



LICENCIATURA EN CIENCIA DE DATOS

**Patrones clave, predicción de demanda,
y análisis geoespacial en el transporte público
de Mar del Plata**



Introducción

El transporte público en mi ciudad es una pieza fundamental en la movilidad urbana, facilitando el acceso a servicios esenciales y promoviendo la sostenibilidad ambiental, siendo el colectivo el único medio de transporte público y accesible. La presente investigación se basa en cuatro enfoques clave: lograr generar un dataset consistente para su análisis, la identificación de patrones de uso y contraste con algunas variables externas como el clima, la predicción de la demanda por día y línea, y el análisis geoespacial del transporte público donde se pueda apreciar claramente la distribución de la demanda. Hago uso de datos públicos y oficiales, proporcionados por el propio municipio de General Pueyrredón y el Ministerio de Transporte de la Nación, y los analizo con todo lo aprendido en la carrera.

Índice

Objetivos.....	3
Metodología.....	3
1 - Origen de los datos.....	4
Generando un dataset consistente.....	4
Exploración previa.....	4
Obteniendo los datos faltantes.....	6
Imputación de valores.....	7
Conclusiones.....	9
2 - Análisis exploratorio.....	10
Análisis univariado.....	13
Análisis bivariado.....	15
Generando nuevas características.....	16
¿Afectan los feriados a la demanda?.....	23
¿Afecta la lluvia, el viento o la temperatura a la demanda?.....	24
Otras gráficas de valor.....	28
Análisis de demanda por línea.....	31
3 - Generando predicciones.....	32
Modelo.....	35
Entrenamiento.....	36
Generando 2024.....	38
Conclusiones.....	43
4 - Análisis espacial de demanda.....	44
Análisis y conclusiones.....	54
Posibles implicaciones y recomendaciones.....	55
Investigación adicional.....	55
Evolución temporal de la demanda.....	58
Conclusión.....	59
Próximos pasos.....	59
Referencias.....	60

Objetivos

Generar un dataset consistente: Disponer de un único dataset en buenas condiciones para poder hacer un análisis fiable sobre la demanda del transporte público.

Identificación de patrones clave: Analizar los datos históricos de uso del transporte público para identificar patrones recurrentes y factores estacionales que afectan la demanda, así como también factores externos como el clima.

Predicción de la demanda: Desarrollar un modelo predictivo robusto que permita anticipar la demanda diaria de transporte, utilizando técnicas de machine learning. El modelo elegido para trabajar es XGBoost.

Análisis geoespacial: Evaluar la demanda de transporte en diferentes zonas de la ciudad mediante el análisis geoespacial, identificando áreas de alta y baja demanda y generando gráficas claras que aporten insights de valor al análisis.

Metodología

Recopilación y preparación de datos: Se utilizaron datos de fuentes oficiales que incluyen detalles de consumo diario de transporte por línea. Los datos fueron limpiados y unificados en un único dataset para su análisis.

Imputación de datos faltantes: Se emplearon técnicas de imputación basadas en la proporción de consumo de años posteriores para llenar los datos faltantes, asegurando la coherencia y completitud del dataset.

Análisis exploratorio y de patrones: Se realizaron análisis univariados y bivariados para comprender la distribución de los datos y la relación entre diferentes variables y la demanda de transporte.

Generación de nuevas características: Se crearon nuevas variables a partir de transformaciones temporales y retrasos para mejorar la capacidad predictiva de los modelos.

Modelado predictivo: Se utilizó XGBoost para predecir la demanda diaria de transporte, evaluando su rendimiento mediante métricas como el error cuadrático medio (MSE) y el coeficiente de determinación (R^2).

Análisis geoespacial: Se analizaron los datos de demanda en un contexto geoespacial, utilizando mapas, clusterización y otras gráficas para identificar patrones de uso en diferentes zonas de la ciudad.

1 - Origen de los datos

Este conjunto de datos proviene de fuentes oficiales proporcionadas por el municipio de General Pueyrredón y el Ministerio de Transporte de la Nación, así como de los datos abiertos de la Nación Argentina. Los datos han sido recopilados de manera oficial y se utilizan con fines de análisis y evaluación. Los datos presentados aquí cumplen con los estándares de Protección de la Privacidad y Protección de la Propiedad Intelectual. No contienen información personal ni sensible de ningún tipo.

Los datos recopilados se presentan de la siguiente manera:

```
df_2020_pre = pd.read_csv("Datasets/sube-mgp-2020.csv")
df_2021_pre = pd.read_csv("Datasets/sube-mgp-2021.csv")
df_2022_pre = pd.read_csv("Datasets/sube-mgp-2022.csv", sep=";") # Este *.csv no esta separado por comas
df_2023_pre = pd.read_csv("Datasets/sube-mgp-2023.csv", sep=";") # Este *.csv no esta separado por comas
df_2024_pre = pd.read_csv("Datasets/sube-mgp-2024.csv", sep=";") # Este *.csv no esta separado por comas y devuelve type-warning
```

Aquí encontramos el detalle de consumo de transporte diario por cada línea de transporte.

Generando un dataset consistente

A fin de lograr un único dataset se fue examinando cada archivo, aquí se presentaron algunos detalles al menos llamativos:

El dataset correspondiente al año 2020, a primeras, contenía toda la información.
El dataset correspondiente al año 2021, contenía información hasta el 28 de febrero.
El dataset correspondiente al año 2022, a primeras, contenía toda la información.
El dataset correspondiente al año 2023, a primeras, contenía toda la información.
El dataset correspondiente al año 2024, contenía información hasta el 3 de febrero y una innumerable cantidad de filas vacías.

Luego de eliminar las filas vacías, procedí a unir los datasets y hacer una breve exploración para evitar errores y redundancias.

Exploración previa

Lo primero que observo son columnas redundantes, osea datos que se repiten durante todo el dataset y no proveen ningún tipo de información para nuestro análisis, estas eran las columnas:

['AMBA', 'TIPO_TRANSPORTE', 'JURISDICCION', 'PROVINCIA', 'MUNICIPIO']

Se eliminan estas columnas y comenzamos a trabajar en el formato fecha, donde se decide separar la fechas por **['DAY', 'MONTH', 'YEAR']** además de la columna **['DATE']**, para tener un mayor control a la hora de iterar sobre el mismo, quedando el dataset resultante se presenta de esta manera:

	DATE	DAY	MONTH	YEAR	NOMBRE_EMPRESA	LINEA	CANTIDAD	DATO_PRELIMINAR
0	2020-01-01	1	1	2020	EMPRESA BATAN S.A.	BS_AS_LINEA_715M	2154.0	NO
1	2020-01-01	1	1	2020	EMPRESA DE TRANSPORTE PERALTA RAMOS SACI	BS_AS_LINEA_512	1889.0	NO
2	2020-01-01	1	1	2020	TRANSPORTES 25 DE MAYO SRL	BSAS_LINEA_501	315.0	NO
3	2020-01-01	1	1	2020	TRANSPORTES 25 DE MAYO SRL	BSAS_LINEA_521	2729.0	NO
4	2020-01-01	1	1	2020	TRANSPORTES 25 DE MAYO SRL	BSAS_LINEA_522	4010.0	NO

Aquí se decide hacer una búsqueda de valores únicos de la variable **LINEA** y codificar estos datos en su representación entera, pasarlal a números. Utilizando la librería **re** en instantes el resultado es el presentado:

	LINEA	LINE_ID
2	BSAS_LINEA_501	501
3	BSAS_LINEA_521	521
4	BSAS_LINEA_522	522
5	BSAS_LINEA_523	523
6	BSAS_LINEA_525	525
7	BSAS_LINEA_531	531
8	BSAS_LINEA_532	532
9	BSAS_LINEA_533	533
10	BSAS_LINEA_541	541
11	BSAS_LINEA_543	543
12	BSAS_LINEA_551	551
13	BSAS_LINEA_552	552
14	BSAS_LINEA_553	553
15	BSAS_LINEA_554	554
16	BSAS_LINEA_555	555
17	BSAS_LINEA_562	562
18	BSAS_LINEA_563	563
19	BSAS_LINEA_571	571

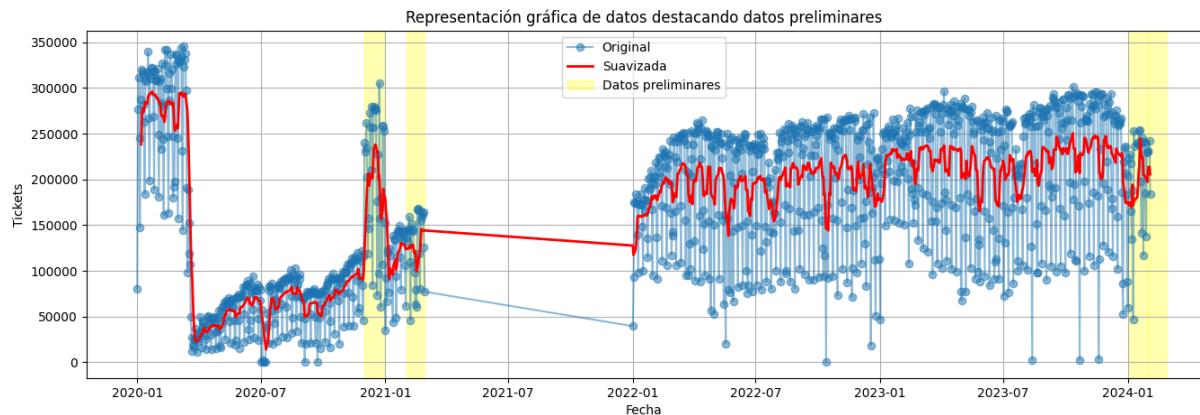
Posterior a esto, se decide contabilizar la variable **DATO_PRELIMINAR**, esta en una columna binaria que, entiendo, habla de la confianza del dato allí expuesto. Se encontró que:

2020, tiene en su mes 12 un total de 833 datos preliminares.

2021, tiene en su mes 2, un total de 753 datos preliminares, recordemos que esto representa el 50% de datos para ese año hasta el momento.

2024, tiene en su mes 1, un total de 783 datos preliminares, y en su mes 2, un total de 56 datos preliminares, recordemos que esto afecta casi a la tonalidad de los datos para dicho año.

