Evaluación de selección para el puesto: Prácticante Preprofesional de Ciencia de Datos

Postulante: Walter Jesús Felipe Tolentino - DNI 74166745 (@felipturing GitHub)

El presente documento ha sido exportado en formato PDF desde Jupyter Notebook (localhost).

Planteamiento del caso práctico:

Cierta compañía internacional de venta directa tiene un catálogo que tiene 21 días de vida, los primeros tres días se reciben alrededor del 30% (en promedio) de los pedidos totales que corresponden a los 10 grupos de venta.

¿Qué se requiere predecir?

Con los pedidos registrados en los primeros tres días por los 10 grupos de venta, se busca **pronosticar los pedidos totales al cierre de los 21 días**, es decir al cierre de cada campaña.

Pasos para encontrar la mejor solución:

Se entiende por **mejor solución** aquel modelo predictivo de Machine Learning que mejor se ajuste a los datos brindados y a las características extraidas a partir de hacer un análisis estadístico.

1. Configurar el entorno de trabajo
2. Cargar los datos (del archivo de Excel)
3. Análisis
   A. Limpieza y análisis general
   B. Ingeniería de características
4. Entrenamiento y métricas
5. Conclusiones
6. Trabajos futuros

Repositorio de la solución : *https://github.com/felipeturing/azzorti*

1. Configurar el entorno de trabajo

Configurar el entorno virtual (Python3 venv), instalar las librerías necesarias para luego importarlas en Python3, entre las más importantes y las más usadas son **Pandas**, **Numpy**, **Matplotlib** y **Scikit-learn**.

A continuación una lista de instrucciones que me va permitir configurar mi entorno virtual de trabajo en Python3 y el repositorio remoto en GitHub.

In [150…

```
!cd ~/workspace/azzorti
!git clone github https://{user}:{token}github.com/felipeturing/azzorti .
!python3 -m venv azzorti-env
!source azzorti-env/bin/activate
!pip3 install numpy openpyxl pandas matplotlib seaborn scikit-learn
!echo "Lista de módulos de Python3 usando PIP3 en el venv azzorti-env"
!pip3 list
```

```
Lista de módulos de Python3 usando PIP3 en el venv azzorti-env
Package         Version
--------------- -------
cycler          0.11.0
et-xmlfile      1.1.0
fonttools       4.34.4
joblib          1.1.0
kiwisolver      1.4.3
matplotlib      3.5.2
numpy           1.23.1
openpyxl        3.0.10
packaging       21.3
pandas          1.4.3
Pillow          9.2.0
pip             20.0.2
pkg-resources   0.0.0
pyparsing       3.0.9
python-dateutil 2.8.2
pytz            2022.1
scikit-learn    1.1.1
scipy           1.8.1
seaborn         0.11.2
setuptools      44.0.0
six             1.16.0
threadpoolctl   3.1.0
xlrd            2.0.1
```

Ahora pasamos a importar todas las librerias necesarias para el desarrollo de la solución.

In [151…

```
import warnings
warnings.filterwarnings("ignore")

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

# Scikit-learn
from sklearn.model_selection import train_test_split # Dividir entrenamiento y prueba
from sklearn.linear_model import LinearRegression # Regresión lineal
from sklearn.ensemble import RandomForestRegressor # Árboles aleatorios
from sklearn.ensemble import GradientBoostingRegressor # GBoosting
from sklearn.cluster import KMeans # Clustering
from sklearn.metrics import mean_squared_error, r2_score, average_precision_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler # Escalar

%matplotlib inline
```

2. Cargar los datos

Se usa la librería Pandas para obtener un objeto de tipo pandas.DataFrame del archivo excel de datos llamado "*Datos_201901_202009.xlsx*". Además se describe los metadatos del DataFrame y se muestra las tres primeras campañas del 2019 y 2020.

