

PROYECCIÓN DE VENTAS DE LOCALES



Versión 4.0

Autor:
Antonio Elvira García

CONTEXTO DEL PROYECTO



FG Burgers es una multinacional de comida rápida con origen en USA con más de 3000 locales distribuidos por todo el mundo, entre los países en los que tiene operación se encuentran UK, Francia, Alemania y España donde la marca cuenta con 22 locales en operación en 5 Comunidades Autónomas distintas (Comunidad de Madrid, Cataluña, Comunidad Valenciana, Andalucía y País Vasco).

En España la marca opera desde el año 2015.

El sector de la comida rápida es un sector muy competitivo solo en España existen más de 3000 locales de las diferentes marcas.

Si bien el sector de la comida rápida tiene generalmente unos altos ratios de facturación por local, es un sector donde la rentabilidad es bastante ajustada debido a los altos costos de inversión y de operación de los locales y se exige unos altos grados de gestión de los costos para así conseguir los niveles de rentabilidad esperados de los locales.

Parte esencial de esa gestión y de la consecución de los objetivos de rentabilidad viene dada por la optimizada proyección de ventas de los locales ya que los principales costos como son la mano de obra y coste de materia prima se ajustan en función de la proyección de ventas realizada.

OBJETIVOS

La marca FG Burgers buscaba una herramienta de proyección de ventas para poder implantar en sus locales en España.

Por lo tanto el objetivo de la investigación es realizar un modelo de proyección de ventas para poder predecir las ventas a futuro de los locales de la marca y también entender el comportamiento de estos en función de sus distintas variables.

Dentro de los objetivos de la investigación está resolver las siguientes preguntas;

01

¿Podemos predecir las ventas de la compañía a futuro utilizando un modelo de Machine Learning?

02

¿Afecta la época del año a las ventas?
¿Hay patrones de estacionalidad?

03

¿Hay relación entre la venta y el periodo del mes?

04

¿Afecta a las ventas que el día sea laborable o que haya algún festivo?

En base a estas preguntas se podrá definir un modelo de predicción de ventas que ayude a la marca a ajustar sus proyecciones de la manera más exacta para adaptar mejor la operación a la demanda de los locales.



FUENTE DEL DATASET Y CRITERIOS DE SELECCIÓN

El dataset fue conformado con datos provenientes de distintas fuentes de información de la empresa como son el programa de facturación de ventas de los locales y la base de datos de los locales.

Se decidió tomar este dataset porque contiene distinta información relevante de los locales actualmente abiertos por la marca para ser usada a la hora de realizar proyecciones de ventas.

Existe un Dataset en formato .CSV el cual dispone de 47.974 filas y 8 columnas con los datos de ventas de 22 locales de la compañía desde el 01/01/2016 hasta el 31/03/2022.

DATASET VENTAS

	A	B	C	D	E	F	G	H
1	Local	Nombre_Loc	Fecha	Dia_Semana	Ventas	Clientes	Laborable	Festivo
2	1	Gran Via	31/03/2022	4	6856	476	1	0
3	2	Parquesur	31/03/2022	4	6449	413	1	0
4	3	Plaza Rio	31/03/2022	4	5599	385	1	0
5	4	La Gavia	31/03/2022	4	7110	468	1	0
6	5	La Maquinist	31/03/2022	4	6356	450	1	0
7	6	Granada	31/03/2022	4	6495	415	1	0
8	7	Plaza Mayor	31/03/2022	4	5436	368	1	0
9	8	Castellana	31/03/2022	4	6423	406	1	0
10	9	Sagrada Fam	31/03/2022	4	6973	437	1	0
11	10	Serrano	31/03/2022	4	6199	392	1	0
12	11	Diagonal Ma	31/03/2022	4	7546	500	1	0
13	12	Plaza Catalui	31/03/2022	4	6810	476	1	0

1 df_ventas								
	Local	Nombre_Local	Fecha	Dia_Semana	Ventas	Clientes	Laborable	Festivo
0	1	Gran Via	31/03/2022	4	8290	576	1	0
1	2	Parquesur	31/03/2022	4	6900	442	1	0
2	3	Plaza Rio	31/03/2022	4	5991	412	1	0
3	4	La Gavia	31/03/2022	4	7608	501	1	0
4	5	La Maquinista	31/03/2022	4	6547	464	1	0
...
47969	18	Diagonal	02/01/2016	5	12181	691	0	1
47970	19	Valencia	02/01/2016	5	11199	628	0	1
47971	20	Paseo de Gracia	02/01/2016	5	11608	663	0	1
47972	21	Xanadu	02/01/2016	5	12569	704	0	1
47973	22	La Vaguada	02/01/2016	5	11508	652	0	1

47974 rows × 8 columns

EXPLICACIÓN DE LOS DATOS DEL DATASET

VENTAS

- 1.Local: Es el Id de cada local
- 2.Nombre_Local: Es el nombre interno de cada local
- 3.Fecha: La fecha en la que se realizó la venta
- 4.Dia_Semana: Número de día de la semana relacionado con la fecha;

- 1 = Lunes
- 2 = Martes
- 3 = Miércoles
- 4 = Jueves
- 5 = Viernes
- 6 = Sábado
- 7 = Domingo

5.Ventas: Ventas netas expresadas en euros realizadas por el local en ese día.

6.Clientes: Cantidad de clientes atendidos en el local en ese día.

7.Laborable: Indica si la fecha fue laborable;

- 1 = Laborable
- 0 = No laborable

Se consideran como no laborables a criterio de la marca los días festivos y los Sábados y Domingos.

1.Festivo: Indica si la fecha fue festivo.

- 1 = Festivo
- 0 = No Festivo

Los festivos tomados son los festivos nacionales de España

SELECCIÓN DEL ALGORITMO

DADOS LOS TIPOS DE DATOS Y EL PROBLEMA PRESENTADO SE DECIDIÓ ABORDAR EL PROBLEMA CON TRES ENFOQUES DISTINTOS Y UTILIZAR TRES TIPOS DE ALGORITMOS, DOS CON ENFOQUES MÁS TRADICIONALES Y UN TERCERO MÁS NOVEDOSO.

dmlc
XGBoost



PROPHET

OPCIÓN ALGORITMO

01

SE DECIDIÓ UTILIZAR XGBOOST POR SU
FACILIDAD DE USO, SU POCO
REQUERIMIENTO COMPUTACIONAL Y SU
BUEN PERFORMANCE ANTE PROBLEMAS
SIMILARES AL ABORDADO

dmlc
XGBoost

SOBRE XGBOOST

XGBoost es una biblioteca optimizada de aumento de gradiente distribuido diseñada para ser altamente eficiente, flexible y portátil. Implementa algoritmos de aprendizaje automático bajo el marco gradient Boosting. XGBoost proporciona un refuerzo de árbol paralelo (también conocido como GBDT, GBM) que resuelve muchos problemas de ciencia de datos de una manera rápida y precisa. El mismo código se ejecuta en los principales entornos distribuidos (Hadoop, SGE, MPI) y puede resolver problemas más allá de miles de millones de ejemplos.

HISTORIA

XGBoost comenzó inicialmente como un proyecto de investigación de Tianqi Chen como parte del grupo Distributed (Deep) Machine Learning Community (DMLC). Inicialmente, comenzó como una aplicación de terminal que se podía configurar mediante un archivo de configuración libsvm . Se hizo muy conocido en los círculos de competencia de ML después de su uso en la solución ganadora del Higgs Machine Learning Challenge. Poco después, se crearon los paquetes Python y R, y XGBoost ahora tiene implementaciones de paquetes para Java, Scala , Julia, Perl y otros lenguajes. Esto llevó la biblioteca a más desarrolladores y contribuyó a su popularidad entre la comunidad de Kaggle , donde se ha utilizado para una gran cantidad de competencias.

Pronto se integró con varios otros paquetes, lo que facilitó su uso en sus respectivas comunidades. Ahora se ha integrado con scikit-learn para usuarios de Python y con el paquete caret para usuarios de R. También se puede integrar en marcos de Data Flow como Apache Spark , Apache Hadoop y Apache Flink utilizando Rabbit y XGBoost4J abstractos. XGBoost también está disponible en OpenCL para FPGA . [13] Tianqi Chen y Carlos Guestrin han publicado una implementación eficiente y escalable de XGBoost.

CARACTERÍSTICAS DE XGBOOST

dmrc
XGBoost

Flexible

Soporta regresión, clasificación, ranking y objetivos definidos por el usuario.

Portable

Se ejecuta en Windows, Linux y OS X, así como en varias plataformas en la nube

Multi lenguaje

Admite múltiples lenguajes incluyendo C++, Python, R, Java, Scala, Julia.

A prueba de batallas

Ganador de muchos desafíos de ciencia de datos y aprendizaje automático. Utilizado en producción por múltiples empresas.

Distribuido en la nube

Admite capacitación distribuida en varias máquinas, incluidos clústeres de AWS, GCE, Azure e Yarn. Se puede integrar con Flink, Spark y otros sistemas de flujo de datos en la nube.

Rendimiento

El sistema backend bien optimizado para el mejor rendimiento con recursos limitados. La versión distribuida resuelve problemas más allá de miles de millones de ejemplos con el mismo código.

<https://xgboost.readthedocs.io/en/stable/>

OPCIÓN ALGORITMO

02

COMO SEGUNDA OPCIÓN SE OPTÓ POR UN ENFOQUE CLÁSICO CON RANDOM FOREST DEBIDO A SU FACILIDAD DE USO, SU ESTABILIDAD, POCAS PREPARACIÓN DE LOS DATOS Y MEJOR PERFORMANCE VS LOS ÁRBOLES DE DECISIONES SIMPLES



SOBRE RANDOM FOREST

Los bosques aleatorios o bosques de decisión aleatorios son un método de aprendizaje conjunto para la clasificación, regresión y otras tareas que opera mediante la construcción de una multitud de árboles de decisión en el momento del entrenamiento. Para las tareas de clasificación, el resultado del bosque aleatorio es la clase seleccionada por la mayoría de los árboles. Para las tareas de regresión, se devuelve la predicción media o promedio de los árboles individuales. Los bosques de decisión aleatorios corren el hábito de los árboles de decisión de sobreajustarse a su conjunto de entrenamiento. Los bosques aleatorios generalmente superan a los árboles de decisión, pero su precisión es menor que la de los árboles impulsados por gradiente. Sin embargo, las características de los datos pueden afectar a su rendimiento.

HISTORIA

El primer algoritmo para bosques de decisión aleatoria fue creado en 1995 por Tin Kam Ho utilizando el método del subespacio aleatorio, que, en la formulación de Ho, es una forma de implementar el enfoque de clasificación de "discriminación estocástica" propuesto por Eugene Kleinberg.

Una extensión del algoritmo fue desarrollada por Leo Breiman y Adele Cutler, quienes registraron "Random Forests" como marca registrada en 2006 (a partir de 2019, propiedad de Minitab, Inc.). La extensión combina la idea de "embolsado" de Breiman y la selección aleatoria de características, introducida primero por Ho y luego de forma independiente por Amit y Geman para construir una colección de árboles de decisión con varianza controlada.

CARACTERÍSTICAS DE RANDOM FOREST



Fácil

Provee una visión resumida y global del comportamiento del modelo

Flexible

Se puede utilizar para problemas de clasificación y de regresión. Es posible usarlo como método no supervisado (clustering) y detección de outliers.

Estabilidad

Probado en millones de problemas a lo largo del tiempo se le considera la "panacea" en todos los problemas de ciencia de datos

Robustez

A diferencia de los árboles de decisión tradicionales es muy robusto debido a que un modelo "débil" se combina para crear un modelo robusto.

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor>

OPCIÓN ALGORITMO

03

DEBIDO A LA FLEXIBILIDAD Y AL TIPO DE DATOS CON ALTA ESTACIONALIDAD DENTRO DE LAS SERIES DE TIEMPO EL ALGORITMO SELECCIONADO FUE PROPHET.

PROPHET

SOBRE PROPHET

Prophet es un procedimiento para pronosticar datos de series temporales basados en un modelo aditivo donde las tendencias no lineales se ajustan a la estacionalidad anual, semanal y diaria, además de los efectos de las vacaciones. Funciona mejor con series temporales que tienen fuertes efectos estacionales y varias temporadas de datos históricos. Prophet es robusto a los datos faltantes y los cambios en la tendencia, y generalmente maneja bien los valores atípicos.

Prophet es un software de código abierto lanzado por el equipo de Core Data Science de Facebook. Está disponible para su descarga en CRAN y PyPI.

HISTORIA

2017, Facebook lanzó Prophet, una herramienta de pronóstico de código abierto en Python y R. La demanda de pronósticos de alta calidad a menudo supera a los analistas que los producen. Esta situación fue la motivación detrás de la construcción de una herramienta como Prophet que facilita que tanto expertos como no expertos entreguen pronósticos de alta calidad.

