TC5035.10 Proyecto Integrador

Dra. Grettel Barceló Alonso Dr. Luis Eduardo Falcón Morales

Liga Github: https://github.com/A01793499-DiegoGuerra/Proyecto-Integrador-Equipo18/tree/main

Equipo 18 : "Modelos para la Optimización de Precios en Estaciones de Autoservicio"

- Diego Fernando Guerra Burgos A01793499
- Esteban Sánchez Retamoza A01740631
- Hansel Zapiain Rodríguez A00469031uipo docente. to

Avance 1. Análisis exploratorio de datos

Abril 2024

Objetivos

2.1 Elegir las características más relevantes para reducir la dimensionalidad y aumentar la capacidad de generalización del modelo.

2.2 Abordar y corregir los problemas identificados en los datos.

Instrucciones

Este primer avance consiste en realizar un análisis exploratorio de datos (EDA - Exploratory Data Analysis), es decir, describir los datos utilizando técnicas estadísticas y de visualización (análisis univariante y bi/multivariante) para hacer enfoque en sus aspectos más relevantes, así como aplicar y justificar operaciones de preprocesamiento, relacionadas con el manejo de valores faltantes, atípicos y alta cardinalidad. Es importante que incluyan sus conclusiones del EDA, identificando tendencias o relaciones importantes.

Las siguientes son algunas de las preguntas comunes que podrán abordar a través del EDA:

¿Hay valores faltantes en el conjunto de datos? ¿Se pueden identificar patrones de ausencia? ¿Cuáles son las estadísticas resumidas del conjunto de datos? ¿Hay valores atípicos en el conjunto de datos? ¿Cuál es la cardinalidad de las variables categóricas? ¿Existen distribuciones sesgadas en el conjunto de datos? ¿Necesitamos aplicar alguna transformación no lineal? ¿Se identifican tendencias temporales? (En caso de que el conjunto incluya una dimensión de tiempo). ¿Hay correlación entre las variables dependientes e independientes? ¿Cómo se distribuyen los datos en función de diferentes categorías? ¿Existen patrones o agrupaciones (clusters) en los datos con características similares? ¿Se deberían normalizar las imágenes para visualizarlas mejor? ¿Hay desequilibrio en las clases

de la variable objetivo? Deberán contar con un repositorio en GitHubLinks to an external site., para compartir los resultados con el equipo docente.

__Importar Librerias__

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import statsmodels.api as sm
import plotly.express as px
import random
import functools

from datetime_truncate import truncate
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.signal import periodogram

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

from deap import base, creator, tools, algorithms
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
path = '99_Datasets/'# this needs to be changed to the directory of the excel files
```

__Consolidación de datos (Opcional)__

In [ ]:
```
cost_files_db = [f'{path}Compras 2022 V2.xlsx', f'{path}Compras 2023 V2.xlsx', f'{p
df_cost = pd.DataFrame()

for file in cost_files_db:
    year_db_cost = pd.read_excel(file, skiprows = 4)
    df_cost = pd.concat([df_cost, year_db_cost], ignore_index = True)

df_cost.to_pickle(f'{path}base_compras_combinada_v2.pkl')


sales_files_db = [f'{path}Ventas 2022 V2.xlsx', f'{path}Ventas 2023 V2.xlsx', f'{pa
df_sales = pd.DataFrame()

for file in sales_files_db:
    year_db_sales = pd.read_excel(file, skiprows = 4)
    df_sales = pd.concat([df_sales, year_db_sales], ignore_index = True)

df_sales.to_pickle(f'{path}base_ventas_combinada_v2.pkl')
```

__Carga de Bases__

Para este proyecto tenemos 2 fuentes de información primarias que se obtuvieron directamente de los sistemas ERP del Grupo Golden

Transacciones de Ventas: Considera todos los despachos ejecutados en una gasolinera de grupo golden, incluyendo la distinción de volumenes de compra así como datos adicionales.

Transacciones de Compra: Considera todas las operaciones de compra de combustible para reabastecer a la gasolinera.

In [ ]:
```
df_sales = pd.read_pickle(f'{path}base_ventas_combinada_v2.pkl')
df_sales.head()
```

In [ ]:
```
df_purchases = pd.read_pickle(f'{path}base_compras_combinada_v2.pkl')
df_purchases.head()
```

__Información Básica Fuentes de Información__

Disponemos de una base de ventas de 15 variables. También tenemos dos variables (Fecha y Hora) las cuales deben ser cambiadas a formato DATETIME para poder analizar componentes temporales. En cuanto a la variable Posición, que es una variable categórica y tiene formato de una variable numérica, debe ser cambiada para reflejar el verdadero tipo de dato.

In [ ]:
```
df_sales.info()
```

In [ ]:
```
df_sales.rename(columns = {'Folio':'folio_venta', 'Precio de Venta del litro con im
```

Disponemos de una base de compras de 14 variables. También tenemos dos variables (Fecha y Hora) las cuales deben ser cambiadas a formato DATETIME para poder unir ambas bases. De la misma manera tenemos datos a nivel factura asi como nivel unitarios. Para determinar el margen bruto de cada operación de despacho, nuestro interes es el costo unitario del combustible por lo que los campos IVA F, IEPS F, Sin Imp F y Precio Factura los descartaremos del data frame más adelante. En cuanto a la variable Tanque, que es una variable categórica, debe ser eliminada del análisis dado que solo es un indicador de donde fue depositado el combustible por lo que no agregara valor al análisis más adelante

In [ ]:
```
df_purchases.info()
```

In [ ]:
```
df_purchases.drop(['Precio Factura', 'IVA F', 'IEPS F', 'Sin imp F', 'Tanque'], axi
df_purchases.rename(columns = {'Folio':'folio_compra','Por litro U':'costo_bruto',
```

__Transformación de Bases__

Para consolidar las bases de datos es importante garantizar que todos los campos de fecha existentes sean los correctos. En los siguientes pasos se crearan las estampas de tiempo que nos permitiran hacer este proceso, de la misma manera nos garantizará que podamos hacer análisis de temporalidad hacia adelante. De la misma manera se eliminaran los campos de

Fecha y Hora dado que todas las operaciones hacia adelante se buscaran hacer con la estampa de tiempo

In [ ]:
```
df_sales['sale_date'] = pd.to_datetime(df_sales['Fecha'], errors = 'coerce', infer_
df_sales['sale_localtime'] = pd.to_timedelta(df_sales['Hora'].astype(str))
df_sales['sale_timeStamp'] = df_sales['sale_date'] + df_sales['sale_localtime']

df_sales.drop(['Fecha', 'Hora'], axis = 1, inplace = True)
df_sales.head()
```

In [ ]:
```
df_purchases['purchase_date'] = pd.to_datetime(df_purchases['Fecha'], errors = 'coe
df_purchases['purchase_localtime'] = pd.to_timedelta(df_purchases['Hora'].astype(st
df_purchases['purchase_timeStamp'] = df_purchases['purchase_date'] + df_purchases['

