END TO END MACHINE LEARNING: DSMARKET

MEMORIA PROYECTO FINAL

ERNESTO COLÁS

EDUARDO MARTÍN

MARIA CATALINA URICOECHEA

MERITXELL PENA

MASTER EN DATA SCIENCE

BARCELONA, OCTUBRE 2022

PARTE 1: ANALISIS TENDENCIA DE VENTAS

METODOLOGÍA

En la primera parte del proyecto se propone analizar la tendencia de ventas de los productos a nivel tienda a partir de diferentes datos de ventas.

Antes de empezar con el análisis de los datos o con la modelización, es necesario transformar las bases de datos en un solo archivo. Los datos provienen de tres bases de datos distintas:

- Sales: venta de los productos por día
- Calendar: contiene el identificador de día que relaciona las fechas de sales con la fecha real.
- Prices: contiene el precio de cada producto semanalmente.

El tener la base de datos unificada permite que para cada una de las partes del trabajo se comience desde un mismo punto, con los mismos datos ya preprocesados, lo que reducirá el tiempo de trabajo futuro y asegurará la integridad de los datos. El principal problema es el tamaño de la base de datos, que dificulta su procesamiento y alarga los tiempos y el esfuerzo de computación. A través de la creación de una serie de notebooks, se consigue evitar errores de memoria que resultaban al realizar todas las operaciones en un mismo notebook.

En el primer notebook (“sales_calendar”) se busca unir las bases de datos de sales y calendar, para conseguir el número de unidades vendidas para cada identificador único (producto-departamento-tienda) en cada semana. Aunque en la base de datos de “sales” existe una variable fecha, esta no es la fecha real. Se necesita de la base “calendar” que contiene el identificador de cada una de estas fechas de “sales” con su fecha real. Para unirlas se necesita primero hacer alguna transformación a “sales”, puesto que en vez de seguir una estructura donde cada fila representara la venta por día de cada producto, tiene las variables temporales en columnas (tiene una columna por fecha). Para ello se pivotan la base de datos a través de la función “pandas.melt” con aquellas variables que no representan fechas como variables identificadoras. Esta operación resulta en una base de datos que tiene una fila por cada identificador

único y fecha. Este formato permite la unión de las bases de datos de “sales” y “calendar”. No obstante, antes se debe transformar ligeramente la base de datos de “calendar”, para obtener un índice de semana. Se busca tener los datos en semana porque los precios tienen precios semanales. Tras esta trasformación que resulta en la creación de una variable con el número de semanas ISO y el año, se unen ambas bases de datos y se consigue el número de unidades vendidas por cada identificador único y semana, que se exporta a “sales_calendar.csv”.

El segundo notebook (“date_id”), tiene como objetivo identificar a cada semana con una fecha real. Esto es ventajoso, puesto que permite tener una variable fecha que facilitará operaciones más adelante. Se realizan prácticamente las mismas operaciones que en el primer notebook, obteniéndose como resultado un archivo “date_id.csv”, que contiene el identificador único, semana-año y la fecha real de un día de esa semana. La razón de hacer dos notebooks es, como se explica previamente, los problemas de memoria. El realizar todas estas operaciones bajo un mismo notebook provoca problemas de memoria, por lo que se decide separarlos.

El tercer notebook (“mergeDB”) consiste en la unión de las bases de datos anteriores, junto con la base de datos de “prices”. Además, se añaden otras variables que se perdían en los notebooks anteriores, como la categoría o la tienda. El resultado es una base de datos que contiene toda la información que se requiere para comenzar a hacer el resto de tareas del trabajo: unidades vendidas de cada identificador único por fecha semanal con su precio de venta. Esta información, tras el preprocesamiento, será la que se utilizará en el dashboard.

Aunque se haya conseguido la base de datos principal, es necesario incluir cierta información que se pierde relacionada con festivos, fines de semana, etc. Para eso se crea el cuarto notebook “Weekend Sales”, que pretende recoger el número de unidades vendidas realizadas en fin de semana y en eventos especiales (navidad, por ejemplo).

Una vez se obtienen todos los datos que se requieren, se crea un último notebook que realizará el preprocesamiento de datos (“Final”). En este notebook, se eliminan los valores repetidos y se imputan los valores nulos y además se crea las variables

necesarias para la realización del clustering. Por lo que tiene dos salidas: una salida destinada a proveer de datos el dashboard y otra salida para el clustering.

ANALISIS TENDENCIA DE VENTAS

Para hacer el análisis de tendencia, seguimos los siguientes pasos:

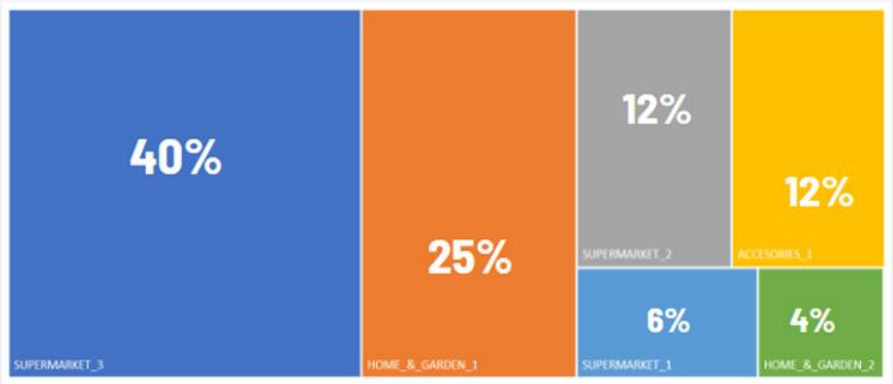
- Realizamos un Análisis de Pareto con el fin de detectar cuales son las referencias con mayor impacto a nivel global.
- Identificamos las ventas acumuladas por cada tienda
- Por cada tienda, se decidió analizar las 10 referencias con mayores ventas acumulados por ciudad por tienda con el fin de simplificar en análisis.
- Graficamos las ventas año a año de cada referencia, por ciudad y tienda.
- De los 100 productos analizados se analizaron aquellas referencias compartidas entre regiones.

CONCLUSIONES

Análisis Pareto Ventas Acumuladas

En total son 3.049 las referencias de DS Market y el 80% de las ventas acumuladas desde el 2011 las generan 1.267 referencias. En el anexo 1 este listado de las 1.267 referencias con sus respectivas ventas.

Para visualizar los datos, identificamos las categorías de las 1.267 referencias e hicimos una gráfica de rectángulos, pues este último nos permite presentar datos organizados en jerarquías de dimensiones.



Gráfica - 1 Pareto por categorías

El departamento SUPERMARKET es el que presenta mejor desempeño y es el único departamento que tiene todas sus categorías dentro de las 1.267 referencias: representa un 58% de las ventas.