Hagamos un análisis visual de cómo están representados los datos:



Aquí se distingue claramente la falta de datos para ciertos períodos, y podemos anticipar el efecto de la pandemia en el análisis, así mismo observamos grandes aumentos sobre el fin de 2020.

Se genera aquí un conteo de los días para los que no se dispone información por año:

2020, sólo presenta 1 día sin información.

2021, presenta 309 días sin información, casi en su totalidad.

2022, solo presenta 2 días sin información.

2023, no presenta días sin información.

2024, presenta 32 días sin información, recordemos este posee datos hasta el 28 de febrero.

Obteniendo los datos faltantes

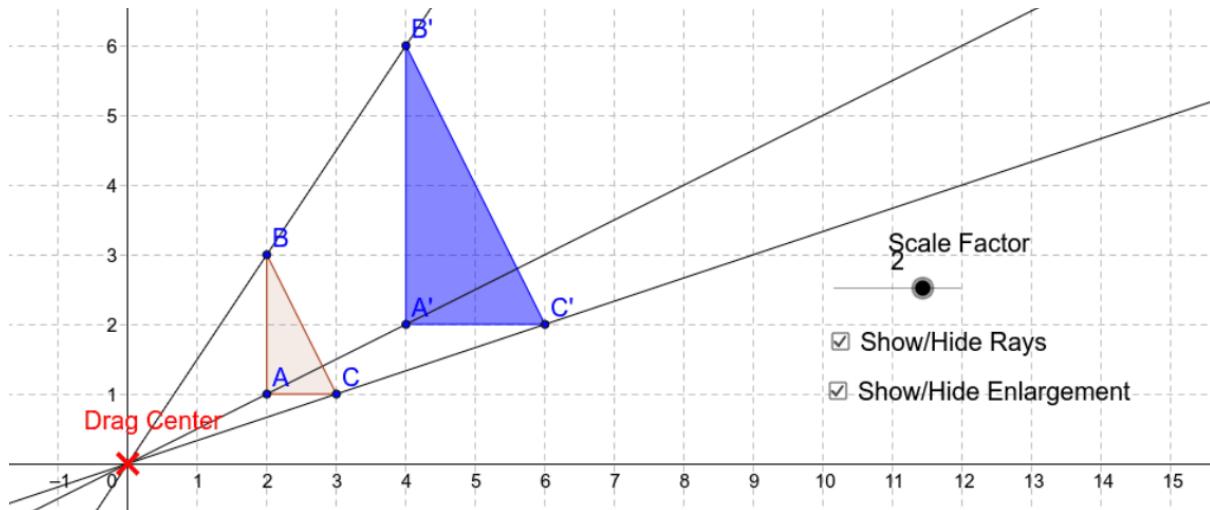
Luego de buscar en los sitios oficiales donde se ofrecía el dataset para su descarga, encontré los datos mensualizados para 2021, lo que supondría una mejora sustancial en la densidad de datos.

	MES	BOLETO PLANO	RESOLUCION 651	GRATUITOS SUBE
0	Enero	1.742.578	1.206.954	176.273
1	Febrero	2.004.008	1.399.245	251.362
2	Marzo	2.472.789	1.711.638	362.311
3	Abril	2.146.961	1.494.908	413.214
4	Mayo	1.510.330	1.072.958	326.536
5	Junio	477.738	2.161.217	296.690
6	Julio	1.866.498	1.424.752	390.185
7	Agosto	2.064.402	1.625.783	565.619
8	Septiembre	2.163.871	1.693.894	661.565
9	Octubre	2.389.732	1.951.151	765.983
10	Noviembre	2.496.485	2.048.112	845.687
11	Diciembre	2.607.659	2.192.850	692.694

Esta valiosa información, no solo supondría poder mejorar la densidad de nuestra información, sino que también nos da información y estadística del desglose del tipo de boleto que se consume.

Imputación de valores

Ahora la pregunta seria, como podemos imputar estos valores? Bueno, acá hay se pueden manejar diversas variantes y abordajes, para mí lo más justo fue establecer un factor de escala por cada línea de transporte y mantener la proporción de consumo por línea de 2022 utilizando los valores proporcionados para 2021, esto teniendo en cuenta que 2022 es un año completamente fuera del periodo de pandemia.



El proceso es relativamente simple, primero filtramos los datos del año 2022 para trabajar únicamente con ese año. Calculamos el promedio diario de tickets para cada línea y mes. Sumamos estos promedios diarios por mes para obtener el total mensual de los promedios en 2022. A continuación, unimos estos datos con los datos de 2021, tanto los promedios diarios como los totales mensuales de 2022. Utilizando esta información, calculamos los datos diarios corregidos para 2021.

```
•••
# Filtramos los datos de 2022
df_2022 = df[df['YEAR'] == 2022].copy()

# Calculamos el promedio diario de tickets por cada LINE_ID y mes en 2022
daily_avg_2022 = df_2022.groupby(['MONTH', 'LINE_ID', 'DAY'])['TICKETS'].mean().reset_index(name='DAILY_AVG_TICKETS')

# Calculamos la suma total de los promedios diarios por mes en 2022
monthly_avg_2022 = daily_avg_2022.groupby('MONTH')[['DAILY_AVG_TICKETS']].sum().reset_index(name='MONTHLY_SUM_AVG_TICKETS')

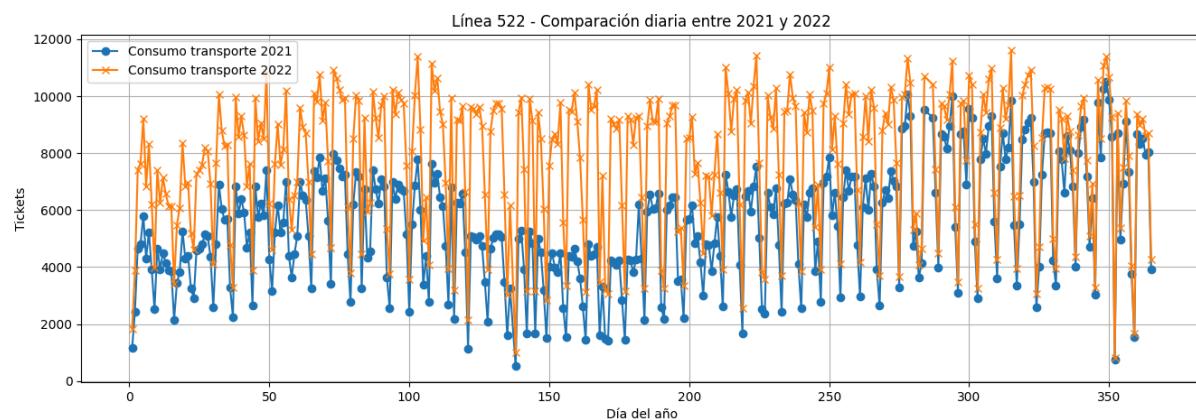
# Unimos los datos de 2021 con los promedios diarios y la suma total de los promedios diarios de 2022
data_2021_with_avg = pd.merge(monthly_data_2021, daily_avg_2022, on='MONTH')
data_2021_with_avg = pd.merge(data_2021_with_avg, monthly_avg_2022, on='MONTH', suffixes=('', '_2022'))

# Calculamos los datos diarios corregidos de 2021
data_2021_with_avg['DAILY_TICKETS_2021'] = data_2021_with_avg['TICKETS'] * (data_2021_with_avg['DAILY_AVG_TICKETS'] / data_2021_with_avg['MONTHLY_SUM_AVG_TICKETS'])

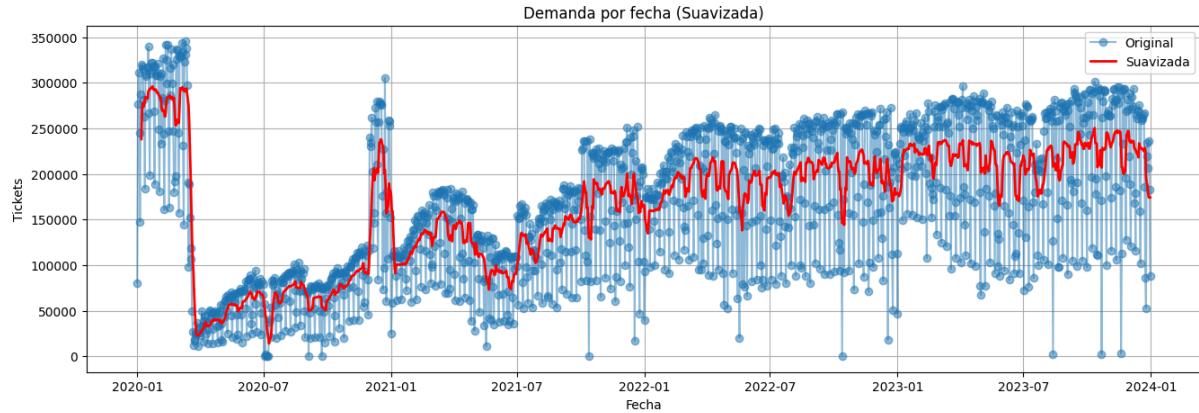
# Creamos un DataFrame con los datos diarios corregidos de 2021
data_2021_with_avg['DATE'] = 2021
data_2021_with_avg['DATE'] = pd.to_datetime(data_2021_with_avg['YEAR'].astype(str) + '-' + data_2021_with_avg['MONTH'].astype(str) + '-' + data_2021_with_avg['DAY'].astype(str), format='%Y-%m-%d')

data_2021_daily = data_2021_with_avg[['DATE', 'YEAR', 'MONTH', 'DAY', 'LINE_ID', 'DAILY_TICKETS_2021']].rename(columns={'DAILY_TICKETS_2021': 'TICKETS'})
display(data_2021_daily)
```

Para finalizar, creamos un DataFrame que contenga estos datos diarios corregidos, y ya podemos ver los resultados, aquí el consumo de la línea 522 en ambos años:



Como podemos observar, se mantiene la relación de consumo de transporte en el año imputado. Ahora veamos nuevamente nuestra línea temporal de demanda con los nuevos datos:



Conclusiones

La imputación de datos faltantes en nuestro análisis busca mantener la integridad y la utilidad de nuestro conjunto de datos. A continuación, detallo algunas conclusiones sobre esta metodología y cómo entiendo, beneficia a nuestro análisis:

- Mantenimiento de la estructura estacional: La demanda de transporte presenta una fuerte componente estacional y semanal. Al utilizar las proporciones de consumo de 2022 para imputar datos en 2021, nos aseguramos que estas variaciones estacionales se mantengan coherentes. Esto es evidente en los gráficos proporcionados, donde la línea suavizada sigue patrones claros y recurrentes, indicando que las imputaciones respetan la estructura temporal y estacional de la demanda.
- Coherencia en la representación de la demanda: La imputación basada en proporciones permite que los totales mensuales imputados en 2021 estén alineados con los totales mensuales oficiales y declarados. Este alineamiento asegura que las tendencias anuales se mantengan precisas, evitando sesgos significativos que podrían surgir de datos faltantes no tratados adecuadamente. Permitiendo así, realizar comparaciones más fiables entre 2021 y 2022.
- Robustez en modelos predictivos: Para cualquier modelo predictivo, la calidad e integridad de los datos de entrenamiento es fundamental. Fortalecer los datos, potencialmente mejora el rendimiento de los modelos predictivos al tener un conjunto de datos más completo y representativo.

