Instituto Tecnológico de Monterrey

Maestría en Inteligencia Artificial Aplicada

Avance 4.- Modelos alternativos

Integrantes:

- Nancy Elena Estanislao Lizárraga - A01169334
- Héctor Raúl Vázquez González - A00565542
- Gustavo Rene Ramos Ojeda - A01793599

Preprocesamiento de Información

Importación de Librerías

```
#Obtención de Datos
import pandas as pd
#Escalamiento
from sklearn.preprocessing import MinMaxScaler
import numpy as np
#Análisis de Características
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from scipy.stats import boxcox
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

#Modelos
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge,
ElasticNet
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import accuracy_score
```

Obtención de Datos

Se obtiene la información desde un sitio en internet, se guarda en un Dataframe. Por ultimo, se eliminan los datos nulos

```
#Path del archivo CSV con la información con la que vamos a Trabajar
FilePath =
'https://raw.githubusercontent.com/gusrro/ProyectoIntegrador/main/
dataF.csv'
#Creación del Dataframe usando el path del archivo
df_original = pd.read_csv(FilePath)

#Creamos una copia del dataframe original
#A partir de este punto se trabajará con esta copia
df_notnull = df_original.copy()
#Eliminamos las filas que contengan espacios nulos
df_notnull.dropna( axis=0, inplace=True)
```

Análisis de la información

Se observan las columnas en la información obtenida

```
df_analisis = df_notnull.copy()
df_analisis.columns
```

```
Index(['day', 'mont', 'year', 'credited', 'debited', 'type',
'week_day',
       'perc_remain'],
      dtype='object')
```

```
df_analisis.describe()
```


```
{"summary":"{\n  \"name\": \"df_analisis\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
337.14419059277066,\n        \"min\": 1.0,\n        \"max\": 968.0,\n
\"samples\": [\n          15.603305785123966,\n          16.0,\n
968.0\n        ],\n        \"num_unique_values\": 8,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mont\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 340.0787801058672,\n
\"min\": 1.0,\n        \"max\": 968.0,\n        \"samples\": [\n
6.632231404958677,\n          7.0,\n          968.0\n        ],\n
\"num_unique_values\": 8,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 756.9768845404928,\n        \"min\": 0.8470504507660989,\n
\"max\": 2023.0,\n        \"samples\": [\n          968.0,\n
2021.3419421487604,\n          2022.0\n        ],\n
\"num_unique_values\": 7,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"credited\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11140414.213549685,\n        \"min\":
2.88225,\n        \"max\": 34564319.43,\n        \"samples\": [\n
8817488.188247407,\n          10134945.395,\n          968.0\
```

```
n        ],\n        \"num_unique_values\": 8,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"debited\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
8383156.500644588,\n        \"min\": -356515.4121,\n        \"max\":
25695066.21,\n        \"samples\": [\n          6807160.406342272,\n
7884899.9035,\n          968.0\n        ],\n
\"num_unique_values\": 8,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"week_day\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 340.94086240844564,\n        \"min\":
1.0,\n        \"max\": 968.0,\n        \"samples\": [\n
3.87603305785124,\n          4.0,\n          968.0\n        ],\n
\"num_unique_values\": 8,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"perc_remain\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 25379.647697954017,\n        \"min\":
0.120654739,\n        \"max\": 72227.74843,\n        \"samples\": [\n
75.13465444102893,\n          0.2291058815,\n          968.0\
n        ],\n        \"num_unique_values\": 8,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
df_analisis.dtypes
```

```
day              int64
mont             int64
year             int64
credited       float64
debited        float64
type            object
week_day         int64
perc_remain    float64
dtype: object
```

El Set de información contiene 968 registros con 8 columnas. Las columnas que se pueden obervar son:

- **day**.- Esta Columna contiene los datos de qué día del mes es el registro. La información es de enteros en el rango de 1 a 31.
- **mont**.-Esta Columna contiene los datos de qué mes es el registro. La información es de enteros en el rango de 1 a 12.
- **year**.-Esta Columna contiene los datos de qué año es el registro. La información es de enteros en el rango de 2020 a 2023.
- **credited**.-Esta columna contiene el dato del monto que la institución depositó en su cuente de Banjico el día especificado por las otras columnas.
- **debited**.- Esta columna contiene el dato del monto que la institución utlizó en su cuente de Banjico el día especificado por las otras columnas.

- **type**.- En esta columna se especifica como se considera la fecha especificada en las otras columnas. La información que contiene esta columna se reduce a un caracter; 'N' para día nomrmal y 'P' para día festivo.
- **week_day**.- Esta Columna contiene los datos de qué día de la semana es la fecha indicada por las demás columnas. La información es de enteros en el rango de 1 a 7.
- **perc_remain**.-Esta Columna contiene los datos del porcentaje del monto que quedó sin utilizar en la fecha indicada por las demás columnas.

Actualmente se puede trabajar con todas las columnas, excepto con la columna de "type" ya que al ser de tipo "objeto" limita mucho las herramientas que se pueden utilizar. Lo que se puede hacer para mejorar este escenario, sin perder nada de informacíón, es cambiar las "N" por "0" y las "P" por "1". De esta forma podemos utilizar esta columna cono entero o boleano, según nos convenga.

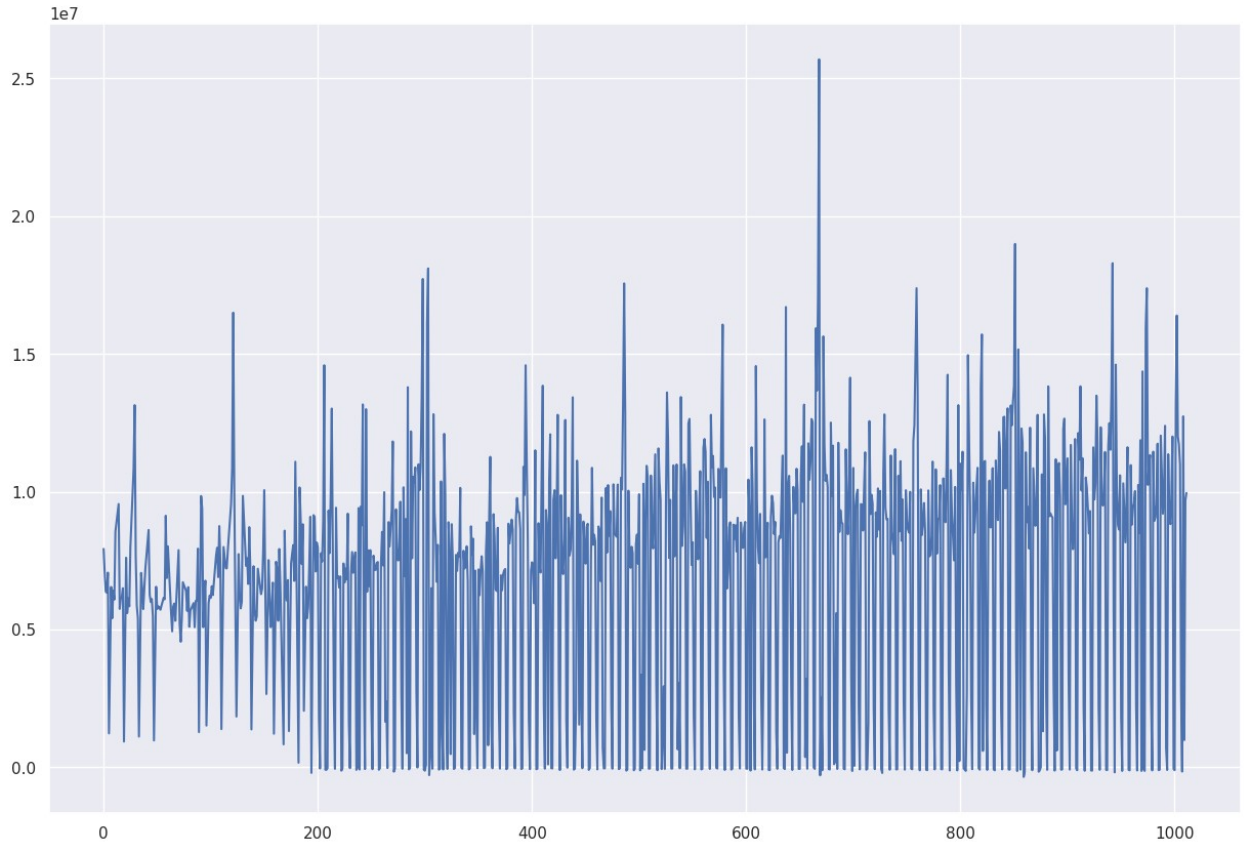
```
#Se defina la función para cambiar el tipo de dato de la columna
"type"
def change_typeColumn(df):
  df['type'] = df['type'].map({'N': False, 'P': True})
  return df

#Se procesa el cambio
df_analisis = change_typeColumn(df_analisis)

#Visualizamos los cambios
df_analisis.dtypes
```

```
day              int64
mont             int64
year             int64
credited       float64
debited        float64
type              bool
week_day         int64
perc_remain    float64
dtype: object
```

Ya que el objetivo del proyecto será "predecir en lo mayor posible cuanto gastará la institución en un particular día" podemos volcar nuestra atención en la columna "debited".

