

# Máster en Data Science (2020-2021)

Trabajo Fin de Máster

“Análisis de series temporales”

---

Jorge de Toro de Murga

Juan A. Álvarez Ortiz

---

**Tutor:** José Manuel Cuadra Troncoso

28/05/2021





## Agradecimientos

Después de un intenso período de casi 16 meses, en un curso presencial que hubo que adaptar debido a la pandemia, hoy es el día. Escribimos este apartado de agradecimientos para finalizar nuestro trabajo de fin de master. Ha sido un período de aprendizaje intenso, tanto en el campo profesional como en el personal.

Nos gustaría agradecer a nuestro tutor José Manuel Cuadra por su valiosa ayuda y apoyo durante nuestro trabajo y a nuestra empresa El Corte Inglés por darnos la oportunidad de formar parte de este master.

El Profesor Dr. José Manuel Cuadra Troncoso, Doctor en Ingeniería Informática, afirma haber dirigido el presente trabajo en el Departamento de Inteligencia Artificial de la Universidad Nacional de Educación a Distancia de Madrid, y que, a su juicio, reúne los requisitos de originalidad y rigor científico necesarios para ser presentado como trabajo fin de máster.

# Contenido

1	Introducción .....	9
1.1	Descripción del negocio .....	9
1.2	Descripción del problema .....	10
1.3	Motivación y objetivos .....	11
1.4	Software utilizado .....	12
1.5	Organización de la memoria .....	14
2	Estado del arte .....	17
2.1	Componentes de una serie temporal.....	18
2.2	Tipos de series temporales.....	18
2.3	Metodología de análisis .....	19
3	Descripción de los modelos utilizados .....	21
3.1	Regresión lineal .....	21
3.2	Modelo ARIMA .....	22
3.2.1	Descripción del modelo AR, I, MA.....	23
3.2.2	Etapas en la elaboración de un modelo ARIMA.....	23
3.2.2.1	Identificación.....	23
3.2.2.2	Estimación .....	24
3.2.2.3	Validación .....	25
3.2.2.4	Predicción.....	25
3.3	Modelo Prophet .....	26
3.3.1	Propiedades del modelo .....	26
3.3.2	Ventajas del modelo.....	26
3.3.3	Descripción del modelo.....	27
3.3.3.1	Componentes de tendencia .....	27
3.3.3.2	Componentes estacionales .....	28
3.3.3.3	Eventos excepcionales .....	28
3.4	Modelo XGBoost .....	29
3.4.1	Estrategia de predicción.....	30
4	Aplicación de modelo y resultados .....	31
4.1	Ánálisis exploratorio de los datos .....	31
4.1.1	Preprocesamiento de los datos.....	31
4.1.1.1	Fichero Venta diaria 152 .....	31
4.1.1.2	Fichero Promociones.....	33

4.1.2	Estacionalidad de la serie .....	34
4.2	Medidas de evaluación de los modelos .....	36
5	Elección y ajuste de modelos .....	39
5.1	Regresión lineal .....	39
5.1.1	Carga de ficheros y módulos .....	39
5.1.2	Aplicación y entrenamiento del modelo mensual .....	41
5.1.3	Aplicación y entrenamiento del modelo diario.....	43
5.2	Random Forest .....	46
5.2.1	Aplicación y entrenamiento del modelo mensual .....	46
5.2.2	Aplicación y entrenamiento del modelo diario.....	47
5.3	ARIMA.....	50
5.3.1	ARIMA conjunto de datos mensuales .....	50
5.3.2	ARIMA conjunto de datos diarios.....	57
5.4	Prophet.....	61
5.4.1	Organización del trabajo .....	61
5.4.1.1	Carga de las librerías necesarias .....	61
5.4.1.2	Carga de datos para realizar el experimento .....	62
5.4.1.3	Ajuste del modelo .....	62
5.4.1.3.1	Modelo Venta diaria.....	62
5.4.1.3.2	Modelo Venta Mensual.....	74
5.5	XGBOOST .....	82
5.5.1	Organización del trabajo .....	82
5.5.1.1	Carga de las librerías necesarias .....	82
5.5.1.2	Carga de datos para realizar el experimento .....	82
5.5.1.3	Ajuste del modelo .....	83
5.5.1.3.1	Modelo Venta Mensual.....	84
5.5.1.3.2	Modelo Venta Diaria .....	94
6	Conclusiones y trabajo futuro .....	96
7	Anexos .....	98
8	Bibliografía .....	99

# Índice de ilustraciones

Ilustración 1 Tabla Cifras de El Corte Inglés.....	9
Ilustración 2. Gráfico Ejemplo de organigrama departamental .....	10
Ilustración 3. Gráfico Venta en unidades por mes año, 2018,2019 y 2020.....	14
Ilustración 4 Gráfico Venta en unidades por día, años 2018,2019 y 2020.....	15
Ilustración 5 Gráfico Predicción diaria Linear Regression .....	15
Ilustración 6 Gráfico Predicción diaria Random Forest.....	16
Ilustración 7 Gráfico Predicción diaria ARIMA.....	16
Ilustración 8 Tabla tipo de datos Ventas .....	32
Ilustración 9 Tabla tipo de datos Ventas .....	32
Ilustración 10 Tabla Resumen de las variables.....	32
Ilustración 11. Tabla Características datos fichero promociones .....	33
Ilustración 12 Gráfico Venta en unidades por trimestre. PowerBi .....	34
Ilustración 13 Gráfico Venta en unidades por año y mes. PowerBi.....	35
Ilustración 14 Tabla Resumen conjunto de datos.....	40
Ilustración 15 Gráfico serie de ventas 2017-2020.....	40
Ilustración 16 Tabla meses de venta.....	41
Ilustración 17 Gráfico Venta real vs predicción modelo lineal mensual.....	42
Ilustración 18 Tabla diaria de venta .....	43
Ilustración 19 Gráfico Venta real vs predicción modelo lineal diario.....	44
Ilustración 20 Tabla RMSE modelo lineal diario .....	45
Ilustración 21 Gráfico Venta real vs Predicción RF mensual.....	47
Ilustración 22 Gráfico Venta real vs Predicción RF diario .....	48
Ilustración 23 Gráfico Venta real vs Predicción RF diario .....	49
Ilustración 24 Tabla RMSE RF diario y mensual .....	49
Ilustración 25 Tabla datos ajustados a ARIMA .....	51
Ilustración 26 Gráfico serie ventas mensuales 2017-2019 .....	51
Ilustración 27 Tabla resultados SARIMAX .....	53
Ilustración 28 Gráfico de predicción ARIMA mensual .....	53
Ilustración 29 Tabla resultados SARIMAX .....	54
Ilustración 30 Gráfico Venta real vs Predicción ARIMA mensual .....	55
Ilustración 31 Gráfico Predicción ARIMA 5 meses .....	56
Ilustración 32 Gráfico train, test y predicción ARIMA mensual.....	56
Ilustración 33 Gráfico serie temporal venta diaria .....	57
Ilustración 34 Tabla resultados SARIMAX diario.....	58
Ilustración 35 Tabla resultados ARIMA diario.....	58
Ilustración 36 Gráfico Venta real vs predicción ARIMA diario.....	59
Ilustración 37 Gráfico predicción 30 días ARIMA diario .....	60
Ilustración 38 RMSE ARIMA diario y mensual .....	60
Ilustración 39 Tabla test, train Prophet .....	63
Ilustración 40 Tabla dataframe Future .....	64
Ilustración 41 Tabla predicción 30 días Prophet diario .....	64

Ilustración 42 Tabla dataframe comp_df.....	64
Ilustración 43 Gráfico Venta real vs predicción año 2019 .....	65
Ilustración 44 Gráfico componentes serie temporal .....	66
Ilustración 45 Tabla Modelo multiplicativo 30 días Prophet diario .....	67
Ilustración 46 Gráfico Venta diaria real vs predicción año 2019 .....	67
Ilustración 47 Tabla performance metrics .....	68
Ilustración 48 Tabla Modelo aditivo 30 días Prophet diario .....	68
Ilustración 49 Gráfico Venta diaria real vs predicción año 2019 .....	69
Ilustración 50 Tabla Modelo aditivo 30 días Prophet diario .....	70
Ilustración 51 Gráfico Venta diaria real vs predicción año 2019 .....	70
Ilustración 52 Tabla festivos España .....	71
Ilustración 53 Gráfico Venta diaria real vs predicción año 2019 .....	71
Ilustración 54 Gráfico Venta diaria real vs predicción año 2019 .....	72
Ilustración 55 Tabla comparativa errores Prophet diario .....	73
Ilustración 56 Gráfico Venta diaria real vs predicción 31 días .....	73
Ilustración 57 Tabla Venta mensual 2017-2019 .....	74
Ilustración 58 Gráfico Predicción mensual Prophet .....	74
Ilustración 59 Gráfico componentes serie mensual Prophet.....	75
Ilustración 60 Gráfico predicción ajustada mensual Prophet.....	75
Ilustración 61 Gráfico modelo aditivo mensual Prophet.....	76
Ilustración 62 Tabla dataframe comp_df.....	77
Ilustración 63 Gráfico Venta mensual real vs predicción 10 meses.....	77
Ilustración 64 Gráfico modelo multiplicativo mensual Prophet .....	78
Ilustración 65 Gráfico Venta mensual real vs predicción 10 meses.....	78
Ilustración 66 Gráfico modelo multiplicativo ajustado mensual Prophet.....	79
Ilustración 67 Gráfico Venta mensual real vs predicción 10 meses.....	79
Ilustración 68 Gráfico modelo aditivo ajustado mensual Prophet.....	80
Ilustración 69 Gráfico Venta mensual real vs predicción 10 meses.....	80
Ilustración 70 Tabla comparativa RMSE Prophet diario .....	81
Ilustración 71 Tabla RMSE Prophet diario y mensual.....	81
Ilustración 72 Gráfico Score XGBoost mensual.....	87
Ilustración 73 Gráficos Learning rate XGBoost mensual.....	88
Ilustración 74 Gráfico max depth XGBoost mensual.....	89
Ilustración 75 Tabla GridSearchCV XGBoost mensual.....	93
Ilustración 76 Tabla hiperparametros XGBoost mensual.....	94
Ilustración 77 Tabla comparativa experimentos con XGBoost .....	95
Ilustración 78 Tabla comparativa mejores resultados por método .....	96

## Índice de ecuaciones

Ecuación 1 Secuencia de variables aleatorias .....	17
Ecuación 2 Regresión lineal.....	21
Ecuación 3 Método de los mínimos cuadrados.....	22
Ecuación 4 Combinación lineal de p valores .....	23
Ecuación 5 Modelo ARMA .....	24
Ecuación 6 Logaritmo de la función de verosimilitud.....	24
Ecuación 7 Modelo Prophet .....	27
Ecuación 8 Modelo Prophet tasa de crecimiento.....	27
Ecuación 9 Serie de Fournier .....	28
Ecuación 10 Error cuadrático medio.....	36
Ecuación 11 Error absoluto medio .....	36
Ecuación 12 Error medio de porcentaje absoluto .....	37
Ecuación 13 Error porcentual absoluto medio .....	37



# 1 Introducción

El trabajo presente tiene como objetivo aplicar los conocimientos adquiridos durante el Máster de Data Science como culminación de este. Para ello se ha seleccionado un conjunto de datos al que se ha realizado un estudio estadístico del mismo y como principal objetivo se han aplicado distintas técnicas de estimación propias de las series temporales. En los apartados siguientes se desarrollarán el conjunto de apartados de los que consta dicho trabajo con el fin de describir el problema seleccionado, desarrollar el mismo y llegar a unas conclusiones.

## 1.1 Descripción del negocio



Ilustración 1 Tabla Cifras de El Corte Inglés

Nosotros, dentro de las diferentes empresas de Grandes Almacenes, nos vamos a centrar en el que mayor volumen de negocio ocupa: El Corte Inglés. Y, aunque tenemos presencia en Portugal, nos centraremos únicamente en territorio peninsular.

Dentro de esta empresa existen muchas divisiones con negocios diferentes. Nosotros nos centraremos en el departamento que más factura dentro de una de las divisiones con más peso de venta de la compañía: La división 21, Electrónica. Y concretamente en el departamento 152: Informática.

La división de Electrónica factura al año unos 1.400 millones de euros aprox. Dentro de esa cifra total el departamento de informática representa más del 33% del peso de la venta de la división.

Para hacernos una idea de la distribución representamos un pequeño organigrama de El Corte Inglés.

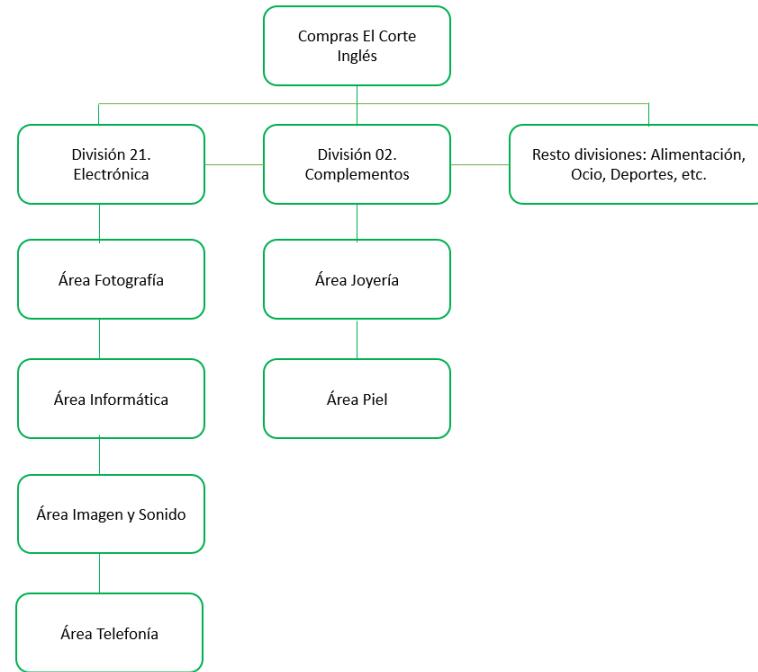


Ilustración 2. Gráfico Ejemplo de organigrama departamental

## 1.2 Descripción del problema

En el sector retail, la previsión de compras, puede ser uno de los objetivos más importantes a la hora de tener éxito o, por el contrario, incurrir en un estrepitoso fracaso. La gestión de la mercancía para que ésta esté en el lugar preciso, en el momento preciso y con el precio correcto es sin duda unos de los factores decisivos a la hora de atraer al cliente.

Asimismo, una adecuada previsión en la compra del producto genera una reducción en los costes, tanto de almacenamiento, como logísticos o como los que se incurre por la obsolescencia del mismo.

Para poder realizar una previsión de compras es necesario conocer previamente la demanda futura, es decir, las unidades de producto que se van a vender. Es ahí donde haremos hincapié en este proyecto. Realizaremos una previsión de ventas que nos permita conocer de la manera más acertada posible, el comportamiento de la demanda para poder conocer de antemano las compras necesarias para cubrirla.

A la hora de prever series temporales lineales los modelos más utilizados son aquellos métodos que incluyen suavizado temporal, descomposición en series de tiempo, ARIMA... Sin embargo, si las series temporales no son lineales estos modelos anteriores presentan ciertas limitaciones siendo necesarios métodos que sean capaces de obtener relaciones lineales y no lineales entre los datos como pueden ser, entre otros, las redes neuronales, los algoritmos de boosting o el modelo Prophet.

## 1.3 Motivación y objetivos

El objetivo final de este Trabajo de Fin de Máster es el de aplicar diferentes técnicas que nos permitan predecir las ventas de mercancía dentro de una empresa del sector retail, como lo es El Corte Inglés, para así poder conseguir la consecución de los objetivos de venta y compra de la manera más eficiente posible.

Para el desarrollo de este proceso contaremos con unos datos obtenidos de la división de Electrónica de El Corte Inglés, específicamente del departamento de Informática.

Mediante la comparación de los resultados obtenidos en los diferentes modelos obtendremos aquel que mejor se adapta a nuestro problema utilizando, para ello, diferentes medidas de precisión.

Los datos existentes en la base de datos que se va a usar han sido obtenidos mediante un datamart propio de El Corte Inglés. Se ha generado un dataset para los datos de venta semanales de un periodo de cuatro años (2017 a 2020) de las diferentes categorías de producto.

Los datos se desglosan en:

- **Empresa:** empresa única 001. El Corte Inglés. Engloba únicamente datos de las tiendas de península y canarias.
- **Uneco:** Número departamental al que pertenece el departamento de informática.
- **Departamento:** departamento de Informática el cuál se estudia para llevar a cabo este proyecto.
- **Medidas de tiempo (año, mes y semana):** se han usado diferentes momentos en el tiempo para poder englobar los datos. Así, el objetivo es poder llegar a predecir las compras de treinta y un días.
- **Medidas de venta (Unidades, Importe Coste, Importe de Venta):** Se obtiene del datamart diferentes indicadores de venta de cada producto.

Además, se añade información acerca de los períodos promocionales que más afectan a la venta de este departamento.

A partir de estos datos, el objetivo que nos ocupa es el de predecir la venta diaria del próximo mes y la venta mensual de los próximos 10 meses.

## 1.4 Software utilizado

Para realizar el estudio de este proyecto, utilizaremos dos de los lenguajes más extendidos entre la comunidad de científicos de datos: **R** y **Python**. Son, además, los lenguajes aprendidos en el curso y aquellos que más documentación tienen para la realización de series temporales.

**R:** (<https://cran.r-project.org/>) creado por Ross Ihaka y Robert Gentleman en 1995, es un lenguaje de programación enfocado en el análisis de datos, la aplicación de métodos estadísticos y visualización gráfica. Además, de estar presente en el mundo de la investigación y universidades, se ha hecho hueco en el mundo empresarial.

Para este trabajo se ha utilizado el entorno de desarrollo R-Studio, que simplifica mucho la creación de código, su posterior testeo y depuración. Adicionalmente, esta herramienta presenta un editor de texto, una consola, un sistema de depurado, una gestión de variables y un visualizador de imágenes y gráficos.

**Python:** que fue creado por Guido Van Rossum en 1991, está inspirado en los lenguajes de programación C, Modula-3 y ABC. Es un lenguaje de programación potente y fácil de aprender orientado a objetos y con estructuras de datos de alto nivel. Lo que le ha llevado a ser uno de los lenguajes de programación más utilizados actualmente.

Cuenta con librerías muy extensas orientadas al análisis de datos.

- **Scipy:** colección de código abierto para matemáticas, ciencia e ingeniería. Numpy, Matplotlib y pandas son librerías que trabajan bajo el paraguas de Scipy.
- **Numpy:** librería que añade capacidades de cómputo científicas como arrays o álgebra lineal entre otros.
- **Pandas:** Especializada en el manejo y análisis de estructuras de datos de una dimensión, dos dimensiones (dataframes) y tres dimensiones(cubos).
- **Scikit-learn:** librería especializada en proporcionar métodos de aprendizaje automático.
- **Matplotlib:** librería perfecta para trabajar con gráficos.
- **Statmodels:** librería que ayuda con la exploración de datos, la estimación de modelos o la realización de test estadísticos.

Para la integración de ambos lenguajes y proporcionar más simplicidad al proyecto, se ha usado el software Anaconda(<https://www.anaconda.com/>). Este software permite de manera sencilla la instalación de los diferentes paquetes necesarios para trabajar con Python y R, así como la instalación de las herramientas de RStudio, Spyder que son editores de código especializados en estos lenguajes.

Asimismo, Anaconda tiene integrada la interfaz web Jupyter Notebook (<http://jupyter.org/>), que permite crear y ejecutar código tanto en lenguaje R como Python de forma ordenada y mostrando en el propio notebook las salidas que devuelve el código. Las salidas de ese código serán las que vayamos incluyendo en este documento.

**PowerBi** (<https://powerbi.microsoft.com/>): es una plataforma de Microsoft que permite conectar, modelar y visualizar datos de forma sencilla mediante la creación de informes personalizados. Microsoft ha sido galardonado por decimocuarto año consecutivo como líder en el informe Gartner Magic Quadrant sobre plataformas de análisis y business intelligence de 2021.

Aunque la herramienta es de pago, permite obtener una versión gratuita de escritorio dónde podemos crear nuestros dashboards y conectar nuestros datos. Sin embargo, es con la versión dónde gana sentido ya que podemos compartir y visualizar en la nube todos los informes que creemos desde cualquier dispositivo.

Dentro de PowerBi podemos hacer cosas como:

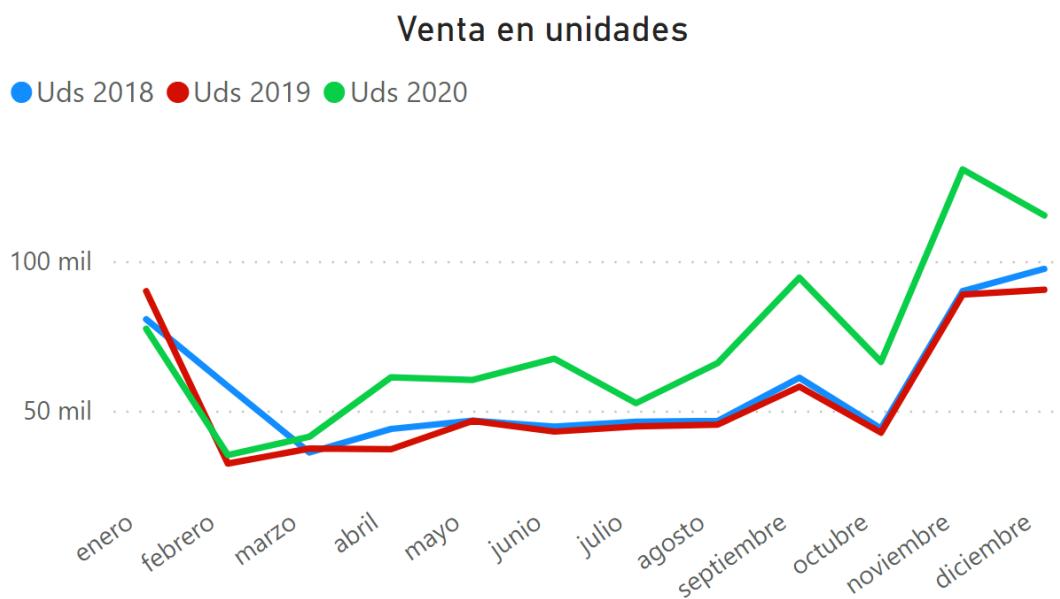
- Creación de Dashboards.
- Agregar objetos y vistas a paneles.
- Desplegar filtros que afecten a todo el informe o solo a una parte o gráfico en concreto
- Experimentar con diferentes formatos y diseño.
- Hacer que el tablero sea interactivo.

Hemos querido usar esta herramienta ya que, a la hora de crear gráficos, ver líneas temporales o conectar los datos es una de las herramientas más completas y fáciles de interpretar. Su principal ventaja es la cantidad de información que se puede obtener después de diseñar fácilmente unos pocos gráficos y tablas conectadas a conjuntos de datos que pueden ser de unos miles de líneas o de millones.

## 1.5 Organización de la memoria

En el presente trabajo, uno de los problemas que nos encontramos fue que uno de los años del conjunto de datos, concretamente el 2020, estaba muy influido por la crisis de la Covid-19, lo que hacía que los datos fueran muy diferentes a los de años anteriores y difícilmente replicables en años posteriores. Ese año no nos valía como dato histórico, ya que la finalidad del trabajo es predecir unas ventas con un comportamiento acorde a un año normal. Por ello, desecharmos ese año y cogimos datos de los tres años inmediatamente posteriores (2017,2018,2019).

En el siguiente gráfico podemos observar el incremento de ventas en el año 2020 respecto a los anteriores.



*Ilustración 3. Gráfico Venta en unidades por mes año, 2018,2019 y 2020.*

Asimismo, se realizaron estudios para la predicción diaria de la venta con algoritmos de Regresión Lineal, random Forest y ARIMA. Sin embargo, en un mercado muy influido por el efecto calendario (cambia mucho el comportamiento de compra de un cliente un domingo o un lunes que el de un viernes), los datos entre un día y ese mismo día el año anterior, o si ese día cambiaba de semana hacían la predicción de datos muy difícil y con muchos errores.

En el siguiente gráfico podemos observar los grandes picos que hay en ciertos días que están influidos por el calendario promocional. Esos días, no coinciden entre un año y el siguiente. Si bien pueden coincidir en el mismo mes tampoco es seguro que eso vaya a pasar. Así, por ejemplo, hablamos de fechas tan importantes como puede ser Black Friday, Días Sin IVA, la Semana de Internet...

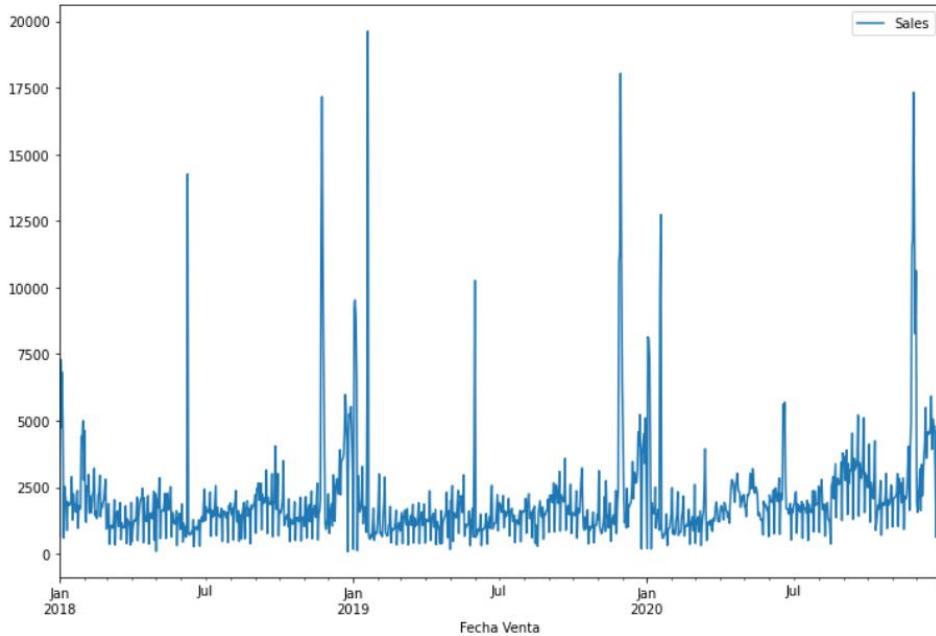


Ilustración 4 Gráfico Venta en unidades por día, años 2018, 2019 y 2020.

Antes de desechar el año 2020, hicimos una predicción diaria de treinta días con tres modelos para poder observar la bondad de los mismos, sabiendo las diferencias existentes entre los distintos días de la semana y el efecto Covid-19 del año 2020.