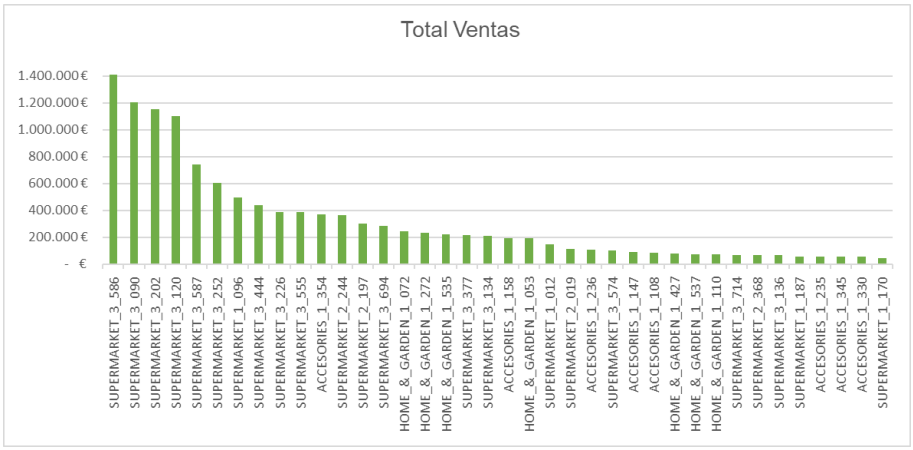
Para los productos que representan el 20% de las ventas (1.782 referencias), se identificaron:

- 12 referencias sin ventas en el año 2016.
- El promedio de ventas acumulado para las 1.782 referencias es de 20.961€ en contraste con el promedio de las 1.267 referencias que es igual a 118.009€

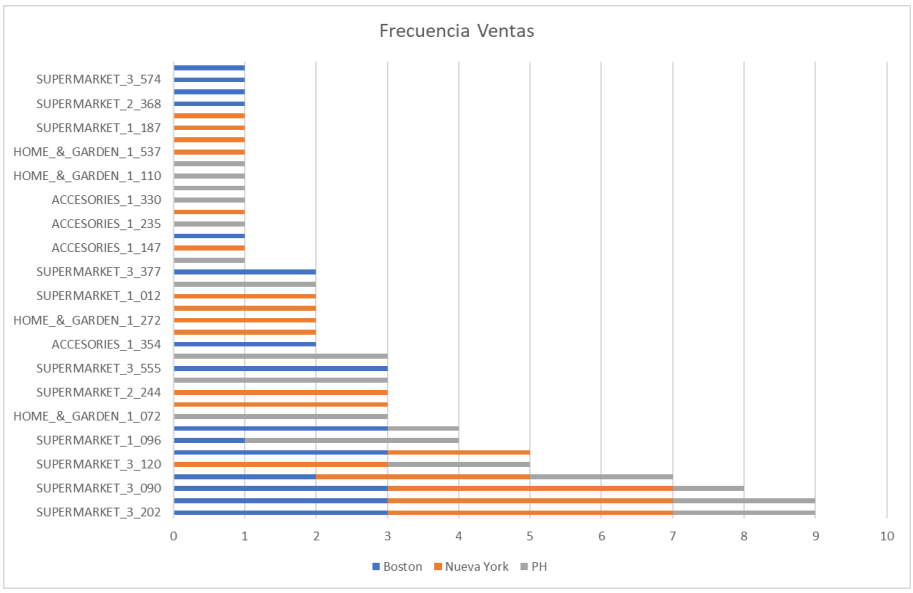
Top 10 referencias por tienda

Para cada una de las tiendas se grafican los 10 mejores referencias por tienda. Las referencias por tienda y año se graficaron utilizando un gráfico con el fin de identificar tendencias. Las gráficas se pueden ver en el anexo 2.

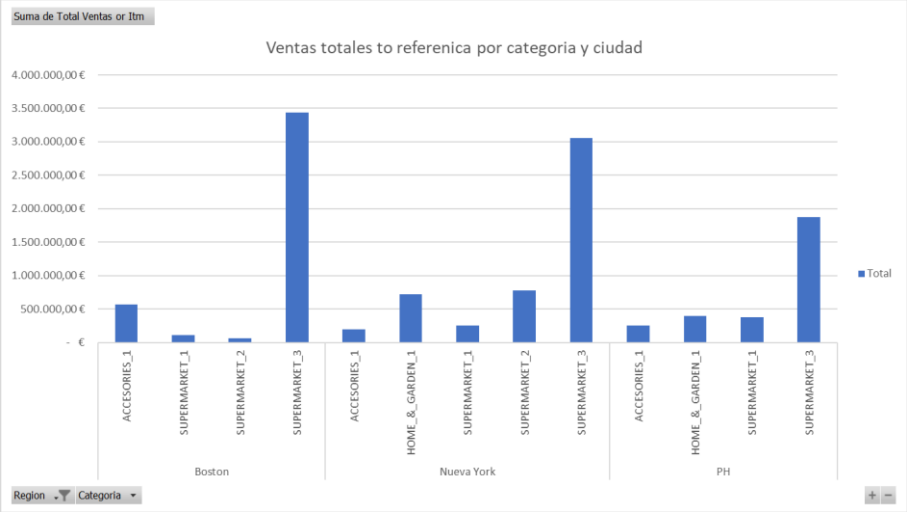
De los 100 productos analizados se identificaron 38 referencias únicas.



De estas 38, 22 referencias se repiten en las tres ciudades.



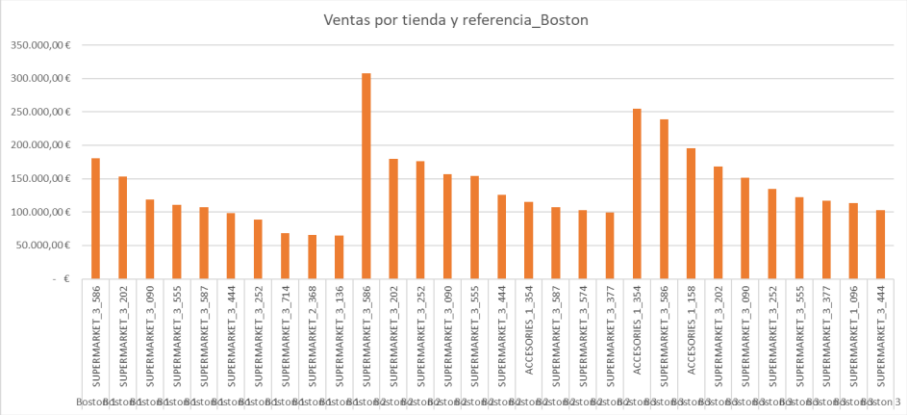
De las 3 categorías, es SUPERMARKET la que tiene más peso entre las 100 referencias analizadas.



El departamento de SUPERMARKET retiene un 82% de las ventas, seguido por HOME & GARDEN con un 9% y ACCESORIES con un 8%.

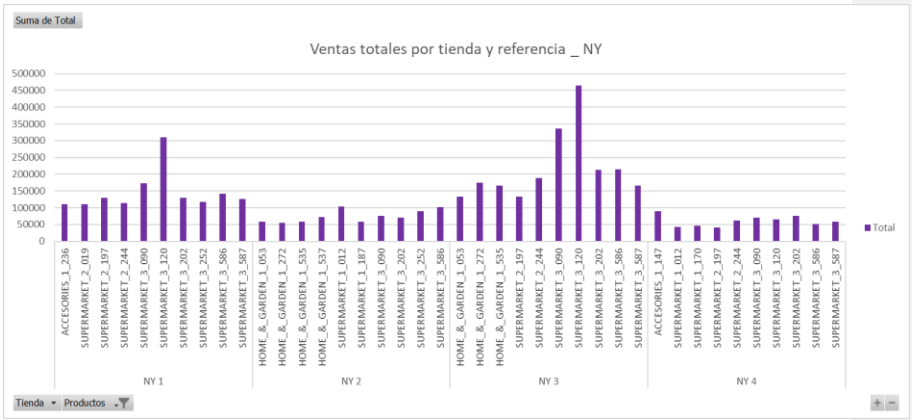
Para Boston

- El producto “SUPERMARKET_3_586” es el más popular en la región de Boston. Seguido por “SUPERMARKET_3_202”.
- Las ventas de los 10 mejores productos caen en promedio un 84% para las 3 tiendas en Boston.
- El 2012 y 2014 fueron en promedio los años con mejor performance.
- Aproximadamente el 41% de los productos para las 3 tiendan no han aumentado en ventas desde el 2011.
- En el 2015 “ ACCESORIES_1_354” presenta un comportamiento inusual en sus ventas. Aportando un 28% en Boston 2 y un 33% en Boston 3.



Para Nueva York:

- “SUPERMARKET_3_120” es el producto con mayor ventas y mejor performance.
- Tanto NY3 como NY2 comparten 4 de los 10 productos con mejores perfomance por tienda.
- El 2012 y 2015 fueron en promedio los años con mejor performance con 1.122.404€ y 1.003.728 €
- Únicamente en NY3 y NY2 productos de H&G entran dentro del top 10, representando un 14% de ventas en total para las 2 tiendas.