Prophet ha ganado una popularidad masiva entre otras herramientas de pronóstico. Hasta la fecha, el paquete Prophet ha sido descargado 21,819,762 veces. Según un informe, Prophet también encabeza entre otros paquetes de series de tiempo de Python cuando se clasifica por descargas mensuales.

CARACTERÍSTICAS DE PROPHET

PROPHET

Rápido y preciso

Prophet se utiliza en muchas aplicaciones de Facebook para producir pronósticos confiables para la planificación y el establecimiento de objetivos. Hemos encontrado que funciona mejor que cualquier otro enfoque en la mayoría de los casos. Ajustamos modelos en Stan para que obtenga pronósticos en solo unos segundos.

Totalmente automático

Obtenga un pronóstico razonable sobre datos desordenados sin esfuerzo manual. Prophet es robusto a los valores atípicos, los datos faltantes y los cambios dramáticos en sus series temporales.

Previsiones ajustables

El procedimiento prophet incluye muchas posibilidades para que los usuarios modifiquen y ajusten los pronósticos. Puede usar parámetros interpretables por humanos para mejorar su pronóstico agregando su conocimiento del dominio y el negocio al que se está aplicando.

Disponible en R y Python

Los desarrolladores han implementado el procedimiento Prophet en R y Python, pero comparten el mismo código Stan subyacente para el ajuste. Por lo que se puede utilizar cualquier lenguaje de programación con el que te sientas cómodo para obtener pronósticos.

<https://facebook.github.io/prophet/>

INSTALACIÓN Y CARGA DE LIBRERÍAS

Para realizar el proyecto se tuvieron que instalar algunas librerías nuevas en función de los algoritmos elegidos y cargar las necesarias para abordar el proyecto

```
1 !pip install ml_metrics  
2 !pip install fbprophet
```

```
1 import pandas as pd  
2 import numpy as np  
3 import matplotlib.pyplot as plt  
4 from IPython.core.pylabtools import figsize  
5 import seaborn as sns  
6 sns.set_style('darkgrid', {'axes.facecolor': '.9'})  
7 sns.set_palette(palette='deep')  
8 sns_c = sns.color_palette(palette='deep')  
9 %matplotlib inline  
10 from sklearn import model_selection  
11 from sklearn.ensemble import RandomForestRegressor  
12 from sklearn.model_selection import cross_val_score  
13 from sklearn.model_selection import train_test_split  
14 from sklearn.model_selection import GridSearchCV  
15 from sklearn.model_selection import ParameterGrid  
16 from sklearn.linear_model import LinearRegression  
17 from sklearn.preprocessing import MinMaxScaler  
18 from sklearn.model_selection import RandomizedSearchCV  
19 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score  
20 import warnings  
21 warnings.filterwarnings('ignore')  
22 from math import sqrt  
23 import multiprocessing  
24 import xgboost as xgb  
25 from xgboost import XGBRegressor  
26 from xgboost import plot_importance, plot_tree  
27 import ml_metrics as metrics  
28 import time  
29 from fbprophet import Prophet  
30 from fbprophet.plot import plot_plotly, plot_components_plotly  
31 from fbprophet.diagnostics import cross_validation  
32 from fbprophet.plot import plot_cross_validation_metric  
33 from fbprophet.diagnostics import performance_metrics  
34 from fbprophet.plot import plot_cross_validation_metric
```

DATA ADQUISITION , DATA WRANGLING Y EDA

Los datos fueron adquiridos del programa NCR ML de FG Burgers el cual está conectado con el software operativo de los locales y registra todos los datos de ventas de los locales.

Para este problema se decidió tomar datos desde el año 2016 hasta la fecha (31/03/2022), para ello se saco el informe del sistema y se exportó a formato .CSV para poder procesarlo posteriormente en Python.

Cabe destacar que el dataset utilizado venía bastante limpio de origen y no se tuvieron que realizar demasiadas transformaciones de los datos, a continuación lo realizado:

Carga de datos al modelo

```
1 url_ventas = 'https://content/drive/MyDrive/Data_Scientist/Trabajo_final/ventas2.csv'  
2 df_ventas = pd.read_csv(url_ventas, sep=';', encoding='latin-1')
```

Visualización y transformación

Se visualiza el dataset cargado

1 df_ventas								
	Local	Nombre_Local	Fecha	Dia_Semana	Ventas	Clientes	Laborable	Festivo
0	1	Gran Via	31/03/2022	4	8290	576	1	0
1	2	Parquesur	31/03/2022	4	6900	442	1	0
2	3	Plaza Rio	31/03/2022	4	5991	412	1	0
3	4	La Gavia	31/03/2022	4	7808	501	1	0
4	5	La Maquinista	31/03/2022	4	6547	464	1	0
...
47969	18	Diagonal	02/01/2016	5	12181	691	0	1
47970	19	Valencia	02/01/2016	5	11199	628	0	1
47971	20	Paseo de Gracia	02/01/2016	5	11608	663	0	1
47972	21	Xanadu	02/01/2016	5	12569	704	0	1
47973	22	La Vaguada	02/01/2016	5	11508	652	0	1

DATA ADQUISITION , DATA WRANGLING Y EDA

Examinamos el tipo de datos de nuestro dataset, podemos observar que la fecha está en formato object.

```
1 df_ventas.dtypes
```

Local	int64
Nombre_Local	object
Fecha	object
Dia_Semana	int64
Ventas	int64
Clientes	int64
Laborable	int64
Festivo	int64
dtype:	object

Convertimos la fecha del formato object al formato fecha ya que es el formato que necesitamos para nuestro modelo especificando el formato en el que queremos que se muestre. Revisamos que se haya realizado de forma correcta.

```
1 df_ventas[ 'Fecha' ] = pd.to_datetime(df_ventas[ 'Fecha' ], format='%d/%m/%Y')  
2 df_ventas
```

	Local	Nombre_Local	Fecha	Dia_Semana	Ventas	Clientes	Laborable	Festivo
0	1	Gran Via	2022-03-31	4	8290	576	1	0
1	2	Parquesur	2022-03-31	4	6900	442	1	0
2	3	Plaza Rio	2022-03-31	4	5991	412	1	0
3	4	La Gavia	2022-03-31	4	7808	501	1	0
4	5	La Maquinista	2022-03-31	4	6547	464	1	0
...
47969	18	Diagonal	2016-01-02	5	12181	691	0	1
47970	19	Valencia	2016-01-02	5	11199	628	0	1
47971	20	Paseo de Gracia	2016-01-02	5	11608	663	0	1
47972	21	Xanadu	2016-01-02	5	12569	704	0	1
47973	22	La Vaguada	2016-01-02	5	11508	652	0	1

47974 rows × 8 columns

```
1 df_ventas.dtypes
```

Local	int64
Nombre_Local	object
Fecha	datetime64[ns]
Dia_Semana	int64
Ventas	int64
Clientes	int64
Laborable	int64
Festivo	int64
dtype:	object

DATA ADQUISITION, DATA WRANGLING Y EDA

Sacamos más información del dataset y revisamos si tenemos algún nulo dentro de nuestros datos y observamos que no hay nulos.

```
1 df_ventas.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 47974 entries, 0 to 47973  
Data columns (total 8 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --          --  
 0   Local        47974 non-null   int64    
 1   Nombre_Local 47974 non-null   object    
 2   Fecha         47974 non-null   datetime64[ns]  
 3   Dia_Semana   47974 non-null   int64    
 4   Ventas       47974 non-null   int64    
 5   Clientes     47974 non-null   int64    
 6   Laborable    47974 non-null   int64    
 7   Festivo      47974 non-null   int64    
dtypes: datetime64[ns](1), int64(6), object(1)  
memory usage: 2.94 MB
```

```
1 df_ventas.isnull().sum()  
  
Local          0  
Nombre_Local   0  
Fecha          0  
Dia_Semana    0  
Ventas         0  
Clientes       0  
Laborable      0  
Festivo         0  
dtype: int64
```

Revisamos los principales estadísticos de nuestro dataset y sacamos las primeras conclusiones de los datos

```
1 round(df_ventas.describe())  
  
   Local  Dia_Semana  Ventas  Clientes  Laborable  Festivo  
count  47974.0  47974.0  47974.0  47974.0  47974.0  47974.0  
mean   11.0     4.0     8042.0    503.0     1.0     0.0  
std    6.0      2.0     1484.0    74.0      0.0     0.0  
min    1.0      1.0     5200.0    329.0     0.0     0.0  
25%    6.0      2.0     6955.0    448.0     0.0     0.0  
50%   11.0     4.0     7845.0    498.0     1.0     0.0  
75%   17.0     6.0     8883.0    551.0     1.0     0.0  
max   22.0     7.0    15550.0    900.0     1.0     1.0
```

Insight

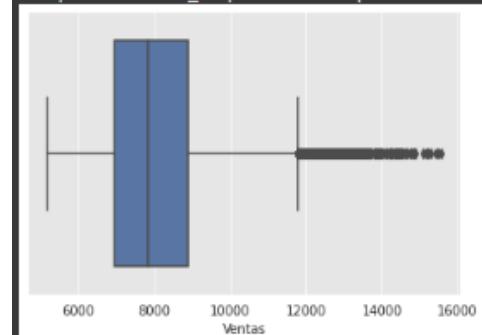
- La media de ventas es **8042**
- La menor venta fue **5200**
- La mayor venta fue **15550**
- La media de clientes fue **503**
- El menor registro de clientes fue **329**
- El mayor registro de clientes fue **900**

DATA ADQUISITION, DATA WRANGLING Y EDA

Realizamos varias visualizaciones para realizar un análisis más profundo de los datos

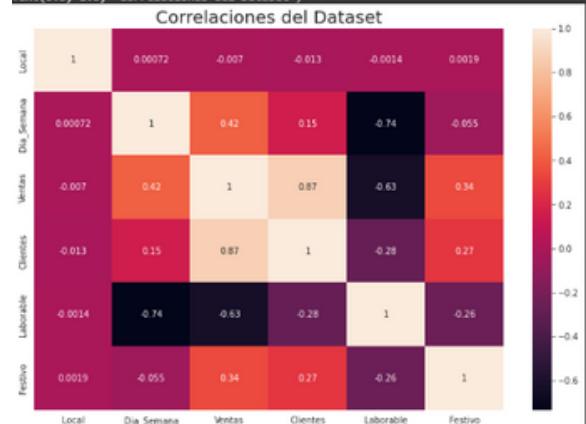
```
1 from seaborn import boxplot  
2 boxplot(df_ventas.Ventas)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f910d7cfb50>
```



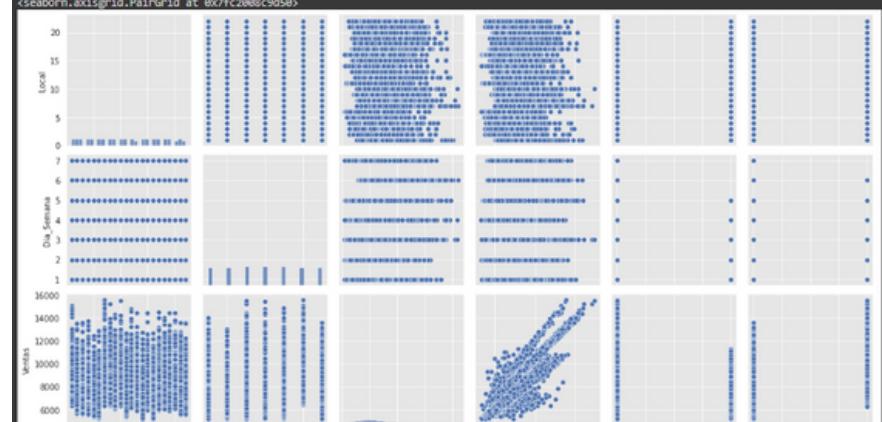
```
1 plt.figure(figsize=(12, 8))  
2 df_corr = df_ventas.corr()  
3 sns.heatmap(df_corr,  
4             xticklabels = df_corr.columns.values,  
5             yticklabels = df_corr.columns.values,  
6             annot = True);  
7 plt.title('Correlaciones del Dataset', fontsize = 20)
```

```
Text(0.5, 1.0, 'Correlaciones del Dataset')
```



```
1 sns.pairplot(df_ventas)
```

```
<seaborn.axisgrid.PairGrid at 0x7fc2000c9d50>
```



DATA ADQUISITION, DATA WRANGLING Y EDA

Al sacar las correlaciones como podíamos esperar podemos observar que la correlación más fuerte existe entre los clientes y las ventas lo cual es bastante obvio dado que si acuden más clientes a los locales es esperable que las ventas suban y a la inversa. También hay cierta relación entre si el día es festivo y que día de la semana es, esto nos puede indicar que hay un patrón de estacionalidad y que seguramente las ventas suban cuando es fin de semana o cuando existe algún festivo.

```
1 correlations_ventas = df_ventas.corr()['Ventas'].sort_values()
2 correlations_ventas

Laborable    -0.629370
Local        -0.007045
Festivo       0.344418
Dia_Semana   0.421451
Clientes     0.872380
Ventas       1.000000
Name: Ventas, dtype: float64
```

Preparamos definitivamente nuestro dataset, como las ventas y los clientes están muy estrechamente relacionados y la variable que queremos predecir son las ventas, no vamos a tener en cuenta la variable clientes, también vamos a eliminar el resto de columnas del dataset dejando solo las ventas y la fecha, aunque el día de la semana y el festivo influye en las ventas en este primer paso también los eliminamos y después descompondremos la fecha en distintos valores para crear unos features que nos ayudaran a captar esta tendencia. Ya que queremos pronosticar las ventas totales de la empresa vamos a agrupar todas las ventas por fecha así obtendremos las ventas totales de la empresa y no de cada local.