Visualicemos como se encuentra la información:

```
#Se establece el tamaño de la gráfica
sns.set( rc= {'figure.figsize':(15, 10)})

#Se grafica la información
df_grafica = df_analisis.copy()
plt.plot(df_grafica["debited"])
plt.show()
```



Podemos observar que existen outliers en la parte superior y en la parte inferior de la tendencia. Estos outliers podrían introducir ruido en el proceso de aprendizaje de nuestros modelos.

Vamos a separar en dataframe diferentes los outliers para estudiarlos por separados.

```
#Se declaran los dataframe que se utilizarán
df_min_noise = []
df_max_noise = []
df_noiseless = []
```

Ahora, estudiariaremos el ruido en la parte inferior de la gráfica. Según la gráfica original, podemos considerar todos los valores por debajo de "5,000,000" como outliers.

Separaremos estos outliers para analizarlos.

```
df_min_noise = df_analisis[df_analisis["debited"] <= 5000000]
df_min_noise
```


```
{"summary":"{\n  \"name\": \"df_min_noise\",\n  \"rows\": 287,\n
\"fields\": [\n    {\n      \"column\": \"day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 1,\n        \"max\": 31,\n        \"samples\": [\
n          1,\n          25,\n          13\n        ],\n
\"num_unique_values\": 31,\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mont\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 3,\n   \"min\": 1,\n   \"max\": 12,\n
\"samples\": [\n   4,\n   3,\n   6\n   ],\n
\"num_unique_values\": 12,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"year\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0,\n   \"min\": 2020,\n   \"max\": 2023,\n
\"samples\": [\n   2021,\n   2023,\n   2020\n
],\n   \"num_unique_values\": 4,\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"credited\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1765661.735126993,\n   \"min\":
2.88225,\n   \"max\": 7218955.388,\n   \"samples\": [\n
3450229.176,\n   141609.8158,\n   254437.384\
n   ],\n   \"num_unique_values\": 287,\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"debited\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1153289.6396639529,\n   \"min\": -356515.4121,\n   \"max\":
4929447.443,\n   \"samples\": [\n   1837791.081,\n
-88436.74162,\n   -24555.06407\n   ],\n
\"num_unique_values\": 287,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"type\",\n   \"properties\": {\n   \"dtype\": \"boolean\",\n
\"samples\": [\n   true,\n   false\n   ],\n
\"num_unique_values\": 2,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"week_day\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1,\n   \"min\": 1,\n
\"max\": 7,\n   \"samples\": [\n   6,\n   2\n
],\n   \"num_unique_values\": 7,\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"perc_remain\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 4263.3958594095875,\n
\"min\": 0.235472641,\n   \"max\": 72227.74843,\n
\"samples\": [\n   0.467342316,\n   1.624509969\n
],\n   \"num_unique_values\": 287,\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df_min_noise"}

Como podemos ver, las fechas indicadas en estos outliers corresponden a sábados y Domingos; son días que la institución no trabaja, y por tanto el modelo no será consultado para dichas fechas.

Hemos platicado con el contacto de la institución y nos ha dicho que sería mejor no tomar en cuante esta información. A continuación, haremos una función para eliminar los outliers que estén por debajo de este parámetro.

```
#Se define una función para eliminar los registros con valores en la
columna de debited menores a 5,000,000
def remove_min_noise(df, column):
  df_temp = df.copy()
  for row in df.index:
    if df_temp[column][row] <= 5000000:
      #df_min_result.append(df.loc[row])
      df_temp.drop(row, axis=0, inplace=True)
  return df_temp

#se procesa el cambio
df_noiseless = remove_min_noise(df_analisis,"debited")
```

Ahora haremos el mismo procedimiento para los outliers mayores a "14,000,000" para analizar la naturaleza de este comportamiento.

```
df_max_noise = df_analisis[df_analisis["debited"] >= 14000000]
df_max_noise
```


```
{"summary":"{\n  \"name\": \"df_max_noise\",\n  \"rows\": 28,\n
\"fields\": [\n    {\n      \"column\": \"day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 2,\n        \"max\": 31,\n        \"samples\": [\
n          4,\n          30,\n          2\n        ],\n
\"num_unique_values\": 11,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mont\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3,\n        \"min\": 1,\n        \"max\": 12,\n
\"samples\": [\n          8,\n          12,\n          2\n        ],\n
\"num_unique_values\": 10,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 2020,\n        \"max\": 2023,\n
\"samples\": [\n          2021,\n          2023,\n          2020\n
],\n        \"num_unique_values\": 4,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"credited\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3481608.3380878717,\n        \"min\":
17464869.34,\n        \"max\": 34564319.43,\n        \"samples\": [\n
22082296.02,\n          19760311.75,\n          19501478.36\
n        ],\n        \"num_unique_values\": 28,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"debited\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2277478.573339203,\n        \"min\": 14138085.3,\n        \"max\":
25695066.21,\n        \"samples\": [\n          16704190.05,\n
15917245.58,\n          14561190.75\n        ],\n
\"num_unique_values\": 28,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"type\",\n      \"properties\": {\n        \"dtype\": \"boolean\",\n
\"samples\": [\n          false,\n          true\n        ],\n
\"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"week_day\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"samples\": [\n          4,\n          1\n
],\n        \"num_unique_values\": 5,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"perc_remain\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.03354386171720702,\n
\"min\": 0.15807757,\n        \"max\": 0.285070262,\n
\"samples\": [\n          0.243548314,\n          0.194484086\n
],\n        \"num_unique_values\": 28,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_max_noise"}
```


Esta información corresponde a fechas de días festivos, o bien, fechas aledañas a días festivos. Por ejemplo, el 24 de Diciembre que no es propiamente un día festivo, sin embargo el 25 lo es. Esto hace que los movimientos del día 24 sean mayores.

Este es un comprortamiento esperado y natural del negocio, por tanto, sería un error eliminar este información. Sin embargo, existe un movimiento atípico en esta información:

```
df_atipico = df_analisis[df_analisis["debited"] >= 20000000]
df_atipico
```


```
{"repr_error":"cannot convert float NaN to
integer","type":"dataframe","variable_name":"df_atipico"}
```


En los tres años de información que se tiene, no se rebasa el límite de 13 millones por día en movimientos; pero en este día se movieron 25 millones.

Cuando vemos que la institución se preparó para este movimiento, acreditando un 25% más de los 25 millones ese día. Podemos inferir que esta información no es erronea y que la institución estaba prevenida para esto, sin embargo, es una situación especial.

Después de comentarlo con nuestro contacto de la institución, llegamos al acuerdo de que podemos tomar el mismo día del año anterior y del año posterior y haer un promedio para darle un valor más cercano al esperado ese día, y de esta forma reducir el ruido para nuestro modelo.

