

PROYECTO FIN DE BOOTCAMP

Plano de Mantenimiento de Máquinas

Enero de 2022
TALITA C COLL CARDENAS
The Bridge Academy

Índice.

Tabla de contenido

1.-OBJETIVO	2
2.-DATOS.....	2
3.-ANALISE EXPLORATORIA	5
ANALICE POR CANTIDAD DE REPUESTOS A LO LARGO DE LOS AÑOS	6
ANALICE POR REGISTRO AGRUPADA POR MÁQUINAS.....	6
ANALICE POR TIPO DE REPUESTO.	7
ANALICE DE LOS COSTES	7
ANALICE POR PLAZOS	9
BREVE ANALICE SOBRE LOS NÚMEROS DE SERIE	10
CORRELACIONES	10
CONCLUSIÓN DEL EDA.....	11
4.-MODELOS PARA MACHINE LEARNING	12
1 – IDENTIFICAR SI LO DATOS SON ESTACIONARIOS.....	14
3- IDENTIFICAR EL GRADO AUTO REGRESIÓN Y MEDIAS MÓVILES.	15
4- APLICANDO EL MODELO A LAS SERIES TEMPORALES.	16
5- OBTENER VALORACIÓN DEL MODELO.....	18
5- ANALIZAR LOS VALORES RESIDUALES COMO FORMA DE SUSTENTAR LOS RESULTADOS OBTENIDOS PREVIAMENTE.....	19
5.-PREPARAR EL MODELO PARA LA PRODUCCIÓN.....	20
DEMONSTRACIÓN CON UN ID DE ALTA FRECUENCIA.	20
6.-CONSIDERACIONES FINALES.	24
7. – BIBLIOGRAFÍA Y CONSULTAS DE APOYO.	24

1.-Objetivo

Este proyecto tiene como finalidad la aplicación de modelos estadísticos de regresión en históricos de datos afines de mejorar la performance en tiempo de servicios suministrados a una empresa ficticia a su cartera de cliente en la base instalada.

Los modelos serán hechos en función a los históricos de consumo de 4 años, donde todos los repuestos han sido consumidos para la reparación de máquinas.

2.-Datos

Los datos para este proyecto han sido basados en un histórico real de una empresa, pero por derechos todos los datos han sido cambiados, apenas mantenido parte de la información para no perder proporcionalidad de los datos, caso contrario el modelo podría no simular una situación real.

Los datos han sido proporcionados en formato .xlsx.

Los datos consisten en código de repuesto, cantidades, fecha de petición del servicio, fecha de realización del servicio y tipo de productos, además de otras informaciones que no relevantes para el modelo.

El rango de las fechas de petición va desde 1 de enero de 2018 hasta 31 de diciembre de 2021, o sea trabajaremos con 48 meses histórico para dibujar el proyecto.

In [5]:	1 df.sample(20)
Out [5]:	id doc_type transaction_date order_type order_date quantity unit_cost extended_cost item_type
	11837 A1 2017-11-30 2017-10-26 0 0.0000 0.00 SPA
	41227 A1 2018-02-12 2018-01-29 0 0.0000 0.00 SPA
	135632 B1 2019-08-29 2019-08-04 1 251.1727 251.17 MOD
	87155 B1 2018-10-26 2018-09-24 0 75.7500 0.00 MOD
	150900 A1 2019-12-05 2019-12-01 0 0.0000 0.00 SPA
	105266 B1 2019-02-28 2019-02-18 1 165.9000 165.90 MOD
	185163 A1 2020-05-23 2020-05-10 0 0.0200 0.02 MOD

Figura 1: El DataSet, con informaciones generales de movimientos de stock.

Exhaustivos bloques de código han sido realizados para reemplazar los códigos originales de manera aleatoria, pero manteniendo la proporcionalidad de los hechos y crear informaciones como modelo de máquina y números seriales.

```
In [19]: 1 #Split the dataframe in modules and spares
2 dfmod = df[df.item_type=='MOD']
3 dfspa = df[df.item_type=='SPA']

In [20]: 1 #This function is creating dataframes to supporting the replacement
2 def code_creat(a,b,c,d,e,f,g):
3     a=[]
4     for i in b:
5         id = c + (d % random.randrange(e)).upper()
6         a.append(id)
7         f = pd.DataFrame(a,columns=[g])
8     return f
```

Figura 2: Función para la creación de datos aleatorios.

```
In [21]: 1 #A dictionary with the features to make replacements in the dataframe
2 codes={'m':{'a':'mback', 'b':range(len(dfmod.id.value_counts())), 'c':'M', 'd':'%06x',
3           'e':999999, 'f':'mdf', 'g':'cid'},
4         's':{'a':'sback', 'b':range(len(dfspa.id.value_counts())), 'c':'S', 'd':'%08x',
5           'e':99999999, 'f':'sdf', 'g':'cid'},
6         'p':{'a':'pback', 'b':range(45), 'c':'P', 'd':'%04x',
7           'e':99999, 'f':'pdf', 'g':'id'}}}
```



```
In [22]: 1 #Apply def in elements of dictionary
2 for k, v in codes.items():
3     for i in k:
4         exec(f'{k}df = code_create(**v)')
```

Figura 2.1: Características y aplicación de la función anterior.

```
In [54]: 1 df.sort_index()
```


Out[54]:

	id	doc_type	transaction_date	order_type	order_date	quantity	unit_cost	extended_cost	item_type
21525	S031B79EA	D1	2017-11-27	U1	2017-10-28	2	2.9800	5.96	SPA
21570	S030F5A8F	D1	2017-12-03	U1	2017-10-29	1	29.8400	29.84	SPA
21571	M0B5DB4	D1	2017-11-25	U1	2017-11-01	1	49.9400	49.94	MOD
21590	S0322692D	D1	2017-12-17	U1	2017-11-12	2	18.7300	37.46	SPA
21591	S04164E76	D1	2017-12-14	U1	2017-11-28	2	2.4300	4.86	SPA
...
284394	M05E910	D1	2017-12-15	U1	2017-12-08	1	24.6200	24.62	MOD
284395	M05E910	D1	2017-12-13	U1	2017-12-06	1	24.6200	24.62	MOD
284396	S0065CACD	D1	2021-12-23	U1	2021-11-20	1	51.0400	51.04	SPA
284397	M0E779F	D1	2021-12-17	U1	2021-11-19	1	440.0000	440.00	MOD
284398	S04BA34B7	D1	2021-12-27	U1	2021-11-24	1	334.4906	334.49	SPA

47798 rows × 9 columns

Figura 3: El DataSet final, con los datos ya tratados sin perder proporcionalidad.

Hemos creado una cartera de clientes ficticias a través de la librería *faker*, para que en la producción se pueda ubicar la máquina cual predeciremos fallar.

```
In [41]: 1 cdf.head()
```


Out[41]:

	customer_id	name	address	zone	cp
0	CS0000001	Ángeles Ámbar Marco Mendoza	Pasaje de Rita Moll 95	Segovia	11224
1	CS0000002	Ernesto Alcázar Zamora	Paseo de Etelvina Mendoza 993 Puerta 3	Cuenca	25092
2	CS0000003	Manolo Cortes Boix	Plaza de Adrián Nogueira 41 Puerta 1	Cuenca	94571
3	CS0000004	Aura Canales Serna	Vial Marisol Amat 39 Puerta 0	Castellón	35790
4	CS0000005	Atilio Dalmau Alemany	Plaza Prudencio Torrecilla 14 Puerta 6	Alicante	30535

Figura 4: Cartera de cliente creada con la librería *faker*.

Para completar la fase anterior, otra tabla de datos ha sido creada con las máquinas instaladas en cada cliente con sus respectivos números de serie.

In [95]: 1 ibdf.sample(10)

Out[95]:

	serial_id	product_id	customer_id	install_date
11961	SN0011962	PF501	CS0000035	2015-06-07
685	SN0000686	P121F	CS0000054	2014-05-10
5234	SN0005235	PE390	CS0000183	2013-12-04
3981	SN0003982	PFE6E	CS0000120	2014-01-30
11736	SN0011737	PB91E	CS0000001	2015-07-20
4502	SN0004503	P75B8	CS0000016	2014-03-13
2152	SN0002153	PFE6E	CS0000159	2016-11-08
5535	SN0005536	P152E9	CS0000004	2014-10-30
9744	SN0009745	P14627	CS0000149	2014-06-04
7412	SN0007413	PFE6E	CS0000116	2016-01-03

Figura 5: Tabla de dimensiones de base instalada.