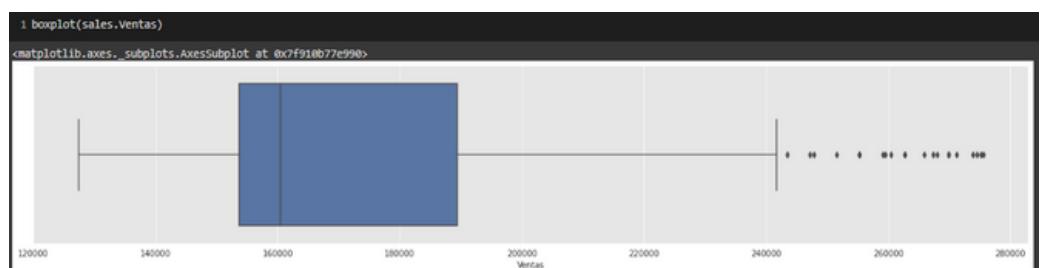
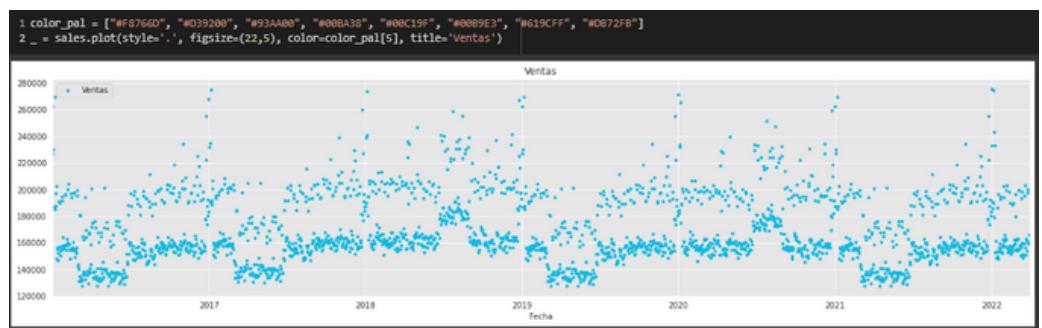
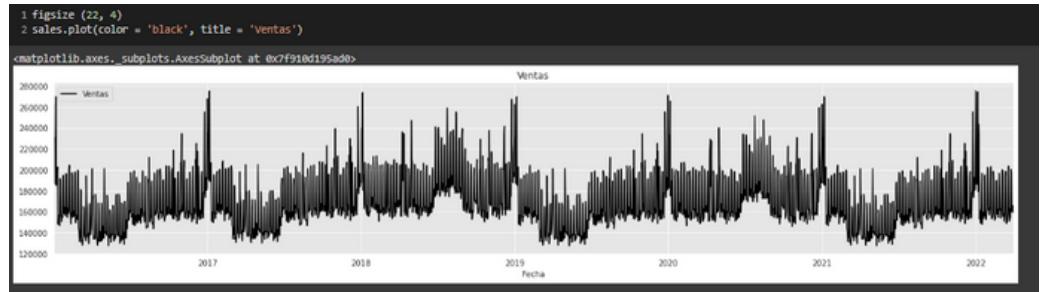
```
1 store_sales = df_ventas.drop(['Local','Nombre_local','Laborable','Festivo','Dia_Semana','Clientes'], axis=1)
2 sales = store_sales.groupby('Fecha').sum().reset_index()
3 sales.set_index('Fecha', inplace = True)
4 sales
```

Ventas	Fecha
262732	2016-01-02
227291	2016-01-03
230354	2016-01-04
186906	2016-01-05
186789	2016-01-06
...	...
197510	2022-03-27
159691	2022-03-28
160170	2022-03-29
165753	2022-03-30
159130	2022-03-31

2281 rows x 1 columns

DATA ADQUISITION, DATA WRANGLING Y EDA

Una vez creado nuestro dataset definitivo realizamos algunas visualizaciones y algún análisis de los datos. Podemos observar que nuestros datos tienen una clara tendencia cíclica y que como vimos en el primer análisis los picos de ventas coinciden con los períodos vacacionales o no lectivos.



```
1 round(sales.describe())
```

	Ventas
count	2281.0
mean	169130.0
std	25668.0
min	127380.0
25%	153770.0
50%	160465.0
75%	189464.0
max	275526.0

Insight

- La media de ventas es **169130**
- La menor venta fue **127380**
- La mayor venta fue **275526**

DEFINICIÓN DE LA VARIANTE TARGET

El principal objetivo del proyecto era poder crear un modelo para predecir las ventas de los locales de la marca por lo que claramente nuestra variante Target a utilizar en este caso será la columna VENTAS del dataset ventas.

1 df_ventas

	Local	Nombre_Local	Fecha	Dia_Semana	Ventas	Clientes	Laborable	Festivo
0	1	Gran Via	31/03/2022	4	6856	476	1	0
1	2	Parquesur	31/03/2022	4	6449	413	1	0
2	3	Plaza Rio	31/03/2022	4	5599	385	1	0
3	4	La Gavia	31/03/2022	4	7110	468	1	0
4	5	La Maquinista	31/03/2022	4	6356	450	1	0
...
17505	18	Diagonal	01/01/2020	3	9306	520	0	1
17506	19	Valencia	01/01/2020	3	8947	497	0	1
17507	20	Paseo de Gracia	01/01/2020	3	12417	700	0	1
17508	21	Xanadu	01/01/2020	3	11439	655	0	1
17509	22	La Vaguada	01/01/2020	3	12692	705	0	1

17510 rows x 8 columns

IMPLANTACIÓN DE LOS MODELOS

dmlc
XGBoost

PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

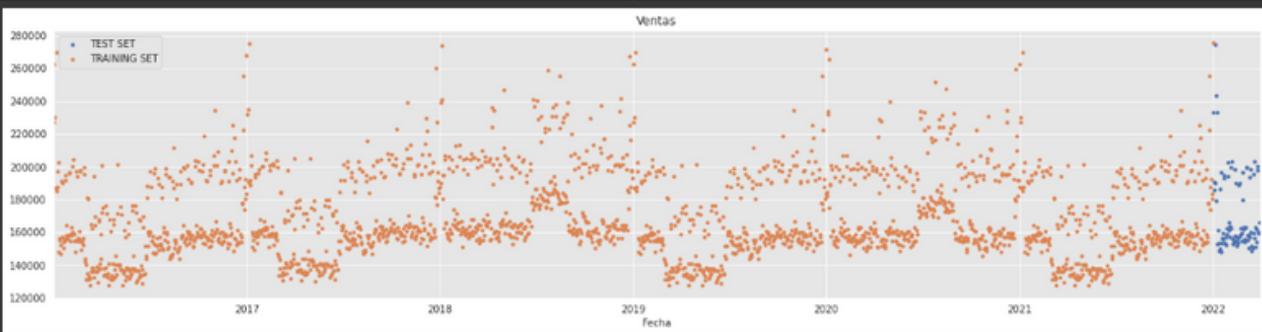
El primer paso a realizar es la división de los datos en dos conjuntos uno de train y otro de test, para esto vamos a hacer un corte en una fecha ya que estamos tratando con series de tiempo, visualizamos como quedó la división de los datos.

```
1 split_date = '01-Jan-2022'
2 data_train_x = sales.loc[sales.index <= split_date].copy()
3 data_test_x = sales.loc[sales.index > split_date].copy()
```

1 data_train_x	
	Ventas
Fecha	
2016-01-02	262732
2016-01-03	227291
2016-01-04	230354
2016-01-05	186906
2016-01-06	186789
...	...
2021-12-28	187465
2021-12-29	192557
2021-12-30	180330
2021-12-31	183470
2022-01-01	275526
2192 rows × 1 columns	

1 data_test_x	
	Ventas
Fecha	
2022-01-02	233023
2022-01-03	190637
2022-01-04	186002
2022-01-05	190020
2022-01-06	274468
...	...
2022-03-27	197510
2022-03-28	159691
2022-03-29	160170
2022-03-30	165753
2022-03-31	159130
89 rows × 1 columns	

```
1 _ = data_test_x \
2     .rename(columns={'Ventas': 'TEST SET'}) \
3     .join(data_train_x.rename(columns={'Ventas': 'TRAINING SET'}), how='outer') \
4     .plot(figsize=(22,5), title='Ventas', style='.'
```



PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

Creamos el modelo, buscamos los mejores parámetros con Random search una vez encontrados estos creamos el modelo definitivo y lo entrenamos con nuestro dataset de entrenamiento.

```

1 reg1 = XGBRegressor(nthread=-1, random_state=100)

1 params1 = {
2     'n_estimators':[100, 500],
3     'min_child_weight':[2,3,4,5],
4     'gamma':[i/10.0 for i in range(3,6)],
5     'subsample':[i/10.0 for i in range(6,11)],
6     'colsample_bytree':[i/10.0 for i in range(6,11)],
7     'max_depth': [2,3,4,6,7],
8     'objective': ['reg:squarederror', 'reg:tweedie'],
9     'booster': ['gbtree', 'gblinear'],
10    'eval_metric': ['rmse'],
11    'eta': [i/10.0 for i in range(3,6)],
12 }

1 CV_reg1 = RandomizedSearchCV(estimator=reg1, param_distributions=params1,
2                                n_iter=100, cv=5, scoring='neg_mean_squared_error')
3 CV_reg1.fit(X_train_x, y_train_x)

RandomizedSearchCV(cv=5, estimator=XGBRegressor(nthread=-1, random_state=100),
                    n_iter=100,
                    param_distributions={'booster': ['gbtree', 'gblinear'],
                                         'colsample_bytree': [0.6, 0.7, 0.8, 0.9,
                                                             1.0],
                                         'eta': [0.3, 0.4, 0.5],
                                         'eval_metric': ['rmse'],
                                         'gamma': [0.3, 0.4, 0.5],
                                         'max_depth': [2, 3, 4, 6, 7],
                                         'min_child_weight': [2, 3, 4, 5],
                                         'n_estimators': [100, 500],
                                         'objective': ['reg:squarederror',
                                                       'reg:tweedie'],
                                         'subsample': [0.6, 0.7, 0.8, 0.9, 1.0]},
                                         scoring='neg_mean_squared_error')

1 CV_reg1.best_params_

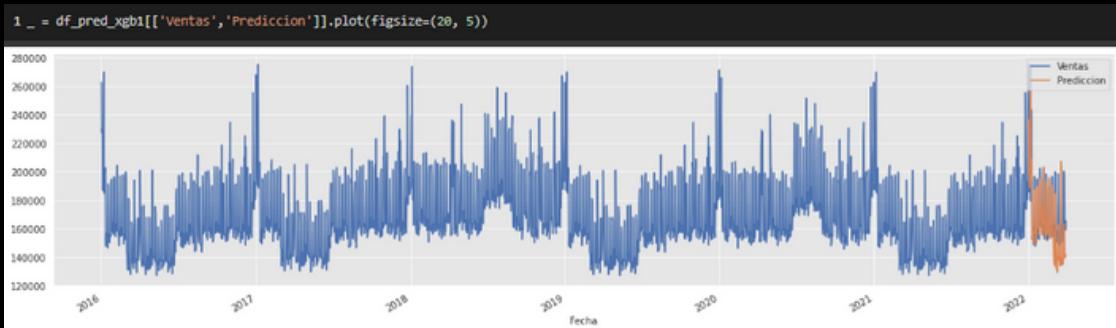
{'booster': 'gbtree',
 'colsample_bytree': 0.6,
 'eta': 0.4,
 'eval_metric': 'rmse',
 'gamma': 0.4,
 'max_depth': 4,
 'min_child_weight': 3,
 'n_estimators': 500,
 'objective': 'reg:squarederror',
 'subsample': 0.7}

1 XGB1= XGBRegressor(
2     random_state=100,
3     n_estimators= 500,
4     min_child_weight= 5,
5     gamma= 0.5,
6     subsample= 0.9 ,
7     colsample_bytree= 0.6,
8     max_depth= 7,
9     objective = 'reg:squarederror',
10    booster = 'gbtree',
11    eval_metric = 'rmse',
12    eta= 0.5,
13 )
14 XGB1.fit(X_train_x, y_train_x)
15 XGB1_pred = XGB1.predict(X_test_x)
```