In [303…

```
%%time
datos = pd.read_excel("datos/Datos_201901_202009.xlsx", index_col=None, engine="openpyxl")
#print("Metadatos del DataFrame \n" + datos.info().__str__())
datos.info()

# Obtener los nombres de las columnas para una manipulación más suave posteriormente
grupos, campania, pedidos_totales = datos.columns[1:-1], datos.columns[0], datos.columns[-1]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CAMPAÑA          27 non-null     int64
 1   GRUPO_1          27 non-null     int64
 2   GRUPO_2          27 non-null     int64
 3   GRUPO_3          27 non-null     int64
 4   GRUPO_4          27 non-null     int64
 5   GRUPO_5          27 non-null     int64
 6   GRUPO_6          27 non-null     int64
 7   GRUPO_7          27 non-null     int64
 8   GRUPO_8          27 non-null     int64
 9   GRUPO_9          27 non-null     int64
 10  GRUPO_10         27 non-null     int64
 11  PEDIDOS_TOTALES  27 non-null     int64
dtypes: int64(12)
memory usage: 2.7 KB
CPU times: user 29.8 ms, sys: 91 µs, total: 29.9 ms
Wall time: 33.4 ms
```

Se muestra todos las variables o columnas

1. **CAMPAÑA**: Indicador de qué campaña se trata. Los primeros cuatro dígitos hacen referencia al año y los siguientes dos dígitos a la campaña. En el año se tienen 18 campañas y cada campaña dura 21 días.
2. **GRUPO_N**: Número de pedidos del grupo N durante los primeros 3 días.
3. **PEDIDOS_TOTALES**: Pedidos totales al cierre de la campaña que dura 21 días, considerada la variable dependiente a predecir.

Todos los tipos de datos que considero Pandas son **int64** y no hay valores nulos desde una primera observación en todos los 27 registros.

A continuación mostramos cierto contenido del DataFrame, las tres primeras campañas del 2019 y 2020.

In [239…

```
pd.concat([datos[ datos[campania] < 202000 ].head(3),
           datos[ datos[campania] > 202000 ].head(3)])
#pd.concat([datos[ datos['CAMPAÑA'] < 202000 ].head(3),
#           datos[ datos.CAMPAÑA > 202000 ].head(3)])
```

Out[239…

	CAMPAÑA	GRUPO_1	GRUPO_2	GRUPO_3	GRUPO_4	GRUPO_5	GRUPO_6	GRUPO_7	GRUPO_8	GRUPO_9	GRUPO_10	PEDIDOS_
0	201901	6492	6062	2961	1652	1190	89	31	410	15	43	
1	201902	4757	3618	2089	855	399	54	13	19	9	20	
2	201903	5047	3031	1018	309	157	63	12	7	8	10	
18	202001	6159	5611	3427	1348	210	151	39	15	23	11	
19	202002	4786	4207	1534	549	148	126	31	9	10	8	
20	202003	5935	4902	2390	481	133	146	31	19	8	16	

3. Análisis

En esta sección o paso de la solución se encuentra la mayor complejidad del problema, debido a que se requiere ser perspicaz y detallista con el estudio de los datos para poder encontrar relaciones y caracteríticas que permitan que los modelos de Machine Learning aplicados sean más robustos.

3.A Limpieza y análisis general

Para realizar la limpieza primero tenemos que encontrar los valores NaN de los datos, luego, o bien igualarlos a cero, colocar promedios, seguir la misma distribución de los demás valores de la variable o bien cualquier otra técnica que no agregué más incertidumbre en los análisis posteriores.

In [230…

```
datos.isna().sum()
```

Out[230…

```
CAMPAÑA            0
GRUPO_1            0
GRUPO_2            0
GRUPO_3            0
GRUPO_4            0
GRUPO_5            0
GRUPO_6            0
GRUPO_7            0
GRUPO_8            0
GRUPO_9            0
GRUPO_10           0
PEDIDOS_TOTALES    0
dtype: int64
```

Ahora como ya se sabe que todos los valores de las variables están "limpios", vamos a describir estadísticamente las variables definidas por los grupos y los pedidos_totales, sin miedo alguno.