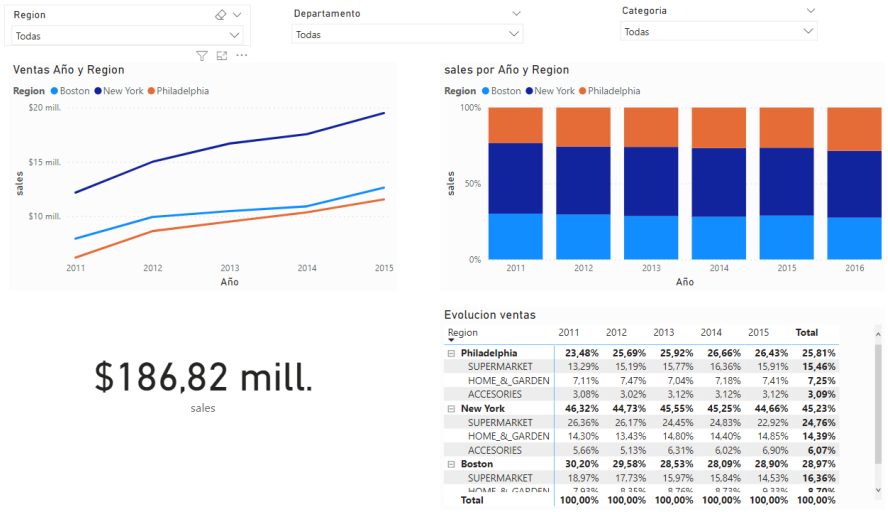
Para Philadelphia:

- Las tres tiendas en Philadelphia mantienen un comportamiento de ventas totales similar. A diferencia de Nueva York y Boston, el mix de productos populares en Philadelphia varía entre las 3 tiendas.
- “SUPERMARKET_3_226” es el producto con mayor ventas y mejor performance.
- Las 3 tiendas comparten 17 productos con mejores ventas. 7 de ellos representan el 80% de las ventas para la región.
- Los productos de la categoría “ACCESORIES” solo se venden en PH 1.

Power BI

Se diseñaron 2 dashboards para medir la evolución de las ventas y del producto.

Dashboard Evaluación Ventas



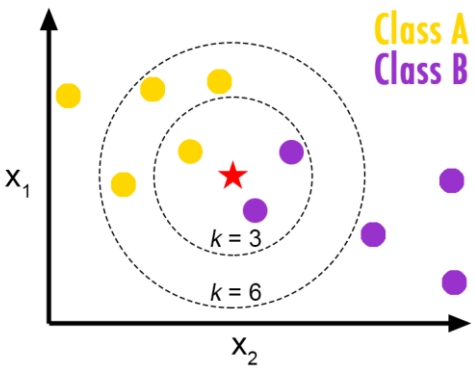
Dashboard Evaluación Producto

PARTE 2: CLUSTERING

En la segunda parte del proyecto se propone identificar grupos de productos que se comporten de manera similar y poder así agruparlos.

Para ello usaremos un aprendizaje basado en instancias o por vecindad, también denominado lazy learning, que establece un procedimiento en el que no se induce un modelo sobre el conjunto de entrenamiento y la tarea de generalización se ”pospone” hasta que se hace la consulta al sistema (predict). Para ello, en el aprendizaje por instancias el conjunto de entrenamiento es guardado en su forma original. Cuando se presenta un nuevo caso y se solicita una predicción, el conjunto de entrenamiento se carga en memoria y se busca el subconjunto de las k instancias más cercanas (en base a la métrica de distancia que corresponda), que se utilizan para clasificar la nueva instancia.

En el aprendizaje por instancias, por tanto, no se genera una hipótesis sobre el target, y sólo se hace la clasificación bajo petición en base a la clase de los k vecinos más próximos. Por este motivo, el método de aprendizaje por instancias se conoce habitualmente como k-Nearest Neighbors o kNN.



El algoritmos de k-means o k-medias es un algoritmo de agrupamiento (clustering) que busca particionar un conjunto de datos $X = \{x_1, x_2, \dots, x_n\}$ en k grupos (clústers) tal que:

- Todo el conjunto de datos X sea asignado en uno de los k clusters

- Todos los clústers tienen al menos un elemento asignado (no están vacíos)
- Cada elemento del conjunto de datos se asigna a un único clúster (no hay solapes)

El racional de k-means es agrupar los elementos por cercanía, de manera que los elementos de cada

uno de los grupos sean lo más parecidos posible entre sí (cercanos en una métrica de distancia) y lo

más distintos posible a los elementos del resto de grupos.

$$argmin \sum_{i=1}^{k} \sum_{x_j \in S}^{m} (x - \mu)^2$$

Dado que el número de combinaciones posibles es demasiado elevado (ver ley de Stirling), establecer la combinación de etiquetas tal que se minimice la distancia intra-clúster es un problema NP-completo o polinómico (en función de las premisas de dimensionalidad y número de grupos), y no es posible resolver el problema de asignación con simple exploración. En este sentido, los diferentes algoritmos de clustering buscan aproximaciones para resolver este problema de una manera eficiente.

Dentro de los algoritmos que intentan resolver el problema del agrupamiento, separamos los algoritmos *top-down* (partimos del conjunto de datos completo y vamos realizando particiones sucesivas que optimicen alguna métrica) de los *aglomerativos* (partimos de un conjunto de un clúster por instancia y los agrupamos sucesivamente por proximidad hasta quedarnos en el número k buscado). El k-means se sitúa entre estos últimos, y funciona de la manera siguiente:

1. **Inicialización**: partimos de un conjunto inicial de k centroides (con k fijada por nosotros y selección aleatoria o con alguna estrategia de optimización)
2. **Asignación**: asignamos cada observación al centroide más cercano (es decir, hacemos una partición de las observaciones con el diagrama de Voronoi generado por los centroides), de manera que cada punto queda asociado a un único centroide

3. **Actualización**: calculamos los nuevos centroides como el centroide de las observaciones de cada grupo

Se considera que el algoritmo converge cuando los centroides ya no cambian, y es fácilmente demostrable que siempre se produce la convergencia. No obstante, es posible que el resultado final varíe según sea la elección inicial de los centroides, por lo que la selección inicial de centroides es un aspecto crítico.

Por otro lado, es importante tener en cuenta que existen diversas variantes del algoritmo k-means para trabajar con tipos concretos de datos:

- K-medians: se utiliza la mediana en lugar de la media para realizar la asignación, lo que proporciona robustez al algoritmo.
- K-modes: se utiliza con variables categóricas, y se basa en ratios de similitud entre la instancia y el centroide.

Para llevar este algoritmo a un script, y una vez el dataset está sin nulos, duplicados etc, lo haremos con lo que se conoce como pipeline.

item	sales_2011	sales_2012	sales_2013	sales_2014	sales_2015	sales_2016	sales	mean_pvp	weekend_sale_total %	summer_sales_
ACCESORIES_1_001	0.00	0.00	6451.46	11684.98	14752.31	3021.34	35910.09	11.005238	0.030604	0.0
ACCESORIES_1_002	3938.88	4757.28	4575.14	4476.11	4488.00	633.60	22869.01	5.279088	0.066334	0.0
ACCESORIES_1_003	0.00	0.00	0.00	1239.94	2709.70	533.25	4482.89	3.935812	0.094582	0.0
ACCESORIES_1_004	31442.90	32909.78	42745.76	43819.34	34700.08	4905.15	190523.01	6.008105	0.059337	0.0
ACCESORIES_1_005	10011.21	8138.21	8264.97	7131.76	9948.79	1900.43	45395.37	3.810894	0.093071	0.0
...	...	...	...	...	...	...	...	...	...	
SUPERMARKET_3_823	11518.08	5626.44	3885.19	5473.80	13051.48	877.10	40432.09	3.311120	0.104076	0.0
SUPERMARKET_3_824	5950.36	6015.48	3879.82	4568.44	821.14	416.52	21651.76	3.059455	0.105950	0.1
SUPERMARKET_3_825	12015.94	4159.35	9822.90	11610.62	11448.10	2571.64	51628.55	4.864652	0.070019	0.0
SUPERMARKET_3_826	0.00	0.00	4733.96	5036.32	4667.74	999.46	15437.48	1.539898	0.199061	0.2
SUPERMARKET_3_827	0.00	0.00	0.00	3236.40	5596.80	1603.20	10436.40	1.200000	0.291480	0.2

3049 rows × 87 columns

Dado que, en el primer paso, nos devuelve un array, usamos nuestro propio *Transformer* para convertir a DataFrame. Creamos variables a nivel de cliente. Por tanto, nuestro dataframe de salida será más pequeño. Filtramos los outliers con nuestro propio *Transformer*. Estandarizamos los valores, usando *StandardScaler*. Hacemos un fit con KMeans para calcular la inertia de los grupos (la dispersión de los datos al centroide).

