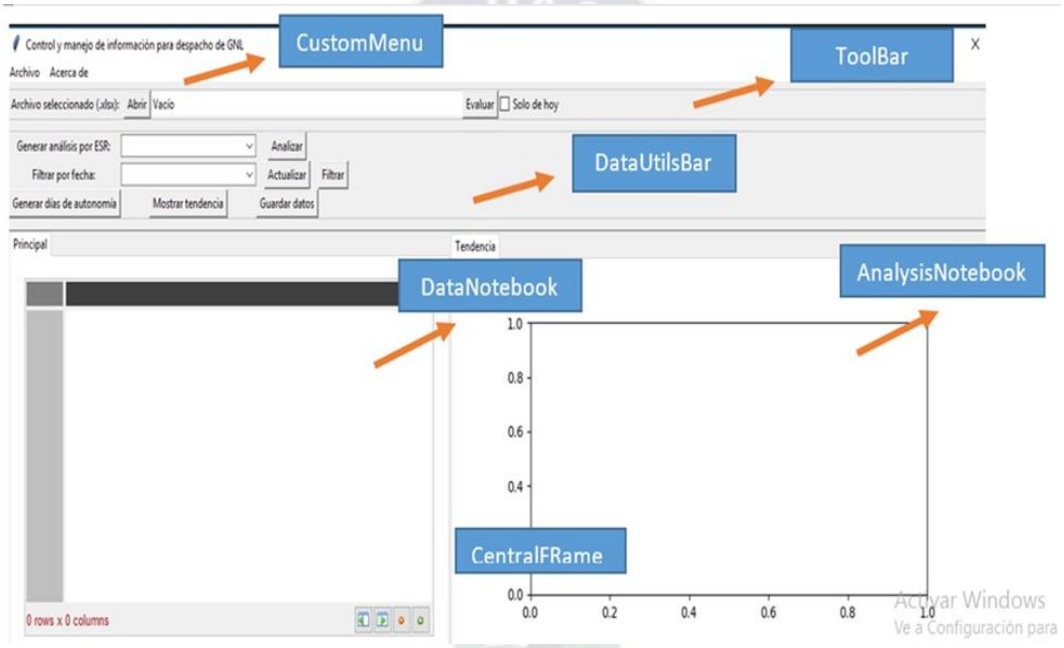
La elección del diseño se realizará con el objetivo de mantener la sencillez en el código a desarrollar, pero otorgando la funcionalidad mínima requerida. Las funciones necesarias

se agruparán de acuerdo al objetivo que deben cumplir, de esta manera se simplificará de la mayor manera, el entendimiento del código y el diseño gráfico de la interfaz.

3.4 Diseño de la interfaz Gráfica de Usuario (GUI)

Para el desarrollo de la interfaz se utilizó la librería Tkinter, realizando la división de la ventada en cuatro partes, es decir en cuatro clases, para su diseño.

Figura 3.11 Diseño de la interfaz gráfica del usuario



Fuente: Elaboración propia

Se utilizó el método" _init _ " de la clase, es un método especial de una clase en Python. El objetivo fundamental del método es inicializar los atributos de objeto que creamos.

1. El método _init_ es el primer método que se ejecuta cuando se crea un objeto.
2. El método _init_ se llama automáticamente. Es decir, es imposible de olvidarse de llamarlo ya que se llamará automáticamente.

Otras características importantes, tomadas en cuenta para utilizarlo son:

- Se ejecuta inmediatamente luego de crear un objeto.
- El método _init_ no puede retornar dato.

- Puede recibir parámetros que se utilizan normalmente para inicializar atributos.

3.5 Requerimientos funcionales

Se diseñó la interfaz de acuerdo a los requerimientos que se identificaron, para un mejor manejo del código este se dividió en varias clases y secciones, el criterio para esta división se basó en la definición de funciones necesarias para generar los componentes de la interfaz y su funcionalidad.

Se tiene cuatro clases para definir las partes de la interfaz mencionadas anteriormente, posterior a esto se generó cuatro clases donde se tiene:

- Class AnalysisNotebook (), definición de la clase que muestra la gráfica de "Tendencia" de las pestañas (tab_frame) de ESR seleccionada.
- Class DataframeView (), define la vista tabulada de datos para pasar el dataframe de pandas a interfaz gráfica de tkinter, es decir se lee el excel seleccionado y se lo muestra directamente en la interfaz y no así generando una ventana emergente.
- Class DataNotebook (), clase que genera el componente de varias páginas, se generá pestañas para mostrar vistas diferentes de tablas, se genera funciones para obtener:
  - La referencia de la pestaña seleccionada.
  - Adición de pestañas nuevas.
  - Reinicio para generación de pestañas, se vuelve a cargar la página principal, cuando se da "click" al botón "Evaluar".
- Class PlotView(), componente de display para gráfico de tendencia, definiendo la función para mostrar la gráfica, obteniendo la referencia de la pestaña seleccionada en el DataNotebook().

Se describirá las clases creadas para el diseño de la interfaz y las funciones definidas:

3.5.1 CustomMenu

El desarrollo para el diseño de la interfaz, se realizó con la librería Tkinter, dónde se define el Menú de ventana y se añade las opciones "Archivo" y "Acerca de", esto para el boceto general de creación de una ventana principal.

Figura 3.12 Código-Clase CustomMenu

```
class CustomMenu(Menu):
    """Menu de ventana"""
    def __init__(self, root):
        super(CustomMenu, self).__init__(root)

        file_menu = Menu(self, tearoff=0) # Menu "Archivo"
        about_menu = Menu(self, tearoff=0) # Menu "Acerca de"

        menu_options = {
            "Archivo": file_menu,
            "Acerca de": about_menu
        }

        # Adicionando los menus al menu principal
        for menu_label, menu_ref in menu_options.items():
            self.add_cascade(label=menu_label, menu=menu_ref)
```

Fuente: Elaboración propia

Como se puede ver en el código se utiliza la función *super* que nos permite invocar y conservar un método o atributo de una *clase padre* (primaria) desde una clase hija (secundaria) sin tener que nombrarla explícitamente. Esto nos brinda la ventaja de poder cambiar el nombre de la *clase padre* (base) o hija (secundaria) cuando queramos y aun así mantener un código funcional, sencillo y mantenible.

En este caso nuestra *clase padre* es *Menu,* la cual es un widget que nos permite crear todo tipo de menú que puedan ser utilizados por nuestras aplicaciones, posterior a esto creamos una nueva instancia la cual su principal función, como ya dijimos, es establecer un estado inicial en el objeto para inicializar sus atributos. Solicitamos a *super* llamar a la *clase Padre* y especificamos los nuevos atributos para *Hijo.*

3.5.2 ToolBar

El inicio de la definición de la función se realiza de la misma manera que la anterior sección, creamos el constructor de la clase especificando atributos, solicitamos a *super* llamar a la clase padre “Frame” y posterior a esto especificamos los nuevos atributos.

Figura 3.13 Código- Clase Toolbar

```
class ToolBar(Frame):
    """Barra superior con opciones de carga de archivo excel"""
    def __init__(self, root):
        super(ToolBar, self).__init__(root, name="toolbar", highlightbackground="#c4c4c4", highlightthickness=1)
        self.grid(sticky="we", padx=5, pady=5)

        self.selected_filepath = StringVar()
        self.checkbox_state = BooleanVar(False)
        self.selected_filepath.set("Vacío")

        self.prompt_label = Label(self, text="Archivo seleccionado (.xlsx): ")
        self.file_selection = Label(self, textvariable=self.selected_filepath, width=60, anchor="w", bg="white")

        self.browser_button = Button(self, text="Abrir", command=lambda: browse_files(self.selected_filepath))
        self.evaluate_button = Button(self, text="Evaluar", command=lambda: load_data(self.selected_filepath.get()
        self.daily_checkbox = Checkbutton(self, text="Solo de hoy", variable=self.checkbox_state)

        self.prompt_label.grid(row=0,column=0)
        self.browser_button.grid(row=0,column=1)
        self.file_selection.grid(row=0,column=2)
        self.evaluate_button.grid(row=0,column=3)
        self.daily_checkbox.grid(row=0,column=4)
```

Fuente: Elaboración propia

Se realiza el código para el diseño de la barra superior con opciones de carga de archivo Excel, donde se crea una clase ToolBar, generando opciones para:

- Abrir: donde browsefiles es la función gráfica de explorador de archivos para obtener archivo Excel o csv a cargar, que se encuentra en una sección en el código.
  "Función grafica de explorador de archivos para obtener archivo excel a cargar"
  filetypes = (
  (Todos los archivos, *),
  (Archivo de excel,.xlsx),
  (Archivo CSV,.csv))
  Esta función crea una nueva ventana, generando un cuadro de diálogo, donde se le da los tipos de documentos que quiere abrir, limitando solo la poción de Excel y csv.

- Evaluar: Se utiliza el método *get()* que retorna el valor del ítem con la llave especificada, en nuestro caso el archivo Excel o csv y de igual manera se verifica si el *checkbox_state()* esta marcado para habilitar la opción "Solo de hoy", para visualizar los datos del día.

- Procesar: Después realizamos el procesamiento de datos con la función *load_data()* que se encuentra en la sección *FileHandling.py* donde se realiza el manejo del

archivo , convirtiéndolo a “dataframe” con la librería de pandas, se actualiza las referencias de la columna de ESR´s y las fechas, para realizar los filtros respectivos con los datos disponibles. Utilizando las funciones definidas en la sección *DataHandling.py* donde se realiza el manejo de datos.

Figura 3.14 Procesamiento de Datos del archivo cargado

```
if file_extension in allowed_extensions:
    # Cargar el archivo de datos como dataframe
    if file_extension == ".xlsx":
        df = pd.read_excel( filepath )
    elif file_extension == ".csv":
        df = pd.read_csv( filepath , sep=',"')
    # Preprocesamiento de datos
    df = preprocess_main_data(df, only_daily)
    # Obtener referencia a la tabla en interfaz
    dataframeview_ref = gui_root.nametowidget("centralframe.datanotebook.dataframeviewmain")
    # Actualizar el dataframe mostrado
    dataframeview_ref.update_dataframe(df)
    # Obtener referencia a las opciones de manejo de datos
    data_utils_ref = gui_root.nametowidget("datautilsbar")
    # Actualizar los ESR's disponibles
    data_utils_ref.update_unique_esr()
    # Actualizar las fechas de filtrado disponibles
    data_utils_ref.update_available_dates()
    # Obtener referencia al notebook en interfaz
    notebook_ref = gui_root.nametowidget("centralframe.datanotebook")
    # Descartar pestañas generadas antes
    notebook_ref.clean_extra_tabs()
    return
```

Fuente: Elaboración propia

La opción “Solo de hoy”, se la diseño para evitar que se repitan datos, en nuestra base datos, es decir el usuario podrá cargar una planilla de reporte diario, pero si esta contiene la fecha de días pasados serían omitidos con solo hacer “click” en esta función.

Cabe recalcar que esta Planilla de Reporte Diario es la obtenida por la creación del formulario en Google para optimizar el tiempo de llenado.

3.5.3 DataUtilsBar

En esta clase se establece las operaciones de manipulación de datos sobre el dataframe cargado, en la anterior sección. Se define la función de carga automática de ESR únicos para ofrecerlos en lista, añadiendo el botón “Analizar”, definida por la función:

Figura 3.15 Código-Función para filtrar por ESR

```
from gui.DataframeView import DataframeView

def gen_esr_analysis(selected_esr, gui_root):
    # Referencia a la vista paginada de tablas
    notebook_ref =  gui_root.nametowidget("centralframe.datanotebook")
    # Referencia a la vista tabulada del dataframe
    df_table_ref =  notebook_ref.dataframe_view_main
    # Obtencion de datos del dataframe
    main_df = df_table_ref.table.model.df
    # Filtrado para el ESR seleccionado
    esr_df = main_df.loc[main_df["ESR"] == selected_esr]
    # Preparando df de ESR seleccionado por fecha
    esr_df = preprocess_esr_data(esr_df)

    esr_df_view = DataframeView(notebook_ref, name=f"dataframeview{selected_esr}")
    esr_df_view.update_dataframe(esr_df)
    notebook_ref.add_tab(f"{selected_esr}", esr_df_view)
```

Fuente: Elaboración propia

Se muestra las referencias obtenidas en la anterior función, para que el usuario pueda escoger la ESR que desee, se filtre el mismo y se muestre en la interfaz.

La siguiente barra de herramientas, determina el filtro por fecha, definida por la función:

Figura 3.16 Código-Función para filtrar por Fecha

```
def filter_by_date(notebook_ref, filter_date_str):
    # Funcion de filtro por fecha
    # El filtro tiene una presicion de dias (ano - mes - dia)
    df_view = notebook_ref.get_current_tab_df()
    df = df_view.table.model.df
    filter_date = datetime.strptime(filter_date_str, '%d-%m-%Y') # Se formatea la fecha seleccionada como filtro
    for attribute in ["FECHA", "Marca temporal"]:
        try:
            # Se filtra la informacion por fecha
            df = df.loc[
                (df[attribute].dt.year == filter_date.year) &
                (df[attribute].dt.month == filter_date.month) &
                (df[attribute].dt.day == filter_date.day)
            ]
            # Se actualiza la vista del dataframe
            df_view.update_dataframe(df)
            return df
        except KeyError:
            print(f"Columna {attribute} no encontrada al preprocesar como valores de fecha")

    return df
```

Fuente: Elaboración propia

De igual manera al filtrado por ESR funciona la función de filtro por fecha, el filtro tiene una precisión de año/mes/día, tomando todos los valores visualizados al cargar el Excel del historial de datos, y realizando su posterior actualización en la vista en el dataframe de la

interfaz. De igual manera utilizamos la sentencia “try - except” debido a que existe la posibilidad de no encontrarse esos valores de fecha.

En esta sección se implementó también la barra de herramientas para diseñar un botón que genere los “Días de autonomía”, que es la división entre el saldo final y la entrega total, siendo esta última la suma entre la Entrega de Gas Domiciliario y la Entrega de Gas GNV, generando una nueva columna con este nombre. Se debe tener en cuenta que debido a que es una base de datos muy amplia existe la posibilidad de división entre cero, por lo que se debe crear un tratamiento de errores con la sentencia, es así que usamos “try − except”, que nos permite capturar un eventual error y hacer algo al respecto en lugar de detener el programa directamente.

Figura 3.17 Código-Automatización para generar los días de autonomía

```
def generate_extra_columns(notebook_ref):
    # Funcion que engloba la generacion de las columnas
    # adicionales "ENTREGA TOTAL" y "DIVISION"
    df_view = notebook_ref.get_current_tab_df()
    df = df_view.table.model.df
    df = generate_entrega_final_col(df)
    df = generate_division_col(df)
    df_view.table.redraw()
    return df

def generate_entrega_final_col(df):
    # Generacion de la columna "ENTREGA TOTAL"
    df["ENTREGA TOTAL"] = df["ENTREGA DE GAS DOMICILIARIO"] + df["ENTREGA DE GAS GNV"]
    return df

def generate_division_col(df):
    def custom_division(i, a, b):
        try:
            return a/b
        except Exception as e:
            # print(f"Error: Division en fila #{i}:{a}/{b}:{e}")
            return 0

    # Generacion de la columna "DIVISION"
    not_division_zero_filter = (df["ENTREGA TOTAL"].isnull()) & (df["ENTREGA TOTAL"] != 0)
    df["DIVISION"] = df.apply(lambda x: custom_division(x.name, x["SALDO FINAL"], x["ENTREGA TOTAL"]), axis=1)
    df["DIVISION"] = df['DIVISION'].replace(np.nan, 0).apply(np.floor)
    return df
```

Fuente: Elaboración propia

Para la evaluación de los datos, se crea el botón “Mostrar tendencia” que generará una gráfica *Fecha versus Saldo Final*, para identificar la tendencia de vaciado de tanque y determinar la fecha probable de llenado de tanque, cabe recalcar que se podrá filtrar por ESR, para evaluar por población la tendencia de vaciado del tanque.

Figura 3.18 Código-Función de generación de gráfico estadístico

```
def get_plot(ax, notebook_ref):
    # Funcion de generacion de grafico estadistico
    # para los atributos (FECHA) vs (SALDO FINAL)
    df_view = notebook_ref.get_current_tab_df() # Obtener referencia a la vista actual de datos
    df = df_view.table.model.df # Obtener el dataframe de dicha vista
    date_column = None
    date_attr = None
    # Obtener el nombre de la columna de fecha
    expected_date_columns = ["FECHA", "Marca temporal"]
    for date_column in expected_date_columns:
        if date_column in df.columns:
            date_attr = date_column
            break

    # Asegurarse de que existe el atributo de tiempo (fecha) en la data
    if date_attr is None:
        print(f"Error: Ninguna columna de fecha encontrada (Se esperaba alguna en:{expected_date_columns})")
        return

    # Fijamos la data de la columna "SALDO FINAL"
    saldo_column = "SALDO FINAL"
    x, y = df[date_attr], df['SALDO FINAL']

    # Generamos la grafica
    ax.plot(x, y, linewidth=.5, color="black")
    ax.scatter(x, y, marker="x", color="black")
    ax.set_title(f'{date_attr} vs {saldo_column}')

    # Retornamos la referencia de la grafica para ser
    # mostrado en interfaz mas adelante
    return ax
```

Fuente: Elaboración propia

Para automatizar el proceso de guardado de Reportes Diarios, y subir los datos de manera automática a nuestra base de datos, para su posterior Análisis de generación de días de autonomía y mostrar su tendencia si se desea, se crea el botón “Guardar datos”, donde lo primero es generar el código para buscar la ruta en la que se encuentra el historial de datos, para realizar la carga de la vista actual del dataframe, es decir el usuario primero debe subir la planilla de reporte y hacer “click” en el botón ”Evaluar” para visualizarlo en el área del dataframe en la parte de la izquierda inferior de la interfaz, es así que se debe leer los datos fila por fila e identificar que en nuestra base de datos no exista el registro de esa fecha, esto para evitar datos repetidos de guardado, posterior a esto se logra automatizar el ordenado de la base de datos por ESR y Fecha, para su posterior análisis con las otras funciones que se tiene de las opciones de botones.