In [244…

```
print(datos[pedidos_totales].describe())
datos[grupos].describe()
```

```
count       27.000000
mean     55574.814815
std       6201.122641
min      43091.000000
25%      50666.000000
50%      56072.000000
75%      59168.500000
max      66863.000000
Name: PEDIDOS_TOTALES, dtype: float64
```

Out[244…

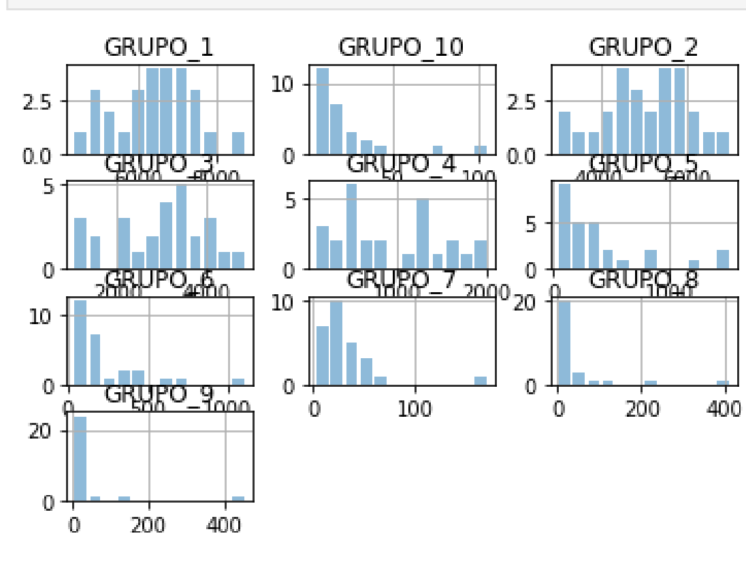
	GRUPO_1	GRUPO_2	GRUPO_3	GRUPO_4	GRUPO_5	GRUPO_6	GRUPO_7	GRUPO_8	GRUPO_9	GRUPO_10
count	27.000000	27.000000	27.000000	27.000000	27.000000	27.000000	27.000000	27.000000	27.000000	27.000000
mean	6423.407407	5044.185185	2922.037037	939.037037	416.074074	232.666667	33.814815	47.370370	35.444444	22.074074
std	1075.220666	998.678051	1069.612528	577.807360	412.075136	255.798148	31.527811	84.199008	87.200623	21.845482
min	4328.000000	3031.000000	1018.000000	100.000000	37.000000	36.000000	3.000000	1.000000	2.000000	6.000000
25%	5836.500000	4436.000000	2072.500000	533.500000	152.500000	81.000000	17.500000	10.000000	8.500000	11.000000
50%	6502.000000	5200.000000	3112.000000	759.000000	219.000000	136.000000	27.000000	19.000000	15.000000	15.000000
75%	7185.000000	5695.000000	3750.500000	1359.000000	517.500000	284.500000	39.500000	37.500000	23.000000	23.500000
max	8783.000000	6990.000000	4894.000000	2013.000000	1529.000000	1110.000000	172.000000	410.000000	458.000000	105.000000

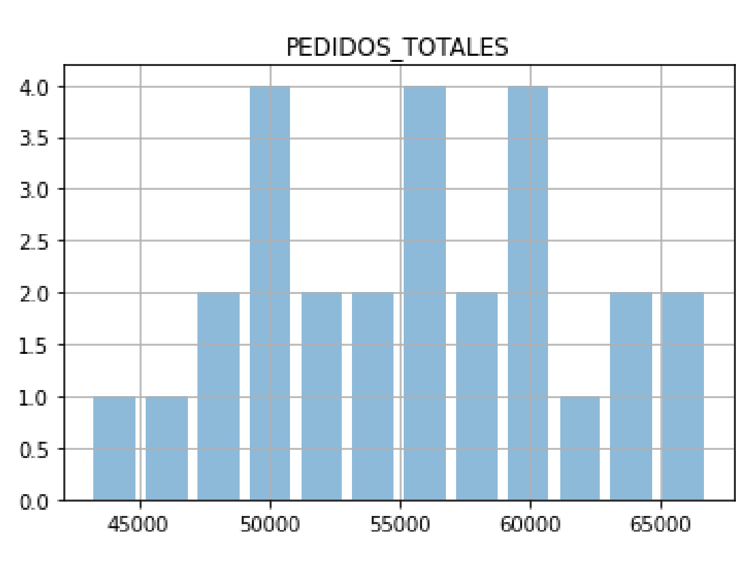
Se observa que generalmente la media va disminuyendo en los grupos desde el GRUPO_1 hasta el GRUPO_10. Además los tres primeros grupos son los que aportan más pedidos a los PEDIDOS_TOTALES, seguramente por influencia de otra variables como la región, zona, localidad, estrategias más efectivas de ventas por catálogo, Por influencia personal, entre otras características que se escapan de este estudio.