La fase de tratamiento de datos es la que más tiempo a llevado debido a la complejidad de los códigos para el reemplazamiento de los datos.

3.-Analise Exploratoria

Con los datos previamente trabajados empezaremos la primera parte de la exploración de los datos.

Primeramente, miramos la descripción general de los datos, cuales ya nos da alguna información sobre casos aislados que no corresponde al comportamiento natural de los servicios y consumos.

In [5]:	1	df.describe()					
Out[5]:							
	quantity	unit_cost	extended_cost	product_age	year	month	leadtime
count	46939.000000	46939.000000	46939.000000	46939.000000	46939.000000	46939.000000	46939.000000
mean	1.410192	170.338178	178.715680	1698.550736	2019.578730	6.566927	18.337289
std	2.430719	313.776672	336.676279	605.870537	1.133771	3.488281	9.803283
min	1.000000	0.008800	0.010000	241.000000	2018.000000	1.000000	1.000000
25%	1.000000	13.460000	16.000000	1269.000000	2019.000000	3.000000	10.000000
50%	1.000000	68.390000	72.380000	1697.000000	2020.000000	7.000000	18.000000
75%	1.000000	166.260000	174.300000	2132.000000	2021.000000	10.000000	27.000000
max	250.000000	6484.210500	11321.170000	3149.000000	2021.000000	12.000000	35.000000

Figura 6: DataFrame describe.

El grafico tipo *boxplot*, podemos tener una observación visual más clara de los casos aislados, donde podemos detectar claramente que los casos de consumo mayores que 70 son aislados y deben ser eliminados de los datos para que haya menos ruido posible en el modelo.

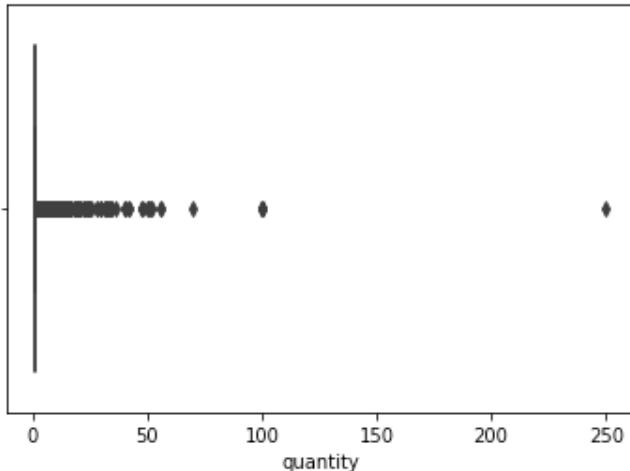


Figura 7: BoxPlot

El grafico tipo *boxplot*, podemos tener una observación visual más clara de los casos aislados, donde podemos detectar claramente que los casos de consumo mayores que 70 son aislados y deben ser eliminados de los datos para que haya menos ruido posible en el modelo.

También hemos utilizado los métodos de cuartiles y agrupamiento de datos con el objetivo de detectar los casos aislados, pero los que has sido más efectivos en este caso han sido el gráfico y el agrupamiento por cantidades, pues las observaciones con cantidades por encima de lo normal, era de muy baja frecuencia, haciendo la densidad casi imperceptible.

```
In [7]: 1 #Quantiles also, can support us to find outliers, but in this case, the volum is too low and we are going to use
2 np.quantile(df.quantity, 0.999)

Out[7]: 28.12399999999616

In [8]: 1 #Check outlier with a basic groupby
2 ratio = df.groupby(['quantity'])['quantity'].count()/len(df)*100
3 ratio.sort_index(ascending=False).head(10)

Out[8]: quantity
250    0.002130
100    0.010652
70     0.002130
56     0.004261
52     0.004261
51     0.002130
50     0.004261
48     0.006391
42     0.004261
40     0.006391
Name: quantity, dtype: float64
```

Figura 7.1: Método de los cuartiles con la librería de Numpy y agrupamiento por cantidad.

Analice por cantidad de repuestos a lo largo de los años.

Al agrupar las cantidades por año, hemos podido observar que en los años de 2020 y 2021, aunque esperábamos una reducción debido a la pandemia que estamos sufriendo desde principios de 2020, la cantidad de repuestos requerida han sufrido un leve aumento con relación a los años anteriores.

```
In [12]: 1 sns.barplot(x=df.year, y=df.quantity);
```

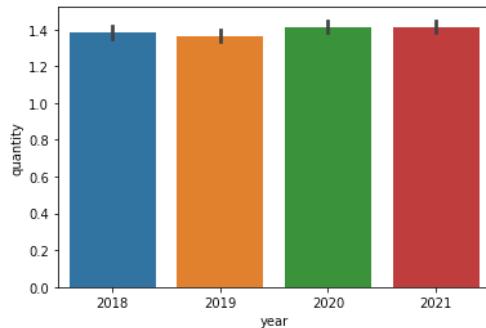


Figura 8: Suma de cantidad de repuestos requeridos por año.

Analice por registro agrupada por máquinas

Hemos agrupado los casos históricos por máquinas con el objetivo de observar las máquinas cuales más servicios han requerido.

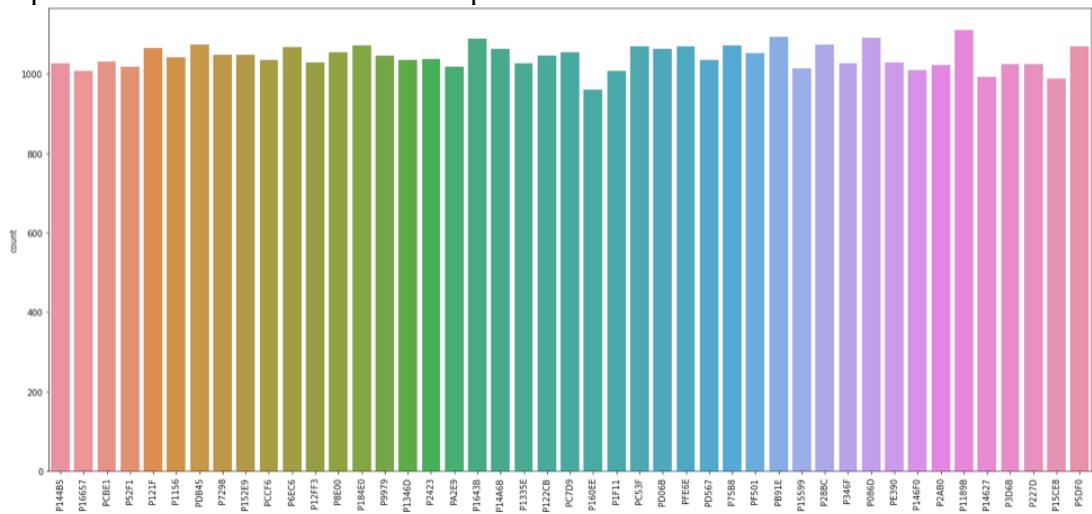


Figura 8: Suma de cantidad de repuestos requeridos por año.

Hemos podido observar que la máquina P5DF0 ha tenido 1582 requerimiento de recambio, con una media de 1,47 unidades por registro y desviación típica de 2,06.

		min	max	sum	mean	std
product_id						
P5DF0	1	32	1582	1.478505	2.059681	
P75B8	1	40	1576	1.471522	2.287488	
P1189B	1	48	1564	1.409009	1.994200	
P121F	1	50	1541	1.445591	2.156189	
P1643B	1	48	1535	1.410846	2.122589	
P9979	1	35	1535	1.468900	2.290293	
PC7D9	1	51	1533	1.453081	2.142776	
P184E0	1	48	1525	1.423903	2.202706	
P28BC	1	32	1510	1.404651	1.648027	
PB91E	1	24	1510	1.380256	1.704043	

Figura 8.1: Los modelos de máquina que encabezan las máquinas con más servicios requeridos.

Hemos podido observar que la máquina PAE51 ha tenido 1623 requerimiento de recambio, con una media de 1,55 unidades por registro y desviación típica de 2,94.

Analice por tipo de repuesto.

Según el grafico a seguir, podemos observar que aproximadamente 65% de los recambios requeridos refiéranse a piezas y no de módulos.

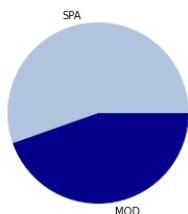


Figura 9: Piezas versus módulos.