Figura 3.19 Código-Función de guardado automática de los datos a la BDD

```
def save_data_in_historic(notebook_ref):
    # Obtener ruta actual
    current_path = os.getcwd()
    # Definir la ruta de caga de los datos historicos
    HISTORIC_DATA_TABLE_FILE = "PARTE DIARIO ESR´S.xlsx"
    # Obtener ruta de los datos historicos
    historic_data_path = os.path.join(current_path, "data", HISTORIC_DATA_TABLE_FILE)
    # Asegurarnos que existe el archivo
    if not os.path.isfile(historic_data_path):
        print(f"Error: Archivo de datos historicos no encontrado:{historic_data_path}")
    # Cargar los datos a un dataframe
    master_df = pd.read_excel(historic_data_path)
    # Limpiar dataframe historico
    master_df = preprocess_main_data(master_df, False)
    # Definir atributo de fecha en historico
    expected_date_columns = ["FECHA", "Marca temporal"]
    for column_name in expected_date_columns:
        if column_name in master_df.columns:
            master_df_date_col = column_name
            break
```

Fuente: Elaboración propia

Figura 3.20 Código- Análisis del dataframe

```
for _, row in current_df.iterrows():
    # Obtener fecha y ESR
    row_date = row[current_df_date_col]
    row_esr = row["ESR"]
    # Filtrar dicha fecha y ESR en el dataframe total (historico)
    df_filter = (master_df[master_df_date_col].dt.year == row_date.year) & \
        (master_df[master_df_date_col].dt.month == row_date.month) & \
        (master_df[master_df_date_col].dt.day == row_date.day) & \
        (master_df["ESR"] == row_esr)
    filtered_master_df = master_df.loc[df_filter]
    # Revisar si existen datos de esta fecha y ESR
    if not filtered_master_df.empty:
        # Si existe: marcar error
        print(f"Error: Datos por editar ya existen en la tabla histórica: {master_df_date_col}={row_date} & ESR={row_esr}:
    else:
        # Sino: apendar
        print(f"Actualizando con datos nuevos")
        row = row.rename({current_df_date_col: master_df_date_col}, axis='columns')
        row["ESR"] = row["ESR"].upper()
        master_df = master_df.append(row, ignore_index = True)

print( "Informacion historica actualizada" )
# Dejar el dataframe final ordenado a partir de los ESR's y las fechas
master_df = master_df.sort_values(by=["ESR", master_df_date_col])
# Guardar el resultado final
master_df.to_excel(historic_data_path, index=False)
print( "Informacion historica guardada en excel historico" )
```

Fuente: Elaboración propia

3.5.4 DataNotebook

Es el componente que utilizamos para mostrar vistas diferentes de tablas, denominando al “DataframeView” el componente de visualización de interfaz, en donde se aplica distintas funciones para adecuar el tamaño de Excel para visualizar en el dataframe, ya sea añadiendo pestañas o quitándolas respectivamente de acuerdo al número de columnas que posee los datos, conformando una clase denominada “DataNotebook” que es el componente de varias páginas para mostrar vistas diferentes de tablas.

Figura 3.21 Código-Clase Notebook

```
class DataNotebook(Notebook):
    """Componente de varias páginas para mostrar vistas diferentes de tablas"""
    def __init__(self, root, name):
        super(DataNotebook, self).__init__(root, name=name)
        self.gui_root = root
        self.tab_frames = {}
        self.dataframe_view_main = DataframeView(self, name="dataframeviewmain") # Componente de visualizacion de interfaz
        self.tab_frames["Principal"] = self.dataframe_view_main # Pestana de vista principal de datos cargados
        self.add(self.dataframe_view_main, text="Principal", padding=20)

    def get_current_tab_df(self):
        # Funcion para obtener la referencia a la pestana seleccionada actualmente
        current_tab_ref = self.select()
        current_tab_name = str(current_tab_ref)[40:]
        if current_tab_name == "main": current_tab_name = "Principal"
        return self.tab_frames[current_tab_name]

    def add_tab(self, new_tab_name, new_tab_view):
        # Funcion de adicion de pestanas nuevas
        self.tab_frames[new_tab_name] = new_tab_view
        for view_name, view in self.tab_frames.items():
            self.add(view, text=view_name, padding=20)
```

Fuente: Elaboración propia

3.5.5 AnalysisNotebook

Se genera una vista tabulada de datos para los resultados de análisis para el dataframe seleccionado, creando un componente “PlotView” para la visualización de la gráfica en la parte inferior derecha de la interfaz, teniendo el ploteo de Tendencia de Saldo, que tomara como referencia al “Notebook” de donde proviene la data.

Figura 3.22 Código-Clase AnalysisNotebook

```
from tkinter import Frame
from tkinter.ttk import Notebook

from .PlotView import PlotView

class AnalysisNotebook(Notebook):
    """Vista tabulada de datos para los resultados de analisis para el dataframe seleccionado"""
    def __init__(self, root, name):
        super(AnalysisNotebook, self).__init__(root, name=name)
        self.gui_root = root
        self.tab_frames = {}
        self.plot_view = PlotView(self, name="plotview") # Componente para la visualizacion de la grafica
        self.tab_frames["Tendencia de saldo"] = self.plot_view # Pestana de grafico

        self.add(self.plot_view, text="Tendencia", padding=20)

    def delete_tabs(self):
        for tab_ref in self.winfo_children():
            tab_ref.destroy()
```

Fuente: Elaboración propia

La función “plot” recibe un conjunto de valores “x” e “y” y los muestra en el plano definido por los ejes como puntos unidos por líneas.

Figura 3.23 Código-Clase PlotView

```
class PlotView(Frame):
    """Componente de display para grafico de tendencia"""
    def __init__(self, root, name):
        super(PlotView, self).__init__(root, name=name)
        self.gui_root = root
        self.grid(sticky="we", padx=5, pady=5)

        self.datanotebook_ref = root.gui_root.data_notebook # Referencia al notebook de donde proviene la data
        # Formateo base de la grafica
        self.figure_ref = Figure(figsize=(7, 3.2))
        self.figure_ax = self.figure_ref.add_subplot(111)

        # Componente de tkinter de graficos pyplot para interfaz
        self.plot_tk = FigureCanvasTkAgg(self.figure_ref, self)
        self.plot_view = self.plot_tk.get_tk_widget()

        self.plot_view.grid(row=0, column=0)
```

Fuente: Elaboración propia

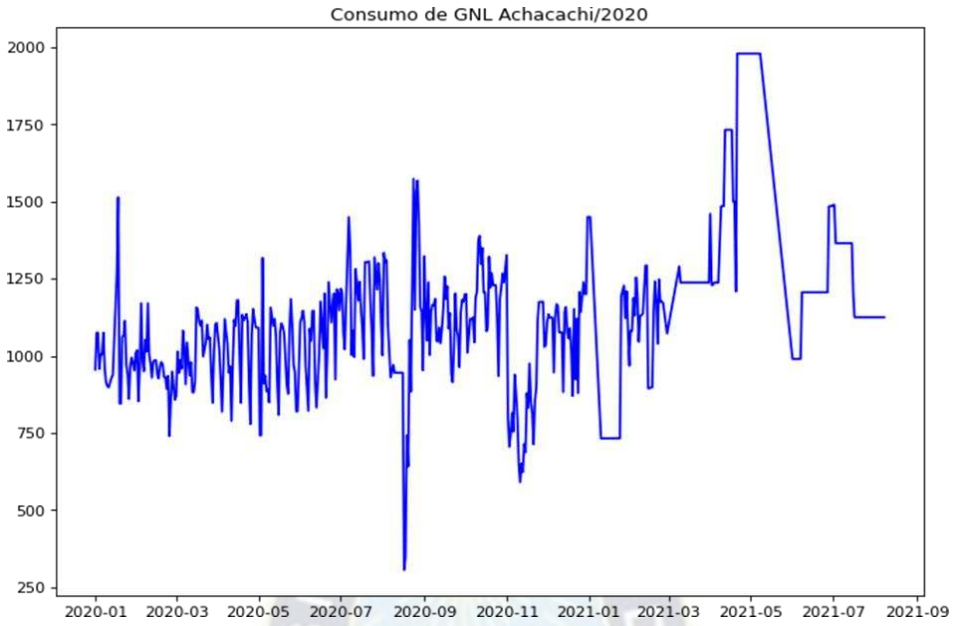
Es importante que ejecutemos la función “plt.show ()” al final del código. Si no lo hacemos, el código devolverá el resultado en forma de texto (haciendo referencia al conjunto de líneas o bloques gráficos que se han generado). Lo importante de generar una gráfica en la misma interfaz, es que no se genera una ventana emergente, por lo tanto, podemos visualizar en nuestra misma interfaz con un componente en tkinter de gráficos *pyplot*.

3.6 Análisis para generar el modelo de predicción

3.6.1 Descomposición de la serie de tiempo

Para evaluar los datos entregados y determinar si se puede utilizar un modelo estadístico se basa primeramente en la visualización de la misma, donde se evalúa si presenta una tendencia y/o estacionalidad, que pueda ayudarnos a crear un modelo estadístico. Trabajaremos primeramente con la localidad de Achacachi, para posteriormente aplicar el mismo modelo a las demás, y si es necesario realizar modificaciones que se adecuan al comportamiento de consumo de cada Estación Satélite de Regasificación.

Figura 3.24 Consumo de GNL-Achacahi, Periodo 2020 y parte del 2021



Generando la gráfica se puede ver que no existe una estacionalidad, y mucho menos una tendencia. Dando una explicación técnica a esto, nos encontramos que en la Planta se da situaciones en las que es necesario generar un venteo debido a un poco demanda de los consumidores, o generalmente debido a la temperatura de la localidad, y la lectura en los medidores se eleva de manera abrupta, e incluso existen situaciones en las que el tanque llega a un nivel inferior, en dónde se suministra gas natural a la población de manera limitada, y por supuesto existen outliers que son puntos de la serie que se escapan de lo normal, es por tanto que primero decidimos evaluar la serie temporal para ver la existencia de una tendencia y/o estacionalidad.

El enfoque de descomposición de series de tiempo, supone que la serie Y (1), ..., Y (N) puede ser expresada como suma o producto de sus cuatro componentes:

- Tendencia
- Componente estacional
- Ciclo
- Término de error aleatorio

Modelo Aditivo:

Ecuación 14 Modelo Aditivo

$$Y(t) = T(t) + E(t) + A(t)$$

Modelo Multiplicativo:

Ecuación 15 Modelo Multiplicativo

$$Y(t) = T(t)E(t)A(t)$$

Donde:

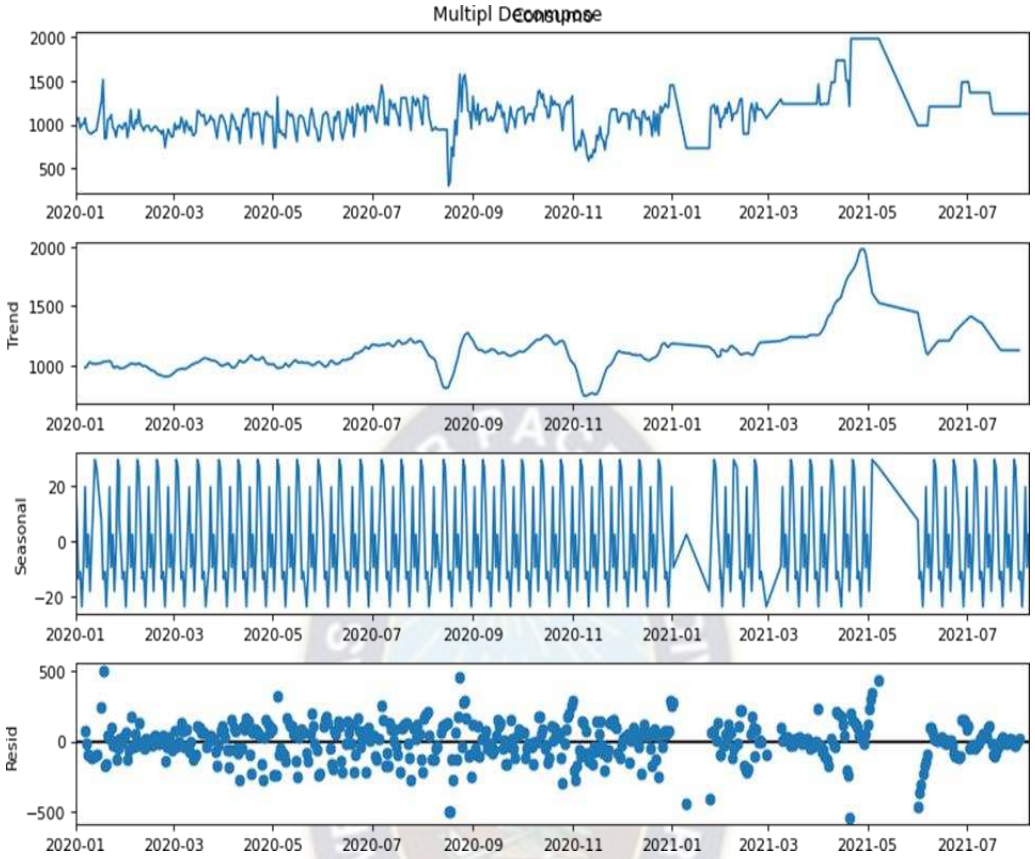
- T: Tendencia de la serie.
- E: Variación Estacional.
- A: Variaciones aleatorias

En Python existen métodos para descomponer automáticamente una serie de tiempo. La biblioteca *statsmodels* proporciona una implementación del método de descomposición clásico en una función llamada *seasonaldecompose*. El cuál requiere que se especifique si el modelo es aditivo o multiplicativo [Jason,.].

Ambos producirán un resultado y se debe tener cuidado de ser crítico al interpretar el resultado. La función *seasonaldecompose ()* devuelve un objeto de resultado que contiene matrices para acceder a cuatro datos de la descomposición.

El modelo aditivo asume que los efectos estacionales son constantes y no dependen del nivel medio de la serie. Por lo contrario, el multiplicativo indica que las componentes estacionales varían en función del nivel medio local desestacionalizado, lo que da a entender que las fluctuaciones estacionales crecen (o decrecen) proporcionalmente con el crecimiento (o de-crecimiento) del nivel medio de la serie.

Figura 3.25 Descomposición de la serie de tiempo - Achacachi



Por ejemplo, el siguiente fragmento que pertenece al consumo de Gas Natural Licuado de la localidad de Achacachi, muestra cómo se descompone una serie en componentes de tendencia, estacionales y residuales, asumiendo un modelo multiplicativo. Podemos ver que no existe ni estacionalidad ni tendencia y existe un centenar de residuos en el gráfico que muestran el ruido que presenta nuestra serie de tiempo.

Se debe validar que la serie de tiempo sea estacionaria en media, y estacionaria en varianza, ya se realizó el análisis gráfico y se evaluó mediante la descomposición de la serie, lo cual no cumple el criterio, además que se cuenta con una base de datos por día, para la cuál es mejor realizar un análisis mediante Deep learning con redes neuronales, para poder encontrar el modelo más acertado debido a la variación en todos sus puntos.

3.6.2 Desarrollo de Forecast avanzado

Lo que modelamos es una neurona que aprende con los datos introducidos, donde se crea una fórmula que se convierte en función de activación, esta función ajusta todos los parámetros que nosotros estamos ingresando. Una sola neurona aprende de lo que nosotros estamos ingresando, es así que tarda menos tiempo en entrenar para predecir los datos.

Lo que utilizaremos para entrenar al modelo es una Red Neuronal Recurrente, dividiremos la secuencia de pasos, en los siguientes:

PASO 1

- Se necesita separar la data en dos grupos, X y y, esta depende del programador, debido a que por ejemplo para cada valor que queremos predecir se necesita las anteriores n observaciones: 1,2,3,4,5,6,7,8 y queremos usar las 6 observaciones anteriores para hacer cada predicción. Siendo que la cantidad máxima que podemos predecir es 2. Es así que se explica que para entender el modelo este funcionara tomando los valores en X para tratar de pronosticar los valores en y.

Como nuestra base de datos comienza de la fecha 01-12-2020 al 08-08-2021, dividiremos la base de datos hasta el 01-12-2020, para entrenar nuestro modelo con los datos que se tiene del año 2021, esto depende de cada uno.

PASO 2

- Después continuamos con la parte de scaler, es decir se tiene que escalar, este término viene de hacer una transformación a nuestros datos a un rango de -1 a +1 para poder utilizar el algoritmo de red neuronal.
- Para poder entrenar, vamos a generar a partir de nuestra data.
  Para iniciar el modelo:

```
model = Sequential()
```

Empezamos a identificar sus parámetros:

- Antes de ingresar a la función de activación los datos pasan por la función de agregación, en la cual se da la sumatoria de todos los pesos, esta permite varias

entradas y las reduce en un valor único, es decir realiza una combinación de las entradas.

- La función de activación, utiliza la función de agregación, a través de una función de penalización, se tienen los métodos mencionados en la parte teórica, los más utilizados son el *linear* y *relu*.

- Después se define la función de optimización para calcular el error cuadrático medio, este error va a minimizar las diferencias nominales entre los valores y las predicciones. El optimizador es un algoritmo que intenta minimizar la función de optimización, en las cuáles tenemos el método *adam, Adagrad y rmsprop*.

- Optimizamos a la red neuronal haciendo uso del vector gradiente, que es un vector con las derivadas parciales de otros parámetros respecto al coste.