```
#se obtienen las cantidades de los días en cuestión
df_diaAnterior_tipico = df_analisis[df_analisis["year"] == 2022]
df_diaAnterior_tipico =
df_diaAnterior_tipico[df_diaAnterior_tipico["mont"] == 5]
df_diaAnterior_tipico =
df_diaAnterior_tipico[df_diaAnterior_tipico["day"] == 31]
df_diaAnterior_tipico
```

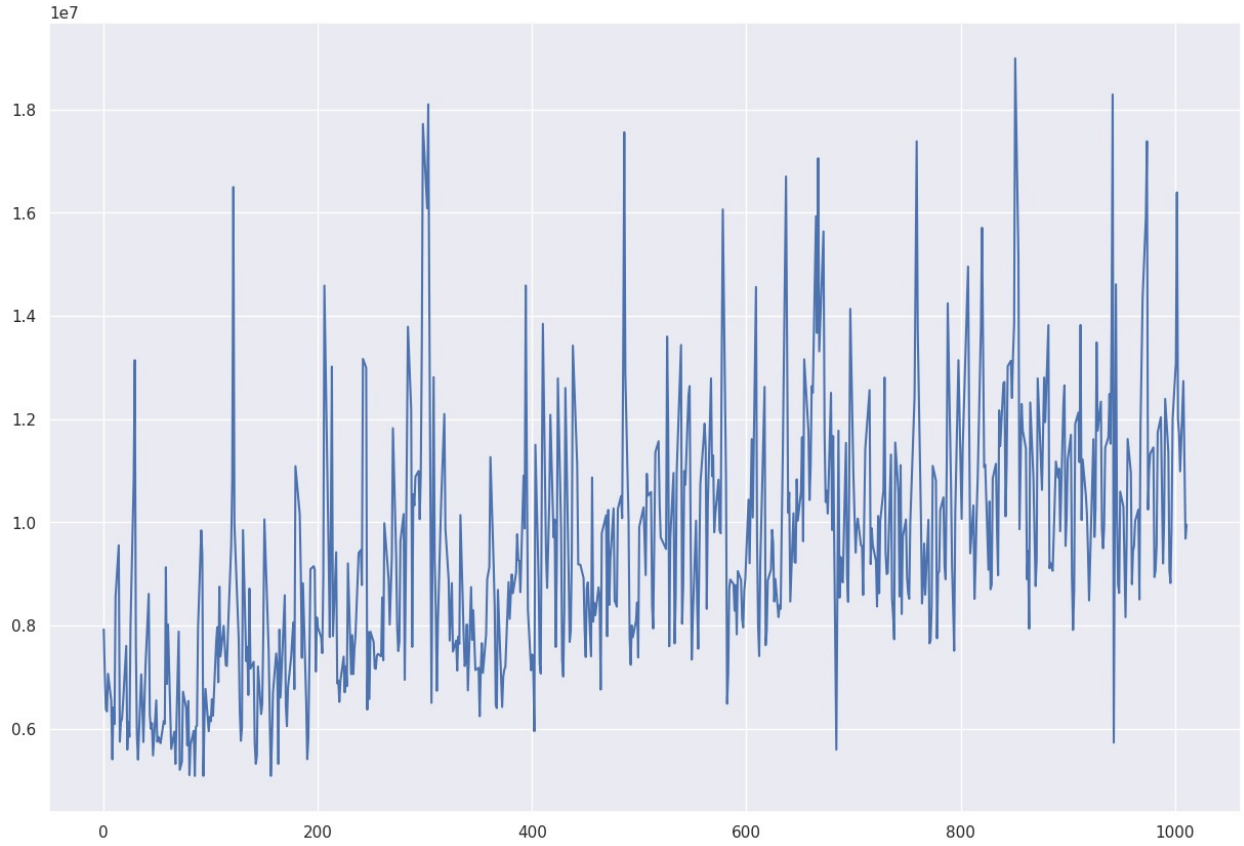
```
{"repr_error":"cannot convert float NaN to
integer","type":"dataframe","variable_name":"df_diaAnterior_tipico"}

df_diaPosterior_tipico = df_analisis[df_analisis["mont"] == 10]
df_diaPosterior_tipico =
df_diaPosterior_tipico[df_diaPosterior_tipico["year"] == 2022]
df_diaPosterior_tipico =
df_diaPosterior_tipico[df_diaPosterior_tipico["day"] == 31]
df_diaPosterior_tipico

{"repr_error":"cannot convert float NaN to
integer","type":"dataframe","variable_name":"df_diaPosterior_tipico"}
```

Ahora, con las cantidades de 12,805,472.2 y 13,821,696.96 podemos hacer el promedio para nuestro valor típico.

```
#Se define la función que cambiará el valor en la fecha designada
def correct_value_onday(day,mont,year,value,df):
  for row in df.index:
    if df["year"][row] == year:
      if df["mont"][row] == mont:
        if df["day"][row] == day:
          df["debited"][row] = value
          break
  return df

#se procesa el cambio
df_noiseless = correct_value_onday(31,3,2022,((12805472.2 +
13821696.96)/2), df_noiseless)

<ipython-input-201-f8dab5b5aa87>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df["debited"][row] = value

#Checamos el resultado
df_atipico = df_noiseless[df_noiseless["day"] == 31]
df_atipico = df_atipico[df_atipico["mont"] == 3]
df_atipico = df_atipico[df_atipico["year"] == 2022]
df_atipico

{"repr_error":"cannot convert float NaN to
integer","type":"dataframe","variable_name":"df_atipico"}
```

Ahora, el valor atípico ha sido cambiado. Vamos a visualizar como ha terminado nuestra gráfica

```
plt.plot(df_noiseless["debited"])
plt.show()
```



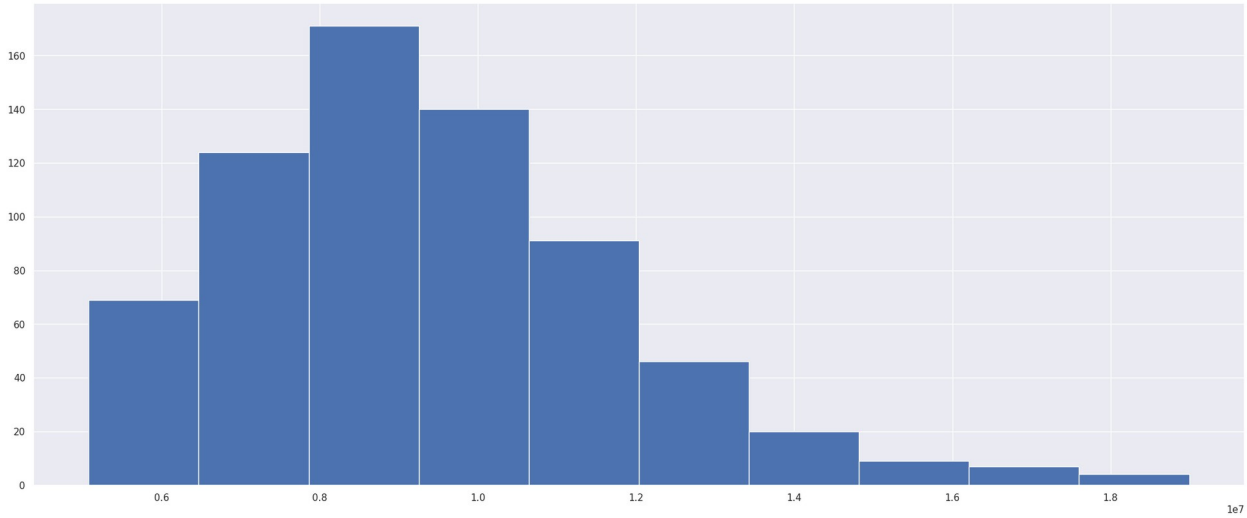
Podemos observar una tendencia mucho más clara que la original.

Transformación

Ahora veamos la distribución de la columna "debited".