df_purchases.drop(['Fecha', 'Hora'], axis = 1, inplace = True)
df_purchases.head()
```

__Información datos faltantes o NA__

En este caso, los datos de venta principales como la fecha de transacción, cantidad de combustible, tipo de combustible, precio, entre otros, no registran valores faltantes. La gran mayoría de datos faltantes se observan en información del cliente como tipo de vehículo, el nombre, placas, etc. En este caso, los patrones de ausencia se deben a la falta de recolección o estandarización de este tipo de datos por parte de la gasolinera.

In [ ]:
```
#Calculando data faltante en la base de datos por columna
missing_percentage_per_column_sales = (df_sales.isnull().mean() * 100).round(2)
print("Presencia de datos faltantes por dimensión (en porcentaje):")
print(missing_percentage_per_column_sales)
```

En este caso, los datos de compras principales como la fecha de transacción, cantidad de combustible, tipo de combustible, precio, entre otros, no registran valores faltantes.

In [ ]:
```
#Calculando data faltante en la base de datos por columna
missing_percentage_per_column_purchases = (df_purchases.isnull().mean() * 100).roun
print("Presencia de datos faltantes por dimensión (en porcentaje):")
print(missing_percentage_per_column_purchases)
```

__Unificación de Bases__

Usando las estampas de tiempo obtendremos el costo de reposición de cada despacho a cliente. Esto nos permitirá obtener un cálculo de margen bruto que al finalizar estará atado con nuestro objetivo final de optimización de precios

In [ ]:
```
df_sales.sort_values(['producto', 'sale_date'], inplace = True)
df_purchases.sort_values(['producto', 'purchase_date'], inplace = True)

df_master = pd.DataFrame()

for product_type in df_sales['producto'].unique():
```

```
    sales_temp = df_sales[df_sales['producto'] == product_type]
    purchases_temp = df_purchases[df_purchases['producto'] == product_type]

    merged_temp = pd.merge_asof(sales_temp, purchases_temp, left_on = 'sale_date',
                                by = 'producto', direction = 'backward')

    # Append the result to the main DataFrame
    df_master = pd.concat([df_master, merged_temp], ignore_index = True)

df_master.head(5000)
```

__Información Básica Data Frame__

In [ ]:
```
df_master.info()
```

In [ ]:
```
df_sales.describe()
```

__Estadísticas de las variables__

In [ ]:
```
unique_values_per_column = df_master.nunique()
unique_values_per_column
```

In [ ]:
```
print('Identificación de manguera dispensadora:')
print(df_master['Posición'].value_counts())
```

In [ ]:
```
print('Tipo de Combustible vendido:')
print(df_master['producto'].value_counts())
```

__Presencia de valores atípicos__

Nuestras variables numéricas son Volumen, Precio Neto, Precio Bruto, Costo Neto, Costo Bruto, Venta Bruta. En estos gráficos de Caja y Densidad podemos observar la presencia de valores atípicos en Cantidad e Importe. Estos datos atípicos reflejan valores demasiado altos, lo cual resulta en que exista una distribución sesgada a la izquierda. Se debería eliminar estos registros para observar la nueva distribución.

En el caso de Precio, podemos observar que existe una distribución ligeramente sesgada a la izquierda, con la presencia de pocos valores atípicos, pero que se encuentran en valores lógicos, a comparación de los valores atípicos en las otras variables.

In [ ]:
```
columnas_numericas = ['volumen_despachado', 'precio_neto', 'precio_bruto', 'costo_n

for column in columnas_numericas:
    plt.figure()
    df_master[column].plot(kind = 'box')
    plt.title(f'Gráfico de caja de {column}')
    plt.show()
```

In [ ]:
```
plt.figure(figsize = (12, 8))
```

```
for i, columna in enumerate(columnas_numericas, 1):
    plt.subplot(2, 3, i)
    df_master[columna].plot(kind = 'density', color = 'skyblue')
    plt.xlabel('Valor')
    plt.ylabel('Densidad')
    plt.title(f'Gráfico de Densidad de {columna}')

plt.tight_layout()
plt.show()
```

__Tratamiento de valores atípicos__

En este caso podemos observar que existen valores atípicos que tal vez son producto de mal registro por parte de la Gasolinera, lo que resulta en valores demasiado altos. Por ejemplo, el valor máximo de *Cantidad* se ubica en 6 millones de galones por una sola venta (un vehículo promedio se llena con 12 galones). Por lo cual para filtrar estos datos atípicos, vamos a eliminar, solamente para este análisis exploratorio, el 10% de los datos más altos de la serie de *Cantidad*.

In [ ]:
```
df_master_filter = df_master[df_master['volumen_despachado'] <= df_master['volumen_

plt.figure(figsize = (12, 8))

for i, columna in enumerate(columnas_numericas, 1):
    plt.subplot(2, 3, i)
    df_master_filter[columna].plot(kind = 'density', color = 'skyblue')
    plt.xlabel('Valor')
    plt.ylabel('Densidad')
    plt.title(f'Gráfico de densidad de {columna}')

plt.tight_layout()
plt.show()
```

Una vez que filtramos los datos atípicos, observamos que las variables *Cantidad* e *Importe*, tienen una distribución similar, con un ligero sesgo a la izquierda. De la misma manera aprovecharemos para eliminar todos aquellos valores que no tengan datos faltantes. De la misma manera normalizamos a nivel hora dado que esto nos permitirá hacer un mejor análisis hacia adelante

In [ ]:
```
df_master_filter.reset_index(drop = True, inplace = True)
df_master_filter.drop('Cod.Externo', axis = 1, inplace = True)
df_master_filter.dropna(axis = 0 , inplace = True)
```

In [ ]:
```
df_master_filter['sale_timeStamp'] = df_master_filter['sale_timeStamp'].dt.floor('h
```

In [ ]:
```
pmap = {' DIESEL':'diesel', ' MAGNA':'magna', ' PREMIUM':'premium'}
df_master_filter['producto'] = df_master_filter['producto'].map(pmap)
```

In [ ]:
```
df_master_filter.info()
```

In [ ]:
```
df_master_filter.to_pickle(f'{path}base_master.pkl')
```

Conclusiones Avance 1

Para este análisis de la base de datos de nuestro Proyecto de Modelo para Optimización de Precios en Estaciones de Autoservicio tomamos como referencia la base diaria de ventas para el periodo 2022-2024 (abril). En primer lugar, consolidamos las bases de datos asegurándonos que cumplan con el tipo de dato correspondiente a la naturaleza de cada variable para poder analizar el contenido de la base de datos.

Las variables completas hacen referencia a información de la transacción de Ventas de 1 gasolinera, en la cual encontramos 3 variables numéricas, 2 variables categóricas, 2 variables que hacen referencia a la fecha de la transacción y 1 variable de identificación de la transacción. Encontramos valores atípicos que pueden deberse a una incorrecta digitación de la información (por ejemplo, una venta fue de 6 millones de galones) por lo cual decidimos filtrar el 10% de los valores más altos de la variable Cantidad para poder realizar el análisis exploratorio.

Avance 2.Ingeniería de características

Mayo 2024

Objetivos

2.3 Crear nuevas características para mejorar el rendimiento de los modelos.

2.4 Mitigar el riesgo de características sesgadas y acelerar la convergencia de algunos algoritmos.

Instrucciones

En esta fase, conocida como ingeniería de características (FE - Feature Engineering):

Se aplicarán operaciones comunes para convertir los datos crudos del mundo real, en un conjunto de variables útiles para el aprendizaje automático. El procesamiento puede incluir: Generación de nuevas características Discretización o binning Codificación (ordinal, one hot,...) Escalamiento (normalización, estandarización, min – max,...) Transformación (logarítmica, exponencial, raíz cuadrada, Box – Cox, Yeo – Johnson,...)

- Todas las decisiones y técnicas empleadas deben ser justificadas.

Además, se utilizarán métodos de filtrado para la selección de características y técnicas de extracción de características, permitiendo reducir los requerimientos de almacenamiento, la complejidad del modelo y el tiempo de entrenamiento. Los ejemplos siguientes son ilustrativos, pero no exhaustivos, de lo que se podría aplicar: Umbral de varianza Correlación Chi-cuadrado ANOVA Análisis de componentes principales (PCA) Análisis factorial (FA)

- Es necesario fundamentar los métodos ejecutados.

Incluir conclusiones de la fase de "Preparación de los datos" en el contexto de la metodología CRISP-ML.

__Análisis básico__

In [ ]:
```
df_master_filter = pd.read_pickle(f'{path}base_master.pkl')
```

In [ ]:
```
def date_info(df_interest, timeStamp):
    df_interest['hour'] = df_interest[timeStamp].dt.hour
    df_interest['month'] = df_interest[timeStamp].dt.month
    df_interest['year'] = df_interest[timeStamp].dt.year
    df_interest['dow_n'] = df_interest[timeStamp].dt.dayofweek