Cálculo del error cuadrático medio:

Ecuación 16 Error cuadrático medio

$$C(a_j^L) = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

$$\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$$

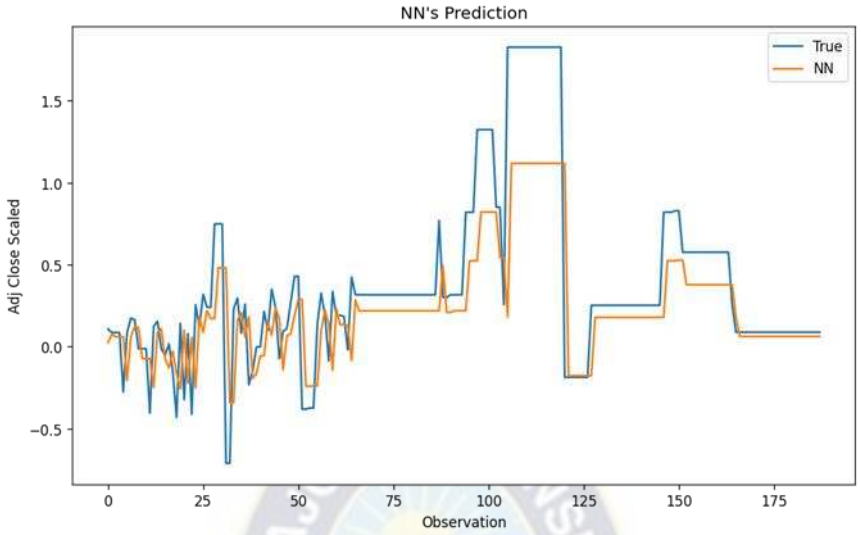
Donde:

$$\frac{\partial C}{\partial a_j^L}$$

Es la derivada parcial de la función de coste respecto a la derivada parcial de la función de activación.

El último análisis que se realiza son los parámetros epochs y el tamaño del batch. Un epoch significa que toda la data será analizada una vez. El tamaño del batch se refiere a la cantidad de observaciones, en nuestro caso tomaremos el tamaño del bloque que tenemos. Al ejecutar el código, el modelo va a entrenarse, esto puede tomar varios minutos dependiendo del hardware y de los parámetros utilizados.

Figura 3.26 Entrenamiento de la Red Neuronal, valores predictivos



Al terminar el entrenamiento, se realiza la predicción de los valores de consumo para el año 2021. En la Figura 3.26 se puede observar los valores predichos por la red neuronal, color naranja, comparados con los reales, color azul, lo que nos da la certeza de que el entrenamiento está funcionando y nuestra RNN está determinando los valores cercanos a los reales. Esta predicción coadyuvará en la operación de determinar los valores de consumo de las localidades y sus fluctuaciones, determinando los meses donde existe un mayor y menor consumo, para aportar a la toma de decisiones en el cronograma de envío de cisternas a cada estación de regasificación para su abastecimiento.

Tabla 3.1 Tabla de comparación de valores reales y valores predichos para el año 2021

Fecha	Real	Predicción
1/2/2021	1081,095	1025,5634
2/2/2021	1081,095	1069,4656
3/2/2021	1187,5669	1069,4656
4/2/2021	1130,1674	1134,5941
5/2/2021	1254,1107	1099,483
6/2/2021	1199,9959	1175,2988
7/2/2021	1045,803	1142,1969
…	…	…
14/3/2021	1237,5	1165,138
15/3/2021	1237,5	1165,138
16/3/2021	1237,5	1165,138
17/3/2021	1237,5	1165,138
18/3/2021	1237,5	1165,138
19/3/2021	1237,5	1165,138
…	…	…
22/4/2021	1980	1619,3228
23/4/2021	1980	1619,3228
24/4/2021	1980	1619,3228
25/4/2021	1980	1619,3228
…	…	…
16/7/2021	1205,825419	1243,1906
17/7/2021	1125,1076	1145,7627
18/7/2021	1125,1076	1096,388
19/7/2021	1125,1076	1096,388
20/7/2021	1125,1076	1096,388
21/7/2021	1125,1076	1096,388
22/7/2021	1125,1076	1096,388
23/7/2021	1125,1076	1096,388
24/7/2021	1125,1076	1096,388
25/7/2021	1125,1076	1096,388

Comparamos los errores, probando diferentes tipos de función de activación entre *relu y linear* y distintas funciones de optimización *rmsprop y Adagrad*.

Nuestros tres modelos para la comparación, es decir para obtener el error cuadrático medio más cercano a 1, que indica una mejor predicción debido al entrenamiento de la red neuronal diseñada. Se basará en los siguientes modelos:

Tabla 3.2 Modelos de entrenamiento

Función de activación	Función de optimización
linear	Adagrad
linear	rmsprop
relu	rmsprop

CAPÍTULO 4 MARCO APLICATIVO

4 Implementación y verificación

4.1 Introducción

Para la implementación, como ya se dijo y se observó en la anterior sección, el lenguaje utilizado ha sido Python, en concreto la versión 3.8.5, la cuál es la última versión que mejora algunas características y optimizaciones.

4.2 Visual Studio Code

Es el entorno físico de desarrollo para la programación del código de este programa, es compatible con él lenguaje a utilizar Python, incluye resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

4.3 Google Colab

“Google Colab” es una herramienta para escribir y ejecutar código Python en la nube de Google. Se utilizó este instrumento debido a que se debe utilizar las librerías keras y sklearn, para el entrenamiento y predicción de los modelos con redes neuronales en las series de tiempo, mismas que son más sencillas de utilizar en “Google Colab” que permite un entorno para llevar a cabo tareas que serían difíciles de realizar en un equipo personal.

4.4 Diseño Tkinter

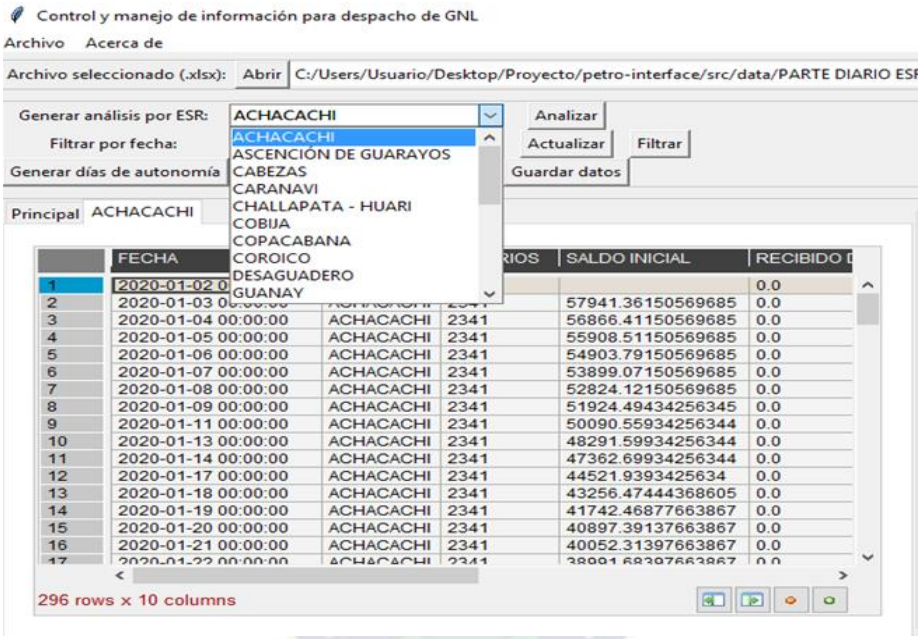
Tkinter es un binding de la biblioteca gráfica para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows [Tk,2021].

4.4.1 Propuesta para el manejo de información

Se implementa las funciones de trabajo con la base de datos, para el almacenamiento de la información requerida. Se automatiza los procedimientos ya descritos anteriormente y se comprueba su funcionalidad.

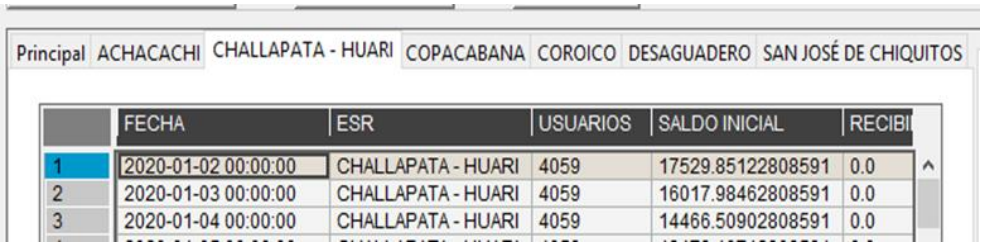
4.4.2 Selección de filtrado por ESR's

Figura 4.1 Funcionamiento de selección de ESR



Se realiza la comprobación del software, seleccionando las distintas opciones que se tiene de acuerdo a las ESR disponibles en la base datos, donde se encuentra la planilla de los partes diarios desde el año 2020 hasta el mes de agosto del 2021. Para un mejor manejo se realizó la creación de pestañas con el nombre de las plantas seleccionadas, es decir el usuario podrá ver de manera separada las estaciones de regasificación que escoja para un mejor análisis, guardando las ya escogidas.

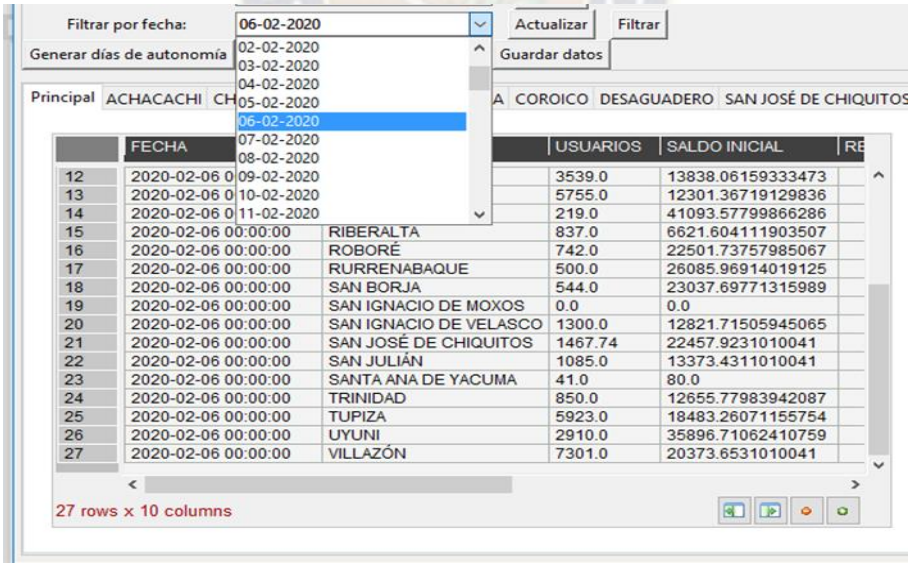
Figura 4.2 Generación de pestañas de acuerdo a la ESR seleccionada



4.4.3 Selección de filtrado por fecha

Se define la función de filtro por fecha, manejando un formato de año/mes/día, se utiliza el bucle *for,* para evaluar todos los datos y comparar con el seleccionado. Se puede realizar esta acción con las distintas pestañas, es decir si el usuario se encuentra en la planilla principal o en las pestañas de las plantas que escogió funcionará de igual manera. Esto para que el usuario pueda manipular todos los datos de acuerdo a su necesidad y así pueda comparar o visualizar los requerimientos necesarios para realizar sus informes respectivos.

Figura 4.3 Vista de filtrado por fecha



El botón *Actualizar*, es utilizado para que se tome los valores de la columna de Fecha mediante un bucle *for* se obtienen las fechas presentes en el documento y se actualiza la lista.

4.4.4 Generar días de autonomía

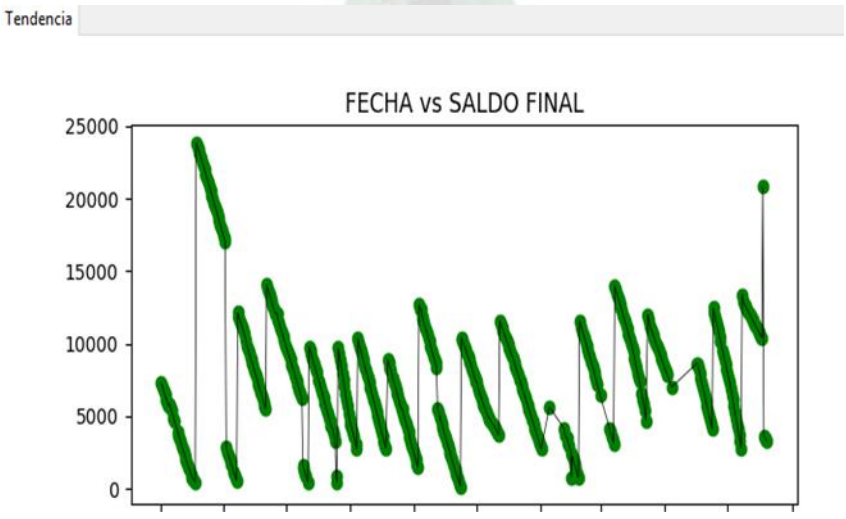
Como se mencionó en la anterior sección del marco práctico, se realiza la división del saldo final entre el consumo de gas domiciliario más el consumo de GNV, esto para calcular un estimado de los días que durará el saldo que se tiene en el tanque.

Para este objetivo se genera dos columnas, una que realiza la suma del consumo de gas domiciliario y consumo de la GNV y otra donde se crea la "División" para sus respectivos días de autonomía, esto se puede realizar con la función *"Solo de hoy"* para que se filtre la fecha actual. Esto ayudará a la toma de decisiones para envío de cisternas, cabe recalcar que esta operación se realiza en la actualidad, realizándolo de manera manual.

4.4.5 Mostrar tendencia

Se observó que es importante mostrar la tendencia del saldo, así el operador podrá observar la declinación de la curva después que se llena el tanque de acuerdo al consumo de cada localidad. Se define la función *get_plot* para generar el gráfico estadístico, se realiza el código para verificar valores vacíos, se añade los parámetros para definir el color y determinar los valores de los ejes.

Figura 4.4 Gráfico de tendencia por Localidad



4.4.6 Guardar datos

Esta sección es la más complicada de desarrollar, se realiza el código para definir la ruta actual del archivo y la ruta en donde se quiere guardar los datos. Y se realiza un bucle para revisar si ya existen esos datos de fecha en nuestro histórico, después realiza el copiado con el método *append* en nuestro Excel.

CAPÍTULO 5 ANÁLISIS ECONÓMICO

5 Introducción Análisis Económico

El objetivo de este capítulo es demostrar que con la utilización de este sistema se obtendrá beneficios para la institución, para calcular el esfuerzo y el costo de desarrollo del software se empleará el modelo COCOMO, que en función del programa expresado en las líneas de código estimadas podemos calcular los valores del esfuerzo y costo.

El Modelo Constructivo de Costos (o COCOMO, por su acrónimo del inglés), es un modelo de base empírica utilizado para estimación de costos de software. Incluye tres sub-modelos, cada uno ofrece un nivel de detalle y aproximación, cada vez mayor, a medida que avanza el proceso de desarrollo del software: básico, intermedio y detallado [Cocomo, 2020].

5.1 Análisis de costos con el modelo COCOMO

Fórmulas que se usarán:

Ecuación 17 Modelo COCOMO, Esfuerzo

$$E = aKLDC^{b}$$

Ecuación 18 Tiempo de desarrollo

$$T = cE^{d}$$

Dónde:

E = Es el esfuerzo expresado en personas por mes.

T= Es el tiempo de desarrollo expresado en meses.

KLDC = Es el tamaño expresado en miles de líneas de código fuente.

Los valores de coeficientes (a, c) y exponentes (b, d) se toman de acuerdo al siguiente cuadro:

Figura 5.1 Valores de coeficientes, método COCOMO

PROYECTO SOFTWARE	a	b	c	d	Descripción
Simple	3,2	1,05	2,5	0,38	Aplicaciones bien comprendidas desarrolladas por equipos pequeños
Moderada	3,0	1,12	2,5	0,35	Proyectos más complejos donde los miembros del equipo tienen experiencia limitada en sistemas relacionados
Incrustada	2,8	1,20	2,5	0,32	Proyectos complejos donde el software es parte de un complejo fuertemente acoplado de hardware, software, reglas y procedimientos operacionales.

Por el tipo de proyecto se tiene los siguientes valores:

a = 3.2 b = 1.05 c = 2.5 d=0.38

El número de líneas de código aproximado del sistema es 400 que incluye para el diseño de toda la interfaz y sus funciones, sin líneas de etiqueta. Se tiene también el código generado para la predicción de modelos por RN de las 27 Estaciones de Regasificación, tomando en cuenta el mejor modelo de código para cada una, se contabiliza 48 líneas de código, multiplicado para el diseño de todas las localidades, tenemos 1296 líneas de código, lo que da un total de 1696.

Entonces reemplazando los valores de los coeficientes, exponentes definidos y con el factor KLDC=1.696 en las ecuaciones se obtiene los siguientes valores:

$E = 3.2 * 1696^{1.05}$ E = 5.57 hombres-mes

$T = 2.5 * 5.57^{0.38}$ T = 4.8 meses

El número de personas que intervienen en el desarrollo del proyecto – sistema viene dado por la siguiente fórmula.

Ecuación 19 Número de personas que intervienen en el desarrollo

$\mathrm{P} = \frac{\mathrm{E}}{\mathrm{T}}$ $\mathrm{P} = \frac{5.57}{4.8}$ P= 1.16 personas

5.2 Costo en el Personal

Según estas cifras se necesita una persona trabajando alrededor de 5 meses, suponiendo que el costo de trabajo de un programador está basado en el pago a nivel Latinoamérica de 500 $ mensuales, entonces tendremos:

Costo en recursos de Personal = 1*5*500

Costo en recursos de Personal=2500 $

5.3 Costo en la Elaboración del Proyecto

Incluyendo los costos de elaboración del proyecto, que incluye principalmente los gastos que se realizaron a lo largo del desarrollo, podemos ver las siguientes:

Tabla 5.1 Costo en la Elaboración del Proyecto

Detalle	Importe ($us)
Internet	80
Cursos de aprendizaje en Python	120
Total	**200**

5.4 Costo Total del Software

El costo total del software se lo obtiene de la sumatoria de los diferentes costos vistos, costo de desarrollo y costo de la elaboración del proyecto. Entonces, tendremos una cantidad aproximada como esta:

Tabla 5.2 Costo Total del software

Detalle	Importe ($us)
Costo de desarrollo	2500
Costo de la elaboración del proyecto	200
Total	**2700**

5.5 Beneficios

Se desarrolla un análisis del antes y después de la utilización del software:

Tabla 5.3 Beneficios de utilización del software

Sin el sistema	**Con el sistema**
No se almacena los datos anteriores, con el cambio de personal en la institución, se perdía la información de fechas anteriores, esto debido a que la información se registraba en distintos Excels que se borraban después de un tiempo.	Se cuenta con una base de datos, que se mantiene en el sistema.
Se tarda bastante tiempo en guardar uno por uno los datos de las 27 Estaciones de Regasificación a la Planilla Histórica, generando susceptibilidad en la copia de los mismos por posibles fallas en el copiado manual que realizan.	Con un botón logramos guardar los datos del Parte Diario a nuestra base de datos, reduciendo el tiempo y eliminando cualquier error de copiado.
Toma de decisiones, evaluando uno por uno los datos que se tienen de las plantas, se generan análisis de los días de autonomía y se evalúa su tendencia de vaciado de los tanques, de manera manual de cada localidad.	Visualización de todas las plantas al mismo tiempo, para determinar posibles variaciones o datos incoherentes, mejora la toma de decisiones de los operadores, generando columnas automáticas de días de autonomía de los tanques y la visualización gráfica del saldo del tanque de las localidades.