```
#se crea un copia para trabajar la transformación
df_transformado = df_noiseless.copy()

df_transformado["debited"].hist( bins=10 ,figsize=(25,10))
```

```
<Axes: >
```



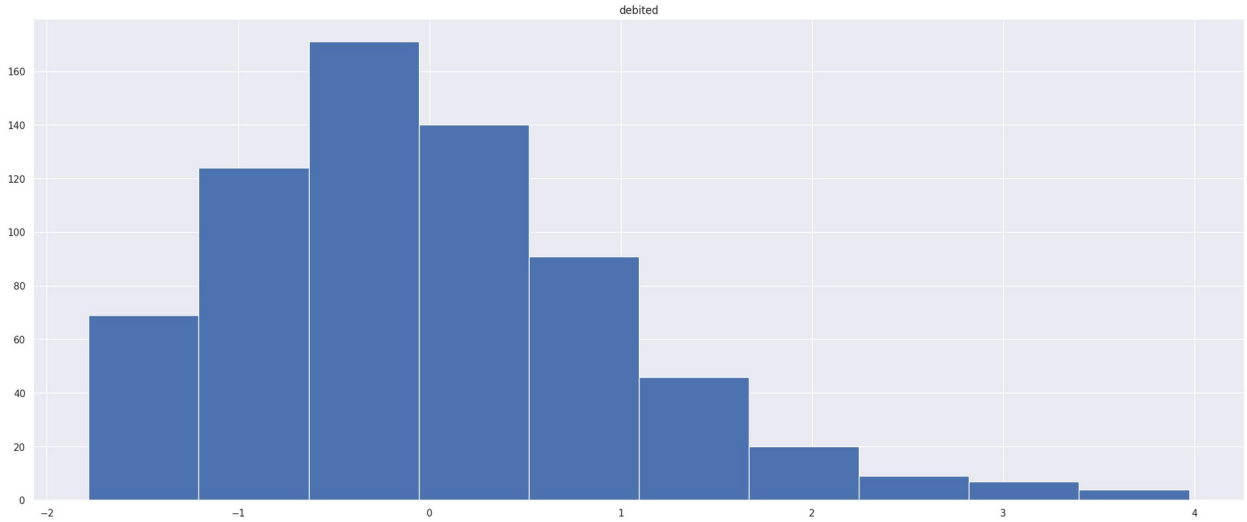
La distribución no está tan mal, tiene una buena forma. Sin embargo, vamos a invbestigar si algún tranformador puede mejorar la distribución normal de estos valores.

```
#Se define las columnas que serán tratadas
columns_to_scale = ['debited']
```

Así que intentaremos con Z-score (Standard Scaler)

```
#Se crea el objeto transformador
z_score_scaler = StandardScaler()
#Se aplica la operación de Z-score
df_transformado[columns_to_scale] =
z_score_scaler.fit_transform(df_transformado[columns_to_scale])
#Y verificaremos que la información tenga una mejor distribución
df_transformado[columns_to_scale].hist( bins=10 ,figsize=(25,10))
```

```
array([[<Axes: title={'center': 'debited'}>]], dtype=object)
```

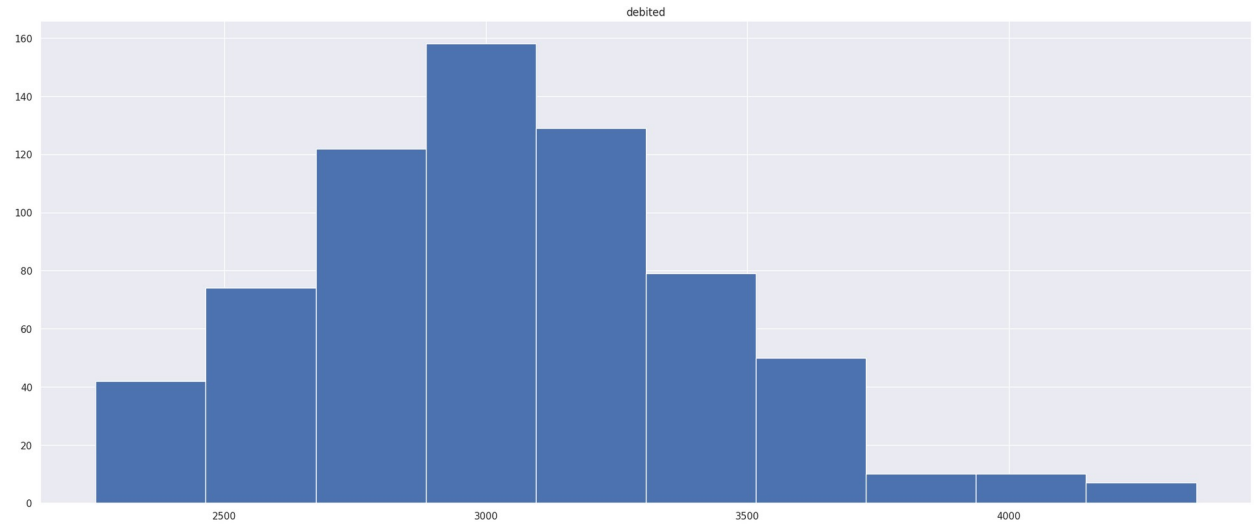


StandardScaler tiene muy poco efecto en la distrtibución, aun que movió los valores en el eje X para que se adapte mejor a la distribución normal.

Ahora intenrtaremos Raíz cuadrada.

```
#se crea un copia para trabajar la transformación
df_transformado = df_noiseless.copy()
# Transformación de raíz cuadrada para las columnas seleccionadas
df_transformado[columns_to_scale] =
np.sqrt(df_transformado[columns_to_scale])
#Ahora, revisaremos el resultado de la operación.
#Y verificaremos que la información tenga una mejor distribución
df_transformado[columns_to_scale].hist( bins=10 ,figsize=(25,10))
```

```
array([[<Axes: title={'center': 'debited'}>]], dtype=object)
```

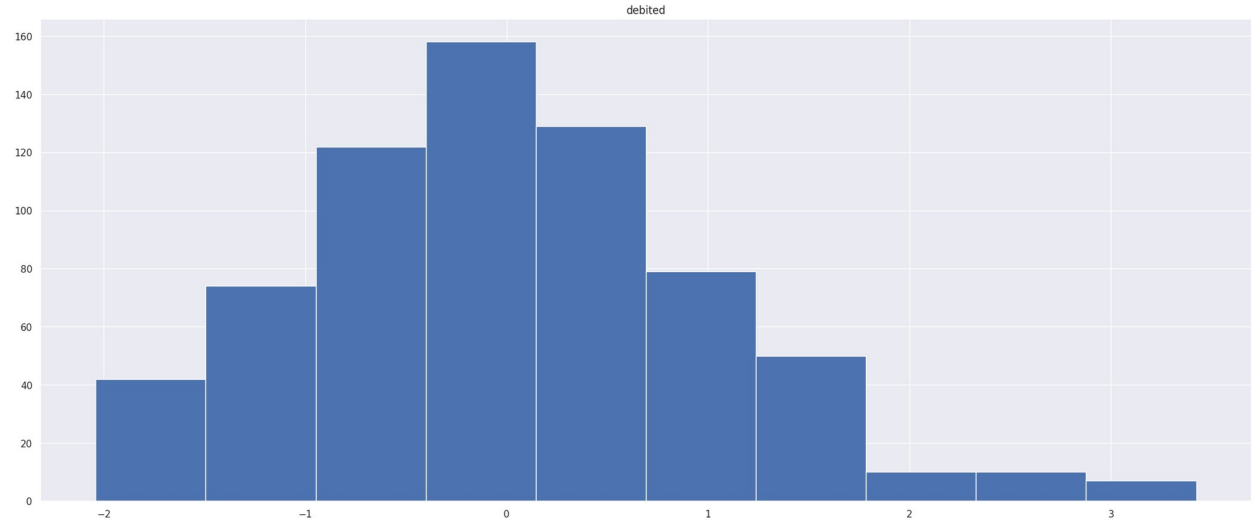


Aunque los valores se movieron en el eje X, podemos obserbar que la distribución de los datos ha mejorado.

Pasaremos ahora por el StandardScaler para ver si podemos mejorar la posición en el eje X de los valores actuales

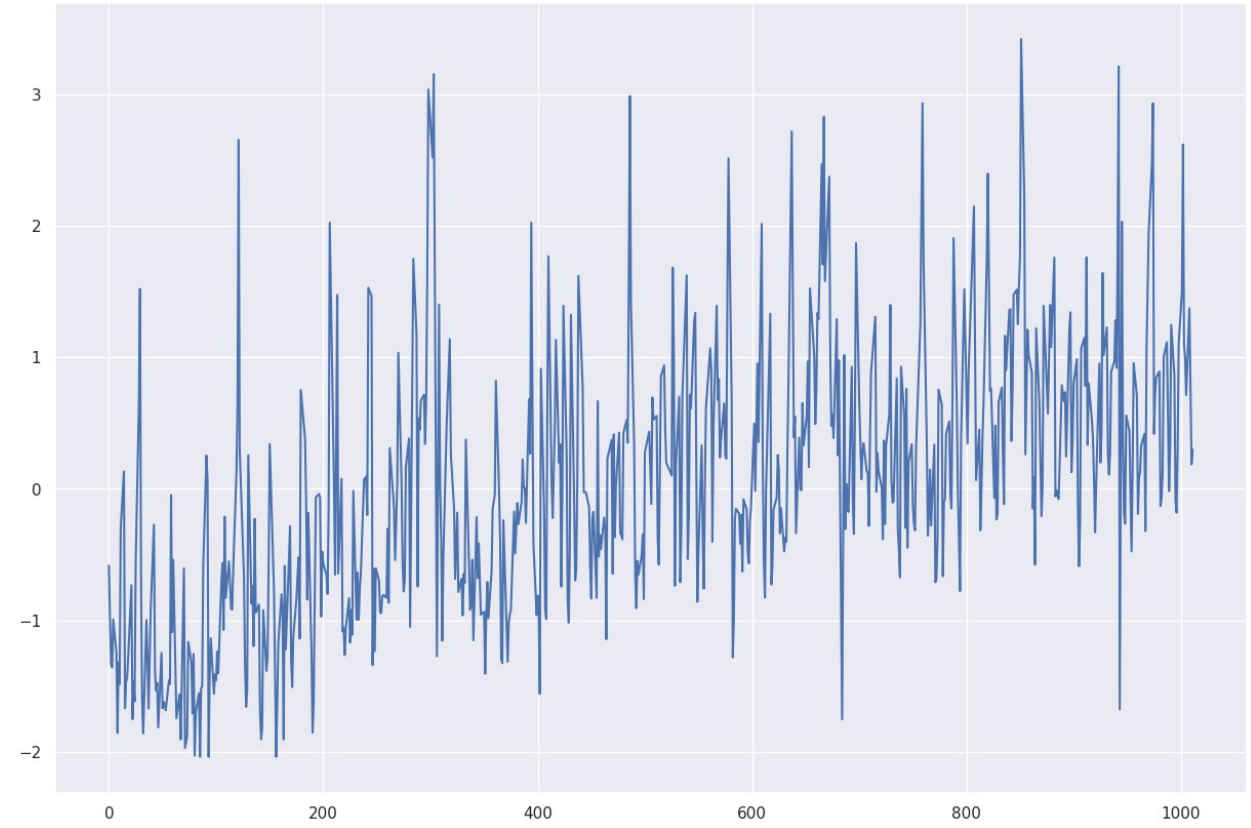
```
#Se aplica la operación de Z-score
df_transformado[columns_to_scale] =
z_score_scaler.fit_transform(df_transformado[columns_to_scale])
#Y verificaremos que la información tenga una mejor distribución
df_transformado[columns_to_scale].hist( bins=10 ,figsize=(25,10))
```