Analice de los costes

Analizando los costes de reparación de las máquinas, podemos observar que en 2018 la pieza de código S05B86751, ha tenido un aumento de costes considerable considerando los demás años. Lo mismo observamos para el componente S018E5099 en 2021.

Aunque no tengamos acceso a la información financiera, este grafico nos proporciona datos suficientes para deducir que algo atípico ja pasado, sea con el coste standard del ítem o algún problema con el sistema y una investigación de costes, es recomendada.

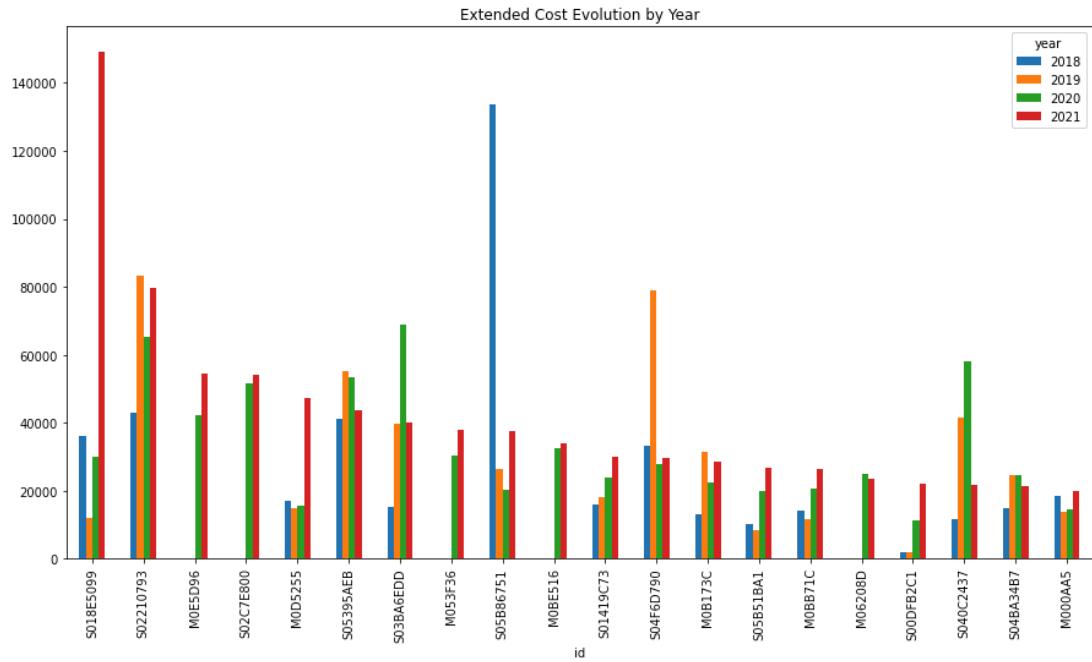


Figura 10: Analice de costes de servicios por componentes separados por año.

En el gráfico a seguir tenemos los costes acumulados por modelo de máquinas, donde observamos que el modelo P128E1, a lo largo de los años ha tenido más significancia que los demás modelos de alto consumo.

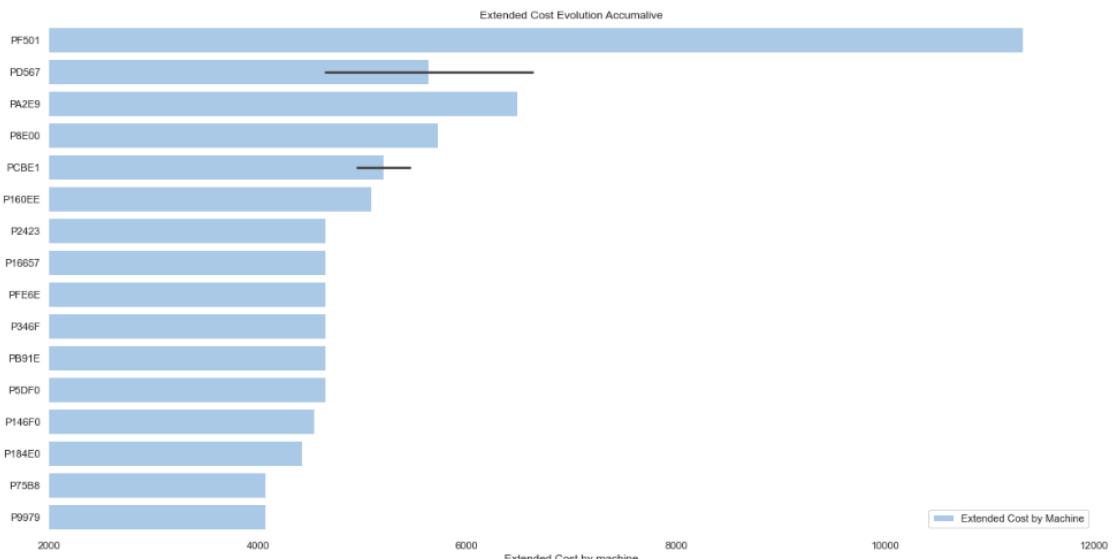


Figura 10.1: Analice de costes de servicios por máquinas acumulado.

Analice por plazos

Ahora analizaremos los plazos de realización del servicio desde el momento que es requerido hasta el momento que es finalizado por el técnico, cual es el real objetivo de este proyecto, que es la reducción de los plazos.

Según la tabla a seguir, los plazos más frecuentes son de 15 días, lo que del punto de vista del negocio es considerablemente alto, comparando con los 2 días de objetivo estipulado por la compañía.

leadtime	count
15	1567
7	1564
2	1524
22	1420
6	1419
8	1411
18	1407
23	1406
11	1404
28	1399

Figura 11: los 10 plazos más frecuentes que tardan en realizar un servicio por completo.

Observado el gráfico de líneas, los años de 2018, 2019 y 2020, los plazos de realización de los servicios tenían una distribución bastante homogénea, pero en 2021, hemos percibido una gran mejora, aumentando el número de servicios realizados en menos días y disminuyendo los servicios de largo plazo.

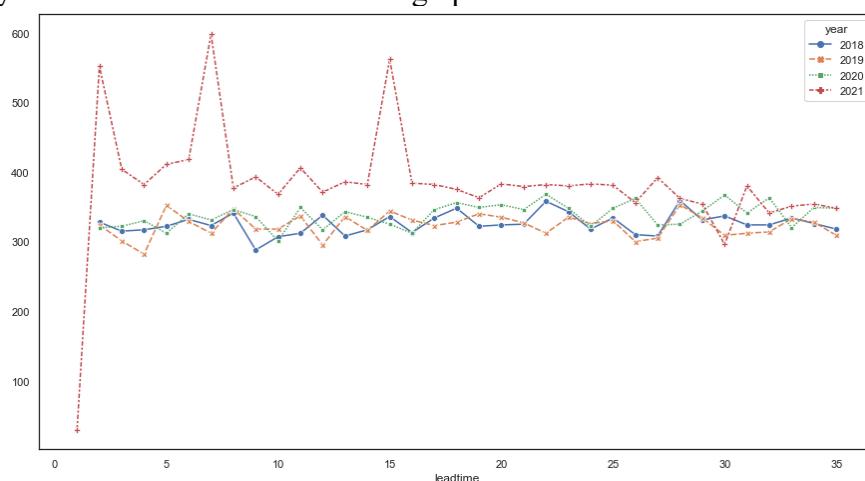


Figura 11.1: La evolución de los plazos de realización de los servicios a lo largo de los años.

También hemos aplicado el método de los cuartiles para observar el agrupamiento por frecuencia.

```
In [36]: 1 #Checking quantiles lead time find the goal
2 q = df.leadtime.quantile(.25, .5, .75)
3 q
```



```
Out[36]: 0.25    10.0
0.50    18.0
0.75    27.0
Name: leadtime, dtype: float64
```

Breve analice sobre los números de serie

La observación de los números de serie, es importante para detectar si hay alguna máquina cuyos los fallos son más frecuentes, cuales puede tener causas ambientales, intensidad de uso o hasta un fallo en la producción del material.

En gráfico podemos observar que la máquina de número SN0010290, ha tenido más costes que todas las demás a lo largo de los años.

Sin embargo, los datos están ordenados por los costes de 2021, lo que significa que el serial de la máquina con mayor coste de 2021 ha sido el número SN0004886.