CAPÍTULO 6 CONCLUSIONES Y RECOMENDACIONES

6 Conclusiones y Recomendaciones

6.1 Conclusiones

Se realizó la automatización del manejo de información de las 27 Estaciones de Regasificación, se pudo observar que se tiene una amplia base de datos, en donde su manejo se facilita para la toma de decisiones y para un estudio visual del comportamiento de cada una, se realiza la prueba del software, donde se ve una disminución en el tiempo de manejo de datos, donde se tiene el Reporte Diario, y se hace el guardado automático, para posteriormente realizar el cálculo de los días de autonomía, para mejorar su toma de decisiones, además que se logra crear filtros por Estación Satelital de Regasificación y por fecha, para identificar cualquier dato ilógico (puede ser por exceso de venteo), que pueda afectar al medio ambiente y al manejo sostenible del gas natural en nuestro territorio.

Se logra automatizar también, el comportamiento del consumo de cada planta, realizando el gráfico del consumo versus la fecha, cabe recalcar que la planilla que se tomó como referencia, indica el consumo de la población, pero esta no incluye solo este valor, sino también el venteo que se realiza en la planta debido al exceso de presión, es debido a esto que nuestra serie en el tiempo no presenta ni tendencia ni estacionalidad.

Cabe destacar que el desarrollo de este software pretende presentar un modelo de interfaz para el diseño posterior en el sistema operativo de la Dirección de Tecnologías de Información y Comunicación (DTIC) de la Agencia Nacional de Hidrocarburos, donde cuentan con su propio servidor de base de datos, para su posterior implementación en la empresa estatal. La generación del formulario Google, para la obtención de las Planillas de Reporte Diario, se realizó para simular el sistema operativo que se creara en la red de la agencia estatal, donde se pretende instalar las interfaces en las estaciones, pera generar el envió de información en tiempo real vía web del sistema de la ANH.

Asimismo, se realizó la prueba del manejo de la interfaz, sus distintas aplicaciones y las herramientas que brinda, para probar su eficacia y corroborar que realiza la visualización de toda la base de datos recabada desde el año 2020 hasta el agosto del presente año.

El entrenamiento de las redes neuronales para generar los modelos que se adecuan más al comportamiento de consumo de las 27 Estaciones Satélite de Regasificación, tuvieron resultados satisfactorios en el 70 % de las poblaciones, comparando su error cuadrático medio para ver el modelo que más se ajusta a la predicción. Se puede afirmar que la predicción de series temporales mediante la utilización de redes neuronales devuelve resultados muy positivos, esto significa que serán útiles para aplicaciones predictivas. Sin embargo, es necesario realizar mejoras en la base de datos, para que tenga una mayor entrada de entrenamiento.

Es importante recalcar que la estructura de la red neuronal utilizada tiene una estructura que se basa en una sola neurona oculta, ya que una red neuronal con mayores neuronas, implicaría un entrenamiento más extenso y de mayor tiempo, lo que no beneficiaría a la mejora en cuanto a la disminución del tiempo para la toma de decisiones de envío de cisternas a las distintas poblaciones. Asimismo, se sugiere realizar el entrenamiento actualizando cada determinado tiempo mediante la base de datos generada por el guardado automático de las planillas diarias, y así generar un mejor entrenamiento, esto para la utilización en el futuro y la constante mejora de la propuesta presentada por el presente Proyecto de Grado.

Las distintas gráficas que se obtuvieron muestran que existen modelos que determinan de manera eficaz la predicción y en cambio muestran otras pocas poblaciones, la existencia de una variación notable, lo que es debido a la presencia de puntos outliers que afectan al entrenamiento, pero que es importante tenerlos para que el usuario pueda determinar u observar la razón y existencia de esos puntos, que afectan en el mal uso de nuestro recurso natural, debido al poco control existe con los datos.

6.2 Recomendaciones

La interfaz diseñada puede tener mejoras e incluso utilizar el mismo prototipo para otro manejo de planillas, como puede ser el caso de las Planillas de Reporte de la Planta de GNL, las cuales igual son recepcionadas por la agencia estatal, para su evolución y supervisión como cumplimiento de la misión de la institución.

Se recomienda ampliar la base de datos para tener un mejor entrenamiento de la red neuronal, así se podrá generar una base de datos con sumatorias de consumo por mes, dónde se podrá comprobar si verdaderamente no existe una estacionalidad en los datos o un comportamiento de ciclo que se genere cada cierto tiempo.

CAPÍTULO 7. BIBLIOGRAFÍA

7 Bibliografía

- [Adhikari and Agrawal, 2013] Adhikari, R. and Agrawal, R. (2013). An Introductory Study on Time Series Modeling and Forecasting Ratnadip Adhikari R. K. Agrawal.
- [Bol,] Bolivia: Reglamento Técnico para el Diseño, Construcción, Operación, Mantenimiento y Abandono de Plantas de Gas Natural Licuado – GNL y Estaciones de Regasificación, 23 de octubre de 2014.
- [Challenger-Perez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, 2014] Challenger-Perez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, R. A. (2014). El lenguaje de programación Python/The programming language Python.
- [Comisión Nacional Para El Uso Eficiente De La Energía CONUEE, 2017] Comisión Nacional Para El Uso Eficiente De La Energía CONUEE (2017). Derivados del petróleo: Gas Natural Licuado. Movilidad y transporte, pages 1–4.
- [Dec,] Bolivia: Decreto Supremo N º 2159, 23 de octubre de 2014.
- [Del and ESR's, 2015]Del, D. and ESR's, P. (2015). Planta de GNL, ESR y Cisternas Planta de GNL, ESR y Cisternas. Technical report.
- [dat, 2020] La librería Datetime — Aprende con Alf.
- [Fierro, 2020] Fierro, A. A. (2020). Predicción de Series Temporales con Redes Neuronales. page 64.
- [González Duque, 2000]González Duque, R. (2000). Python para todos. Number 6.
- [Gratis,] Gratis, E. Aprendizaje: Pandas.
- [Hidrocarburos and De, 2021] Hidrocarburos, M. D. E. and De, A. S. A. (2021). De Cuentas Inicial 2021. pages 1–132.
- [Hunter Jhon, Dale Darren, Firing Eric, 2021] Hunter Jhon, Dale Darren, Firing Eric, D. M. (2021). Matplotlib: Python plotting — Matplotlib 3.4.3 documentation.
- [Jason,] Jason, B. How to Decompose Time Series Data into Trend and Seasonality.
- [num,] Numpy.
- [pyt,]Welcome to Python.org.
- [Pro,] Programación orientada a objetos en Python: Definición de clases — Solución de problemas con algoritmos y estructuras de datos.
- [os,] os — Interfaces misceláneas del sistema operativo — documentación de Python.

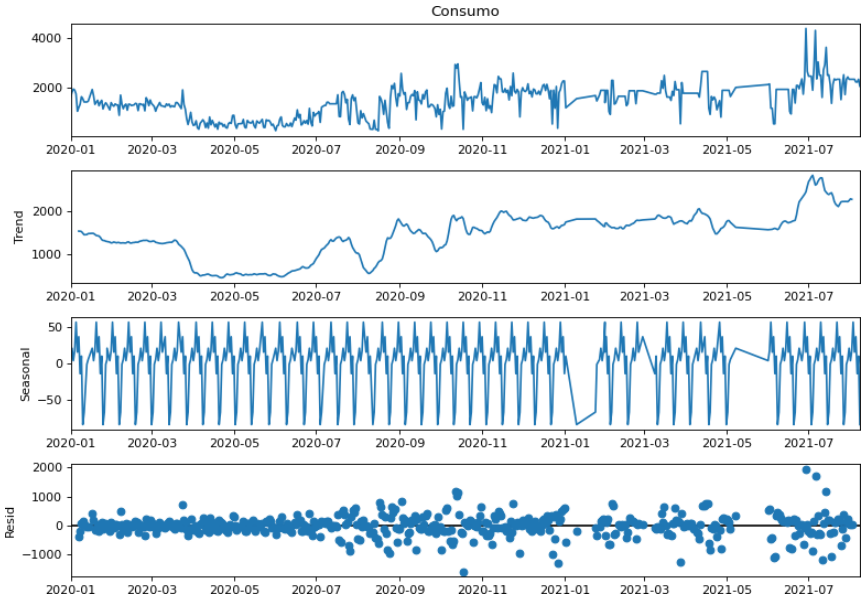
- [Reg,] Bolivia: 28 DE OCTUBRE DE 2015.- Reglamento Técnico y de Seguridad para el Servicio de Transporte de Gas Natural Licuado – GNL, 28 de octubre de 2015.
- [SENER, 2016] SENER (2016). Comienza a operar oficialmente el gasoducto virtual de Bolivia, Noticias SENER.
- [Soediono, 1989] Soediono, B. (1989). Think Python, volumen 53.
- [Villena, 2019]Villena, R. (2019). Programación de Sistemas Programación Orientada a Objetos Programación de Sistemas Programación BASADA en Objetos.
- [vis,] Visual Studio now supports debugging Linux apps; Code editor now open source.
- [YPFB,] YPFB. Gerencia nacional de redes de gas y ductos, Memoria Descriptiva. [YPFB, 2014] YPFB (2014). Sistema virtual GNL beneficiara con gas a 27 poblaciones donde no llegan los gasoductos.
- [YPFB, 2015]YPFB (2015). Gerencia De Redes De Gas Y Ductos. Technical report.
- [Tk, 2021]Tkinter, Wikipedia.org/wiki/Tkinter

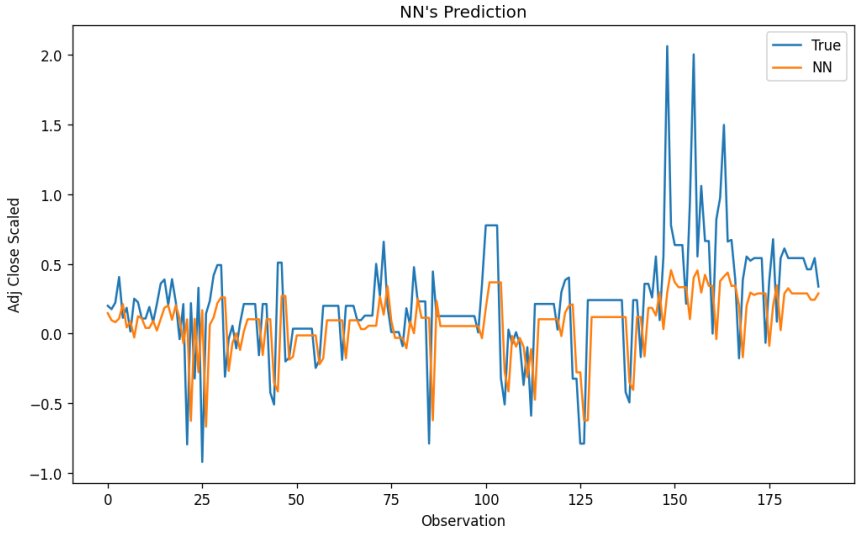
CAPÍTULO 8. ANEXOS

8 Anexos

8.1 Anexo A. Descomposición de las series de tiempo y su predicción para cada Localidad.

Ascensión de Guarayos

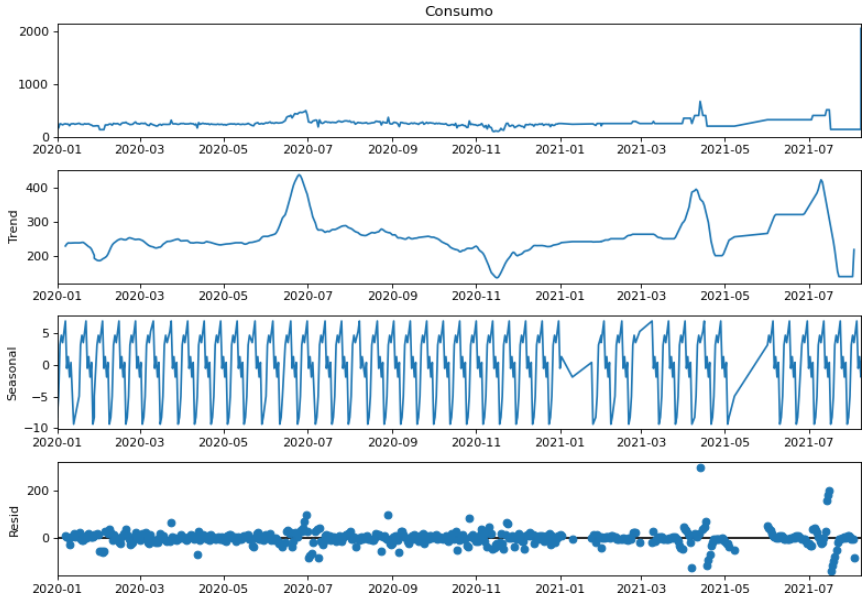


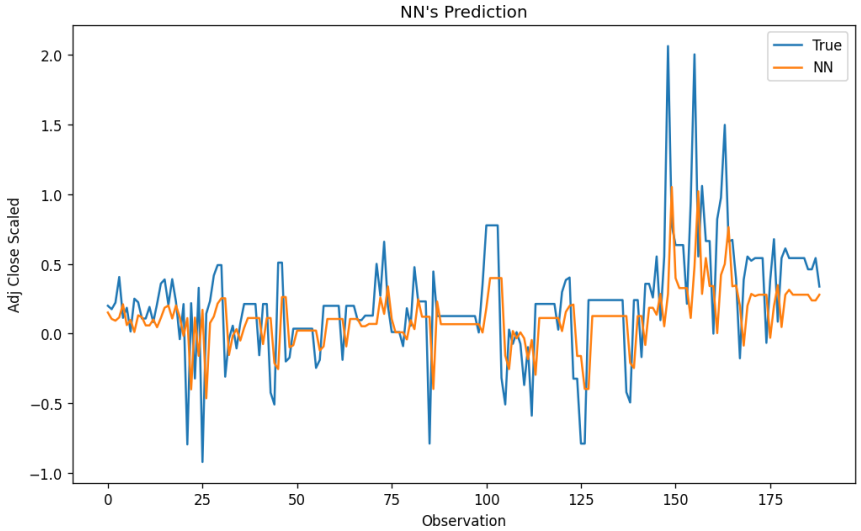


Se encuentran puntos outliers que afectan al entrenamiento de la red neuronal.

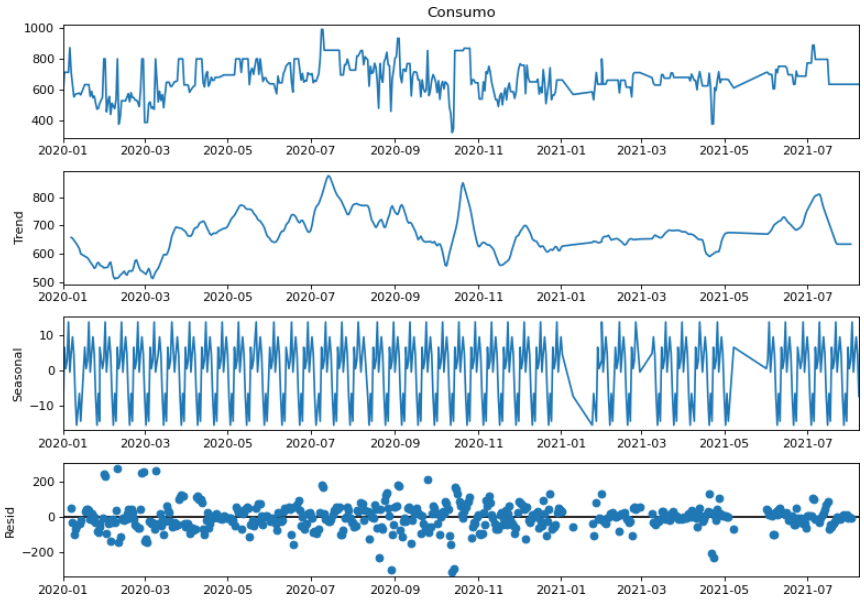
La gráfica naranja representa el entrenamiento de la Red neuronal, logrando generar una predicción, que se asemeja al comportamiento real (gráfica azul) del consumo que se genera en la Estación de Regasificación.