```
array([[<Axes: title={'center': 'debited'}>]], dtype=object)
```



Y nos quedaremos con el resultado de esta técnica combinada para los valores procesados.

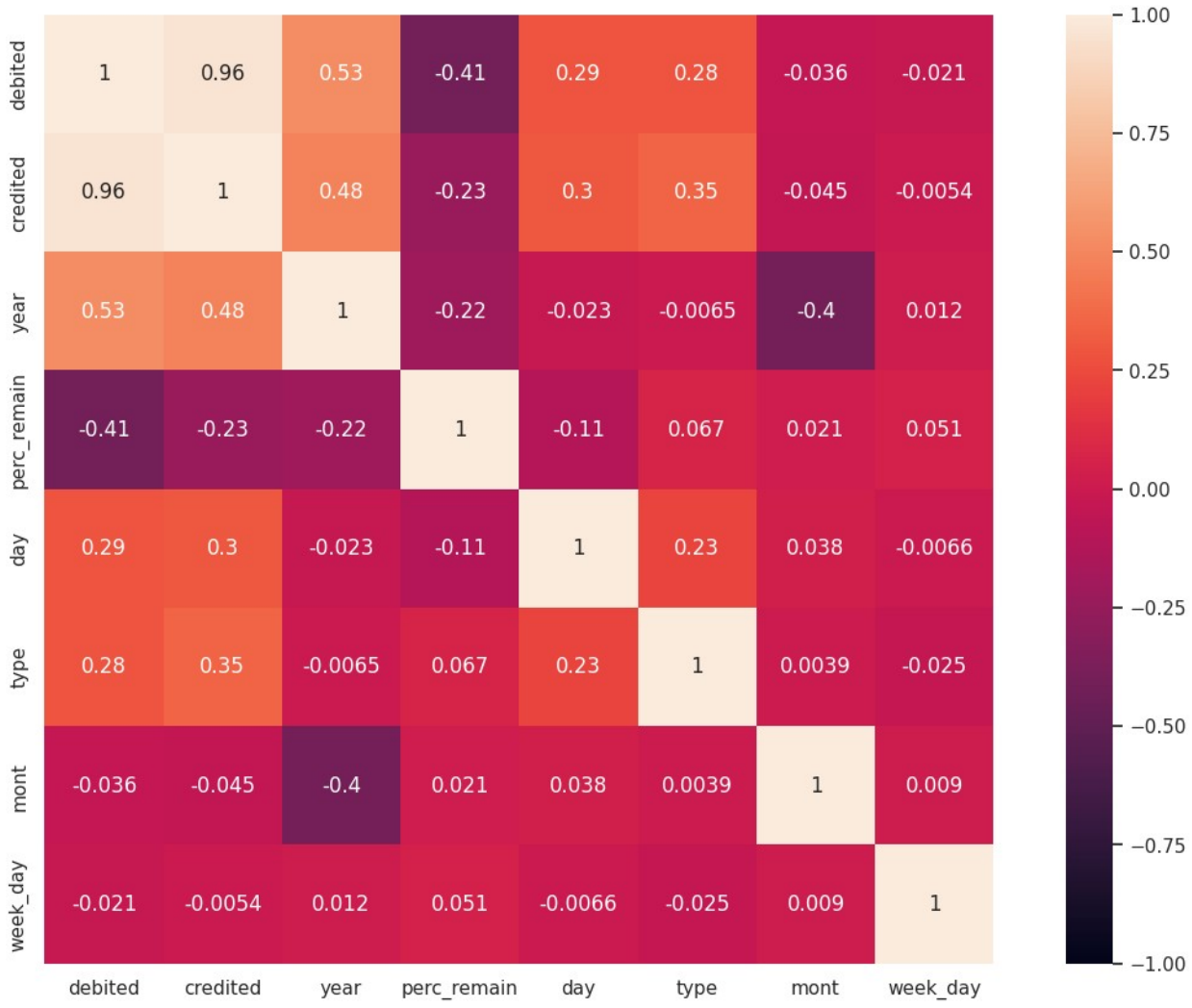
```
plt.plot(df_transformado["debited"])
plt.show()
```



Ahora, vamos a checar los valores de correlación que existen entre las columnas

```
#Vamos a ver la correlación con la nueva información
#Se obtiene la matriz de correlación del Dataframe con el que se está trabajando
#(Se ha organizado conforme a la correlación con la columna debited)
MatrizDeCorrelación = df_transformado[['debited','credited','year','perc_remain','day','type','mont','week_day']].corr()

sns.heatmap( MatrizDeCorrelación, vmin = -1, vmax = 1, square=True, annot = True)
```

```
<Axes: >
```



Ya con la información de manera visual, podemos avanzar a la eliminación de características que se consideran que no deben/pueden ser admitidas como variables de entrada.

Características que se generaron en el futuro; las siguientes características se generaron cuando el día se terminó y por tanto, no pueden ser una entrada en nuestro modelo (se supone que será consultado al principio del día).

- **Credited**.- Se refiere al monto que se depositó durante el día.
- **perc_remain**.- Se refiere al porcentaje del monto que no se utilizó durante el día.

```
#Se crea una copia del Dataframe para seguir trabajando
df_limpio = df_transformado.copy()

df_limpio.drop(['credited','perc_remain'], axis=1, inplace=True)
```

Modelos

En esta sección se analizarán los modelos propuestos para lograr predecir la tendencia de nuesto set de datos.