Usaremos la técnica del *Elbow Curve o codo*: todo esto lo hacemos en una loop porque queremos ver cuando hay un cambio brusco en la inercia y por tanto, aumentar más el número de centroides no nos sale a cuenta porque la ganancia marginal es muy pequeña. Las variables sobre las que calcularemos serán las siguientes:

```
'sales', 'mean_pvp', 'summer_sales_total %', 'weekend_sale_tot
al %', 'christmas_sales_total %', 'exhibition_time', '2016_vs_
2015', '2015_vs_2014','2014_vs_2013', 'mean_discount']
```

```
pipe = Pipeline(steps = [
    ("Imputer", KNNImputer()),
    ("ArrayToDataFrame", ArrayToDataFrame(columns, index = index)),
    ("OutlierFilter", OutlierFilter(q = 0.99, col_to_filter = ['sales', 'm
    ("StandardScaler", StandardScaler())
])
```

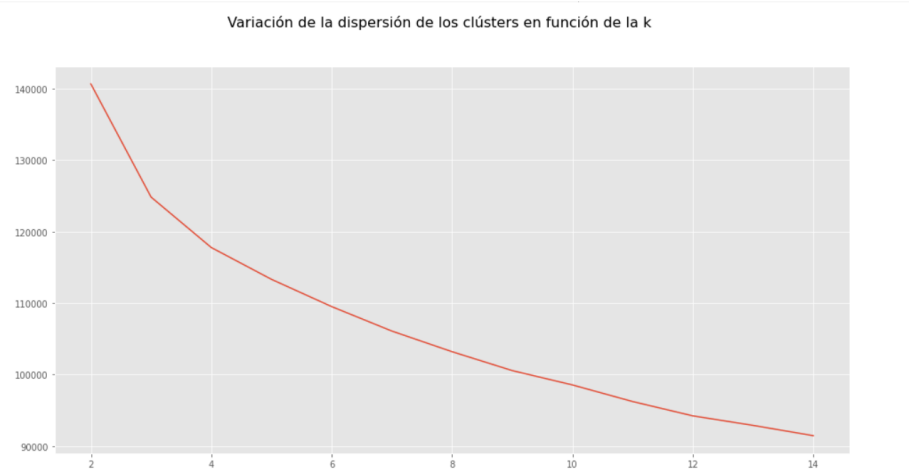
Utilizamos estos transfomers en el papeline. Es un ejemplo de lo que se llama programación orientada a objetos.

```
class ArrayToDataFrame(BaseEstimator, TransformerMixin):
    '''
    Clase que transforma un array en un DataFrame.
    Necesita como parámetros el nombre de las columnas y el índice.
    '''

    def __init__(self, columns, index = None):
        self.columns = columns
        self.index = index

    def fit(self, X, y = None):
        return self

    def transform(self, X, y = None):

        if self.index != None:
            df = pd.DataFrame(X, columns = self.columns, index = self.index)

        else:
            df = pd.DataFrame(X, columns = self.columns)

        return df
```

```python
class OutlierFilter(BaseEstimator, TransformerMixin):
    '''
    Clase que filtra los outliers utilizando np.quantile()
    Los cuantiles a filtrar así como las columnas a filtrar son los parámetros de la clase.
    '''

    def __init__(self, q, col_to_filter):
        self.q = q
        self.col_to_filter = col_to_filter

    def fit(self, X, y = None):
        return self

    def transform(self, X, y = None):
        '''
        El método considera outlier a aquel cliente que es outlier en todas las columnas que le pasas.
        Es decir: si tiene que filtrar importe y número de pedidos, sólo va a eliminar aquellos clientes
        que son outlier tanto en importe como número de pedidos. Si eres outlier en importe pero no en pedido
        no se te va a filtrar del dataset.
        '''

        # lista vacía
        criteria_list = []

        # agregamos a la lista los clientes que son outliers
        for col in self.col_to_filter:
            criteria = X[col] < np.quantile(X[col], q = self.q)
            criteria_list.append(criteria)

        # si hay más de 1 columna
        if len(self.col_to_filter) > 1:

            # creamos el criterio global: es decir outlier en todas las columnas
            global_criteria = criteria_list[0]
```

A continuación, utilizaremos el KMeans y su posterior transformación para cada k desde 2 hasta 15.

```python
if CALCULATE_ELBOW:
    st = time.time()

    sse = {}

    for k in range(2, 15):

        print(f"Fitting pipe with {k} clusters")

        clustering_model = KMeans(n_clusters = k)

        clustering_model.fit(df_scaled_transformed_no_outliers)

        sse[k] = clustering_model.inertia_

    et = time.time()
print("Elbow curve took {} seconds.".format(round((et - st)), 2))
```



El codo se hace especialmente marcado en la k igual a 4. Ahora que hemos determinado el número de centroides correcto podemos fittear nuestro pipeline con la k adecuada. Dado que vamos a realizar nuestra segmentación con KMeans y vamos a suministrarle las variables de nuestro interés, a veces, a KMeans se le conoce como segmentación no supervisada pero guiada*. Guiada porque de alguna manera el data scientist le dice (lo guía) a que discrimine usando unas variables y no otras.

```
pipe = Pipeline(steps = [
    ("Imputer", KNNImputer()),
    ("ArrayToDataFrame", ArrayToDataFrame(columns, index = index)),
    ("OutlierFilter", OutlierFilter(q = 0.99, col_to_filter = ['sales', 'mean
    ("StandardScaler", StandardScaler()),
    ("Clustering", KMeans(n_clusters = 4))
])
```

Una parte muy interesante de los pipelines es que la podemos filtrar (igual que una lista de python) y usar sólo parte de los pasos que tenemos implementados.

Esto viene muy útil porque en nuestro caso, cuando vamos a hacer el predict (asignar a cada cliente su centroide), queremos imputar los nulos, crear las variables necesarias y estandarizar (paso 1, 3 y 5 del pipe), pero **no filtrar los outliers** (todos

los productos deben tener un grupo). Si hacemos el predict con todo el pipeline, algunos clientes no se van a asignar a ningún grupo.

```
#  creamos un dataframe escalado con los pasos 1 - 2 y 4
X_processed = pipe[:2].transform(df_final)
X_scaled = pipe["StandardScaler"].transform(X_processed)
X_scaled.shape
```

```
(3049, 87)
```

```
# hacemos el predict, en este caso tendremos para cada cliente su centroide/clúster.
labels = pipe["Clustering"].predict(X_scaled)
```

```
pipe["Clustering"]
```

```
KMeans(n_clusters=4)
```

```
#le asignamos al DataFrame procesado los centroides.
#SI LO HACEMOS AL ESCALADO LOS NÚMEROS PERDERAN SU SIGNIFICADO ESCALA Y SERÁN MÁS DÍFICILES DE INTERPRETAR.
X_processed["cluster"] = labels
```

```
X_processed.shape
```

```
(3049, 88)
```