Además la mediana (el percentil 50%) se aproxima en muchos grupos a la media, esto puede dar a entender que se puede aproximar a una distribución normal, debemos comprobarlo visualmente con las gráficas de distribuciones de las variables.

In [215…

```
datos[grupos].hist(bins=12, alpha=0.5, rwidth=0.8)
datos[pedidos_totales].hist(bins=12, alpha=0.5, rwidth=0.8)
plt.show()
```



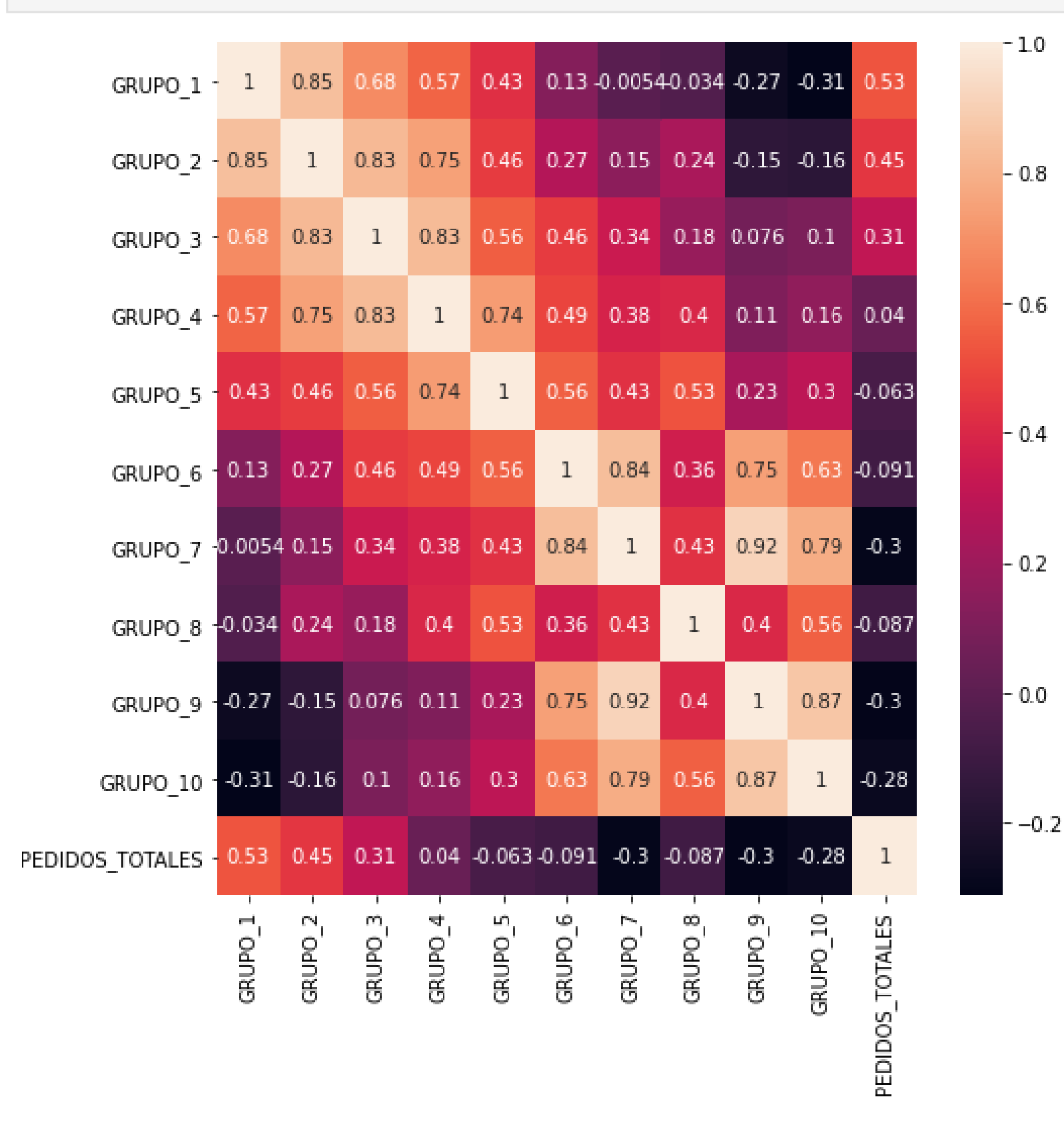


Hay algunos grupos que se aproximan a la distribución normal, se considerará para usar el StandardScaler antes del entrenamiento, por otro lado también probar con el MinMaxScaler para ver si reduce la incertidumbre.

Ahora vamos a ver si existe alguna correlación entre las variables grupos usando el muy conocido Heatmap

In [248…

```
_, ejes = plt.subplots(figsize = (8, 8))
sns.heatmap(datos[datos.columns[1:]].corr(), annot=True, ax=ejes)
plt.show()
```



Se observa que en el primer cuadrante hay una intención de correlación directa, es decir los GRUPO_1, GRUPO_2, GRUPO_3 Y GRUPO_4 están caracterizados por "detalles" similares, ya que todos siguen el fujo de correlación. Asímismo con el cuarto cuadrante entre los demás grupos, excepto los GRUPO_5 y GRUPO_8. Evidentemente también se observa que los tres primeros grupos son los que más aportan al cálculo de PEDIDOS_TOTALES.

Son detalles a considerar a los hora de definir las características (en la Ingeniería de características)

3.B Ingeniería de características