2 - Análisis exploratorio

Al finalizar el análisis exploratorio previo, se generó un dataset con el cual trabajar. Este se presenta de la siguiente manera:

	DATE	DAY	MONTH	YEAR	COMPANY	LINE_ID	TICKETS
0	2020-01-01	1	1	2020	25 DE MAYO	501	315
1	2020-01-01	1	1	2020	PERALTA RAMOS	511	11463
2	2020-01-01	1	1	2020	PERALTA RAMOS	512	1889
3	2020-01-01	1	1	2020	25 DE MAYO	521	2729
4	2020-01-01	1	1	2020	25 DE MAYO	522	4010
5	2020-01-01	1	1	2020	25 DE MAYO	523	4791
6	2020-01-01	1	1	2020	25 DE MAYO	525	253
7	2020-01-01	1	1	2020	12 DE OCTUBRE	531	2339
8	2020-01-01	1	1	2020	12 DE OCTUBRE	532	1982
9	2020-01-01	1	1	2020	12 DE OCTUBRE	533	2425

En esta etapa analizamos las dimensiones, evaluamos tipos de datos de cada columna, buscamos valores nulos y duplicados. De los *describes* observamos que en la columna 'TICKETS' el consumo va de 1 a 40.067 repartidos en 39.245 registros. En cuanto a las columnas categóricas, observamos que las líneas de transporte son 28, y están operadas por 5 empresas.

Nuevamente buscamos días sin información y observamos:

YEAR	MONTH	DAY
2020	7	[7]
2021	5	[17]
	10	[12]
2022	5	[17]
	10	[12]

7 de julio de 2020: Paro de transporte
17 de mayo de 2021: (Herencia 2022)
12 de octubre de 2021: (Herencia 2022)
17 de mayo de 2022: Paro de transporte
12 de octubre de 2022: Paro de transporte

Datos provistos por 0223.com.ar

Los datos faltantes de 2022 se heredan en la imputación de datos para 2021. El resto de los días corresponden a paros de transporte en la ciudad.

Posteriormente, buscamos información faltante por cada día, por cada línea de transporte, y este fue el resultado:

LINE_ID	501	511	512	521	522	523	525	531	532	533	541	542	543	551	552	553	554	555	562	563	571	573	581	591	593	715	717	720
YEAR																												
2020	359	360	360	358	358	359	352	360	352	353	358	359	183	359	358	348	358	358	361	358	359	358	360	358	359	364	359	9
2021	361	362	362	362	362	362	362	362	356	356	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	8
2022	361	362	362	362	362	362	362	362	356	356	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	362	8
2023	364	365	365	365	365	366	365	365	365	365	365	365	365	367	365	365	365	365	366	365	366	365	366	365	365	365	365	349

Podemos apreciar que la línea 543 para 2020 no tiene suficientes datos. Por otro lado la línea 720 directamente tiene datos solo de 2023.

Con respecto a la línea 523 la vamos a dejar, ya que seguramente no entrenemos el modelo con 2020, que de por sí es un outlier todo el periodo. Por el lado de la línea 720, al no tener información suficiente, optamos por descartarla del análisis y la predicción.

Para generar estos datos faltantes tenemos varios caminos. Uno podría ser utilizar KNN para buscar los datos cercanos, estimación por árboles, o como vemos a continuación el interpolado de datos:

```

● ● ●

# Convertimos la columna DATE a tipo datetime si no lo está ya
df['DATE'] = pd.to_datetime(df['DATE'])

# Generamos un rango de fechas desde el 1 de enero de 2020 hasta el 31 de diciembre de 2023
full_date_range = pd.date_range(start='2020-01-01', end='2023-12-31')

# Obtenemos los LINE_ID únicos
unique_line_ids = df['LINE_ID'].unique()

# Creamos un DataFrame de referencia con todas las combinaciones de fechas y LINE_ID
reference_df = pd.MultiIndex.from_product([full_date_range, unique_line_ids], names=['DATE', 'LINE_ID']).to_frame(index=False)

# Hacemos un merge con el DataFrame original para identificar los días faltantes
merged_df = reference_df.merge(df, on=['DATE', 'LINE_ID'], how='left')

# Rellenamos los valores de COMPANY hacia adelante para manejar los casos de adquisición, como vamos a ver mas adelante
merged_df['COMPANY'] = merged_df.groupby('LINE_ID')['COMPANY'].fillna(method='ffill')

# Lista para almacenar los DataFrames interpolados
interpolated_dfs = []

# Recorremos el DataFrame resultante buscando los valores de TICKETS faltantes
for line_id in unique_line_ids:
    line_df = merged_df[merged_df['LINE_ID'] == line_id].sort_values(by='DATE')

    # Identificamos los días faltantes antes de la interpolación
    missing_days = line_df[line_df['TICKETS'].isnull()]['DATE']

    # Interpolamos los valores de TICKETS
    line_df['TICKETS'] = line_df['TICKETS'].interpolate(method='linear').round().astype('Int64')

    # Identificamos las filas donde los valores de TICKETS fueron interpolados
    interpolated_days = line_df[line_df['DATE'].isin(missing_days)]

    # Imprimimos los días faltantes
    if not missing_days.empty:
        print(f"Día perdido para LINE_ID: {line_id}:")
        print(missing_days.tolist())

    # Rellenamos las columnas DAY, MONTH, y YEAR
    interpolated_days['DAY'] = interpolated_days['DATE'].dt.day
    interpolated_days['MONTH'] = interpolated_days['DATE'].dt.month
    interpolated_days['YEAR'] = interpolated_days['DATE'].dt.year

    # Agregamos el DataFrame interpolado a la lista
    interpolated_dfs.append(interpolated_days)

# Concatenamos todos los DataFrames interpolados
interpolated_df = pd.concat(interpolated_dfs, ignore_index=True)

# Concatenamos el DataFrame original con el DataFrame interpolado
complete_df = pd.concat([df, interpolated_df], ignore_index=True)

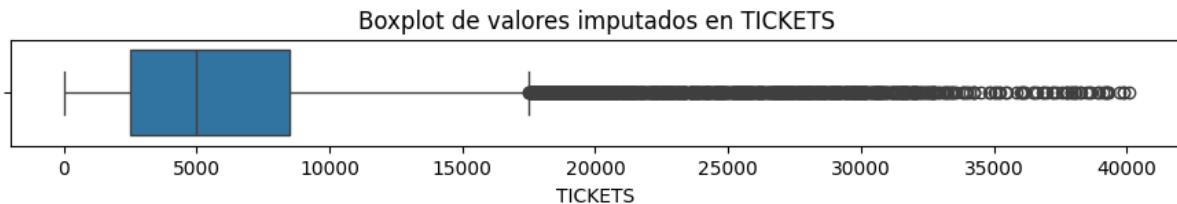
# Ordenamos por DATE y LINE_ID
complete_df = complete_df.sort_values(by=['DATE', 'LINE_ID']).reset_index(drop=True)

# Nos aseguramos que la salida sea entera
complete_df['LINE_ID'] = pd.to_numeric(complete_df['LINE_ID'], errors='coerce').astype('Int64')

```

Luego de extensas pruebas, pude concluir que si bien los datos se pueden generar correctamente, éstos no presentan una mejoría en el modelo de predicción. Incluso, pueden introducir un sesgo al trabajar con datos aproximados por cercanía. La robustez de XGBoost para manejar datos faltantes es uno de los pilares de por qué elegí este modelo.