El último paso, una vez que tenemos hecha nuestra segmentación completa es crear una *ficha resumen* de cada grupo con las principales variables de negocio o con aquellas que no se han utilizado en la segmentación para hacer un seguimiento periódico de los grupos o para enviar como documento al resto de los departamentos de la empresa.

		cluster	0	1	2	3
Grupo Indicadores	Indicador	Estadístico				
General	Clúster	Tamaño	1911.000000	790.000000	230.000000	118.000000
Ventas	sales	Media	22858.485285	73986.834177	167067.175957	392492.372542
		Desviación	13404.392109	20976.140186	36843.772117	236016.209477
		Mínimo	470.890000	13297.320000	106617.580000	46127.000000
		Perc. 25	12033.665000	57224.440000	138272.065000	280495.802500
...	...	...	...	...	...	...
Descuentos	mean_discount	Mínimo	-0.652778	-0.361795	-0.249502	-0.224781
		Perc. 25	-0.019985	-0.038445	-0.036108	-0.025968
		Perc. 50	-0.000232	-0.000505	-0.000813	0.000000
		Perc. 75	0.005351	0.027037	0.018458	0.028704
		Máximo	1.539023	0.317325	0.299235	0.302433

71 rows × 4 columns

A partir de aquí se obtienen las características de cada clúster.

Comentado [1]: @ernestocolasl@gmail.com aca falta algo, las caracteristicas da cada cluster?
Assigned to Ernesto CL

- Clúster 1 → Precios muy bajos, apenas suponen facturación. Son productos que se venden en las rebajas. Probablemente de otras temporadas. Producto “ola”.

- Clúster 2→ Precios medio-bajos. Efecto bajo de campañas, apenas suponen ingresos. Producto “perro”.

- Clúster 3→ Precios muy altos. Productos orientados a personas de alto nivel adquisitivo. Producto “diamante”.

- Clúster 4 → Precio medio-alto, el más vendido. Supone 80% facturación por ítem. Crece a un ritmo altísimo. Producto “vaca”.

Nuestras recomendaciones para cada grupo son:

Clúster **Ola**→ Precios muy bajos, apenas suponen facturación. Son productos que se venden en las rebajas. Probablemente de otras temporadas.

Recomendación -> **Mantener** el número de productos y el precio pues sirven para

Clúster **Perro** → Precios medio-bajos. Efecto bajo de campañas, apenas suponen ingresos.

Recomendación -> **Eliminar** estos productos. No aportan absolutamente nada.

Clúster **Diamante** → Precios muy altos. Productos orientados a personas de alto nivel adquisitivo.

Recomendación -> Mantener el número e **incrementar precio**. Son productos de alta gama y este cliente no mira precio sino calidad.

Clúster **Vaca** → Precio medio-alto, el más vendido. Supone 80% facturación por ítem. Crece a un ritmo altísimo.

Recomendación -> **Incrementar el número** de estos productos pues aún son pocos comparado con otros clúster. Seguir con este precio para no parar el enorme crecimiento

PARTE 3: SALES FORECASTING

En la tercera parte del proyecto nos proponían crear un modelo de predicción de ventas. Hasta el momento, lo que estaban utilizando eran modelos independientes de predicción por tienda, ciudad y producto; y al final tenían un modelo agregado. Lo que haremos nosotros es a la inversa. Primero vamos a testear un modelo de predicción agregado, con toda la base de datos completa para sacar la predicción de ventas a un mes vista. A partir de este modelo, podemos filtrar por ciudad, tienda o producto para obtener la información más detallada.

La segunda parte de esta modelización va a ser generar modelos con la base de datos inicial más pequeña donde se obtenga la información directamente por ciudad, tienda o producto. Son dos maneras diferentes de obtener la misma información y comprobaremos cuál de las dos es la mejor.

Primero explicaremos el **modelo** que utilizaremos, la **metodología** para procesar la base de datos y las **variables generadas** para mejorar la predicción y finalmente los resultados.

MODELO

Para hacer esta predicción de ventas utilizaremos el modelo XGBRegressor. Los motivos principales para utilizarlo son que este lee bien la mezcla de variables categóricas y numéricas, puede soportar bases de datos con muchas observaciones y es muy eficiente, ya que tarda poco tiempo en ejecutarse.

METODOLOGÍA

Tenemos una base de datos con 5.345.528 observaciones. Haciendo un análisis exploratorio previo, observamos que muchos de los productos no tienen ventas o tienen muy pocas ventas. Por este motivo, el primer cribaje que vamos a hacer es quedarnos solo con los productos que han tenido más ventas. Estos son los datos que más nos interesan para sacar la predicción de ventas, y el riesgo de añadir productos con menos ventas es que nuestra base de datos no sea muy homogénea y tengamos un biax del output predecido hacia abajo y un RMSE más grande. Para conseguir esto, hemos ejecutado lo siguiente:

```
x = sales_df["revenue"].mean()
print(x)
print(len(sales_df))
sales_df.drop(sales_df[sales_df['revenue'] <= x].index, inplace = True)
print(len(sales_df))
```

```
34.95851388487722
5345528
1482691
```

```
x = sales_df["revenue"].mean()
print(x)
print(len(sales_df))
sales_df.drop(sales_df[sales_df['revenue'] <= x].index, inplace = True)
print(len(sales_df))
```

```
99.01576923310391
1482691
405603
```

Al final, vemos que tenemos un total de 405.603 productos con lo que trabajar.

Operaciones básicas al construir la base de datos antes de introducirla al modelo:

1. Creación de un cartesian product donde tendremos cada producto definido por una fecha. Las fechas en nuestro modelo van a ser mensuales. Lo que significa que para cada producto, vamos a tener un definido un mes a lo largo del rango de tiempo de los datos.

```
cartesian_product = pd.MultiIndex.from_product([date_range, unique_id], names = ["date", "unique_id"])
len(cartesian_product)
```

```
658556
```

We will be working with a DataFrame resampled by Months. We must resample the sales_df.

1. Una vez tengamos la base de datos agrupada por unique_id y por meses, podemos empezar a generar las nuevas variables que van a ayudarnos a mejorar el modelo.

Por ahora tenemos las siguientes variables:

```
Data columns (total 27 columns):
 #   Column           Dtype
---  ------           -----
 0   id               object
 1   item             object
 2   category         object
 3   department       object
 4   store            object
 5   store_code       object
 6   region           object
 7   date             object
 8   sell_price       float64
 9   year             int64
 10  week             int64
 11  sales            float64
 12  month            int64
 13  day              int64
 14  weekday_int      float64
 15  weekend_sales    float64
 16  event_dummy      float64
 17  event_sales      float64
 18  units            int64
 19  summer_sales     int64
 20  christmas_sales  int64
 21  sales_2011       int64
 22  sales_2012       int64
 23  sales_2013       int64
 24  sales_2014       int64
 25  sales_2015       int64
 26  sales_2016       int64
dtypes: float64(6), int64(13), object(8)
memory usage: 1.1+ GB
```

Lo que podemos observar en la foto de arriba son las variables con tiempo real. Las nuevas variables a construir van a tener temporalidad, y nos van a indicar si han afectado a lo largo de los años. Estas van a ser las siguientes:

- Ventas agrupadas por tienda y fecha. Esto nos dará la información de si la tienda influye en las ventas de nuestros productos o no.
- Ventas agrupadas por región y fecha. Esto nos dará la información de si la ciudad influye en las ventas de nuestros productos o no.
- Ventas agrupadas por categoría y fecha. Esto nos dará la información de si el tipo de producto influye en las ventas o no.
- Ventas agrupadas por tienda y fecha. Esto nos dará la información de si la tienda influye en las ventas de nuestros productos o no.
- Ventas agrupadas por mes. Esto nos dará la información de si el mes de venta influye en las ventas de nuestros productos o no.
- Ventas agrupadas por fin de semana y fecha. Esto nos dará la información de si el hecho de que sea fin de semana influye en las ventas de nuestros productos o no.
- Ventas agrupadas por temporada de verano y fecha. Esto nos dará la información de si el hecho de que sea verano influye en las ventas de nuestros productos o no.
- Ventas agrupadas por temporada de navidad y fecha. Esto nos dará la información de si el hecho de que sea navidad influye en las ventas de nuestros productos o no.
- Ventas agrupadas por item y fecha. Esto nos dará la información de si el producto influye en nuestras ventas, o es más por otros factores.