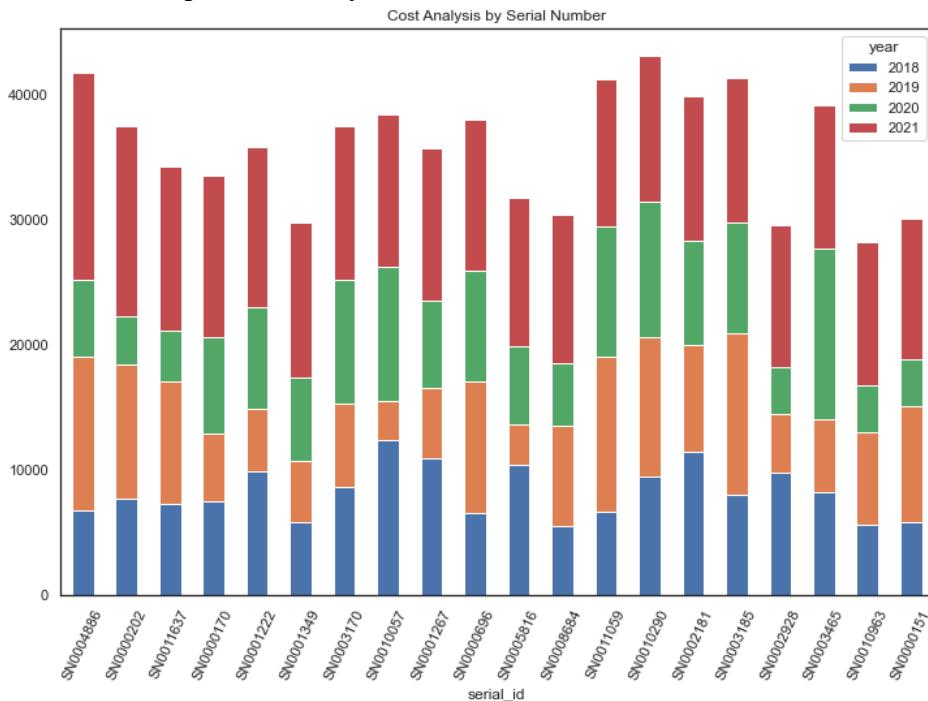


Figura 12: Costes por número de serie con los costes apilados por años.

Correlaciones

Nos gustaría detectar si hay alguna relación con costes y plazo, o costes y edad de la edad de la máquina para detectar patrones, pero las correlaciones son básicamente insignificantes, lo que nos dificulta encontrar una causa raíz de los fallos solamente observando los datos que tenemos.

	quantity	unit_cost	extended_cost	product_age	year	month	leadtime
quantity	1.000000	-0.099397	-0.064559	0.007832	0.008549	-0.000270	0.007493
unit_cost	-0.099397	1.000000	0.961951	-0.021085	-0.033297	0.008981	0.002300
extended_cost	-0.064559	0.961951	1.000000	-0.020308	-0.031193	0.008598	0.002003
product_age	0.007832	-0.021085	-0.020308	1.000000	0.698774	0.180212	-0.026241
year	0.008549	-0.033297	-0.031193	0.698774	1.000000	0.006956	-0.032457
month	-0.000270	0.008981	0.008598	0.180212	0.006956	1.000000	-0.032377
leadtime	0.007493	0.002300	0.002003	-0.026241	-0.032457	-0.032377	1.000000

Figura 12: Tabla de correlación entre los datos aportados.

El mapa de calor ha seguir nos da una visión de las correlaciones que son muy débiles, solamente a las características intrínsecamente relacionadas, como coste unitario y coste total o el año del servicio y la edad de la máquina.

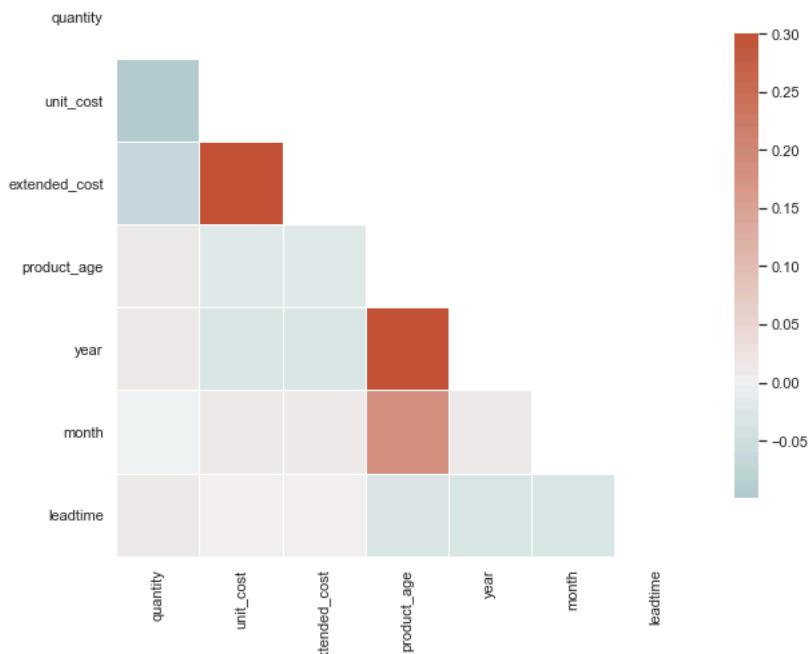


Figura 12: Mapa de calor de correlaciones de las características de los registros.

Conclusión del EDA

- Aunque hemos empezado la situación actual del COVID 19, los servicios han aumentado desde 2020.
- Si miramos los datos con una perspectiva global, la desviación standard es bastante baja.
- La necesidad de cambiar piezas es casi el doble de la necesidad de cambiar módulos, lo que indica que algunos cambios pueden ser realizados por los técnicos en campo.
- Básicamente no hay correlación entre los datos, apenas los intrínsecos.
- El plazo de servicio más común es de 22 días, bastante lejos del aceptable, lo demuestra que el modelo de predicción debe ser mejorado.

4.-Modelos para Machine Learning

Como el objetivo es mejorar los plazos de servicios por falta de repuestos, queremos mejorar la predicción de necesidad de repuestos en series temporales. Basados en pesquisas e indicaciones, vamos trabajar el modelo de Auto Regresión Integrada de Medias Móviles (ARIMA).

La parte de auto regresión (AR), representada por la letra p , indica que la variable evolutiva de interés es devuelta a sus propios valores desfasados, o sea, el valor previo. La parte de medias móvil (MA), representada por la letra q , indica el error de la regresión, en otras palabras, es la combinación lineal de las ordenes de error que ocurren en momentos pasados.

La parte integrada (I), representada por la letra d , indica los valores que han sido sustituidos por la diferencia entre sus valores actuales y los valores previos. El proceso de diferenciación puede ser realizado más de una vez.

El propósito de las ordenes, pdq , es hacer que el modelo se ajuste a los datos de la mejor manera posible.

$$y_t = -(\Delta^d Y_t - Y_t) + \phi_0 + \sum_{i=1}^p \phi_i \Delta^d Y_{t-i} - \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t$$

Para aplicar el modelo, debemos seguir el método de decisiones Jenkins Box, donde detectamos parámetros y evaluamos el modelo.

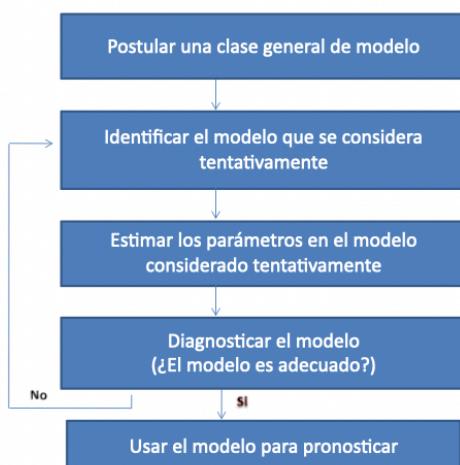


Figura 13: Diagrama de Jenkins Box.

Desarrollo del modelo

Primeramente, debemos ajustar el dataset, agrupándolo por los períodos que deseamos modelar. En los primeros modelos hemos ajustado por meses y testeando el dataset como un todo, apenas considerando el volumen de servicios realizados en dichos períodos.

```
In [11]: 1 general
Out[11]: order_date    944
          2018-01-31    944
          2018-02-28    936
          2018-03-31    973
          2018-04-30    995
          2018-05-31    757
          2018-06-30    978
          2018-07-31    881
```

Figura 14: DataSet con datos ajustados.