Este proceso es el más complejo, puesto que se necesita tener buena observación y creatividad, para poder definir nuevas características a partir de las variables ya definidas.

Pero antes de eso vamos a realizar una escala logarítmica debido a la naturaleza de los datos.

In [304…

```
datos[grupos] = np.log(datos[grupos]).replace(-np.inf, 0)
datos[pedidos_totales] = np.log(datos[pedidos_totales]).replace(-np.inf, 0)
pd.concat([datos[ datos[campania] < 202000 ].head(3),
           datos[ datos[campania] > 202000 ].head(3)])
```

Out[304…

	CAMPAÑA	GRUPO_1	GRUPO_2	GRUPO_3	GRUPO_4	GRUPO_5	GRUPO_6	GRUPO_7	GRUPO_8	GRUPO_9	GRUPO_10	PEDIDOS_
0	201901	8.778326	8.709795	7.993282	7.409742	7.081709	4.488636	3.433987	6.016157	2.708050	3.761200	
1	201902	8.467372	8.193677	7.644441	6.751101	5.988961	3.988984	2.564949	2.944439	2.197225	2.995732	
2	201903	8.526549	8.016648	6.925595	5.733341	5.056246	4.143135	2.484907	1.945910	2.079442	2.302585	
18	202001	8.725670	8.632484	8.139441	7.206377	5.347108	5.017280	3.663562	2.708050	3.135494	2.397895	
19	202002	8.473450	8.344505	7.335634	6.308098	4.997212	4.836282	3.433987	2.197225	2.302585	2.079442	
20	202003	8.688622	8.497399	7.779049	6.175867	4.890349	4.983607	3.433987	2.944439	2.079442	2.772589	

Las características de los grupos que vamos a considerar son aquellas que brinden intensidad a los grupos que más aportan a los PEDIDOS_TOTALES y los que están más relacionados entre sí, de acuerdo al Heatmap. Además registrar las tasas de crecimiento.