    dmap = {0:'Mon', 1:'Tue', 2:'Wed', 3:'Thu', 4:'Fri', 5:'Sat', 6:'Sun'}

    df_interest['dow'] = df_interest['dow_n'].map(dmap)
    df_interest['clean_date'] = df_interest[timeStamp].dt.floor('D')
```

In [ ]:
```
df_alt = df_master_filter.copy()
date_info(df_alt,'sale_timeStamp')
```

In [ ]:
```
sns.countplot(x = 'dow', hue = 'producto', data = df_alt[df_alt['year'] == 2023], p
plt.show()
```

In [ ]:
```
sns.countplot(x = 'month', hue = 'producto', data = df_alt[df_alt['year'] == 2023],
plt.show()
```

__Transformación de datos para procesamiento por fecha__

Dado que estaremos trabajando con análisis de regresión para estimar demandas es necesario construir todas las estampas de manera que nos permita identificar patrones de demanda específicos (i.e. temporalidades). Es por esto que tenemos que generar un proceso de sumarización, además de asegurar que tenemos datos para la frecuencia mínima decidida (i.e. día hora) dentro los dataframes. De la misma manera se calcula el margen bruto para futuros análisis.

In [ ]:
```
df_master_summary = df_master_filter.groupby(by = ['producto','sale_timeStamp']).ag
    'precio_bruto' : [('precio_bruto', 'mean')],
    'precio_neto' : [('precio_neto', 'mean')],
    'volumen_despachado' : [('volumen_despachado', 'sum'),('transaction_num', 'coun
    'venta_neta' : [('venta_neta', 'sum')],
    'venta_bruta' : [('venta_bruta', 'sum')],
    'costo_bruto' : [('costo_bruto', 'mean')],
    'costo_neto' : [('costo_neto', 'mean')],
    'purchase_timeStamp' : [('purchase_timeStamp', 'min')]
}).reset_index()
df_master_summary.columns = [col[1] if col[1] else col[0] for col in df_master_summ
```

In [ ]:
```
df_master_summary['margen_bruto'] = df_master_summary['venta_bruta'] - df_master_su
df_master_summary['margen_neto'] = df_master_summary['venta_neta'] - df_master_summ
```

In [ ]:
```
df_master_summary.head()
```

In [ ]:
```
start_date = df_master_summary['sale_timeStamp'].min()
end_date = df_master_summary['sale_timeStamp'].max()

all_dates = pd.date_range(start = start_date, end = end_date, freq = 'h')
```

In [ ]:
```
def process_product_type(product_type):
    product_df = df_master_summary[df_master_summary['producto'] == product_type]
    product_df = product_df.set_index('sale_timeStamp').reindex(all_dates).rename_a

    product_df['precio_bruto'] = product_df['precio_bruto'].ffill()
    product_df['precio_neto'] = product_df['precio_neto'].ffill()

    product_df['costo_bruto'] = product_df['costo_bruto'].ffill()
    product_df['costo_neto'] = product_df['costo_neto'].ffill()

    product_df['purchase_timeStamp'] = product_df['purchase_timeStamp'].ffill()

    product_df['venta_neta'] = product_df['venta_neta'].fillna(0)
    product_df['venta_bruta'] = product_df['venta_bruta'].fillna(0)

    product_df['volumen_despachado'] = product_df['volumen_despachado'].fillna(0)
    product_df['transaction_num'] = product_df['transaction_num'].fillna(0)
    product_df['margen_bruto'] = product_df['margen_bruto'].fillna(0)
    product_df['margen_neto'] = product_df['margen_neto'].fillna(0)

    product_df['precio_bruto'] = product_df['precio_bruto'].fillna(0)
    product_df['precio_neto'] = product_df['precio_neto'].fillna(0)

    product_df['costo_bruto'] = product_df['costo_bruto'].fillna(0)
    product_df['costo_neto'] = product_df['costo_neto'].fillna(0)