Analizamos la distribución de los servicios a largo de los 4 años trabajados.

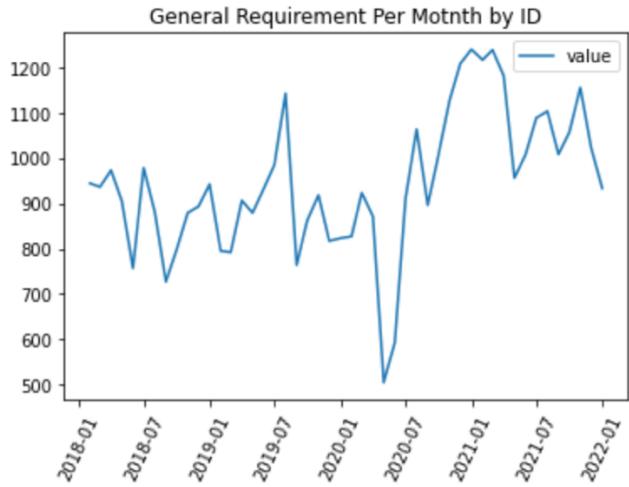


Figura 15: Grafico de linea sobre los niveles de servicios generales.

En el siguiente grafico podemos observar los niveles de servicios descompuestos por tendencia, estacionalidad, y valores residuales.

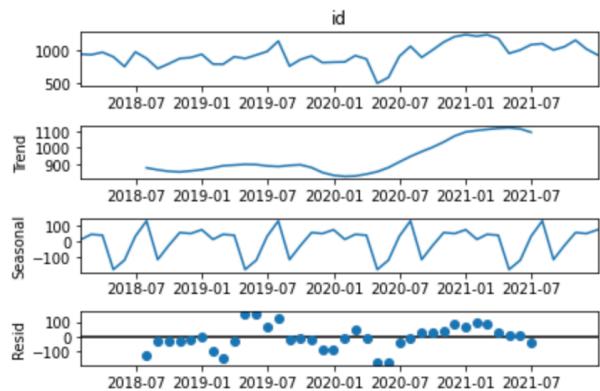


Figura 16: Grafico de estacionalidad desmontado.

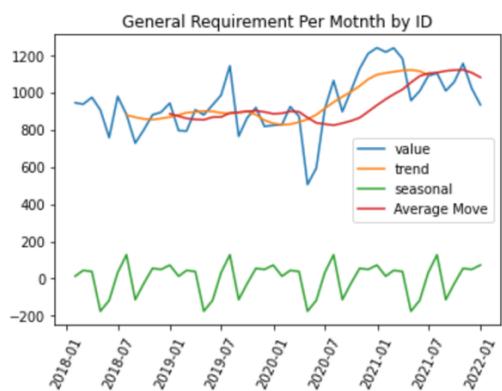


Figura 17: Grafico de estacionalidad desmontado con la composición sobrepuesta.

Mirando los grafico, es bastante evidente la estacionalidad que los datos contienen. Se ve una bajada en los servicios en verano y temporada post fiestas de finales de año.

1 – Identificar si los datos son estacionarios.

Para poder realizar la detección de la estacionaridad, aplicamos la prueba Dickey-Fuller Aumentada que consiste en rechazar la hipótesis nula.

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_{p-1} \Delta y_{t-p+1} + \epsilon_t$$

Según el test de Fuller, el p-valor es mayor que 0.05, lo que nos muestra que no podemos rechazar la hipótesis nula, lo que significa que las series temporales no son estacionarias, resultando en la búsqueda de la diferencia integrada.

```
#Check if the general set is stationary.
def adfuller_test(data):
    results=adfuller(data)
    labels = ['ADF Test Statistic','p-value','#Lags used','Number of observations']
    for v,l in zip(results,labels):
        print(l+' :'+str(v))
    if results[1] <=.05:
        print('P value is less than 0.05, that means we can reject the null hypothesis,\n'
              'so we conclude that series is stationary.')
    else:
        print('P value shows we can't reject the Null Hypothesis, that means the series is not stationary.')
    return
```

Figura 19: Función para aplicar test ADFuller.

```
ADF Test Statistic:-1.8071493411014237
p-value:0.37691806560754193
#Lags used:2
Number of observations:45
```

Figura 20: Resultado del test ADFuller.

2- Identificar el grado de diferenciación integrada.

Una vez que hemos confirmado que las series temporales no son estacionarias, tenemos que identificar el grado de diferenciación integrada.

Hemos preparado la siguiente función para detectarlo(d).

```
1 #Find difference for first and second order
2 def plot_diff(data):
3     if adfuller(data)[1]<=0.05:
4         print('P value is less than 0.05, that means we can reject the null hypothesis,\n'
5               'so we conclude that series is stationary.')
6         d=0
7     else:
8         for i in range(1,3):
9             diff = data.diff(i)[i:]
10            if adfuller(data.diff().dropna())[1]<0.05:
11                d=1
12                print(f'P_value:{adfuller(data.diff().dropna())[1]},the order for model is {d}.')
13            elif adfuller(data.diff().diff().dropna())[1]<0.05:
14                print(f'P_value:{adfuller(data.diff().diff().dropna())[1]},the order for model is {d}.')
15                d=2
16            else:
17                print("ERROR!!!!")
18
19
20 plot_diff(general)
P_value:9.768367820613255e-12,the order for model is 1.
```

Figura 20: Función para identificar la diferencia y su resultado.

En la función anterior (figura 20), hemos identificado la diferenciación integrada de orden $d=1$, sin embargo, en el siguiente gráfico, el grado de diferenciación es bastante evidente.

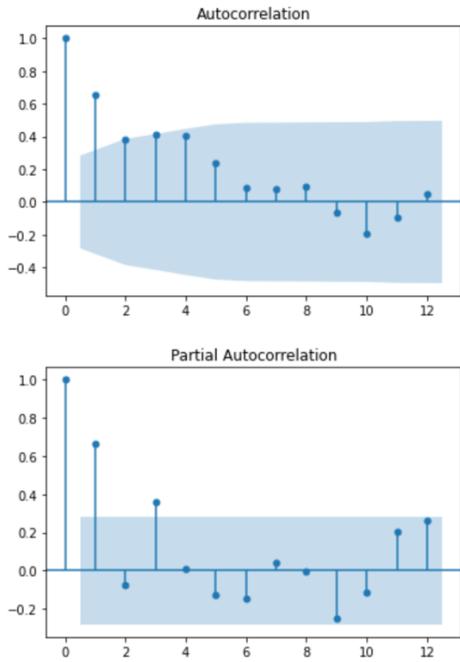


Figura 21: Grafico de auto correlación y auto correlación parcial.

3- Identificar el grado auto regresión y medias móviles.

Para poder detectar los grados de p y q , hemos aplicado el Criterio de Información Bayesiano, BIC y creamos una función donde optimizamos la detección del los grado AR y MA, aplicando tal método.

```

1 #we are going to apply ARIMA model to find the best order
2 def getpq(data):
3     #We have identified previously the Integrated Difference
4     d=1
5     #Maximus iterations will be the set divided in 12, what means we are making 4 iterations
6     p_max = int(len(data) / 12)
7     q_max = int(len(data) / 12)
8     matrix_bic = []
9     for p in range(p_max +1):
10         aux= []
11         for q in range(q_max+1):
12             try:
13                 model = ARIMA(data, order=(p,d,q)).fit(disp=False)
14                 aux.append(model.bic)
15             except:
16                 aux.append(None)
17             matrix_bic.append(aux)
18     matrix_bic = pd.DataFrame(matrix_bic)
19     #Assigning p and q to matrix in order to find the best(lowest) relation.
20     p,q = matrix_bic.stack().astype('float64').idxmin()
21     print(matrix_bic)
22     #Plot a heatmap to identify the best set of orders.
23     #The darkest is the section in the graph, the lowest is the order and the best is the correlation
24     fig, ax = plt.subplots(figsize=(10, 8))
25     ax = sns.heatmap(matrix_bic,
26                         mask=matrix_bic.isnull(),
27                         ax=ax,
28                         annot=True,
29                         fmt=' .2f',
30                         );
31     ax.set_title('BIC');
32     return p,q

```

Figura 22: Función para detectar las ordenes de p y q .

Hemos limitados las interacciones posibles con la longitud de los datos partido en 12, o sea, si tenemos 48 series temporales, las interacciones se han limitado a 4+1.