Ahora llega el turno de evaluar los valores anómalos dentro de la variable objetivo. Aquí pudimos observar:



Además se realizó un desglose por cada línea de transporte para identificar aquellas cuya demanda presenta consumos extraños:

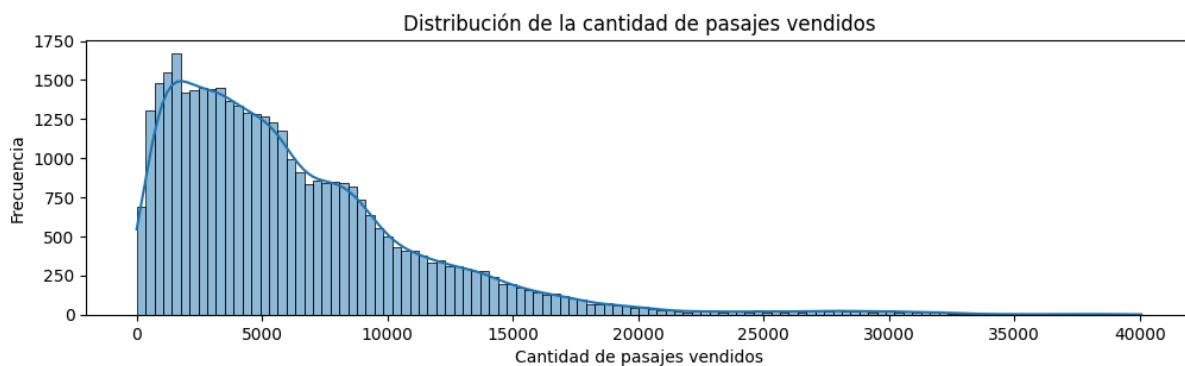
LINE_ID	NUM_OUTLIERS
0	511
1	523
2	531
3	551
4	571

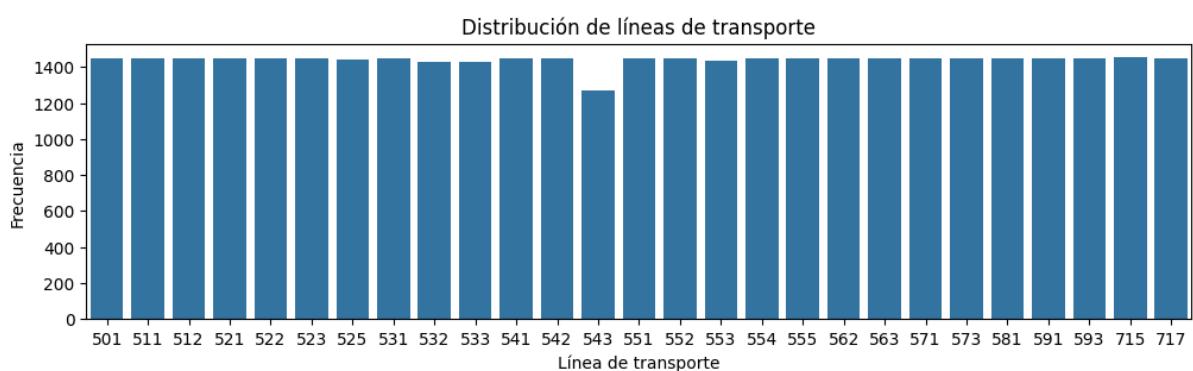
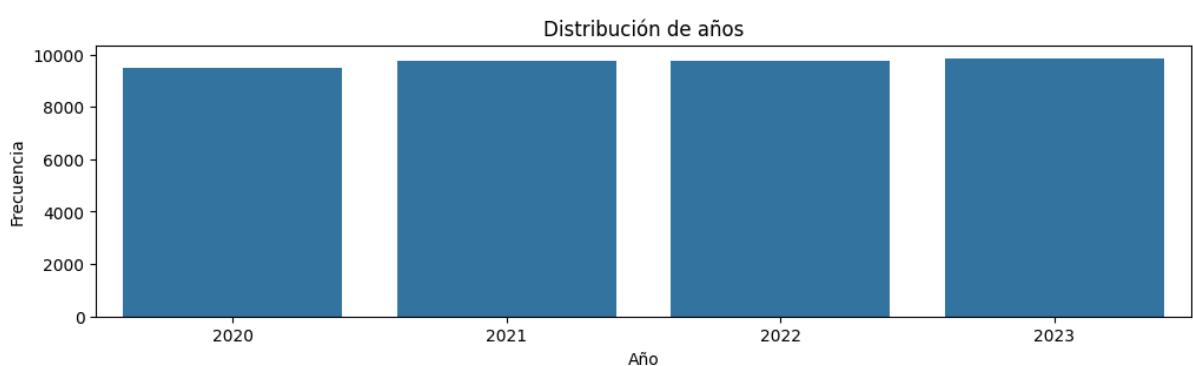
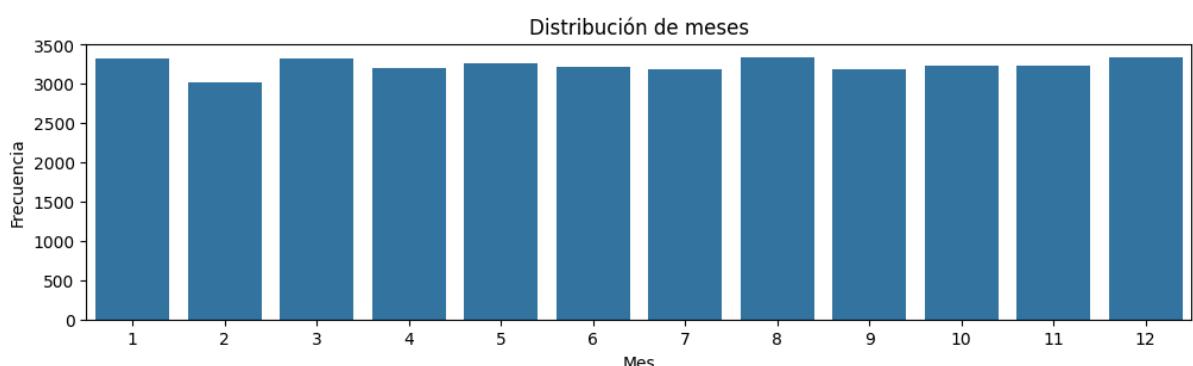
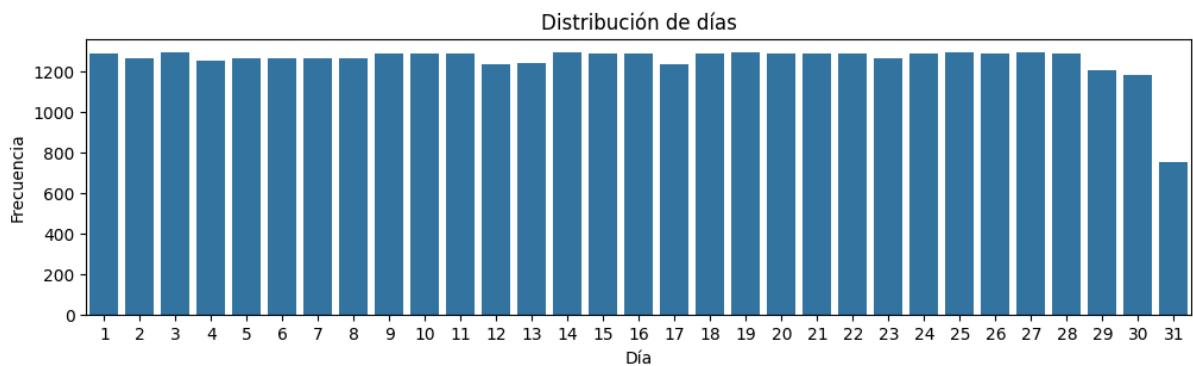
Mayormente se trata de la línea 511, que dispone de varios ramales o recorridos, la más utilizada por turistas y marplatenses en épocas de verano para trasladarse a las playas, además esta línea contiene buses dobles, pudiendo disponer de hasta 60 asientos.

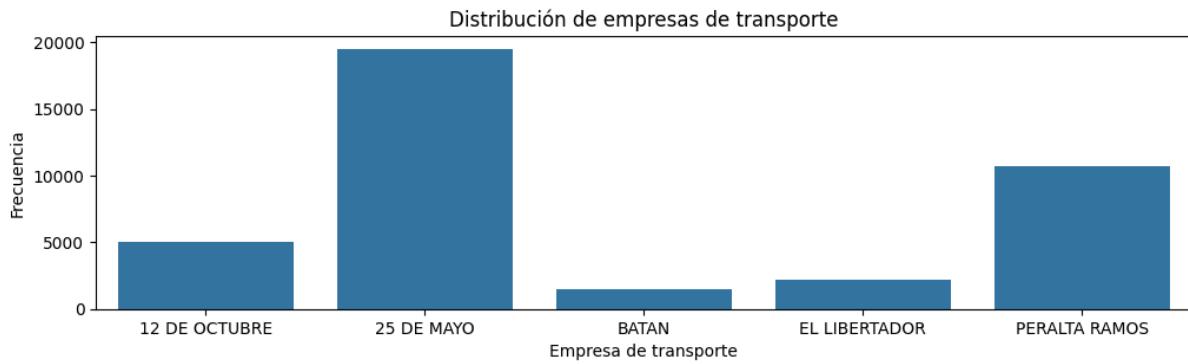
Por el momento los outliers se quedan, ya que confiamos en que son datos reales, y la mayoría corresponden a 2020 que no vamos a incluir en el entrenamiento. Al momento del modelado vamos a evaluar y realizar diferentes pruebas.

Análisis univariado

Aquí se presenta la frecuencia de cada una de las variables:

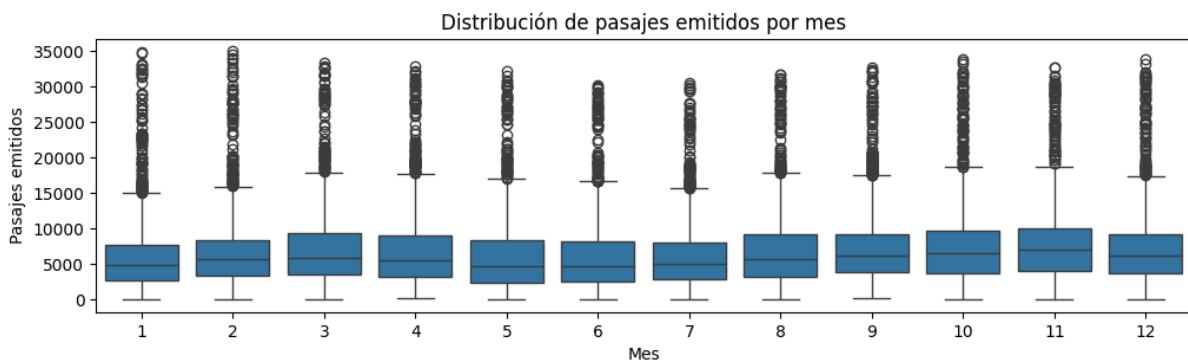
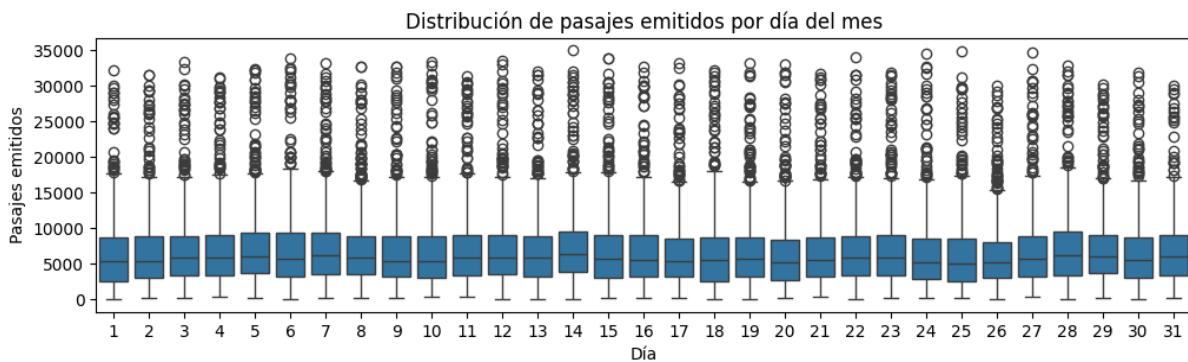