A continuación, vemos las tres predicciones que arrojaron los modelos:

#### Modelo de regresión lineal:

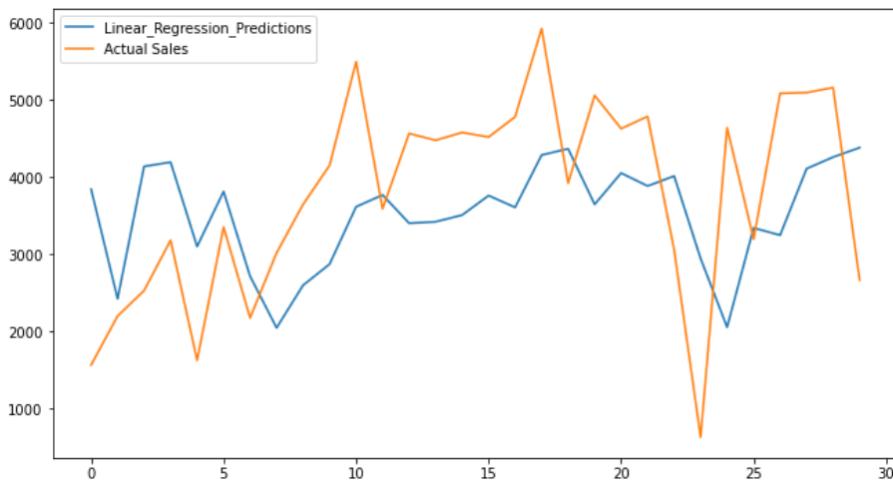


Ilustración 5 Gráfico Predicción diaria Linear Regression

Modelo Random Forest:



Ilustración 6 Gráfico Predicción diaria Random Forest

Modelo ARIMA:

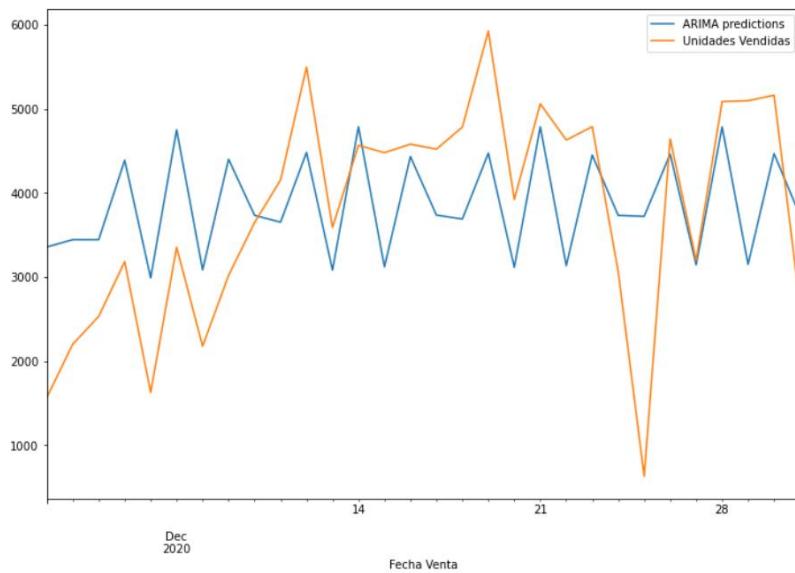


Ilustración 7 Gráfico Predicción diaria ARIMA

En los tres modelos, se consiguen prever ciertos picos al alza y a la baja en la serie temporal. Sin embargo, las predicciones se adelantan o retrasan ya sean picos al alza o a la baja.

Con los resultados obtenidos decidimos que lo mejor sería descartar el año 2020 y usar solo los años 2017, 2018 y 2019. Además, realizaremos el ejercicio, tanto con los datos de venta diaria como con los datos agrupando las ventas por mes y ver qué predicciones obtendríamos con estos modelos. El problema de hacer la agrupación de venta mensual era que el conjunto de datos se reducía considerablemente (tenemos datos de 36 meses), lo que en algoritmos como XGBoost podía suponer un problema.

## 2 Estado del arte

El presente Trabajo de Fin de Máster va a estar orientado al uso de las series temporales, la cuáles poseen una gran importancia en el campo económico; de hecho, constituyen la mayor parte del material estadístico con el que trabajan los economistas.

El objetivo fundamental del estudio de las series temporales es conocer el comportamiento de una variable a través del tiempo, para en función de ese comportamiento, poder predecir su comportamiento futuro mediante predicciones, bajo el supuesto de que no haya cambios estructurales, y reducir así la incertidumbre existente, mediante la aplicación de un determinado modelo calculado previamente.

Empecemos definiendo una serie temporal:

Una serie temporal es una sucesión de observaciones de una variable realizadas a intervalos regulares de tiempo. En función de la medida de la variable considerada tendremos series temporales discretas o continuas (dependiendo del intervalo de tiempo considerado para su medición), de flujo o stock (dependiendo de si determinamos un periodo de tiempo o una fecha determinada). Si profundizamos más, podemos, en función de la medida empleada, encontrar series temporales en euros, litros, centímetros... o si nos centramos en la periodicidad podemos encontrar series temporales semanales, trimestrales, anuales, etc.

Por lo tanto, una serie temporal podría definirse como un conjunto de observaciones ordenadas de forma cronológica, dónde el pasado de la variable influye en su futuro, no siendo posible conocer su valor futuro con una certeza real. Existe una ecuación matemática que permite representar una serie temporal como una secuencia de variables aleatorias:

$$X_i(t) \forall t \in \mathbb{R}$$

*Ecuación 1 Secuencia de variables aleatorias*

Dónde  $X_i$  es la secuencia de variables aleatorias y  $t$  es el valor temporal que define la serie.

## 2.1 Componentes de una serie temporal

Una serie temporal tiene diversos componentes que la conforman. Entre esos componentes los más importantes son:

- **Tendencia:** cambio a largo plazo que se produce en relación al nivel medio, o el cambio a largo plazo de la medida. La tendencia se identifica con un movimiento suave de la serie a largo plazo
- **Efecto estacional:** la mayoría de las series presentan cierta periodicidad. Esos períodos de cualquier unidad de tiempo pueden influir en el comportamiento de la serie. Estos tipos de efectos se pueden llegar a entender y medir explícitamente o incluso eliminar del conjunto de datos, desestacionalizando la serie original.
- **Homocedasticidad:** cuando la varianza de la serie es siempre constante y no depende de  $t$  se dice que es homocedástica. En caso contrario, la serie será heterocedástica (la variabilidad de la serie aumenta o disminuye a lo largo del tiempo).
- **Ciclo:** componente similar a la estacionalidad, sin embargo, la fluctuación alrededor de la tendencia es de una duración irregular.
- **Componente Aleatoria:** residuos aleatorios que no responden a un comportamiento sistemático que son imposibles de predecir y quedan fuera de la tendencia, estacionalidad o ciclos de la serie.

## 2.2 Tipos de series temporales

Las series temporales se pueden clasificar en:

- **Estacionarias:** Una serie es estacionaria cuando es estable, es decir, cuando su media y su varianza son constantes a lo largo del tiempo. Cuando su media es constante decimos que es estacionaria con respecto a la media. Si su varianza es constante, decimos que es estacionaria en varianza.
- **No estacionarias:** son series en las cuales la media y/o variabilidad cambian a lo largo del tiempo. Estos cambios determinan una tendencia a crecer o decrecer a largo plazo, por lo que la serie no oscila alrededor de un valor constante.

Es importante conocer que la estacionalidad (en media y varianza) es un requisito indispensable para poder aplicar modelos paramétricos de análisis y predicción de serie de datos. Con series estacionarias se pueden obtener predicciones de forma sencilla debido a que con media constante se puede realizar estimación con todo el conjunto de datos y utilizar esta estimación para predecir una nueva observación. Se podrá, además, generar intervalos de confianza para las predicciones.

En función de las variables observadas las series temporales se pueden clasificar en:

- **Univariantes:** la serie temporal es un conjunto de observaciones sobre una única característica o variable.
- **Multivariantes:** la serie temporal es un conjunto de observaciones de varias variables.

## 2.3 Metodología de análisis

A la hora de analizar una serie temporal se pueden seguir estos pasos:

1. **Definición del problema:** una vez identificado y entendido el problema, el siguiente paso es identificar las variables y los datos necesarios para resolver dicho problema.
2. **Recopilación de la información:** es importante contar con una muestra representativa que permita conocer la variable a predecir de la mejor forma posible para que los resultados sean lo suficientemente fiables.
3. **Preprocesamiento de los datos:** es necesario un primer análisis exploratorio de los datos para conocerlos mejor, y poder identificar patrones, estacionalidades, ciclos, errores, etc.
4. **Elección y ajuste de los modelos:** No existe un modelo único, si no que depende mucho del caso de estudio. En función del tamaño de la muestra, de los tipos de datos, la correlación entre las variables... Entre los modelos a usar los más utilizados son modelo de Box-Jenkins, ARIMA, Prophet y redes neuronales.
5. **Uso y evaluación del modelo elegido:** Una vez comparados los modelos y elegido el que mejor se adapta al caso en cuestión podemos evaluar la efectividad de dicho modelo.



## 3 Descripción de los modelos utilizados

Los modelos que se van a desarrollar en este caso serán:

- Regresión lineal
- Random Forest
- Modelo ARIMA
- Modelo Prophet
- Modelo basado XGBoost

### 3.1 Regresión lineal

La regresión lineal es una técnica paramétrica de machine learning. La fórmula de regresión lineal con una variable es:

$$Y = wX + b$$

*Ecuación 2 Regresión lineal*

El aprendizaje consistirá en encontrar cuáles son los mejores parámetros para nuestro conjunto de datos. Estos coeficientes serán aquellos que minimicen alguna medida de error, como por ejemplo el error cuadrático medio.

Para que esta técnica pueda aprender de nuestros datos, tendremos que proporcionar los resultados  $y$ , en forma de vector de  $M$  elementos, y los datos de entrada,  $x$ , en forma de matriz. Cada fila  $M$  será un dato (por ejemplo, una fecha), y cada columna  $N$  será un atributo relevante (como, por ejemplo, cuantos productos se venden en esa fecha).

En general, los problemas de machine learning suelen tener muchas variables. Por tanto, podríamos expandir la expresión de regresión lineal para múltiples variables. Quedaría así:

Si tenemos un dato con  $N$  variables (lo llamaremos  $X$ ) y con múltiples parámetros (los llamaremos  $W$ ) tendremos:

$$X = [x_0, x_1, x_2, \dots, x_n]$$

$$W = [w_0, w_1, w_2, \dots, w_n]$$

Si hacemos que  $x_0=1$ ,  $w_0=b$  entonces  $y=b+wx=w_0x_0+w_1x_1$

Por tanto, el proceso de aprendizaje sería el de averiguar qué parámetros  $W$  minimizan el error cuadrático medio entre los resultados reales y los estimados.

Un método, que no es el único, que podemos usar para ello es el conocido como El método de los mínimos cuadrados. Este método proporciona una solución analítica. En la práctica hay librerías numéricas que calculan por nosotros esto de forma automática. Aunque dependiendo de la cantidad de datos y atributos que tengamos puede ser un proceso muy costoso computacionalmente hablando. La expresión del método de los mínimos cuadrados es:

$$\hat{W} = (X^T X)^{-1} X^T y$$

*Ecuación 3 Método de los mínimos cuadrados.*

Otra forma para resolver el problema de la regresión lineal puede ser el gradiente descendiente. Este método no solo sirve para resolver regresiones lineales, también es muy útil para el aprendizaje automático de redes neuronales y Deep learning.

## 3.2 Modelo ARIMA

El modelo ARIMA es una metodología econométrica basada en modelos dinámicos que utiliza datos de series temporales. La metodología utilizada fue descrita inicialmente por George E. Pelham Box y por Gwilym M. Jenkins en 1970.

Para trabajar con modelos ARIMA hay que tener en cuenta ciertas consideraciones generales:

- a) **Proceso estocástico:** sucesión de variables aleatorias que dependen de un parámetro, en el caso de las series temporales ese parámetro es el tiempo.
- b) **Ruido blanco:** sucesión de variables aleatorias que se caracterizan por tener una esperanza constante e igual a cero, igual varianza y con covarianza cero (independientes a lo largo del tiempo).
- c) **Sendero aleatorio:** proceso estocástico que se caracteriza porque su primera diferencia es un ruido blanco.
- d) **Estacionario:** un proceso estocástico será estacionario si se cumple que su media y su varianza son constantes en cualquier periodo de tiempo y la covarianza de ambas solo depende del lapso de tiempo que transcurre entre ellas.

### 3.2.1 Descripción del modelo AR, I, MA

El modelo ARIMA se puede componer de tres submodelos principales:

- Modelo autorregresivo Ar(p)
- Modelo de medias móviles MA(q)
- Modelo integrado I(d)

### 3.2.2 Etapas en la elaboración de un modelo ARIMA

#### 3.2.2.1 Identificación

Para identificar el proceso ARIMA generado por una serie temporal determinada los datos deben ser estacionarios (sin tendencias) y no pueden presentar fluctuaciones. Si la dispersión no es constante la serie no es estacionaria en varianza y habría que transformarla, siendo la transformación logarítmica la más utilizada.

Una vez la serie es estacionaria hay que obtener las funciones de correlación simple y parcial muestrales para establecer el proceso ARIMA(p,d,q) más adecuado.

Para identificar un proceso autorregresivo de orden p, como indica Jenkis en su libro (Time Series Análisis: Forecasting and Control): "es necesario que la función de autocorrelación simple no se anule y decrezca de forma exponencial o sinusoidal hacia cero, mientras que, la función de autocorrelación parcial tenga solo p coeficientes distintos de cero".

Matemáticamente se puede describir esta combinación lineal de p valores mediante la siguiente ecuación:

$$Y_t = \varphi_1 Y_{t-1} + \dots + \varphi_p Y_{t-p} + a_t \text{ o bien } (1 - \varphi_1 L - \dots - \varphi_p L^p) Y_t = a$$

Ecuación 4 Combinación lineal de p valores

Dónde  $a_t$  es un ruido blanco y  $L$  es el operador de retardos.

Por el contrario, si el proceso es de medias móviles de orden p, el comportamiento de las funciones de autocorrelación simple y parcial será el contrario de los procesos autorregresivos.

Mientras que los procesos autorregresivos son siempre invertibles y es necesario que las p raíces del polinomio estén fuera del círculo unidad para que sean estacionarios. En los procesos de medias móviles son siempre estacionarios y para que sea invertible las p raíces del polinomio deben estar fuera del círculo unidad.

En el caso de que el proceso sea un ARMA, ninguna de las funciones de autocorrelación se anula y se encarece su identificación que si fuera un proceso AR o un MA. Los procesos ARMA se obtienen de combinar los procesos AR y MA. Su expresión general es:

$$Y_t = \varphi_1 Y_{t-1} + \dots + \varphi_p Y_{t-p} + a_t + \theta_1 a_{t-1} + \dots + \theta_q a_{t-q} \quad \text{o bien} \quad (1 - \varphi_1 L - \dots - \varphi_p L^p) Y_t = (1 - \theta_1 L - \dots - \theta_q L^q) a_t$$

Ecuación 5 Modelo ARMA

Dónde  $a_t$  es ruido blanco. En el caso de los procesos ARMA si las  $p$  raíces del polinomio de la parte autorregresiva están fuera del círculo unidad serán estacionarios. Y será invertible siempre que las  $q$  raíces del polinomio de las medias móviles estén también fuera del círculo unidad.

### 3.2.2.2 Estimación

Una vez que se ha identificado el modelo es necesario estimar de los parámetros de los que depende. Si se parte del proceso general de que la serie temporal es estacionaria e invertible y ha sido generada por un ARIMA(p,d,q) entonces la transformación estacionaria  $W_t = (1 - L)dY_t$  será un ARMA(p,q) dónde los parámetros a estimar son:  $\varphi_1, \dots, \varphi_p, \theta_1, \dots, \theta_q$ .

Asumiendo que la distribución de la serie es una Normal, podemos estimar estos parámetros por máxima verosimilitud siendo el logaritmo de la función de verosimilitud:

$$\text{Log } L = -\frac{T}{2} \log(2\pi) - \frac{(T-1)}{2} \sigma_a^2 - \frac{1}{2\sigma_a^2} \sum_{t=2}^T \left[ y_t - E \left[ \frac{w_t}{w_{t-1}} \right] \right]^2 - \frac{1}{2} \log(\sigma^2) - \frac{(w_1 - \mu)^2}{2\sigma^2}$$

Ecuación 6 Logaritmo de la función de verosimilitud

dónde  $T$  es el número de observaciones;  $\sigma_a^2$  es la varianza del ruido,  $\mu$  es la media de la serie y  $\sigma^2$  su varianza.

### 3.2.2.3 Validación

Una vez estimado el modelo tenemos que comprobar que se ajusta de forma adecuada a los datos observados. Para ello se suele realizar un análisis de los parámetros estimados, de los residuos, de la bondad del ajuste y de estabilidad.

- a) **Análisis de los parámetros estimados:** lo primero que hay que analizar es si los parámetros estimados son estadísticamente significativos. Si no fuera significativo habría que eliminarlo del modelo. Asimismo, hay que verificar que los parámetros estimados cumplen las condiciones de estacionariedad e invertibilidad. Haremos esto mediante los contrastes de hipótesis.

**$H_0$ : Hipótesis Nula** es la hipótesis sobre la que se desea decidir.

**$H_1$ : Hipótesis Alternativa** es la hipótesis que se acepta, si se rechaza la hipótesis nula. Generalmente la hipótesis alternativa es la negación de la hipótesis nula.

Definiremos los siguientes tipos de error:

**Error de tipo I (error  $\alpha$ ):** Rechazar  $H_0$  cuando es cierta.

$$P[\text{rechazar } H_0 / H_0 \text{ es cierta}] = \alpha; 0 \leq \alpha \leq 1$$

**Error de tipo II (error  $\beta$ ):** Aceptar  $H_0$  cuando es falsa.

$$P[\text{Aceptar } H_0 / H_0 \text{ es falsa}] = \beta; 0 \leq \beta \leq 1$$

- b) **Análisis de los residuos:** en el modelo ARIMA se supone que las perturbaciones aleatorias son ruido blanco. Sin embargo, al ser inobservables necesitamos comprobar si lo son o no. Para ello existen varias herramientas de comprobación: el gráfico de los residuos, la función de autocorrelación simple y parcial, el contraste de Portmanteau, entre otros.
- c) **Análisis de la bondad del ajuste:** Para observar la adecuación que existe entre la serie estimada y la real podemos usar el coeficiente de determinación. Cuanto más cercano a uno esté el coeficiente mejor será el modelo ajustado.
- d) **Análisis de la estabilidad:** mediante este análisis podemos saber si el modelo ARIMA estimado para el periodo muestral es también estable para períodos futuros. Para este análisis podemos usar el test de estabilidad de Chow.

### 3.2.2.4 Predicción

Una vez se ha obtenido el modelo ARIMA con sus coeficientes correspondientes, se pueden realizar predicciones de la distribución para valores futuros. La mejor predicción será aquella en la que se obtenga el menor error cuadrático medio, es decir, aquella que más se parezca a la realidad.

Para el cálculo de este error se pueden emplear dos métodos:

- **Puntual:** se calculará el valor esperado de la variable en el periodo futuro  $T+1$  condicionado al conjunto de información disponible hasta el periodo  $T$ .
- **Intervalo:** teniendo un intervalo de confianza del 95%, sumaremos y restaremos a la predicción puntual la desviación típica del error de predicción, multiplicada por el valor tabulado para el intervalo de confianza.

### 3.3 Modelo Prophet

Prophet es una biblioteca de código abierto desarrollada por Facebook y que fue diseñada para el pronóstico automático de datos de series de tiempo univariadas. Uno de sus puntos fuertes es que es fácil de usar y está diseñada para encontrar de forma automática un buen conjunto de hiperparámetros del modelo para poder realizar pronósticos hábiles para datos con tendencias y estructura estacional. El modelo está implementado para poder ser usado con R y Python.

#### 3.3.1 Propiedades del modelo

Para el correcto uso del modelo hay que tener en cuenta ciertos aspectos:

- El funcionamiento del modelo incrementa su eficiencia para series temporales que poseen fuertes efectos estacionales y varias temporadas de datos históricos.
- Es robusto a valores perdidos
- Maneja bien los valores atípicos.
- Soporta bien los cambios en las tendencias de históricos

El modelo descompondrá los datos en:

- I. **Tendencia:** no periódicas y sistemáticas.
- II. **Efectos estacionales:** modelará en función de períodos semanales o anuales. En la semanal tendrá en cuenta el día para la modelación del valor. En la anual tendrá en cuenta el mes.
- III. **Festivos y eventos relevantes:** Los períodos festivos (vacaciones, festivos, etc.) y los eventos que influyen en la venta (Black Friday, promociones, etc.) tienen un gran impacto en las series temporales. Por norma general, no siguen un patrón periódico, por tanto, no pueden modelarse de la misma forma que otros componentes.

#### 3.3.2 Ventajas del modelo

El modelo Prophet presenta una serie de ventajas:

- I. **Entrenamiento rápido:** el modelo nos da la capacidad de explorar desde distintos puntos de forma rápida y sencilla debido a su rapidez de cálculo.
- II. **Flexibilidad:** Existe la posibilidad de incluir diferentes períodos con estacionalidades diversas que permitirán estudiar diferentes tendencias al analista.
- III. **No equiespaciados:** No es requerida la estimación de los valores perdidos e intermedios. Asimismo, no es necesaria la homogeneidad entre el valor actual( $t$ ) y el valor futuro( $t+1$ )
- IV. **Fácil interpretación:** su fácil interpretación ayuda al analista a poder varias los valores de los parámetros y ver diferencias significativas de forma rápida.

### 3.3.3 Descripción del modelo

El modelo se formula con los siguientes términos:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Ecuación 7 Modelo Prophet

Dónde:

- $g(t)$ : función que describe la tendencia lineal o de crecimiento logístico de largo plazo (modelado de los cambios no periódicos de la serie)
- $s(t)$ : función que define la estacionalidad (anual, mensual...)
- $h(t)$ : hechos concretos que pueden alterar los valores de la serie (vacaciones, promociones, etc)
- $\varepsilon_t$ : término de error para aquellos cambios en los valores de la serie no ajustados por el modelo.

Podemos observar que Prophet intenta ajustar las funciones (lineales o no) a los datos, sumando los distintos efectos mientras el tiempo es usado como regresor. Es importante aclarar que Prophet no busca encontrar la dependencia intrínseca entre los eventos y el tiempo, sino que es un ajuste a la curva.

A continuación, explicamos brevemente los componentes del modelo:

#### 3.3.3.1 Componentes de tendencia

Tendencia: Prophet tiene implementados dos modelos: uno lineal y otro de saturación del crecimiento. En este caso el predeterminado de Prophet es un modelo de tendencia de crecimiento lineal. Es decir, la tasa de crecimiento será:

$$g(t) = \alpha \cdot t$$

Ecuación 8 Modelo Prophet tasa de crecimiento

Sin embargo, dentro de Prophet se incluyen los denominados puntos de cambio. Que son puntos en los que la tendencia cambia. Estos puntos se introducen para que la tasa de crecimiento lineal pueda variar. Por defecto, Prophet detecta automáticamente estos puntos, sin embargo, cabe la posibilidad de que el analista los pueda introducir manualmente. Esta detección automática Prophet la realiza de una forma muy sencilla: el modelo especifica una gran cantidad de puntos potenciales en los que es posible que la tasa de crecimiento cambie. A continuación, selecciona únicamente aquellos puntos donde la magnitud del cambio es mayor, quedándose con una pequeña muestra de puntos de cambio.

No todas las series tienen una tendencia basada en un modelo lineal. Hay situaciones en donde se da una saturación que no permite el crecimiento más allá de un límite determinado. En este caso el primer término se modela como una curva de crecimiento logístico (curva sigmoidea).

### 3.3.3.2 Componentes estacionales

En el caso del segundo término (el ajuste estacional), Prophet utiliza series de Fournier. Particularmente se define  $s(t)$  como:

$$s(t) = \sum_{n=1}^N [a_n \cos\left(\frac{2n\pi}{T}t\right) + b_n \sin\left(\frac{2n\pi}{T}t\right)]$$

Ecuación 9 Serie de Fournier

Dónde  $T$  es el período de tiempo correspondiente (valor igual a 7 en el caso de que sea semanal, y valor igual a 365,25 en el caso de que sea anual). Tanto los parámetros  $a$  y  $b$  serán estimados por un dado  $N$ , que será el encargado de determinar si los componentes de alta frecuencia deben ser considerados relevantes o ruido. Cuanto mayor sea  $N$  mayores serán las pequeñas diferencias que el modelo puede ajustar, de no crear una instancia manual Prophet utilizará un  $N=10$  para el periodo anual y  $N=3$  para el semanal.

Para cambiar estos parámetros se utilizan los siguientes argumentos dentro de Prophet:

- weekly\_seasonality
- yearly\_seasonality

Un ejemplo de uso sería para un componente estacional anual:

```
Prophet(yearly_seasonality=20)
```

En caso de querer definir la estacionalidad a medida, Prophet lo permite mediante el método add\_seasonality. Por ejemplo, si queremos un componente estacional denominado "mes" de 30.5 días y con orden de Fourier 5 sería:

```
m.add_seasonality(name='mes', period=30.5, fourier_order=5)
```

### 3.3.3.3 Eventos excepcionales

Para aquellas partes de la serie que no pueden ser explicadas por patrones periódicos o por el componente estacional se modelan en este tercer término  $h(t)$ . Las series normalmente se ven afectadas por eventos externos, ya sea el periodo vacacional, la vuelta al cole o una promoción de gran impacto como puede ser el Black Friday en el mercado tecnológico, o el periodo de rebajas en el mercado textil.

Prophet nos permite mediante los parámetros holiday (nombre del evento) y ds (fecha del evento) proporcionar una lista de eventos personalizados.

Asimismo, podemos indicarle que incluya días anteriores y posteriores a la fecha (ds) mediante los parámetros lower\_window y upper\_window. Si observamos que las vacaciones están sobrareajustándose podemos modificar el impacto que tendrá el evento mediante el parámetro prior\_scale. Por último, podemos añadir regresores adicionales utilizando add\_regressor, permitiendo añadir un efecto adicional unos días concretos (por ejemplo, los domingos de liga de fútbol).

## 3.4 Modelo XGBoost

XGBoost o Extreme Gradient Boosting es, actualmente, uno de los algoritmos de tipo supervisado usados en machine learning. Cuenta con implementaciones en varios lenguajes y obtiene buenos resultados de predicción con poco esfuerzo computacional, sobre todo si se compara con otros modelos más complejos.

Este modelo está basado en el principio de boosting. Es decir, la idea es la de generar múltiples modelos de predicción débiles, y que cada uno se apoye en el anterior para crear un modelo de predicción más fuerte, con una mejor estabilidad y capacidad de predicción (empleando un algoritmo de optimización Gradient Descent).

Gradient Boosting es un modelo conformado por un conjunto de árboles de decisión individuales que están entrenados de forma secuencial, para que cada árbol intente mejorar los errores de predicción del árbol inmediatamente anterior. Este modelo incluye tres elementos principales:

- Una función de pérdida que necesitará ser optimizada.
- Un aprendiz débil con el que hacer predicciones. Xgboost admite árboles de regresión (los de uso más común) y regresiones lineales.
- Un modelo aditivo: se irán añadiendo aprendices débiles de forma secuencial y sin modificar en árboles ya existentes, con la intención de minimizar la función de pérdida.

Un árbol de decisión es un tipo de algoritmo de aprendizaje automático supervisado que funciona tanto para salidas categóricas (clasificación) como continuas (regresión). Su funcionamiento está basado en la división de las observaciones sucesivamente en dos o más grupos hasta conseguir separarlos de la manera más homogénea o diferenciadora posible. Uno de los principales problemas de este algoritmo es el sobreentrenamiento. Sin embargo, este problema se puede aliviar estableciendo restricciones como el acotamiento en el crecimiento del árbol (poda) o poniendo restricciones en los parámetros del modelo. Como ventaja podemos decir que no necesitan preprocessamiento de los datos. Son útiles para detectar variables importantes y su relación. No presuponen nada sobre la distribución de los datos (método no paramétrico)

Mediante la iteración de cada modelo, se van comparando los modelos nuevos con los anteriores. En caso de que el modelo obtenga mejores resultados se tomará este modelo como base. Este proceso se repetirá hasta que las diferencias entre los modelos sean insignificantes o se llegue a las iteraciones máximas.

### 3.4.1 Estrategia de predicción

A la hora de predecir el valor de una serie temporal se puede abordar desde dos horizontes en función del horizonte temporal desde dónde queramos predecir:

- **One-step ahead prediction:** A la hora de predecir el instante  $t$ , utilizaremos datos existentes de un periodo anterior  $t-1$ . Una vez que se haya conseguido predecir el instante  $t$ , utilizaremos esa predicción para predecir el instante  $t+1$ .
- **Multi-step ahead prediction:** al igual que en el caso anterior utilizaremos datos existentes del instante  $t-1$ , sin embargo, en este caso ampliaremos la ventana de predicción a  $n$  instantes ( $t, t+1, t+2, \dots, t+n-1$ ). Para la predicción de estos  $n$  instantes se pueden utilizar varias estrategias:
  - Iterativo: El modelo predice  $n$  instantes utilizando el método One-step. Utilizando la predicción del instante  $t$  se predice  $t+1$ .
  - Directo: con la información de la que dispone el modelo en el instante  $t-1$  predice los  $n$  instantes.

Para el proyecto en curso es necesario generar un conjunto de variables con el que entrenar al algoritmo de aprendizaje supervisado; en este caso, utilizaremos datos retrasados de  $t-12$  meses. Sin embargo, existen más formas de generar este conjunto de variables, como pueden ser:

- **Estadísticos de una venta temporal existentes en el instante anterior a  $t$ :** (máximos y mínimos de un periodo de la serie, venta promedio de las últimas  $n$  semanas, años ...)
- **Variables temporales relacionadas con el instante que se desea predecir:** (mes, día o año específico de la realización de la venta)
- **Variables regresores externas:** por ejemplo, festividades, promociones recurrentes o eventos repetitivos cada cierto tiempo, como por ejemplo un evento deportivo.