    product_df['purchase_timeStamp'] = product_df['purchase_timeStamp'].fillna(star

    product_df['producto'] = product_type

    return product_df
```

In [ ]:
```
product_types = df_master_summary['producto'].unique()
product_dfs = {ptype: process_product_type(ptype) for ptype in product_types}
```

In [ ]:
```
df_magna = product_dfs['magna']
df_premium = product_dfs['premium']
df_diesel = product_dfs['diesel']
```

In [ ]:
```
df_product_time = pd.concat([df_magna, df_premium,df_diesel], ignore_index = True)
date_info(df_product_time,'sale_timeStamp')
```

In [ ]:
```
df_product_time.head()
```

__Análisis de Temporalidad General__

En esta sección se analizó de manera general si existe un patrón de demanda en algún periodo en particular. Parte de los hallagazgos fueron que la demanda de la gasolina magna ha disminuido significativamente desde finales de 2022, mientras que la venta del combustible PREMIUM muestra una ligera tendencia creciente en el mismo periodo. Esto genera una incógnita sobre si esta caida obedece a un cámbio en la dinámica de precios o si se debe a un competidor nuevo cercano a dicha sucursal o un factor exógeno del que el equipo no tenga conocimiento.

In [ ]:
```
df_timeseries = df_product_time[['clean_date', 'hour', 'month', 'year', 'dow', 'vol
```

In [ ]:
```
grouped_date = df_timeseries.groupby(by = ['clean_date', 'producto']).sum().reset_i
grouped_dayHour = df_timeseries.drop('clean_date', axis = 1).groupby(by = ['dow', '
grouped_dayMonth = df_timeseries.drop('clean_date', axis = 1).groupby(by = ['dow',
```

In [ ]:
```
plt.figure(figsize = (14, 7))
sns.lineplot(data = grouped_date, x = 'clean_date', y = 'volumen_despachado', hue =
plt.show()
```

En general se puede observa que no existe actividad significativa los días domingos en esta sucursal lo cuál hace mucho sentido dado que es una sucursal que se encuentra al costado de una carretera que generalmente tiene la mayoría de su tráfico entre semana.

In [ ]:
```
plt.figure(figsize = (14,7))
sns.heatmap(grouped_dayHour, cmap = 'viridis')
plt.show()
```

In [ ]:
```
plt.figure(figsize = (14,7))
sns.clustermap(grouped_dayHour, cmap = 'coolwarm', method = 'centroid')
plt.show()
```

De la misma manera se puede observar que los primeros meses del año son aquellos que existe más tráfico en la sucursal.

In [ ]:
```
plt.figure(figsize = (14,7))
sns.heatmap(grouped_dayMonth, cmap = 'viridis')
plt.show()
```

In [ ]:
```
plt.figure(figsize = (14,7))
sns.clustermap(grouped_dayMonth, cmap = 'coolwarm', method = 'centroid')
plt.show()
```

__Análisis de Demanda__

A continuación se incluyen los componentes de demanda que nos permitirán decidir un modelo de predicción de demanda para cada uno de los productos que existen (Magna, Premium y Diesel). Estos factores son tendencia, temporalidad y residuales y nos ayudaran a

identificar si existen ciclos continuos de demanda que faciliten el entrenamiento de nuestros algorítmos. Dentro de cada uno de los productos se puede visualizar que hay un alto componente de temporalidad que se tiene que considerar a la hora de construir el modelo, lo que es un indicativo que requeriremos los diferentes niveles de las estampas de tiempo (Hora, Dia, Mes).

In [ ]:
```
df_magna.set_index('sale_timeStamp', inplace = True)
df_magna.sort_index(inplace = True)
df_premium.set_index('sale_timeStamp', inplace = True)
df_premium.sort_index(inplace = True)
df_diesel.set_index('sale_timeStamp', inplace = True)
df_diesel.sort_index(inplace = True)

#daily_magna_df = df_magna['Volumen despachado'].resample('D').sum()
#daily_premium_df = df_premium['Volumen despachado'].resample('D').sum()
#daily_diesel_df = df_diesel['Volumen despachado'].resample('D').sum()
```

__Análisis de Demanda Magna__

In [ ]:
```
if not isinstance(df_magna.index, pd.DatetimeIndex):
    df_magna.index = pd.to_datetime(df_magna.index)

print("Start date:", df_magna.index[0])
print("End date:", df_magna.index[-1])
print("Number of data points:", len(df_magna))

if df_magna.index[0] <= pd.Timestamp.max - pd.DateOffset(days = len(df_magna)):
    if not df_magna.index.freq:
        df_magna.index.freq = 'h'
```

In [ ]:
```
decomposition_magna = sm.tsa.seasonal_decompose(df_magna['volumen_despachado'], mod

trend_magna = decomposition_magna.trend
seasonal_magna = decomposition_magna.seasonal
residual_magna = decomposition_magna.resid

fig = decomposition_magna.plot()
plt.show()
```

In [ ]:
```
fig = px.line(x = df_magna['volumen_despachado'].index, y = seasonal_magna, labels=
fig.show()
```

In [ ]:
```
print("Statistical Summary of Trend Component:")
print(trend_magna.describe())

print("\nStatistical Summary of Seasonal Component:")
print(seasonal_magna.describe())

print("\nStatistical Summary of Residual Component:")
print(residual_magna.describe())
```

In [ ]:
```
original = df_magna['volumen_despachado']
print("Correlation with Original Series:")
print("Trend Correlation:", original.corr(trend_magna))
print("Seasonal Correlation:", original.corr(seasonal_magna))
print("Residual Correlation:", original.corr(residual_magna))
```

In [ ]:
```
frequencies_magna, spectrum_magna = periodogram(seasonal_magna.dropna(), scaling =
plt.figure()
plt.plot(frequencies_magna, spectrum_magna)
plt.title('Power Spectrum of the Seasonal Component')
plt.xlabel('Frequency')
plt.ylabel('Spectral Power')
plt.show()
```

In [ ]:
```
plot_acf(residual_magna.dropna())
plt.title('Autocorrelation of Residuals')
plt.show()

plot_pacf(residual_magna.dropna())
plt.title('Partial Autocorrelation of Residuals')
plt.show()
```

__Análisis de Demanda Premium__

In [ ]:
```
if not isinstance(df_premium.index, pd.DatetimeIndex):
    df_premium.index = pd.to_datetime(df_premium.index)

print("Start date:", df_premium.index[0])
print("End date:", df_premium.index[-1])
print("Number of data points:", len(df_premium))

if df_premium.index[0] <= pd.Timestamp.max - pd.DateOffset(days = len(df_premium)):
    if not df_premium.index.freq:
        df_premium.index.freq = 'h'
```

In [ ]:
```
decomposition_premium = sm.tsa.seasonal_decompose(df_premium['volumen_despachado'],

trend_premium = decomposition_premium.trend
seasonal_premium = decomposition_premium.seasonal
residual_premium = decomposition_premium.resid

fig = decomposition_premium.plot()
plt.show()
```

In [ ]:
```
fig = px.line(x = df_premium['volumen_despachado'].index, y = seasonal_premium, lab
fig.show()
```

In [ ]:
```
print("Statistical Summary of Trend Component:")
print(trend_premium.describe())

print("\nStatistical Summary of Seasonal Component:")
print(seasonal_premium.describe())
```

```
print("\nStatistical Summary of Residual Component:")
print(residual_premium.describe())
```

In [ ]:
```
original = df_premium['volumen_despachado']
print("Correlation with Original Series:")
print("Trend Correlation:", original.corr(trend_premium))
print("Seasonal Correlation:", original.corr(seasonal_premium))
print("Residual Correlation:", original.corr(residual_premium))
```

In [ ]:
```
frequencies_premium, spectrum_premium = periodogram(seasonal_premium.dropna(), scal
plt.figure()
plt.plot(frequencies_premium, spectrum_premium)
plt.title('Power Spectrum of the Seasonal Component')
plt.xlabel('Frequency')
plt.ylabel('Spectral Power')
plt.show()
```

In [ ]:
```
plot_acf(residual_premium.dropna())
plt.title('Autocorrelation of Residuals')
plt.show()

plot_pacf(residual_premium.dropna())
plt.title('Partial Autocorrelation of Residuals')
plt.show()
```

__Análisis de Demanda Diesel__

In [ ]:
```
if not isinstance(df_diesel.index, pd.DatetimeIndex):
    df_diesel.index = pd.to_datetime(df_diesel.index)

print("Start date:", df_diesel.index[0])
print("End date:", df_diesel.index[-1])
print("Number of data points:", len(df_diesel))

if df_diesel.index[0] <= pd.Timestamp.max - pd.DateOffset(days = len(df_diesel)):
    if not df_diesel.index.freq:
        df_diesel.index.freq = 'h'
```

In [ ]:
```
decomposition_diesel = sm.tsa.seasonal_decompose(df_diesel['volumen_despachado'], m