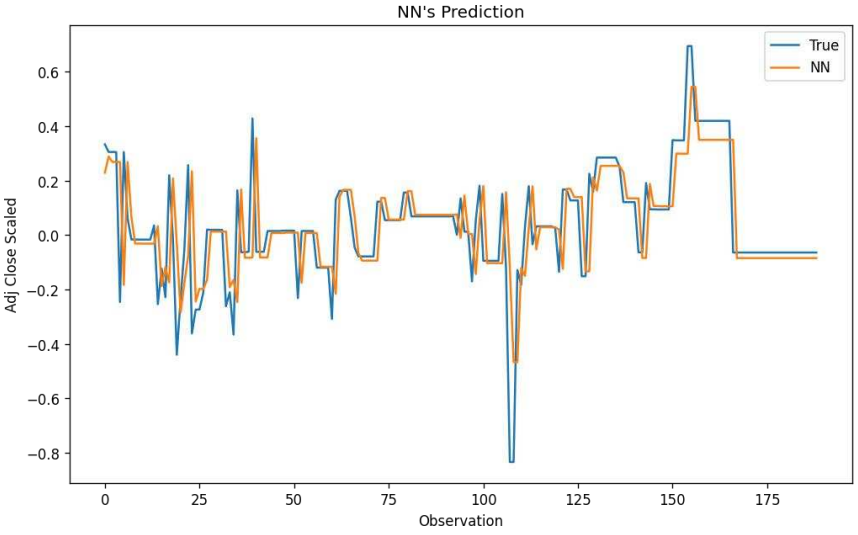
Cabezas



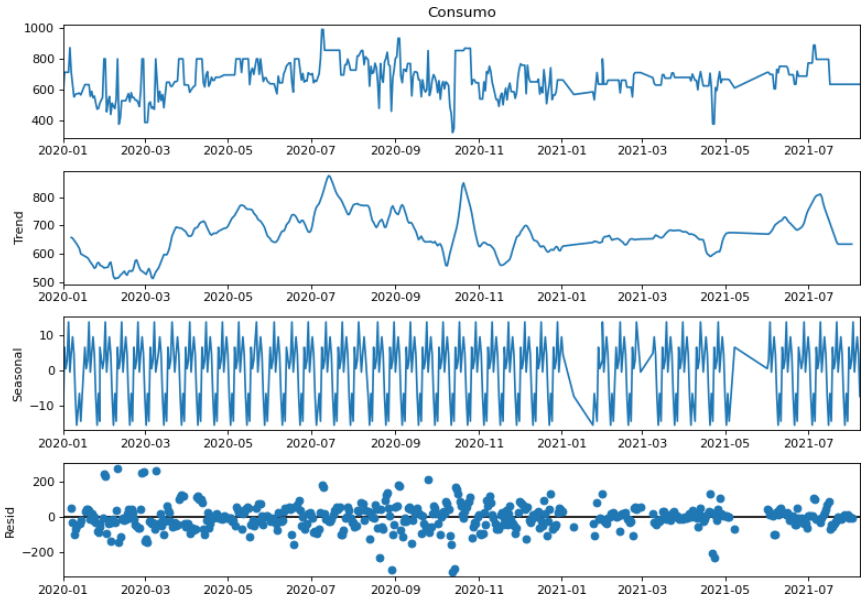


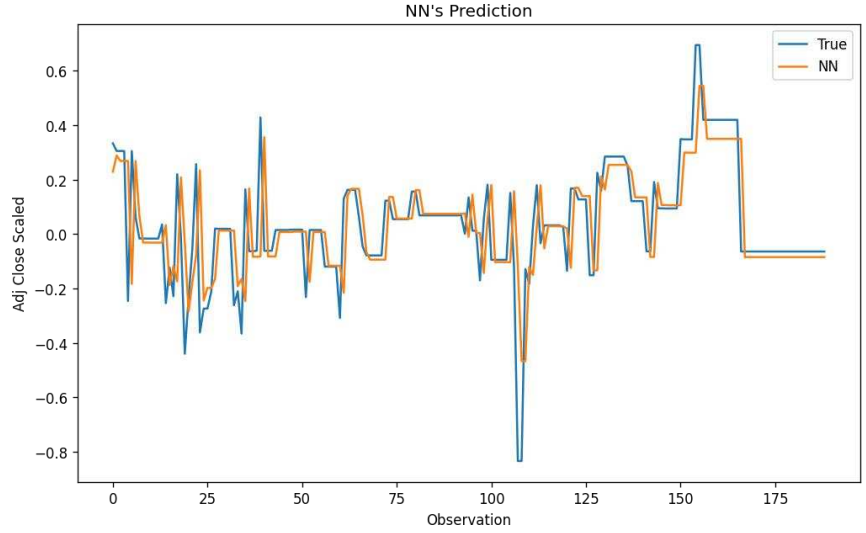
Caranavi


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

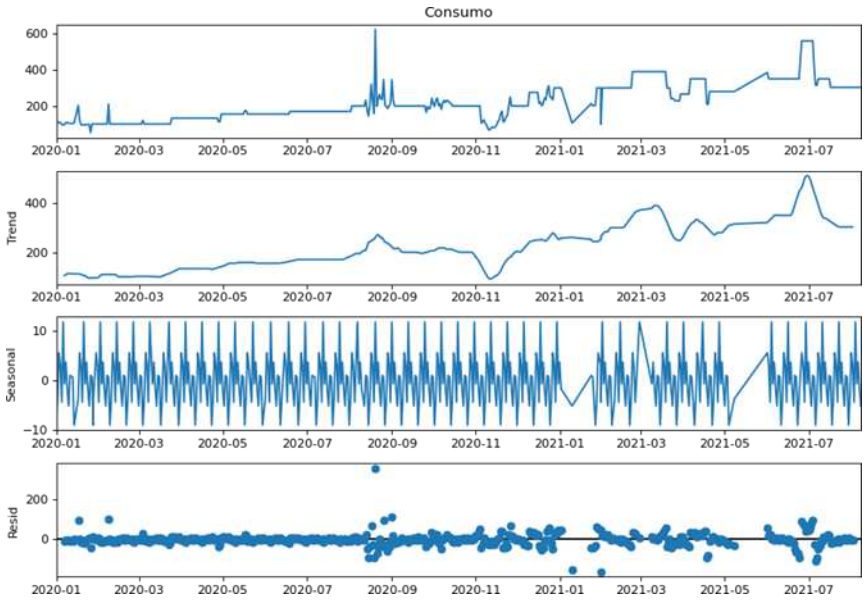

NN's Prediction
True
NN
Adj Close Scaled
Observation

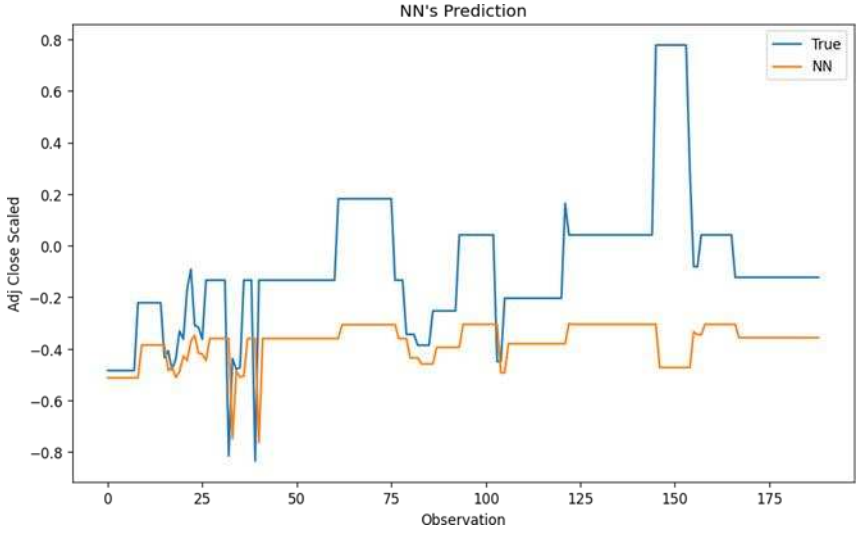
Challapata-Huari



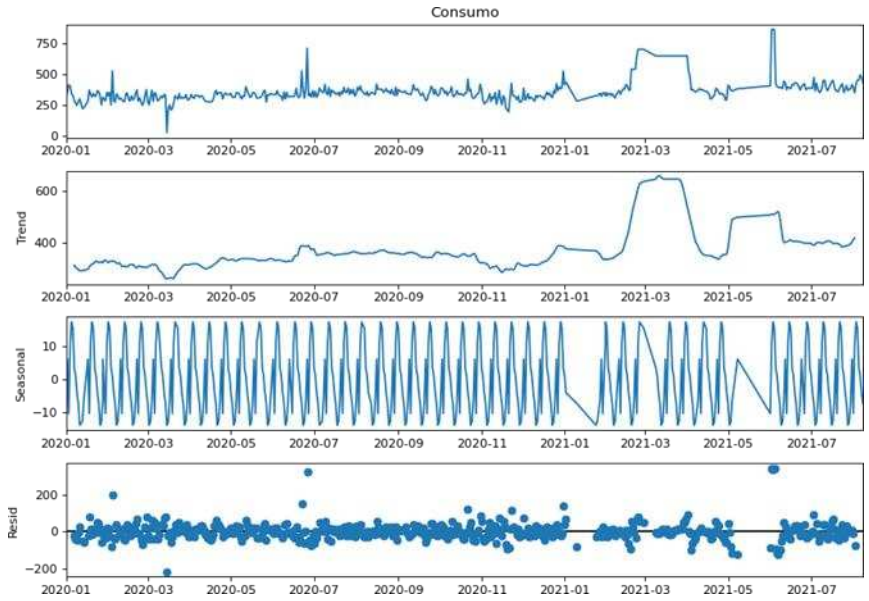


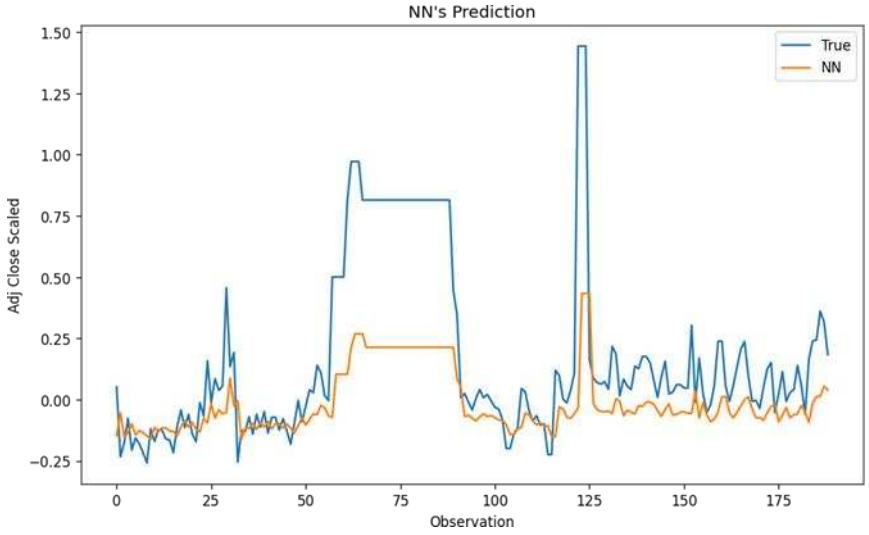
Cobija


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07


NN's Prediction
True
NN
Adj Close Scaled
Observation

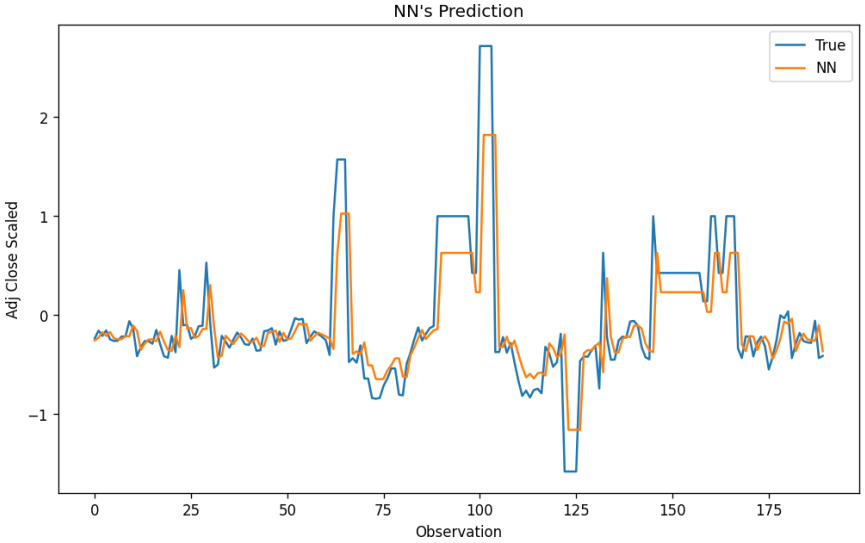
Copacabana


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

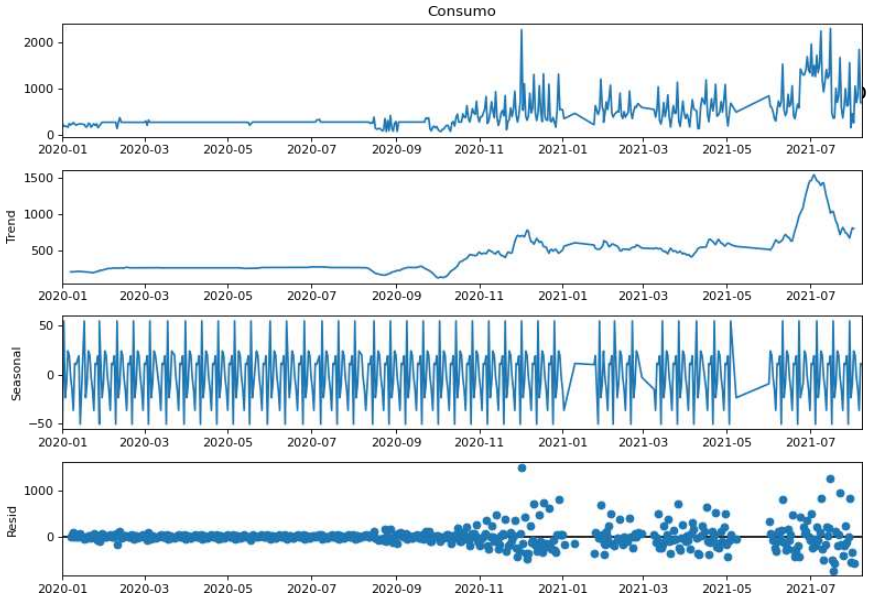

NN's Prediction
True
NN
Adj Close Scaled
Observation

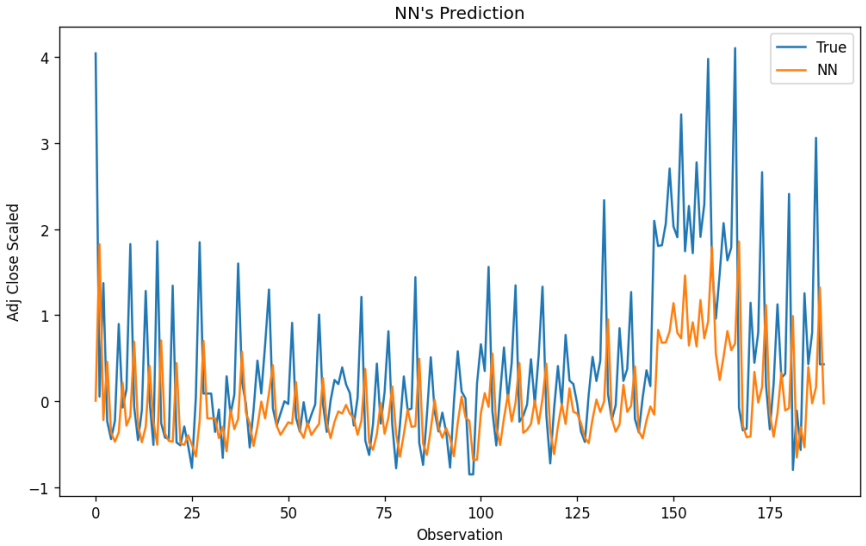
Coroico


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

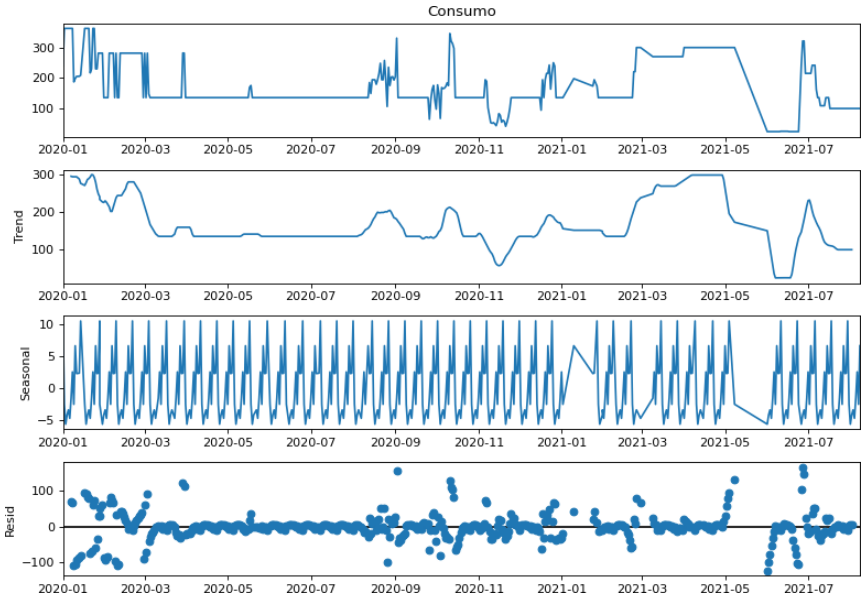

NN's Prediction
True
NN
Adj Close Scaled
Observation

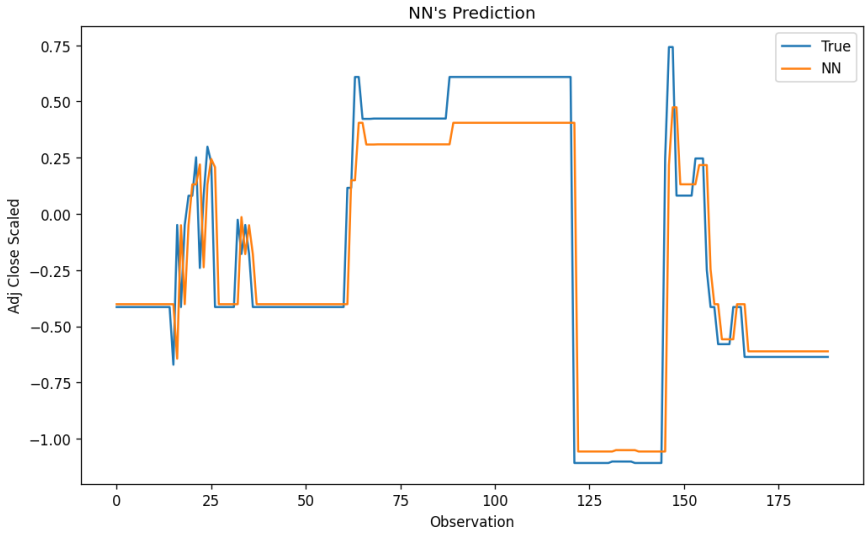
Desaguadero



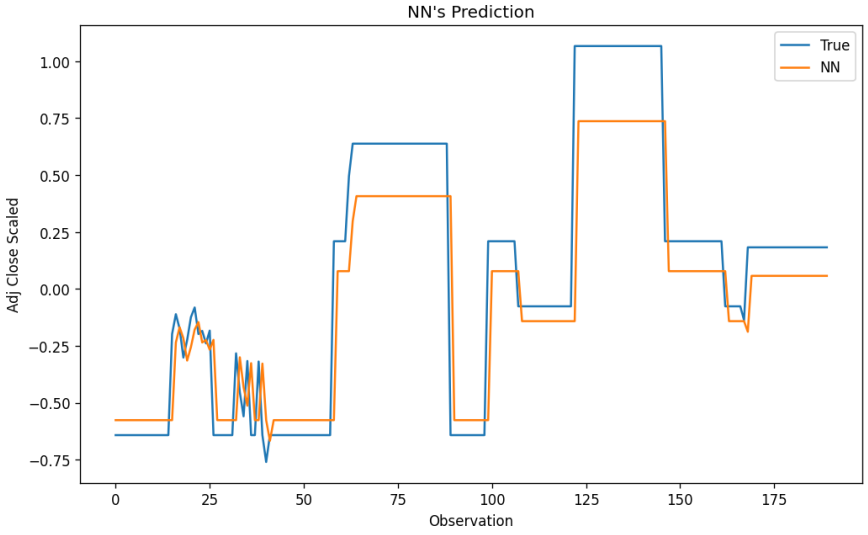


Guanay


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

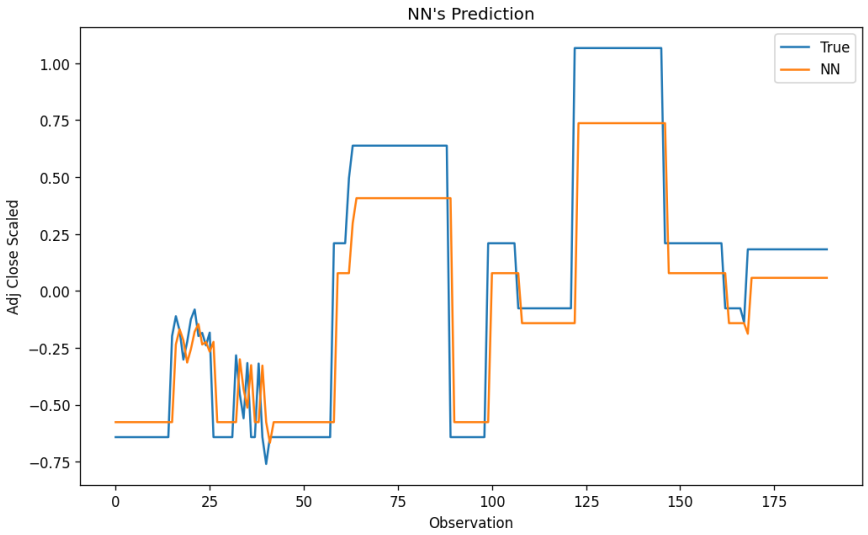

NN's Prediction
True
NN
Adj Close Scaled
Observation

Guayaramerín

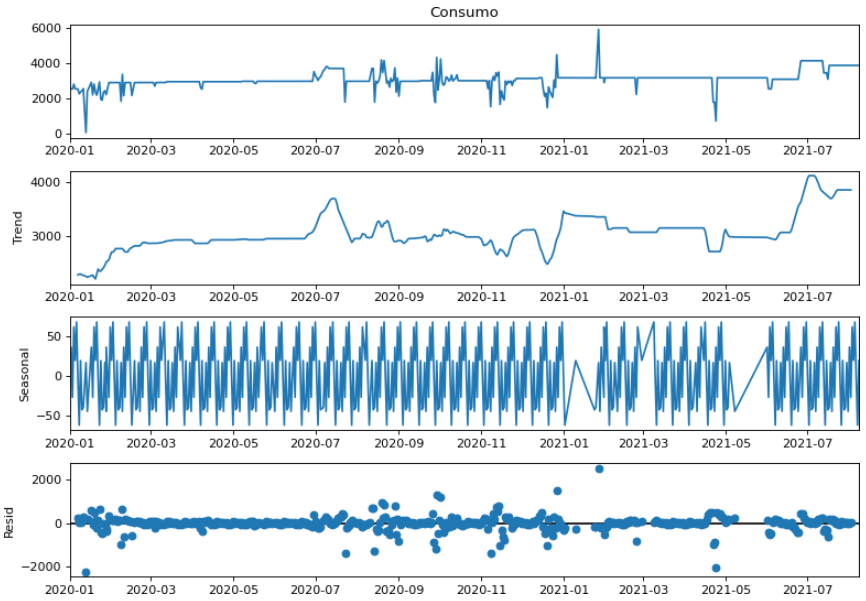


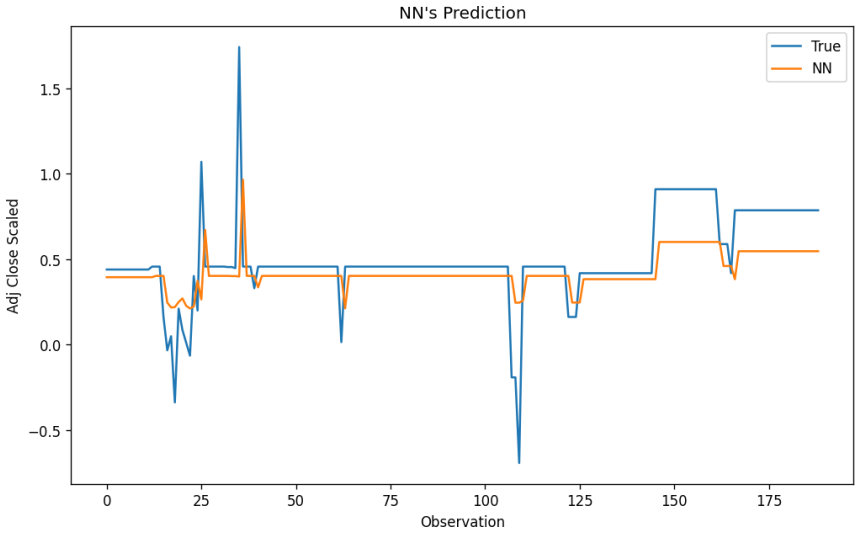
Huanuni


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

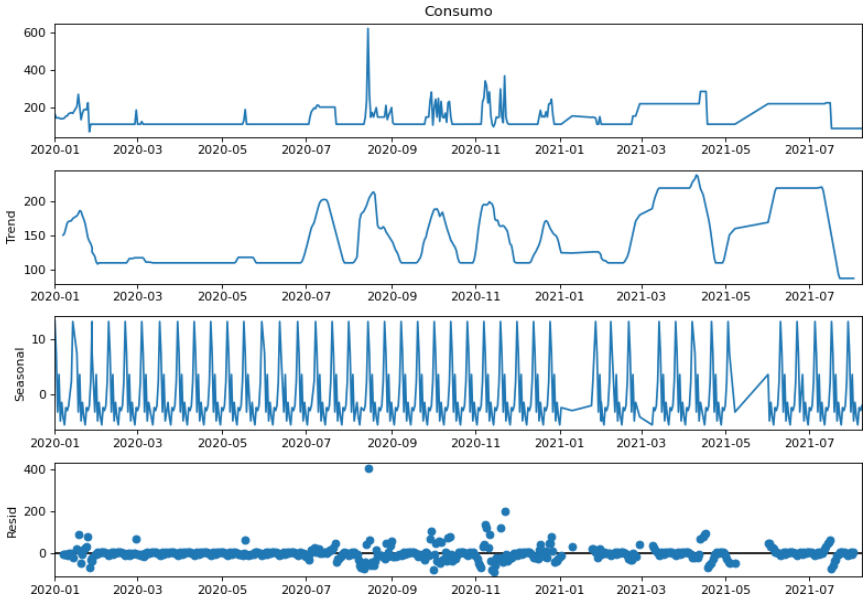

NN's Prediction
True
NN
Adj Close Scaled
Observation

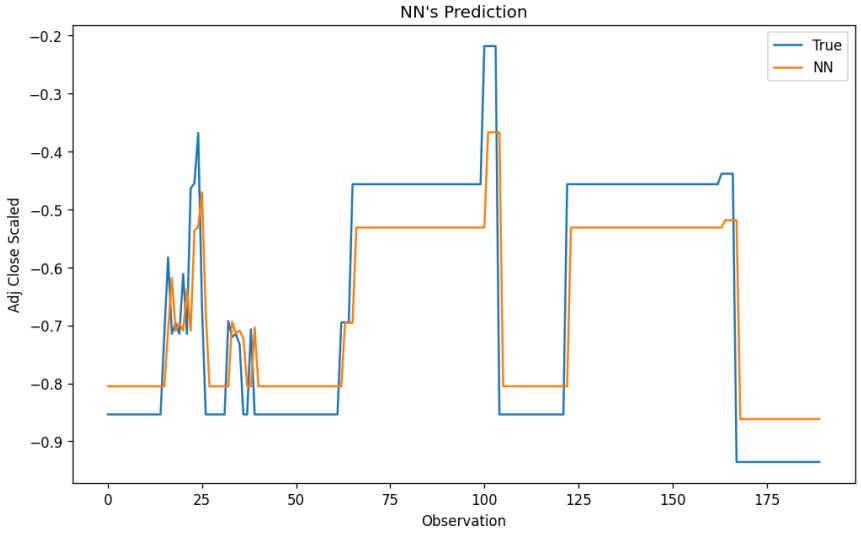
Llallagua


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

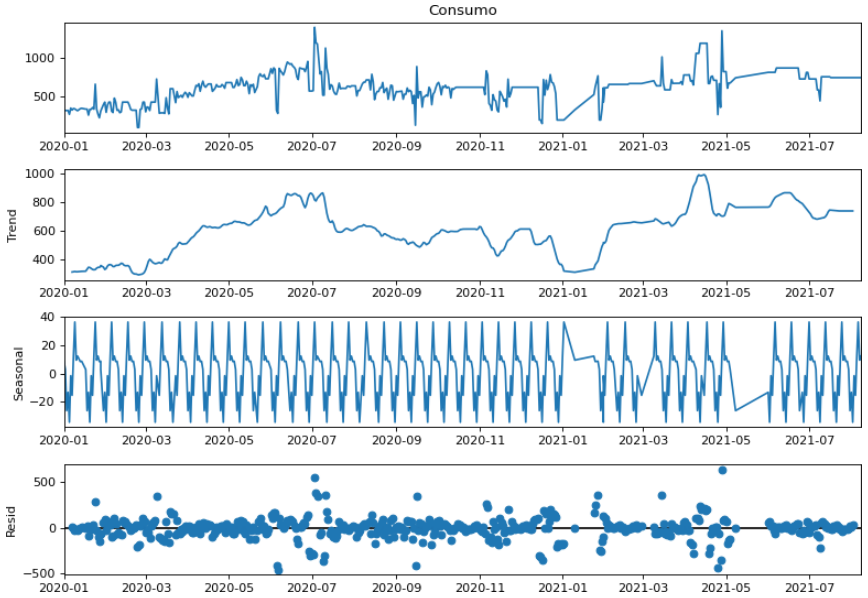

NN's Prediction
True
NN
Adj Close Scaled
Observation

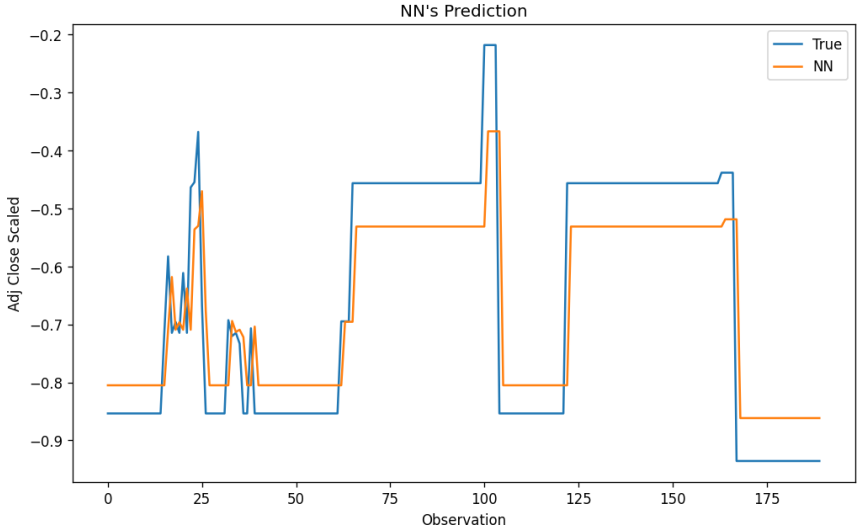
Mora


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

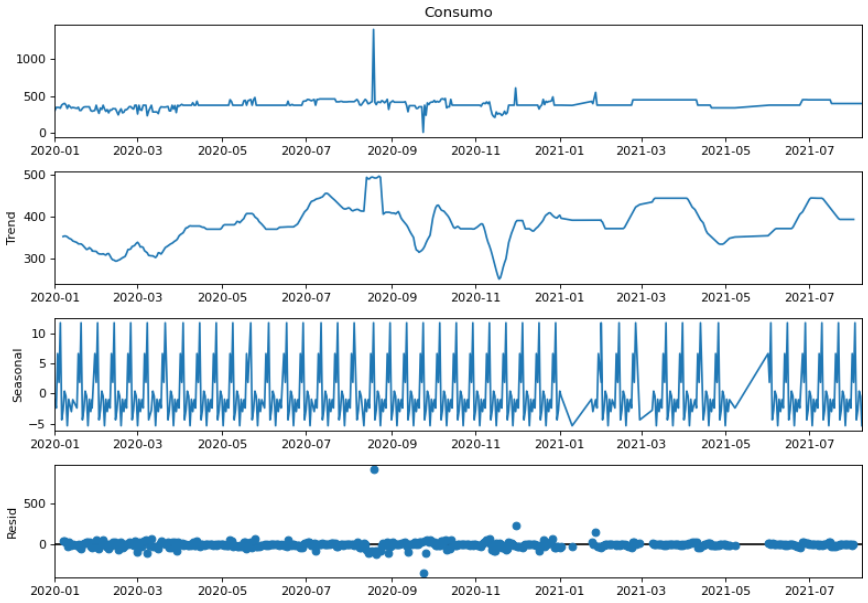

NN's Prediction
True
NN
Adj Close Scaled
Observation

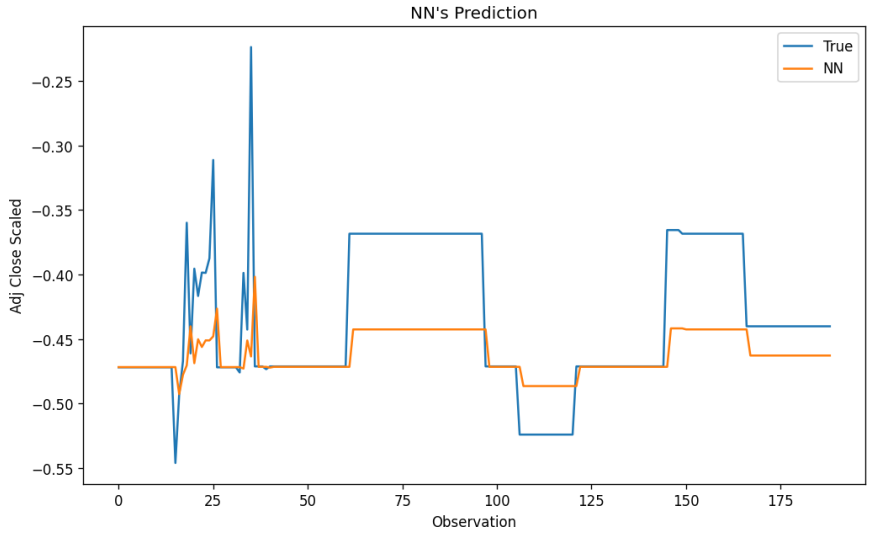
Riberalta


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

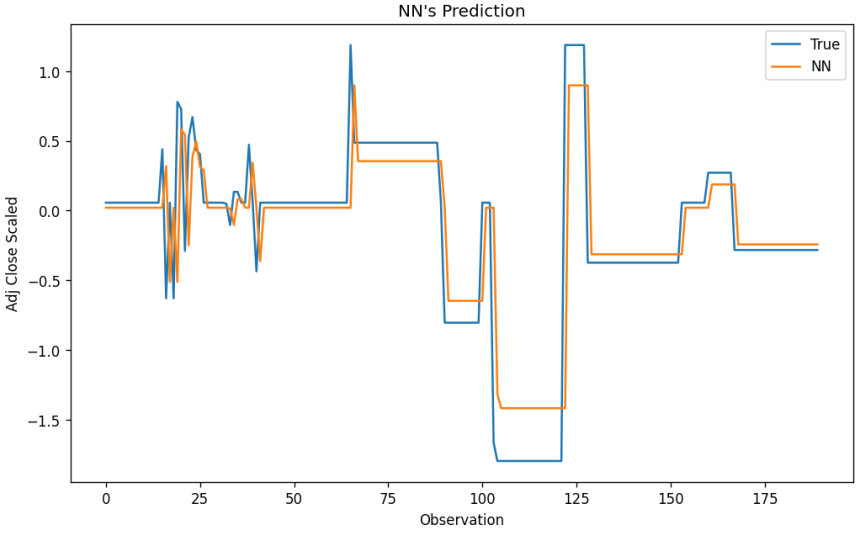

NN's Prediction
True
NN
Adj Close Scaled
Observation

Roboré

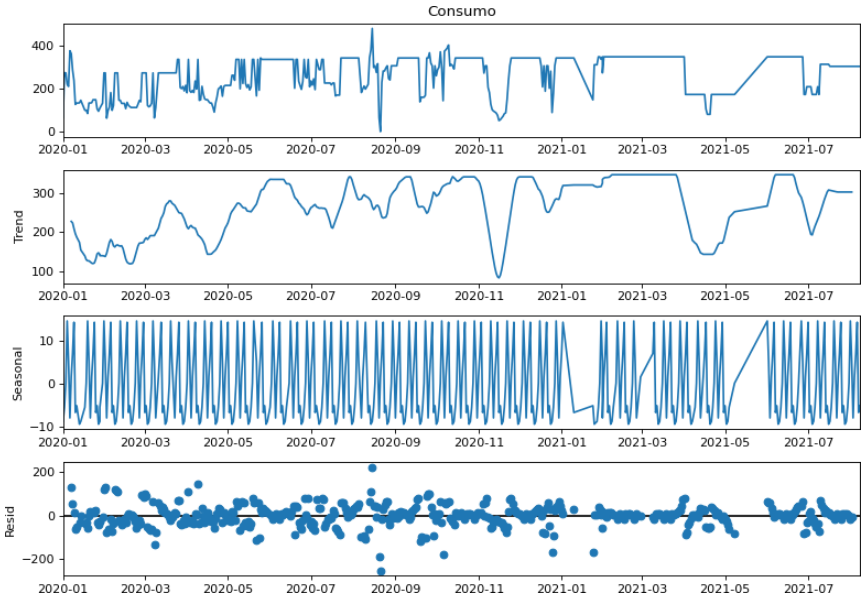

Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

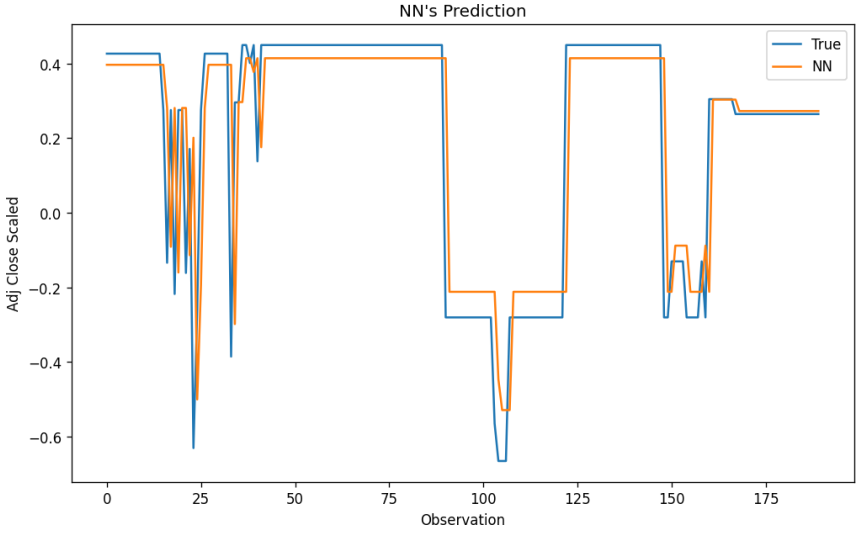