## 4 Aplicación de modelo y resultados

En este apartado haremos un análisis exhaustivo de los datos con el objeto de entender de manera profunda la muestra con la que se está trabajando, que tipos de datos tenemos y su comportamiento. Utilizaremos diferentes herramientas de análisis para representar este conocimiento, como pueden ser distintos diagramas de frecuencia, el uso de estadísticos descriptivos o el cruce de las variables principales.

Seguidamente, realizaremos un diseño del estudio que queremos realizar, dónde aplicaremos los distintos modelos vistos con anterioridad para comprobar cuál de ellos se adapta mejor para el problema objeto de estudio.

En este documento se irán mostrando los pasos seguidos, así como el código empleado y los resultados obtenidos.

### 4.1 Análisis exploratorio de los datos

El objetivo de este apartado es el de comprender mejor la muestra obtenida y el comportamiento de los distintos datos y variables.

El conjunto de datos ha sido obtenido de los datamart de El Corte Inglés, en concreto la aplicación web usada ha sido Microstrategy. La exportación se ha tratado con Excel. En función de los datos necesarios la fuente inicial está alojada en diferentes proyectos del datamart, por lo que la exportación genera distintos ficheros. Para agilizar y facilitar el proceso se ha hecho el inner-join de los datos mediante Excel.

#### 4.1.1 Preprocesamiento de los datos

##### 4.1.1.1 Fichero Venta diaria 152

Una vez obtenido el conglomerado final con todos los datos, se ha guardado mediante fichero CSV para facilitar su carga y lectura a la hora de usar Python. El problema surge a la hora de leer los datos con Python ya que el formato CODEUTF8 no admite signos de puntuación, por lo que se han cambiado.

Al cargar el fichero, nos damos cuenta de que los datos no son leídos con el tipo deseado. Mediante código cambiaremos esto para que, a partir de ahora, siempre que carguemos el conjunto de datos se lea correctamente.

```
RangeIndex: 1096 entries, 0 to 1095
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          -----          --    
 0   Empresa      1096 non-null   int64  
 1   Uneco        1096 non-null   int64  
 2   Departamento 1096 non-null   object  
 3   Fecha Venta  1096 non-null   object  
 4   Semana       1096 non-null   int64  
 5   Mes          1096 non-null   int64  
 6   Mes1         1096 non-null   object  
 7   Año          1096 non-null   int64  
 8   Unidades Vendidas 1096 non-null   int64  
 9   Importe Coste 1096 non-null   float64 
 10  Importe de Venta 1096 non-null   float64 
dtypes: float64(2), int64(6), object(3)
memory usage: 94.3+ KB
```

Ilustración 8 Tabla tipo de datos Ventas

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1096 entries, 0 to 1095
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          -----          --    
 0   Empresa      1096 non-null   string  
 1   Uneco        1096 non-null   string  
 2   Departamento 1096 non-null   string  
 3   Fecha Venta  1096 non-null   datetime64[ns]
 4   Semana       1096 non-null   int64  
 5   Mes          1096 non-null   int64  
 6   Mes1         1096 non-null   string  
 7   Año          1096 non-null   string  
 8   Unidades Vendidas 1096 non-null   int32  
 9   Importe Coste 1096 non-null   float64 
 10  Importe de Venta 1096 non-null   float64 
dtypes: datetime64[ns](1), float64(2), int32(1), int64(2), string(5)
memory usage: 90.0 KB
None
```

Ilustración 9 Tabla tipo de datos Ventas

A continuación, haremos un resumen básico del conjunto de datos para ver las distintas variables.

	Semana	Mes	Unidades Vendidas	Importe Coste	\	Importe de Venta
count	1096.000000	1096.000000	1.096000e+03	1.096000e+03		1.096000e+03
mean	26.881387	6.521898	1.905627e+04	8.716599e+05		1.089613e+06
std	15.077716	3.450561	2.133714e+05	7.554095e+05		8.927331e+05
min	1.000000	1.000000	8.700000e-01	3.247632e+04		4.140784e+04
25%	14.000000	4.000000	1.155750e+03	5.100364e+05		6.492802e+05
50%	27.000000	7.000000	1.636000e+03	7.274020e+05		9.026448e+05
75%	40.000000	10.000000	2.259750e+03	9.763765e+05		1.238440e+06
max	53.000000	12.000000	5.060001e+06	8.656273e+06		9.749694e+06

Ilustración 10 Tabla Resumen de las variables

En el siguiente listado podemos encontrar las variables que contiene el conjunto de datos, durante el estudio seleccionaremos cuales de ellas usaremos para la construcción de los modelos.

- **Empresa:** Será una variable de un único valor ('001'). Es una variable irrelevante.
- **Uneco:** Número del departamento objeto de estudio. Será siempre valor único ('0152').
- **Departamento:** Nombre del departamento objeto de estudio. Será siempre valor único (ORD.PERSONALES (HARDWARE)).
- **Año-Mes-Mes1-Semana natural-Fecha Venta:** Variables de tiempo que nos ayudarán a establecer el histórico. Vemos que como año mínimo tenemos 2018 y máximo 2020. Esos tres años los tenemos desglosados por día, la semana que natural de ese día y el mes al que pertenece. Es una variable que nos ayudará a entender el diferente comportamiento existente de la venta entre un mes y otro, ya que este mercado está íntimamente sujeto al periodo promocional existente (Navidades, vuelta al cole, Black Friday...).
- **Unidades Vendidas:** Son las unidades vendidas de cada categoría de producto. Existen valores negativos debido a que se tienen en cuenta las devoluciones efectuadas por los clientes.
- **Importe Coste:** Es el valor a coste de las unidades vendidas.
- **Importe Venta:** Es el valor a venta de las unidades vendidas.

#### 4.1.1.2 Fichero Promociones

Obtenemos los datos sobre las promociones más relevantes para este departamento (Black Friday, Cyber Monday, Semana de Internet y Dia sin IVA) a partir del calendario promocional de El Corte Inglés de los años 2017, 2018 y 2019.

Estos datos obtenidos se guardan en un fichero CSV, al igual que el anterior, para facilitar su lectura en Python.

Para este fichero también hemos tenido que hacer algunas transformaciones en el formato. Obtenemos la siguiente información:

```
Index(['Fecha', 'PROMO', 'Mes', 'Año'], dtype='object')

      Fecha        PROMO   Mes  Año
0  2017-01-21    DIA SIN IVA Enero  2017
1  2017-09-05  SEMANA INTERNET Mayo  2017
2  2017-10-05  SEMANA INTERNET Mayo  2017
3  2017-11-05  SEMANA INTERNET Mayo  2017
4  2017-12-05  SEMANA INTERNET Mayo  2017

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   Fecha    42 non-null    datetime64[ns]
 1   PROMO    42 non-null    string  
 2   Mes      42 non-null    string  
 3   Año      42 non-null    string  
dtypes: datetime64[ns](1), string(3)
memory usage: 1.4 KB
None

      Fecha        PROMO   Mes  Año
count      42          42   42   42
unique      42          4    5    3
top    2017-05-15 00:00:00  SEMANA INTERNET Mayo  2018
freq       1             30   30   15
first     2017-01-21 00:00:00      NaN  NaN  NaN
last      2019-12-05 00:00:00      NaN  NaN  NaN
```

Ilustración 11. Tabla Características datos fichero promociones

El fichero es bastante sencillo, contiene unas columnas con información acerca de las fechas de las promociones y otra con el nombre de cada promoción:

- **FECHA:** Variable de tiempo que nos permitirá conocer en qué días se produjeron las promociones.
- **PROMO:** Nombre de la promoción con cuatro valores posibles (BLACK FRIDAY, DIA SIN IVA, SEMANA INTERNET y CYBER MONDAY).
- **MES:** Mes de la promoción.
- **AÑO:** Año de la promoción.

## 4.1.2 Estacionalidad de la serie

Uno de los puntos importantes antes de predecir es el de estudiar la estacionalidad de nuestra serie temporal. La primera herramienta descriptiva básica es el gráfico temporal.

Un gráfico temporal se construye situando los valores de la serie en el eje de ordenadas y los instantes temporales en el eje de abscisas. Construir este gráfico es una herramienta muy útil que nos ayudará a observar el comportamiento de nuestra serie.

Para la construcción de este gráfico utilizaremos la herramienta PowerBi. Como se ha explicado con anterioridad (véase punto 1.4 Software utilizado), esta herramienta es muy útil para crear gráficos de forma sencilla, sobre todo, si se tienen una gran cantidad de datos y/o, datos repartidos entre varias bases de datos.

En el gráfico temporal podemos ver de forma rápida las ventas en unidades por mes y por año para hacernos una idea del comportamiento de las mismas a través del tiempo.

Empezaremos con un gráfico temporal por trimestre. Posteriormente indagaremos un poco más descendiendo a mes.

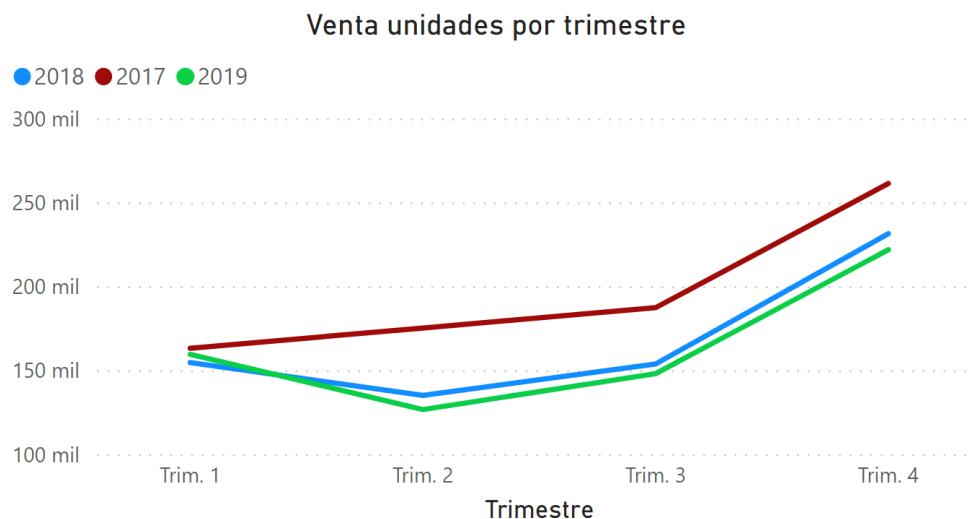


Ilustración 12 Gráfico Venta en unidades por trimestre. PowerBi

Se puede observar un comportamiento muy similar entre 2018 y 2019. En el año 2017 el comportamiento es igual a partir del tercer trimestre. Sin embargo, el descenso de ventas entre el primer trimestre y el segundo, que ha tenido lugar en los años posteriores, no se observa. Al contrario, desde el primer trimestre al tercero hay un ascenso controlado (sin grandes picos) de las ventas.

Descendamos un nivel más y observemos que comportamiento tienen las ventas por mes.

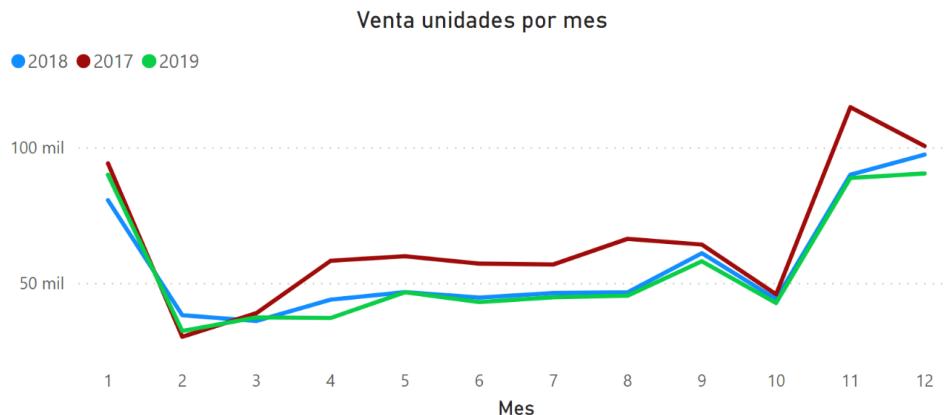


Ilustración 13 Gráfico Venta en unidades por año y mes. PowerBi

Podemos observar cómo año tras año los comportamientos de la venta tanto en los picos como en los descensos se comportan de forma similar. A primera vista, podemos ver como las ventas tienen un gran descenso con la llegada de febrero, luego crecen un poco y se mantienen en niveles más o menos parecidos con nuevo pico al alza en septiembre, para volver a niveles estables en octubre antes de la subida más fuerte durante los meses de navidad (noviembre-diciembre).

Como veíamos con el gráfico por trimestre el crecimiento de ventas en el año 2017 es superior al de años anteriores. Sin embargo, los picos descenso sí que coinciden en tiempo y cantidad en los tres años.

Por tanto, podemos decir que nuestra serie temporal tiene un comportamiento periódico a lo largo del tiempo. Tenemos una serie temporal con aumento de ventas en los meses de invierno, un descenso al terminar el invierno, un comportamiento controlado durante la primavera y el verano, con un pequeño repunte cuando éste está finalizando, para después descender antes de la entrada de la temporada navideña. En resumen, podemos concluir que nuestra serie tiene un comportamiento estacional.

Como hemos visto a la hora de definir los modelos es necesario hacer un trazado de los datos para saber si son estacionarios o no. Podríamos resumir esto en: Cualquier dato de serie temporal que deba modelarse debe ser estacionario. Estacionario significaría entonces que sus propiedades estadísticas son más o menos constantes con el tiempo. Estas propiedades serían:

- Media constante.
- Variación constante (puede haber variaciones. Sin embargo, éstas no deben ser irregulares).
- Sin estacionalidad (sin patrones repetidos en el conjunto de datos).

## 4.2 Medidas de evaluación de los modelos

Para realizar la evaluación de los modelos es necesario conocer su precisión, para ello, utilizaremos validación cruzada. En este caso la validación cruzada se realizará utilizando un conjunto test de  $n$  observaciones y un conjunto test de 31 días (Modelo venta diaria) o 10 meses (Modelo venta mensual). En ambos casos los conjuntos se desplazarán hacia adelante en el tiempo haciendo predicciones sobre los siguientes intervalos de tiempo.

En estadística, un error de pronóstico es la diferencia entre el valor real o real y el predicho o pronosticado de una serie de tiempo o cualquier otro fenómeno de interés. Dado que el error de pronóstico se deriva de la misma escala de datos, solo se pueden hacer comparaciones entre los errores de pronóstico de diferentes series cuando las series están en la misma escala. Las medidas más utilizadas son:

- Error cuadrático medio (MSE): MSE básicamente mide el error cuadrado promedio de nuestras predicciones. Para cada punto, calcula la diferencia cuadrada entre las predicciones y el objetivo y luego promedia esos valores. Cuanto mayor sea este valor, peor es el modelo. Nunca es negativo, ya que estamos cuadrando los errores de predicción individuales antes de sumarlos, pero sería cero para un modelo perfecto.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Ecuación 10 Error cuadrático medio

- Error cuadrático medio (RMSE): RMSE es solo la raíz cuadrada de MSE. La raíz cuadrada se introduce para hacer que la escala de los errores sea igual a la escala de los objetivos. RMSE y MSE son realmente similares en términos de puntuación de modelos.
- Error absoluto medio (MAE): se calcula como un promedio de diferencias absolutas entre los valores objetivo y las predicciones. El MAE es una puntuación lineal, lo que significa que todas las diferencias individuales se ponderan por igual en el promedio. Por ejemplo, la diferencia entre 10 y 0 será el doble de la diferencia entre 5 y 0. Sin embargo, lo mismo no es cierto para RMSE. Lo más importante de MAE respecto a RMSE es que no es tan sensible a los valores atípicos como el error cuadrático medio. Matemáticamente, se calcula utilizando esta fórmula:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

Ecuación 11 Error absoluto medio

Las medidas basadas en errores porcentuales permiten comparar el rendimiento del pronóstico entre diferentes conjuntos de datos, ya que no se mide en unidades si no en porcentaje. La medida que utilizaremos será:

- Error medio de porcentaje absoluto (MAPE): El MAPE nos entrega la desviación en términos porcentuales y no en unidades como las anteriores medidas. Es el promedio del error absoluto o diferencia entre el valor real y el pronóstico, expresado como un porcentaje de los valores reales.

$$MAPE = \frac{\sum_{t=1}^n \frac{|A_t - F_t|}{|A_t|}}{n}$$

*Ecuación 12 Error medio de porcentaje absoluto*

- Error porcentual absoluto medio (MdAPE): El error porcentual absoluto medio (MdAPE) se encuentra al ordenar el error porcentual absoluto (APE) del más pequeño al más grande, y usando este valor del medio (o el promedio de los dos valores de en medio si N es un número par) como la media.

$$MdAPE = \text{median}(p_1, p_2, \dots, p_N)$$

$$p_t = \left| \frac{y_t - f_t}{y_t} \right| = \left| \frac{e_t}{y_t} \right|$$

*Ecuación 13 Error porcentual absoluto medio*



## 5 Elección y ajuste de modelos

En el siguiente apartado, se realizará el procesamiento del conjunto de datos seleccionado mediante los modelos comentados anteriormente (Regresión lineal, Random Forest, ARIMA, Prophet y XGBoost), así como los resultados que muestra cada uno. En cada uno de los casos, se realizará tanto para los datos mensuales, como para los datos diarios.

### 5.1 Regresión lineal

En este apartado se desarrollará el modelo de Regresión Lineal. Utilizaremos la herramienta de Jupyter Notebook para plasmar mediante código en Python todos los pasos necesarios y los cálculos que se precisen.

#### 5.1.1 Carga de ficheros y módulos

Cargaremos el fichero “VentaMes2017-2019.csv”. Utilizaremos, como se ha comentado previamente, código Python para su carga, lectura y tratamiento de los datos. Para hacer la regresión lineal en Python, vamos a usar scikit-learn, que es una librería de aprendizaje automático. Dentro de esta librería usaremos la clase LinearRegression ya que implementará la parte teórica que hemos descrito con anterioridad.

```
# Importamos todos los módulos necesarios

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#modelos Regresión Lineal y Random Forest
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller

#medidas de error
from sklearn.metrics import mean_squared_error, confusion_matrix
from math import sqrt

df =pd.DataFrame()
df = pd.read_csv('C:/Users/jorge/El Corte Inglés, S.A/0tb y el master - Documentos/VentaMes2017-2019.csv'
                  ,';',index_col='Fecha Venta',parse_dates=True)
```

```
#Convertimos el índice en fecha
df.index = pd.to_datetime(df.index)
print(df.head())
df.info()

#df['Año'] = df.index.year
#df['Mes'] = df.index.month
#df['Dia'] = df.index.day_name()
#df['Semana'] = df.index.week
```

	Unidades Vendidas
Fecha Venta	Ventas
2017-01-01	94158
2017-02-01	30152
2017-03-01	38901
2017-04-01	58229
2017-05-01	59880

<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 36 entries, 2017-01-01 to 2019-12-01  
Data columns (total 1 columns):  
 # Column Non-Null Count Dtype   
--- --   
 0 Unidades Vendidas 36 non-null int64   
dtypes: int64(1)  
memory usage: 576.0 bytes

Ilustración 14 Tabla Resumen conjunto de datos

Una vez hemos cargado el conjunto de datos, hemos cargado las librerías necesarias y hemos visto una pequeña información del mismo es hora de representar la serie temporal en un gráfico. La representación gráfica de una serie es muy importante, ya que permite de una forma visual, obtener una aproximación del comportamiento de la misma.

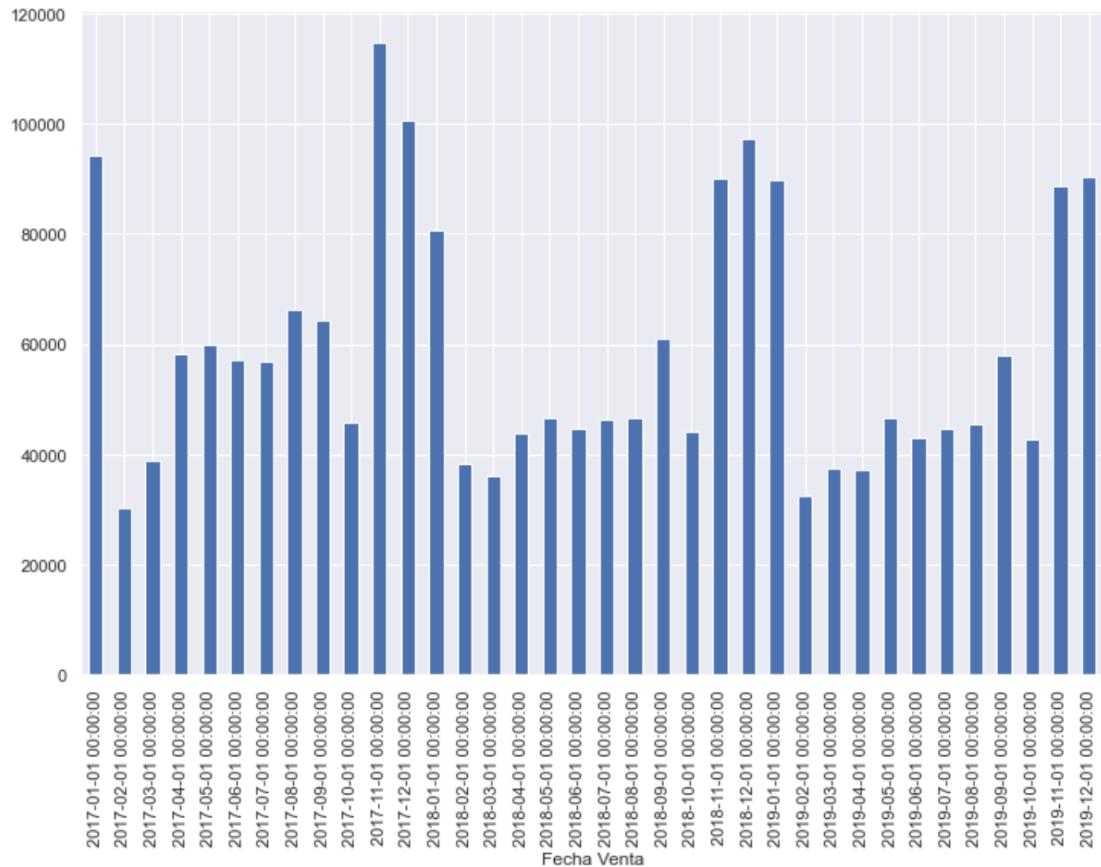


Ilustración 15 Gráfico serie de ventas 2017-2020

En la serie del Gráfico se observa una tendencia definida. Hay períodos con picos altos que coinciden con los meses de alta actividad promocional o fechas de ventas ya muy establecidas en nuestro país, como pueden ser el periodo navideño o la "vuelta al cole". Se observan varios valores muy extremos con picos muy altos respecto a la media con grandes descensos a continuación. Si bien durante los tres años podemos observar como la tendencia es casi la misma, los picos y sus recesos coinciden casi en el mismo periodo.

### 5.1.2 Aplicación y entrenamiento del modelo mensual

Lo primero que haremos será crear una matriz ordenada con columnas de los doce meses inmediatamente anteriores:

```
df['Sale_LastMonth'] = df['Unidades Vendidas'].shift(+1)
df['Sale_2Monthsback'] = df['Unidades Vendidas'].shift(+2)
df['Sale_3Monthsback'] = df['Unidades Vendidas'].shift(+3)
df['Sale_4Monthsback'] = df['Unidades Vendidas'].shift(+4)
df['Sale_5Monthsback'] = df['Unidades Vendidas'].shift(+5)
df['Sale_6Monthsback'] = df['Unidades Vendidas'].shift(+6)
df['Sale_7Monthsback'] = df['Unidades Vendidas'].shift(+7)
df['Sale_8Monthsback'] = df['Unidades Vendidas'].shift(+8)
df['Sale_9Monthsback'] = df['Unidades Vendidas'].shift(+9)
df['Sale_10Monthsback'] = df['Unidades Vendidas'].shift(+10)
df['Sale_11Monthsback'] = df['Unidades Vendidas'].shift(+11)
df['Sale_12Monthsback'] = df['Unidades Vendidas'].shift(+12)

df=df.dropna()
df.head(10)
```

	Unidades Vendidas	Sale_LastMonth	Sale_2Monthsback	Sale_3Monthsback	Sale_4Monthsback	Sale_5Monthsback	Sale_6Monthsback
Fecha Venta							
2019-01-01	89955	97409.0	89998.0	44042.0	61019.0	46556.0	463
2019-02-01	32295	89955.0	97409.0	89998.0	44042.0	61019.0	465
2019-03-01	37323	32295.0	89955.0	97409.0	89998.0	44042.0	610
2019-04-01	37094	37323.0	32295.0	89955.0	97409.0	89998.0	440
2019-05-01	46596	37094.0	37323.0	32295.0	89955.0	97409.0	895
2019-06-01	42994	46596.0	37094.0	37323.0	32295.0	89955.0	974
2019-07-01	44742	42994.0	46596.0	37094.0	37323.0	32295.0	895
2019-08-01	45341	44742.0	42994.0	46596.0	37094.0	37323.0	322
2019-09-01	58027	45341.0	44742.0	42994.0	46596.0	37094.0	375
2019-10-01	42642	58027.0	45341.0	44742.0	42994.0	46596.0	370

Ilustración 16 Tabla meses de venta

Como podemos observar los primeros meses del conjunto de datos no tienen meses anteriores, por lo que eliminaremos los NaN para evitar futuros errores.

Haremos una instancia al modelo de regresión de la biblioteca scikit-learn y luego le pasaremos los valores en forma de array de numpy.

```
#hacemos instancia al modelo de Regresión Lineal
lin_model=LinearRegression()
```

```
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y = df['Sale_LastMonth'],df['Sale_2Monthsback'],df['Sale_3Monthsback'],df['Sale_4Monthsback']
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y=np.array(x1),np.array(x2),np.array(x3),np.array(x4),np.array(x5),np.array(x6),np.array(x7),
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y=final_x[0:-1],x2.reshape(-1,1),x3.reshape(-1,1),x4.reshape(-1,1),x5.reshape(-1,1),x6.reshape(-1,1),y
final_x=np.concatenate((x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12),axis=1)
print(final_x)
```

A continuación, dividiremos el conjunto de datos en dos: la parte de entrenamiento y la parte test. Después, mediante método fit existente en scikit-learn entrenaremos el modelo. El cual ajustará los parámetros de regresión lineal a nuestros datos.

```
: # separamos el conjunto en dos : parte de entrenamiento y parte test
X_train,X_test,y_train,y_test=final_x[:-10],final_x[-10:],y[:-10],y[-10:]

: # entrenamos el modelo
lin_model.fit(X_train,y_train)
print('w = ' +str(lin_model.coef_) + ', b = ' + str(lin_model.intercept_))

w = [[ 0.51836717 -0.28824718  0.07583199  0.20876215 -0.22527906  0.05954564
       0.03658558 -0.05242031 -0.07932635  0.233845   -0.15729829  0.81109425]], b = [-14938.69891118]
```

Podemos ver, mediante la librería matplotlib la predicción que realiza a la parte test:

```
: # una vez entrenado, predecimos la parte test
lin_pred=lin_model.predict(X_test)

sns.set(rc={'figure.figsize':(11, 6)})
# plt.rcParams["figure.figsize"] = (11,6)
plt.plot(lin_pred,label='Linear_Regression_Predictions')
plt.plot(y_test,label='Actual Sales')
plt.legend(loc="upper left")
plt.show()
```



Ilustración 17 Gráfico Venta real vs predicción modelo lineal mensual

### Medidas de error:

La medida de error que usaremos será una de las más usadas en los problemas de regresión lineal supervisada: el error cuadrático medio.

Error cuadrático medio: representa a la raíz cuadrada de la distancia cuadrada promedio entre el valor real y el valor pronosticado.

