TFM

Predicción de llamadas en Contact Center

Memoria

Antonio Ales
9-7-2021

Contenido

Introducción 2

Estado del arte 2

Objetivo 2

Estudio de los datos 3

Datos de Envíos 3

Datos de Llamadas 4

Metodología 6

Diferenciación entre modelos 8

Resultados 9

Conclusiones 11

Limitaciones 11

Manual Frontend 12

Introducción

Las empresas de mensajería tienen un gran volumen de llamadas telefónicas, se podría decir que el 90% de los envíos que se realizan por particulares son a través de una llamada telefónica.

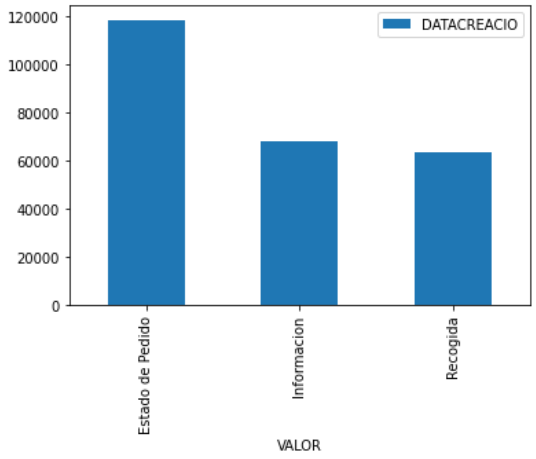
La empresa ANGEL 24 SL, dispone de un departamento de Contact Center, donde se presta servicio a empresas de toda índole.

Uno de los clientes del departamento de Contact center es un grupo de agencias de mensajería de una conocida marca de transporte urgente que opera a nivel nacional.

En esta campaña de atención telefónica, se reciben llamadas de varios tipos:

- Solicitar un envío
- Solicitar información sobre agencias
- Solicitar información sobre el estado de un envío

El porcentaje de llamadas que hacen referencia al tercer punto, *'Solicitar información sobre el estado de un envío'*, es bastante alto, y tiene relación con la cantidad de envíos que llegan a las oficinas, para posteriormente ser entregados.



Estado del arte

Actualmente hay un equipo fijo de teleoperadores calculado en base a la media de llamadas diarias, no se utilizan métodos predictivos para el dimensionamiento del equipo. El redimensionado del equipo es puramente reactivo.

Objetivo

El objetivo de este proyecto es poder configurar a los agentes telefónicos de manera proactiva en base a la predicción de la cantidad de llamadas. De esta manera se presta mejor servicio y el departamento es más rentable.

Estudio de los datos

Datos de Envíos

Los envíos están disponibles en un fichero “data/shippings.csv” con esta estructura:

```
[5]: df = df_shippings
     df.columns
```

```
[5]: Index(['Unnamed: 0', 'Fecha envío', 'Número envío', 'Id. Fiscal',
            'Nombre Comercial', 'Código servicio', 'Nombre Rem', 'Población Rem',
            'CP Rem', 'Nombre vía Rem', 'Nombre', 'Población', 'Código postal',
            'Nombre vía', 'Total bultos', 'Franquicia origen', 'Franquicia destino',
            'Total', 'Estado', 'Tipo anomalía', 'Motivo', 'Importe Total'],
           dtype='object')
```

Cada linea del fichero hace referencia a un envio, desechamos las columnas que no nos interesan y nos quedamos con:

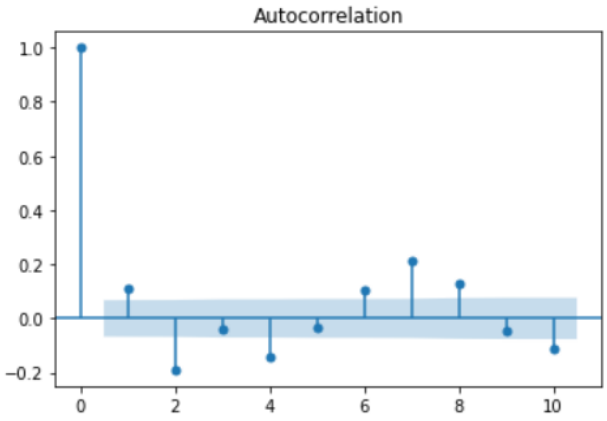
	Fecha envío	Código servicio	Franquicia destino
0	01/01/2019	Económico	4800
1	01/01/2019	Marítimo baleares	4800
2	01/01/2019	E-commerce	4800
3	01/01/2019	E-commerce	4800
4	01/01/2019	E-commerce	4800