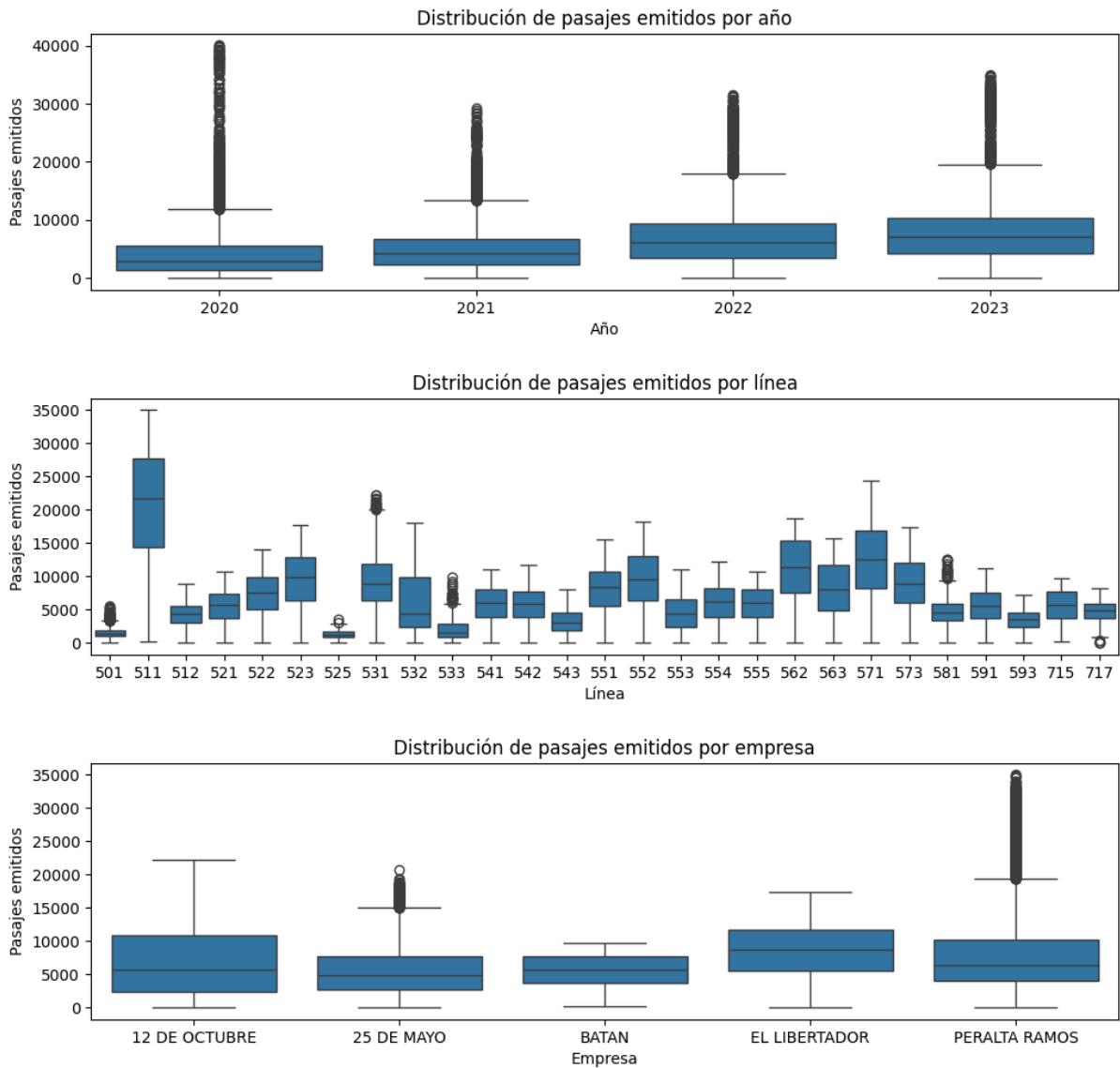




Análisis bivariado

Aquí se contrasta cada una de las variables contra la variable objetivo ‘TICKETS’, y si bien se analizó todo el rango de datos, es mi deseo presentar los gráficos de 2021 a 2023, para tener un panorama más cercano y real con el que trabajar:





Generando nuevas características

Ahora, aprovechando que disponemos de datos temporales, propongo crear una serie de nuevas características mediante la transformación de las temporales disponibles.

Primero vamos a obtener los feriados para todo el periodo:

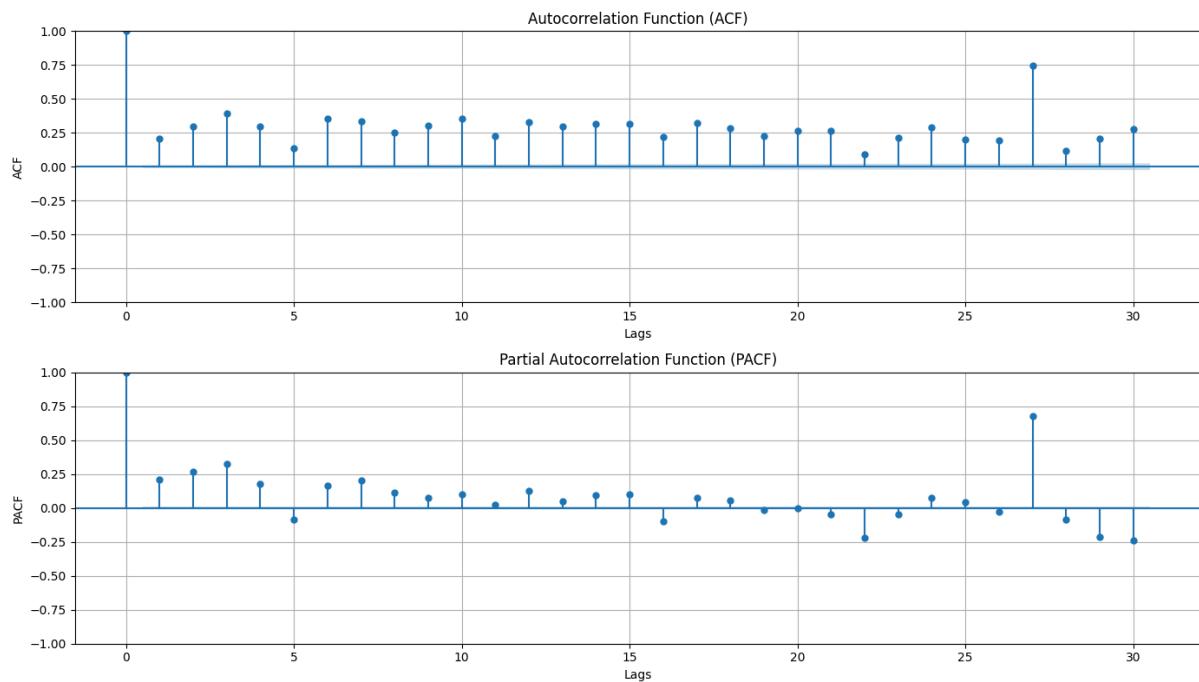
```
import holidays

# Obtenemos feriados
ar_holidays = holidays.Argentina(years=[2020, 2021, 2022, 2023, 2024])
for date, holiday in ar_holidays.items():
    print(f'{date}: {holiday}')

✓ 0.6s

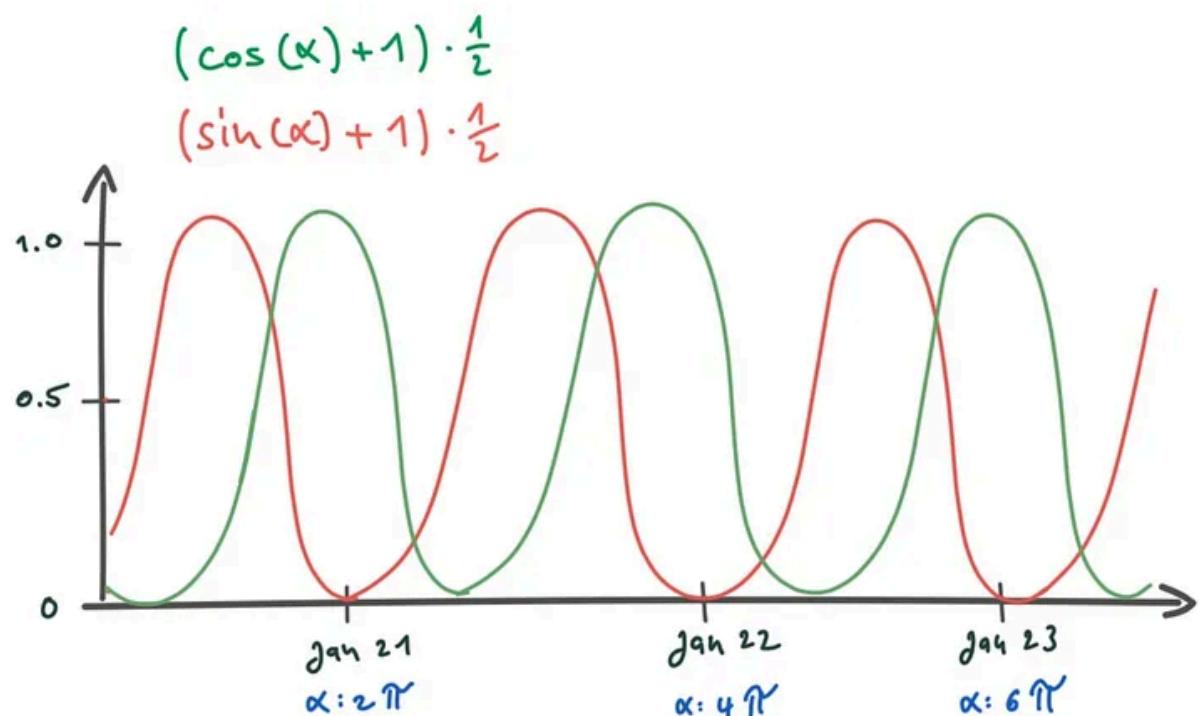
2020-01-01: Año Nuevo
2020-02-24: Día de Carnaval
2020-02-25: Día de Carnaval
2020-03-24: Día Nacional de la Memoria por la Verdad y la Justicia
2020-03-31: Día del Veterano y de los Caídos en la Guerra de Malvinas
2020-04-10: Viernes Santo
2020-05-01: Día del Trabajo
2020-05-25: Día de la Revolución de Mayo
2020-06-20: Paso a la Inmortalidad del General Don Manuel Belgrano
2020-07-09: Día de la Independencia
2020-12-08: Inmaculada Concepción de María
2020-12-25: Navidad
2020-06-15: Paso a la Inmortalidad del General Don Martín Miguel de Güemes (observado)
2020-08-17: Paso a la Inmortalidad del General Don José de San Martín
2020-10-12: Día del Respeto a la Diversidad Cultural
2020-11-23: Día de la Soberanía Nacional (observado)
2020-03-23: Feriado con fines turísticos
2020-07-10: Feriado con fines turísticos
2020-12-07: Feriado con fines turísticos
2021-01-01: Año Nuevo
2021-02-15: Día de Carnaval
2021-02-16: Día de Carnaval
2021-03-24: Día Nacional de la Memoria por la Verdad y la Justicia
2021-04-02: Día del Veterano y de los Caídos en la Guerra de Malvinas; Viernes Santo
2021-05-01: Día del Trabajo
...
...
```