```
# utilizamos el error cuadrático medio como medida de error.  
rmse_lr=sqrt(mean_squared_error(lin_pred,y_test))  
  
print('Mean Squared Error for Linear Regression Model is:',rmse_lr)  
Mean Squared Error for Linear Regression Model is: 10189.525156691261
```

### 5.1.3 Aplicación y entrenamiento del modelo diario

Ahora que hemos obtenido los resultados mensuales, vamos a realizar el experimento con un conjunto de datos diarios para, posteriormente, comparar el resultado de ambos modelos.

Realizaremos el mismo método de trabajo que hemos seguido con el modelo mensualizado.

Sin embargo, para el paso de tiempo (ahora de 30 días) usaremos un bucle *for* que nos hará el paso para cada día.

```
#bucle para convertir pasos anteriores de tiempo  
valor = 1  
  
for sale in range(1,31):  
    df['Sale_LastDay'+str(valor)] = df['Unidades Vendidas'].shift(valor)  
    valor = valor +1  
  
df=df.dropna()  
df.head(10)
```

	Unidades Vendidas	Sale_LastDay1	Sale_LastDay2	Sale_LastDay3	Sale_LastDay4	Sale_LastDay5	Sale_LastDay6	Sale_LastDay7	Sale_LastDay8	Sale_LastDay9
Fecha Venta										
2018-01-31	2775.0	5001.0	4199.0	4406.0	1719.0	1676.0	1803.0	962.0	2379.0	1860.0
2018-02-01	4623.0	2775.0	5001.0	4199.0	4406.0	1719.0	1676.0	1803.0	962.0	2379.0
2018-02-02	1235.0	4623.0	2775.0	5001.0	4199.0	4406.0	1719.0	1676.0	1803.0	962.0
2018-03-02	1188.0	1235.0	4623.0	2775.0	5001.0	4199.0	4406.0	1719.0	1676.0	1803.0
2018-04-02	2545.0	1188.0	1235.0	4623.0	2775.0	5001.0	4199.0	4406.0	1719.0	1676.0
2018-05-02	2368.0	2545.0	1188.0	1235.0	4623.0	2775.0	5001.0	4199.0	4406.0	1719.0
2018-06-02	2980.0	2368.0	2545.0	1188.0	1235.0	4623.0	2775.0	5001.0	4199.0	4406.0
2018-07-02	1543.0	2980.0	2368.0	2545.0	1188.0	1235.0	4623.0	2775.0	5001.0	4199.0
2018-08-02	2164.0	1543.0	2980.0	2368.0	2545.0	1188.0	1235.0	4623.0	2775.0	5001.0
2018-09-02	1536.0	2164.0	1543.0	2980.0	2368.0	2545.0	1188.0	1235.0	4623.0	2775.0

10 rows x 31 columns

Ilustración 18 Tabla diaria de venta

Una vez obtenidas las columnas convertiremos el dataframe a una matriz de arrays de numpy para poder trabajar con el modelo.

```
#guardamos columnas en variables
x1,x2,x3,x4,x5 = df['Sale_LastDay1'], df['Sale_LastDay2'], df['Sale_LastDay3'],df['Sale_LastDay4'],df['Sale_LastDay5']
x6,x7,x8,x9,x10,x11 = df['Sale_LastDay6'], df['Sale_LastDay7'], df['Sale_LastDay8'],df['Sale_LastDay9'],df['Sale_LastDay10'],df[
x12,x13,x14,x15,x16 = df['Sale_LastDay12'],df['Sale_LastDay13'],df['Sale_LastDay14'],df['Sale_LastDay15'],df['Sale_LastDay16']
x17,x18,x19,x20,x21 = df['Sale_LastDay17'],df['Sale_LastDay18'],df['Sale_LastDay19'],df['Sale_LastDay20'],df['Sale_LastDay21']
x22,x23,x24,x25,x26 = df['Sale_LastDay22'],df['Sale_LastDay23'],df['Sale_LastDay24'],df['Sale_LastDay25'],df['Sale_LastDay26']
x27,x28,x29,x30,y = df['Sale_LastDay27'],df['Sale_LastDay28'],df['Sale_LastDay29'],df['Sale_LastDay30'],df['Unidades Vendidas']

#convertimos las variables en arrays
x1,x2,x3,x4,x5=np.array(x1),np.array(x2),np.array(x3),np.array(x4),np.array(x5)
x6,x7,x8,x9,x10,x11 = np.array(x6),np.array(x7),np.array(x8),np.array(x9),np.array(x10),np.array(x11)
x12,x13,x14,x15,x16 = np.array(x12),np.array(x13),np.array(x14),np.array(x15),np.array(x16),
x17,x18,x19,x20,x21 = np.array(x17),np.array(x18),np.array(x19),np.array(x20),np.array(x21),
x22,x23,x24,x25,x26 = np.array(x22),np.array(x23),np.array(x24),np.array(x25),np.array(x26),
x27,x28,x29,x30,y = np.array(x27),np.array(x28),np.array(x29),np.array(x30),np.array(y)

x1,x2,x3,x4,x5=x1.reshape(-1,1),x2.reshape(-1,1),x3.reshape(-1,1),x4.reshape(-1,1),x5.reshape(-1,1)
x6,x7,x8,x9,x10,x11 = x6.reshape(-1,1),x7.reshape(-1,1),x8.reshape(-1,1),x9.reshape(-1,1),x10.reshape(-1,1),x11.reshape(-1,1)
x12,x13,x14,x15,x16 = x12.reshape(-1,1),x13.reshape(-1,1),x14.reshape(-1,1),x15.reshape(-1,1),x16.reshape(-1,1),
x17,x18,x19,x20,x21 = x17.reshape(-1,1),x18.reshape(-1,1),x19.reshape(-1,1),x20.reshape(-1,1),x21.reshape(-1,1),
x22,x23,x24,x25,x26 = x22.reshape(-1,1),x23.reshape(-1,1),x24.reshape(-1,1),x25.reshape(-1,1),x26.reshape(-1,1),
x27,x28,x29,x30,y = x27.reshape(-1,1),x28.reshape(-1,1),x29.reshape(-1,1),x30.reshape(-1,1),y.reshape(-1,1)

#createmos la matriz
final_x=np.concatenate((x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,y))

print(final_x)
print(final_x.shape)
```

Una vez con los datos en el formato correcto, crearemos un conjunto de entrenamiento y otro de test.

Entrenaremos el modelo, y haremos predicciones sobre el conjunto test.

```
# conjunto de entrenamiento y conjunto de test

X_train,X_test,y_train,y_test=final_x[:-30],final_x[-30:],y[:-30],y[-30:]
print(len(X_train))
```

1035

```
#Regresión lineal predicción

lin_pred=lin_model.predict(X_test)

plt.rcParams["figure.figsize"] = (11,6)
plt.plot(lin_pred,label='Linear_Regression_Predictions')
plt.plot(y_test,label='Actual Sales')
plt.legend(loc="upper left")

plt.show()
```

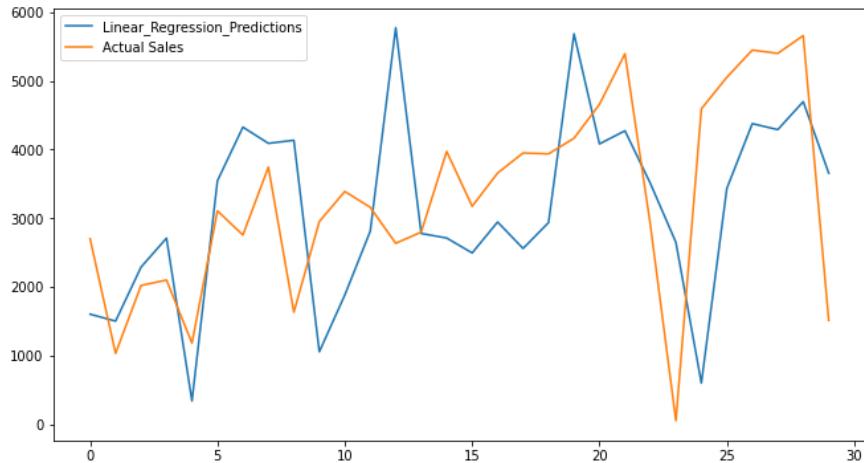


Ilustración 19 Gráfico Venta real vs predicción modelo lineal diario

Como medida de error usaremos la misma que para el modelo mensualizado: error cuadrático medio (RMSE).

```
rmse_lr=sqrt(mean_squared_error(lin_pred,y_test))
print('Mean Squared Error for Linear Regression Model is:',rmse_lr)

Mean Squared Error for Linear Regression Model is: 1532.245526220537
```

A continuación, veremos una tabla comparativa de los errores de cada modelo. En el apartado 6. Conclusiones veremos una explicación conjunta de las diferencias entre todos los modelos y las repercusiones de cada uno.

Modelo	Tipo de predicción	
Linear Regression	Mensual	Diario
RMSE	9.769	1.532

*Ilustración 20 Tabla RMSE modelo lineal diario*

## 5.2 Random Forest

### 5.2.1 Aplicación y entrenamiento del modelo mensual

En este apartado se desarrollará el modelo Random Forest. Utilizaremos la herramienta de Jupyter Notebook para plasmar mediante código en Python todos los pasos necesarios y los cálculos que se precisen.

Al igual que hicimos con el modelo de regresión lineal importaremos la librería scikit-learn de la que obtendremos el regresor del Random Forest y las herramientas necesarias para el entrenamiento del modelo. Asimismo, una vez más, utilizaremos la librería matplotlib para representar gráficamente los resultados.

Al tener explorado, representado y dividido el conjunto de datos en la parte test y train para el modelo anterior no repetiremos estas fases. Haremos instancia al modelo mediante la clase RandomForestRegressor para luego usarlo en la predicción. Asimismo, entrenaremos el modelo usando el conjunto dividido entre test y train.

```
# entrenamiento Random Forest
print(model.fit(X_train,y_train))
model.score(X_test, y_test)

RandomForestRegressor(max_features=3, random_state=1)

0.8460775082953083
```

Una vez entrenado el modelo realizamos la predicción sobre el conjunto X\_test y lo representamos:

```
#Random Forest prediction

pred=model.predict(X_test)

plt.rcParams["figure.figsize"] = (12,8)
plt.plot(pred,label='Random_Forest_Predictions')
plt.plot(y_test,label='Actual Sales')
plt.legend(loc="upper left")

plt.show()
```

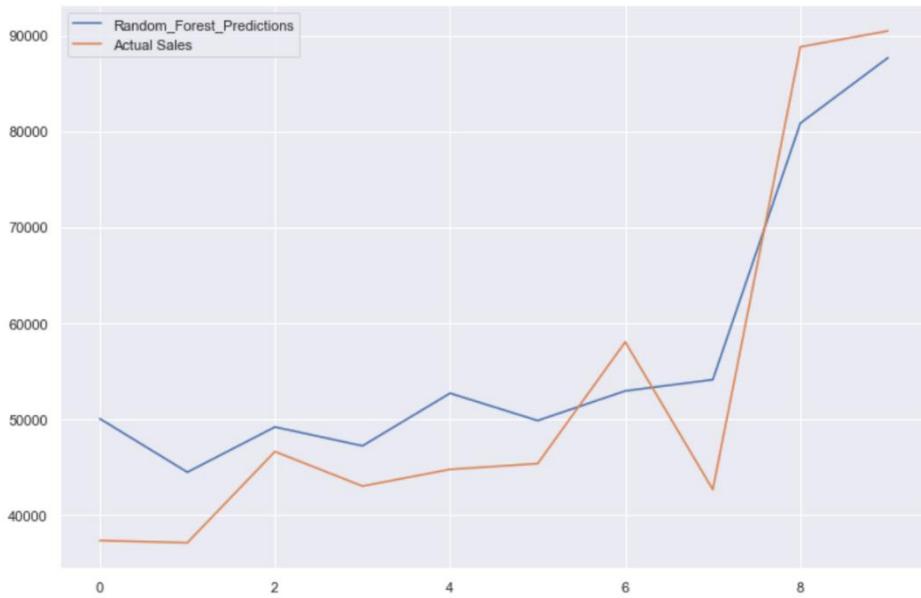


Ilustración 21 Gráfico Venta real vs Predicción RF mensual

Como medida de error usaremos, al igual que en modelo anterior, el error cuadrático medio.

```
rmse_rf=sqrt(mean_squared_error(pred,y_test))
print('Mean Squared Error for Random Forest Model is:',rmse_rf)
```

Mean Squared Error for Random Forest Model is: 7424.550565939328

## 5.2.2 Aplicación y entrenamiento del modelo diario

En este apartado, como hemos hecho con el modelo de regresión lineal, realizaremos el experimento para un conjunto de datos diarios y, posteriormente, compararemos resultados entre ambos modelos.

Utilizaremos la matriz de arrays que hemos preparado con anterioridad para el modelo diario de regresión lineal (véase punto 5.1.3 Aplicación y entrenamiento del modelo para datos diarios).

Separaremos el conjunto de entrenamiento y test para, a continuación, realizar las predicciones sobre el conjunto test.

```
#hacemos instancia al modelo de Random Forest
model = RandomForestRegressor(n_estimators=100,max_features=3, random_state=1)

#entrenamiento Random Forest
print(model.fit(X_train, y_train))
model.score(X_test, y_test)
```

```
#Random Forest prediction
pred=model.predict(X_test)

plt.rcParams["figure.figsize"] = (12,8)
plt.plot(pred,label='Random_Forest_Predictions')
plt.plot(y_test,label='Actual Sales')
plt.legend(loc="upper left")

plt.show()
```



Ilustración 22 Gráfico Venta real vs Predicción RF diario

Como medida de error utilizaremos, como viene siendo habitual, el error cuadrático medio.

```
#medidas de error para ver la bondad del modelo

rmse_rf=sqrt(mean_squared_error(pred,y_test))
print('Mean Squared Error for Random Forest Model is:',rmse_rf)
```

Mean Squared Error for Random Forest Model is: 1132.711152162295

Sin embargo, para este experimento hemos utilizado cien árboles (`n_estimators=100`). En el apartado anterior (random Forest mensual) utilizamos mil árboles para realizar el entrenamiento. Veremos pues cómo se comporta el modelo con mil árboles además de con cien.

```
#modificamos el nº de árboles

model2 = RandomForestRegressor(n_estimators=1000,max_features=3, random_state=1)

#entrenamiento Random Forest

print(model2.fit(X_train, y_train))
model2.score(X_test, y_test)
```

```
#Random Forest prediction
pred2=model2.predict(X_test)

plt.rcParams["figure.figsize"] = (12,8)
plt.plot(pred,label='Random_Forest_Predictions')
plt.plot(y_test,label='Actual Sales')
plt.legend(loc="upper left")

plt.show()
```

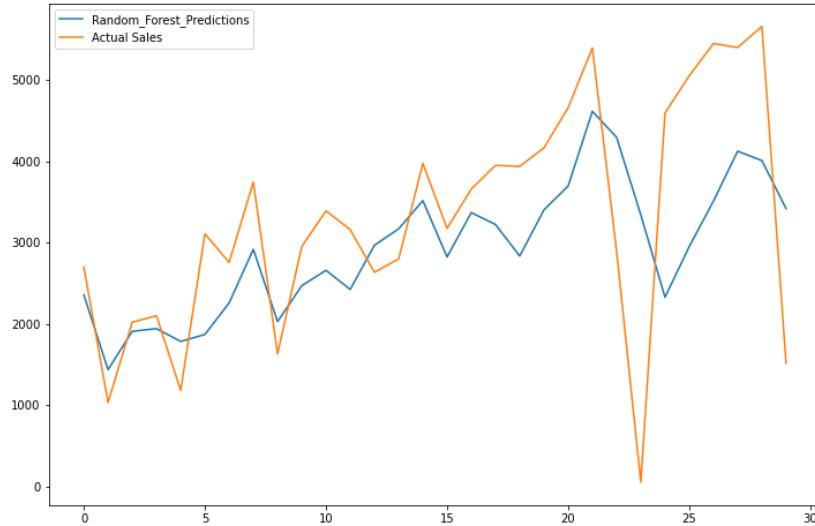


Ilustración 23 Gráfico Venta real vs Predicción RF diario

Una vez entrenado el modelo con mil árboles y realizar la predicción, utilizaremos de nuevo el RMSE como medida de error.

```
#medidas de error para ver la bondad del modelo

rmse_rf=sqrt(mean_squared_error(pred2,y_test))
print('Mean Squared Error for Random Forest Model is:',rmse_rf)
```

Mean Squared Error for Random Forest Model is: 1199.9288407649904

Como podemos observar, no hay mejoría con el modelo con cien árboles. Esto puede deberse a que al aumentar el número de árboles se ha producido un poco de sobreentrenamiento.

A continuación, veremos una tabla comparativa de los errores de cada modelo. En el apartado 6. Conclusiones veremos una explicación conjunta de las diferencias entre todos los modelos y las repercusiones de cada uno.

Modelo	Tipo de predicción		
Random Forest	Mensual	Diario	Diario 2
RMSE	13.470	1.133	1.200
nº árboles	1000	100	1000
Score	59,2%	34,6%	26,6%

Ilustración 24 Tabla RMSE RF diario y mensual

## 5.3 ARIMA

### 5.3.1 ARIMA conjunto de datos mensuales

En este apartado se desarrollará el modelo ARIMA. Utilizaremos la herramienta de Jupyter Notebook para plasmar mediante código en Python todos los pasos necesarios y los cálculos que se precisen.

Lo primero que haremos será importar las librerías necesarias para la manipulación de datos.

```
|: # pip install pmdarima
import pandas as pd
import numpy as np
import datetime

import csv

import matplotlib.pyplot as plt
%matplotlib inline
```

A continuación, importaremos las dependencias necesarias para la construcción de modelos de predicción ARIMA.

```
#dependencias para desarrollar ARIMA Forecasting Models

from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.api import acf, pacf, graphics
from statsmodels.tsa.ar_model import AutoReg, ar_select_order
from statsmodels.tsa.stattools import adfuller
import pmdarima as pm
import statsmodels.api as sm

from pandas.plotting import autocorrelation_plot

from sklearn.metrics import mean_squared_error
```

Seguidamente, cargaremos el conjunto de datos y lo representaremos.

```
df = pd.read_csv('C:/Users/jorge/El Corte Inglés, S.A/0tb y el master - Documentos/VentaMes2017-2019.csv')

df['Fecha Venta'] = pd.to_datetime(df['Fecha Venta'],format='%Y/%m/%d')
df=df.dropna()

#set date as index column. (required in time series)
df = df.set_index('Fecha Venta')
df = df.asfreq('MS')
print('Shape of data',df.shape)
print(df.dtypes)
df.head()
```

La variable Fecha Venta se carga como string, tenemos que convertirla a formato datetime. Para ello usamos una funcionalidad que nos trae la librería Pandas: to\_datetime. Una vez que la tenemos en el formato correcto para poder usar las funcionalidades de pandas vamos a convertir la columna 'Fecha Venta' en el índice de la serie (df = df.set\_index('Fecha Venta')). Asimismo, dado que la frecuencia con la que vamos a trabajar es mensual, lo indicamos (df.asfreq('MS')).

Imprimiremos las primeras líneas para comprobar que todo ha quedado de la forma correcta y usaremos un paso para verificar que la serie temporal está completa y no hay cortes.

Unidades Vendidas	
Fecha Venta	
2017-01-01	94158
2017-02-01	30152
2017-03-01	38901
2017-04-01	58229
2017-05-01	59880

```
# Paso para verificar que un índice temporal está completo
```

```
(df.index == pd.date_range(start=df.index.min(),
                           end=df.index.max(),
                           freq=df.index.freq)).all()
```

```
True
```

Ilustración 25 Tabla datos ajustados a ARIMA

A continuación, como hemos hecho en los modelos anteriores, representamos la serie para tener una primera visión global del comportamiento de nuestros datos.

```
df['Unidades Vendidas'].plot(figsize=(12,5),kind='bar')
```

```
<AxesSubplot:xlabel='Fecha Venta'>
```

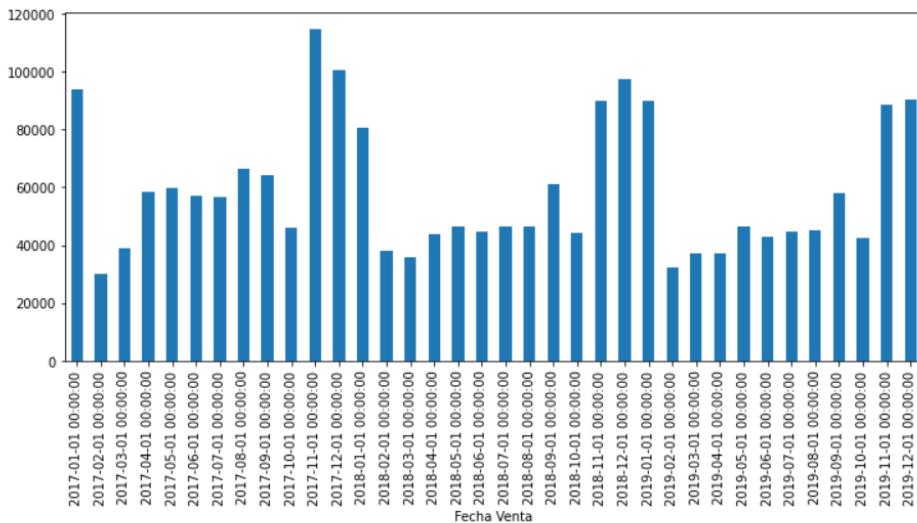


Ilustración 26 Gráfico serie ventas mensuales 2017-2019

Como sabemos, los datos de tiempo tienen que ser estacionarios (media, varianza y covarianza constantes a lo largo del tiempo). Es decir, para poder aplicar el modelo ARIMA los datos deben ser estacionarios. Por lo tanto, lo primero que haremos será comprobar la estacionalidad de nuestros datos.

Para probar la estacionalidad de la serie usaremos:

- **El test de Dickey-Fuller Aumentado (Augmented Dickey-Fuller Test (ADF)),** que elimina la autocorrelación e indica si una serie es estacionaria o no.
- **El test de Phillips-Perron (PP),** es una modificación de test de Dickey-Fuller. Este test corrige la autocorrelación y heterocedasticidad en los errores.

En estos test, la hipótesis nula es que la serie tiene raíces unitarias, por tanto, no es estacionaria. Por ende, la hipótesis alternativa es que la serie es estacionaria (p-value < 0.05 estacionaria, p-value > 0.05 no es estacionaria)

Con un p-valor inferior a 0.05, la hipótesis nula se suele rechazar.

```
: from statsmodels.tsa.stattools import adfuller

def adf_test(dataset):
    dfstest = adfuller(dataset, autolag = 'AIC')
    print("1. ADF Statistic : ",dfstest[0])
    print("2. P-Value : ", dfstest[1])
    print("3. Num Of Lags : ", dfstest[2])
    print("4. Num Of Observations Used For ADF Regression and Critical Values Calculation : ", dfstest[3])
    print("5. Critical Values :")
    for key, val in dfstest[4].items():
        print("\t",key, ":", val)

: adf_test(df['Unidades Vendidas'])

#p-value < 0.05 estacionaria
#p-value > 0.05 no es estacionaria
```

1. ADF Statistic : -4.8718801184069935  
2. P-Value : 3.9546700444868994e-05  
3. Num Of Lags : 8  
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 27  
5. Critical Values :  
1% : -3.6996079738860943  
5% : -2.9764303469999494  
10% : -2.627601001371742

Podemos ver que nuestra serie es estacionaria (p-value=0.00000395 < 0.05).

Una vez que hemos comprobado que nuestra serie es estacionaria vamos a resolver el orden del modelo ARIMA. Para ello, de la biblioteca pmldarima importamos la clase auto\_arima que será la encargada de darnos los resultados.

```
stepwise_fit = auto_arima(df['Unidades Vendidas'], seasonal=True, m=12)
stepwise_fit.summary()
```

SARIMAX Results						
Dep. Variable:	y	No. Observations:	36			
Model:	SARIMAX(3, 0, 3)x(1, 0, [1], 12)	Log Likelihood	-392.678			
Date:	Tue, 15 Jun 2021	AIC	805.356			
Time:	09:57:57	BIC	821.191			
Sample:	0	HQIC	810.883			
	- 36					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
intercept	9.056e+04	2.29e+05	0.396	0.692	-3.58e+05	5.39e+05
ar.L1	-1.7337	3.720	-0.466	0.641	-9.024	5.557
ar.L2	-1.4894	2.997	-0.497	0.619	-7.364	4.385
ar.L3	-0.7358	2.768	-0.266	0.790	-6.162	4.690
ma.L1	1.9918	3.914	0.509	0.611	-5.679	9.663
ma.L2	1.9666	4.280	0.460	0.646	-6.422	10.355
ma.L3	0.9532	3.874	0.246	0.806	-6.640	8.546
ar.S.L12	0.7034	0.357	1.970	0.049	0.004	1.403
ma.S.L12	0.1563	0.922	0.170	0.865	-1.651	1.964
sigma2	2.014e+08	77.354	2.6e+06	0.000	2.01e+08	2.01e+08
Ljung-Box (L1) (Q):	0.05	Jarque-Bera (JB):	6.24			
Prob(Q):	0.82	Prob(JB):	0.04			
Heteroskedasticity (H):	0.29	Skew:	0.81			
Prob(H) (two-sided):	0.04	Kurtosis:	4.25			

Ilustración 27 Tabla resultados SARIMAX

El resumen nos devuelve que nuestro modelo SARIMAX (Seasonal AutoRegressive Integrated Moving Average) es de orden (3,0,3) x (1,0,[1],12).

Ahora que hemos resuelto el orden de nuestro modelo vamos a separar una parte de entrenamiento y otra de test para realizar los entrenamientos y predicciones.

```
: print(df.shape)
train=df.iloc[:-10]
test=df.iloc[-10:]
#print(train.shape,test.shape)
#print(test.iloc[0],test.iloc[-1])
fig, ax=plt.subplots(figsize=(9, 4))
train.plot(ax=ax, label='train')
test.plot(ax=ax, label='test')
ax.legend();
(36, 1)
```

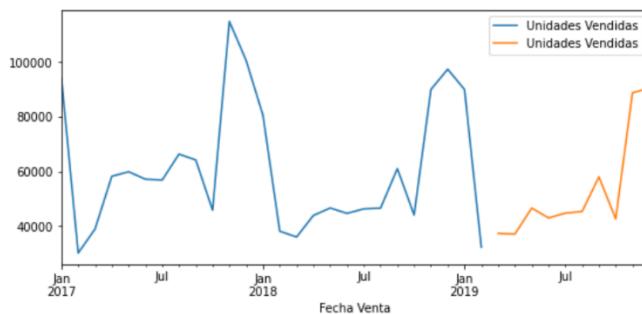


Ilustración 28 Gráfico de predicción ARIMA mensual

Es momento de entrenar el modelo. Para ello le indicaremos el orden y el seasonal\_order. Como estamos con periodicidad mensual le indicamos m=12.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model=SARIMAX(train['Unidades Vendidas'],order=(3, 0, 3), seasonal_order=(1, 0, [1], 12))
model.fit()
model.summary()
```

SARIMAX Results

Dep. Variable:	Unidades Vendidas	No. Observations:	26			
Model:	SARIMAX(3, 0, 3)x(1, 0, [1], 12)	Log Likelihood	-291.642			
Date:	Tue, 15 Jun 2021	AIC	601.283			
Time:	10:44:32	BIC	612.606			
Sample:	01-01-2017 - 02-01-2019	HQIC	604.544			
Covariance Type:						
	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.3206	0.841	0.381	0.703	-1.327	1.968
ar.L2	-0.0781	1.818	-0.043	0.966	-3.640	3.484
ar.L3	0.7285	1.859	0.392	0.695	-2.915	4.372
ma.L1	0.1001	1.087	0.092	0.927	-2.030	2.230
ma.L2	0.0989	1.589	0.062	0.950	-3.016	3.214
ma.L3	-0.8365	1.135	-0.737	0.461	-3.062	1.389
ar.S.L12	0.7894	2.285	0.345	0.730	-3.689	5.268
ma.S.L12	-0.1540	5.136	-0.030	0.976	-10.220	9.912
sigma2	3.506e+08	3.53e-09	9.92e+16	0.000	3.51e+08	3.51e+08
Ljung-Box (L1) (Q):	0.09	Jarque-Bera (JB):	1.85			
Prob(Q):	0.76	Prob(JB):	0.40			
Heteroskedasticity (H):	0.34	Skew:	0.43			
Prob(H) (two-sided):	0.13	Kurtosis:	3.98			

Ilustración 29 Tabla resultados SARIMAX

Con el modelo ya entrenado vamos a realizar las predicciones sobre el conjunto test y los representamos junto con la venta real para ver de una forma gráfica el acierto de la predicción.

```
start=len(train)
end=len(train)+len(test)-1
#index_future_dates=pd.date_range(start='2017-01-01',end='2019
pred=model.predict(start=start,end=end,typ='levels').rename('A
#pred.index=index_future_dates
pred.plot(legend=True)
test['Unidades Vendidas'].plot(legend=True)

<AxesSubplot:xlabel='Fecha Venta'>
```

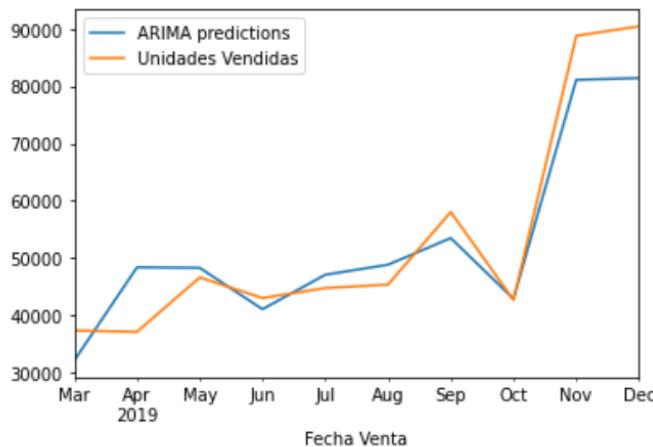


Ilustración 30 Gráfico Venta real vs Predicción ARIMA mensual

Al igual que hemos hecho en otros modelos, vamos a utilizar como medida de error el error cuadrático medio para ver la desviación del modelo.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse=sqrt(mean_squared_error(pred,test['Unidades Vendidas']))
print(rmse)
```

5829.452602043905

Con el modelo entrenado y las predicciones sobre el conjunto test hechas, vamos a realizar una predicción a futuro para los próximos 5 meses. Representaremos esa predicción en un gráfico. Nuestro último mes es diciembre del año 2019. Haremos la predicción para los meses de enero a mayo del año 2020.