trend_diesel = decomposition_diesel.trend
seasonal_diesel = decomposition_diesel.seasonal
residual_diesel = decomposition_diesel.resid

fig = decomposition_diesel.plot()
plt.show()
```

In [ ]:
```
fig = px.line(x = df_diesel['volumen_despachado'].index, y = seasonal_diesel, label
fig.show()
```

In [ ]:
```
print("Statistical Summary of Trend Component:")
print(trend_diesel.describe())

print("\nStatistical Summary of Seasonal Component:")
```

```
print(seasonal_diesel.describe())

print("\nStatistical Summary of Residual Component:")
print(residual_diesel.describe())
```

In [ ]:
```
original = df_diesel['volumen_despachado']
print("Correlation with Original Series:")
print("Trend Correlation:", original.corr(trend_diesel))
print("Seasonal Correlation:", original.corr(seasonal_diesel))
print("Residual Correlation:", original.corr(residual_diesel))
```

In [ ]:
```
frequencies_diesel, spectrum_diesel = periodogram(seasonal_diesel.dropna(), scaling
plt.figure()
plt.plot(frequencies_diesel, spectrum_diesel)
plt.title('Power Spectrum of the Seasonal Component')
plt.xlabel('Frequency')
plt.ylabel('Spectral Power')
plt.show()
```

In [ ]:
```
plot_acf(residual_diesel.dropna())
plt.title('Autocorrelation of Residuals')
plt.show()

plot_pacf(residual_diesel.dropna())
plt.title('Partial Autocorrelation of Residuals')
plt.show()
```

In [ ]:
```
df_magna.to_pickle(f'{path}base_magna.pkl')
df_premium.to_pickle(f'{path}base_premium.pkl')
df_diesel.to_pickle(f'{path}base_diesel.pkl')
```

Identificación de Factores Exógenos

Un factor importante para determinar el precio de venta de gasolina es el precio del petróleo del cual se derivó la gasolina que se venderá. Para esto tomamos la base del EIA (US Energy Information Administration) del precio de petróleo BRENT (referencia para México). En esta sección analizaremos la relación del precio del petróleo con los precios de compra (es decir los precios a los cuales las gasolineras adquirieron el producto). Debido a que el precio internacional del petróleo no tiene un efecto inmediato sobre el precio de los proveedores, se debe realizar un análisis de correlación con rezago, para poder determinar el impacto del precio del petróleo en los precios de distribución.

In [ ]:
```
base_master = pd.read_pickle(f'{path}base_master.pkl')
```

In [ ]:
```
db_petroleo = pd.read_csv(f'{path}Europe_Brent_Spot_Price_FOB.csv', header = None)
db_petroleo.columns = ['fecha', 'precio_brent']
db_petroleo['market_date'] = pd.to_datetime(db_petroleo['fecha'])
```

In [ ]:
```
all_days = pd.date_range(start = db_petroleo['market_date'].min(), end = db_petrole
```

In [ ]:
```
db_petroleo = db_petroleo.set_index('market_date').reindex(all_days).ffill()
db_petroleo = db_petroleo.reset_index()
db_petroleo = db_petroleo.drop(index = 0).reset_index(drop = True)
db_petroleo = db_petroleo.rename(columns = {'index': 'market_date'})
ppetroleo = db_petroleo[db_petroleo['market_date'] > '2021-12-30']
```

Básicamente podemos observar que existe una correlación visual entre el precio del petróleo Brent y el precio neto de compra al cual la gasolinera adquiere los derivados de petróleo. Con esta información visual, haremos un análisis de correlación con rezago para determinar el rezago óptimo del precio del petróleo con el costo de adquisición por tipo de combustible.

In [ ]:
```
base_combustible = base_master.loc[base_master['producto'] == 'magna']

precio_promedio_diario = base_combustible.groupby('purchase_date')['costo_neto'].me

precio_promedio_diario.columns = ['purchase_date', 'costo_neto']

merged_db_price_cost = pd.merge(precio_promedio_diario, ppetroleo, left_on = 'purch

data_p_c = merged_db_price_cost[['purchase_date', 'costo_neto', 'market_date', 'pre

fig, ax1 = plt.subplots(figsize=(10, 6))

color = 'tab:blue'
ax1.set_xlabel('Fecha')
ax1.set_ylabel('Precio Brent', color = color)
ax1.plot(data_p_c['market_date'], data_p_c['precio_brent'], color = color, label =
ax1.tick_params(axis='y', labelcolor=color)


ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Costo Neto', color = color)
ax2.plot(data_p_c['market_date'], data_p_c['costo_neto'], color=color, label='Costo
ax2.tick_params(axis='y', labelcolor=color)

# Título y leyenda
plt.title('Costo Neto Promedio Diario MAGNA vs Precio diario del Petróleo Brent')
fig.tight_layout()
plt.show()
```

In [ ]:
```
combustibles = ['magna', 'premium', 'diesel']

for combustible in combustibles:
    base_combustible = base_master.loc[base_master['producto'] == combustible]

    precio_promedio_diario = base_combustible.groupby('purchase_date')['costo_neto'
    precio_promedio_diario.columns = ['purchase_date', 'costo_neto']
    merged_data = pd.merge(precio_promedio_diario, ppetroleo, left_on = 'purchase_d

    data = merged_data[['purchase_date', 'costo_neto', 'market_date', 'precio_brent
```

```
    lag_acf = plot_acf(data['precio_brent'], lags = 45)

    plt.xlabel('Rezago (días)')
    plt.ylabel('Autocorrelación')
    plt.title(f'Autocorrelación del precio del petróleo Brent {combustible}')
    plt.show()

  # Calcular la correlación entre el precio del petróleo Brent y el costo neto con
    correlation_results = {}
    for lag in range(1, 46):
        data['precio_brent_lagged'] = data['precio_brent'].shift(lag)
        correlation = data[['precio_brent_lagged', 'costo_neto']].corr().iloc[0, 1]
        correlation_results[lag] = correlation


    plt.plot(list(correlation_results.keys()), list(correlation_results.values()),
    plt.xlabel('Rezago (días)')
    plt.ylabel('Correlación')
    plt.title(f'Correlación entre el precio del petróleo Brent y el precio neto - {
    plt.show()

    optimal_lag = max(correlation_results, key = lambda key: abs(correlation_result

    print("Rezago óptimo:", optimal_lag)
```

Tras el análisis de correlación con rezago pudimos determinar que existen diferentes rezagos óptimos para cada tipo de combustible. Para MAGNA, el rezago óptimo es de 2 días, es decir, el precio del petróleo con un rezago de 2 días tiene mayor correlación con el costo de adquisición. Para PREMIUM, el rezago óptimo es de 17 días y para DIESEL, el rezago óptimo es de 8 días. Esta diferencia en rezagos puede deberse a la naturaleza del tipo de combustible. Debido a que encontramos esta correlación, debemos agregar la variable del precio del petróleo a nuestra base de datos. Para esto agregaremos el precio del petróleo con su rezago óptimo por cada tipo de combustible para cada data frame de combustible.

In [ ]:
```
df_magna = pd.read_pickle(f'{path}base_magna.pkl')
df_premium = pd.read_pickle(f'{path}base_premium.pkl')
df_diesel = pd.read_pickle(f'{path}base_diesel.pkl')
```

In [ ]:
```
df_magna['purchase_timeStamp'] = df_magna['purchase_timeStamp'].dt.floor('D')
df_premium['purchase_timeStamp'] = df_premium['purchase_timeStamp'].dt.floor('D')
df_diesel['purchase_timeStamp'] = df_diesel['purchase_timeStamp'].dt.floor('D')
```

In [ ]:
```
rezago_magna = 2
rezago_premium = 17
rezago_diesel = 8