NN's Prediction
True
NN
Adj Close Scaled
Observation

Rurrenabaque

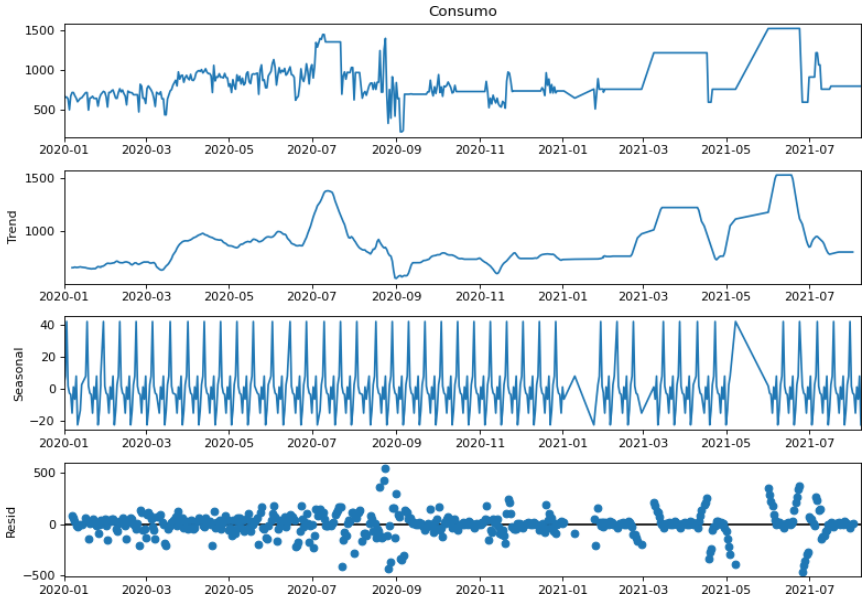


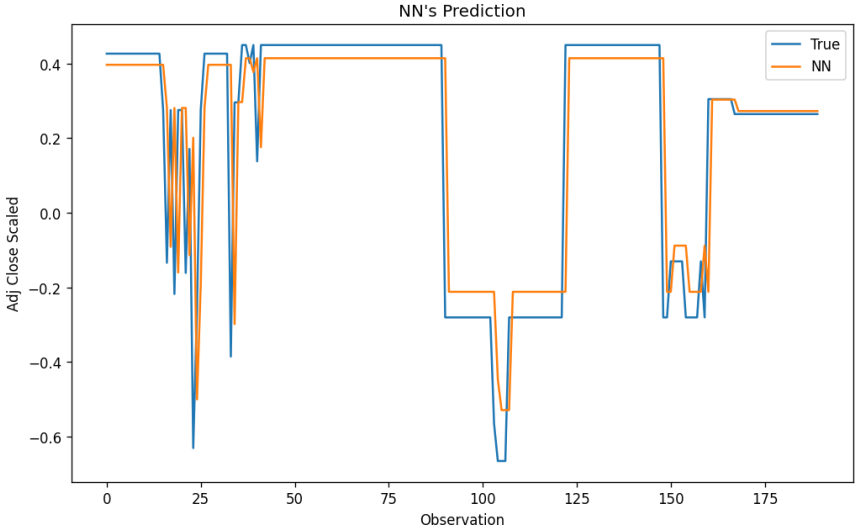
San Borja


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

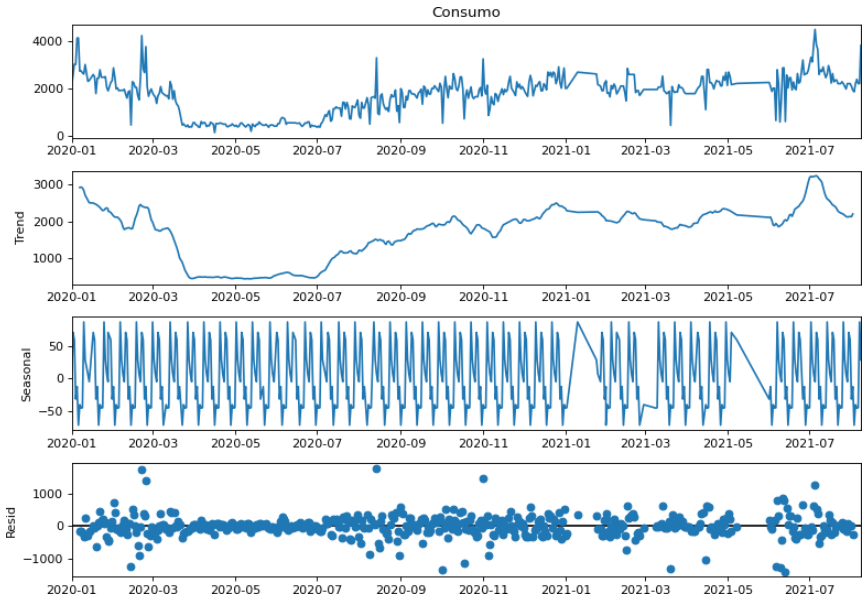

NN's Prediction
True
NN
Adj Close Scaled
Observation

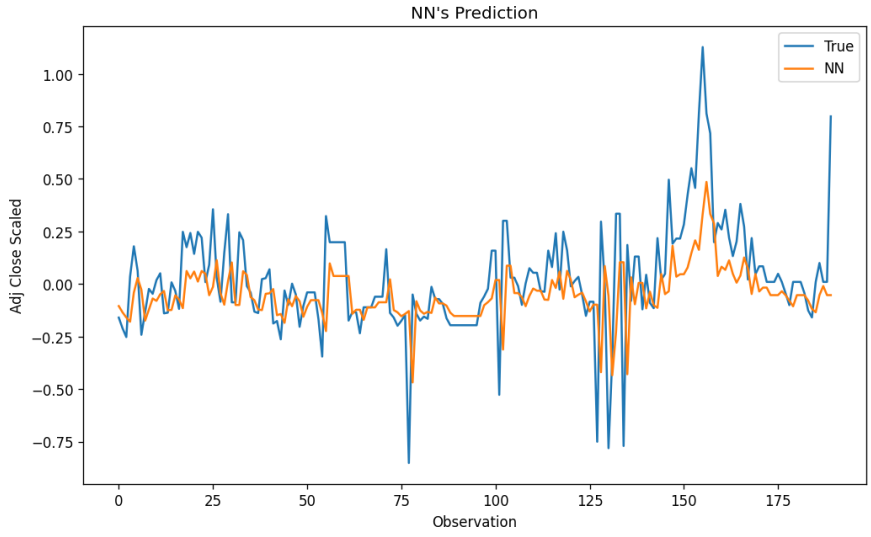
San Ignacio de Velasco



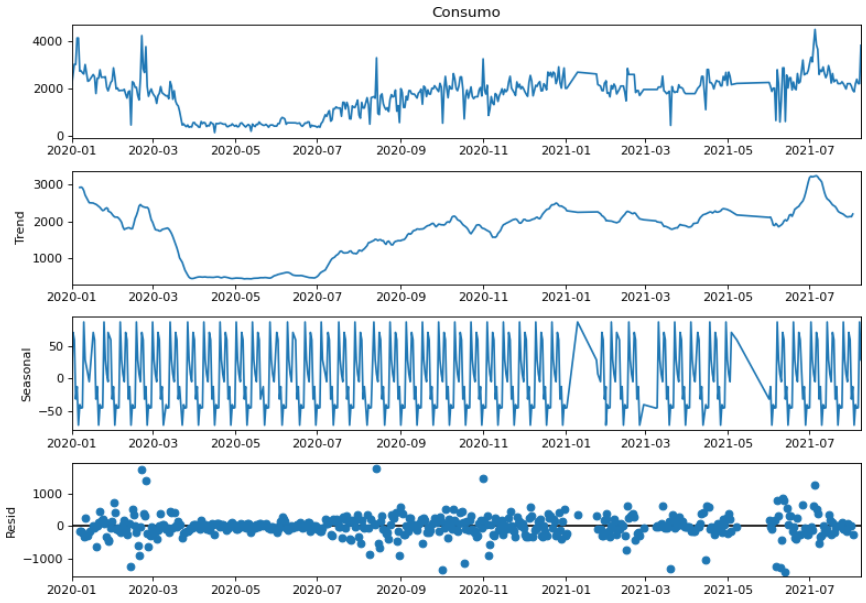


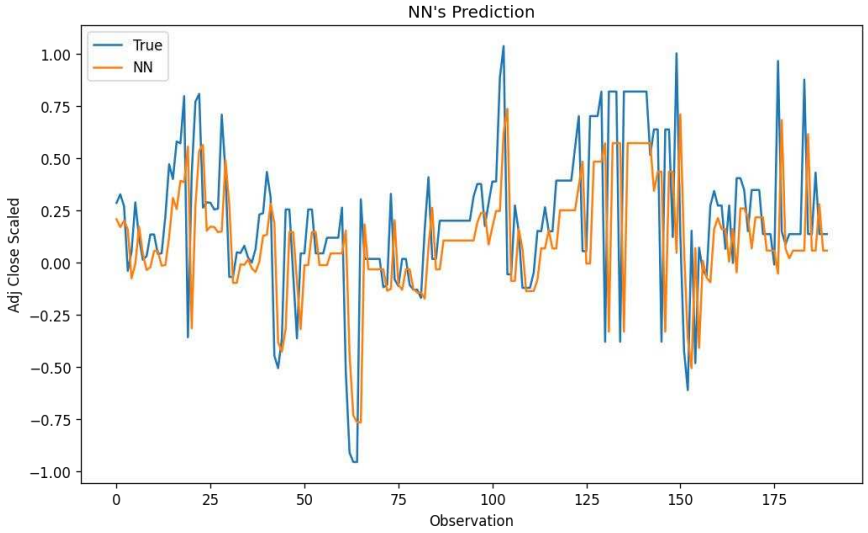
San José de Chuiquitos



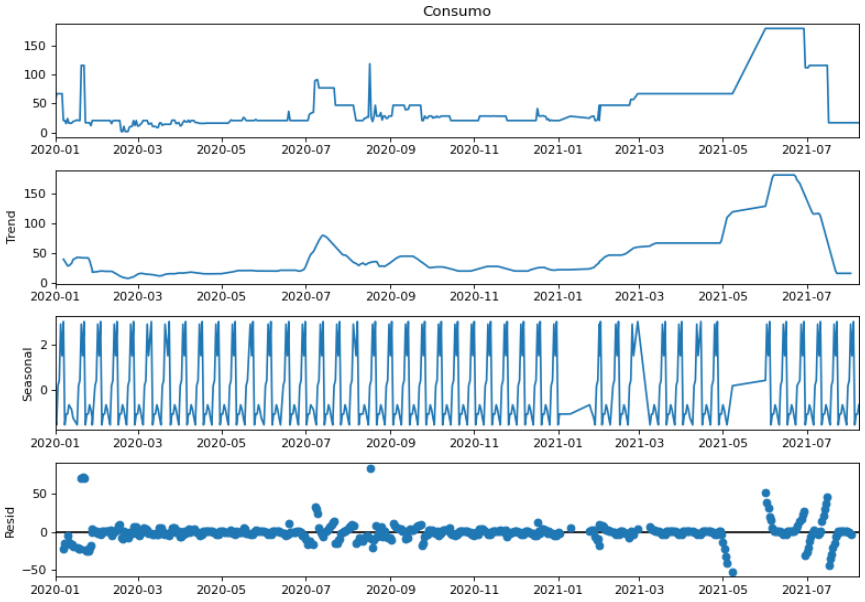


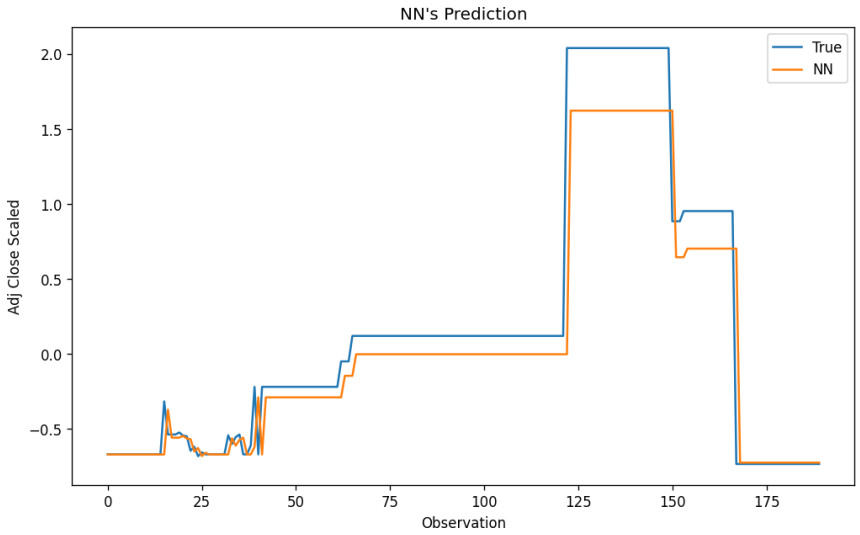
San Julián


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

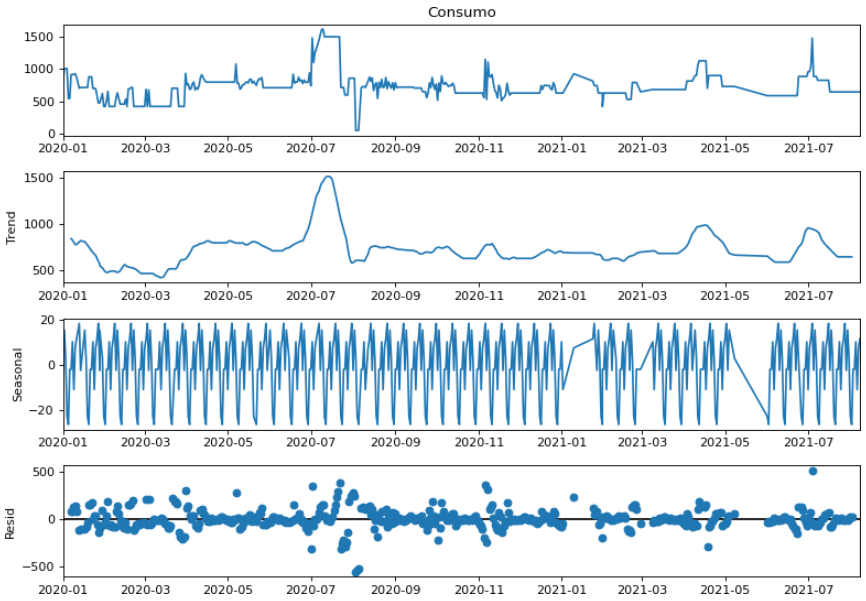

NN's Prediction
True
NN
Adj Close Scaled
Observation

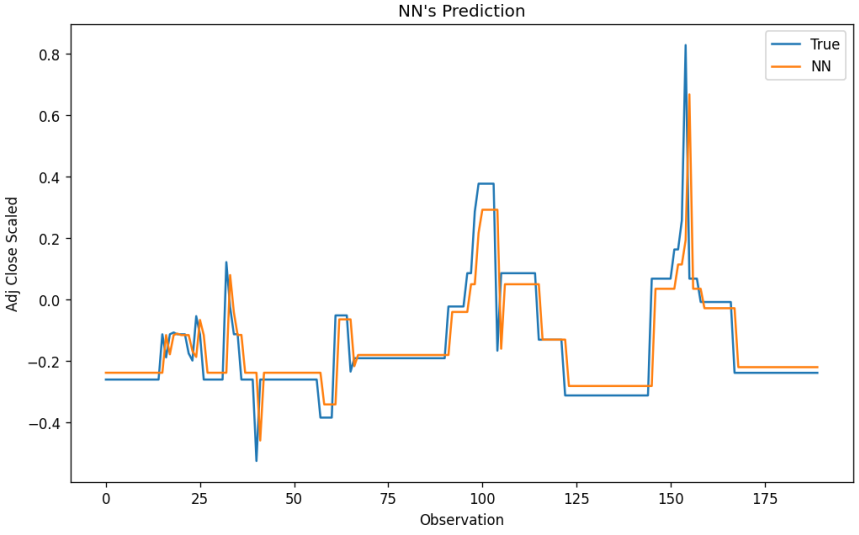
Santa Ana de Yacuma



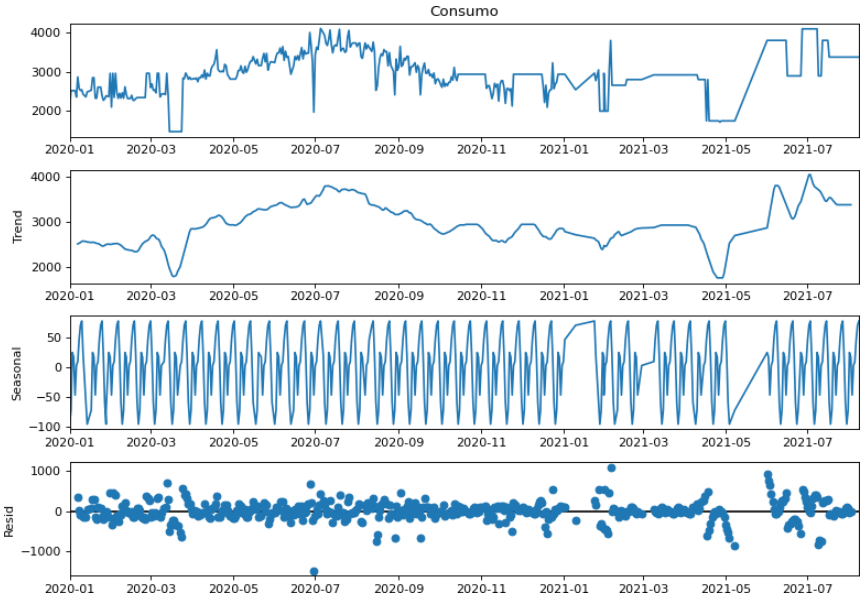


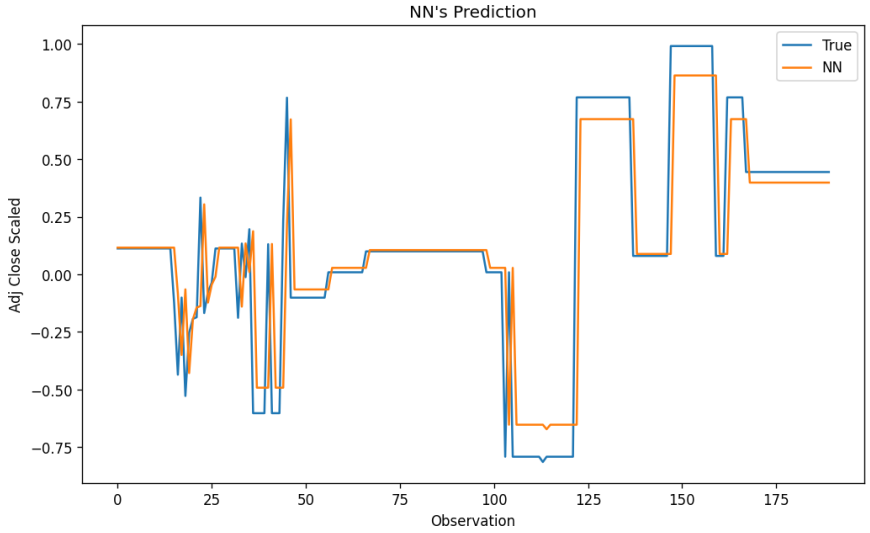
Trinidad


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

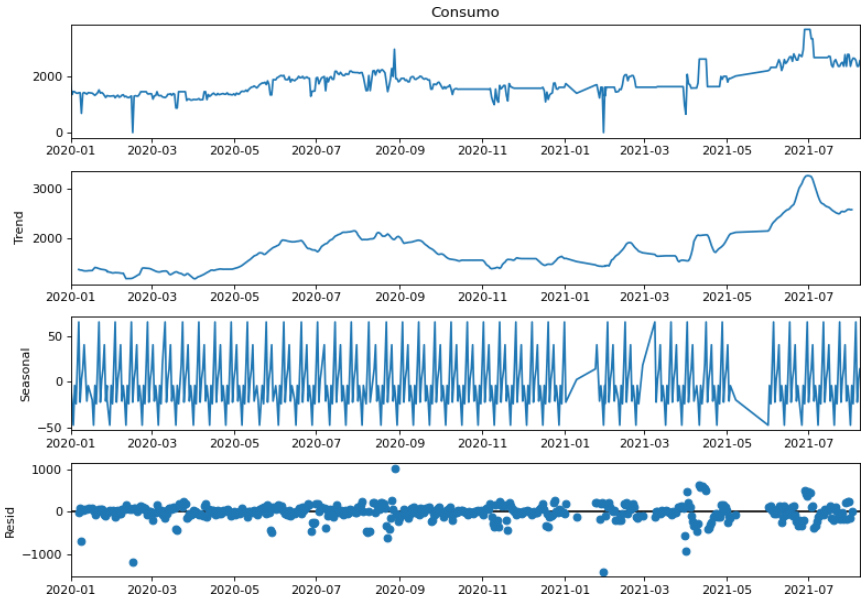

NN's Prediction
True
NN
Adj Close Scaled
Observation

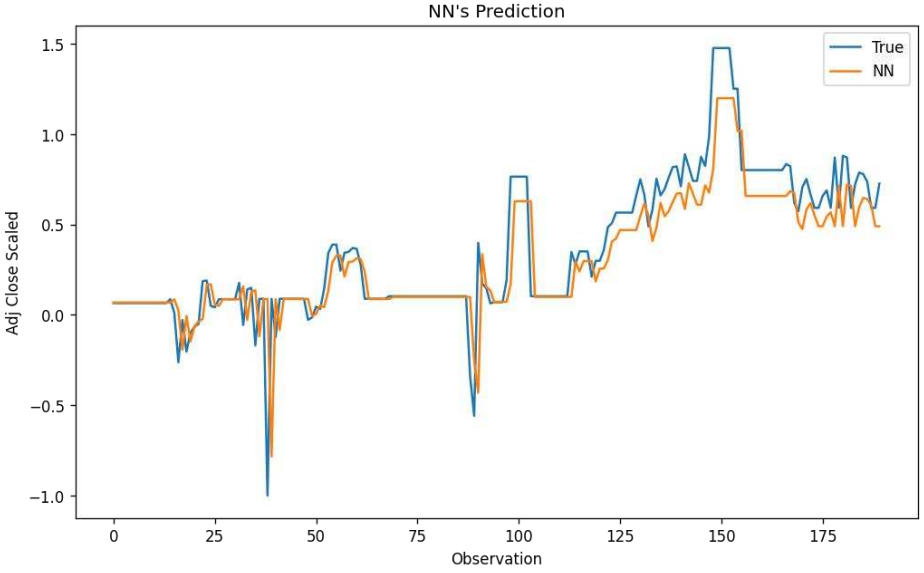
Tupiza


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07

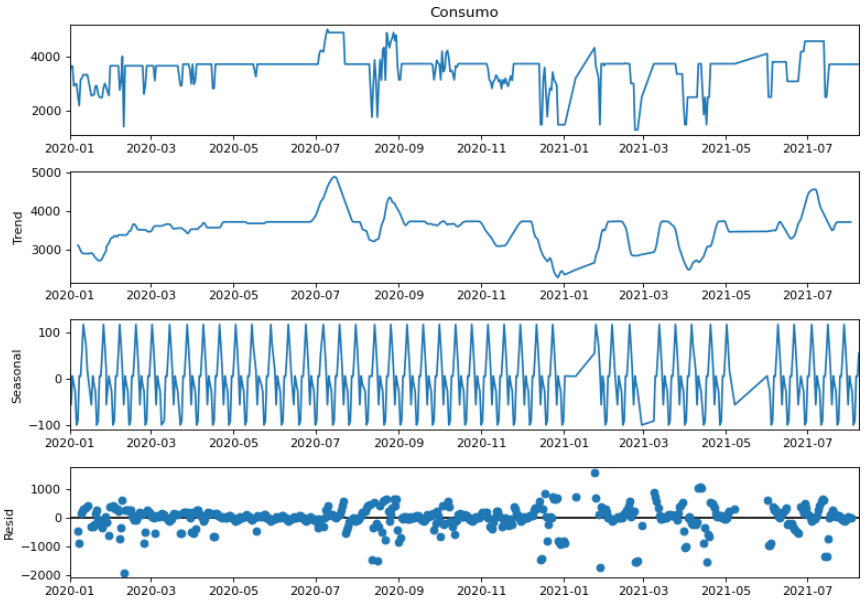

NN's Prediction
True
NN
Adj Close Scaled
Observation

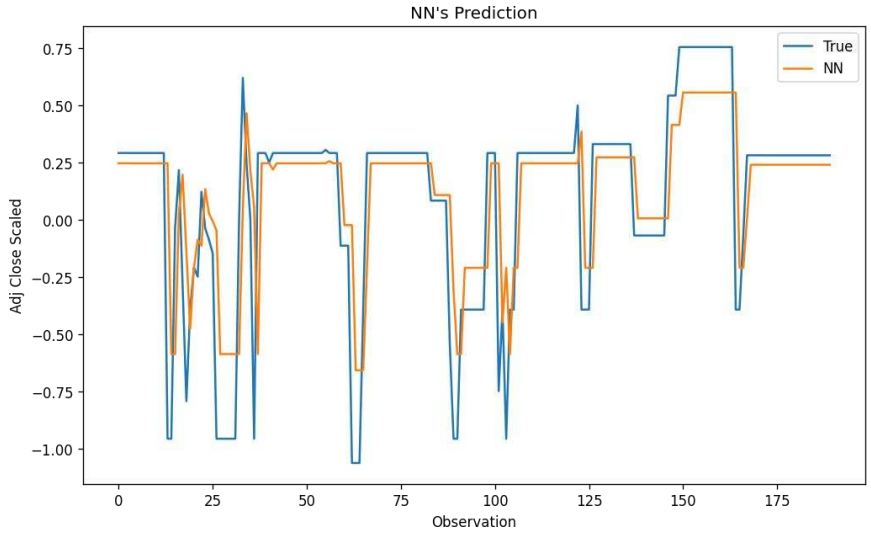
Uyuni


Consumo
Trend
Seasonal
Resid
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07


NN's Prediction
True
NN
Adj Close Scaled
Observation

Villazón





Se pudo observar que en las cinco Estaciones de Regasificación que presentan Gas Natural Vehicular:

- Ascensión de Guarayos

- Desaguadero
- Huanuni
- San José de Chuiquitos
- San Julián

Estas localidades son las que presentan un mayor ruido en sus datos, es decir se puede observar las subidas y bajadas alteradas debido a un sistema de distribución para vender Gas Natural para uso automotor a través de dispensadores.

Tabla de comparación del error cuadrático medio obtenido por el entrenamiento de la red neuronal, de acuerdo a su predicción. Obteniendo la comparación de tres modelos, para obtener el más óptimo y adecuado, dependiendo del comportamiento de cada Estación Satélite de Regasificación:

ESR	Función de activación	Función de optimización	Error cuadrático medio