Observamos que los jueves santos no estaban incluidos, los agregamos manualmente. Ahora vamos a buscar los retardos (lags) que posean una estrecha relación con la demanda actual, a fin de apoyar al modelo de predicción. Mediante una función de auto-correlación podemos observar:

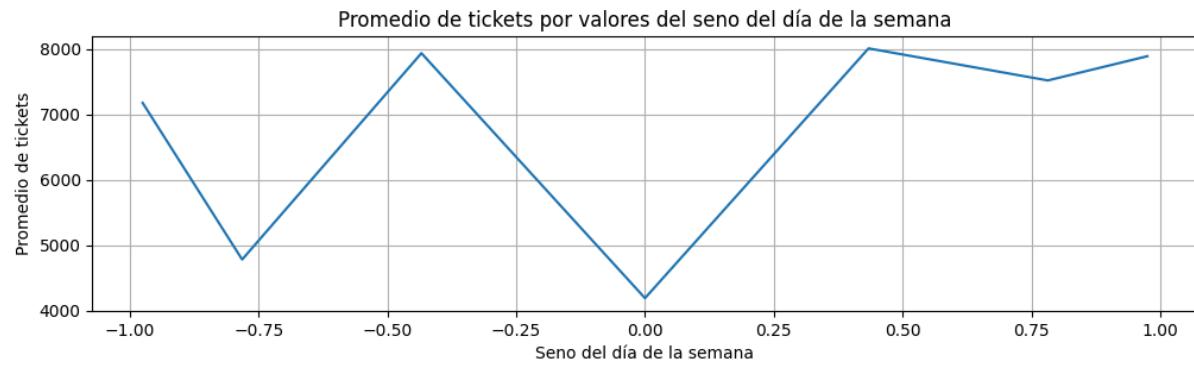
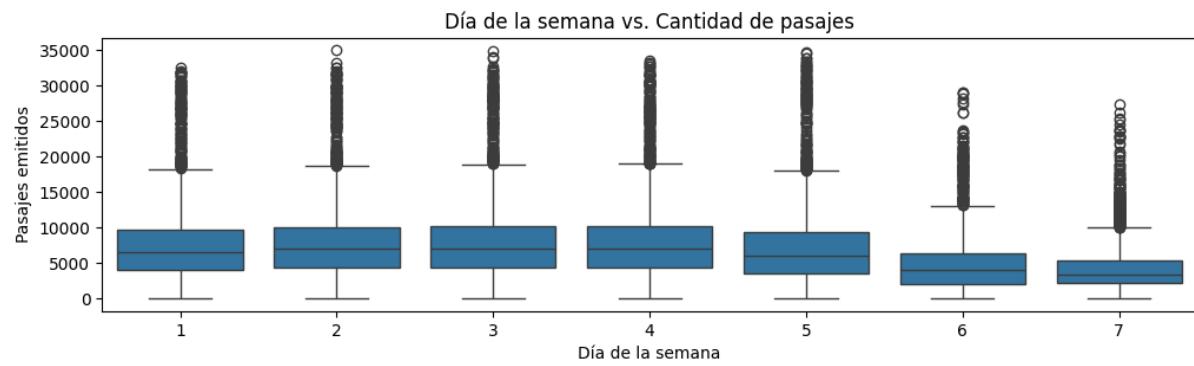
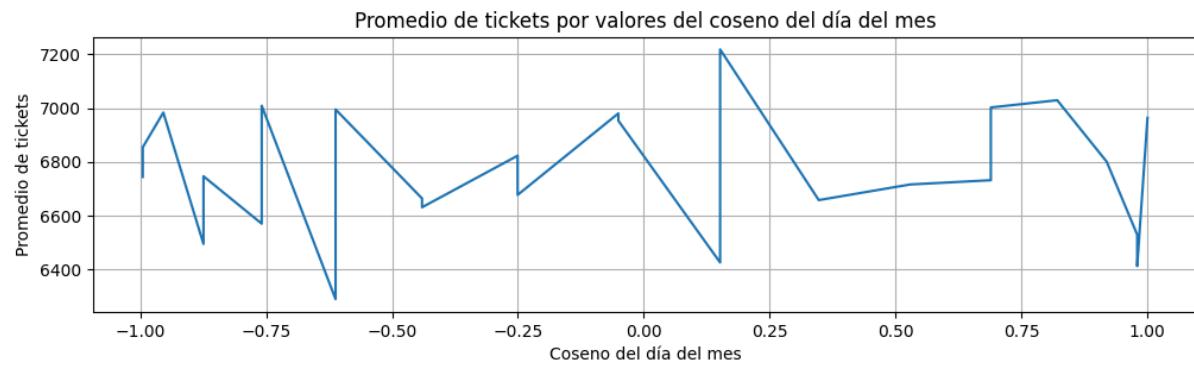
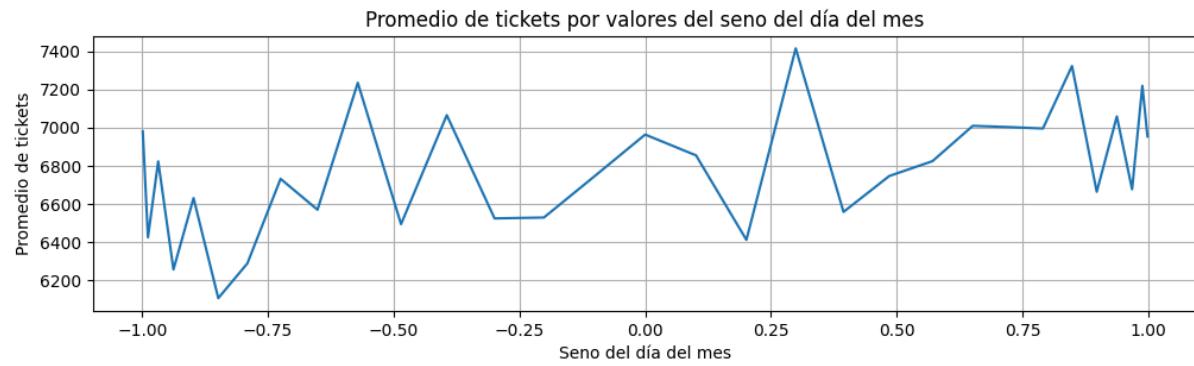


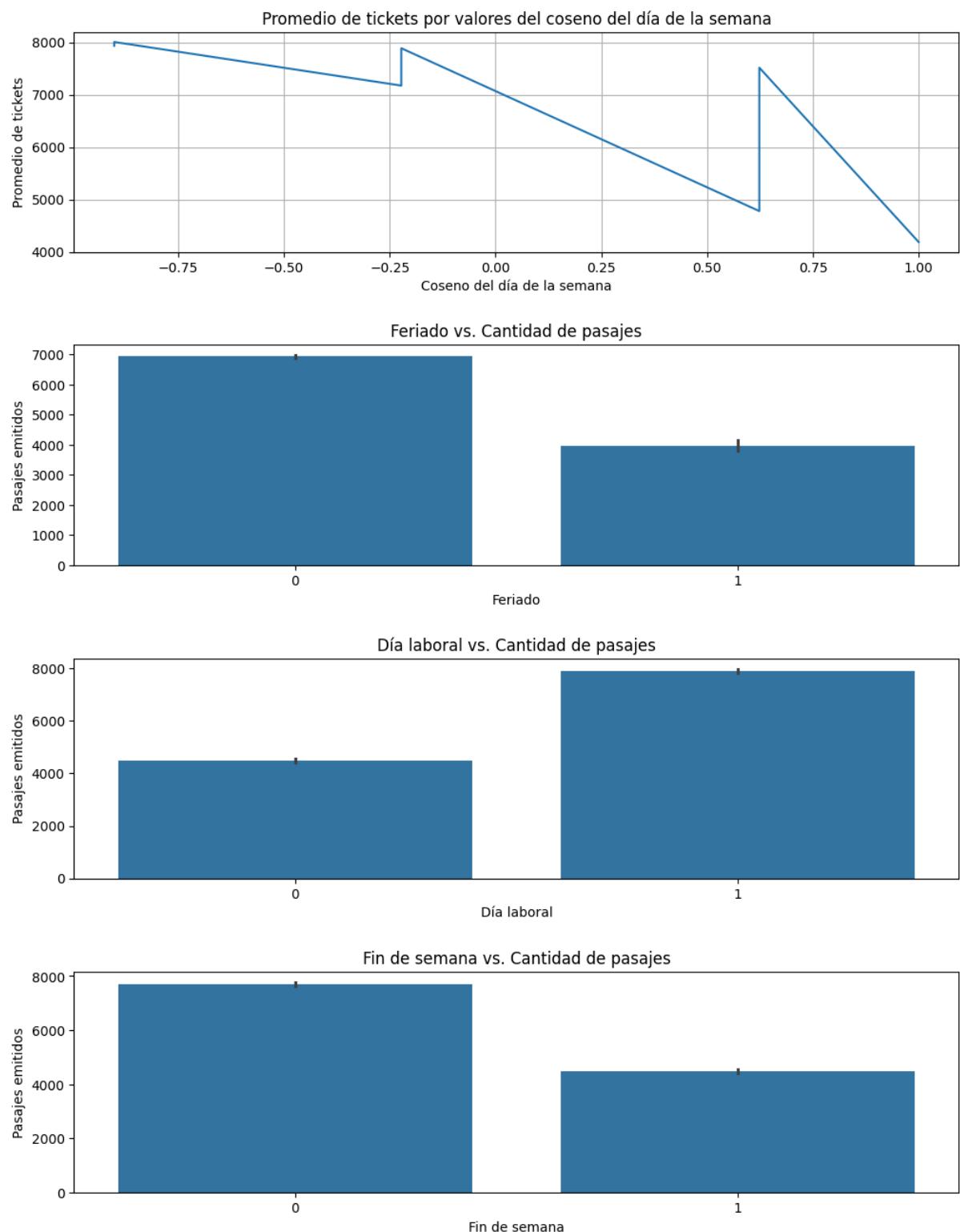
Como observamos, el 'LAG_28' ($t - 28$), puede explicar muy bien la variable objetivo. Es decir, existe una correlación entre la demanda actual y la de hace 4 semanas atrás.