```
#Se hace una copia del dataframe para trabajar con los modelos
df_modelos = df_limpio.copy()
df_modelos.shape
```

```
(681, 6)
```

Primero crearemos los set de datos para X y Y, y en seguida se separarán los datos en grupos para entrenamiento,prueba y evaluación (train,test y val)

```
#Dividiremos en X y Y
X = df_modelos[['day', 'mont', 'year', 'type', 'week_day']].copy()
Y = df_modelos['debited'].copy()

#Ahora separaremos en los grupos antes mencionados
X_train, X_test, y_train, y_test = train_test_split( X , Y ,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42)

#Por último, checaremos las proporciones
print("Tamaño de X_train: ", str(len(X_train)), " - ",
round((len(X_train)*100)/len(df_modelos),2),"%")
print("Tamaño de X_test: "+ str(len(X_test)), " - ",
round((len(X_test)*100)/len(df_modelos),2),"%")
print("Tamaño de X_val: "+ str(len(X_val)), " - ",
round((len(X_val)*100)/len(df_modelos),2),"%")
print("**********************************")
print("Tamaño de y_train: "+ str(len(y_train)))
print("Tamaño de y_test: "+ str(len(y_test)))
print("Tamaño de y_val: "+ str(len(y_val)))
```

```
Tamaño de X_train:  408  -  59.91 %
Tamaño de X_test: 137  -  20.12 %
Tamaño de X_val: 136  -  19.97 %
**********************************
Tamaño de y_train: 408
Tamaño de y_test: 137
Tamaño de y_val: 136
```

Los Modelos a utlizar

Ahora, creamos una función que contenga los modelos con los que vamos a probar. Lo haremos de esta forma en caso de que más adelante se decida agregar más modelos a este proceso.

El modelo que se utilizará es el de Regresion logística, Lasso, Ridge y Elastic Net, todos los modelos estarán con sus características por defecto.

```
def get_models():
  modelos = list()
  nombres = list()
  alpha_value = .0001

  # LR - Regresión Logística:
  modelos.append(LinearRegression())
  nombres.append('LR')

  modelos.append((Lasso(alpha=alpha_value)))
  nombres.append('LASSO')

  modelos.append((Ridge(alpha=alpha_value)))
  nombres.append('RIDGE')

  modelos.append((ElasticNet(alpha=alpha_value, l1_ratio=.0005)))
  nombres.append('EN')

  return modelos, nombres

modelos, nombres = get_models()  # cargamos los modelos a comparar
resultados = list() #Creamos la liste para los resultados
```

Primero crearemos un pipeline para la transformación de las columnas numéricas. Los pasos serán:

- un imputador
- un escalador

Para el imputador utilizaremos "la mediana", mientras que el escalador será in Minmax en el rango de 1 a 2.

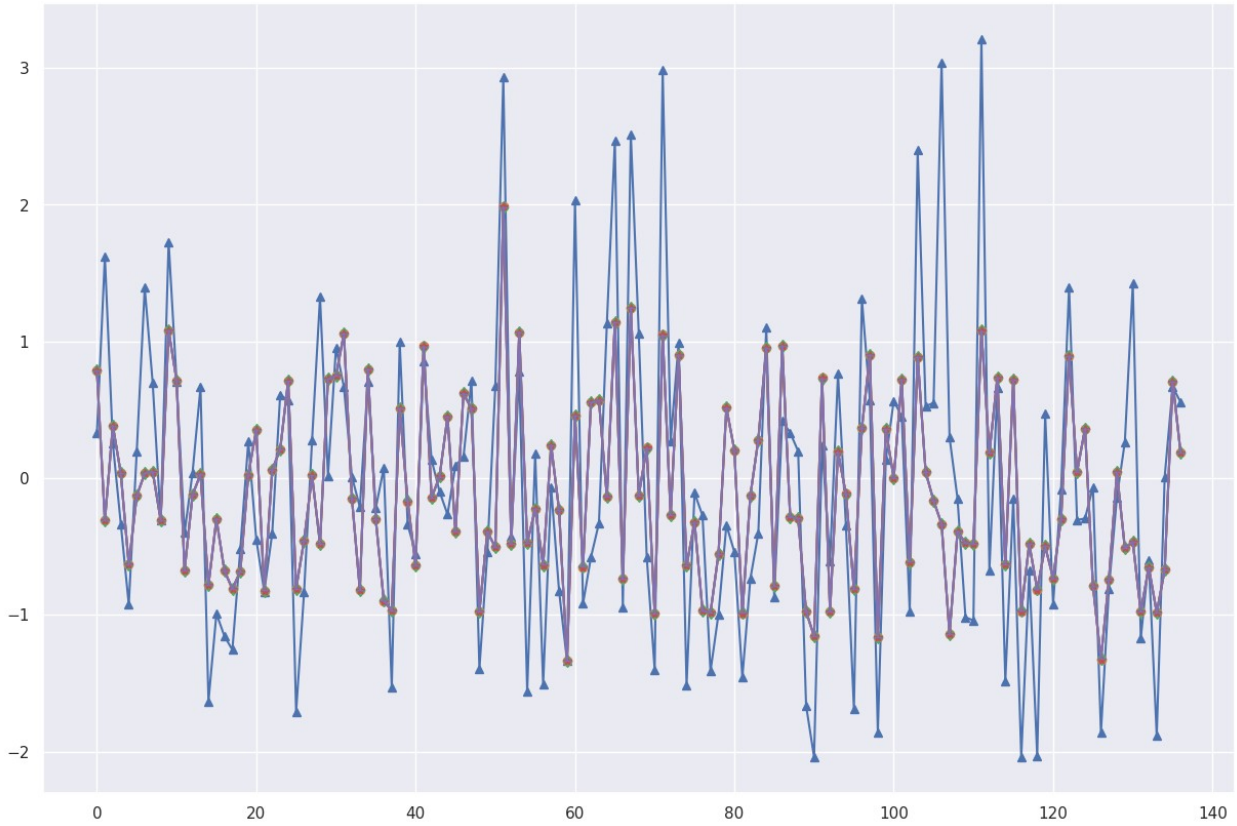
Después, declararemos un transformador de columnas con nuestro pipeline. Las columnas que se pasarán a travéz de este paso serán las declaradas como "columns_to_process"; el resto de las columnas pasarán sin cambios.

```
#Columnas a tratar
columns_to_process = ['day', 'mont', 'year', 'week_day']

# Transformaciones a factores numéricos de entrada:
num_pipeline = Pipeline(steps = [('impMediana',
SimpleImputer(strategy='median')),
                                 ('escalaNum',
MinMaxScaler(feature_range=(1,2)))])

# Conjuntamos las transformaciones numéricas y categóricas que se
estarán aplicando a los datos de entrada:
columnasTransformer = ColumnTransformer(transformers = [('numpipe',
num_pipeline, columns_to_process)],
                                        remainder='passthrough')

y_pred = []
for i in range(len(modelos)):

  pipeline = Pipeline(steps=[('ct',columnasTransformer),
('m',modelos[i])])

  pipeline.fit(X_train, y_train)
  scores = pipeline.score(X_test,y_test)

  y_pred.append(pipeline.predict(X_test))


  resultados.append(scores)
  print('>> %s: %.3f ' % (nombres[i], np.mean(scores)))
```

```
>> LR: 0.501
>> LASSO: 0.500
>> RIDGE: 0.501
>> EN: 0.500
```

```
x1 = np.arange(0,len(y_test))

plt.plot(x1, np.ravel(y_test), marker='^', label='Test')
plt.plot(x1, np.ravel(y_pred[0]), marker='o', label='LR')
plt.plot(x1, np.ravel(y_pred[1]), marker='d', label='LASSO')
plt.plot(x1, np.ravel(y_pred[2]), marker='*', label='RIDGE')
plt.plot(x1, np.ravel(y_pred[3]), marker='_', label='EN')
```

```
[<matplotlib.lines.Line2D at 0x7bbc98e64070>]
```



Vemos una mejora en el rendimiento de los modelos solo con mejorar el preprocesamiento de los datos de entrada. A continuación podemos comparar los resultados:

- Línear Regression - de 37.3% a 50.1%
- Lasso - de 38.4% a 50%
- Ridge - de 37.1% a 50.1%
- Elastic net - de 14.2 a 50%

Modelos Alternos

El primer modelo que trataremos de utilizar será el de la empresa Meta, llamado Prophet. Así que procedemos a instalarlo

```
!!pip install -q prophet
```

```
[]
```

```
from prophet import Prophet
```

Para este modelo se necesita un Dataframe especial que contenga la fecha en formato YYYY-MM-DD en una columna llamada "ds" y otra con el valor a predecir, llamada "y". Así que nuestro primer paso será crear este Dataframe.