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

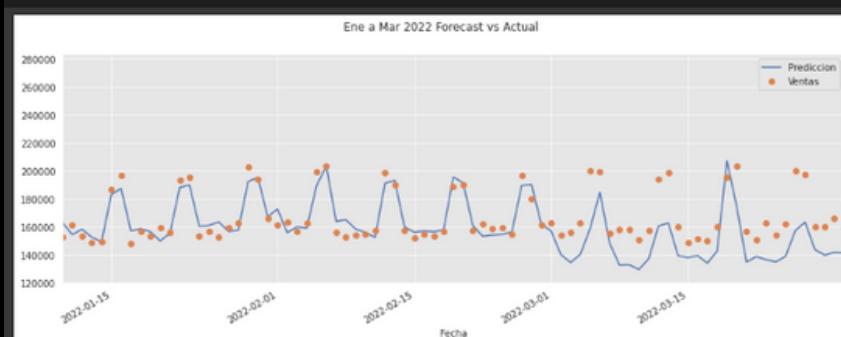
Realizamos las predicciones del modelo y las comparamos con nuestro dataset de test, elaboramos varias gráficas y tablas con las mejores y peores predicciones del modelo.

```
1 data_test_x['Prediccion'] = XGB1.predict(X_test_x)
2 df_pred_xgb1 = pd.concat([data_test_x, data_train_x], sort=False)
```

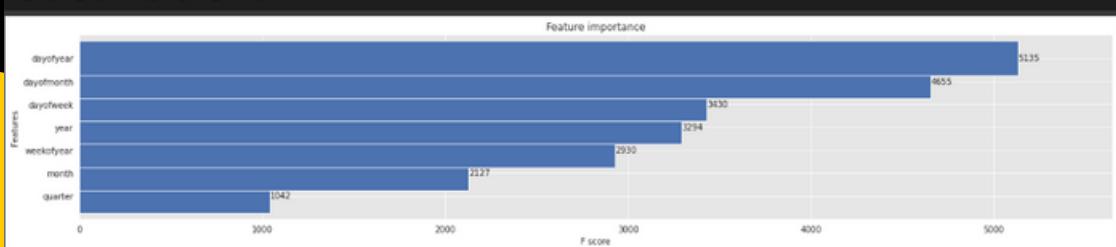


	Fecha	2022-01-02	2022-01-03	2022-01-04	2022-01-05	2022-01-06
Ventas		233023	190637	186002	190020	274468
date		2022-01-02 00:00:00	2022-01-03 00:00:00	2022-01-04 00:00:00	2022-01-05 00:00:00	2022-01-06 00:00:00
hour		0	0	0	0	0
dayofweek		6	0	1	2	3
quarter		1	1	1	1	1
month		1	1	1	1	1
year		2022	2022	2022	2022	2022
dayofyear		2	3	4	5	6
dayofmonth		2	3	4	5	6
weekofyear		52	1	1	1	1
Prediccion		236689.90625	198505.546875	188743.984375	202192.875	257505.03125

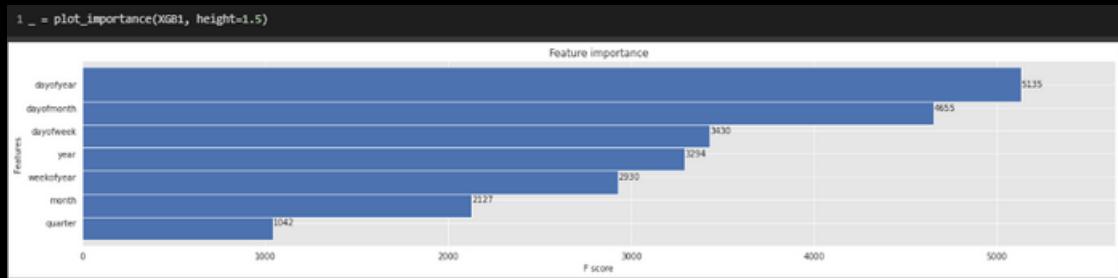
```
1 f, ax = plt.subplots(1)
2 f.set_figheight(5)
3 f.set_figwidth(15)
4 _ = df_pred_xgb1[['Prediccion','Ventas']].plot(ax=ax, style=['-','o'])
5 ax.set_xbound(lower='2022-01-10', upper='2022-03-31')
6 plot = plt.suptitle('Ene a Mar 2022 Forecast vs Actual')
```



```
1 _ = plot_importance(XGB1, height=1.5)
```



IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO



```
1 # Peores predicciones
2 Wpd = error_by_day.sort_values('error', ascending=True).head()
3 Wpd
```

		Ventas	Prediccion	error	abs_error
year	month	dayofmonth			
2022	1	7	179314.0	199190.656250	-19876.656250
		5	190020.0	202192.875000	-12172.875000
	2	8	152984.0	164985.953125	-12001.953125
	3	19	195105.0	207036.078125	-11931.078125
	2	1	161154.0	172622.062500	-11468.062500

```
1 # Mejores predicciones
2 Bpd = error_by_day.sort_values('abs_error', ascending=True).head()
3 Bpd
```

		Ventas	Prediccion	error	abs_error
year	month	dayofmonth			
2022	2	28	161654.0	161755.687500	-101.687500
		6	203019.0	203276.734375	-257.734375
	1	14	149668.0	149330.984375	337.015625
		21	155798.0	155349.765625	448.234375
		31	166205.0	167115.781250	910.781250

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Sacamos las medidas de performance del modelo

```

1 mse_xgb = mean_squared_error(y_true=data_test_x['Ventas'],
2                               y_pred=data_test_x['Prediccion'])
3
4 r2_xgb= r2_score(y_true=data_test_x['Ventas'],
5                   y_pred=data_test_x['Prediccion'])
6
7 rmse_xgb= sqrt(mean_squared_error(y_true=data_test_x['Ventas'],
8                                    y_pred=data_test_x['Prediccion']))
9
10 mae_xgb= mean_absolute_error(y_true=data_test_x['Ventas'],
11                                y_pred=data_test_x['Prediccion'])
12
13 def mean_absolute_percentage_error(y_true, y_pred):
14     """Calculamos el MAPE teniendo y_true e y_pred"""
15     y_true, y_pred_x = np.array(y_true), np.array(y_pred)
16     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
17
18 mape_xgb= mean_absolute_percentage_error(y_true=data_test_x['Ventas'],
19                                            y_pred=data_test_x['Prediccion'])
20
21 xgb_stats = [rmse_xgb, mae_xgb, mape_xgb, r2_xgb]
22
23 xgb_stats

```

[15885.739699193946, 11857.725245786516, 6.782095115209348, 0.5654377617905781]

Finalmente hacemos predicciones para los dos próximos meses y las guardamos en un dataset que previamente hemos creado en excel con las fechas las cuales queremos predecir.

```

1 url_predicciones='/content/drive/MyDrive/Data Scientist/Trabajo final/proyecciones.xlsx'
2 test = pd.read_excel(url_predicciones, parse_dates=[0], index_col=[0] )
3 pred_x, pred_y = create_features(test, label='Ventas')
4
5 test['Predicciones_xgb'] = XGB1.predict(pred_x)
6
7 test.drop(['Ventas','hour', 'dayofweek', 'quarter', 'month', 'year', 'dayofyear',
8            'dayofmonth', 'weekofyear', 'date'], axis=1, inplace=True)
9 test.head(61)

      Predicciones_xgb
    Fecha
2022-04-01  167519.140625
2022-04-02  210972.890625
2022-04-03  172278.890625
2022-04-04  141294.546875
2022-04-05  140522.500000
...
2022-05-27  138093.812500
2022-05-28  167836.546875
2022-05-29  167500.828125
2022-05-30  143051.390625
2022-05-31  143657.437500

```

61 rows x 1 columns

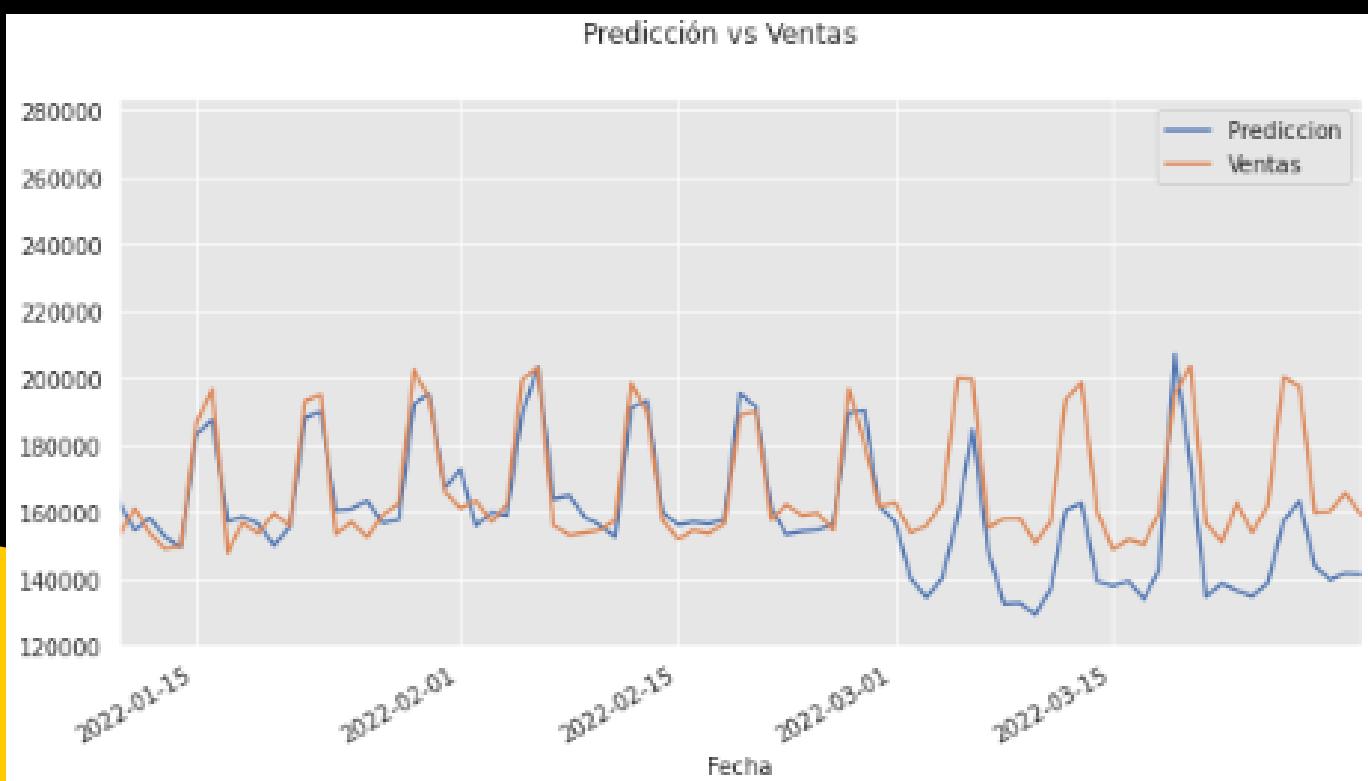
CONCLUSIONES DEL MODELO

01

El modelo pese a no tener muchas variables de entrada funciono de manera aceptable obteniendo las siguientes métricas

1 rmse_xgb**15885.739699193946****1 r2_xgb****0.5654377617905781****1 mae_xgb****11857.725245786516****1 mape_xgb****6.782095115209348****02**

Observamos que el modelo predice de forma acertada en los primeros meses pero que en las últimas fechas va perdiendo efectividad, seguramente porque disponemos de menos observaciones para esas fechas