1. Achacachi	linear	Adagrad	**0.567**
	linear	rmsprop	**0.638**
	relu	rmsprop	**0.606**
2. Ascensión de Guarayos	linear	Adagrad	**0.589**
	linear	rmsprop	**0.361**
	relu	rmsprop	**0.522**
3. Cabezas	linear	Adagrad	**0.740**
	linear	rmsprop	**0.689**
	relu	rmsprop	**0.710**
4. Caranavi	linear	Adagrad	**0.5**
	linear	rmsprop	**0.5**
	relu	rmsprop	**0.497**
5. Challapata	linear	Adagrad	**0.480**
	linear	rmsprop	**-**
	relu	rmsprop	**0.449**
6. Cobija	linear	Adagrad	**0.881**
	linear	rmsprop	**0.449**
	relu	rmsprop	**0.555**
7. Copacabana	linear	Adagrad	**0.265**
	linear	rmsprop	**-**
	relu	rmsprop	**0.782**
8. Coroico	linear	Adagrad	**0.592**
	linear	rmsprop	**0.539**
	relu	rmsprop	**0.481**
9. Desaguadero	linear	Adagrad	**0.226**
	linear	rmsprop	**0.310**
	relu	**rmsprop**	**0.976**

ESR	Función de activación	Función de optimización	Error cuadrático medio
10. Guanay	linear	Adagrad	**0.850**
	linear	rmsprop	**0.836**
	relu	rmsprop	**0.859**
11. Guayaramerín	linear	Adagrad	**0.815**
	linear	rmsprop	**0.835**
	relu	rmsprop	**0.766**
12. Huanuni	linear	Adagrad	**0.492**
	linear	rmsprop	**0.459**
	relu	rmsprop	**0.480**
13. Llallagua	linear	Adagrad	**-**
	linear	rmsprop	**-**
	relu	rmsprop	**0.261**
14. Mora	linear	Adagrad	**0.755**
	linear	rmsprop	**0.745**
	relu	rmsprop	**0.754**
15. Riberalta	linear	Adagrad	**0.539**
	linear	rmsprop	**0.532**
	relu	rmsprop	**0.533**
16. Roboré	linear	Adagrad	**-**
	linear	rmsprop	**-**
	relu	rmsprop	**0.2**
17. Rurrenabaque	linear	Adagrad	**0.723**
	linear	rmsprop	**0.725**
	relu	rmsprop	**0.668**
18. San Borja	linear	Adagrad	**0.695**
	linear	rmsprop	**0.720**
	relu	**rmsprop**	**0.718**

ESR	Función de activación	Función de optimización	Error cuadrático medio
19. San Ignacio de Velasco	linear	Adagrad	**0.772**
	linear	rmsprop	**0.743**
	relu	rmsprop	**0.800**
20. San Joséde Chiquitos	linear	Adagrad	**0.721**
	linear	rmsprop	**0.509**
	relu	rmsprop	**0.701**
21. San Juliá	linear	Adagrad	**0.570**
	linear	rmsprop	**0.355**
	relu	rmsprop	**0.582**
22. Santa Ana	linear	Adagrad	**0.879**
	linear	rmsprop	**0.908**
	relu	rmsprop	**0.654**
23. Trinidad	linear	Adagrad	**0.692**
	linear	rmsprop	**0.646**
	relu	rmsprop	**0.695**