In [305…

```
%%time
suma_grupos = datos['GRUPO_1'] + datos['GRUPO_2'] + datos['GRUPO_3'] + datos['GRUPO_4'] + datos['GRUPO_7']
                                + datos['GRUPO_8'] + datos['GRUPO_9'] + datos['GRUPO_10']
suma_grupos_primer_cuadrante = datos['GRUPO_1'] + datos['GRUPO_2'] + datos['GRUPO_3'] + datos['GRUPO_4']
suma_grupos_cuarto_cuadrante = datos['GRUPO_6'] + datos['GRUPO_7'] + datos['GRUPO_9'] + datos['GRUPO_10']

datos['SUMA_GRUPOS'] = suma_grupos
datos['SUMA_GRUPOS_PRIMER_CUADRANTE'] = suma_grupos_primer_cuadrante
datos['SUMA_GRUPOS_CUARTO_CUADRANTE'] = suma_grupos_cuarto_cuadrante
datos['PROMEDIO_GRUPOS'] = suma_grupos / 10
datos['PROMEDIO_GRUPOS_PRIMER_CUADRANTE'] = suma_grupos_primer_cuadrante / 4
datos['PROMEDIO_GRUPOS_CUARTO_CUADRANTE'] = suma_grupos_cuarto_cuadrante / 4
datos['CAMBIO_PORCENTUAL_GRUPO_1']  = (datos['GRUPO_1'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_2']  = (datos['GRUPO_2'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_3']  = (datos['GRUPO_3'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_4']  = (datos['GRUPO_4'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_5']  = (datos['GRUPO_5'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_6']  = (datos['GRUPO_6'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_7']  = (datos['GRUPO_7'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_8']  = (datos['GRUPO_8'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_9']  = (datos['GRUPO_9'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
datos['CAMBIO_PORCENTUAL_GRUPO_10'] = (datos['GRUPO_10'].pct_change(periods=1)).fillna(0).replace(np.inf, 0)
```

```
CPU times: user 21.6 ms, sys: 158 µs, total: 21.7 ms
Wall time: 24.3 ms
```

Ahora se va analizar los grupos por campañas quizá se encuentra alguna relación interesante por año y por cada tres campañas (se puede analizar mejor)