El resultado de la función nos trae la mejor combinación de ordenes posibles dentro de este rango de interacciones.

Al observar el siguiente grafico, hemos podido detectar que la mejor combinación de ordenes es de $p=2, q=0$ y como ya lo habíamos visto previamente, $d=1$.

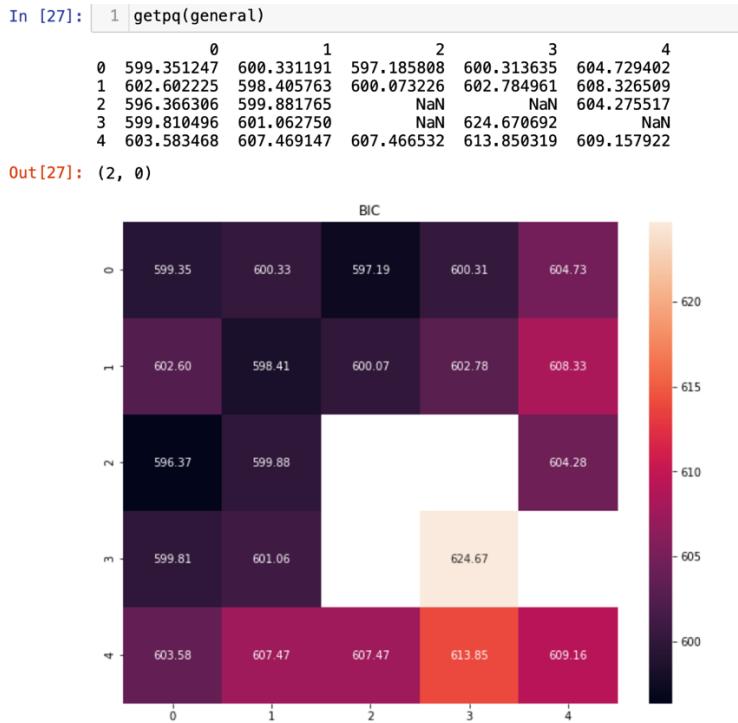


Figura 23: Resultado de la función getpdq.

4- Aplicando el modelo a las series temporales.

Hemos determinado los test de entrenamiento y test en las proporciones de train=75% y test=25%.

```
1 #Definir the train as 75% from dataset and 25% for test
2 tsize = int(len(general)*.75)
3 train = general.iloc[:tsize]
4 test = general.iloc[tsize:]
```

Figura 24: Train y test Split.

Aplicamos el modelo en el set de pruebas y la combinación al parecer es bastante adecuada. Para estar seguros si la combinación es la adecuada para el set de entrenamiento, hemos ejecutado las funciones anteriores y hemos confirmado que la combinación es adecuada.

```
: 1 #Using the previous combination for fit our model
2 model = ARIMA(train, order=(2,1,0))
3 model_fit = model.fit(disp=0)
4 print(model_fit.summary())
```

ARIMA Model Results

Dep. Variable:	D.id	No. Observations:	35
Model:	ARIMA(2, 1, 0)	Log Likelihood	-218.957
Method:	css-mle	S.D. of innovations	125.369
Date:	Sun, 30 Jan 2022	AIC	445.914
Time:	00:06:09	BIC	452.135
Sample:	02-28-2018	HQIC	448.062
	- 12-31-2020		

coef	std err	z	P> z	[0.025	0.975]	
const	7.5838	13.407	0.566	0.572	-18.694	33.862
ar.L1.D.id	-0.1938	0.151	-1.282	0.200	-0.490	0.103
ar.L2.D.id	-0.4166	0.147	-2.831	0.005	-0.705	-0.128

Roots

Real	Imaginary	Modulus	Frequency	
AR.1	-0.2326	-1.5317j	1.5493	-0.2740
AR.2	-0.2326	+1.5317j	1.5493	0.2740

Figura 25: Resultado de entrenamiento del modelo.

Con las ordenes detectadas, vamos entrenar el set y extraer las predicciones. Estas 2 funciones extraen la diferenciación que serán usadas en la próxima función donde extraeremos las predicciones.

```

1 #This function generate the difference to be used in the next step
2 def difference(train,interval = 1):
3     diff = list()
4     for i in range (interval, len(train)):
5         value = train[i] - train[i - interval]
6         diff.append(value)
7     return diff
8
9 #This function will give the original data
10 def inverse_difference(history,predict,interval=-1):
11     return predict + history[-interval]
```

Figura 26: Función de diferenciación.

En este bloque creamos el histórico para almacenar la información, la lista de predicciones, entrenamos el modelo y extraemos las predicciones utilizando la diferenciación inversa. Como ya hemos aplicado la función de diferenciación no aplicaremos en este código el orden 1, para que no sobre diferenciar los resultados.

```

1 #we are going to create a history to stock information
2 history = [x for x in train]
3
4 #List to stock predictions
5 predictions = list()
6
7 #Fit the set
8 for t in range(len(test)):
9
10    #Differnce the data
11    period = 4 #Our dataset has 4 years, so we are working with 4
12    diff = difference(history,period)
13    #Create a model with history set
14    model=ARIMA(diff, order=(2,0,0))#difference is 0, because we made the difference previously
15    #Training the model
16    model_fit = model.fit(disp=0)
17    #Forecast predicted by model
18    fc = model_fit.forecast()[0]
19    #Forecat receive the inverse value
20    fc = inverse_difference(history, fc, period)
21    #Adding the forecat value in the forecast list
22    predictions.append(fc)
23    #Current value to check with the test set
24    current_value = test[t]
25    #Adding the current value in the history
26    history.append(current_value)
27
28
```

Figura 27: Código de entrenamiento y predicción.

	id	forecast
order_date		
2021-01-31	1216	1195
2021-02-28	1238	1247
2021-03-31	1181	1277
2021-04-30	956	1234
2021-05-31	1007	1065
2021-06-30	1088	1141
2021-07-31	1103	1108
2021-08-31	1008	921
2021-09-30	1056	1046
2021-10-31	1155	1116
2021-11-30	1023	1143
2021-12-31	933	959

Figura 27.1: Resultado obtenido del bloque anterior.

Con una simples mirada, podemos asumir que el entreno comparado al set de prueba está adecuado.

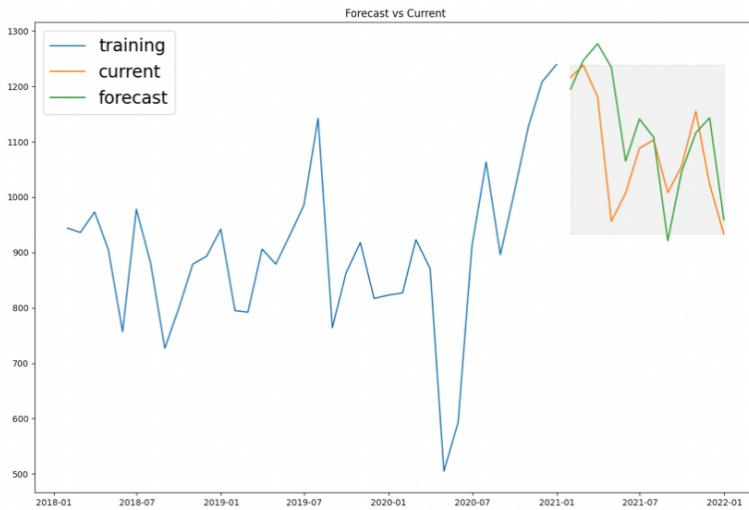


Figura 27.2: Grafico que demuestra los valores de las predicciones comparadas con los valores del test.

5- Obtener valoración del modelo.