Trabajamos el campo día de la semana de la fecha con OneHotEncoding y hacemos una agrupación por día para tener los totales diarios, quedando el dataframe de la siguiente manera:

Fecha envío	total	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
2019-01-01	33	0	0	0	0	0	1	0
2019-01-02	1923	0	0	0	0	0	0	1
2019-01-03	1408	0	0	0	0	1	0	0
2019-01-04	3220	1	0	0	0	0	0	0
2019-01-05	23	0	0	1	0	0	0	0

Análisis datos de envíos

Si analizamos el campo total, tenemos una autocorrelación con un lag de 7, en nuestro caso, días.

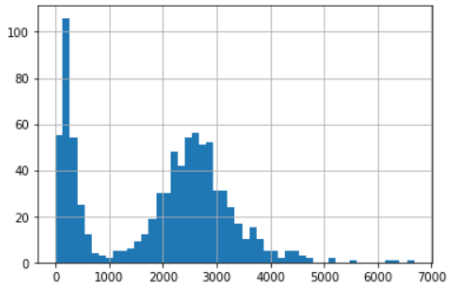


Negocio nos confirma este resultado y nos comenta que los lunes son los días más fuertes, entonces queda confirmada esta periodicidad de 7 días.

El total de envíos se presenta con la siguiente distribución de datos:

```
train['total'].hist(bins=50)
```

```
<AxesSubplot:>
```




```
train['total'].describe()
```

```
count     851.000000
mean     1946.068155
std      1276.674395
min         1.000000
25%       389.500000
50%      2273.000000
75%      2829.000000
max      6675.000000
Name: total, dtype: float64
```

Tenemos una distribución bimodal, que tras revisar los datos y hablar con negocio se debe al efecto pandemia, durante esta, y debido al confinamiento la gente compró mucho eCommerce y el número de envíos se multiplicó.

Datos de Llamadas

Los datos de las llamadas que recibe el Contact Center los tenemos en un fichero “data/calls.csv” con esta estructura:

```
df_calls = pd.read_csv('data/new_calls.csv', low_memory=False)
```

```
df_calls.columns
```

```
Index(['Unnamed: 0', 'IDCAMPANYA', 'IDSUJETO', 'VALOR', 'DATACREACIO'], dtype='object')
```

Cada linea del fichero hace referencia a una llamada, desechamos la columna “IDSUJETO” que no nos interesa y nos quedamos con:

	IDCAMPANYA	VALOR	DATACREACIO
0	100000021	No tiene Numero	2019-01-02 08:06:49
1	100000022	Recogida	2019-01-02 08:08:18
2	100000015	Estado de Pedido	2019-01-02 08:09:06
3	100000015	Recogida	2019-01-02 08:13:47
4	100000015	No tiene Numero	2019-01-02 08:15:15

Trabajamos el dataframe, y haciendo agrupaciones por “DATACREACIO” e “IDCAMPANYA” tenemos:

			dateCreacion
Date	**IDCAMPANYA**	**VALOR**	
2019-01-02	**100000015**	**Estado de Pedido**	69
		Informacion	38
		No tiene Numero	17
		Recogida	43
	100000021	**Estado de Pedido**	83

Hablamos con negocio y nos dicen que el “VALOR”:

- Estado de pedido
- No tiene Numero

Se pueden tratar del mismo modo, al final es una llamada para saber el estado de pedido. Agrupamos los datos, generamos los totales y nos queda este dataframe de llamadas