CONCLUSIONES DEL MODELO

03

Finalmente se entrena al modelo dividiendo los datos de entrenamiento y test en un 70%-30% y podemos ver que el modelo si bien pierde algo de efectividad sigue siendo valido.

```
1 data_train_a, data_test_a = train_test_split(sales, test_size = 0.30, random_state = 100)

2 X_train_a, y_train_a = create_features(data_train_a, label='Ventas')
3 X_test_a, y_test_a = create_features(data_test_a, label='Ventas')
3 X_train_a

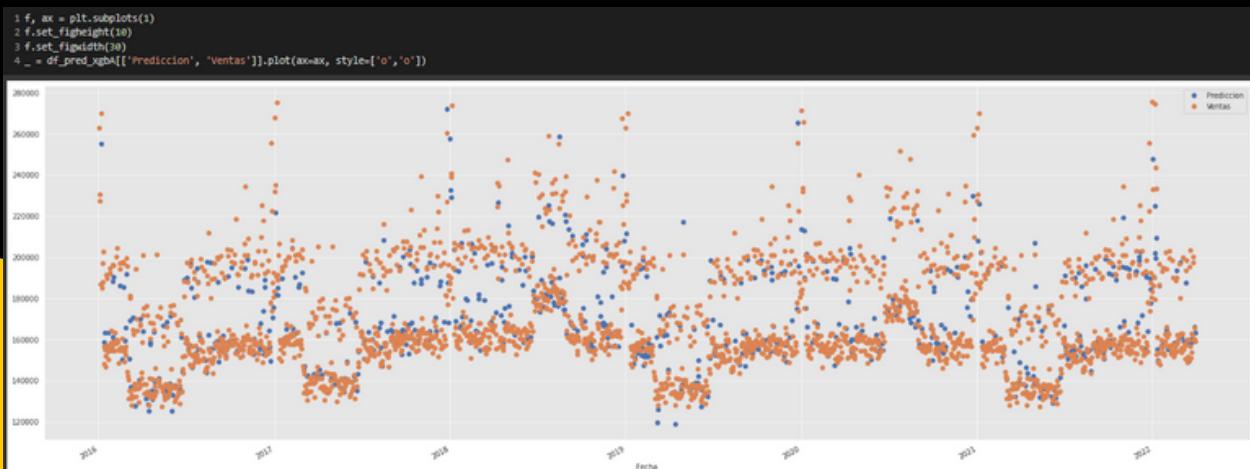
hour dayofweek quarter month year dayofyear dayofmonth weekofyear
Fecha
2019-12-18 0 2 4 12 2019 352 18 51
2020-08-20 0 3 3 8 2020 233 20 34
2017-12-06 0 2 4 12 2017 340 6 49
2019-12-04 0 2 4 12 2019 338 4 49
2016-02-13 0 5 1 2 2016 44 13 6
...
2016-12-17 0 5 4 12 2016 352 17 50
2021-04-15 0 3 2 4 2021 105 15 15
2016-03-21 0 0 1 3 2016 81 21 12
2021-02-03 0 2 1 2 2021 34 3 5
2020-03-25 0 2 1 3 2020 85 25 13
1596 rows x 8 columns
```

```
1 XGBA= XGBRegressor(
2     random_state=100,
3     n_estimators= 500,
4     min_child_weight= 5,
5     gamma= 0.5,
6     subsample= 0.9 ,
7     colsample_bytree= 0.6,
8     max_depth= 7,
9     objective = 'reg:squarederror',
10    booster = 'gbtree',
11    eval_metric = 'rmse',
12    eta= 0.5,
13 )
14 XGBA.fit(X_train_a, y_train_a)
15 XGBA_pred = XGBA.predict(X_test_a)

1 data_test_a['Prediccion'] = XGBA.predict(X_test_a)
2 df_pred_xgbA = pd.concat([data_test_a, data_train_a], sort=False)
```

```
1 r2_xgba= r2_score(y_true=data_test_a['Ventas'],
2                     y_pred=data_test_a['Prediccion'])
3 r2_xgba

0.7647338478472632
```



IMPLANTACIÓN DE LOS MODELOS



Random Forest



PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

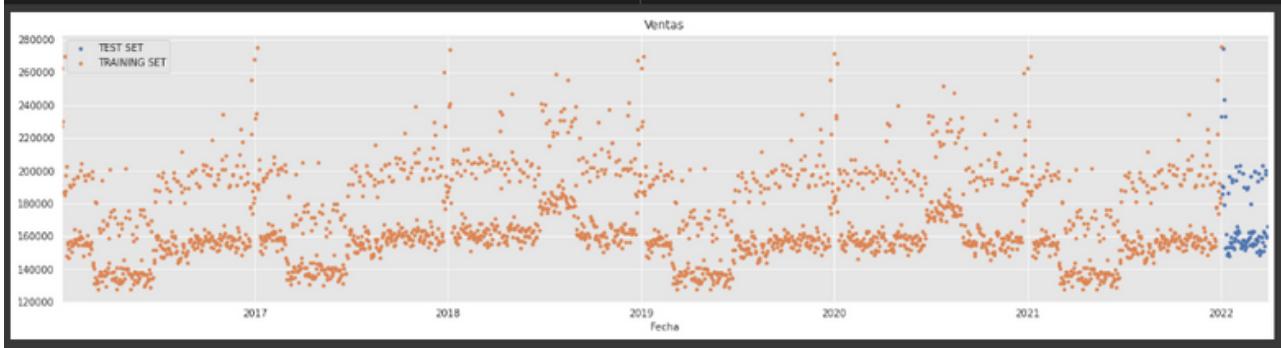
El primer paso a realizar es la división de los datos en dos conjuntos uno de train y otro de test, para esto vamos a hacer un corte en una fecha ya que estamos tratando con series de tiempo, visualizamos como quedó la división de los datos.

```
1 split_date = '01-Jan-2022'
2 data_train_rf = sales.loc[sales.index <= split_date].copy()
3 data_test_rf = sales.loc[sales.index > split_date].copy()
```

	Ventas
Fecha	
2016-01-02	262732
2016-01-03	227291
2016-01-04	230354
2016-01-05	186906
2016-01-06	186789
...	...
2021-12-28	187465
2021-12-29	192557
2021-12-30	180330
2021-12-31	183470
2022-01-01	275526
?	192 rows x 1 columns

	Ventas
Fecha	
2022-01-02	233023
2022-01-03	190637
2022-01-04	186002
2022-01-05	190020
2022-01-06	274468
...	...
2022-03-27	197510
2022-03-28	159691
2022-03-29	160170
2022-03-30	165753
2022-03-31	159130
89 rows x 1 columns	

```
1 _ = data_test_x \
2 .rename(columns={'Ventas': 'TEST SET'}) \
3 .join(data_train_x.rename(columns={'Ventas': 'TRAINING SET'}), how='outer') \
4 .plot(figsize=(22,5), title='Ventas', style='.'
```



PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

Creamos el modelo, buscamos los mejores parámetros con Grid search una vez encontrados estos creamos el modelo definitivo y lo entrenamos con nuestro dataset de entrenamiento.

```
1 rfr=RandomForestRegressor(random_state=100)
```

```
1 param_grid = {
2     'n_estimators': [1, 100],
3     'max_features': ['auto', 'sqrt', 'log2'],
4     'max_depth': [4, 5, 6, 7, 8],
5     'criterion' : ['squared_error','mae']
6 }
```

```
1 CV_rfr = GridSearchCV(estimator=rfr, param_grid=param_grid, cv= 5)
2 CV_rfr.fit(X_train_rf, y_train_rf)

GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=100),
             param_grid={'criterion': ['squared_error', 'mae'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [1, 100]})
```

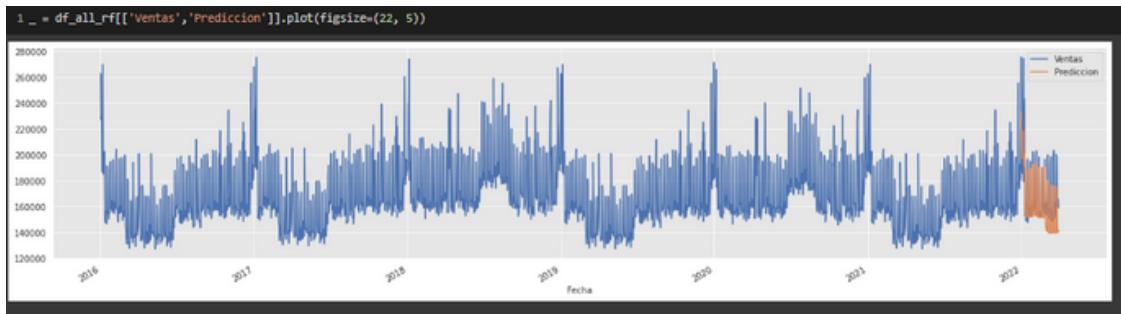
```
1 CV_rfr.best_params_
{'criterion': 'mae',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 100}
```

```
1 rf=RandomForestRegressor(random_state=100,
2                           criterion= 'mae',
3                           max_depth= 8,
4                           max_features= 'auto',
5                           n_estimators= 100)
6 rf.fit(X_train_rf, y_train_rf)
7 rf_pred = rf.predict(X_test_rf)
```

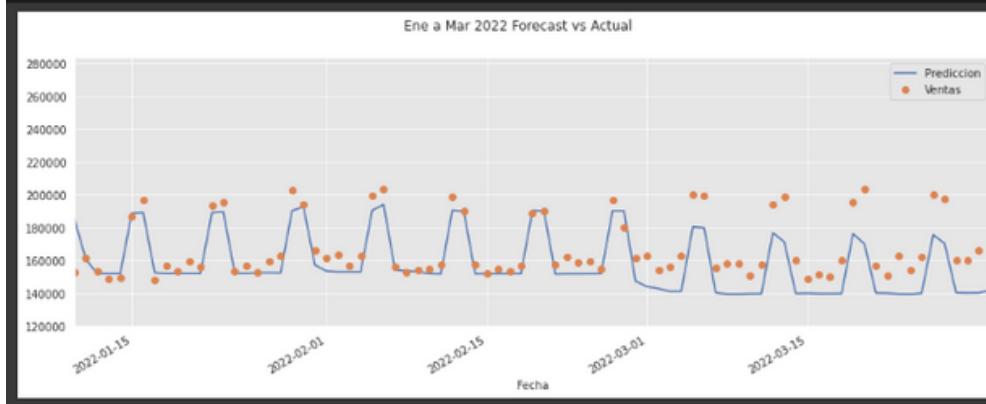
IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Realizamos las predicciones del modelo y las comparamos con nuestro dataset de test, elaboramos varias gráficas para analizar el performance del modelo.

```
1 data_test_rf['Prediccion'] = rf.predict(X_test_rf)
2 df_all_rf = pd.concat([data_test_rf, data_train_rf], sort=False)
```



```
1 f, ax = plt.subplots(1)
2 f.set_figheight(5)
3 f.set_figwidth(15)
4 _ = df_all_rf[['Prediccion', 'Ventas']].plot(ax=ax, style=['-', 'o'])
5 ax.set_xbound(lower='2022-01-10', upper='2022-03-31')
6 plot = plt.suptitle('Ene a Mar 2022 Forecast vs Actual')
```



IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Sacamos las medidas de performance del modelo

```
1 rmse_rf = mean_squared_error(  
2     y_true = y_test_rf,  
3     y_pred = rf_pred,  
4     squared = False  
5 )  
6  
7 mae_rf = mean_absolute_error(  
8     y_true = y_test_rf,  
9     y_pred = rf_pred,  
10    )  
11  
12 r2_rf = r2_score(  
13     y_true = y_test_rf,  
14     y_pred = rf_pred,  
15    )  
16  
17 mape_rf = mean_absolute_percentage_error(y_true=y_test_rf,  
18                 y_pred=rf_pred)  
19  
20 rf_stats = [rmse_rf, mae_rf, mape_rf, r2_rf]  
21 rf_stats
```

[15814.022219846973, 11818.05713483146, 6.658967746792643, 0.5693526387626127]

CONCLUSIONES DEL MODELO

01

Podemos observar que el modelo funcionó mejor que XGBoost obteniendo mejores métricas

1 rmse_rf
15814.022219846973

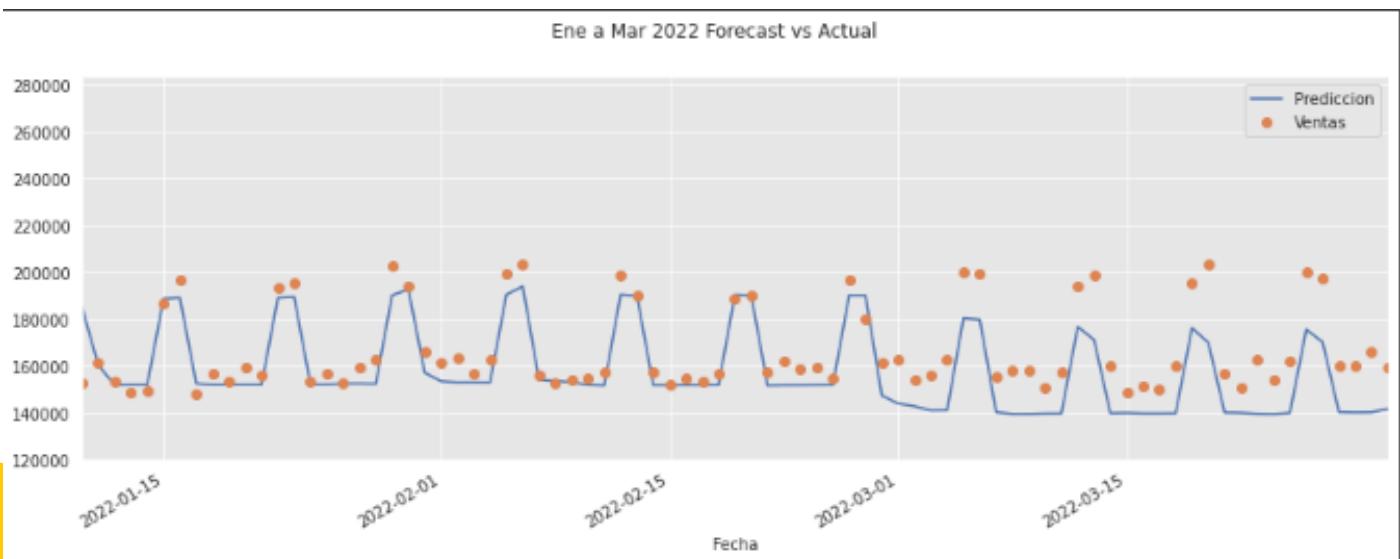
1 r2_rf
0.5693526387626127

1 mae_rf
11818.05713483146

1 mape_rf
6.658967746792643

02

Al igual que pasó con XGBoost, observamos que el modelo predice de forma acertada en los primeros meses pero que en las últimas fechas va perdiendo efectividad, seguramente porque disponemos de menos observaciones para esas fechas