24. Tupiza	linear	Adagrad	**0.732**
	linear	rmsprop	**0.616**
	relu	rmsprop	**0.663**
25. Uyuni	linear	Adagrad	**0.759**
	linear	rmsprop	**0.581**
	relu	rmsprop	**0.727**
26. Villazón	linear	Adagrad	**0.564**
	linear	rmsprop	**0.570**
	relu	**rmsprop**	**0.495**

8.2 Anexo B. Código de la serie de tiempo con Redes Neuronales.

```
import pandas as pd
import numpy as np
from google.colab import files

%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping

from keras.layers import LSTM


plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})
data= pd.read_excel('Uyuni.xlsx')
ind_df = data[['Consumo','Fecha']]
ind_df = ind_df.set_index(['Fecha'], drop=True)
ind_df.head()


ind_df = ind_df.sort_index()
plt.figure(figsize=(10, 6))
ind_df['Consumo'].plot();
df =  ind_df['Consumo']
train = df.loc[:'2020-12-01']
test = df.loc['2020-12-01':]


# Graficamos los datos de entrenamiento y prueba
plt.figure(figsize=(10, 6))
plt.plot(pd.DataFrame(train).index, pd.DataFrame(train),color='tab:blue')
plt.plot(pd.DataFrame(test).index, pd.DataFrame(test),color='tab:red')
```

```
plt.legend(['train', 'test']);


# escalamos
scaler = MinMaxScaler(feature_range=(-1, 1))
train_sc = scaler.fit_transform(train.values.reshape(-1,1))
test_sc = scaler.transform(test.values.reshape(-1,1))



X_train = train_sc[:-1]
y_train = train_sc[1:]

X_test = test_sc[:-1]
y_test = test_sc[1:]


nn_model = Sequential()
nn_model.add(Dense(12, input_dim=1, activation='linear'))
nn_model.add(Dense(1))
nn_model.compile(loss='mean_squared_error', optimizer='Adagrad')
early_stop = EarlyStopping(monitor='loss', patience=1, verbose=1)
history = nn_model.fit(X_train, y_train, epochs=800, batch_size= 1,
verbose=2,callbacks=[early_stop], shuffle=False)


y_pred_test_nn = nn_model.predict(X_test)
y_train_pred_nn = nn_model.predict(X_train)


print("The R2 score on the Train set is:\t{:0.3f}".format(r2_score(y_train, y_train_pred_nn)))
print("The R2 score on the Test set is:\t{:0.3f}".format(r2_score(y_test, y_pred_test_nn)))


plt.figure(figsize=(10, 6))
plt.plot(y_test, label='True')
plt.plot(y_pred_test_nn, label='NN')
```

```
plt.title("NN's Prediction")
plt.xlabel('Observation')
plt.ylabel('Adj Close Scaled')
plt.legend()
plt.show()
```