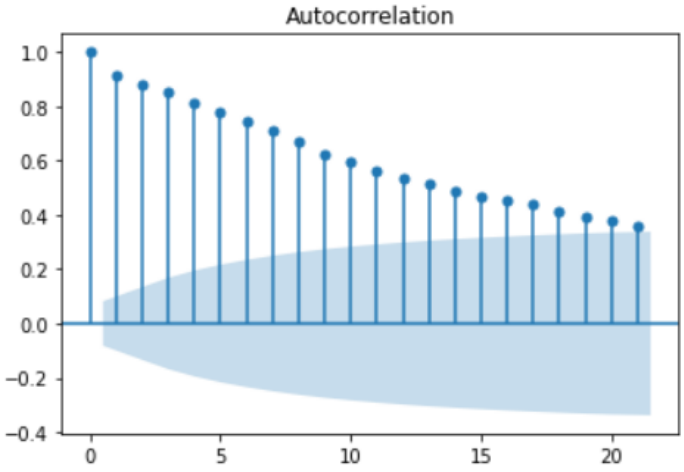
```
df_calls = df_calls[(df_calls['VALOR'] == 'Estado de Pedido') | (df_calls['VALOR'] == 'No tiene Numero')]
```

```
df_calls.head()
```

	dateCreacion
Date	
2019-01-02	281
2019-01-03	283
2019-01-04	351
2019-01-07	173
2019-01-08	174

Análisis datos de llamadas

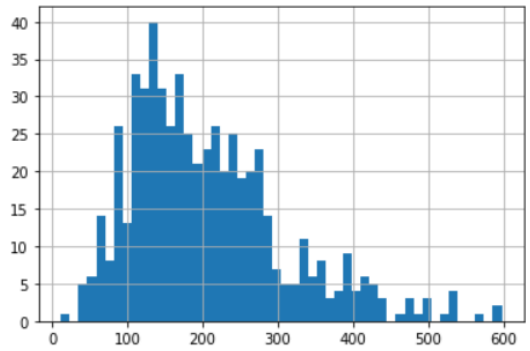
Si analizamos el campo dateCreacion, vemos una autocorrelación lineal



Con la siguiente distribución de llamadas:

```
df_calls['dateCreacion'].hist(bins=50)
```

```
<AxesSubplot:>
```




```
df_calls['dateCreacion'].describe()
```

```
count    575.000000
mean     205.789565
std      103.396648
min       11.000000
25%      130.000000
50%      185.000000
75%      259.000000
max      598.000000
Name: dateCreacion, dtype: float64
```

Metodología

A partir de los dataframes vistos, lo que se pretende es hacer un *"join"* para únicamente trabajar con uno, donde se tengan llamadas y envíos por fecha.

Tras unir los dataframes de envíos y llamadas:

	Fecha	total	Monday	Tuesday	Wednesday	Thursday	Friday	dateCreacion
1	2019-01-02	1923.0	0.0	0.0	1.0	0.0	0.0	281.0
2	2019-01-03	1408.0	0.0	0.0	0.0	1.0	0.0	283.0
3	2019-01-04	3220.0	0.0	0.0	0.0	0.0	1.0	351.0
6	2019-01-07	4595.0	1.0	0.0	0.0	0.0	0.0	173.0
7	2019-01-08	2924.0	0.0	1.0	0.0	0.0	0.0	174.0

obtenemos un dataframe con un total de 546 registros (hay festivos que, si hay reparto, pero no servicio de Contact Center de ahí la reducción de registros tras hacer el *"join"*), comprendidos entre el 14-01-2019 y el 21-04-2021

```
df.head(1)['index'],df.tail(1)['index']
```

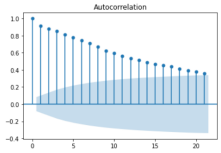
```
(0    2019-01-14
 Name: index, dtype: object,
 545    2021-04-21
 Name: index, dtype: object)
```

De los datos cabe destacar, que las llamadas son de envíos, en gran porcentaje, de días anteriores, aunque también hay envíos inmediatos que se recogen y entregan el mismo día. Por esto y porque hemos visto la relación temporal de los datos, vamos a tratar este proyecto como un problema de Series Temporales.