CONCLUSIONES DEL MODELO

03

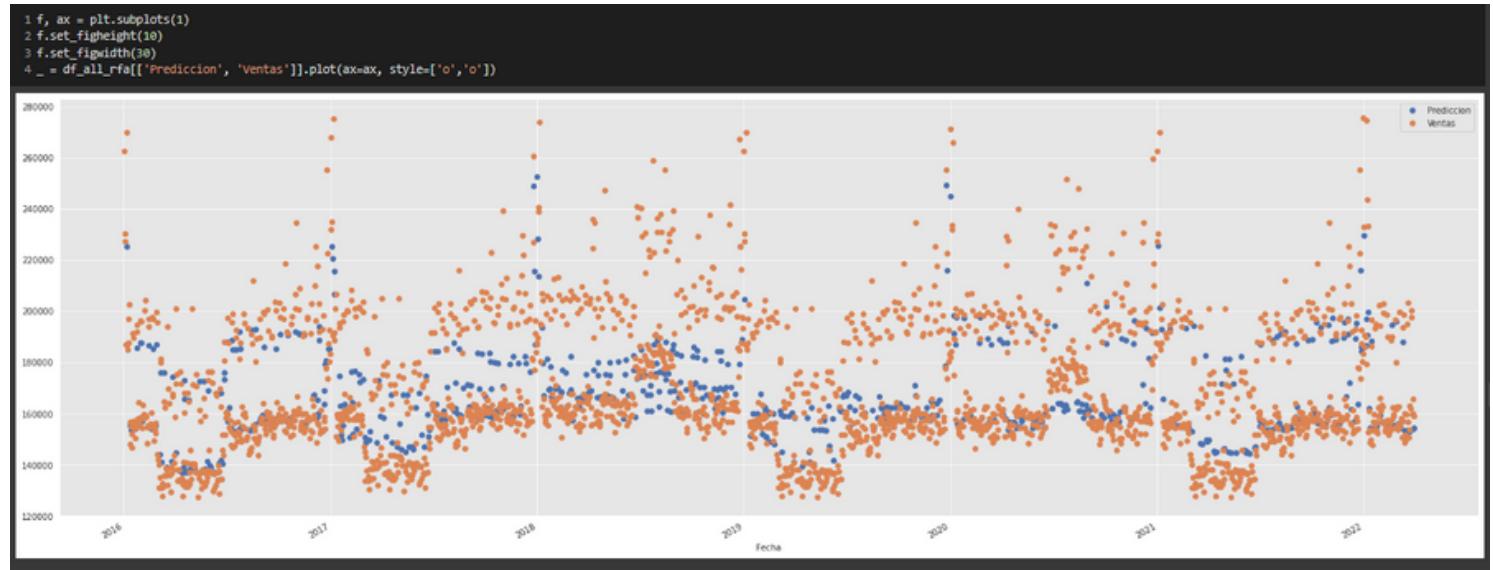
Finalmente se entrena al modelo dividiendo los datos de entrenamiento y test en un 70%-30% y podemos ver que al igual que nos pasó anteriormente con XGBoost el modelo si bien pierde algo de efectividad sigue siendo valido.

```
[ ] 1 rfa=RandomForestRegressor(random_state=100,
2                           criterion= 'mae',
3                           max_depth= 8,
4                           max_features= 'auto',
5                           n_estimators= 100)
6 rfa.fit(X_train_a, y_train_a)
7 rfa_pred = rfa.predict(X_test_a)

[ ] 1 data_test_a['Prediccion'] = rfa.predict(X_test_a)
2 df_all_rfa = pd.concat([data_test_a, data_train_a], sort=False)

[ ] 1 r2_rfa= r2_score(y_true=data_test_a['Ventas'],
2                      y_pred=data_test_a['Prediccion'])
3 r2_rfa

0.5809530207298041
```



IMPLANTACIÓN DE LOS MODELOS

PROPHET

PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

Para utilizar el algoritmo de Prophet primeramente tenemos que realizar algunas transformaciones en los datos así que primeramente realizaremos una copia de nuestro dataset, después quitamos la fecha del indice y renombramos nuestra columna fecha como 'ds' y nuestra columna ventas como 'y', indicar que esto es requisito para que el algoritmo de Prophet funcione.

```
1 sales_pro = sales.copy()  
2 sales_pro
```

	Ventas
	Fecha
0	2016-01-02 262732
1	2016-01-03 227291
2	2016-01-04 230354
3	2016-01-05 186906
4	2016-01-06 186789
...	...
2276	2022-03-27 197510
2277	2022-03-28 159691
2278	2022-03-29 160170
2279	2022-03-30 165753
2280	2022-03-31 159130

2281 rows × 1 columns

```
1 sales_pro=sales_pro.reset_index()
```

```
1 sales_p=sales_pro.rename(columns={'Fecha': 'ds', 'Ventas': 'y'})  
2 sales_p
```

	ds	y
0	2016-01-02	262732
1	2016-01-03	227291
2	2016-01-04	230354
3	2016-01-05	186906
4	2016-01-06	186789
...
2276	2022-03-27	197510
2277	2022-03-28	159691
2278	2022-03-29	160170
2279	2022-03-30	165753
2280	2022-03-31	159130

2281 rows × 2 columns

PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

Posteriormente creamos nuestras features de las fechas y graficamos nuestro dataset

```

1 sales_p['dayofweek'] = sales_p['ds'].dt.dayofweek
2 sales_p['quarter'] = sales_p['ds'].dt.quarter
3 sales_p['month'] = sales_p['ds'].dt.month
4 sales_p['year'] = sales_p['ds'].dt.year
5 sales_p['dayofyear'] = sales_p['ds'].dt.dayofyear
6 sales_p['dayofmonth'] = sales_p['ds'].dt.day
7 sales_p['weekofyear'] = sales_p['ds'].dt.weekofyear
8 sales_p

```

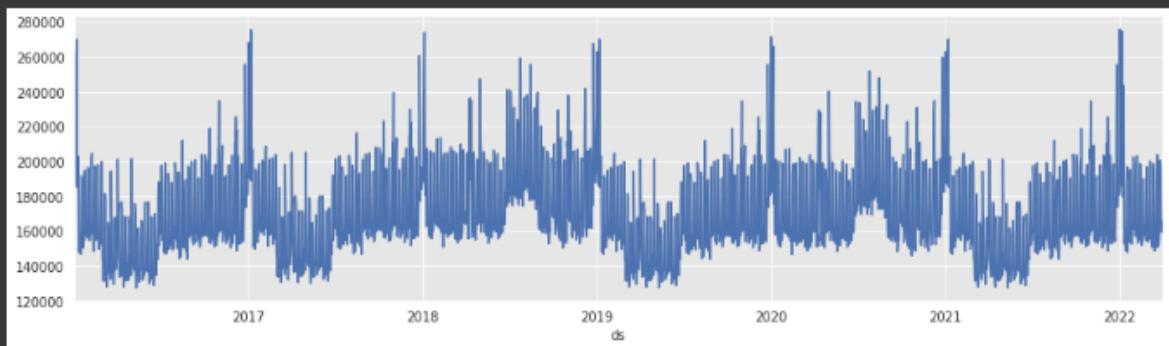
	ds	y	dayofweek	quarter	month	year	dayofyear	dayofmonth	weekofyear
0	2016-01-02	262732	5	1	1	2016	2	2	53
1	2016-01-03	227291	6	1	1	2016	3	3	53
2	2016-01-04	230354	0	1	1	2016	4	4	1
3	2016-01-05	186906	1	1	1	2016	5	5	1
4	2016-01-06	186789	2	1	1	2016	6	6	1
...
2276	2022-03-27	197510	6	1	3	2022	86	27	12
2277	2022-03-28	159691	0	1	3	2022	87	28	13
2278	2022-03-29	160170	1	1	3	2022	88	29	13
2279	2022-03-30	165753	2	1	3	2022	89	30	13
2280	2022-03-31	159130	3	1	3	2022	90	31	13

2281 rows × 9 columns

```

1 figsize(15,4)
2 sales_p.set_index('ds').y.plot().get_figure()

```



PARAMETRIZACIÓN Y CREACIÓN DEL MODELO

Creamos el modelo y lo entrenamos, hay que tener en cuenta que Prophet es bastante automático y que dentro de la parametrización de las cosas más importantes es la las tendencias de temporalidad

```
1 model = Prophet(seasonality_mode='multiplicative', changepoint_range=1, changepoint_prior_scale=0.75)
2 model.fit(data_train_p)

INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7fcdb48bc50>
```

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Para realizar las proyecciones primeramente tenemos que crear un dataframe en donde guardaremos nuestras predicciones

```
1 future = model.make_future_dataframe(periods=90, freq = 'd')
2 future.tail()

ds  ⚙
2277 2022-03-28
2278 2022-03-29
2279 2022-03-30
2280 2022-03-31
2281 2022-04-01
```

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Realizamos nuestras predicciones y las visualizamos en el dataframe creado previamente