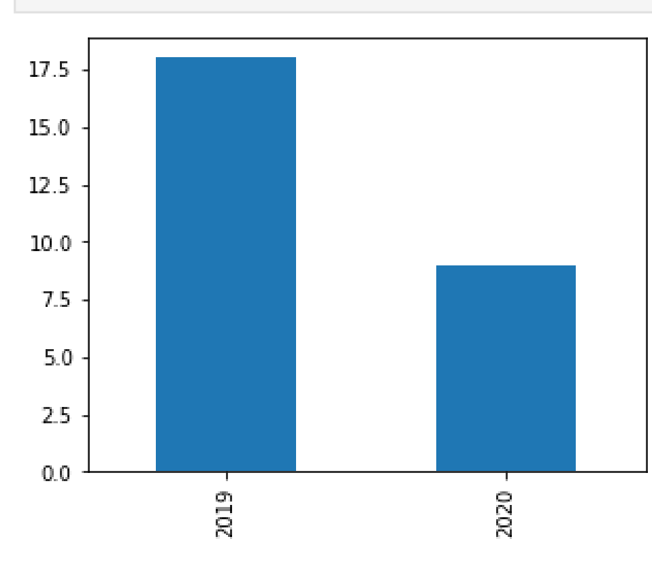
In [306…

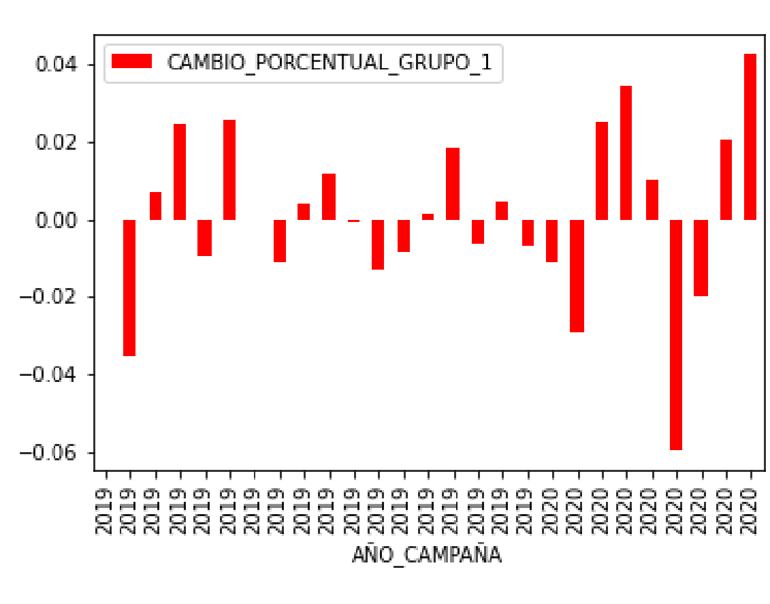
```
%%time
datos['AÑO_CAMPAÑA'] = datos['CAMPAÑA'] // 100
datos['TERCIO_AÑO_CAMPAÑA'] = (datos['CAMPAÑA'] % 100) // 3 # Falta optimizar esta característica
```

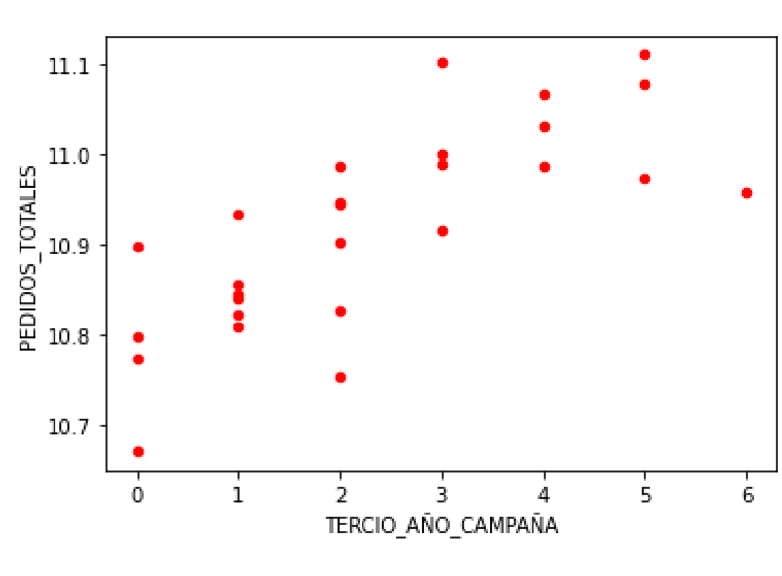
```
CPU times: user 2.01 ms, sys: 129 µs, total: 2.13 ms
Wall time: 2.06 ms
```

In [307…

```
datos["AÑO_CAMPAÑA"].value_counts().plot(kind="bar", figsize=(5, 4))
datos.plot(kind='bar',x='AÑO_CAMPAÑA', y='CAMBIO_PORCENTUAL_GRUPO_1',color='red')
datos.plot(kind='scatter',x='TERCIO_AÑO_CAMPAÑA', y='PEDIDOS_TOTALES',color='red')
plt.show()
```







Es algo predecible que finalizando las campañas halla más cambios bruscos por las ofertas y demandas apresuradas. Además se ve que de las campañas 8 hasta la 15 de cada año se tiene a compra más.

4. Entrenamiento y métricas

Se va modelar el problema usando:

```
1. Linear Regresion
2. RandomForestRegressor
3. GradientBoostingRegressor
```

Apoyandome en la librería Scikit-learn.

La variables predictoras se analizaran de acuerdo al proceso de Ingeniería de características realizado previamente.