Data Engineering

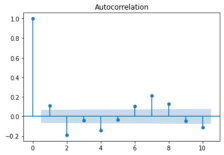
Se han generado parámetros basados en diferentes lags de los envíos y de las llamadas.

En el caso de las llamadas, hemos creado los últimos 7 días ya que su correlación era:




```
df['llamadaslag1'] = df['dateCreacion'].shift(periods=1)
df['llamadaslag2'] = df['dateCreacion'].shift(periods=2)
df['llamadaslag3'] = df['dateCreacion'].shift(periods=3)
df['llamadaslag4'] = df['dateCreacion'].shift(periods=4)
df['llamadaslag5'] = df['dateCreacion'].shift(periods=5)
df['llamadaslag6'] = df['dateCreacion'].shift(periods=6)
df['llamadaslag7'] = df['dateCreacion'].shift(periods=7)
```

En los envíos, teníamos otra correlación entre los datos, donde crearemos los siguientes lags:




```
df['envioslag1'] = df['total'].shift(periods=1)
df['envioslag6'] = df['total'].shift(periods=6)
df['envioslag7'] = df['total'].shift(periods=7)
df['envioslag8'] = df['total'].shift(periods=8)
```

Remarcar que nuestro periodo es de un día. El dataframe resultante es el siguiente:

envioslag1	envioslag6	envioslag7	envioslag8	envios	llamadaslag1	llamadaslag2	llamadaslag3	llamadaslag4	llamadaslag5	llamadaslag6	llamadaslag7	llamadas
NaN	NaN	NaN	NaN	1923.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	281.0
1923.0	NaN	NaN	NaN	1408.0	281.0	NaN	NaN	NaN	NaN	NaN	NaN	283.0
1408.0	NaN	NaN	NaN	3220.0	283.0	281.0	NaN	NaN	NaN	NaN	NaN	351.0
3220.0	NaN	NaN	NaN	4595.0	351.0	283.0	281.0	NaN	NaN	NaN	NaN	173.0
4595.0	NaN	NaN	NaN	2924.0	173.0	351.0	283.0	281.0	NaN	NaN	NaN	174.0

Para mejorar el modelo de datos, se han añadidos parámetros estacionales de los lags que se han tenido en cuenta, se han representados las funciones seno y coseno de los lags:

```
df['sinlag1'] = np.sin(2*np.pi*(1/1)*df.index)
df['coslag1'] = np.cos(2*np.pi*(1/1)*df.index)

df['sinlag2'] = np.sin(2*np.pi*(1/2)*df.index)
df['coslag2'] = np.cos(2*np.pi*(1/2)*df.index)

df['sinlag3'] = np.sin(2*np.pi*(1/3)*df.index)
df['coslag3'] = np.cos(2*np.pi*(1/3)*df.index)

df['sinlag4'] = np.sin(2*np.pi*(1/4)*df.index)
df['coslag4'] = np.cos(2*np.pi*(1/4)*df.index)

df['sinlag5'] = np.sin(2*np.pi*(1/5)*df.index)
df['coslag5'] = np.cos(2*np.pi*(1/5)*df.index)

df['sinlag6'] = np.sin(2*np.pi*(1/6)*df.index)
df['coslag6'] = np.cos(2*np.pi*(1/6)*df.index)

df['sinlag7'] = np.sin(2*np.pi*(1/7)*df.index)
df['coslag7'] = np.cos(2*np.pi*(1/7)*df.index)
```

Quedando un dataframe de (546, 33).

Vamos a afrontar el problema de Series temporales con:

- Machine Learning
  - Regresión lineal
  - K-vecinos
  - XGBoost
  - Random Forest
- Métodos estadísticos
  - ARIMA

Diferenciación entre modelos

En los modelos de ML, se podrán utilizar utilidades como:

```
from sklearn.model_selection import train_test_split

X = df_data.iloc[:,1:-1]
y = df_data['llamadas']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Para separar los datos de entrenamiento de los datos de prueba, o incluso:

Cross-Val-Score REGLINEAL