En total, vamos a tener todas estas variables:

```
Data columns (total 39 columns):
 #   Column                                    Dtype
---  ------                                    -----
 0   date                                      datetime64[ns]
 1   unique_id                                 object
 2   sales                                     float64
 3   item_price                                float64
 4   revenue                                   float64
 5   region                                    int64
 6   store_code                                int64
 7   store                                     object
 8   category                                  int64
 9   item                                      int64
 10  department                                int64
 11  year                                      int64
 12  month                                     int64
 13  weekday_int                               float64
 14  weekend_sales                             float64
 15  event_dummy                               float64
 16  summer_sales                              int64
 17  christmas_sales                           int64
 18  sales_2014                                int64
 19  sales_2015                                int64
 20  sales_2016                                int64
 21  date_store_code_sales_sum                 float64
 22  date_store_code_sales_sum_shift_1         float64
 23  date_category_sales_sum                   float64
 24  date_category_sales_sum_shift_1           float64
 25  date_item_sales_sum                       float64
 26  date_item_sales_sum_shift_1               float64
 27  date_christmas_sales_sales_sum            float64
 28  date_christmas_sales_sales_sum_shift_1    float64
 29  date_summer_sales_sales_sum               float64
 30  date_summer_sales_sales_sum_shift_1       float64
 31  date_weekend_sales_sales_sum              float64
 32  date_weekend_sales_sales_sum_shift_1      float64
 33  date_month_sales_sum                      float64
 34  date_month_sales_sum_shift_1              float64
 35  date_department_sales_sum                 float64
 36  date_department_sales_sum_shift_1         float64
 37  date_region_sales_sum                     float64
 38  date_region_sales_sum_shift_1             float64
dtypes: datetime64[ns](1), float64(24), int64(12), object(2)
memory usage: 3.9+ GB
```

EJECUCIÓN DEL MODELO

Primero separamos los datos para el entreno, la validación, y el testeo. A partir de aquí, ejecutamos nuestro XGBRegressor y vemos que el dataset test y el dataset de validación nos dan un resultado similar. El rango de tener un buen RMSE está entre 5 y 3, siempre dependiendo de tus datos, pero como aquí obtenemos números similares podemos decir que tenemos un buen modelo.

```
st = time.time()

model = XGBRegressor(seed = 175)

model_name = str(model).split("(")[0]

day = str(datetime.now()).split()[0].replace("-", "_")
hour = str(datetime.now()).split()[1].replace(":", "_").split(".")[0]
t = str(day) + "_" + str(hour)

model.fit(X_train, Y_train, eval_metric = "rmse",
    eval_set = [(X_train, Y_train), (X_valida, Y_valida)],
    verbose = True,
    early_stopping_rounds = 10)

et = time.time()

print("Training took {} minutes!".format((et - st)/60))
```

```
[0]     validation_0-rmse:19.84287      validation_1-rmse:19.18989
[1]     validation_0-rmse:14.20769      validation_1-rmse:13.65461
[2]     validation_0-rmse:10.37691      validation_1-rmse:9.95109
[3]     validation_0-rmse:7.84146       validation_1-rmse:7.59138
[4]     validation_0-rmse:6.22629       validation_1-rmse:6.17899
[5]     validation_0-rmse:5.24910       validation_1-rmse:5.39986
[6]     validation_0-rmse:4.69316       validation_1-rmse:5.01888
[7]     validation_0-rmse:4.38843       validation_1-rmse:4.84145
[8]     validation_0-rmse:4.22636       validation_1-rmse:4.77128
[9]     validation_0-rmse:4.13072       validation_1-rmse:4.75185
[10]    validation_0-rmse:4.09130       validation_1-rmse:4.75384
[11]    validation_0-rmse:4.06325       validation_1-rmse:4.75876
[12]    validation_0-rmse:4.04722       validation_1-rmse:4.76260
[13]    validation_0-rmse:4.03561       validation_1-rmse:4.77111
[14]    validation_0-rmse:4.02013       validation_1-rmse:4.76494
[15]    validation_0-rmse:4.01036       validation_1-rmse:4.75597
[16]    validation_0-rmse:3.99151       validation_1-rmse:4.77373
[17]    validation_0-rmse:3.97990       validation_1-rmse:4.77321
[18]    validation_0-rmse:3.97313       validation_1-rmse:4.76940
[19]    validation_0-rmse:3.96287       validation_1-rmse:4.76077
Training took 1.1245903611183166 minutes!
```

Finalmente, obtenemos:

RMSE_VALIDA = 4.75185

Predict and model evaluation

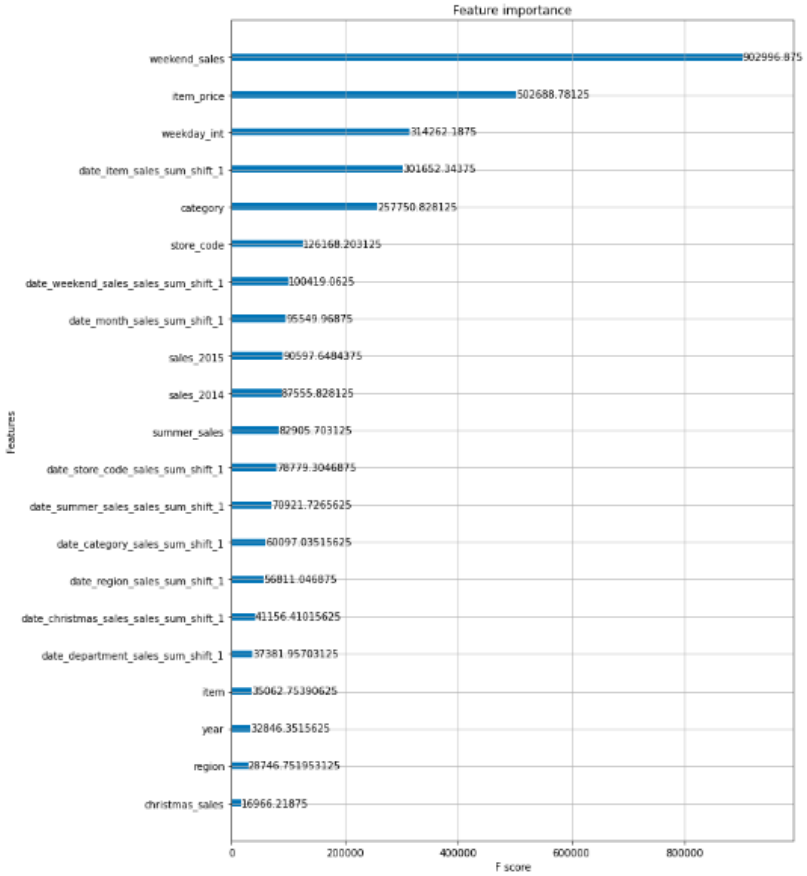
Go back to the table of contents

```
Y_valida_pred = model.predict(X_valida)

rmse_valida = sqrt(metrics.mean_squared_error(Y_valida, Y_valida_pred))
rmse_valida
```

```
4.751854843876487
```

LECTURA DEL MODELO

Si hacemos el cálculo de la importancia de cada variable que tenemos, podemos sacar este gráfico.



Las variables que más influyen en nuestras ventas son:

- Ventas el fin de semana → Los fines de semana se vende más que los días laborales
- Precio del producto → El precio del producto influye mucho en las ventas
- Tipo de producto → Los productos con más popularidad son los que van a tener más ventas el mes siguiente.
- Tipo de categoría → Lo mismo que el tipo de producto. Aquellas categorías con más popularidad en sus productos, van a tener más ventas el mes siguiente.