```
#Partiremos del dataframe Limpio
df_prophet = df_limpio.copy()
df_prophet.columns
```

```
Index(['day', 'mont', 'year', 'debited', 'type', 'week_day'],
dtype='object')
```

```
#se define un función que crea esta nueva columna (ds) y le da los
valores correspondientes
def create_Y_column(df):
  df.insert(6,"ds",True)
  for row in df.index:
    df['ds'][ row ] = str(df['year'][row]) +"-"+ str(df['mont'][row])
+"-"+ str(df['day'][row])
  return df

#Se procesa el cambio
df_prophet = create_Y_column(df_prophet)
```

```
<ipython-input-228-3826ec84c6b8>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['ds'][ row ] = str(df['year'][row]) +"-"+ str(df['mont'][row])
+"-"+ str(df['day'][row])
```

```
#Hacemos le renombrado de la columna debited
df_prophet = df_prophet.rename(columns={'debited':'y'})

df_prophet.drop(['day','mont','year','type','week_day'], axis=1,
inplace=True)

#Se Visualiza el Cambio
df_prophet.columns
```

```
Index(['y', 'ds'], dtype='object')
```

```
df_prophet.head()
```


```
{"summary":"{\n  \"name\": \"df_prophet\",\n  \"rows\": 681,\n
\"fields\": [\n    {\n      \"column\": \"y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.000735023987516,\n        \"min\": -2.0399544456064853,\n
\"max\": 3.422588160405378,\n        \"samples\": [\n
0.3271713067749213,\n          1.6204413294164712,\n
0.252859163163816\n        ],\n        \"num_unique_values\": 681,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ds\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"min\": \"2020-10-1\",\n
\"max\": \"2023-3-9\",\n        \"samples\": [\n          \"2023-1-
```

```
20\",\n          \"2021-8-13\",\n          \"2020-8-31\"\n        ],\n
\"num_unique_values\": 681,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_prophet"}
```


Se comienza con la implementación del modelo

```
m = Prophet()
m.fit(df_prophet)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp66kgqqdq/nxs5wwx6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp66kgqqdq/_ss0i9vx.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=89776', 'data', 'file=/tmp/tmp66kgqqdq/nxs5wwx6.json',
'init=/tmp/tmp66kgqqdq/_ss0i9vx.json', 'output',
'file=/tmp/tmp66kgqqdq/prophet_model_huvk4gd/prophet_model-
20240219045202.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
04:52:02 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
04:52:03 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
<prophet.forecaster.Prophet at 0x7bbc98fcf5e0>
```

```
#Se crea la predicción
future = m.make_future_dataframe(periods=365)
forecast = m.predict(future)
```

```
forecast
```

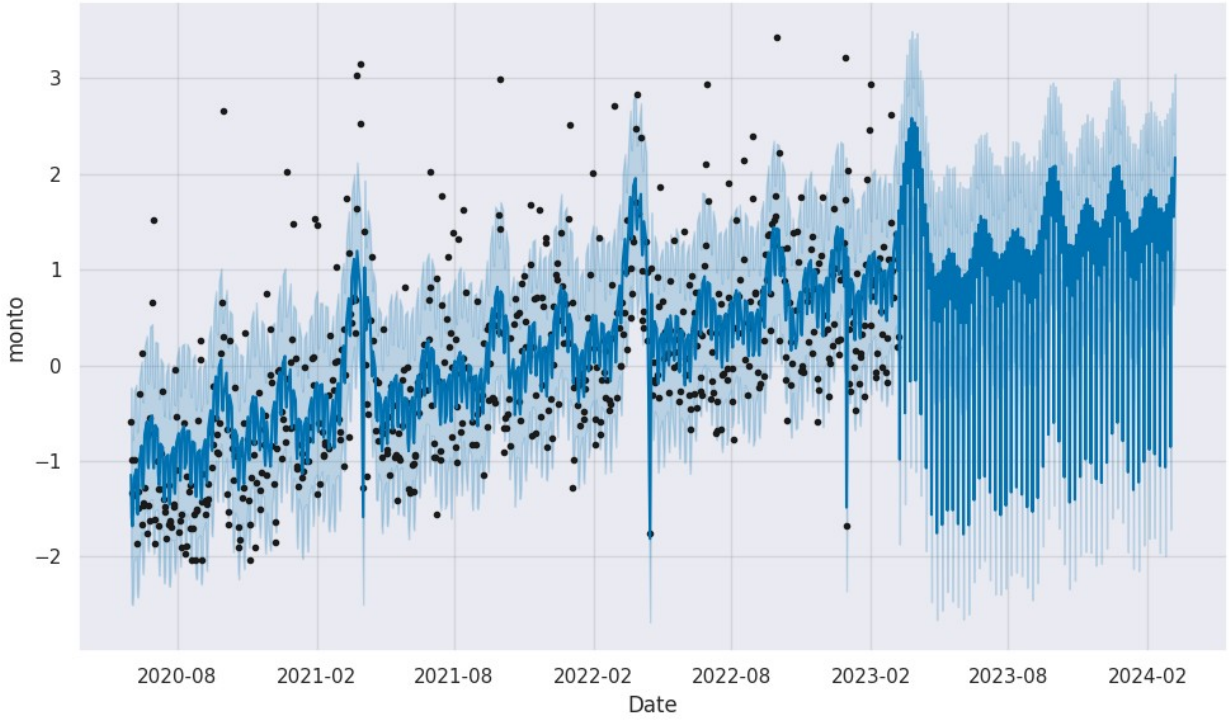

```
{"summary":"{\n  \"name\": \"forecast\",\n  \"rows\": 1046,\n
\"fields\": [\n    {\n      \"column\": \"ds\",\n      \"properties\":
{\n        \"dtype\": \"date\",\n        \"min\": \"2020-06-01
00:00:00\",\n        \"max\": \"2024-03-08 00:00:00\",\n
\"samples\": [\n          \"2023-06-10 00:00:00\",\n          \"2021-
07-22 00:00:00\",\n          \"2022-12-27 00:00:00\"\n        ],\n
\"num_unique_values\": 1046,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trend\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.7580270054542363,\n        \"min\": -1.326092050513545,\n
\"max\": 1.262041412020019,\n        \"samples\": [\n
0.7865790991694135,\n          -0.4364742162505427,\n
0.49815526968283314\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"yhat_lower\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9220323996591572,\n        \"min\": -2.684641479804703,\n
\"max\": 1.7604214188656486,\n        \"samples\": [\n          -
2.6012297326607574,\n          -1.4451946851483073,\n
0.05842610883538541\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yhat_upper\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9265479561051089,\n        \"min\": -0.9230083015809686,\n
\"max\": 3.491291862813789,\n        \"samples\": [\n          -
0.7093836187757223,\n          0.36696471487424803,\n
1.822015143787552\n        ],\n        \"num_unique_values\": 1046,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"trend_lower\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7553422063160777,\n        \"min\": -1.326092050513545,\n
\"max\": 1.2460563726254874,\n        \"samples\": [\n
0.7848801957441066,\n          -0.4364742162505427,\n
0.49815526968283314\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trend_upper\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7607618261224751,\n        \"min\": -1.326092050513545,\n
\"max\": 1.2783926529976275,\n        \"samples\": [\n
0.7884131942693781,\n          -0.4364742162505427,\n
0.49815526968283314\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"additive_terms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.6779214964000583,\n        \"min\": -2.5407204806982815,\n
\"max\": 1.9360731319413969,\n        \"samples\": [\n          -
2.4571456730362877,\n          -0.09041544951238661,\n
0.4488867998351665\n        ],\n        \"num_unique_values\": 1046,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"additive_terms_lower\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.6779214964000583,\n        \"min\": -2.5407204806982815,\n
\"max\": 1.9360731319413969,\n        \"samples\": [\n          -
2.4571456730362877,\n          -0.09041544951238661,\n
0.4488867998351665\n        ],\n        \"num_unique_values\": 1046,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"additive_terms_upper\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.6779214964000583,\n        \"min\": -2.5407204806982815,\n
\"max\": 1.9360731319413969,\n        \"samples\": [\n          -
2.4571456730362877,\n          -0.09041544951238661,\n
0.4488867998351665\n        ],\n        \"num_unique_values\": 1046,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