ppetroleo_magna = db_petroleo.copy()
ppetroleo_magna['market_date'] = ppetroleo_magna['market_date'] + pd.DateOffset(day
df_magna = pd.merge(df_magna.reset_index(), ppetroleo_magna.drop(['fecha'], axis =
df_magna.drop('market_date', axis = 1, inplace = True)
date_info(df_magna,'sale_timeStamp')

ppetroleo_premium = db_petroleo.copy()
```

```
ppetroleo_premium['market_date'] = ppetroleo_premium['market_date'] + pd.DateOffset
df_premium = pd.merge(df_premium.reset_index(), ppetroleo_premium.drop(['fecha'], a
df_premium.drop('market_date', axis = 1, inplace = True)
date_info(df_premium,'sale_timeStamp')

ppetroleo_diesel = db_petroleo.copy()
ppetroleo_diesel['market_date'] = ppetroleo_diesel['market_date'] + pd.DateOffset(d
df_diesel = pd.merge(df_diesel.reset_index(), ppetroleo_diesel.drop(['fecha'], axis
df_diesel.drop('market_date', axis = 1, inplace = True)
date_info(df_diesel,'sale_timeStamp')
```

In [ ]:
```
df_magna.to_pickle(f'{path}base_magna_final.pkl')
df_premium.to_pickle(f'{path}base_premium_final.pkl')
df_diesel.to_pickle(f'{path}base_diesel_final.pkl')
```

Conclusiones Avance 2

Para nuestro modelo de optimización de precios la preparación de datos es el paso más importante previo al modelamiento debido a que logramos comprender la base de datos disponible por parte de las gasolineras, al igual que el contexto económico del negocio y las variables que afectan las ventas de combustible.

En la comprensión de la base de datos, identificamos variables de oferta y demanda de las gasolineras al igual que obtuvimos nuestra variable dependiente para nuestro modelo que es el Margen de Ganancias. De igual manera, obtuvimos información sobre el comportamiento temporal de nuestras variables para extraer insights sobre el funcionamiento del negocio y su estado actual (como mencionamos anteriormente, encontramos una reducción en el volumen de combustible vendido a lo largo del periodo 2022- 2024). También ligamos el comportamiento de los costos de combustibles al precio internacional del petróleo, de tal manera que obtuvimos una variable externa que nos otorga bastante información que puede ser útil en el modelamiento de optimización de precios.

También comprendimos que es necesaria la división de nuestra base de datos en tipo de Combustible ofertado. Esto debido a que la estacionalidad y tendencia de ventas varía de acuerdo con el tipo de combustible, al igual que el precio del petróleo influye de diferente manera a cada tipo de combustible. Por lo cual, en lugar de crear variables dummies para cada tipo de combustible, vamos a crear un modelo especifico para cada tipo de combustible disponible en nuestra base de datos.

Avance 3.Ingeniería de características

Mayo 2024

Objetivos

3.1 Establecer las medidas de calidad del modelo de aprendizaje automático.

3.2 Proporcionar un marco de referencia para evaluar y mejorar modelos más avanzado .

Instruccion*

Este avance consiste en construir un modelo de referencia que permita evaluar la viabilidad del problema. Si el baseline tiene un rendimiento similar al azar, podría indicar que el problema es intrínsecamente difícil o que los datos no contienen suficiente información para predecir el objetivo. De lo contrario, el baseline podría como una solución mínima aceptable cuando se trabaja en escenarios donde incluso un modelo simple puede proporcionar valor.

Un baseline facilita también la gestión de expectativas, tanto dentro del equipo como con los stakeholders, pues proporciona una comprensión inicial de lo que se puede lograr con métodos simples antes de invertir tiempo y recursos en enfoques más complejos.

Las siguientes son algunas de las preguntas que deberán abordar durante esta fase:

¿Qué algoritmo se puede utilizar como baseline para predecir las variables objetivo? ¿Se puede determinar la importancia de las características para el modelo generado? Recuerden que incluir características irrelevantes puede afectar negativamente el rendimiento del modelo y aumentar la complejidad sin beneficios sustanciales. ¿El modelo está sub/sobreajustando los datos de entrenamiento? ¿Cuál es la métrica adecuada para este problema de negocio? ¿Cuál debería ser el desempeño mínimo obtener? o real, en un conjunto de variables útiles para el aprendizaje automático. El procesamiento puede incluir: Generación de nuevas características Discretización o binning Codificación (ordinal, one hot,…) Escalamiento (normalización, estandarización, min – max,…) Transformación (logarítmica, exponencial, raíz cuadrada, Box – Cox, Yeo – Johnson,…)

- Todas las decisiones y técnicas empleadas deben ser justificadas.

Además, se utilizarán métodos de filtrado para la selección de características y técnicas de extracción de características, permitiendo reducir los requerimientos de almacenamiento, la complejidad del modelo y el tiempo de entrenamiento. Los ejemplos siguientes son ilustrativos, pero no exhaustivos, de lo que se podría aplicar: Umbral de varianza Correlación Chi-cuadrado ANOVA Análisis de componentes principales (PCA) Análisis factorial (FA)

- Es necesario fundamentar los métodos ejecutados.

Incluir conclusiones de la fase de "Preparación de los datos" en el contexto de la metodología CRISP-ML.

Para nuestro modelo baseline vamos a elegir un modelo de Random Forest para la demanda (volumen_despachado) y un modelo genético para la determinación del precio. Elegimos este tipo de algortimo porque nos facilita capturar las relaciones de mayor complejidad entre variable dependiente y variables independientes. De igual manera, el Random Forest ayuda a la reducción de *overfitting*. También porque nos permite identificar las variables importantes en nuestra predicción, de tal manera que podamos eliminar aquellas que no afectan significativamente a la variable objetivo y puedan causar ruido en la predicción si

son incluidas. De igual manera, utilizamos un modelo genético debido a su capacidad de capturar relaciones complejas entre las variables, la capacidad de incorporar restricciones a las diferentes variables del negocio, la penalización a la variable objetivo, que en este caso evita que el precio óptimo se encuentre en un rango fuera de lo normal (es decir, evita que el precio óptimo sea absurdamente alto porque a mayor precio mayor rentabilidad, y lo ubica dentro de un rango razonable dentro de las datos observados).

In [4]:
```
df_magna = pd.read_pickle(f'{path}base_magna_final.pkl')
df_premium = pd.read_pickle(f'{path}base_premium_final.pkl')
df_diesel = pd.read_pickle(f'{path}base_diesel_final.pkl')
```

Para nuestro modelo de optimización de precios de gasolina, reiteramos que vamos a trabajar sobre 3 tipos de combustible por separado (MAGNA, PREMIUM, DIESEL), de tal manera que obtendremos 3 modelos diferentes de optimización de precios.

In [5]:
```
df_magna_alt = df_magna[['volumen_despachado', 'precio_neto']].copy()
original_type_a = df_magna['volumen_despachado'].dtype
original_type_b = df_magna['precio_neto'].dtype