MODELO POR CIUDAD Y POR TIENDA

Para este tipo de modelos, lo que hemos hecho es cambiar la primera base de datos. Primero de todo, seleccionas aquella ciudad, tienda o item que quieras estudiar y eliminas todos los que no lo son.

Hemos utilizado el siguiente código:

```
In [6]:
sales_df.region.value_counts()
```

```
Out[6]:
1    2115712
0    1636338
2    1593478
Name: region, dtype: int64
```

```
In [5]:
encoder = LabelEncoder()
encoder.fit(sales_df["region"])
sales_df["region"] = encoder.transform(sales_df["region"])
sales_df["region"] = LabelEncoder().fit_transform(sales_df["region"])
```

Hacemos un Label Encoder de la variable región. La variable numero 1, que en este caso es la que tiene más observaciones, es Nueva York. Vamos a trabajar con esta variable 1.

```
In [9]:
x = 0
y = 2
```

```
In [10]:
print(x)
print(len(sales_df))
sales_df.drop(sales_df[sales_df['region'] == x].index, inplace = True)
print(len(sales_df))
```

```
0
5345528
3709190
```

```
In [11]:
print(y)
print(len(sales_df))
sales_df.drop(sales_df[sales_df['region'] == y].index, inplace = True)
print(len(sales_df))
```

```
2
3709190
2115712
```

Una vez tengamos solo variables de NY, seguiremos exactamente los mismos pasos que el modelo agrupado. La única diferencia es que solo eliminaremos una vez las variables inferiores a la media, ya que en caso contrario nos quedaríamos sin las observaciones necesarias para una buena predicción. Tiene sentido eliminar aquellos productos que tienen poco ingreso, puesto que de no eliminarlos tendríamos biax hacia negativo del resultado de la predicción. El nuevo riesgo es que no tengamos suficientes observaciones para tener un RMSE fiable.

In [14]:
```
z = sales_df.revenue.mean()
z
```

Out[14]:
```
39.98047253123299
```

In [16]:
```
print(z)
print(len(sales_df))
sales_df.drop(sales_df[sales_df['revenue'] <= z].index, inplace = True)
print(len(sales_df))
```

```
39.98047253123299
2115712
599167
```

LECTURA DEL MODELO

Obtenemos un RMSE = 0.00237. Este RMSE es muy bajo. La principal causa es la reducción del número de observaciones, ya que sin hacer el cribaje inicial en la primera base de datos y ejecutando los mismos códigos, nos da un resultado muy diferente y con poco sentido.

Visto este resultado, no vale la pena ejecutar un modelo por tienda porque la reducción de observaciones aún va a ser menor y el RMSE no va a ser válido.

PARTE 4: PREVISIÓN DE STOCK

El stock necesario en una tienda está muy relacionado con la previsión de ventas que hemos hecho en la tercera parte del trabajo. Realmente, con una buena previsión de ventas, eres capaz de predecir lo que vas a vender el mes o la semana siguiente. Si lo interpretas con un lag de más, sabiendo lo que vas a vender la semana o el mes que viene, sabes lo que tienes que pedir hoy y el stock que necesitas aquel día para abastecer toda la demanda.

No podemos utilizar el modelo mensual, ya que no es adecuado por capacidad de almacenaje y gestión logística del centro. La mejor manera es tener un sistema semanal de reposición de stock y por esto, hemos modificado el cálculo del anterior modelo de mensual a semanal. Lo que sí hemos hecho es mantener todo el código, porque las variables que influyen en la venta son las mismas que influyen en el stock.

RESULTADOS DEL MODELO AGREGADO

Podemos observar que la predicción a semanas nos da un RMSE mucho mejor igual a 3.436. Esto sería un buen apunte para tener en cuenta la previsión de ventas, ya que con exactamente el mismo código se mejora el resultado de predicción.

Predict and model evaluation

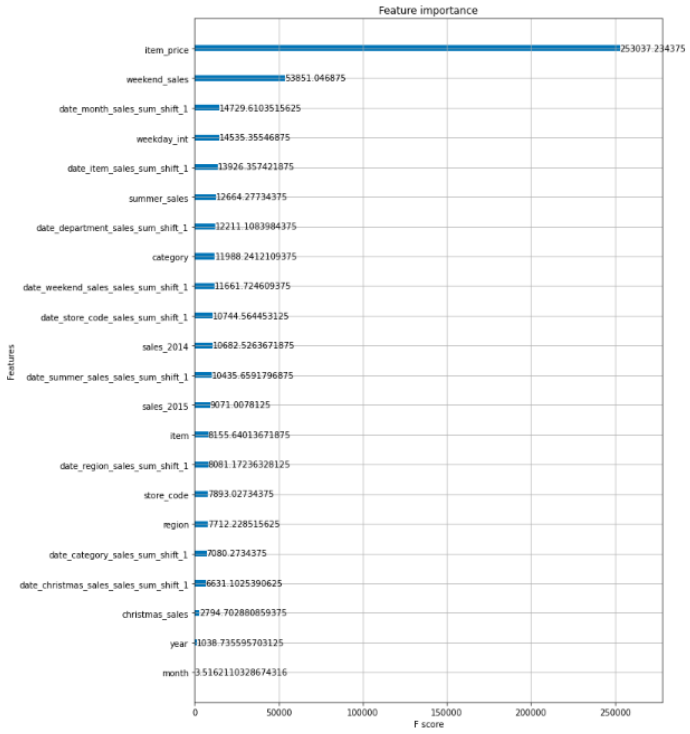
Go back to the table of contents

```
Y_valida_pred = model.predict(X_valida)

rmse_valida = sqrt(metrics.mean_squared_error(Y_valida, Y_valida_pred))
rmse_valida
```

```
3.4386483127365834
```

Las variables que han influido más aquí son las siguientes:

- Precio de los items afecta a la cantidad que necesitas en stock para la semana siguiente
- Las ventas en fin de semana influyen mucho en el modelo, así que una reposición de stock a principios de semana es la mejor opción.
- Los items y categorías más populares también afectan a las ventas semanales y por consecuencia al stock necesario.
- La temporada de verano influye más de lo que influye en el modelo semanal. Aquí en verano la necesidad de stock va a ser mayor.



Igual que en el modelo anterior, también hemos ejecutado modelos por ciudad y tienda.

MODELO POR CIUDAD, POR TIENDA

Con la misma estrategia de cribar inicialmente la base de datos, obtenemos los siguientes resultados:

- Error del modelo por ciudad (NY) → RMSE = 1.57
- Error del modelo por tienda (NY_1) → RMSE = 0.000162

Podemos sacar la misma conclusión que en el caso anterior. Si reducimos tanto el número de observaciones, el modelo va a ser menos fiable y va a tener un RMSE demasiado bajo.

5. IMPLEMENTACIÓN DEL MODELO

Una vez se ha obtenido el modelo de previsión de ventas orientado al cálculo de los envíos óptimos para la mejora del stock en tiendas, se requiere que este modelo pueda ser utilizado por otros stakeholders de la empresa y que vaya actualizándose con nuevos datos.

Los datos de venta se cargan en las bases de datos de la empresa al final de cada día. Estos datos serán los primeros con los que se trabajara, y tendrán el mismo formato que las bases de datos histórica que se han utilizado para la creación del modelo (los precios se cargaran en prices y las ventas en sales). Una vez se carguen los datos a estas bases de datos, se procede a realizar el mismo procedimiento que se ha seguido con la de históricos: unir las bases de datos y hacer quitar duplicados e imputar nulos si es necesario. Este proceso que para la base histórica se ha requerido de varios notebooks por la cantidad de datos, se unificará en un solo script. Tras esto, los datos se anexarán a una base de datos nueva donde se mantendrán hasta completar la semana. Una vez completa la semana esta información se anexará a la base de datos histórica de la que se nutre el modelo de machine learning. Estos datos, ya procesados se guardarán en un servicio de almacenamiento en la nube para que sean accesibles.