```
lin_score = cross_val_score(regl, X_train, y_train, cv=100, scoring = 'neg_mean_squared_error')
lin_score_n = -lin_score
cvs_regl = np.mean(np.sqrt(lin_score_n))
result['cvs_regl'] = cvs_regl
print(cvs_regl)
```

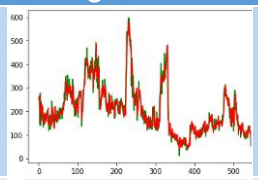
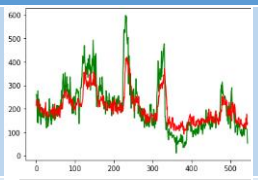
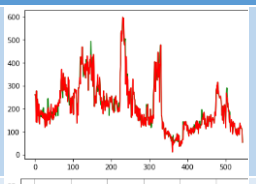
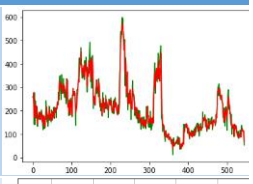
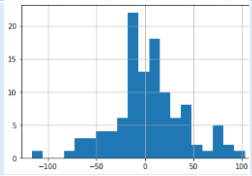
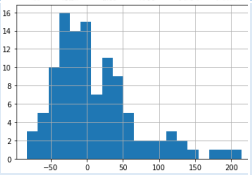
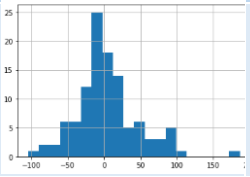
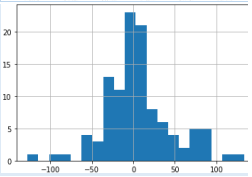
Mientras que en ARIMA, deberán ser datos contiguos y ordenados por fecha:

```
i, m = X.shape

to_train = int(i*0.8)
to_test = i - to_train

print(to_train, to_test)
print(to_train+to_test)
```

Lo que para los modelos de Machine Learning serán parámetros, para ARIMA serán variables exógenas

Resultados

Los modelos han sido evaluados mediante el error cuadrático medio MSE, en la tabla se muestra los resultados de las implementaciones de train_test_split y cross_val_score para los modelos de ML.

	Reg. Lineal	K-vecinos	XGBoost	Random Forest
Predict				
Dist. error				
Detalle error	count 110.000000 mean 3.795990 std 37.390763 min -116.288501 25% -15.903645 50% 1.977554 75% 24.141759 max 103.061431	count 110.000000 mean 11.688770 std 57.734181 min -82.705882 25% -27.147059 50% -1.058824 75% 39.044118 max 213.941176	count 110.000000 mean 5.051632 std 44.175360 min -103.938385 25% -17.392778 50% 0.096153 75% 24.382280 max 186.308411	count 110.000000 mean 5.590000 std 40.961797 min -127.500000 25% -16.437500 50% 2.275000 75% 19.850000 max 133.150000

modelo	MSE	CVS
Reg. Lineal	37.413487	36.464857
Random Forest	41.156574	38.326838
XGBoost	44.263313	39.873525
Kvecinos	58.647770	58.818409

En cuanto al modelo estadístico ARIMA, se ha estudiado la serie de datos mediante la prueba de Dickey-Fuller, obteniendo:

```
from statsmodels.tsa.stattools import adfuller

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

```
adf_test(y)
```

```
1. ADF :  -2.974966475504673
2. P-Value :  0.03729927680264395
3. Num Of Lags :  2
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 543
5. Critical Values :
         1% :  -3.442450336733213
         5% :  -2.8668774460774498
         10% :  -2.5696126760816553
```

Con un **P-valor** de 0.03, quedando anulada la hipótesis nula H0, la serie es estacionaria. Mediante la librería de auto_arima, obtenemos el mejor modelo de ARIMA, en nuestro caso un SARIMAX(2,1,0).

```
Best model:  ARIMA(2,1,0)(0,0,0)[0]
Total fit time: 3.829 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	546
Model:	SARIMAX(2, 1, 0)	**Log Likelihood**	-2780.526
Date:	Wed, 16 Jun 2021	**AIC**	5567.052
Time:	12:05:02	**BIC**	5579.954
Sample:	0	**HQIC**	5572.096
	- 546		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3174	0.030	-10.734	0.000	-0.375	-0.259
ar.L2	-0.1603	0.033	-4.827	0.000	-0.225	-0.095
sigma2	1584.6875	54.165	29.257	0.000	1478.527	1690.848

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	520.24
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.34	**Skew:**	-0.57
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.65

La distribución de los datos para entrenamiento y pruebas quedan de la siguiente manera:

```
i, m = X.shape

to_train = int(i*0.8)
to_test = i - to_train

print(to_train, to_test)
print(to_train+to_test)
```

```
X_train, y_train = df_data.iloc[:to_train, ((df_data.columns != 'Fecha') & (df_data.columns != 'llamadas'))], df_data['llamadas'][:to_train]
X_test, y_test = df_data.iloc[-to_test:, ((df_data.columns != 'Fecha') & (df_data.columns != 'llamadas'))], df_data['llamadas'][-to_test:]

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

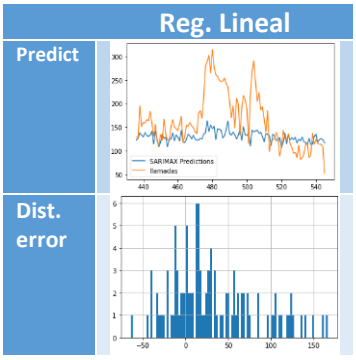
```
(436, 31) (436,)
(110, 31) (110,)
```

Se normalizan los datos:

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

y_train = np.log1p(y_train)
## y_test = np.log1p(y_test)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

El resultado del modelo no es todo lo bueno que se esperaba, la tendencia condiciona mucho el modelo.



Detalle error	count 110.000000 mean 32.553175 std 50.069813 min -63.864022 25% -2.054306 50% 18.405069 75% 60.461709 max 166.783027
MSE	MSE SARIMAX 59.530703

Conclusiones

El modelo que mejor rendimiento tiene es la **Regresión lineal**. Si nos fijamos en sus errores, es el que tiene el error más pequeño de media, con un valor de 3.79, y su desviación típica de 37.39

```
count    110.000000
mean       3.795990
std       37.390763
min     -116.288501
25%      -15.903645
50%        1.977554
75%       24.141759
max      103.061431
```

El método estadístico SARIMAX es con diferencia el peor de todos con un MSE de:

	MSE
SARIMAX	59.530703

Con una media de error de 32.55 y una desviación típica de 50.06

No siempre los problemas más complejos requieren modelos complejos, en esta ocasión, el modelo más simple, la regresión lineal, produce predicciones más precisas que modelos como XGBoost, Random Forest, K-vecinos.

Limitaciones

En este proyecto la cantidad de registros ha sido una limitación teniendo que entrenar modelos de ML únicamente con 546 registros, además, en este intervalo ha habido una situación muy especial que ha sido la pandemia, donde hay un impacto en el comportamiento, se ha decidido no quitar estos datos debido al número de datos para analizar.

Manual Frontend

El cuadro de mando es una webApp donde, cargando los ficheros de los envíos que se van a recibir en las oficinas, se realiza una predicción de la cantidad de llamadas que se va a recibir en el departamento de Contact Center.

Los ficheros con la información de los envíos se obtienen de la plataforma de informes de la empresa de mensajería.

Se adjuntará por correo electrónico los siguientes ficheros para la carga en el dashboard ya que no están públicos en el repo.