df_magna_alt.replace(0, pd.NA, inplace = True)
df_magna_alt.dropna(inplace = True)

df_magna_alt['volumen_despachado'] = df_magna_alt['volumen_despachado'].astype(orig
df_magna_alt['precio_neto'] = df_magna_alt['precio_neto'].astype(original_type_b)
```

In [6]:
```
df_premium_alt = df_premium[['volumen_despachado', 'precio_neto']].copy()
original_type_a = df_premium['volumen_despachado'].dtype
original_type_b = df_premium['precio_neto'].dtype

df_premium_alt.replace(0, pd.NA, inplace = True)
df_premium_alt.dropna(inplace = True)

df_premium_alt['volumen_despachado'] = df_premium_alt['volumen_despachado'].astype(
df_premium_alt['precio_neto'] = df_premium_alt['precio_neto'].astype(original_type_
```

In [7]:
```
df_diesel_alt = df_diesel[['volumen_despachado', 'precio_neto']].copy()
original_type_a = df_diesel['volumen_despachado'].dtype
original_type_b = df_diesel['precio_neto'].dtype

df_diesel_alt.replace(0, pd.NA, inplace = True)
df_diesel_alt.dropna(inplace = True)

df_diesel_alt['volumen_despachado'] = df_diesel_alt['volumen_despachado'].astype(or
df_diesel_alt['precio_neto'] = df_diesel_alt['precio_neto'].astype(original_type_b)
```

In [8]:
```
df_magna_alt['log_price'] = np.log(df_magna_alt['precio_neto'])
df_magna_alt['log_demand'] = np.log(df_magna_alt['volumen_despachado'])

df_premium_alt['log_price'] = np.log(df_premium_alt['precio_neto'])
df_premium_alt['log_demand'] = np.log(df_premium_alt['volumen_despachado'])
```

```
df_diesel_alt['log_price'] = np.log(df_diesel_alt['precio_neto'])
df_diesel_alt['log_demand'] = np.log(df_diesel_alt['volumen_despachado'])
```

Para los features de nuestro modelo estamos incluyendo precio_neto, costo_neto, precio_brent (rezagado), y variables temporales de hora, día y mes. Nuestra variable objetivo es el volumen despachado o como lo definimos la demanda de la gasolinera.

In [9]:
```
features_magna = df_magna[['precio_neto', 'costo_neto', 'precio_brent', 'hour', 'mo
target_magna = df_magna['volumen_despachado'].copy()

features_premium = df_premium[['precio_neto', 'costo_neto', 'precio_brent', 'hour',
target_premium = df_premium['volumen_despachado'].copy()

features_diesel = df_diesel[['precio_neto', 'costo_neto', 'precio_brent', 'hour', '
target_diesel = df_diesel['volumen_despachado'].copy()
```

Antes de desarrollar el modelo realizamos una codificación de las variables temporales para capturar el ciclo temporal en los datos. Esto se realiza con el seno y coseno de las variables temporales de hora, día y mes.

In [10]:
```
def feature_engineering(df_interest):
    df_interest['hour_sin'] = np.sin(2 * np.pi * df_interest['hour']/23.0)
    df_interest['hour_cos'] = np.cos(2 * np.pi * df_interest['hour']/23.0)
    df_interest['day_sin'] = np.sin(2 * np.pi * df_interest['dow_n']/6.0)
    df_interest['day_cos'] = np.cos(2 * np.pi * df_interest['dow_n']/6.0)
    df_interest['month_sin'] = np.sin(2 * np.pi * df_interest['month']/11.0)
    df_interest['month_cos'] = np.cos(2 * np.pi * df_interest['month']/11.0)
    df_interest.drop(['hour', 'dow_n', 'month'], axis = 1, inplace = True)
```

In [11]:
```
feature_engineering(features_magna)
feature_engineering(features_premium)
feature_engineering(features_diesel)
```

Otra variable que estamos agregando para el modelo genético es la elasticidad de la demanda. Esta variable también es relevante porque nos ayudará a determinar como cambia la demanda cuando el precio se modifica. La elasticidad es importante porque nos da información sobre el producto, por ejemplo, si una pequeña variación del precio genera un cambio significativo en la demanda entonces el producto se considera elástico); o pese a un cambio importante en el precio, la cantidad demandada se mantiene igual, el producto se considera inelástico. Para obtener la elasticidad de la demanda estamos realizando una regresión linear con los logaritmos de nuestros datos de precio_neto y volumen_despachado.

In [12]:
```
model_lr_elasticity_magna = LinearRegression()
model_lr_elasticity_magna.fit(df_magna_alt[['log_price']], df_magna_alt[['log_deman
price_elasticity_magna = model_lr_elasticity_magna.coef_[0]

model_lr_elasticity_premium = LinearRegression()
model_lr_elasticity_premium.fit(df_premium_alt[['log_price']], df_premium_alt[['log
price_elasticity_premium = model_lr_elasticity_premium.coef_[0]
```

```
model_lr_elasticity_diesel = LinearRegression()
model_lr_elasticity_diesel.fit(df_diesel_alt[['log_price']], df_diesel_alt[['log_de
price_elasticity_diesel = model_lr_elasticity_diesel.coef_[0]
```

In [13]:
```
print(f'Elasticidad Magna: {price_elasticity_magna[0]:.2f}')
print(f'Elasticidad Premium: {price_elasticity_premium[0]:.2f}')
print(f'Elasticidad Diesel: {price_elasticity_diesel[0]:.2f}')
```

```
Elasticidad Magna: -0.45
Elasticidad Premium: 2.15
Elasticidad Diesel: 0.10
```

In [14]:
```
scaler = StandardScaler()
```

In [15]:
```
features_scaled_magna = scaler.fit_transform(features_magna)
features_scaled_premium = scaler.fit_transform(features_premium)
features_scaled_diesel = scaler.fit_transform(features_diesel)
```

In [16]:
```
X_train_magna, X_test_magna, y_train_magna, y_test_magna = train_test_split(feature
X_train_premium, X_test_premium, y_train_premium, y_test_premium = train_test_split
X_train_diesel, X_test_diesel, y_train_diesel, y_test_diesel = train_test_split(fea
```

In [17]:
```
model_magna = RandomForestRegressor(n_estimators = 100, random_state = 42)
model_premium = RandomForestRegressor(n_estimators = 100, random_state = 42)
model_diesel = RandomForestRegressor(n_estimators = 100, random_state = 42)
```

In [18]:
```
model_magna.fit(X_train_magna, y_train_magna)
model_premium.fit(X_train_premium, y_train_premium)
model_diesel.fit(X_train_diesel, y_train_diesel)
```

Out[18]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [19]:
```
y_pred_magna = model_magna.predict(X_test_magna)
print('Demand Prediction Model MSE (Magna):', mean_squared_error(y_test_magna, y_pr

y_pred_premium = model_premium.predict(X_test_premium)
print('Demand Prediction Model MSE (Premium):', mean_squared_error(y_test_premium,

y_pred_diesel = model_diesel.predict(X_test_diesel)
print('Demand Prediction Model MSE (Diesel):', mean_squared_error(y_test_diesel, y_
```