In [316…

```
predictores = ['GRUPO_1', 'GRUPO_2', 'GRUPO_3', 'GRUPO_4', 'GRUPO_5', 'GRUPO_6', 'GRUPO_7', 'GRUPO_8',
               'GRUPO_9','GRUPO_10',
               'SUMA_GRUPOS', 'SUMA_GRUPOS_PRIMER_CUADRANTE','SUMA_GRUPOS_CUARTO_CUADRANTE',
               'PROMEDIO_GRUPOS', 'PROMEDIO_GRUPOS_PRIMER_CUADRANTE', 'PROMEDIO_GRUPOS_CUARTO_CUADRANTE',
               'TERCIO_AÑO_CAMPAÑA','AÑO_CAMPAÑA',
               'CAMBIO_PORCENTUAL_GRUPO_1', 'CAMBIO_PORCENTUAL_GRUPO_2', 'CAMBIO_PORCENTUAL_GRUPO_3',
               'CAMBIO_PORCENTUAL_GRUPO_4', 'CAMBIO_PORCENTUAL_GRUPO_5', 'CAMBIO_PORCENTUAL_GRUPO_6',
               'CAMBIO_PORCENTUAL_GRUPO_7', 'CAMBIO_PORCENTUAL_GRUPO_8', 'CAMBIO_PORCENTUAL_GRUPO_9',
               'CAMBIO_PORCENTUAL_GRUPO_10']

prediccion = 'PEDIDOS_TOTALES'

X = datos[predictores]
y = datos[prediccion]

#ESTANDARIZANDO
scaler = MinMaxScaler()
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Partición de los datos en entrenamiento y prueba, luego configurando los modelos

In [324…

```
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split( X, y,
                                                                          train_size   = 0.8,
                                                                          random_state = 1234,
                                                                          shuffle      = True)

# Configuración de los modelos
modelo_linear_regression = LinearRegression()
modelo_random_forest_regressor = RandomForestRegressor(max_depth=6, max_features='sqrt',
                                                       n_estimators=50,n_jobs=1)
modelo_gradient_boosting_regressor = GradientBoostingRegressor(n_estimators=60, learning_rate=0.5,
                                                               max_depth=2, random_state=0)

modelo_linear_regression.fit(X_entrenamiento, y_entrenamiento)
modelo_random_forest_regressor.fit(X_entrenamiento, y_entrenamiento)
modelo_gradient_boosting_regressor.fit(X_entrenamiento, y_entrenamiento)

y_predict_linear_regression = modelo_linear_regression.predict(X_prueba)
y_predict_random_forest_regressor = modelo_random_forest_regressor.predict(X_prueba)
y_predict_gradient_boosting_regressor = modelo_gradient_boosting_regressor.predict(X_prueba)
y_predict_linear_regression
```

Out[324…

```
array([10.79527338, 11.01334456, 10.8925331 , 10.98030712, 10.86406123,
       10.87211873])
```

Estableciendo una función que las métricas más importantes para los modelos predictivos

In [325…

```
def metricas(y_test, y_pred):
    r2score = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print("MSE", mse)
    print("RMSE", rmse)
    print("RMSE %", rmse * 100)
```

In [331…

```
print("LinearRegression \n")
metricas(y_prueba, y_predict_linear_regression)
```

```
LinearRegression

MSE 0.008740111198226876
RMSE 0.09348856605098166
RMSE % 9.348856605098167
```

In [330…

```
print("Random Forest Regressor \n")
metricas(y_prueba, y_predict_random_forest_regressor)
```

```
Random Forest Regressor

MSE 0.010731945708409804
RMSE 0.10359510465465926
RMSE % 10.359510465465927
```

In [329…

```
print("Gradient Boosting Regressor \n")
metricas(y_prueba, y_predict_gradient_boosting_regressor)
```

```
Gradient Boosting Regressor

MSE 0.008325705193014963
RMSE 0.09124530230655692
RMSE % 9.124530230655692
```

5. Conclusiones

1. Debido al escalado StandardScaler, la regresión lineal se comporta mejor, teniendo buenas métricas, un error porcentual el 9.3 %.
2. El modelo que mejor se ajusta es el GBoosting (Ensemble), básicamente son árboles de decisión secuenciales, llegando a un RMSE del 0.0912.

6. Trabajos Futuros

Me gustaría tener más carga de datos, para poder generar nuevas características y entender el contexto de la empresa, muchas gracias por la oportunidad.