4800.xlsx	06/07/2021 16:24	Hoja de cálculo d...	342 KB
4802.xlsx	06/07/2021 16:26	Hoja de cálculo d...	581 KB
4803.xlsx	06/07/2021 16:26	Hoja de cálculo d...	384 KB
4806.xlsx	06/07/2021 16:27	Hoja de cálculo d...	386 KB
4810.xlsx	06/07/2021 16:28	Hoja de cálculo d...	1.641 KB
df_Call	05/07/2021 16:14	Archivo	13 KB
df_data	05/07/2021 16:00	Archivo	145 KB
df_Envios	05/07/2021 16:10	Archivo	52 KB
new_calls.csv	10/05/2021 13:50	Archivo de valores...	14.633 KB
pass.csv	08/07/2021 11:34	Archivo de valores...	1 KB
regl.pkl	05/07/2021 16:00	Archivo PKL	1 KB
scaler.pkl	05/07/2021 16:00	Archivo PKL	2 KB
shippings.csv	10/05/2021 13:50	Archivo de valores...	391.084 KB

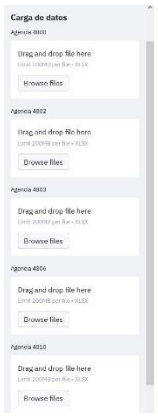
Los datos de las llamadas se obtienen de una consulta SQL a una base de datos Azure.

En el repositorio no está el password de la conexión SQL en texto plano, sino que está en un fichero csv (pass.csv) dentro de la carpeta data (se adjunta por mail, ya que no es pública en el repo).

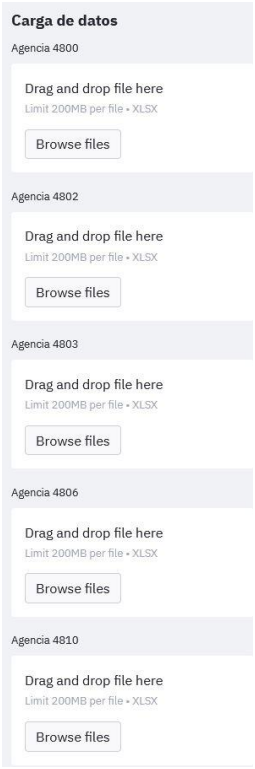
4800.xlsx	06/07/2021 16:24	Hoja de cálculo d...	342 KB
4802.xlsx	06/07/2021 16:26	Hoja de cálculo d...	581 KB
4803.xlsx	06/07/2021 16:26	Hoja de cálculo d...	384 KB
4806.xlsx	06/07/2021 16:27	Hoja de cálculo d...	386 KB
4810.xlsx	06/07/2021 16:28	Hoja de cálculo d...	1.641 KB
df_Call	05/07/2021 16:14	Archivo	13 KB
df_data	05/07/2021 16:00	Archivo	145 KB
df_Envios	05/07/2021 16:10	Archivo	52 KB
new_calls.csv	10/05/2021 13:50	Archivo de valores...	14.633 KB
pass.csv	08/07/2021 11:34	Archivo de valores...	1 KB
regl.pkl	05/07/2021 16:00	Archivo PKL	1 KB
scaler.pkl	05/07/2021 16:00	Archivo PKL	2 KB
shippings.csv	10/05/2021 13:50	Archivo de valores...	391.084 KB

Los datos que se visualizaran son datos demo de un día en concreto, 06/07/2021.

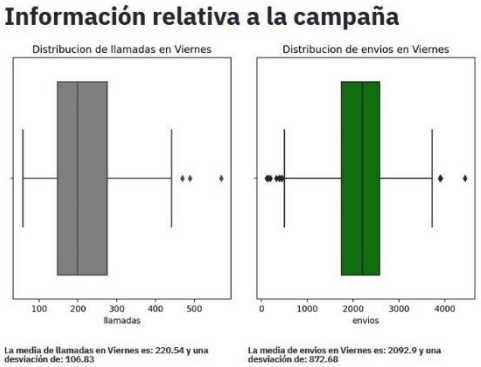
La disposición del cuadro de mando es de la siguiente manera:



Tenemos un panel lateral donde se podrán subir los ficheros con los datos de los envíos, uno de cada oficina.



En la parte central tenemos información de las distribuciones de los envíos y de las llamadas de nuestro set de datos, para los que está entrenado el modelo



Una vez cargado los ficheros, el propio dashboard hace una llamada SQL y carga los datos de las llamadas, y tras el procesamiento de la información, hace la predicción de llamadas para el día en el cual se han cargado los datos. El modelo que se ha utilizado para el dashBoard es la Regresión lineal al ser el que obtiene los mejores resultados.