```
Demand Prediction Model MSE (Magna): 24260.906214291863
Demand Prediction Model MSE (Premium): 1659.8372604914277
Demand Prediction Model MSE (Diesel): 1059.0441196169459
```

Optmización de Modelo: Modelo Genómico

Una vez que obtenemos el RandomForest del volumen despachado, procedemos a realizar un algoritmo genético para la optimización de precio. Dentro del modelo genético estamos generando individuos, población, cruce, mutación y selección. Una vez que definimos estos

parametros procedemos a calcular el margun bruto utilizando nuestra base de datos de entrenamiento, nuestro modelo RandomForest, la elasticidad del precio (que calculamos con regresión lineal) y el precio actual. Por último, se incluye una penalización calculada en función a la diferencia del precio_pred con el precio actual.

In [20]:
```
creator.create('FitnessMax', base.Fitness, weights = (1.0,))
creator.create('Individual', list, fitness = creator.FitnessMax)
```

In [21]:
```
toolbox_magna = base.Toolbox()
toolbox_magna.register('attr_float', random.uniform, 15, 30)  # Price range
toolbox_magna.register('individual', tools.initRepeat, creator.Individual, toolbox_
toolbox_magna.register('population', tools.initRepeat, list, toolbox_magna.individu
```

In [22]:
```
def evalGrossMargin(individual, train_data, model_type, feature_names, current_pric
    try:
        # Price from the individual
        price = individual[0]

        # Calculate mean values of each feature in train_data
        relevant_columns = [col for col in feature_names if col != 'precio_neto']
        df = pd.DataFrame(train_data[:, 1:], columns = relevant_columns)
        mean_values = df[relevant_columns].mean()

        data_array = np.insert(mean_values.values, 0, price)
        temp_features = pd.DataFrame([data_array], columns = feature_names)
        scaled_features = scaler.transform(temp_features)


        predicted_demand = model_type.predict(scaled_features)[0]
        cost_of_goods_sold = temp_features['costo_neto'].iloc[0]
        gross_margin = (price - cost_of_goods_sold) * predicted_demand

        penalty_factor = 10
        penalty = penalty_factor * (abs(price - current_price) / current_price) **

        gross_margin -= penalty

        return (gross_margin,)
    except Exception as e:
        print("Error in evalGrossMargin:", e)
        return (0,)  # Return a default or zero fitness in case of error
```

In [23]:
```
current_price_magna = df_magna['precio_neto'].mean()
upper_bound_magna = current_price_magna * 1.3
lower_bound_magna = current_price_magna * 0.7
```

In [24]:
```
toolbox_magna.register('attr_float', random.uniform, lower_bound_magna, upper_bound
toolbox_magna.register('evaluate', functools.partial(evalGrossMargin, train_data =
toolbox_magna.register('mate', tools.cxBlend, alpha = 0.5)
toolbox_magna.register('mutate', tools.mutGaussian, mu = 0, sigma = 10, indpb = 0.2
toolbox_magna.register('select', tools.selTournament, tournsize = 3)
```

In [25]:
```
population_magna = toolbox_magna.population(n = 50)

NGEN = 40
CXPB = 0.5
MUTPB = 0.2
```

In [26]:
```
for gen in range(NGEN):
    offspring = algorithms.varAnd(population_magna, toolbox_magna, cxpb = CXPB, mut
    fits = toolbox_magna.map(toolbox_magna.evaluate, offspring)
    print(f"Generation {gen}: Fitness Values - {fits}")
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population_magna = toolbox_magna.select(offspring, k = len(population_magna))
```

```
Generation 0: Fitness Values - <map object at 0x000002D9E561C700>
Generation 1: Fitness Values - <map object at 0x000002D9E561D240>
Generation 2: Fitness Values - <map object at 0x000002D9E561FB80>
Generation 3: Fitness Values - <map object at 0x000002D9E561C190>
Generation 4: Fitness Values - <map object at 0x000002D9E561C310>
Generation 5: Fitness Values - <map object at 0x000002D9E561D240>
Generation 6: Fitness Values - <map object at 0x000002D9E561C310>
Generation 7: Fitness Values - <map object at 0x000002D9E561DAE0>
Generation 8: Fitness Values - <map object at 0x000002D9E561D9F0>
Generation 9: Fitness Values - <map object at 0x000002D9E561D240>
Generation 10: Fitness Values - <map object at 0x000002D9E561D9F0>
Generation 11: Fitness Values - <map object at 0x000002D9E561D240>
Generation 12: Fitness Values - <map object at 0x000002D9E561D9F0>
Generation 13: Fitness Values - <map object at 0x000002D9E561D240>
Generation 14: Fitness Values - <map object at 0x000002D9E6C037C0>
Generation 15: Fitness Values - <map object at 0x000002D9E561D240>
Generation 16: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 17: Fitness Values - <map object at 0x000002D9E561D240>
Generation 18: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 19: Fitness Values - <map object at 0x000002D9E561D240>
Generation 20: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 21: Fitness Values - <map object at 0x000002D9E561D240>
Generation 22: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 23: Fitness Values - <map object at 0x000002D9E561D240>
Generation 24: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 25: Fitness Values - <map object at 0x000002D9E561D240>
Generation 26: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 27: Fitness Values - <map object at 0x000002D9E561D240>
Generation 28: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 29: Fitness Values - <map object at 0x000002D9E561D240>
Generation 30: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 31: Fitness Values - <map object at 0x000002D9E561D240>
Generation 32: Fitness Values - <map object at 0x000002D9E6C037C0>
Generation 33: Fitness Values - <map object at 0x000002D9E561D240>
Generation 34: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 35: Fitness Values - <map object at 0x000002D9E561D240>
Generation 36: Fitness Values - <map object at 0x000002D9E6C037C0>
Generation 37: Fitness Values - <map object at 0x000002D9E561D240>
Generation 38: Fitness Values - <map object at 0x000002D9E56AB640>
Generation 39: Fitness Values - <map object at 0x000002D9E561D240>
```

In [27]:
```
top_individual = tools.selBest(population_magna, k = 1)[0]
max_gross_margin = float(top_individual.fitness.values[0])

print(f'Optimal price: ${top_individual[0]:.2f}')
print(f'Maximum Gross Margin: ${max_gross_margin:.2f}')
```

```
Optimal price: $200.03
Maximum Gross Margin: $87791.22
```

```
C:\Users\hzapi\AppData\Local\Temp\ipykernel_69228\3682175674.py:2: DeprecationWarnin
g: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before performing this o
peration. (Deprecated NumPy 1.25.)
  max_gross_margin = float(top_individual.fitness.values[0])
```