```
index_future_months = pd.date_range(start='2020-01-01', periods=5, freq='M')
pred = model2.predict(start=len(df), end=len(df)+4, typ='levels').rename('ARIMA Predictions')
pred.index = index_future_months
print(pred)

2020-01-31    95956.511376
2020-02-29    48760.664482
2020-03-31    55473.019597
2020-04-30    60067.201389
2020-05-31    57839.458917
Freq: M, Name: ARIMA Predictions, dtype: float64
```

```
pred.plot(figsize=(12,5), legend=True)
```

```
<AxesSubplot:>
```

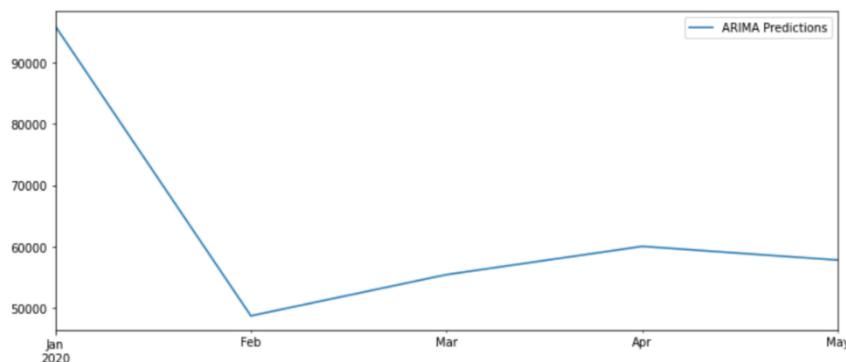


Ilustración 31 Gráfico Predicción ARIMA 5 meses

Finalmente, haremos un plot de las tres partes del modelo para tener una visión global en una línea temporal de la parte de entrenamiento, el test y la predicción de los cinco meses inmediatamente posteriores.

```
fig, ax = plt.subplots(figsize=(12, 5))
train.plot(ax=ax, label='train')
test.plot(ax=ax, label='test')
pred.plot(ax=ax, label='predicciones')
ax.legend();
```

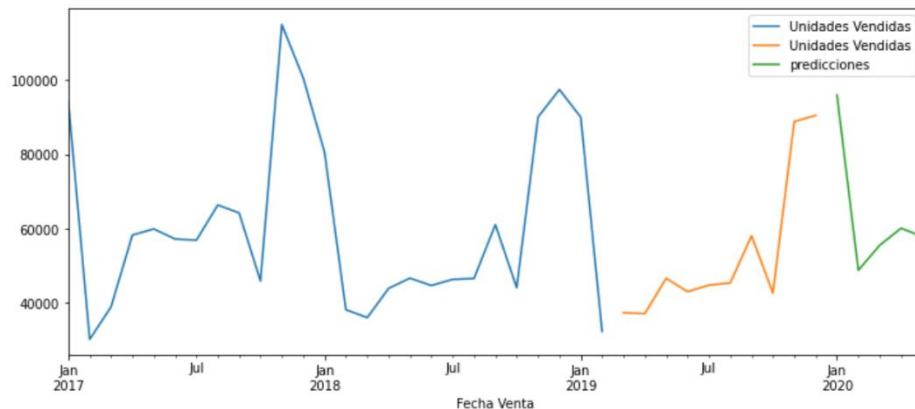


Ilustración 32 Gráfico train, test y predicción ARIMA mensual

### 5.3.2 ARIMA conjunto de datos diarios

Como con el resto de modelos vamos a realizar el modelo ARIMA introduciendo datos diarios de venta. Una vez tengamos los resultados compararemos ambos modelos.

Seguiremos el mismo proceso de trabajo que para el modelo con datos mensualizados. (Todo el código utilizado se encuentra en el notebook Modelo ARIMA (tfm) Véase punto 7 Anexo)).

1. Haremos carga de datos en archivo csv.
2. Verificaremos que el índice temporal está completo.
3. Chequeamos con adfuller la estacionalidad de la serie.
4. Resolveremos el orden del modelo ARIMA.
5. Dividiremos el conjunto en parte test y parte entrenamiento
6. Entrenaremos el modelo
7. Haremos las predicciones sobre el conjunto test
8. Predeciremos datos futuros a 30 días.

La representación diaria de la serie temporal queda de la siguiente manera.

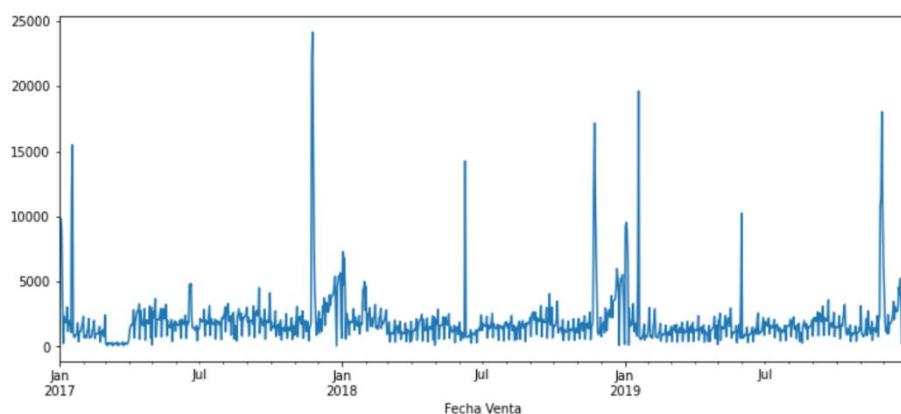


Ilustración 33 Gráfico serie temporal venta diaria

Podemos observar cierta estacionalidad al ver el gráfico, algo que nos confirmará adfuller.

```
adf_test(df_diario['Unidades Vendidas'])

#p-value < 0.05 estacionaria
#p-value > 0.05 no es estacionaria

1. ADF Statistic : -4.775710243738713
2. P-Value : 6.051339939311522e-05
3. Num Of Lags : 22
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 1072
5. Critical Values :
 1% : -3.4364647646486093
 5% : -2.864239892228526
10% : -2.5682075189699822
```

En el caso del modelo mensual nos salía un orden de modelo ARIMA (3,0,3). En este caso el modelo cambia a SARIMAX (1,0,0).

```
: stepwise_fit = auto_arima(df_diario['Unidades Vendidas'], seasonal=True,
                           suppress_warnings=True)

stepwise_fit.summary()

SARIMAX Results

Dep. Variable: y No. Observations: 1095
Model: SARIMAX(1, 0, 0) Log Likelihood -9651.610
Date: Thu, 24 Jun 2021 AIC 19309.221
Time: 17:16:17 BIC 19324.216
Sample: 0 HQIC 19314.895
- 1095

Covariance Type: opg

coef std err z P>|z| [0.025 0.975]
intercept 813.1306 88.463 9.192 0.000 639.746 986.516
ar.L1 0.5780 0.009 66.998 0.000 0.561 0.595
sigma2 2.651e+06 3.19e+04 83.112 0.000 2.59e+06 2.71e+06

Ljung-Box (L1) (Q): 0.03 Jarque-Bera (JB): 93184.02
Prob(Q): 0.85 Prob(JB): 0.00
Heteroskedasticity (H): 0.87 Skew: 4.14
Prob(H) (two-sided): 0.18 Kurtosis: 47.43
```

Ilustración 34 Tabla resultados SARIMAX diario

Una vez dividido el conjunto de datos en test y entrenamiento entrenamos el modelo:

#### Entrenamos el modelo

```
17]: from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(train['Unidades Vendidas'],order=(1,0,0))
model=model.fit()
model.summary()

17]: ARMA Model Results

Dep. Variable: Unidades Vendidas No. Observations: 1065
Model: ARMA(1, 0) Log Likelihood -9393.896
Method: css-mle S.D. of innovations 1638.324
Date: Thu, 24 Jun 2021 AIC 18793.792
Time: 17:16:24 BIC 18808.704
Sample: 01-01-2017 HQIC 18799.442
- 12-01-2019

coef std err z P>|z| [0.025 0.975]
const 1904.8660 119.085 15.996 0.000 1671.464 2138.268
ar.L1.Unidades Vendidas 0.5790 0.025 23.112 0.000 0.530 0.628

Roots

Real Imaginary Modulus Frequency
AR.1 1.7272 +0.0000j 1.7272 0.0000
```

Ilustración 35 Tabla resultados ARIMA diario

Ahora es el momento de realizar predicciones sobre el conjunto test:

Hacemos las predicciones sobre el conjunto test

```
|: start=len(train)
|: end=len(train)+len(test)-1
|: #index_future_dates=pd.date_range(start='2018-01-01',end='2020-12-31')
|: pred=model.predict(start=start,end=end,typ='levels').rename('ARIMA predictions')
|: #pred.index=index_future_dates
|: pred.plot(legend=True)
|: test['Unidades Vendidas'].plot(legend=True)
|: <AxesSubplot:xlabel='Fecha Venta'>
```

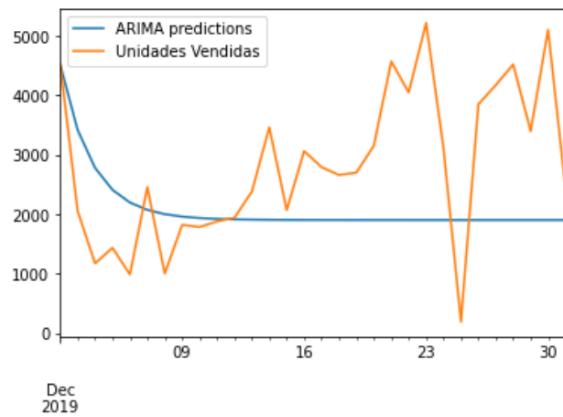


Ilustración 36 Gráfico Venta real vs predicción ARIMA diario

Como podemos observar el modelo no se ajusta nada bien al conjunto de datos diarios. Con los datos mensualizados se ajustaba bastante mejor.

La medida de error (rmse) nos arroja un resultado de 1541,5.

```
|: from sklearn.metrics import mean_squared_error
|: from math import sqrt
|: rmse=sqrt(mean_squared_error(pred,test['Unidades Vendidas']))
|: print(rmse)
```

1541.5217208217364

Sabiendo que no es el mejor modelo, y que no está ajustado vemos que resultado nos da la predicción sobre datos futuros a 30 días:

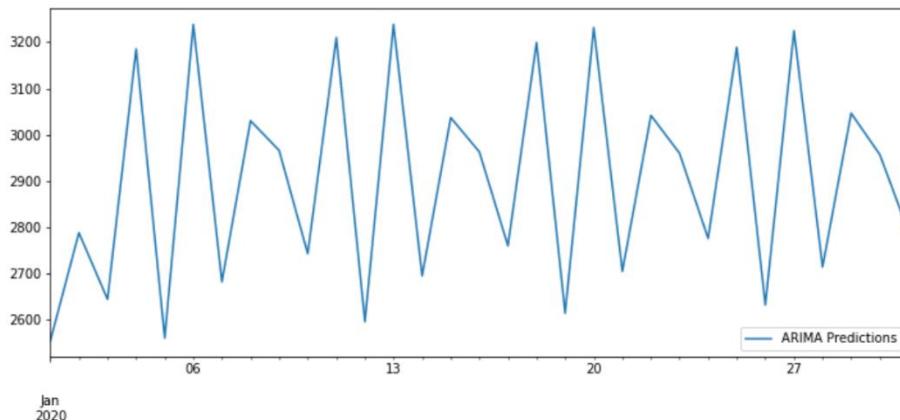


Ilustración 37 Gráfico predicción 30 días ARIMA diario

En el punto 6 Conclusiones veremos la comparativa de todos los modelos y la bondad si lo comparamos unos con otros. Sin embargo, el modelo ARIMA con datos diarios, es por ahora, el que peor se ha ajustado a nuestro conjunto de datos.

Representamos ahora los resultados para los modelos diario y mensual, utilizando ARIMA:

Modelo	Tipo de predicción	
ARIMA	Mensual	Diario
RMSE	5.829,45	1.541,52

Ilustración 38 RMSE ARIMA diario y mensual

## 5.4 Prophet

En este apartado se desarrollará el modelo de predicción de Facebook open source Prophet. Utilizaremos la herramienta de Jupyter Notebook para plasmar mediante código en Python todos los pasos necesarios y los cálculos que se precisen.

Para ejecutar el proceso se ha utilizado el código del notebook tfmTime\_Series\_Month.ipynb (véase sección 7 Anexos). Se han utilizado las siguientes librerías: fbprophet, pandas, numpy, matplotlib, dateutil, monthdelta, plotly y scikit-learn.

El estudio se ha llevado a cabo de la siguiente manera:

1. Carga de datos necesarios para la realización del experimento: se carga un fichero en formato csv con los datos necesarios.
2. Aplicación del método de predicción: iremos realizando diferentes pruebas de predicción de venta sobre los datos del mismo departamento que en modelos anteriores. Se irán aplicando técnicas que vayan mejorando los resultados de la prueba anterior.
3. Evaluación de los resultados: tendremos en cuenta medidas de accuracy y error para observar la bondad del modelo.

Para Prophet la entrada debe ser un dataframe con dos columnas: ds e y. La columna ds es la columna timestamp y debe tener el formato AAAA-MM-DD que es el esperado por Pandas. Puede ser AAAA-MM-DD HH:MM:SS en el caso de que sea una marca de tiempo. La columna y, en cambio, debe ser una columna numérica y representará la medida que queremos pronosticar.

### 5.4.1 Organización del trabajo

#### 5.4.1.1 Carga de las librerías necesarias

```
#pip install pystan == 2.19.1.1
#pip install fbprophet
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from dateutil import relativedelta
from monthdelta import monthdelta
import pickle
import fbprophet
from fbprophet import Prophet
from fbprophet.diagnostics import cross_validation

print('Prophet %s' % fbprophet.__version__)

Prophet 0.7.1
```

### 5.4.1.2 Carga de datos para realizar el experimento

En esta sección cargamos los conjuntos de datos (VentaDiaria 2017-2019.csv) y VentaMes2017-2019.csv) y los convertimos en dataframes.

```
df_MonthSales =pd.DataFrame()
df_DailySales =pd.DataFrame()
df_Promo =pd.DataFrame()

df_MonthSales= pd.read_csv('C:/Users/txico/El Corte Inglés, S.A/0tb y el master - Documentos/VentaMes2017-2019.csv'
                           ,';',index_col='Fecha Venta',parse_dates=True)

df_DailySales= pd.read_csv('C:/Users/txico/El Corte Inglés, S.A/0tb y el master - Documentos/VentaDia 2017-2019.csv'
                           ,';',index_col='Fecha Venta',parse_dates=True)

df_Promo= pd.read_csv('C:/Users/txico/El Corte Inglés, S.A/0tb y el master - Documentos/PromoDia 2017-2019.csv'
                           ,';')
```

### 5.4.1.3 Ajuste del modelo

La biblioteca del Prophet es una biblioteca de código abierto diseñada para hacer pronósticos de conjuntos de datos de series temporales univariadas. Tiene buen comportamiento a la hora de modelar series de tiempo que tienen múltiples estacionales y evita problemas de los anteriores algoritmos.

En primer lugar, desarrollaremos el modelo usando la venta diaria, posteriormente se realizará utilizando la venta mensual.

#### 5.4.1.3.1 Modelo Venta diaria

En este caso aplicaremos el método a la serie de venta univariante. Posteriormente, se añadirán las promociones relevantes que impactan en la venta.

Comprobamos que la entrada a Prophet tiene las características antes descritas: ser un dataframe compuesto por las columnas ds e y. La columna ds tiene que tener el formato esperado por Pandas, AAAA-MM-DD. La columna y debe ser numérica.

El primer paso es seleccionar los conjuntos de test y entrenamiento:

```
#fecha de entrenamiento del primer caso de test
test_date_from = "12/01/2019"
test_date_from

'12/01/2019'

#conjuntos de test y entrenamiento
df_test = df_DailySales.loc[df_DailySales.ds >= test_date_from]
df_train = df_DailySales.loc[df_DailySales.ds < test_date_from]
```

En este caso, hemos seleccionado el último mes del conjunto de datos (diciembre 2019) como conjunto test, y el resto de los datos como conjunto de entrenamiento. El conjunto df\_test contiene la venta diaria desde el 01/12/2019 al 31/12/2019, mientras que df\_train está compuesto por la venta diaria desde el 01/01/2017 hasta el 30/11/2019..

	ds	y		ds	y	
1064	2019-12-01	3279.0		0	2017-01-01	243.000
1065	2019-12-02	672.0		1	2017-01-02	2.299
1066	2019-12-03	1133.0		2	2017-01-03	2.414
1067	2019-12-04	1352.0		3	2017-01-04	622.000
1068	2019-12-05	766.0		4	2017-01-05	126.000

<class 'pandas.core.frame.DataFrame'>	<class 'pandas.core.frame.DataFrame'>
Int64Index: 31 entries, 1064 to 1094	Int64Index: 1064 entries, 0 to 1063
Data columns (total 2 columns):	Data columns (total 2 columns):
# Column Non-Null Count Dtype	# Column Non-Null Count Dtype
---	---
0 ds 31 non-null datetime64[ns]	0 ds 1064 non-null datetime64[ns]
1 y 31 non-null float64	1 y 1064 non-null float64
dtypes: datetime64[ns](1), float64(1)	dtypes: datetime64[ns](1), float64(1)
memory usage: 744.0 bytes	memory usage: 24.9 KB
None	None

Ilustración 39 Tabla test, train Prophet

Aplicaremos primero Prophet a la serie temporal de ventas sin añadir las promociones. Como hemos observado hasta ahora se observa una estacionalidad semanal y anual, la venta es diferente según el día de la semana y según el mes. Por ello, estableceremos los parámetros weekly\_seasonality y yearly\_seasonality como TRUE. Además, se aplicará un modelo multiplicativo (seasonality\_mode='multiplicative').

Para aplicar Prophet se crea una instancia de clase Prophet (m) y luego llamamos a sus métodos de ajuste y predicción.

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=True, seasonality_mode='multiplicative')
m.fit(df_train)
```

Para crear el periodo de predicción de 31 días, utilizamos el método make\_future\_dataframe. Y realizamos la predicción.

```
future=m.make_future_dataframe(periods=31)
```

```
future.tail()
```

ds	
1090	2019-12-27
1091	2019-12-28
1092	2019-12-29
1093	2019-12-30
1094	2019-12-31

Ilustración 40 Tabla dataframe Future

```
forecast=m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
1090	2019-12-27	3234.573208	1046.391975	5567.626300
1091	2019-12-28	3360.736395	1239.489514	5547.483774
1092	2019-12-29	2607.227800	243.322705	4923.647186
1093	2019-12-30	2828.170586	610.526607	5127.115853
1094	2019-12-31	2843.279783	528.031208	5093.618013

Ilustración 41 Tabla predicción 30 días Prophet diario

Con esto obtenemos un dataframe con en el que cada línea es una fecha con la predicción de venta para ese día (yhat) y su intervalo de incertidumbre del 80% (yhat\_lower - yhat\_upper).

Para comparar la predicción con la venta real de esos días, creamos un dataframe (comp\_df) indexado por la fecha de venta (ds) para añadir la venta real (la variable y de df\_DailySales).

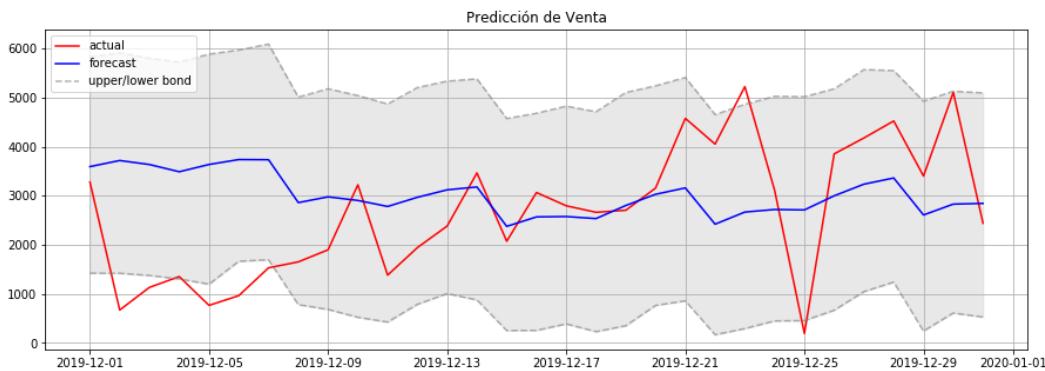
```
comp_df=make_comparison_dataframe(df_DailySales,forecast)
comp_df.tail()
```

	ds	yhat	yhat_lower	yhat_upper	y
	2019-12-27	3234.573208	1046.391975	5567.626300	4176.0
	2019-12-28	3360.736395	1239.489514	5547.483774	4522.0
	2019-12-29	2607.227800	243.322705	4923.647186	3401.0
	2019-12-30	2828.170586	610.526607	5127.115853	5104.0
	2019-12-31	2843.279783	528.031208	5093.618013	2439.0

Ilustración 42 Tabla dataframe comp\_df

Para continuar con la comparación, lo representamos gráficamente, mostrando los valores reales (actual) y las predicciones (forecast) para el mes de diciembre.

```
show_forecast(comp_df,31,31, 'Predicción de Venta')
plt.show()
```



*Ilustración 43 Gráfico Venta real vs predicción año 2019*

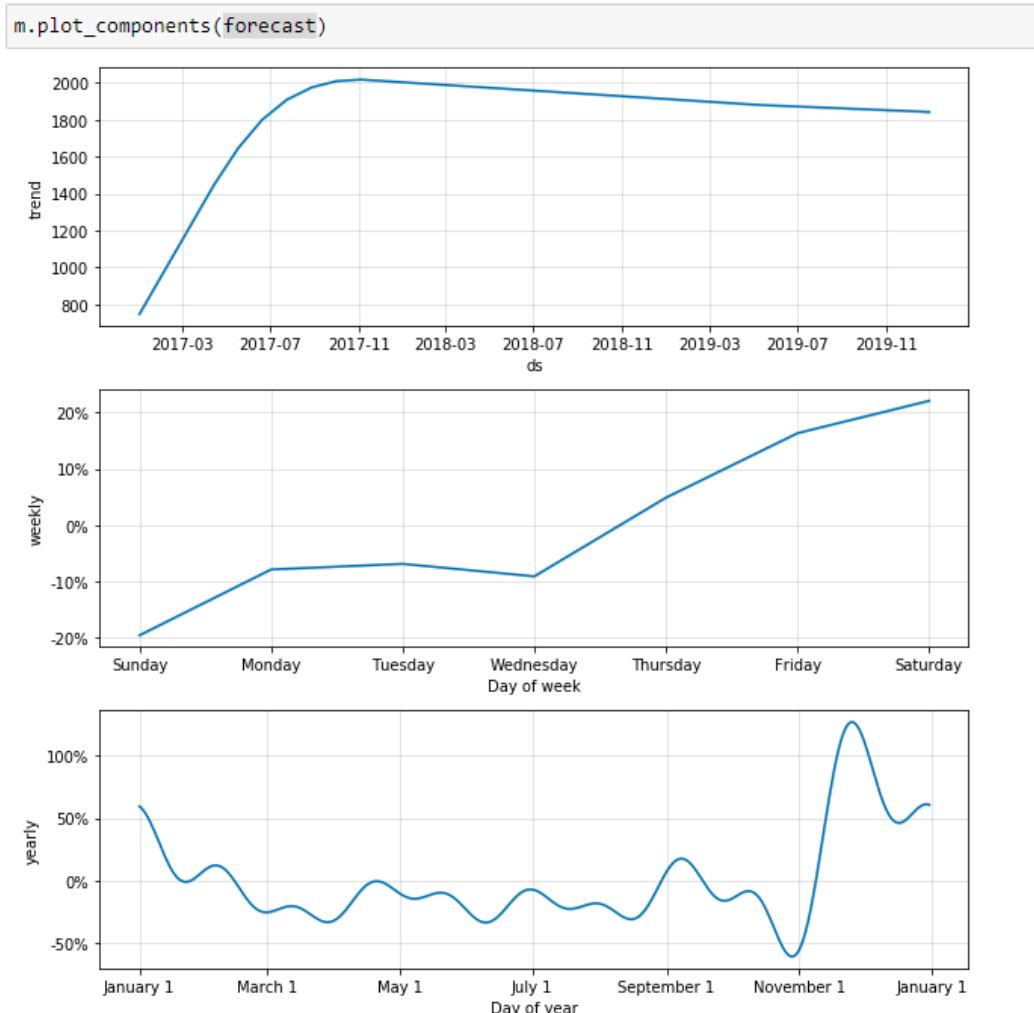
Las series en temporales con frecuencia tienen cambios bruscos en sus trayectorias. Prophet utiliza por defecto un modelo lineal, pero, aunque detecta los puntos de cambio, existen unos argumentos de entrada que podemos usar:

- `changepoint_prior_scale`: Si los cambios de tendencia se sobreajustan (demasiada flexibilidad) o no se ajustan bien (no hay suficiente flexibilidad), puede ajustar la intensidad de la dispersión antes de usar el argumento de entrada `changepoint_prior_scale`. De forma predeterminada, este parámetro está establecido en 0.05. Incrementarlo hará que la tendencia sea más flexible.
- `changepoint_range`: Por defecto, los puntos de cambio solo se infieren para el primer 80% de la serie de tiempo para tener suficiente pista para proyectar la tendencia hacia adelante y evitar sobreajuste al final de la serie de tiempo. Este valor predeterminado funciona en muchas situaciones, pero no en todas, y se puede cambiar mediante el argumento `changepoint_range`. Por ejemplo, `m = Prophet(changepoint_range=0.9)` colocará los puntos de cambio potenciales en el primer 90% de la serie de tiempo.

Respecto a las estacionalidades, el orden de Fourier (las estacionalidades se estiman usando series de Fourier) es un parámetro que determina qué tan rápido puede cambiar la estacionalidad. Y se puede modificar con los siguientes parámetros:

- `yearly_seasonality`: El valor por defecto para la estacionalidad anual es 10. Si lo incrementamos podemos conseguir que la estacionalidad se ajuste a ciclos de cambio más rápidos, pero también puede conducir a un sobreajuste.
- `Weekly_seasonality`: El valor por defecto para la estacionalidad semanal es 3.

En la siguiente gráfica se representan los componentes de la serie temporal, podemos encontrar la tendencia y las estacionalidades anual y semanal.



*Ilustración 44 Gráfico componentes serie temporal*

La tendencia tiene un fuerte tramo positivo desde principios de 2017, hasta noviembre de ese mismo año, a partir de ese momento comienza un leve decrecimiento, que continua hasta el final de la serie (31 diciembre 2019).

Respecto a la estacionalidad semanal, podemos observar que la venta comienza a ser mayor a partir del miércoles hasta llegar a su máximo el sábado, el domingo es la más baja. También podemos observar la estacionalidad anual, a finales de octubre es mínima y a finales de noviembre es máxima.

Para poder comparar este y otros modelos, se realizarán diferentes variaciones y se evaluará el rendimiento de cada predicción en un horizonte de 31 días, para ello, se realizará validación cruzada, generando 12 conjuntos test de 31 días cada uno. Como el conjunto de datos tiene 1095 días comenzaremos con un conjunto de entrenamiento de 722 días ( $1095 - 12 \times 31 - 1$ ).

- Modelo multiplicativo con estacionalidad semanal y anual

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=True, seasonality_mode='multiplicative')
m.fit(df_DailySales)
pr_cv=cross_validation(m,initial='722 days',period='31 days', horizon='31 days')
pr_cv.head()
```

ds	yhat	yhat_lower	yhat_upper	y	cutoff
2018-12-25	3201.399593	1268.836965	5542.018442	87.0	2018-12-24
2018-12-26	3044.033404	906.870350	5027.942876	4600.0	2018-12-24
2018-12-27	3642.243377	1538.057329	5942.015419	5252.0	2018-12-24