Vamos a generar las nuevas características, estas incluyen los feriados, el día de la semana, si es un día laboral (Lunes a viernes, no feriado), si es fin de semana, que trimestre es, la semana del año, el día del año, la semana del mes. Además para tratar de captar la relación que tienen, por ejemplo, el día 31 de enero y el 1 de febrero, optamos por hacer el seno y el coseno de la información antes propuesta. Esto va a suponer una mejora para el modelo, haciendo que comprenda mejor la relación temporal existente.

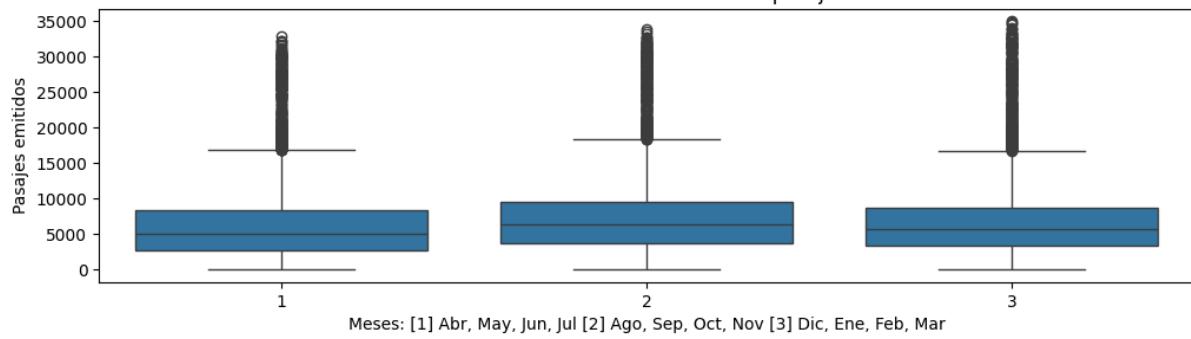


Veamos las gráficas contra nuestra variable objetivo de las nuevas características creadas:

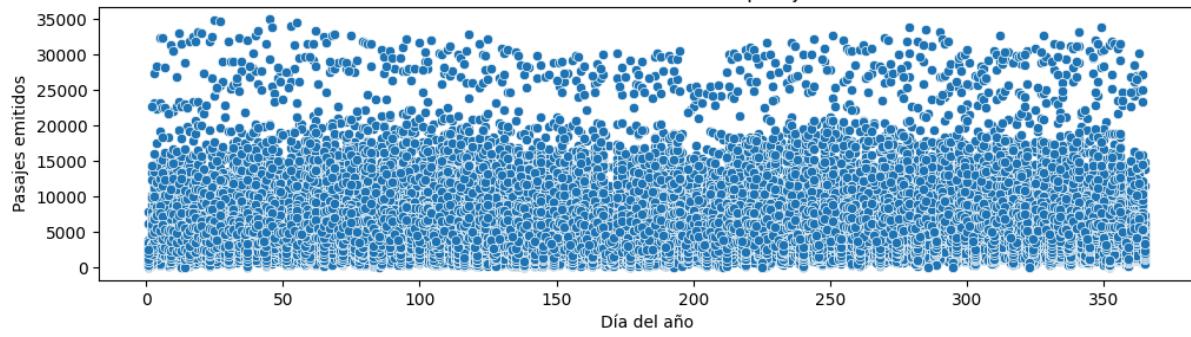




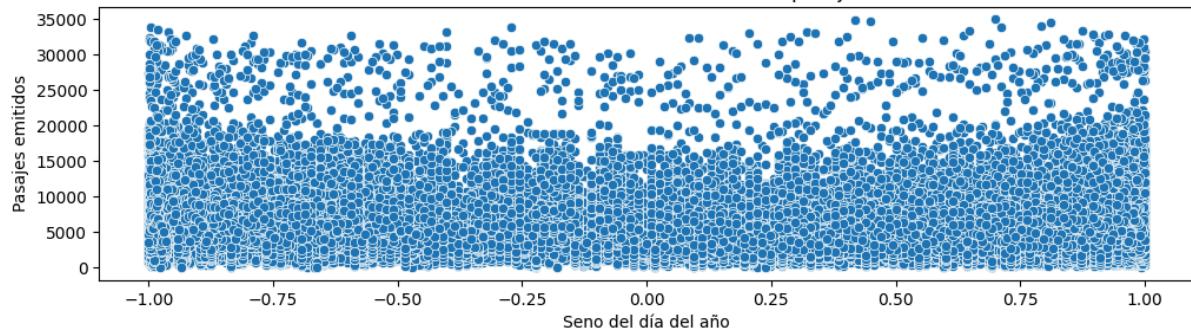
Estación del año vs. Cantidad de pasajes



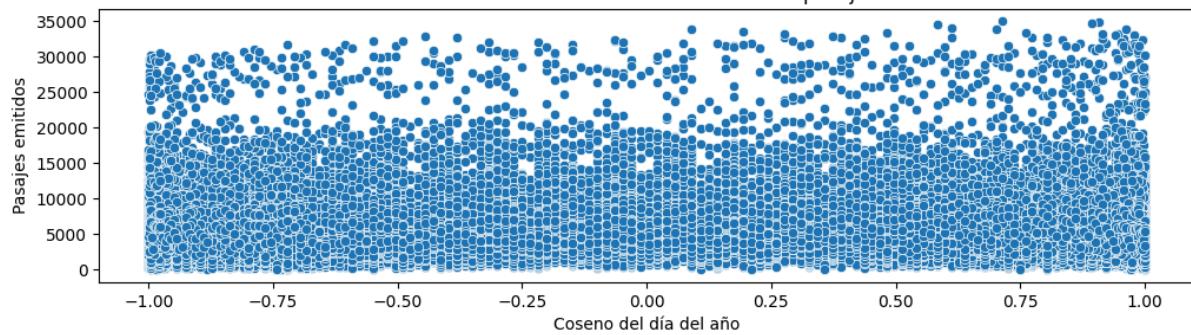
Día del año vs. Cantidad de pasajes



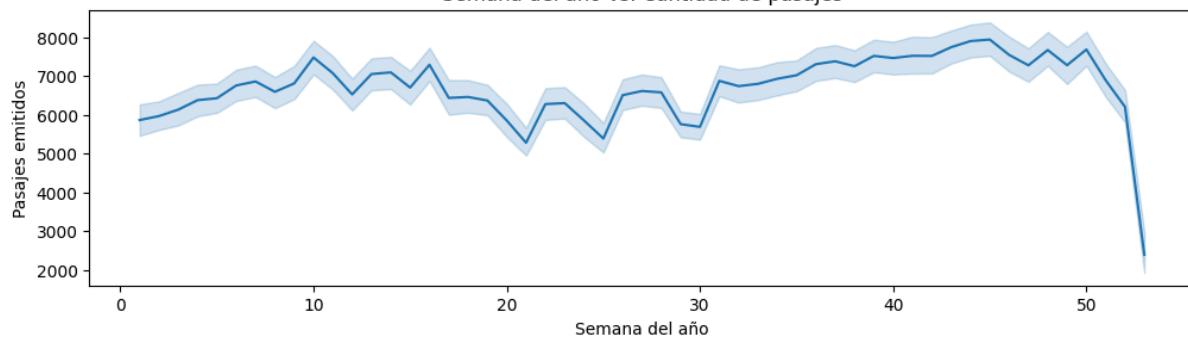
Seno del día del año vs. Cantidad de pasajes



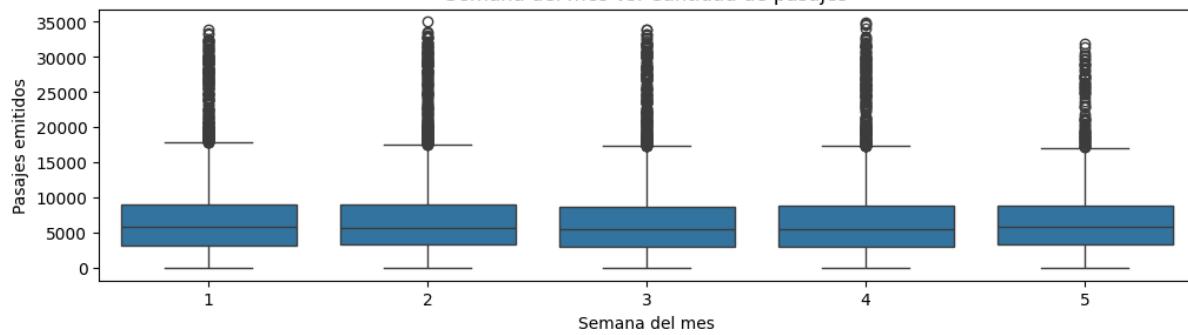
Coseno del día del año vs. Cantidad de pasajes