Los siguientes KPIs, tienen resultados bastante prometedores, si analizamos el MAPE, el porcentaje de error es de apenas 6,5%, lo que en una empresa, es bastante tolerable.

```

1 def kpi(pred, test):
2     dif = pred - test
3
4     #Mean Percentage Error (MPE)
5     mpe = np.mean((dif)/test)
6
7     #Mean Absolute Percentage Error (MAPE)
8     mape = np.mean(np.abs(dif)/np.abs(test))
9
10    #Mean Absolute Error (MAE)
11    mae = np.mean(np.abs(dif))
12
13    #Root Mean Square Error(RMSE)
14    rmse = np.sqrt(mean_squared_error(test,predictions))
15
16    #Correlation
17    corr = np.corrcoef(pred, test)[0,1]
18
19    #MinMax
20    mins = np.amin(np.hstack([pred[:,None],
21                             test[:,None]]), axis=1)
22    maxs = np.amax(np.hstack([pred[:,None],
23                             test[:,None]]), axis=1)
24    minmax = 1 - np.mean(mins/maxs)
25
26    #Autocorrelation Function (from statsmodel)
27    acf1 = acf(dif)[1]
28
29
30
31    return({'mpe':mpe, 'mae': mae,
32           'mape': mape, 'rmse':rmse, 'acf1':acf1,
33           'corr':corr, 'minmax':minmax})
34    fc, se, conf = model_fit.forecast(12, alpha=0.05)
35

```

```
1 kpi(results.forecast, results.id)
```

```
{'mpe': 0.04071240998641649,
'mae': 66.83333333333333,
'mape': 0.06518160855100981,
'rmse': 99.09042563050139,
'acf1': 0.2539435981819961,
'corr': 0.6054045152767572,
'minmax': 0.05766783334507031}
```

Figura 28: KPIs de errores del modelo.

5- Analizar los valores residuales como forma de sustentar los resultados obtenidos previamente.

Como forma de sostener los resultados anteriores, analizamos los valores residuales en forma de detectar patrones, así saber si el modelo está recogiendo bien los parámetros.

```
0
count    43.000000
mean     2.142892
std      149.538662
min     -350.963219
25%     -87.027888
50%      11.962014
75%     102.847094
max     277.173386
```

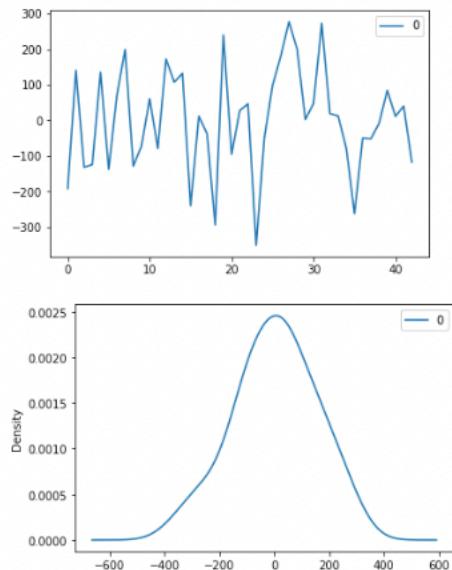


Figura 29: Descripción y visualización de los valores residuales.

Como se observa que no hay un padrón en los residuos, pues son bastante aleatorios, podemos asumir que el modelo funciona.

5.-Preparar el modelo para la producción.

Ahora que hemos testeado el modelo con los mejores parámetros detectados, pasaremos al modelo final para pasar a la producción, que a principios será una API en Flask donde se consulta el ID del código y la API te devuelve las fechas y cantidades referentes a los próximos 12 meses.

El ID debe contener algunos parámetros, como:

- Haber sido utilizado en los últimos 12 meses;
- El porcentaje de valores nulos, no puede pasar de 9% de los datos, pues significa que los consumos son demasiado esporádicos para que determinemos un valor.
- Hay que tener al menos 12 datos de registro para que se pueda hacer la predicción.

Si no lo cumple con lo requisitos anteriores el usuario tiene la opción de mirar los históricos.

```
1 while True:
2     answeryes = ['y', 'Y', 'Yes', 'yes', 'yep', 'YES', 'YE', 'ye', 'YE']
3     consult=input('Which item would you like to consult?\n')
4     c = str.upper(consult)
5     if c in (df.id.unique()):
6         data = pd.read_excel(f'/Users/talitacoll/The Bridge/MMP/tempdatam/{consult}.xlsx').set_index('order_dat
7         obdate = (datetime.now() - timedelta(days=365))
8         frecuence_min = (data[(data.id <1)].value_counts()) / len(data)
9         if len(data) < 12:
10             print('No information enough to model a forecast. Would like to see the historical?')
11             consult2=input()
12             if consult2 in answeryes:
13                 print(data[data.id>0])
14             else:
15                 print('thanks')
16         elif frecuence_min[0] > 0.09:
17             print(f'Item {c} has a exoporadical frecuence, forecast should be inputed mannually.\n'
18             'Would like to see the historical?')
19             consult2=input()
20             if consult2 in answeryes:
21                 print(data[data.id>0])
22             else:
23                 print('thanks')
24         elif max(data.index) < obdate:
25             print(f'Item {c} has not been used on last 12 months, it must be obsolete, please check replacement
26
27         else:
28             print(f'Item {c} is in process, please wait.')
29             break
30     else:
31         print(f'Item {c} is not available, please check the code before consult.')
```

Figura 30: Input inicial con puntos de control para la consulta.

A pesar de los testes anteriores en el momento de la producción, los subsets fragmentados por IDs, cuyas longitudes son cortas, el modelo no funciona.

Demonstración con un ID de alta frecuencia.

Hemos consolidado las funciones en una sola para que ejecute en un solo bloque el modelo:

- Train/test Split;
- Adfuller test;
- Diferenciación Integrada;
- Detección de AR y MA;
- Entreno del Modelo;
- Predicción asignada;
- Plot.

```

1 def model_arima(data):
2     tsize = int(len(data)*.75)
3     train = data.iloc[:tsize]
4     test = data.iloc[tsize:]
5
6     #Detecting estationarity
7     result = adfuller(train)
8     if result[1]<=0.5:
9         d=0
10    else:
11        for i in range(1,3):
12            diff = data.diff(i)[1:]
13            if adfuller(data.diff().dropna())[1]<0.05:
14                d=i
15            else:
16                adfuller(data.diff().diff().dropna())[1]<0.05:
17                d=2
18
19     #Maximum iterations will be the set divided in 12, that means, if the lengt is 36, the system will do 3 iterations
20     p_max = int(len(data) / 12)
21     q_max = int(len(data) / 12)
22     matrix_bic = []
23     for p in range(p_max +1):
24         aux = []
25         for q in range(q_max+1):
26             try:
27                 model = ARIMA(data, order=(p,d,q)).fit(disp=False)
28             except:
29                 aux.append(None)
30         matrix_bic.append(aux)
31     matrix_bic = pd.DataFrame(matrix_bic)
32
33     #Assigning p and q to matrix in order to find the best(lowest) relation.
34     p,q = matrix_bic.stack().astype('float64').idxmin()
35     print(matrix_bic)
36     print("ARIMA Orders:",p,d,q)
37     #Plot a heatmap to identify the best set of orders.
38     #The darkest red section in the graph, the lowest is the order and the best is the correlation
39     fig, ax = plt.subplots(figsize=(10, 8))
40     ax = sns.heatmap(matrix_bic,
41                       mask=matrix_bic.isnull(),
42                       ax=ax,
43                       annot=True,
44                       fmt=".2f",
45                       );
46     ax.set_title("BIC");
47
48     model=ARIMA(train, order=(0,0,1))
49     mfitted = model.fit(disp=0)
50     fc, se, conf = mfitted.forecast(12, alpha=0.95)
51     fc_series = pd.Series(fc, index=test.index)
52     pred = fc_series
53     results = pd.DataFrame(test)
54
55     results = pd.DataFrame(test)
56     results['forecast']=x for x in list(pred)
57
58     print(results,(f'RMSE:{mean_squared_error(results.forecast,results.id)},MAPE:{mean_absolute_percentage_error(results.forecast,results.id)})')

```

Figura 31: Función completa de ARIMA.

En la siguiente imagen podemos observar un ID con frecuencia de 48 sobre 48, aunque que el MAPE no sea tan alto como en los testes, aun así, los KPIs son aceptables.

	0	1	2	3	4
0	279.699161	275.406688	278.476488	281.629973	288.184444
1	275.882152	278.834636	281.317142	284.803399	NaN
2	278.226630	281.880226	285.397044	285.383773	292.529051
3	281.680556	284.958330	288.820832	292.683090	293.046341
4	285.361014	288.819864	291.975794	293.028851	NaN

ARIMA Orders: 0 0 1
order_date
id forecast
2021-01-31 15 16.555496
2021-02-28 12 16.902831
2021-03-31 17 16.902831
2021-04-30 16 16.902831
2021-05-31 16 16.902831
2021-06-30 14 16.902831
2021-07-31 22 16.902831
2021-08-31 20 16.902831
2021-09-30 18 16.902831
2021-10-31 14 16.902831
2021-11-30 15 16.902831
2021-12-31 23 16.902831 RMSE:10.21028566243596,MAPE:0.15525486711855



Figura 31.1: Resultados del modelo para gran subset de gran volumen con periodo mensual.