Ilustración 45 Tabla Modelo multiplicativo 30 días Prophet diario

```
comp_df=make_comparison_dataframe(df_DailySales,pr_cv)
show_forecast(comp_df,372,372, 'Predicción de Venta')
```

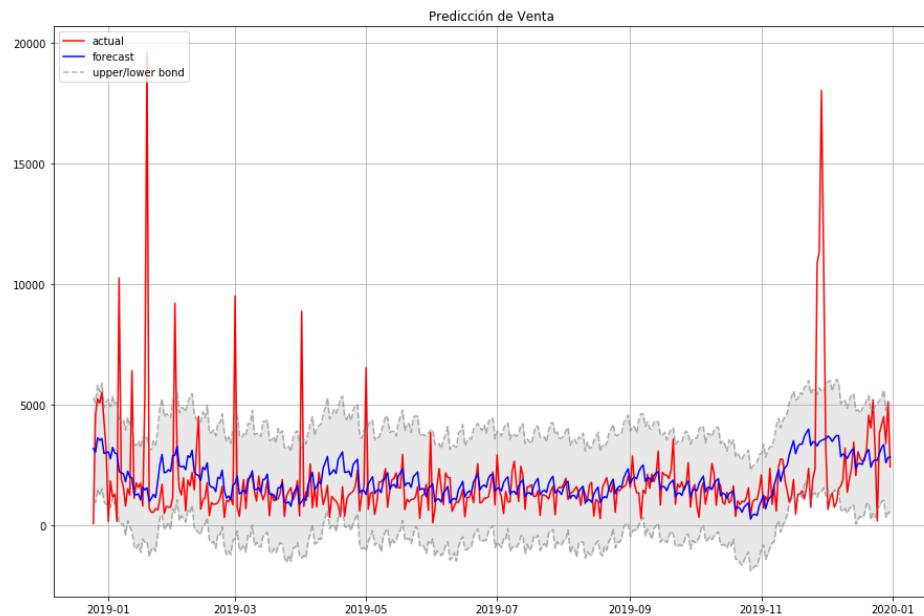


Ilustración 46 Gráfico Venta diaria real vs predicción año 2019

Calculamos algunas estadísticas útiles para conocer el rendimiento de la predicción (yhat, yhat\_lowery y yhat\_upper comparados con y). Los estimadores que calcula son: el error cuadrático medio (MSE), el error cuadrático medio medio (RMSE), el error absoluto medio (MAE), el error porcentual absoluto medio (MAPE), el error porcentual absoluto medio (MDAPE) y la cobertura de las estimaciones yhat\_lowery yhat\_upper. Para ello utilizamos performance\_metrics:

```
df_p = performance_metrics(pr_cv)
df_p.tail()
```

	horizon	mse	rmse	mae	mape	mdape	coverage
23	27 days	1.006794e+07	3173.001877	1287.473272	0.017561	0.478637	0.891892
24	28 days	1.091227e+07	3303.373012	1289.018679	0.576665	0.476261	0.939189
25	29 days	4.428565e+06	2104.415695	1033.344480	0.621737	0.476261	0.943694
26	30 days	9.466223e+06	3076.722759	1390.210002	0.631721	0.445891	0.891892
27	31 days	9.550424e+06	3090.376072	1402.707859	0.678926	0.492929	0.889640

Ilustración 47 Tabla performance metrics

Para realizar un seguimiento, recopilamos los datos agregados medios de nuestra validación cruzada de la siguiente manera:

```
df_p.mean()
```

```
horizon      17 days 12:00:00
mse           3585316.282495
rmse          1723.626186
mae            998.9715
mape           0.887676
mdape          0.413136
coverage       0.926641
dtype: object
```

Observamos valores bastante altos tanto de RMSE como de MAPE, realizaremos modificaciones en el modelo para tratar de mejorar los resultados.

- Modelo aditivo con estacionalidad semanal y anual.

Utilizamos primero el modelo aditivo con las mismas características de estacionalidad (semanal y anual) y sin ajustar la tendencia.

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=True)
m.fit(df_DailySales)
pr_cv=cross_validation(m,initial='722 days',period='31 days', horizon='31 days')
pr_cv.head()
```

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2018-12-25	3061.259926	867.359591	5244.452909	87.0	2018-12-24
1	2018-12-26	2916.479244	764.621159	4996.692413	4600.0	2018-12-24
2	2018-12-27	3409.949576	1216.272078	5622.363797	5252.0	2018-12-24
3	2018-12-28	3321.815649	1032.044890	5573.883252	5083.0	2018-12-24
4	2018-12-29	3335.177373	1161.645561	5396.254594	5530.0	2018-12-24

Ilustración 48 Tabla Modelo aditivo 30 días Prophet diario

```
comp_df=make_comparison_dataframe(df_DailySales,pr_cv)
show_forecast(comp_df,372,372, 'Predicción de Venta')
```

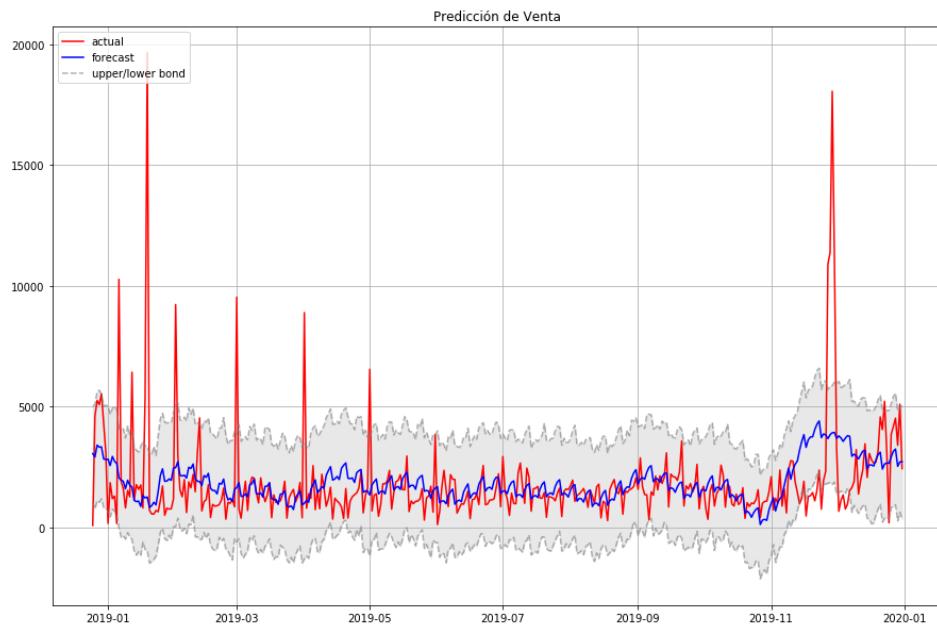


Ilustración 49 Gráfico Venta diaria real vs predicción año 2019

Comprobamos si mejora la predicción:

```
df_p = performance_metrics(pr_cv)
df_p.tail()
```

```
df_p.mean()

horizon      17 days 12:00:00
mse           3599928.178109
rmse          1729.878388
mae            983.146693
mape           0.85141
mdape          0.40147
coverage       0.908784
dtype: object
```

Comprobamos que se produce una ligera mejora del estimador MAPE, pasando de 88% a 85%. Aun así, continúa siendo demasiado alto.

- Modelo aditivo (yearly\_seasonality=15, Weekly\_seasonality=True, changepoint\_range =0.85).

Se han probado varias combinaciones respecto a la estacionalidad en el modelo aditivo. El que mejor resultados nos ofrece es yearly\_seasonality=15, con esto conseguimos mejor ajuste. Mantenemos por defecto Weekly\_seasonality=True.

Para la tendencia, ajustaremos los parámetros respecto a los puntos de cambio de tendencia: mantenemos changepoint\_prior\_scale en 0.05, y changepoint\_range lo incrementamos de 80% al 85%.

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=15, changepoint_range=0.85)
m.fit(df_DailySales)
pr_cv=cross_validation(m,initial='722 days',period='31 days', horizon='31 days')
pr_cv.head()
```

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2018-12-25	4343.395641	2399.585902	6444.111302	87.0	2018-12-24
1	2018-12-26	3965.620221	1970.257361	5995.012221	4600.0	2018-12-24
2	2018-12-27	4185.295539	2253.009405	6103.659488	5252.0	2018-12-24

Ilustración 50 Tabla Modelo aditivo 30 días Prophet diario

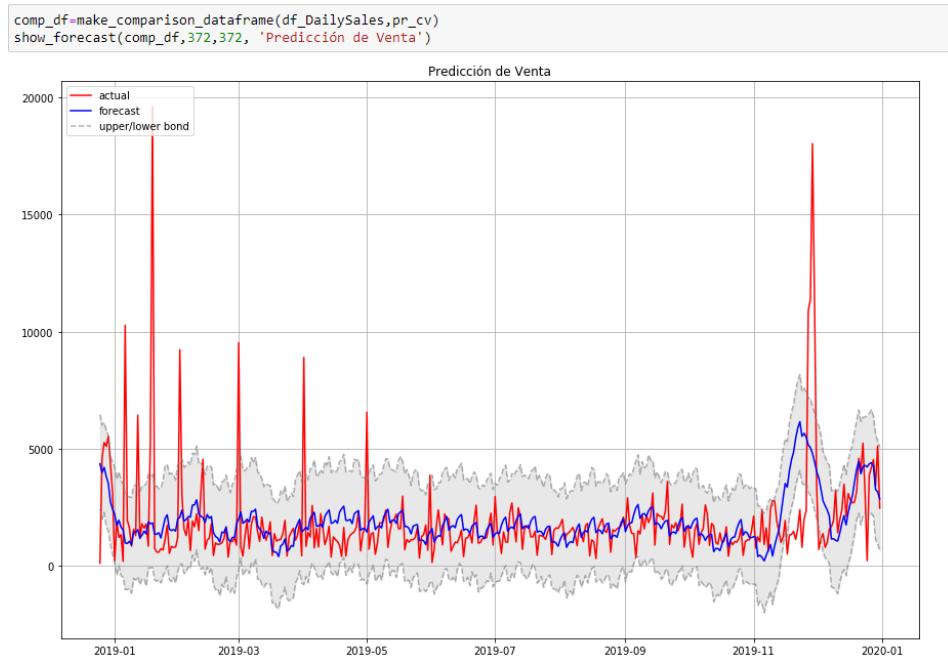


Ilustración 51 Gráfico Venta diaria real vs predicción año 2019

```
df_p = performance_metrics(pr_cv)
df_p.mean()

horizon    17 days 12:00:00
mse        3543833.990723
rmse       1741.11962
mae        970.262339
mape       0.8444845
mdape      0.376546
coverage   0.916667
dtype: object
```

Continúan mejorando muy levemente, MAPE, MDAP y MAE.

- Modelo aditivo (yearly\_seasonality=15, Weekly\_seasonality=True, changepoint\_range =0.85) incluyendo vacaciones en España.

Para este modelo utilizaremos el anterior añadiendo las festividades del país. Para ello, utilizaremos add\_country\_holidays, las vacaciones de cada país las proporciona el paquete en Python, holidays. De este modo, el modelo tendrá en cuenta esas fechas significativas en cada uno de los años.

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=15, changepoint_range=0.85)
m.add_country_holidays(country_name='ES')
m.fit(df_DailySales)
pr_cv=cross_validation(m,initial='722 days',period='31 days', horizon='31 days')
pr_cv.head()
```

Las siguientes festividades están incluidas en el paquete:

0	Año nuevo (Trasladado)
1	Epifanía del Señor
2	Día del Trabajador
3	Asunción de la Virgen
4	Día de la Hispanidad
5	Todos los Santos
6	Día de la Constitución Española
7	La Inmaculada Concepción
8	Navidad
9	Año nuevo
10	Epifanía del Señor (Trasladado)
11	La Inmaculada Concepción (Trasladado)

Ilustración 52 Tabla festivos España

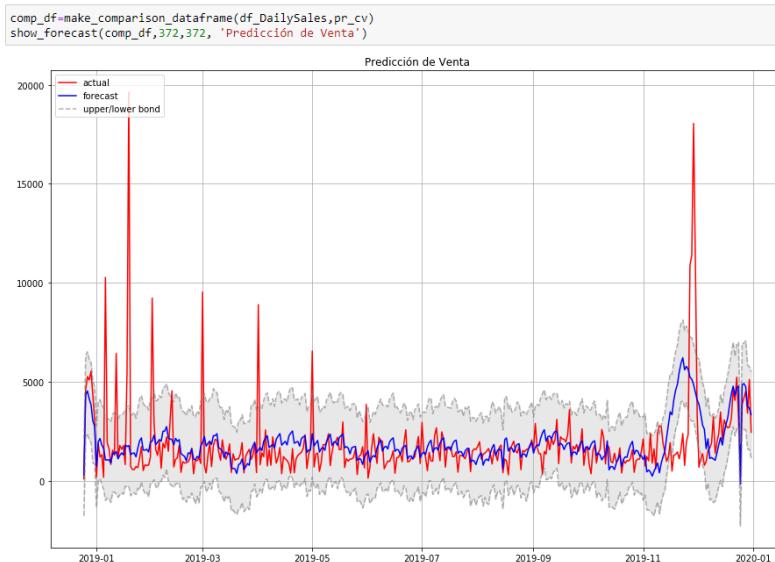


Ilustración 53 Gráfico Venta diaria real vs predicción año 2019

```
df_p = performance_metrics(pr_cv)
df_p.mean()

horizon    17 days 12:00:00
mse        3443984.165994
rmse       1714.003798
mae         942.202246
mape        0.7464
mdape       0.38118
coverage    0.923665
dtype: object
```

Se produce una mejora significativa respecto al error MAPE, reduciéndose de 0.84 a 0.74.

- Modelo aditivo (yearly\_seasonality=15, Weekly\_seasonality=True, changepoint\_range =0.85) incluyendo vacaciones en España y eventos.

Por último, añadimos los eventos promocionales significativos para la venta (Black Friday, día sin IVA, Semana de Internet). Para incluir estos eventos al modelo, utilizamos el fichero de promociones, que previamente transformamos en un dataframe válido para Prophet, df\_Promo.

Realizamos algunas transformaciones sobre el dataframe para añadir las columnas upper\_window y lower\_window. De esa manera podemos indicar cuantos días previos y posteriores, a la fecha de una promoción determinada, va a tener en cuenta el modelo. En este caso se establece tener en cuenta los días anterior y posterior, en las promociones más importantes, y se añaden los dos días siguientes al Black Friday, ya que la oferta se amplía a los días posteriores.

```
df_Promo['upper_window']=0
df_Promo['lower_window']=0

df_Promo.loc[df_Promo.holiday=='BLACK FRIDAY','upper_window']=2
df_Promo.loc[df_Promo.holiday=='BLACK FRIDAY','lower_window']=-1
df_Promo.loc[df_Promo.holiday=='DIA SIN IVA','lower_window']=-1
df_Promo.loc[df_Promo.holiday=='CYBER MONDAY','lower_window']=-1
```

Podemos incluir este nuevo dataframe compuesto por la fecha, el nombre de la promoción y los límites de días superior e inferior, en el modelo anterior de la siguiente manera.

```
m = Prophet(weekly_seasonality=True, yearly_seasonality=15, holidays=df_Promo, changepoint_range=0.85)
m.add_country_holidays(country_name='ES')
m.fit(df_DailySales)
pr_cv=cross_validation(m,initial='722 days',period='31 days', horizon='31 days')
pr_cv.head()
```

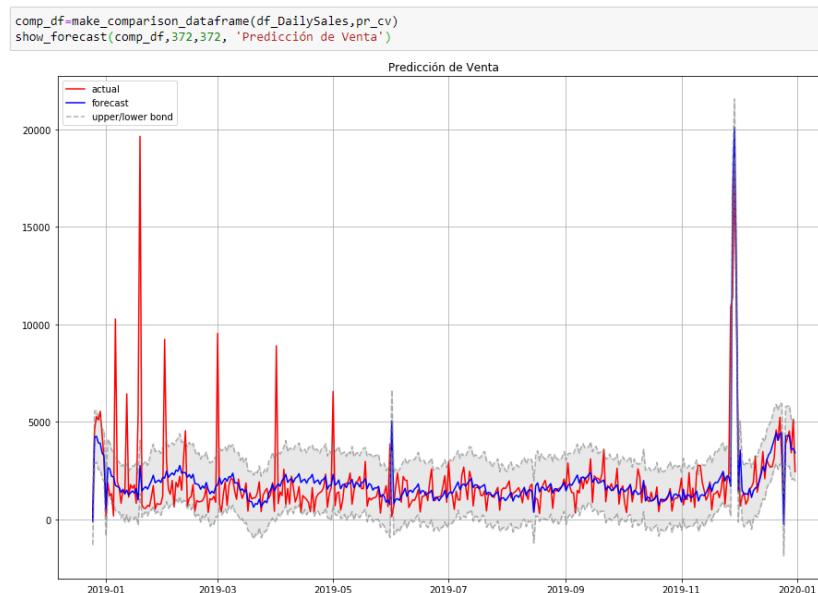


Ilustración 54 Gráfico Venta diaria real vs predicción año 2019

```
df_p = performance_metrics(pr_cv)
df_p.mean()

horizon      17 days 12:00:00
mse          2645070.192158
rmse         1459.351085
mae           778.068966
mape          0.7273
mdape         0.304811
coverage      0.931548
dtype: object
```

Con la inclusión de estos eventos significativos para la venta, se consigue mejorar los resultados de todos los errores (RMSE, MAE, MAPE, MDAPE).

Tras la realización de este último modelo, se incluye una tabla comparativa para cada uno de los ejemplos realizados para el modelo de venta diaria diario utilizando Prophet:

Modelos Prophet con validación cruzada	RMSE	MAE	MAPE	MDAPE
Modelo multiplicativo con estacionalidad semanal y anual	1.724	999	0,8876	0,4131
Modelo aditivo con estacionalidad semanal y anual.	1.730	983	0,8514	0,4015
Modelo aditivo (yearly_seasonality=15, Weekly_seasonality=True, changepoint_range =0.85)	1.741	970	0,8448	0,3765
Modelo aditivo (yearly_seasonality=15, Weekly_seasonality=True, changepoint_range =0.85) incluyendo vacaciones en España	1.714	942	0,7464	0,3812
Modelo aditivo (yearly_seasonality=15, Weekly_seasonality=True, changepoint_range =0.85) incluyendo vacaciones en España y eventos.	1.459	778	0,7273	0,3048

Ilustración 55 Tabla comparativa errores Prophet diario

Comprobamos que el que mejor comportamiento tiene es el último, todos los errores presentan valores mínimos respecto al resto de ejemplos.

Para este modelo seleccionado representamos la predicción para los siguientes 31 días (mes de diciembre) junto con los valores reales.

```
plt.rcParams["figure.figsize"] = (12,8)
fig, ax = plt.subplots()
ax.pr_cv[-31:].ds
plt.plot(pr_cv[-31:].ds,pr_cv[-31:].yhat,label='Prophet_Predictions')
plt.plot(pr_cv[-31:].ds,pr_cv[-31:].y,label='Actual Sales')
plt.legend(loc="upper left")
plt.show()
```

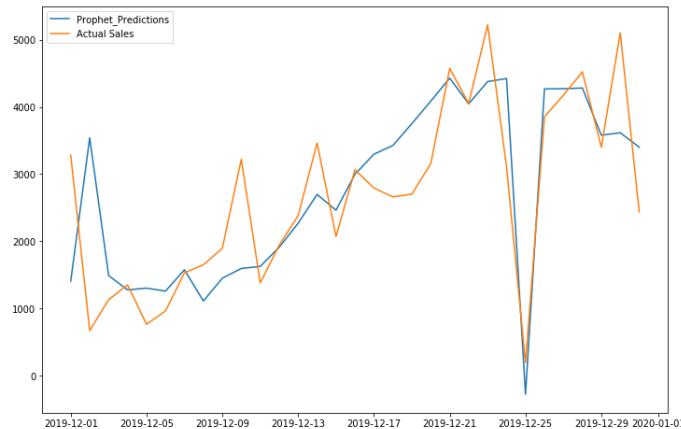


Ilustración 56 Gráfico Venta diaria real vs predicción 31 días

Al igual que en el resto de modelos, calculamos el RMSE para la predicción de los 31 días.

```
from sklearn.metrics import mean_squared_error, confusion_matrix
from math import sqrt
rmse_lr=sqrt(mean_squared_error(pr_cv[-31:].yhat,pr_cv[-31:].y))

rmse_lr
899.7981615439065
```

#### 5.4.1.3.2 Modelo Venta Mensual

En este caso, utilizaremos la venta mensual, con el objetivo de pronosticar la venta de los próximos 10 meses. Para esto hemos construido el dataframe df\_MonthSales con el formato adecuado para usar en Prophet, ds indica el mes de la venta e y nos indica la suma de la venta de todos los días de cada mes.

```
df_MonthSales.head()
```

	ds	y
0	2017-01-01	94158
1	2017-02-01	30152
2	2017-03-01	38901
3	2017-04-01	58229
4	2017-05-01	59880

Ilustración 57 Tabla Venta mensual 2017-2019

Observamos en el dataframe que la venta total del mes está asignada al día 1 de cada mes. Prophet puede utilizarse para ajustar los datos mensuales. Sin embargo, el modelo es de tiempo continuo, lo que significa que puede ofrecer resultados extraños si ajusta el modelo a los datos mensuales y luego solicita pronósticos diarios. Este sería el pronóstico para el próximo año:

```
m = Prophet().fit(df_MonthSales)
future = m.make_future_dataframe(periods=365)
fcst = m.predict(future)
fig = m.plot(fcst)

INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

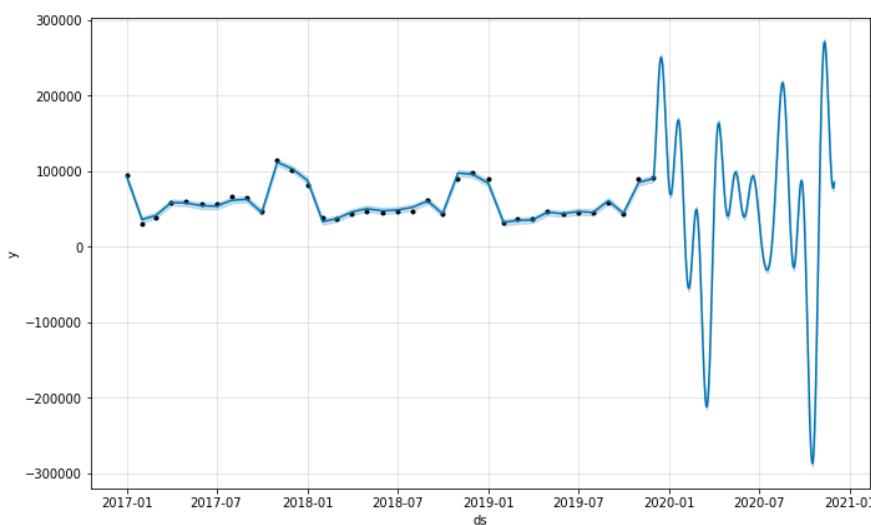


Ilustración 58 Gráfico Predicción mensual Prophet

Cuando ajustamos la estacionalidad anual, solo tiene datos para el primer día de cada mes y los componentes de estacionalidad para los días restantes no son identificables y están sobreajustados. Esto se puede ver claramente haciendo MCMC para ver la incertidumbre en la estacionalidad:

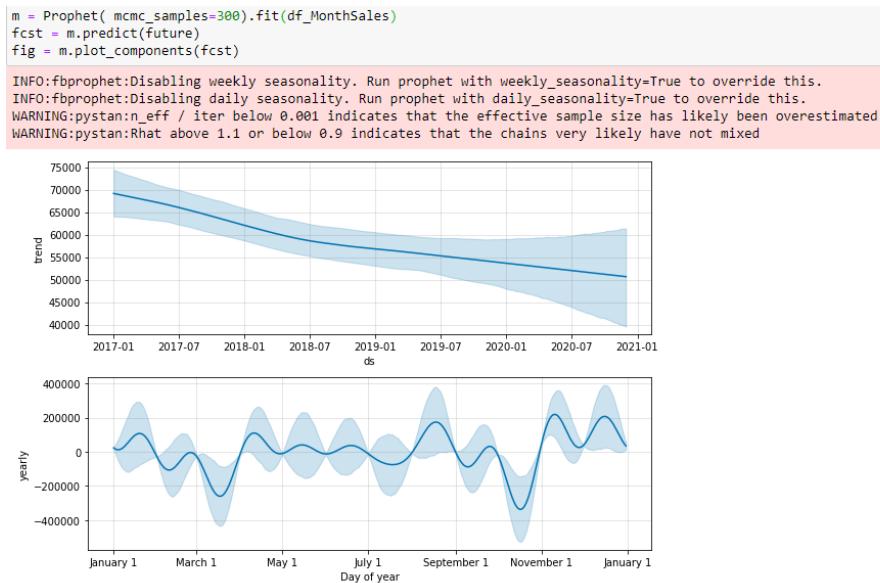


Ilustración 59 Gráfico componentes serie mensual Prophet

La estacionalidad tiene poca incertidumbre al comienzo de cada mes donde hay puntos de datos, pero tiene una varianza posterior muy alta en el medio. Hay que ajustar Prophet a los datos mensuales, para que solo haga pronósticos mensuales. Esto se puede hacer pasando la frecuencia a mensual, mediante make\_future\_dataframe.

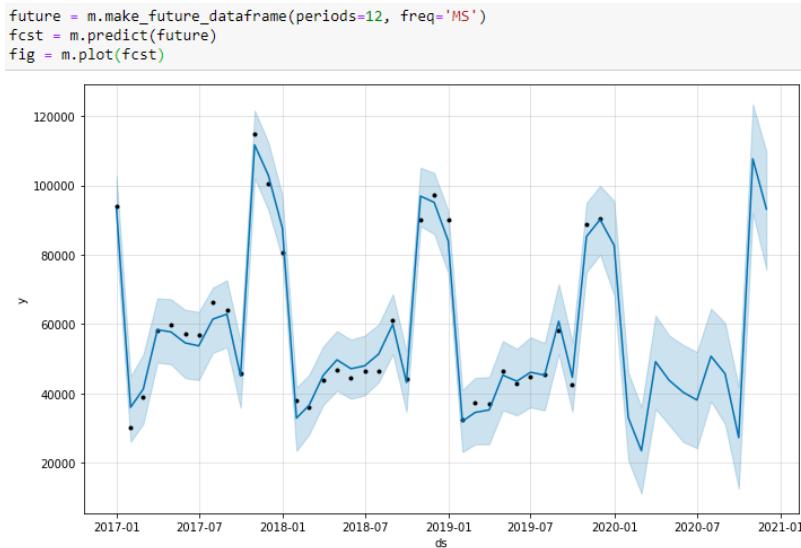


Ilustración 60 Gráfico predicción ajustada mensual Prophet

Al igual que en el modelo de venta diaria, es necesario hacer diferentes ajustes del modelo para mejorar la predicción, en este caso, de los 10 meses. Para ello, estableceremos nuestros conjuntos test y train.

```
#fecha de entrenamiento del primer caso de test
test_date_from = "03/01/2019"
test_date_from

'03/01/2019'

#conjuntos de test y entrenamiento
df_test = df_MonthSales.loc[df_MonthSales.ds >= test_date_from]
df_train = df_MonthSales.loc[df_MonthSales.ds < test_date_from]
```

De este modo tendremos un conjunto test de los 10 últimos meses del conjunto de datos MonthSales y un conjunto de entrenamiento de 26 meses, desde enero de 2017. Aplicaremos diferentes variaciones del modelo al conjunto tipo test y compraremos los resultados de la predicción con los datos reales de venta de los últimos 10 meses. Primero utilizaremos el modelo aditivo sin ninguna modificación de parámetros y obtenemos los siguientes resultados.