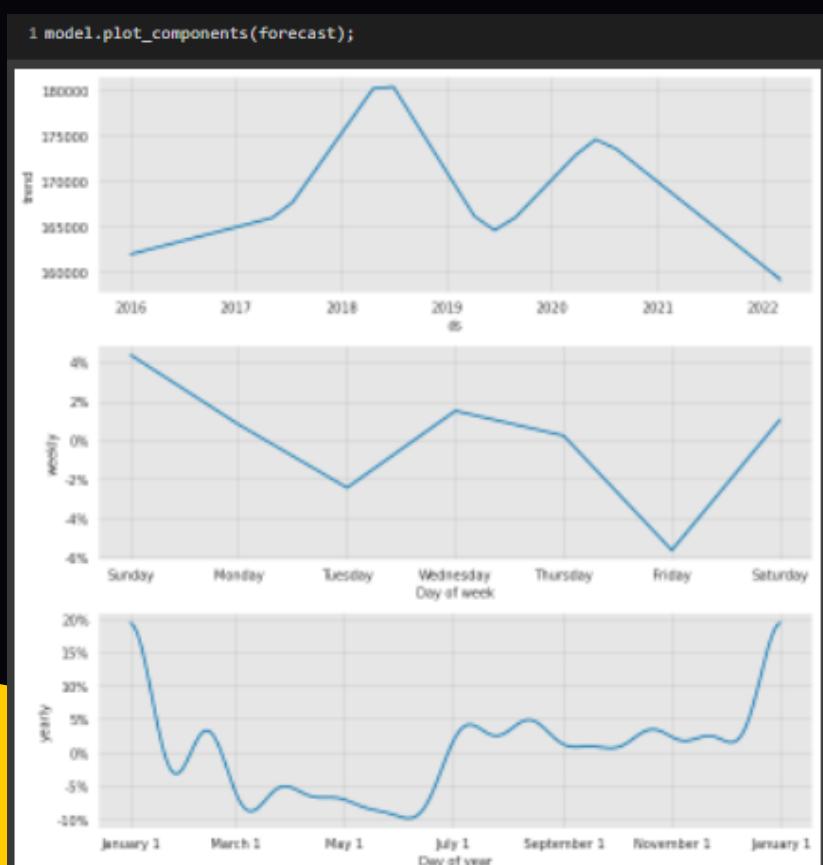
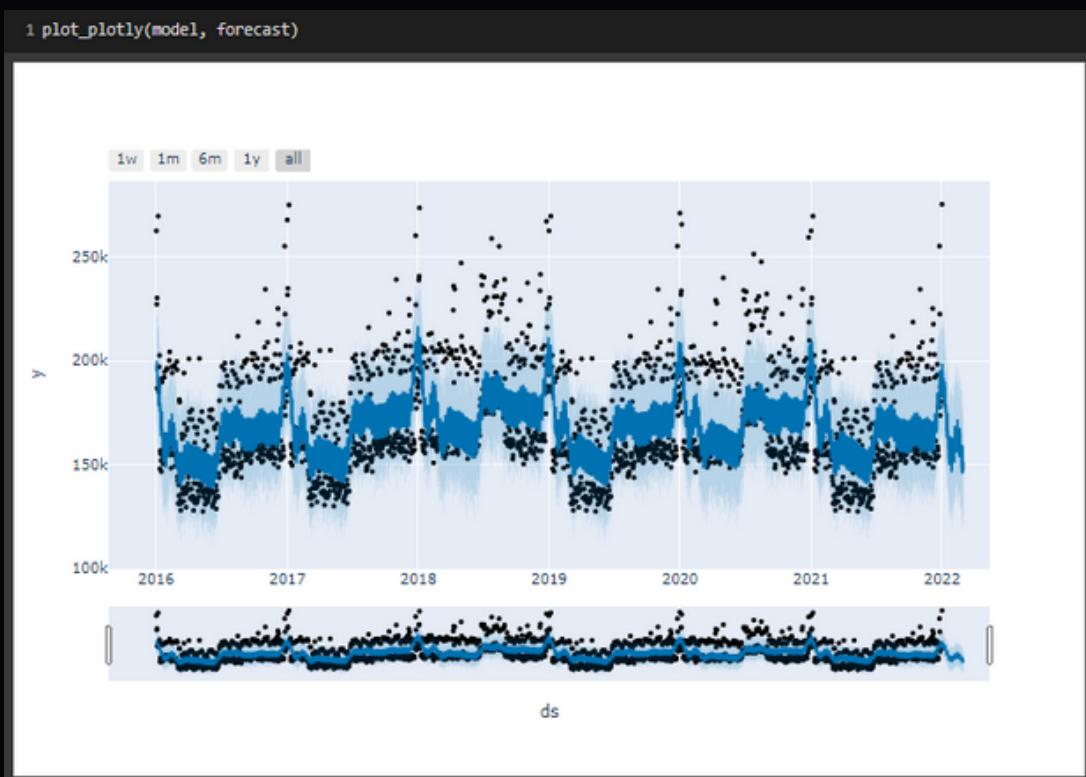
Forecast Dataframe														
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	multiplicative_terms	multiplicative_terms_lower	multiplicative_terms_upper	weekly	weekly_lower	weekly_upper	yearly	yearly_lower
0	2016-01-02	161948.842080	167338.383769	221396.405447	161948.842080	161948.842080	0.203867	0.203867	0.203867	0.010736	0.010736	0.010736	0.193131	0.193131
1	2016-01-03	161957.029867	173183.870727	228640.695722	161957.029867	161957.029867	0.234634	0.234634	0.234634	0.043850	0.043850	0.043850	0.190784	0.190784
2	2016-01-04	161965.217653	165957.932395	218926.813688	161965.217653	161965.217653	0.194943	0.194943	0.194943	0.008122	0.008122	0.008122	0.186821	0.186821
3	2016-01-05	161973.405440	159079.239928	214155.806367	161973.405440	161973.405440	0.157106	0.157106	0.157106	-0.024173	-0.024173	-0.024173	0.181279	0.181279
4	2016-01-06	161981.593227	165056.616871	221928.818382	161981.593227	161981.593227	0.189409	0.189409	0.189409	0.015183	0.015183	0.015183	0.174226	0.174226

Prophet nos da muchas medidas de tendencia pero para simplificar los datos quitamos las que no nos interesan y nos quedamos solo con las variables que queremos, aclarar que yhat es la predicción realizada por el modelo, graficamos las predicciones y las tendencias.

Forecast Dataframe (Selected Columns)			
	ds	yhat	yhat_lower
0	2016-01-02	194964.915172	167338.383769
1	2016-01-03	199957.613526	173183.870727
2	2016-01-04	193539.219600	165957.932395
3	2016-01-05	187420.374175	159079.239928
4	2016-01-06	192662.331365	165056.616871

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Una de las ventajas que tiene Prophet es que el gráfico es interactivo y podemos ampliar, seleccionar fechas y hacer análisis con estas características.



IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Trabajamos los datos para crear un dataframe en el que tengamos la predicción de ventas realizadas por el modelo y las ventas reales.

```
1 df_forecast = forecast[['ds', 'yhat']]
2 df_forecast
```

	ds	yhat	🔗
0	2016-01-02	194964.915172	
1	2016-01-03	199957.613526	
2	2016-01-04	193539.219600	
3	2016-01-05	187420.374175	
4	2016-01-06	192662.331365	
...	
2247	2022-02-26	155032.864099	
2248	2022-02-27	159078.899819	
2249	2022-02-28	152217.581831	
2250	2022-03-01	145979.722485	
2251	2022-03-02	151241.338380	

2252 rows × 2 columns

```
1 sales_p['ds'] = pd.to_datetime(sales_p['ds'], format='%d/%m/%Y')
2 sales_p
```

	ds	y	🔗
0	2016-01-02	262732	
1	2016-01-03	227291	
2	2016-01-04	230354	
3	2016-01-05	186906	
4	2016-01-06	186789	
...	
2276	2022-03-27	197510	
2277	2022-03-28	159691	
2278	2022-03-29	160170	
2279	2022-03-30	165753	
2280	2022-03-31	159130	

2281 rows × 2 columns

```
1 metric_df = pd.merge(df_forecast, sales_p, on= ['ds'])
2 metric_df
```

	ds	yhat	y	🔗
0	2016-01-02	194964.915172	262732	
1	2016-01-03	199957.613526	227291	
2	2016-01-04	193539.219600	230354	
3	2016-01-05	187420.374175	186906	
4	2016-01-06	192662.331365	186789	
...	
2247	2022-02-26	155032.864099	196646	
2248	2022-02-27	159078.899819	180042	
2249	2022-02-28	152217.581831	161654	
2250	2022-03-01	145979.722485	162774	
2251	2022-03-02	151241.338380	153738	

2252 rows × 3 columns

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

Sacamos las métricas de performance del modelo

```

1 r2_prophet = r2_score(metric_df.y, metric_df.yhat)
2 r2_prophet
0.2831266487995675

1 rmse_prophet = metrics.rmse(metric_df.y, metric_df.yhat)
2 rmse_prophet
21791.82782356921

1 mae_prophet = mean_absolute_error(metric_df.y, metric_df.yhat)
2 mae_prophet
17593.866522388656

1 mape_prophet = mean_absolute_percentage_error(metric_df.y, metric_df.yhat)
2 mape_prophet
10.11340853837484

```

Prophet nos permite ver como es el performance del modelo en base a varios parámetros y como es la evolución del performance de este mediante la cross validation

```

1 df_cv = cross_validation(
2     model=model,
3     # Periodo de entrenamiento inicial
4     initial='546 days',
5     # Longitud del periodo para el que realizamos el cross validation
6     period='90 days',
7     # Horizonte de la predicción
8     horizon = '31 days'
9 )
10
11 df_cv

```

INFO:fbprophet:Making 18 forecasts with cutoffs between 2017-09-23 00:00:00 and 2021-12-01 00:00:00

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2017-09-24	183407.529582	185789.087429	202375.808409	159001	2017-09-23
1	2017-09-25	196161.372514	177120.978097	214920.730147	207753	2017-09-23
2	2017-09-26	175250.234446	157109.140293	193070.314565	204467	2017-09-23
3	2017-09-27	158225.004173	138568.071588	174949.312117	180103	2017-09-23
4	2017-09-28	159384.396608	140728.424538	179380.255717	187757	2017-09-23
...
553	2021-12-28	197840.392963	171087.922939	223501.747062	187465	2021-12-01
554	2021-12-29	205288.185749	178811.922423	231844.705887	192557	2021-12-01
555	2021-12-30	203823.706055	178311.331807	229912.587438	180330	2021-12-01
556	2021-12-31	193846.090841	166255.758215	218519.838342	183470	2021-12-01
557	2022-01-01	204522.310631	177412.041329	230833.555562	275526	2021-12-01

558 rows × 6 columns

IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

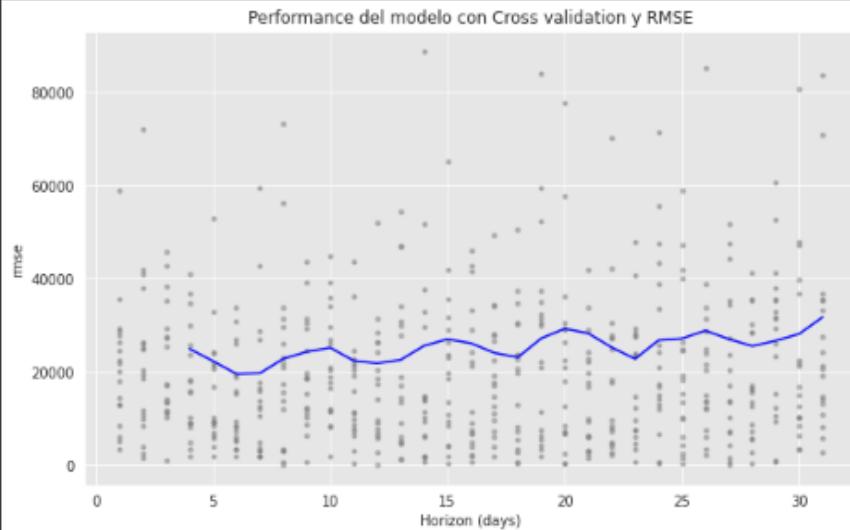
Podemos crear una tabla con la evolución de nuestro modelo según avanza el horizonte de nuestras predicciones que le marcamos anteriormente

	horizon	mse	rmse	mae	mape	mdape	coverage
0	4 days	6.903842e+08	26274.783049	21464.006185	0.120445	0.118079	0.703267
1	5 days	6.817168e+08	26109.708790	21593.007359	0.121733	0.118079	0.702359
2	6 days	5.928976e+08	24349.489143	19865.651182	0.112835	0.118079	0.735935
3	7 days	4.407176e+08	20993.274552	17192.468363	0.100673	0.101935	0.804900
4	8 days	4.107979e+08	20415.628786	16014.406346	0.092116	0.081188	0.823049
5	9 days	4.805258e+08	21920.899125	16838.412502	0.095948	0.082583	0.823956
6	10 days	5.628129e+08	23723.677807	18621.813281	0.105305	0.091516	0.774047
7	11 days	5.728267e+08	23933.798457	19042.422111	0.112478	0.096217	0.737750
8	12 days	4.674581e+08	21620.778817	18375.147849	0.108485	0.091478	0.754991
9	13 days	4.288743e+08	20709.279420	17251.292655	0.098906	0.085031	0.754083
10	14 days	4.535869e+08	21207.580133	16036.849226	0.093785	0.075629	0.787859
11	15 days	5.347054e+08	23125.643243	17453.938615	0.093181	0.073807	0.773140
12	16 days	5.074153e+08	24442.079555	18360.010931	0.097656	0.075629	0.788566
13	17 days	5.934876e+08	24381.600611	18570.254393	0.101009	0.083612	0.737750
14	18 days	5.371041e+08	23175.505881	18569.738673	0.104078	0.094188	0.720508
15	19 days	5.638514e+08	23741.343822	19856.478865	0.111321	0.121938	0.688784
16	20 days	7.276494e+08	26974.976925	22022.330418	0.118606	0.124231	0.666062
17	21 days	8.540606e+08	29224.314779	23161.896582	0.121082	0.122444	0.667877
18	22 days	7.407963e+08	27217.574107	20897.069045	0.110422	0.101991	0.752269
19	23 days	5.482771e+08	23415.318179	17881.853144	0.096678	0.093910	0.838475

Graficamos estas métricas

```
1 fig = plot_cross_validation_metric(df_cv, metric='rmse')
2 plt.title('Performance del modelo con Cross validation y RMSE')
```

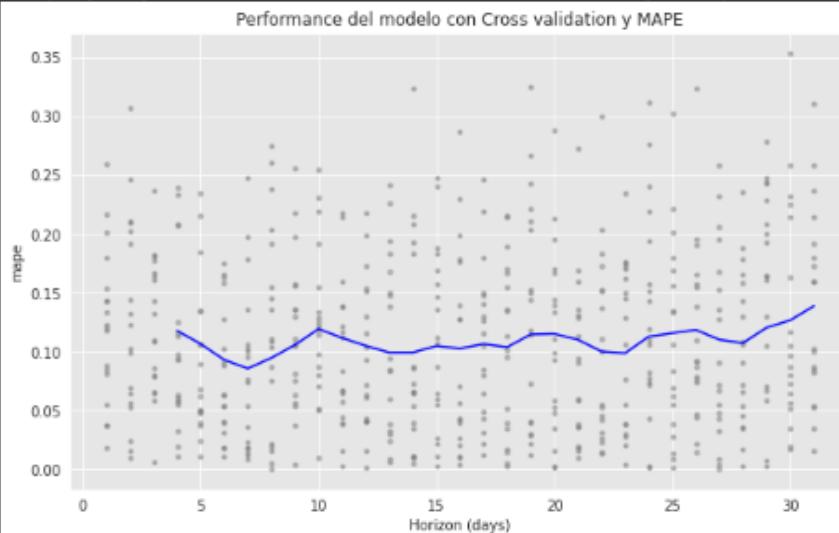
Text(0.5, 1.0, 'Performance del modelo con Cross validation y RMSE')



IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

```
1 fig = plot_cross_validation_metric(df_cv, metric='mape')
2 plt.title('Performance del modelo con Cross validation y MAPE')
```

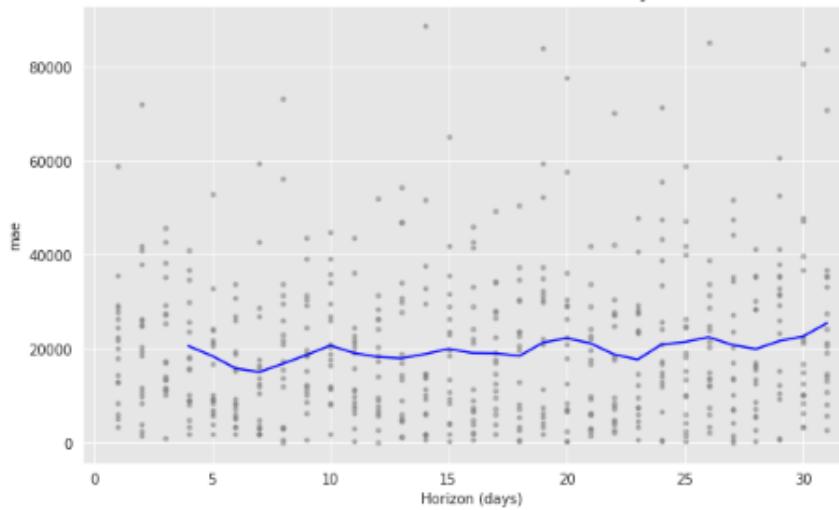
Text(0.5, 1.0, 'Performance del modelo con Cross validation y MAPE')



```
1 fig = plot_cross_validation_metric(df_cv, metric='mae')
2 plt.title('Performance del modelo con Cross validation y MAE')
```

Text(0.5, 1.0, 'Performance del modelo con Cross validation y MAE')

Performance del modelo con Cross validation y MAE



CONCLUSIONES DEL MODELO

01

Si bien el modelo da unas predicciones decentes dado los pocos parámetros y la sencillez del mismo es el que peor performance tiene

```
1 r2_prophet = r2_score(metric_df.y, metric_df.yhat)
2 r2_prophet
```

```
0.2831260487995675
```

```
1 rmse_prophet = metrics.rmse(metric_df.y, metric_df.yhat)
2 rmse_prophet
```

```
21791.02782356921
```

```
1 mae_prophet = mean_absolute_error(metric_df.y, metric_df.yhat)
2 mae_prophet
```

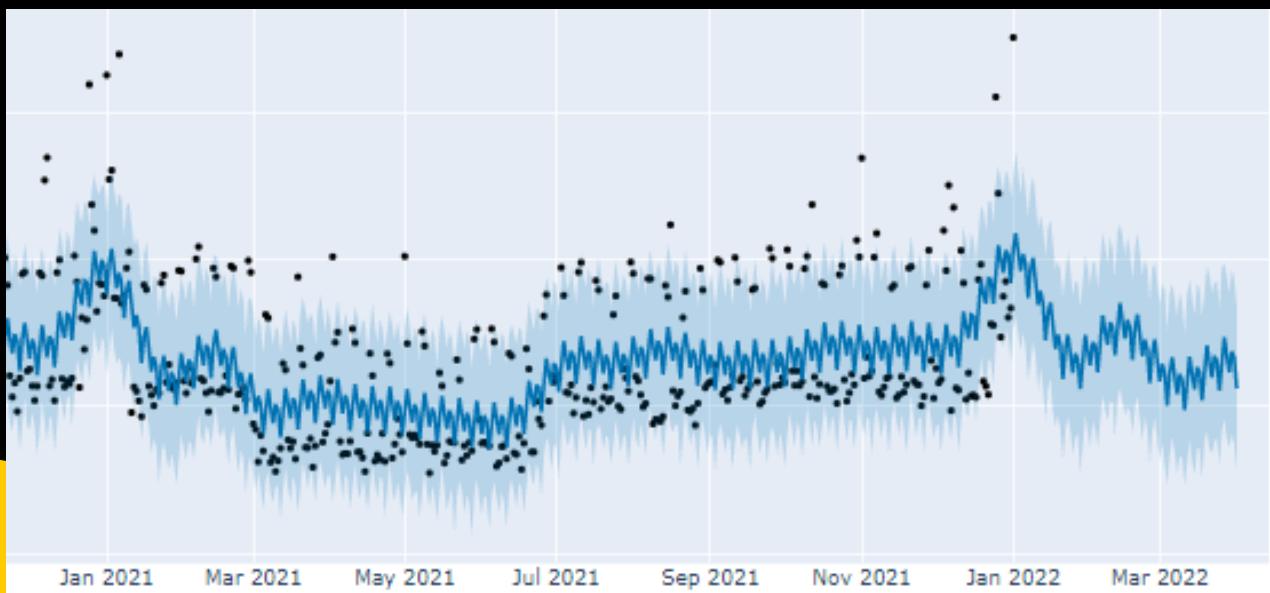
```
17593.866522388656
```

```
1 mape_prophet = mean_absolute_percentage_error(metric_df.y, metric_df.yhat)
2 mape_prophet
```

```
10.11340853837484
```

02

A diferencia del resto de modelos este se mantiene estable en las predicciones durante todo el tiempo



CONCLUSIONES FINALES

TRAS REALIZAR LAS PRUEBAS CON LOS TRES ALGORITMOS PLANTEADOS AL INICIO DEL PROYECTO PODEMOS CONCLUIR QUE TODOS ELLOS PUEDEN REALIZAR UNA BUENA PROYECCIÓN DE LAS VENTAS AUN TENIENDO pocas VARIABLES DE ENTRADA PARA REALIZAR Dicha PREDICCIÓN, POR LO QUE SI AÑADIMOS NUEVAS FEATURES A NUESTRO MODELO SEGURAMENTE TENGAMOS UNA MEJOR PERFORMANCE

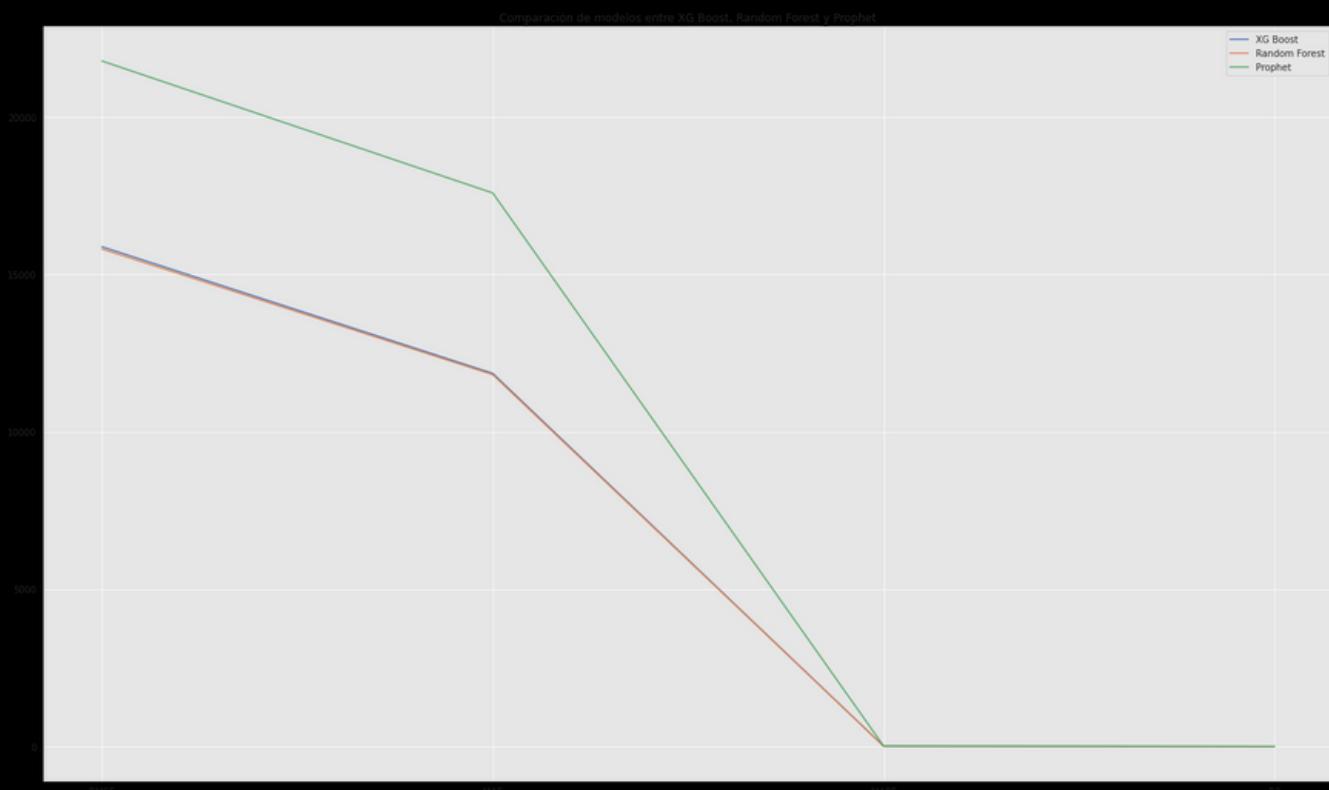
Si realizamos una tabla con todas las métricas de performance de los tres algoritmos podemos concluir que no existe mucha diferencia entre XGBoost y Random Forest y que Prophet es el que peor performance tiene pero es el más fácil de implementar y el que menos trabajo requiere, sería importante en un futuro probar Prophet intentando optimizar los parámetros y utilizando algún regresor extra que pudiese mejorar el performance del modelo.

También pudimos observar que los modelos más tradicionales basados en árboles de decisión funcionan perfectamente aun cuando entran con series de tiempo con muestras aleatorias.

	RMSE	MAE	MAPE	R2
XGBoost	15885.739699	11857.725246	6.782095	0.565438
Random Forest	15814.022220	11818.057135	6.658968	0.569353
Prophet	21791.027824	17593.866522	10.113409	0.283126

CONCLUSIONES FINALES

Ahora observemos la comparación de cada una de las métricas en un gráfico donde podemos observar que los dos modelos antes mencionados no tienen casi diferencia.



CONCLUSIONES FINALES

Ahora contestemos a las preguntas que nos hacíamos al inicio del proyecto

01

¿Podemos predecir las ventas de la compañía a futuro utilizando un modelo de Machine Learning?

Efectivamente nuestros modelos han demostrado que se pueden realizar predicciones de ventas bastante acertados con pocas variables de entrada por lo que si enriquecemos los datos con más variables podríamos llegar a obtener predicciones mucho más exactas aun.

02

¿Afecta la época del año a las ventas?
¿Hay patrones de estacionalidad?

Como pudimos observar durante los distintos análisis en el proyecto existe un claro patrón de estacionalidad en las ventas de los locales que coinciden con la época en que normalmente hay vacaciones y no es periodo lectivo.

03

¿Hay relación entre la venta y el periodo del mes?

En el análisis de correlaciones pudimos observar que hay una ligera tendencia al alza en las ventas de los locales entre la última semana del mes y la primera del siguiente lo cual coincide con las fechas en las que la gente suele cobrar su salario.

04

¿Afecta a las ventas que el día sea laborable o que haya algún festivo?

Finalmente también pudimos comprobar que tanto los fines de semana como los festivos hay un ligero aumento de las ventas en los locales.

LÍDER DEL PROYECTO

ANTONIO ELVIRA GARCÍA

Tengo 35 años soy español y actualmente resido entre España y Argentina y me desempeño como gerente de expansión de la marca Mostaza, teniendo la responsabilidad la apertura de nuevos locales de la marca en distintos países. También a lo largo de los años trabajé en otras empresas del sector como FIVE GUYS y McDonald's.

Mi formación principal fue en administración y operacional pero actualmente he incursionado en el mundo del análisis de datos lo que me lleva a cursar la carrera de Data Scientist en CoderHouse conocimientos que pretendo aplicar a mis responsabilidades actuales.