La alternativa que hemos planteado en separar las series en bloques semanales, para que la frecuencia sea más distribuida, sin embargo, esta solución no es adecuada para los subsets de gran frecuencia, ya que los resultados no son tan altos cuando en

```

0 863.256172 867.794896 872.523022 875.737703 880.984215
1 867.876779 869.937607 875.258070 880.503059 885.288678
2 872.755803 875.256644 879.157493 883.186750 887.369283
3 876.169996 879.072740 882.214581 884.756579 890.010809
4 881.188653 884.370251 887.962838 890.004218 899.819774
ARIMA Orders: 0 0 0
RMSE:3.276500837828752,MAPE:0.3535898221431537

```

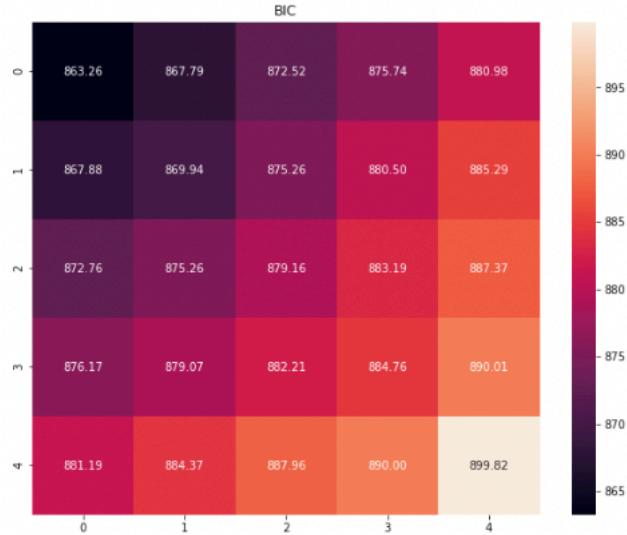


Figura 31.2: Resultados del modelo para gran subset de gran volumen con períodos semanales.

El error del modelo mensual es de 15.5%, sin embargo, el modelo semanal, tiene error de 35,4%, lo que hace cuestionable el modelo semanal.

Otra alternativa planteada, ha sido trabajar con la Suavización Exponencial y nos gustaría aplicar el método Holt completo, pero los datos no son suficiente para aplicarlos, así que hemos testeado apenas con el método de suavización simples y la tendencia lineal de Holt, pero los demás parámetros, el modelo no aceptaba debido que, al hacer el entrenamiento lineal, los valores pasaban a ser negativos y no lo aceptaba.

Hemos testeado con diversos grados del nivel de suavización y tendencia, pero todos con resultados insatisfactorios.

```

1 def Simple_Exponential_Smoothing(data):
2     #train for Smoothing
3     tsize = int(len(data)*.80)
4     train = data.iloc[:tsize]
5     test = data.iloc[tsize:]
6
7     def smoothing(data):
8         #Simple Exponential Smoothing with alpha optimized.
9         fitses = SimpleExponentialSmoothing(data, initialization_method='estimated').fit(optimized=True)
10        fcastses = fitses.forecast(len(test)).rename(r"\alpha=%s" % fitses.model.params["smoothing_level"])
11        fitholt = Holt(data, initialization_method="estimated").fit(
12            smoothing_level=0.9, smoothing_trend=0.2, optimized=False)
13        fcastholt = fitholt.forecast(len(test)).rename("Holt's linear trend")
14
15        plt.figure(figsize=(12, 8))
16        plt.plot(data, marker='o', color='black', linestyle = 'None')
17        plt.plot(fitests.fittedvalues, color='b')
18        (line1,) = plt.plot(fcastses, color='b')
19        plt.plot(fitholt.fittedvalues, color='r')
20        (line2,) = plt.plot(fcastholt, color='r')
21
22
23
24        plt.legend([line1,line2], [fcastses.name,fcastholt.name]);
25
26        #print(fcastses)
27        return (fcastses)
28
29
30        #print(mean_squared_error(smoothing(train.values),test))
31        print(mean_absolute_percentage_error(smoothing(train),test))
32

```

Figura 32: Función de Suavización de Exponencial.

1.716049388271269

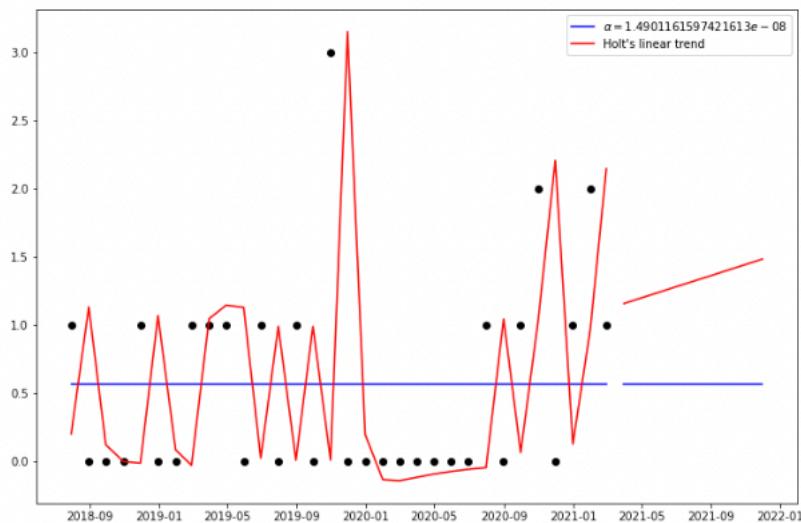


Figura 32.1: Resultado de la Función de Suavización de Exponencial para proyección en meses.

Aplicando el método de Suavización en periodos semanales, el resultado es todavía peor.

2.1176471567849737

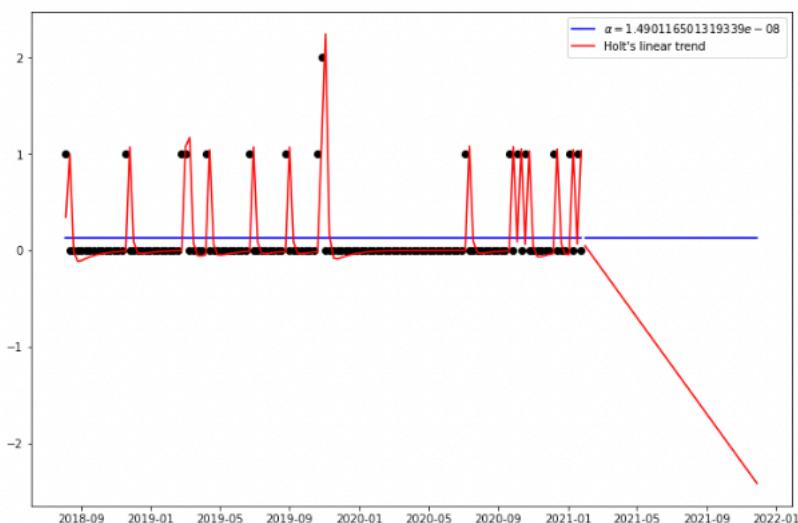


Figura 32.2: Resultado de la Función de Suavización de Exponencial para proyección en meses.

El MAPE por encima de 100%, nos demuestra que definitivamente la Suavización Exponencial Simple y el método Holt solamente aplicando solamente la tendencia lineal, no es recomendable.

6.-Consideraciones finales.

Aunque ARIMA sea altamente recomendable para predecir series temporales, para este conjunto de datos en concreto, hay que ser mejorado.

Solamente funciona en casos cuyas frecuencias son bastante consideradas, sea esté cerca de las 48 frecuencias.

Para IDs con baja frecuencia de requerimiento, es necesario plantear otros métodos de regresión.

El método de Suavización Exponencial, puede ser trabajado para obtener mejores resultados.

La API no ha sido desarrollada, una vez que los resultados no son satisfactorios.

7. – Bibliografía y consultas de apoyo.

Vandeput, Nicolas. (2021).*Data Science For Supply Chain Forecasting*. Berlin, De Gruyter

<https://www.statsmodels.org/stable/index.html>

<https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/>

<https://towardsdatascience.com/forecasting-in-a-spare-parts-business-45437cadf195>

Repositorio GitHub:

https://github.com/talitacardenas/MMP_Final.git