- Modelo Aditivo:

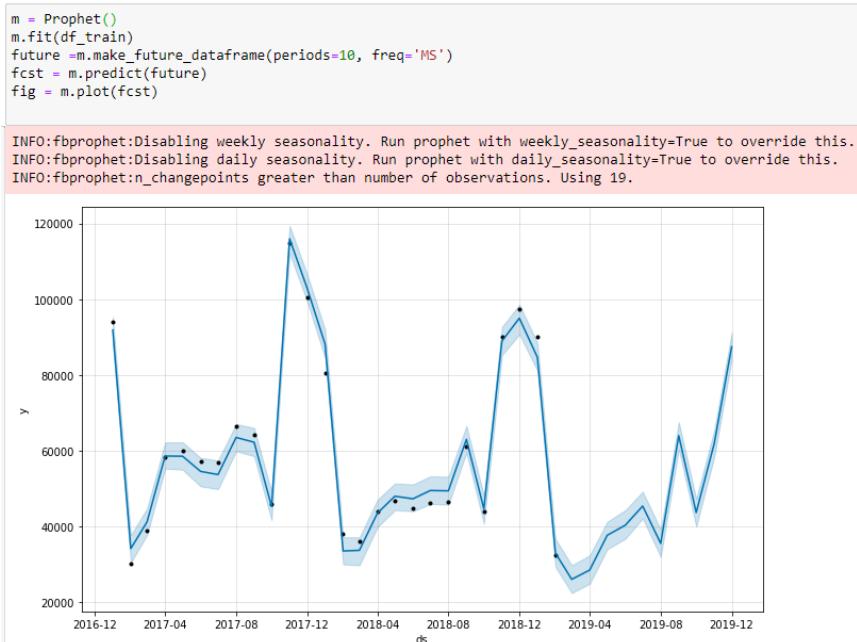


Ilustración 61 Gráfico modelo aditivo mensual Prophet

Ahora comparamos los resultados de la predicción con los datos reales de venta:

```
comp_df=make_comparison_dataframe(df_MonthSales,fcst)
comp_df.tail()
```

ds	yhat	yhat_lower	yhat_upper	y
2019-08-01	35494.997678	31910.130585	39175.334463	45341
2019-09-01	63957.020025	60331.547084	67545.407150	58027
2019-10-01	43653.808253	39960.946229	47325.683769	42642
2019-11-01	61870.556941	58266.576410	65401.246662	88793
2019-12-01	87433.878048	84001.552988	91156.739990	90449

Ilustración 62 Tabla dataframe comp\_df

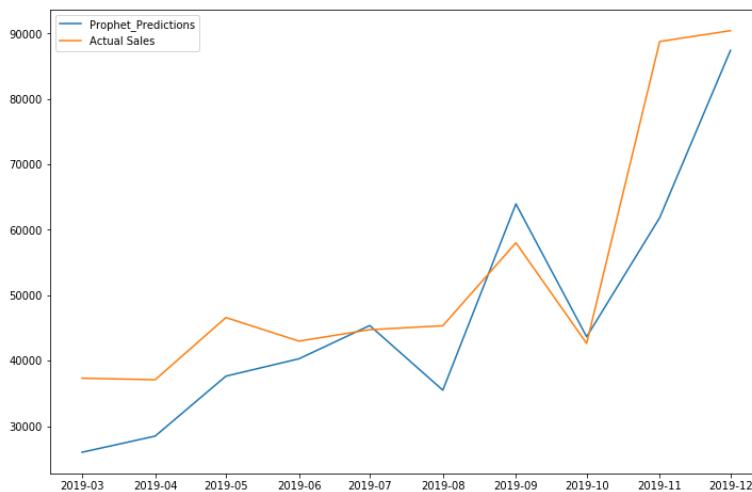


Ilustración 63 Gráfico Venta mensual real vs predicción 10 meses

Se calcula ahora el RMSE:

```
rmse_lr=sqrt(mean_squared_error(fcst[-10:].yhat,df_test.y))
rmse_lr
10757.409540715491
```

- Modelo multiplicativo

```
m = Prophet(seasonality_mode='multiplicative')
m.fit(df_train)
future = m.make_future_dataframe(periods=10, freq='MS')
fcst = m.predict(future)
fig = m.plot(fcst)
```

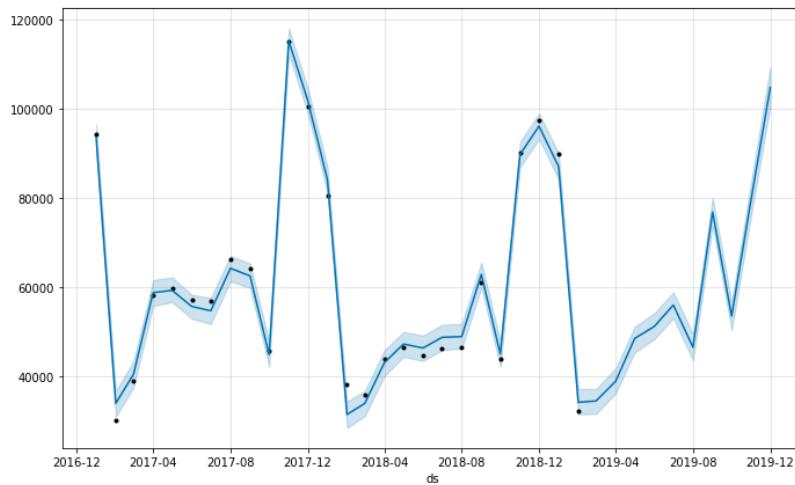


Ilustración 64 Gráfico modelo multiplicativo mensual Prophet

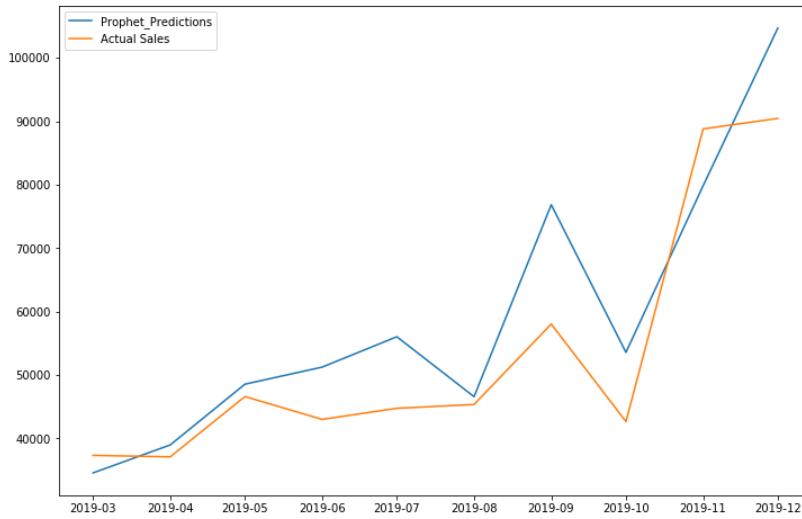


Ilustración 65 Gráfico Venta mensual real vs predicción 10 meses

Y calculamos el RMSE para compararlo con el modelo aditivo.

```
rmse_lr=sqrt(mean_squared_error(fcst[-10:].yhat,df_test.y))

rmse_lr
9838.550107183219
```

Comprobamos que se produce una mejora en el RMSE utilizando el modelo multiplicativo. Como, la diferencia no es tan significativa, vamos a probar ambos para ajustar parámetros y tratar de mejorar el modelo.

- Modelo multiplicativo ajustando estacionalidad.

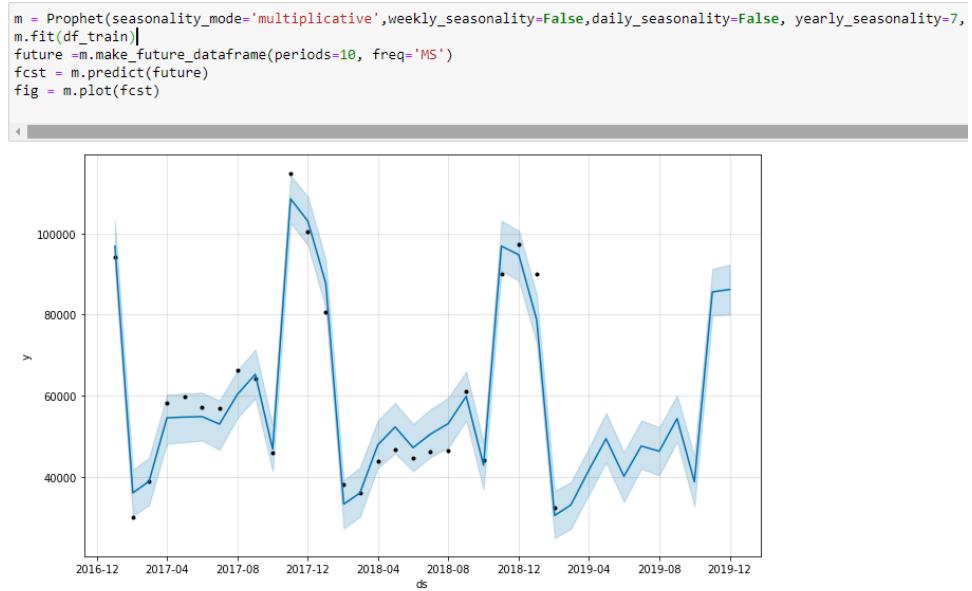


Ilustración 66 Gráfico modelo multiplicativo ajustado mensual Prophet

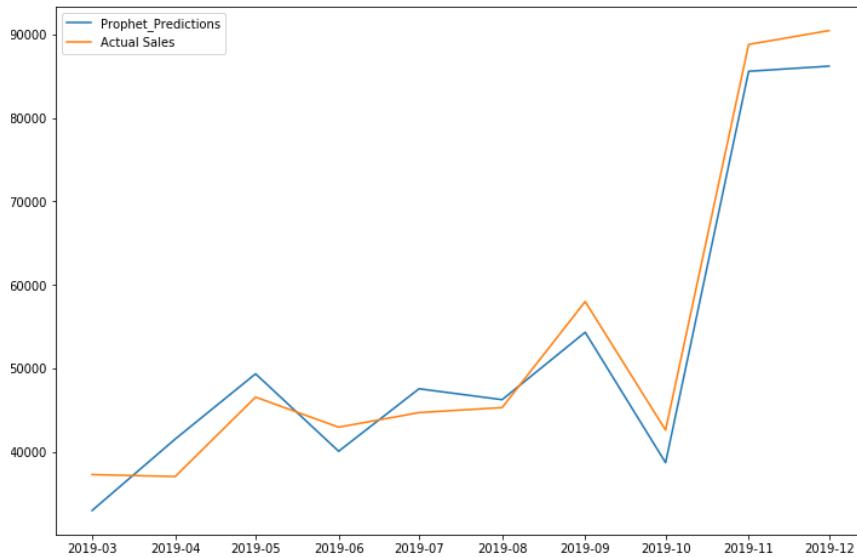


Ilustración 67 Gráfico Venta mensual real vs predicción 10 meses

Tras varias pruebas ajustando parámetros, comprobamos que los mejores resultados que nos ofrece el modelo corresponden a un ajuste de la estacionalidad anual (yearly\_seasonality=7). De esta manera hemos conseguido reducir el error RMSE a 3.473.48.

```
rmse_lr=sqrt(mean_squared_error(fcst[-10:].yhat,df_test.y))
rmse_lr
```

3473.484953934847

- Modelo aditivo ajustando estacionalidad

```
m = Prophet(weekly_seasonality=False,daily_seasonality=False, yearly_seasonality=7, n_changepoints=19)
m.fit(df_train)
future = m.make_future_dataframe(periods=10, freq='MS')
fcst = m.predict(future)
fig = m.plot(fcst)
```

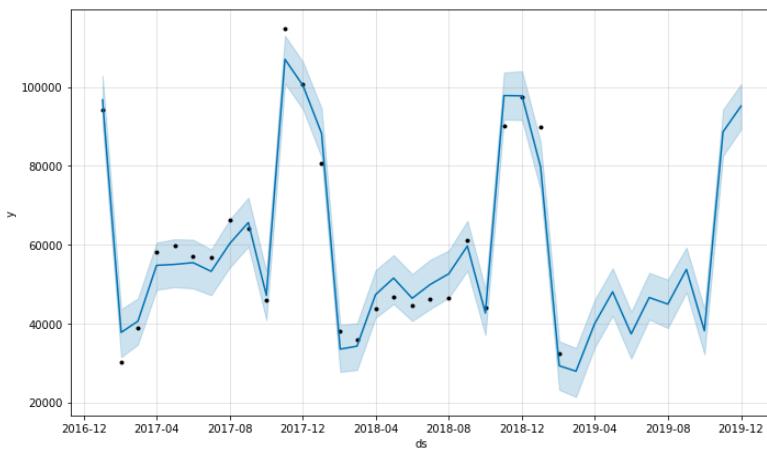


Ilustración 68 Gráfico modelo aditivo ajustado mensual Prophet

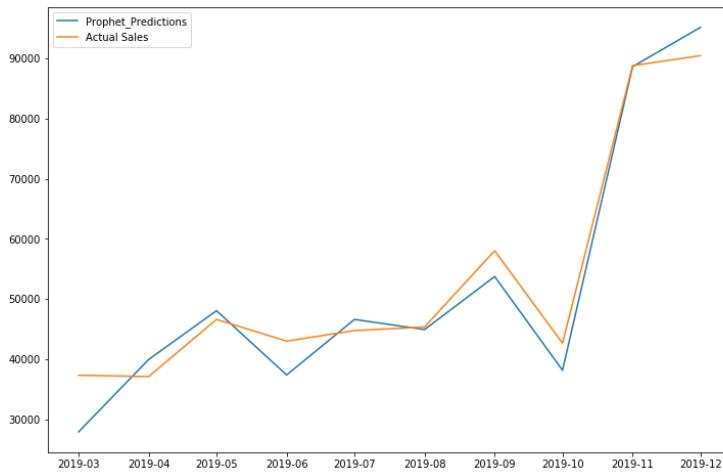


Ilustración 69 Gráfico Venta mensual real vs predicción 10 meses

El modelo aditivo no mejora los resultados de la predicción, mejora significativamente respecto al modelo aditivo sin ajustar, pero no los resultados con el modelo multiplicativo.

```
rmse_lr=sqrt(mean_squared_error(fcst[-10:].yhat,df_test.y))
rmse_lr
```

4418.261824507179

En el caso del modelo mensual, no se ha utilizado validación cruzada, ya que el procedimiento Prophet está desarrollado para trabajar con datos diarios. En la siguiente tabla se muestra la comparativa entre los diferentes modelos utilizados para la venta mensual.

Modelo Prophet Mensual	RMSE
Aditivo	10.757,41
Aditivo Ajustado	4.418,26
Multiplicativo	9.838,55
Multiplicativo Ajustado	3.473,48

*Ilustración 70 Tabla comparativa RMSE Prophet diario*

El mejor resultado de rmse lo hemos obtenido al utilizar la estacionalidad multiplicativa y reduciendo el ajuste de la estacionalidad anual, obteniendo un RMSE = 3.473,48.

Por último, mostramos los mejores resultados de los modelos para venta diaria y mensual, de la predicción a 30 días y 10 meses, respectivamente. Para ello utilizamos el RMSE.

Modelo	Tipo de predicción	
	Mensual	Diario
PROPHET		
RMSE	3.473,48	899,79

*Ilustración 71 Tabla RMSE Prophet diario y mensual*

## 5.5 XGBOOST

En este apartado se desarrollará el modelo de predicción Xgboost desarrollado por Tianqi Chen. Una librería que se ha vuelto viral en estos tiempos y se usa mucho en la industria y en competiciones de Kaggle. Utilizaremos la herramienta de Jupyter Notebook para plasmar mediante código en Python todos los pasos necesarios y los cálculos que se precisen.

Para ejecutar el proceso se ha utilizado el código del notebook Modelo Xgboost (tfm).ipynb (véase sección 7 Anexos). Se han utilizado las siguientes librerías: xgboost, pandas, numpy, scikit-learn, matplotlib, dateutil, calendar, plotly, datetime, monthdelta y multiprocessing.

### 5.5.1 Organización del trabajo

#### 5.5.1.1 Carga de las librerías necesarias

```
#importamos las librerías necesarias
import xgboost as xgb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
import itertools
import multiprocessing

from dateutil import relativedelta
from monthdelta import monthdelta

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import ParameterGrid

from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.inspection import permutation_importance
```

#### 5.5.1.2 Carga de datos para realizar el experimento

```
df = pd.read_csv('C:/Users/jorge/El Corte Inglés, S.A/Otb y el master - Documentos/VentaMes2017-2019.csv')
df['Fecha Venta'] = pd.to_datetime(df['Fecha Venta'])
df=df.dropna()

#set date as index column. (required in time series)
df = df.set_index('Fecha Venta')

print('Shape of data',df.shape)
print(df.dtypes)
df.head()
```

### 5.5.1.3 Ajuste del modelo

El modelo Xgboost cuenta con una gran cantidad de parámetros que podemos utilizar para mejorar la predicción del modelo. Haremos una breve descripción de los parámetros que se han utilizado para llevar a cabo el experimento.

#### Parámetros básicos:

- **Booster:** el refuerzo que se va a usar. Puede ser gbtree, gblinear o dart. En el caso de gbtree y dart usa modelos basados en árboles mientras que gblinear usa funciones lineales.
- **Verbosity:** tipo de mensajes de salida. Los valores válidos son 0 (silencioso), 1 (advertencia), 2 (información), 3 (depuración). Xgboost ante un error inesperado intenta siempre aumentar la verbosidad del mensaje.
- **Validate\_parameters:** parámetro en función experimental. Si el valor es True Xgboost validará los parámetros de entrada para verificar si se usan o no.
- **Nthread:** número de hilos o subprocessos utilizados para ejecutar Xgboost.
- **Disable\_default\_eval\_metric:** mediante su activación (1) o no (0) se puede deshabilitar la métrica predeterminada.
- **Num\_pbuffer:** este parámetro está configurado automáticamente por XGBoost. Es el tamaño del búfer de predicción, normalmente establecido con el número de instancias del entrenamiento.
- **Num\_feature:** es la dimensión de la función que se usa para “impulsar”. Es un parámetro configurado automáticamente por XGBoost.

#### Parámetros para Tree Booster:

- **eta** (alias learning\_rate): valor con rango entre 0 y 1, predeterminado en 0.3. Es un parámetro que se encarga de evitar el sobreajuste. Después de cada impulso podemos obtener los pesos de las nuevas funciones y reducir los pesos para hacer el proceso de impulso sea más conservador.
- **Gamma** (alias min\_split\_loss): a mayor gamma más conservador será el modelo. Es necesario una reducción mínima de pérdidas para realizar una nueva partición en un nodo del árbol. Su rango va de cero a infinito, siendo cero el valor predeterminado.
- **max\_depth:** es la profundidad máxima del árbol. El valor predeterminado es 6. Cuanto más aumentemos este parámetro más complejo será el modelo y aumentaremos la probabilidad de sobreajuste. Asimismo, la memoria consumida por XGBoost se aumentará de forma considerable a la hora de entrenar profundamente un árbol.
- **min\_child\_weight:** suma mínima de peso de instancia necesaria de un hijo. El proceso dejará de particionar en el momento en el que el peso de la partición del árbol sea menor que el valor de este parámetro. Rango de cero a infinito. A mayor min\_child\_weight más conservador será el algoritmo.
- **max\_delta\_step:** paso delta máximo que se debe permitir para la estimación ponderada de cada árbol. El valor predeterminado de cero es el valor sin ninguna restricción. Cuanto más alto sea el valor del parámetro, más conservador será el modelo.
- **Subsample:** razón de la instancia de entrenamiento. Si establecemos el valor en 0,5 XGBoost recolectará de forma aleatoria la mitad de las instancias de datos para hacer crecer árboles evitando el sobreajuste.

- **Colsample\_bytree, \_bylevel, \_bynodel**: es una familia de parámetros, que funcionan de forma acumulativa, para el submuestreo de columnas. Tienen un rango [0,1] con valor predeterminado de 1. Especifican la fracción de columnas que se submuestrean.
  - **Colsample\_bytree**: es la proporción de submuestra de columnas al construir cada árbol. El submuestreo ocurre una vez por cada árbol construido.
  - **Colsample\_bylevel**: proporción de submuestras de columnas para cada nivel. Ocurre una vez por cada nuevo nivel de profundidad que se alcanza en un árbol.
  - **Colsample\_bynode**: proporción de submuestra de columnas para cada nodo. El submuestreo ocurre cada vez que se evalúa un nuevo nodo.
- **Lambda (alias reg\_lambda)**: Término de regularización L2 en ponderaciones. A mayor lambda más conservador será el modelo.
- **Alpha (alias reg\_alpha)**: Término de regularización L1 en ponderaciones. A mayor alpha más conservador será el modelo.
- **Scale\_pos\_weight**: controla el balance de ponderaciones positivas y negativas. Útil para clases desequilibradas.

#### Parámetros de la tarea de aprendizaje:

- **Objective**: Se puede seleccionar entre los siguientes tipos de objetivo de tarea de aprendizaje: reg:linear, reg:logistic, reg:gamma, reg:tweedie, count:poisson, rank:pairwise, binary:logistic o multi.
- **Base\_score**: sesgo global. Es la puntuación inicial de predicción de todas las instancias. El valor predeterminado es 0.5.
- **Seed**: semilla de número aleatorio. Cero es el valor predeterminado.

#### 5.5.1.3.1    **Modelo Venta Mensual**

Una vez que hemos visto que quieren decir algunos de los parámetros que tiene este modelo vamos a reestructurar nuestra serie de datos temporales para convertirlo en un problema de aprendizaje supervisado. Utilizaremos el instante de tiempo anterior para predecir el instante siguiente de tiempo. A esta representación se le conoce comúnmente como ventana corrediza.

Al hacer esto mediante la función shift de pandas tendremos algunos instantes de tiempo que no nos servirán para entrenar el modelo. Eliminaremos esos valores.

```

df['Sale_LastMonth'] = df['Unidades Vendidas'].shift(+1)
df['Sale_2Monthsback'] = df['Unidades Vendidas'].shift(+2)
df['Sale_3Monthsback'] = df['Unidades Vendidas'].shift(+3)
df['Sale_4Monthsback'] = df['Unidades Vendidas'].shift(+4)
df['Sale_5Monthsback'] = df['Unidades Vendidas'].shift(+5)
df['Sale_6Monthsback'] = df['Unidades Vendidas'].shift(+6)
df['Sale_7Monthsback'] = df['Unidades Vendidas'].shift(+7)
df['Sale_8Monthsback'] = df['Unidades Vendidas'].shift(+8)
df['Sale_9Monthsback'] = df['Unidades Vendidas'].shift(+9)
df['Sale_10Monthsback'] = df['Unidades Vendidas'].shift(+10)
df['Sale_11Monthsback'] = df['Unidades Vendidas'].shift(+11)
df['Sale_12Monthsback'] = df['Unidades Vendidas'].shift(+12)

df=df.dropna()
df.head(10)

```

Ahora que ya tenemos un dataframe con todos los datos juntos y organizados de la forma necesaria, vamos a crear una matriz compuesta de arrays de numpy para poder llevar a cabo los pasos necesarios para el entrenamiento y predicción del modelo.

```
#convertimos el dataframe en una matriz
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y = df['Sale_LastMonth'],df['Sale_2Monthsback'],df['Sale_3Monthsback']
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y=np.array(x1),np.array(x2),np.array(x3),np.array(x4),np.array(x5),
x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,y=x1.reshape(-1,1),x2.reshape(-1,1),x3.reshape(-1,1),x4.reshape(-1,1),
final_x=np.concatenate((x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12),axis=1)

print(final_x)
print(final_x.shape)
```

Es necesario separar los datos en variables de entrada (x) y de salida (y), asimismo, separaremos el conjunto de datos entre train y test para entrenar con una parte y realizar, una vez entrenado el modelo, la predicción con la parte test. Debido a que el conjunto de datos no es muy grande, la parte test tendrá un tamaño del 20%.

```
#División de los datos entre train y test
#usaremos train_test_split para ello.

X = df.iloc[:, 0:-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state=123)
```

Con el conjunto de datos ya preparado, creamos el estimador. Al estar compuesto por hiperparámetros no podemos conocer previamente el mejor valor para cada uno. Por lo que, dejaremos todos los valores por defecto salvo el número de árboles que empezaremos con un valor de 10 árboles.

```
#XGBRegressor para problemas de regresión
xgb_reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                           colsample_bylevel=1, colsample_bynode=1,
                           colsample_bytree=1, gamma=0, importance_type='gain',
                           learning_rate=0.1, max_delta_step=0, max_depth=3,
                           min_child_weight=1, missing=1, n_estimators=10,
                           n_jobs=1, nthread=None, objective='reg:squarederror',
                           random_state=0, reg_alpha=0, reg_lambda=1,
                           scale_pos_weight=1, seed=None, silent=None,
                           subsample=1, verbosity=1) #queremos warning
```

El siguiente paso será el de entrenar el modelo para después evaluar la capacidad predictiva empleando el conjunto test.

```
#entrenamos el modelo
xgb_reg.fit(X_train, y_train)

# Predicción del modelo
preds = xgb_reg.predict(X_test)
```

Para finalizar vamos a comprobar el acierto del modelo y usaremos mean\_squared\_error como medida de error.

```
# vemos el acierto del modelo

accuracy_xgb = float(np.sum(preds == y_test))/y_test.shape[0]
print('Accuracy de XGBoost: ', accuracy_xgb)
accuracy_lr = xgb_reg.score(X_test, y_test)
print('Accuracy de Regresion Lineal: ', accuracy_lr)
```

Accuracy de XGBoost: 0.0  
Accuracy de Regresion Lineal: -6.841282417031456

```
# Medida de error
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

RMSE: 20125.051440

El acierto del modelo es del 0.0% y la medida de error de 20.125. No hemos conseguido nada de acierto con los valores por defecto y un número de árboles igual a 10.

Para poder mejorar el modelo tenemos que empezar a conocer que valores de los parámetros son los más adecuados para nuestro conjunto de datos concreto. Para ello vamos a utilizar estrategias de validación como cross-validation e iremos viendo los parámetros más decisivos uno a uno.

#### Ajuste de los parámetros – estrategias de validación

En ese punto vamos a ver paso por paso como llegar a obtener los mejores valores en nuestros hiperparámetros para conseguir el mejor accuracy posible. Para ello vamos a empezar utilizando el algoritmo más general: Gradient Boosting. Una vez obtengamos los valores necesarios con este algoritmo seguiremos con XGBoost. Hemos creído conveniente empezar con este algoritmo más general para después pasar a implementaciones más complejas.

Recordemos primero que es Gradient Boosting:

Gradient Boosting nos permite utilizar cualquier función de coste siempre que esta sea diferenciable ya que es una generalización del algoritmo AdaBoost. Este algoritmo nos permite aplicar boosting a problemas de regresión o de clasificación múltiple entre otros, de ahí su gran éxito. Existen varias adaptaciones, pero el núcleo general de todas ellas es el mismo: entrenar modelos de forma secuencial, para que cada modelo ajuste los errores de los anteriores. Sin embargo, este hecho es el que lleva al modelo a ser susceptible de overfitting.

El mayor problema de este algoritmo es que computacionalmente necesita muchos recursos. Debido a que el algoritmo busca puntos de corte (thresholds) óptimos en cada árbol (división), tiene que iterar sobre todos los valores de los predictores y ordenarlos cada vez.

Para no encontrarnos con este problema se utiliza la estrategia de binning (característica que sí está presente en la implementación de XGBoost). Mediante límites en cada bin indicamos el número de observaciones que habrá en cada intervalo lo que agiliza el proceso. Sin embargo, podemos tener pérdidas de información, ya que la búsqueda de thresholds estarán limitados a cada bin.

En el modelo de XGBoost hemos empleado 10 árboles. Mediante validación cruzada vamos a obtener el número de árboles más óptimo. Crearemos para ello un rango de valores a evaluar.