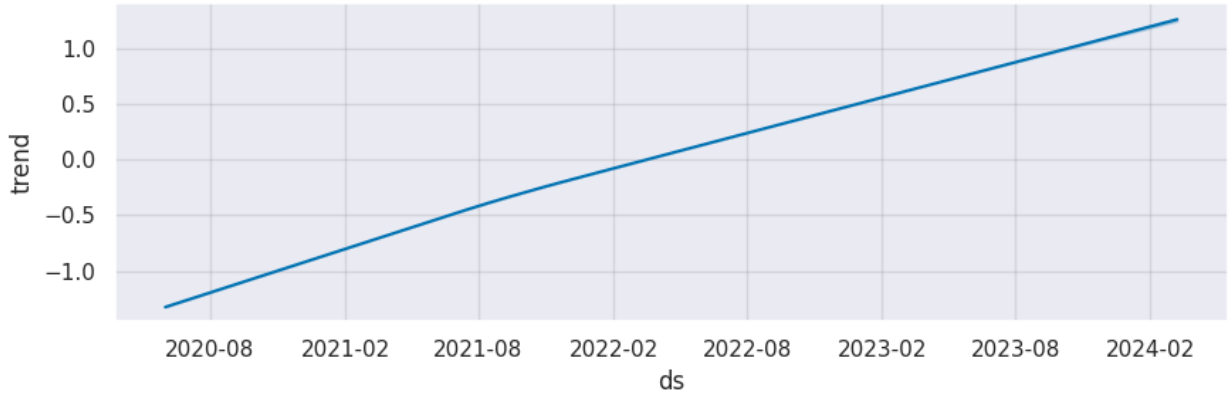
```
n    },\n    {\n      \"column\": \"weekly\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.5811325791799932,\n        \"min\": -2.0895139582822777,\n
\"max\": 0.6217823491975928,\n        \"samples\": [\n          -
2.089513958281255,\n          0.2429804857835866,\n
0.17719142761085005\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"weekly_lower\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5811325791799932,\n        \"min\": -2.0895139582822777,\n
\"max\": 0.6217823491975928,\n        \"samples\": [\n          -
2.089513958281255,\n          0.2429804857835866,\n
0.17719142761085005\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"weekly_upper\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5811325791799932,\n        \"min\": -2.0895139582822777,\n
\"max\": 0.6217823491975928,\n        \"samples\": [\n          -
2.089513958281255,\n          0.2429804857835866,\n
0.17719142761085005\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yearly\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.36826376772148406,\n        \"min\": -0.45339717557355835,\n
\"max\": 1.3142907827466639,\n        \"samples\": [\n          -
0.3676317147550328,\n          -0.3333959352959732,\n
0.26769537222431644\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yearly_lower\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.36826376772148406,\n        \"min\": -0.45339717557355835,\n
\"max\": 1.3142907827466639,\n        \"samples\": [\n          -
0.3676317147550328,\n          -0.3333959352959732,\n
0.26769537222431644\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"yearly_upper\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.36826376772148406,\n        \"min\": -0.45339717557355835,\n
\"max\": 1.3142907827466639,\n        \"samples\": [\n          -
0.3676317147550328,\n          -0.3333959352959732,\n
0.26769537222431644\n        ],\n        \"num_unique_values\": 1046,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"multiplicative_terms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"multiplicative_terms_lower\",\n      \"properties\": {\n
```

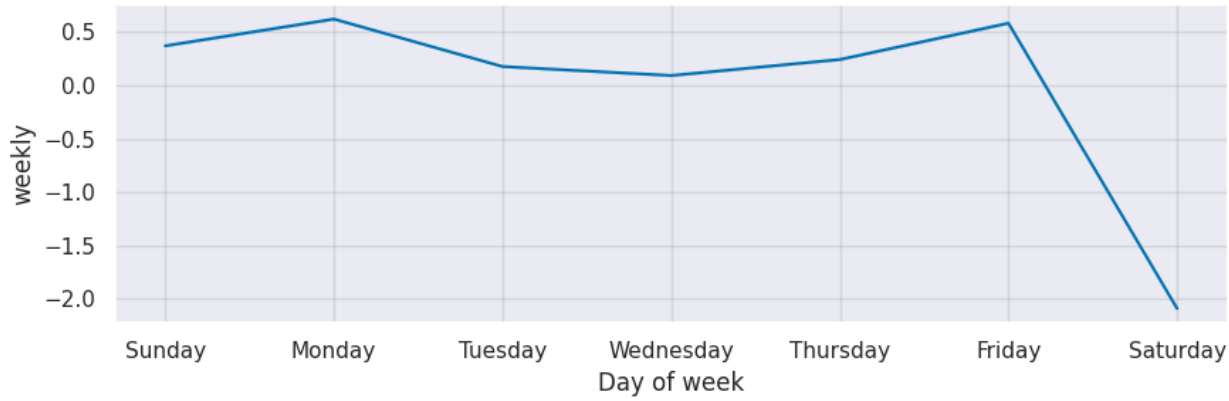
\"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n
\"max\": 0.0,\n   \"samples\": [\n   0.0\n   ],\n
\"num_unique_values\": 1,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"multiplicative_terms_upper\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n
\"max\": 0.0,\n   \"samples\": [\n   0.0\n   ],\n
\"num_unique_values\": 1,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"yhat\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.922715824116922,\n   \"min\": -1.8137603418275778,\n
\"max\": 2.583389197675525,\n   \"samples\": [\n   -
1.6705665738668742\n   ],\n   \"num_unique_values\": 1046,\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"forecast"}

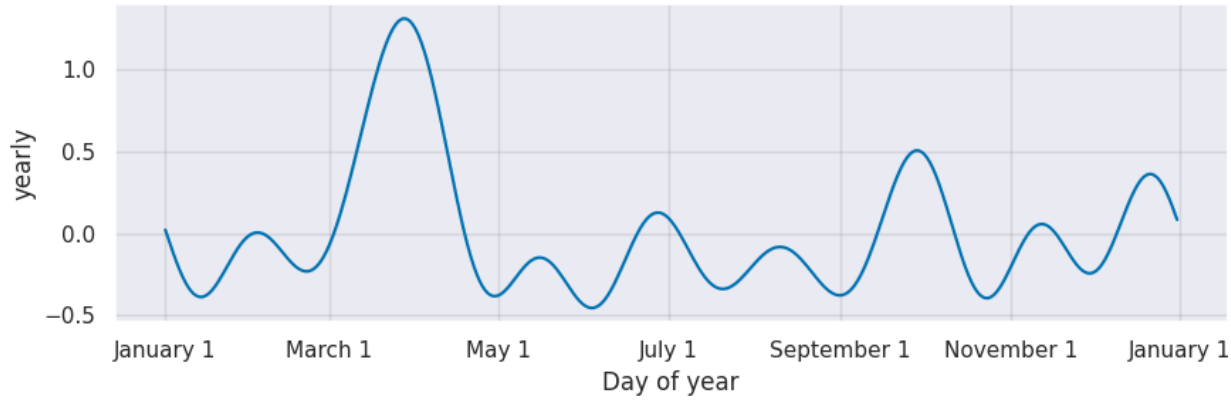
```
figure = m.plot(forecast, xlabel='Date', ylabel='monto')
```




```
figure3 = m.plot_components(forecast)
```


trend
1.0
0.5
0.0
−0.5
−1.0
2020-08
2021-02
2021-08
2022-02
2022-08
2023-02
2023-08
2024-02
ds


weekly
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week


yearly
1.0
0.5
0.0
−0.5
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year