El modelo de predicción de ventas debe nutrirse de información nueva semanalmente para que poder hacer predicciones correctas. Sin embargo surgen dos problemas: la automatización de todo el proceso y la calidad de las predicciones del modelo. Es por ello que la puesta en producción del modelo se realizara por partes.

1. Primera Etapa: Automatización y Presentación de Herramienta a Negocio

Esta primera parte tiene dos objetivos principales: dar a conocer la herramienta al resto de la empresa con el objetivo de crear una cultura data driven y automatizar el proceso de reentrenamiento del modelo. En esta fase, el modelo se sigue actualizando de forma manual por parte del equipo de analítica con el objetivo de proveer estudios a las diferentes áreas de la empresa (previsiones de stock, estudios de pricing etc). Es importante esta fase, ya que permitirá ir comprendiendo como se van ajustando las predicciones del modelo conforme se va reentrenando, permitirá a la vez ir automatizando el script y sobre todo pondrá en contacto con la herramienta al resto de departamentos para que comiencen a familiarizarse con ella.

2. Segunda Etapa: Reentrenamiento semanal

Una vez finalizada la parte uno, el modelo llegará al usuario final (encargados de los envíos a tiendas, por ejemplo) para que este lo comience a utilizar. El objetivo de esta etapa es mejorar las predicciones de productos-tienda individualmente y mejorar la experiencia de los usuarios finales para que lo utilicen más. Semanalmente, el modelo se reentrenará de la siguiente manera. Primero se anexarán los datos de la semana anterior a la base de datos histórica, el modelo hará las predicciones para todos los productos y para todas las tiendas para la siguiente semana, tomando como muestra un periodo de tiempo a decidir (por ejemplo, cogerá datos de un año atrás o de dos). Estas predicciones se guardarán en un archivo (excel o csv) que se enviarán a los stakeholders. Este fichero de salida tendrá el producto, la tienda y la predicción. Todas estas predicciones se irán anexando en otra base de datos para luego comprobar si el modelo tiene buenas predicciones. En esta parte del proyecto el proceso de reentrenamiento es un proceso batch, en el que se marcará un momento en el que se realizará y no habrá cambios en las predicciones en una misma semana. Es importante durante esta fase conocer las necesidades de negocio en cuanto a la forma de enseñarles los resultados, con el objetivo de mejorar la aceptación de la herramienta (creación de dashboards).

3. Tercera Etapa: Reentrenamiento On-Demand y API

La última parte de la implementación es una mejora del modelo y liberar carga de trabajo al equipo de analítica. Para el caso de la mayoría de los productos, una predicción semanal es suficiente. Sin embargo, otros productos (como los del Supermercado), sí que pueden tener más de una entrega por semana. El cambio de unidad de tiempo de semana a día en el modelo permitiría hacer predicciones al modelo con los últimos días, pudiendo incorporar por ejemplo un jueves los datos de lunes a miércoles de esa semana y así mejorar las predicciones. Este cambio también obligará a modificar el proceso de reentrenamiento, ya que a nivel computacional sería costoso calcular diariamente predicciones para estos productos. De esta manera, será el propio stakeholder el que pedirá a través de una aplicación web que el modelo vuelva a entrenarse con los últimos datos cargados hasta el momento.

Para conseguir esto se construirá un pipeline en un proveedor de servicios en la nube (Amazon Web Services), que permitirá crear un flujo de trabajo desde el preprocesamiento de datos hasta la aplicación web desde la que los usuarios finales harán las peticiones de reentreno. A través de la herramienta de AWS Step Function, se pueden crear este tipo de flujos de forma sencilla integrando diferentes aplicaciones de AWS. Primero de todo, se cargan los datos y pasan por el modelo (creado en Sagemaker). Una vez el modelo este entrenado, desde una aplicación web desde la que los usuarios se logearan, estos podrán pedir las predicciones más recientes. Si existen nuevos datos (que irán cargándose conforme lleguen), el modelo se reentrenará incluyendo estos nuevos datos, si no mandará las predicciones ya calculadas anteriormente. La idea es que entregue predicciones para los siete días siguientes de forma separada (es decir, día uno, una cantidad, día dos, otra, etc.). La razón de utilizar este tipo de servicios es que la infraestructura viene dada completamente por el proveedor del servicio, es una herramienta visual que facilita la programación y permite un escalado fácil de la aplicación. Sin embargo, y es la razón por la que este paso será el último en la etapa de puesta en producción del modelo, requiere una inversión tanto de tiempo (para crear todo el workflow) como de dinero, puesto que estos servicios tienen un coste. Por lo que teniendo en cuenta, que la mayoría de productos que se venden tienen envíos semanales, que ya estarían cubiertos por la etapa anterior, puede que no sea necesario llegar a esta última parte.

EVALUACIÓN DEL MODELO

La evaluación del modelo se basará en dos pilares:

- Evaluación analítica: contiene todas las evaluaciones que tengan que ver con métricas.
    - Agregado: Consistirá en comprobar a nivel agregado las diferencias entre las predicciones de ventas del modelo y las ventas reales.
    - Test A/B: Primero el modelo estará capado para algunos productos (selección aleatoria) de esta forma se podrá comprobar si el stock se ha reducido entre los productos que sí que utilizan el modelo-
    - Grado de utilización: Con el objetivo de conocer si los usuarios finales están teniendo en cuenta las predicciones para este test se calcularán las diferencias entre el stock enviado real y las predicciones.
- Opinión de los usuarios: Comprobar si subjetivamente el modelo facilita su trabajo a través de encuestas de satisfacción. Se centraran en diversas temas como la interfaz que utilicen para ver los datos (el archivo de excel/dashboards o la aplicacion web), la utilidad que ven al modelo, si mejora su productividad y si mejora en general su trabajo.

Relación de Scripts y BBDD:

NÚMERO	NOMBRE	EXTENSIÓN	INPUT 1	INPUT 2	INPUT 3	INPUT 4	OUTPUT 1	OUTPUT 2	OUTPUT 3
1	Sales_calendar	ipynb	item_sales	csv daily_calendar_with_events	csv		sales_calendar_1	csv	
2	date_id	ipynb	item_sales	csv daily_calendar_with_events	csv		date_id_2	csv	
3	MergeDB	ipynb	sales_calendar_1	csv item_sales	csv date_id_2	csv item_prices csv	database_v5	csv db_powerbi_3	xlsx database_v5_floats csv
4	Weekend Sales	ipynb	daily_calendar_with_events	csv item_sales	csv item_prices	csv	database_weekends_events_4	csv	
5	Final	ipynb	database_v5_floats	csv database_weekends_events_4	csv		DF_FINAL	csv bd_clustering	xlsx
6	Dashboard	pbix	database_v5	csv					
7	Script Clustering	ipynb	bbdd_clustering_v1 *				clusters_v1_7	xlsx bd_grupos_1_7	xlsx
8	capstone-project-timeseries-sales-prediction	ipynb	DF_FINAL	csv					
9	cp-ts-sales-pred-x-region	ipynb	DF_FINAL	csv			PREDICTION_DF_9	csv FINAL_SALES_PRED_NY_DF_9	csv
10	capstone-project-agregate-stock-prediction	ipynb	DF_FINAL	csv			STOCK_PREDICTION_DF_10	csv FINAL_STOCK_PRED_DF_10	csv
11	cp-timeseries-stock-predi-tienda-ny-1	ipynb	DF_FINAL	csv			STOCK_PREDICTION_DF_11	csv FINAL_STOCK_PRED_DF_11	csv
12	cp-timeseries-stock-predi-region-ny	ipynb	DF_FINAL	csv			STOCK_PREDICTION_DF_12	csv FINAL_STOCK_PRED_DF_12	csv
			* Fichero mejorado del bd_clustering (más información)						