```
#valores evaluados
estimator_range = range(1, 500, 25)
```

Mediante un bucle for entrenaremos el modelo con cada uno de esos estimadores para después extraer el error de entrenamiento y de validación cruzada.

```
for n_estimators in estimator_range:
    xgb_reg = GradientBoostingRegressor(
        n_estimators = n_estimators,
        loss = 'ls',
        max_features = 'auto',
        random_state = 123
    )
    # Error de train
    xgb_reg.fit(X_train, y_train)
    predicciones = xgb_reg.predict(X = X_train)
    rmse = mean_squared_error(
        y_true = y_train,
        y_pred = predicciones,
        squared = False
    )
    train_scores.append(rmse)

    # Error de validación cruzada
    scores = cross_val_score(
        estimator = xgb_reg,
        X = X_train,
        y = y_train,
        scoring = 'neg_root_mean_squared_error',
        cv = 5,
        n_jobs = multiprocessing.cpu_count() - 1,
    )
    # Se agregan los scores de cross_val_score() y se pasa a positivo
    cv_scores.append(-1*scores.mean())
```

Como resultado los valores por validación cruzada nos indican que se consigue un score mínimo con 26 árboles.

Valor óptimo de n\_estimators: 26

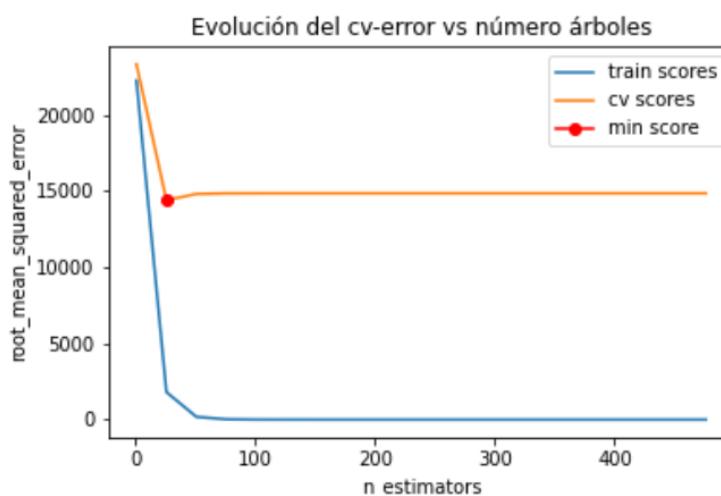


Ilustración 72 Gráfico Score XGBoost mensual

Ahora veremos que ratio de aprendizaje (learning\_rate) es el más apropiado. Este hiperparámetro es muy importante ya que con él le vamos a indicar al modelo lo rápido que debe aprender. Si no ajustamos bien el hiperparámetro corremos el riesgo de llegar al sobreentrenamiento (overfitting). El learning rate y el número de árboles son hiperparámetros interdependientes, ya que cuanto menor es el valor del learning rate más árboles se van a necesitar para conseguir buenos resultados.

Pasaremos a un bucle (véase 7. Anexos Modelo XgBoost) estos valores de learning\_rate y n\_estimators (N.º de árboles):

```
learning_rates = [0.001, 0.01, 0.1]
n_estimators   = [10, 20, 100, 200, 300, 400, 500, 1000, 2000, 5000]
```

Podemos ver el resultado para cada valor del learning\_rate en estos dos gráficos:

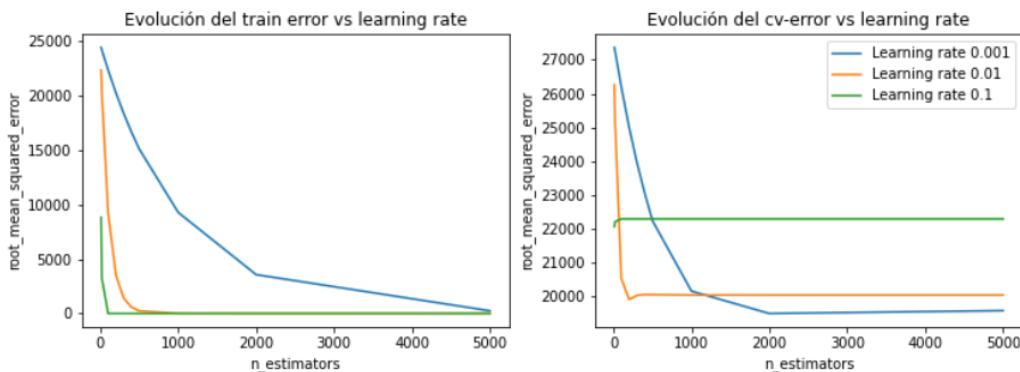


Ilustración 73 Gráficos Learning rate XGBoost mensual

Observamos que cuanto más alto es el valor del learning\_rate más rápido aprende el modelo, pero antes puede aparecer el overfitting.

Por último, veremos que profundidad es la más óptima para nuestros árboles. En estos algoritmos este hiperparámetro suele tener un valor bajo debido al coste computacional que conlleva.

Pasaremos por el bucle los valores de 1, 3, 5, 10 y 20 (véase 7. Anexos Modelo XgBoost). Asimismo, le indicaremos que tiene que tener 26 árboles dado que es valor óptimo que hemos obtenido con validación cruzada.

El resultado una vez pasados los valores anteriores es que la profundidad de nuestros árboles es de 1.

Valor óptimo de max\_depth: 1

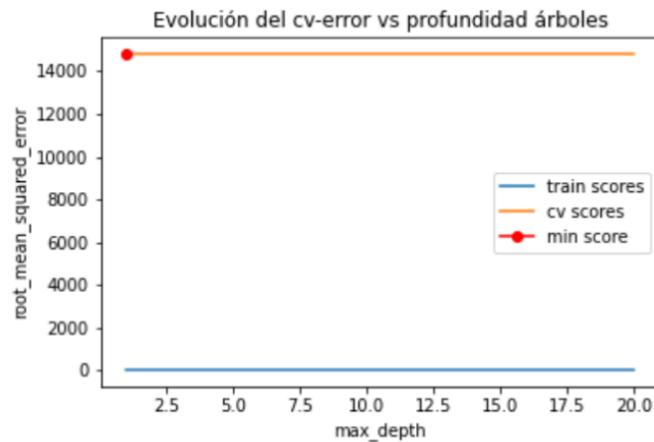


Ilustración 74 Gráfico max depth XGBoost mensual

Ahora que hemos visto de manera individual que valores deben tener estos hiperparámetros nos podemos hacer una idea de los rangos en los que deben moverse. Sin embargo, aunque esta forma es una manera rápida de ver y entender que valores deben tener los hiperparámetros dentro de nuestro modelo, debemos saber que este estudio no debe hacerse de forma secuencial, ya que estos hiperparámetros interaccionan unos entre otros. La mejor forma de obtener los valores óptimos de estos hiperparámetros es la de generar combinaciones de ellos y analizar los resultados obtenidos. Las estrategias comúnmente utilizadas para lograr este análisis son las de Grid search y Random search.

En este estudio, vamos a utilizar la estrategia de Grid search. Es decir, vamos a introducir en un grid varias combinaciones de parámetros. Asimismo, vamos a seguir una práctica muy común y en lugar de indicar distintos valores de árboles no muy elevados, vamos a indicar un valor elevado de árboles y vamos a indicarle una parada temprana (early\_stopping).

Como hemos visto antes, una de las flaquezas de los logaritmos basados en gradient boosting es que una vez alcanzado un número de weak\_learners, el modelo final acaba ajustándose perfectamente alcanzando el temido overfitting. Para no llegar a este caso, se tiene que entrenar el modelo pasando una cantidad de árboles que nos permita no llegar a tener overfitting. Esto quiere decir que tendríamos que entrenar el modelo con cientos o miles de valores hasta encontrar el valor adecuado.

Para evitar esto, existe en las distintas implementaciones estrategias que permiten ir entrenando el modelo hasta que alcance el instante en el que ya no mejora más y detener el proceso en ese punto. Para ello, lo más común es utilizar un conjunto de validación. Sin embargo, en las implantaciones nativas de scikit-learn este conjunto de validación se extrae automáticamente de los valores de entrenamiento que se van obteniendo en cada ajuste. En implementaciones como XGBoost sí que necesitaremos separar un conjunto de validación del total del conjunto de entrenamiento para poder implementar una parada temprana.

En sickit-learn esta estrategia de early\_stopping está soportada bajo los argumentos:

- *validation\_fraction*: indicamos la fracción del dataset completo de entrenamiento que debe separar para hacer la validación. Valor predeterminado del 10%.
- *n\_iter\_no\_change*: se le indica el límite en el que el modelo debe seguir entrenando hasta que no consiga mejoras en las últimas épocas de al menos un *tol*.

Dividiremos el código en tres partes: en la primera estableceremos un grid con los hiperparámetros, en la segunda parte realizaremos mediante Grid Search una validación cruzada de esos hiperparámetros; en la tercera parte veremos los resultados.

```
#Primera parte: Grid de hiperparámetros evaluados

param_grid = {'max_features' : ['auto', 'sqrt', 'log2'],
              'max_depth' : [None, 1, 3, 5, 10, 20],
              'subsample' : [0.5, 1],
              'learning_rate' : [0.001, 0.01, 0.1]
            }

# Segunda parte: Búsqueda por grid search con validación cruzada

grid = GridSearchCV(
    estimator = GradientBoostingRegressor(
        n_estimators      = 1000,
        random_state     = 123,
        # Early_stopping
        validation_fraction = 0.1,
        n_iter_no_change   = 5,
        tol                = 0.0001
    ),
    param_grid = param_grid,
    scoring    = 'neg_root_mean_squared_error',
    n_jobs     = multiprocessing.cpu_count() - 1,
    cv         = RepeatedKFold(n_splits=3, n_repeats=1, random_state=123),
    refit     = True,
    verbose    = 0,
    return_train_score = True
)

grid.fit(X = X_train, y = y_train)

# Tercera Parte: Resultados

resultados = pd.DataFrame(grid.cv_results_)
resultados.filter(regex = '(param.*|mean_t|std_t)') \
    .drop(columns = 'params') \
    .sort_values('mean_test_score', ascending = False) \
    .head(4)
```

Dentro del grid de resultados obtenidos por validación cruzada nos quedaremos con los mejores:

```
# Mejores hiperparámetros obtenidos por validación cruzada

print("-----")
print("Mejores hiperparámetros encontrados (cv)")
print("-----")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)

-----
Mejores hiperparámetros encontrados (cv)
-----
{'learning_rate': 0.1, 'max_depth': 1, 'max_features': 'auto', 'subsample': 1}
: -17479.097622892823 neg_root_mean_squared_error
```

Como hemos pasado dentro del GridSearch la parada temprana, aunque hemos usado 1000 árboles, es bastante posible que el modelo se haya detenido antes de llegar al valor del hiperparámetro. Si hubiera llegado, aumentaríamos el valor de nuevo ya que existiría posibilidad de mejora.

```
# Número de árboles del modelo final (early stopping)
print(f"Número de árboles del modelo: {grid.best_estimator_.n_estimators_}")
Número de árboles del modelo: 69
```

Como habíamos supuesto el modelo se ha detenido en los 69 árboles. A partir de ahí, el modelo no consigue mejora para un *n\_iter\_no\_change* igual a 5, y un *tol* con valor de 0.0001.

Una vez que hemos obtenido los mejores hiperparámetros para nuestro modelo en concreto entrenaremos el modelo con esos valores.

Como en la estrategia Grid Search hemos activado el argumento *refit*. (*refit=True*), este reentrenamiento se realiza automáticamente y se queda almacenado en el argumento *best\_estimator*.

Por lo que, realizamos la predicción del conjunto *X\_test* y obtenemos de nuevo el *mean\_squared\_error* como medida de error.

```
# Error de test del modelo final
xgb_reg_final = grid.best_estimator_
predicciones = xgb_reg_final.predict(X = X_test)
rmse = mean_squared_error(
    y_true = y_test,
    y_pred = predicciones,
    squared = False
)
print(f"El error (rmse) de test es: {rmse}")
```

El error (rmse) de test es: 7682.142896069201

Anteriormente, el valor del rmse con XGBoost sin modificar los parámetros por defecto era de 20.125.

```
# Medida de error XGBoost (hiperparámetros por defecto)
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

RMSE: 20125.051440

Ahora, obteniendo los valores óptimos por validación cruzada con el logaritmo de GradientBoostingRegressor hemos conseguido reducirla a 7.682

Obteniendo esta gran reducción del error vamos a realizar la misma estrategia, pero con el algoritmo XGBoost. Para este caso, sí que tendremos que separar nosotros un porcentaje del dataset para utilizarlo de conjunto de validación para la parada temprana.

En primer lugar, crearemos un grid con combinaciones de parámetros y valores. Más adelante, usando la estrategia de validación cruzada, obtendremos los mejores hiperparámetros.

```
# Grid de hiperparámetros evaluados
# =====
param_grid = {'max_depth' : [None, 1, 3, 5, 10, 20],
              'subsample' : [0.5, 1],
              'learning_rate' : [0.001, 0.01, 0.1],
              'booster' : ['gbtree']}
```

En segundo lugar, vamos a implementar una parada temprana (early stopping) que fijaremos en 5 rondas, con una métrica de evaluación de rmse (mean squared error).

```
# XGBoost necesita pasar los parámetros específicos del entrenamiento al llamar
# al método .fit()
fit_params = {"early_stopping_rounds" : 5,
               "eval_metric" : "rmse",
               "eval_set" : [(X_val, y_val)],
               "verbose" : 0}
```

Sin embargo, para poder crear una parada temprana usando xgboost es necesario crear un conjunto de validación (ver 5.5.1.4 Ajuste de los parámetros – estrategias de validación).

```
# Crear conjunto de validación
# =====
np.random.seed(123)
idx_validacion = np.random.choice(
    X_train.shape[0],
    size= int(X_train.shape[0]*0.1),
    replace=False
)

X_val = X_train.iloc[idx_validacion, :].copy()
y_val = y_train.iloc[idx_validacion].copy()

X_train_grid = X_train.reset_index(drop = True).drop(idx_validacion, axis = 0).copy()
y_train_grid = y_train.reset_index(drop = True).drop(idx_validacion, axis = 0).copy()
```

Por último, antes de entrenar el modelo, buscaremos usando GridSearchCV los mejores valores de nuestros hiperparámetros para este modelo. Una vez, lo obtengamos entrenaremos el modelo. Marcaremos refit=True para que el modelo se entrene automáticamente con los valores óptimos obtenidos. Este entrenamiento se quedará grabado en la variable best\_estimator.

```
# Búsqueda por grid search con validación cruzada
# =====
grid = GridSearchCV(
    estimator = XGBRegressor(
        n_estimators = 1000,
        random_state = 123
    ),
    param_grid = param_grid,
    scoring = 'neg_root_mean_squared_error',
    n_jobs = multiprocessing.cpu_count() - 1,
    cv = RepeatedKFold(n_splits=3, n_repeats=1, random_state=123),
    refit = True,
    verbose = 0,
    return_train_score = True
)

grid.fit(X = X_train_grid, y = y_train_grid, **fit_params)
```

	param_booster	param_learning_rate	param_max_depth	param_subsample	mean_test_score	std_test_score	mean_train_score	std_train_score
23	gbtree	0.01	20	1	-14341.249817	2534.913862	-10038.985983	5704.172605
21	gbtree	0.01	10	1	-14341.249817	2534.913862	-10038.985983	5704.172605
19	gbtree	0.01	5	1	-14341.249817	2534.913862	-10038.985983	5704.172605
13	gbtree	0.01	None	1	-14341.249817	2534.913862	-10038.985983	5704.172605

Ilustración 75 Tabla GridSearchCV XGBoost mensual

Una vez entrenado el modelo, veamos que valores óptimos de nuestros hiperparámetros hemos obtenido con validación cruzada.

```
# Mejores hiperparámetros por validación cruzada
# =====
print("-----")
print("Mejores hiperparámetros encontrados (cv)")
print("-----")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)

# Número de árboles del modelo final (early stopping)
# =====
n_arboles_incluidos = len(grid.best_estimator_.get_booster().get_dump())
print(f"Número de árboles incluidos en el modelo: {n_arboles_incluidos}")

-----
Mejores hiperparámetros encontrados (cv)
-----
{'booster': 'gbtree', 'learning_rate': 0.01, 'max_depth': None, 'subsample': 1} : -14341.249817155267
neg_root_mean_squared_error
Número de árboles incluidos en el modelo: 1000
```

La estrategia de validación cruzada nos indica que los mejores valores para los hiperparámetros son:

- Booster: gbtree.
- Learning\_rate: 0.01
- Max\_depth: None. (dentro de los valores pasados en el grid teníamos ninguno, 1, 3, 5, 10 y 20).
- Subsample: 1
- Rmse: -14.341,249
- El número de árboles incluidos es de mil. Al hacer early-stopping hemos incluido un valor alto para que el modelo deje de entrenar cuando ya no consiga mejorar los resultados.

Una vez que ya tenemos nuestro modelo entrenado vamos a predecir los valores usando el conjunto test y pasando los valores de los hiperparámetros óptimos (guardados en la variable best\_estimator).

```
# Error de test del modelo final
# =====
modelo_final = grid.best_estimator_
predicciones = modelo_final.predict(X = X_test)
rmse = mean_squared_error(
    y_true = y_test,
    y_pred = predicciones,
    squared = False
)
print(f"El error (rmse) de test es: {rmse}")
```

El error (rmse) de test es: 10192.774207841183

El error de test obtenido es de 10.192,77. Es un resultado mejor que cuando hemos entrenado el modelo con los valores de los hiperparámetros fijados por defecto (recordemos que habíamos obtenido un error de 20.125). Sin embargo, utilizando Gradient Boosting hemos obtenidos un valor de error aún más bajo: 7.682.

### 5.5.1.3.2 Modelo Venta Diaria

Una vez que hemos visto los resultados del modelo para las ventas mensuales, vamos a realizar el experimento con la venta diaria. Utilizaremos también validación cruzada para encontrar los mejores hiperparámetros de este conjunto de datos a día.

Seguiremos el mismo método de trabajo que para el conjunto de datos mensual:

1. Añadiremos las bibliotecas necesarias.
2. Importaremos el conjunto de datos
3. Utilizaremos la estrategia del valor en el paso de tiempo anterior para predecir el valor en el siguiente paso de tiempo.
4. Crearemos un conjunto de entrenamiento y otro de test
5. Entrenaremos el modelo usando los mejores hiperparámetros encontrados por validación cruzada.
6. Usaremos RMSE como medida de error para comparar resultados frente al experimento mensual.

La realización de validación cruzada nos devuelve estos resultados:

	param_booster	param_learning_rate	param_max_depth	param_subsample	mean_test_score	std_test_score	mean_train_score	std_train_score
28	gbtree	0.1	3	0.5	-1548.480665	350.123554	-967.731055	209.142599
17	gbtree	0.01	3	1	-1558.002397	363.374065	-822.261376	98.472265
29	gbtree	0.1	3	1	-1565.533114	342.700198	-773.903721	86.450087
34	gbtree	0.1	20	0.5	-1572.432911	343.885690	-812.081133	207.609419

Ilustración 76 Tabla hiperparámetros XGBoost mensual

Siendo estos los mejores valores para los hiperparámetros:

```
-----  
Mejores hiperparámetros encontrados (cv)  
-----  
{'booster': 'gbtree', 'learning_rate': 0.1, 'max_depth': 3, 'subsample': 0.5} : -1548.480665406825 neg_root_mean_squared_error  
Número de árboles incluidos en el modelo: 40
```

El valor de la medida de error (RMSE) es el siguiente:

```
# Error de test del modelo final  
# ======  
modelo_final = grid.best_estimator_  
predicciones = modelo_final.predict(X = X_test)  
rmse = mean_squared_error(  
    y_true = y_test,  
    y_pred = predicciones,  
    squared = False  
)  
print(f"El error (rmse) de test es: {rmse}")
```

El error (rmse) de test es: 1050.1059361956516

Como podemos ver en la siguiente tabla comparativa, el mejor resultado de rmse lo hemos obtenido entrenando el modelo con XGBoost para una previsión diaria con 40 árboles. Hasta ahora no habíamos conseguido un error tan bajo ni con gradient boosting ni con xgboost utilizando como conjunto de datos un conjunto de venta mensualizada.

Modelo	Con validación cruzada		Sin validación cruzada	
	Mensual	Diario	XGBOOST	Mensual
RMSE	10.193	1.050	RMSE	20.125
booster	gbtree	gbtree	booster	gbtree
nº árboles	1000	40	nº árboles	10
Learning rate	0,01	0,1	Learning rate	0,1
max_depth	None	3	max_depth	3
subsample	1	0,5	subsample	1
Gradient Boosting	Mensual	Diario		
RMSE	7.682	-	-	-
booster	auto	-	-	-
nº árboles	69	-	-	-
Learning rate	0,1	-	-	-
max_depth	1	-	-	-
subsample	1	-	-	-

Ilustración 77 Tabla comparativa experimentos con XGBoost

## 6 Conclusiones y trabajo futuro

Tal y como comentábamos en el apartado 1.3. Motivación y objetivos, Con la realización de este Trabajo de Fin de Máster teníamos el objetivo de aplicar diferentes técnicas para predecir las ventas de un departamento (informática) de El Corte Inglés. Y de esta manera, poder realizar la compra necesaria en el momento preciso, para cumplir con las expectativas de demanda que nos indica la predicción.

Para construir esta aplicación, que pueda predecir de manera automática la venta del departamento, se han seleccionado cinco métodos: Lineal, Random Forest, ARIMA, Prophet y XGBoost.

Hemos aplicado los métodos a datos de venta diarios y a datos de venta mensual, para comprobar cual se ajusta mejor a nuestras necesidades en cada caso (30 días o 10 meses).

Comparando los resultados obtenidos en los diferentes modelos obtendremos aquel que mejor se adapta a nuestro problema. Para ello, compararemos los mejores modelos de cada uno de los métodos que hemos utilizado.

Modelo	RMSE	
	Mensual	Diario
Regresión Lineal	9.769,17	1.532,24
Random Forest	13.469,91	1.132,71
ARIMA	5.829,45	1.541,52
Prophet	3.473,48	899,79
XGBoost	10.192,77	1.050,11

*Ilustración 78 Tabla comparativa mejores resultados por método*

Analizando los resultados de RMSE de cada método utilizado, llegamos a las siguientes conclusiones. El método que parece que mejor se ajusta, tanto al modelo para datos de venta diaria, como para los de venta mensual, es el método Prophet. En segundo lugar, para el modelo mensual, se situaría el modelo ARIMA, y para el modelo de venta diaria, el modelo XGBoost.

Lo que hemos podido comprobar, para todos los modelos, es que todos son mejorables mediante el ajuste de sus parámetros. Además, previamente, hemos tenido que adaptar los datos para que fueran utilizables en cada uno de estos métodos, siendo la misma información, la manera en la que los datos deben ser introducidos cambia.

Centrándonos en el que mejor resultados nos ofrece (Prophet), sabemos que permite pronosticar series temporales en base a la tendencia de la serie, efectos estacionales y variables regresoras.

Este método se comporta bastante bien, sobre todo, en la predicción de series temporales utilizando datos diarios, aunque también en mensuales. Las ventajas frente al resto de modelos son: su facilidad de aplicación, sus parámetros son fácilmente interpretables, se muestra muy sensible a los valores atípicos, su interpretación y representación gráfica es sencilla, y, además, es un método que funciona mejor con series temporales con fuertes efectos estacionales y varias temporadas de datos históricos.

Tal vez, uno de los problemas con el que nos hemos encontrado, sobre todo con el modelo de venta mensual, es el de la falta de más datos históricos de venta, para hacer una predicción de 10 meses, partíamos de solo 36 meses de datos de venta del departamento.

El método Prophet utilizado para la venta diaria, para predecir 31 días (diciembre 2019) sufre una mejora muy importante cuando ajustamos sus diversos parámetros. Esto es más notable a medida que introducimos datos adicionales, tales como, los días festivos y las promociones relevantes respecto a la venta de nuestro departamento. En el momento en el que se dota al modelo con la información para localizar esos días con picos de venta (positivos o negativos), el modelo se ajusta mucho mejor y nos proporciona unas mejores predicciones de venta futura.

Para terminar, respecto a posibles mejoras en el futuro para este modelo, sería interesante aumentar nuestra base de datos histórica de venta y así poder mejorar el entrenamiento del modelo. Otra modificación podría ser, la introducción de más información respecto al comportamiento pasado de la venta, como, por ejemplo, el descuento medio del departamento en fechas concretas, momentos con mayor porcentaje de descuento, etc. Además, sería interesante, bajar un nivel en los datos y obtenerlos a nivel producto o a familia de producto, así podríamos ser más precisos en el tipo de mercancía necesaria para cumplir con nuestra demanda.

Como conclusión final podemos decir que el objetivo de este Trabajo de Fin de Master, se ha conseguido. Hemos probado distintos métodos, los hemos ajustado para conseguir mejores resultados y, finalmente, hemos comparado cada método para saber cuál cumple mejor con nuestro objetivo propuesto.

## 7 Anexos

Se proporciona como anexo los siguientes ficheros:

1. Notebooks.zip
2. VentaMes2017-2019.csv
3. VentaDia 2017-2019.csv
4. VentaDia.csv
5. PromoDIA 2017-2019.csv
6. Promociones.csv

En ellos se encuentran las distintas bases de datos utilizadas para llevar a cabo este estudio. El código necesario para llevarlo a cabo se encuentra dentro de los notebooks que están comprimidos en el fichero Notebooks.zip. Es necesario descomprimirlo para poder acceder a los diferentes notebooks:

- Modelo LR, RF (tfm).ipynb
- Modelo ARIMA (tfm).ipynb
- Modelo Prophet(tfm).ipynb
- Modelo XgBoost (tfm).ipynb

## 8 Bibliografía

Gwilym Meirion Jenkins (1970), Time Series Análisis: Forecasting and Control.

Jason Brownlee, Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python.

John Gamboa, Deep Learning for Time-Series Analysis.

Fernandez, S. d. (s.f.). Series Temporales – Modelo Arima. Facultad de Ciencias Económicas y Empresariales. Departamento de Economía aplicada. UAM. <http://www.estadistica.net/ECONOMETRIA/SERIES-TEMPORALES/modelo-arima.pdf>.

Modelos ARIMA, <https://guiasjuridicas.wolterskluwer.es>

Modelo Prophet, <https://topbigdata.es/prevision-de-la-serie-de-tiempo-con-el-profeta-en-python>

Kourentzes, N. (s.f.). Benchmarking Facebook's Prophet. <http://kourentzes.com/forecasting/2017/07/29/benchmarking-facebook-s-prophet/>.

Facebook. (s.f.). Prophet: Automatic Forecasting Procedure (Github). <https://github.com/facebook/prophet>.

Prashant Banerjee, ARIMA Model for Time Series Forecasting, (<https://www.kaggle.com/prashant111/arima-model-for-time-series-forecasting>)

Merino y Barneo, Trabajo Fin de Máster: Análisis de Series Temporales.

Si Si Huan Zhang, S.S.-J. Output, Gradient Boosted Decision Trees for High Dimensional Sparse.

Forecasting series temporales con Python y Scikitlearn by Joaquín Amat Rodrigo, available under an Attribution 4.0 International (CC BY 4.0) at <https://www.cienciadedatos.net/py27-forecasting-series-temporales-python-scikitlearn.html>

Gradient Boosting con Python by Joaquín Amat Rodrigo, available under an Attribution 4.0 International (CC BY 4.0) at [https://www.cienciadedatos.net/documentos/py09\\_gradient\\_boosting\\_python.html](https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python.html)

Business forecasting with Facebook Prophet by Leif Arne Bakker <https://towardsdatascience.com/business-forecasting-with-facebook-prophet-b9aaf7121398>

Error de pronóstico - [https://es.xcv.wiki/wiki/Forecast\\_error](https://es.xcv.wiki/wiki/Forecast_error)