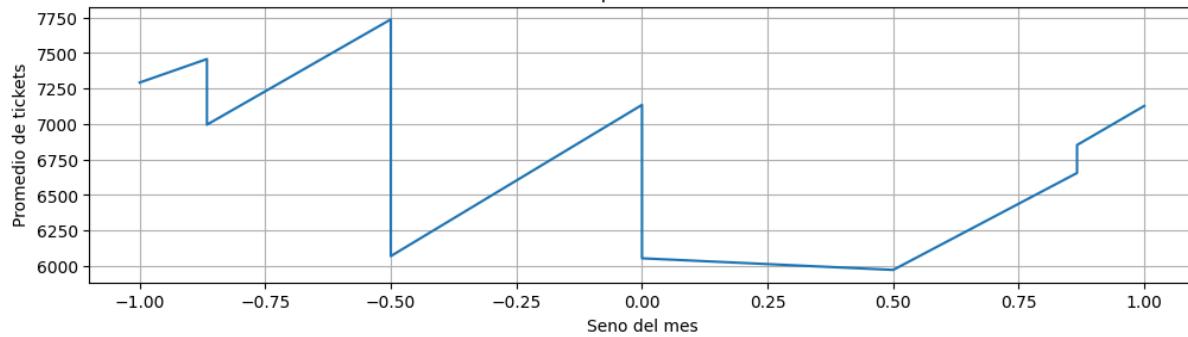
Semana del año vs. Cantidad de pasajes



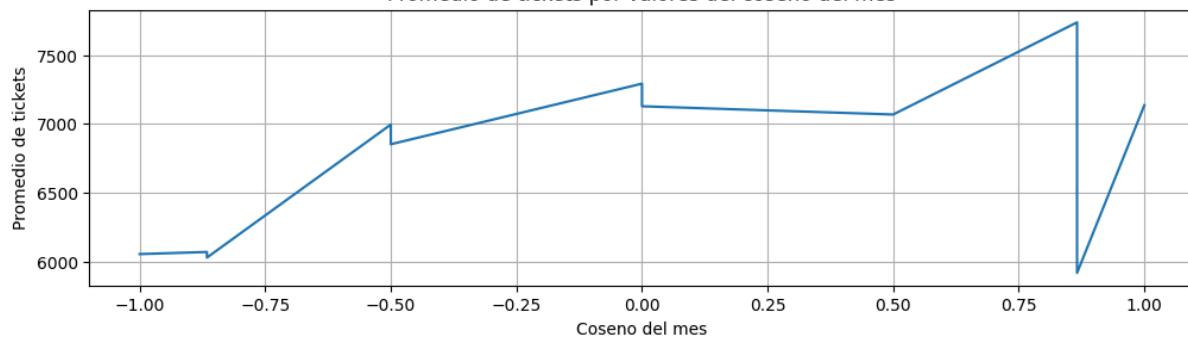
Semana del mes vs. Cantidad de pasajes

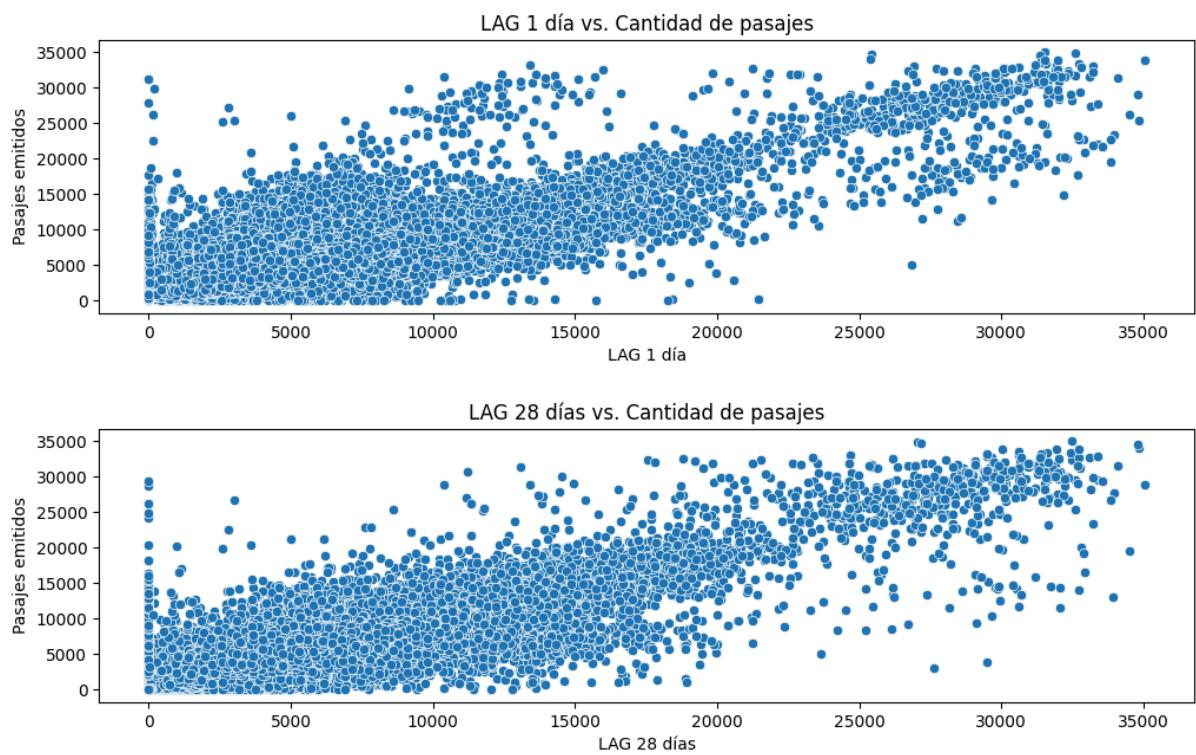


Promedio de tickets por valores del seno del mes



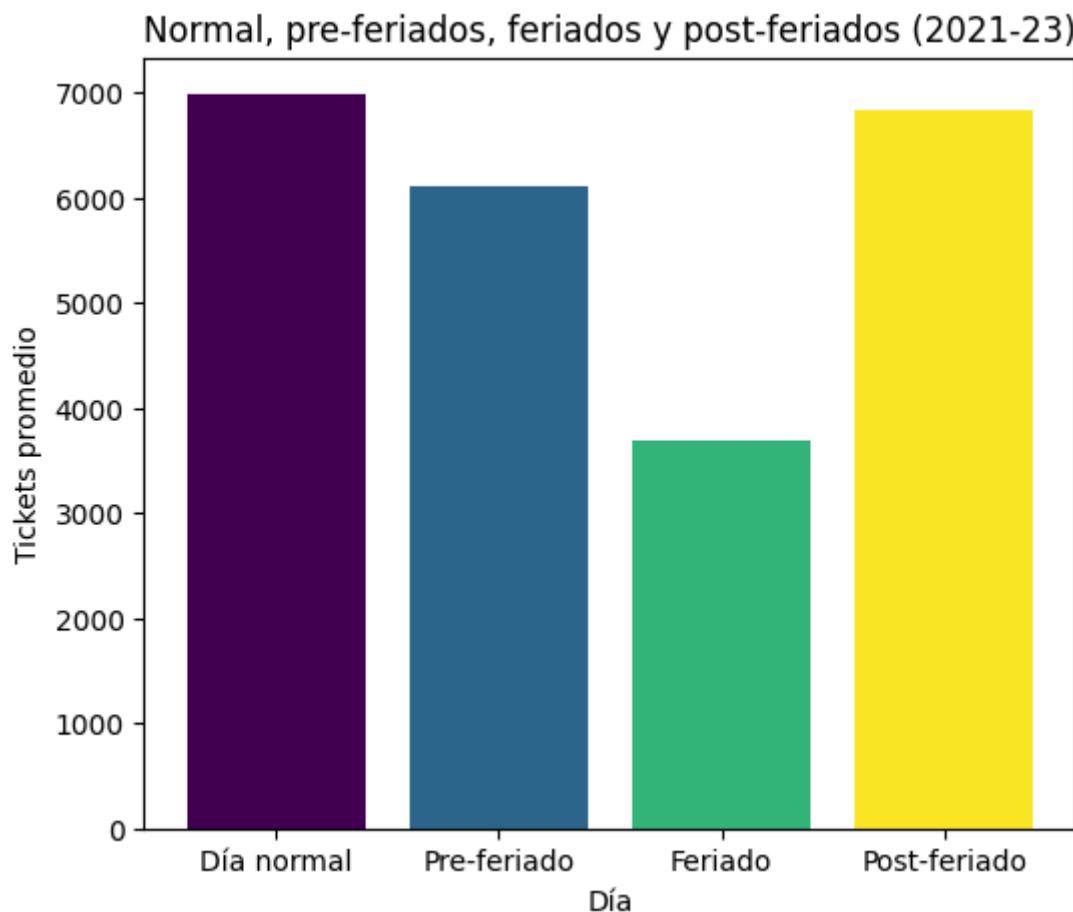
Promedio de tickets por valores del coseno del mes





¿Afectan los feriados a la demanda?

Una vez generadas las nuevas características, podemos indagar y resolver preguntas. Una de las que se me presentó fue: ¿Afectan los feriados a la demanda del transporte?



Resultados del experimento:

En conclusión, los feriados tienen un impacto significativo en la demanda de transporte, reduciendo la cantidad de boletos vendidos en los días pre-feriado y considerablemente los feriados. Esto sugiere que las personas tienden a no usar el transporte público durante los feriados, y pre-feriados. Los periodos evaluados van del 1-1-2020 al 31-12-2023.

¿Afecta la lluvia, el viento o la temperatura a la demanda?

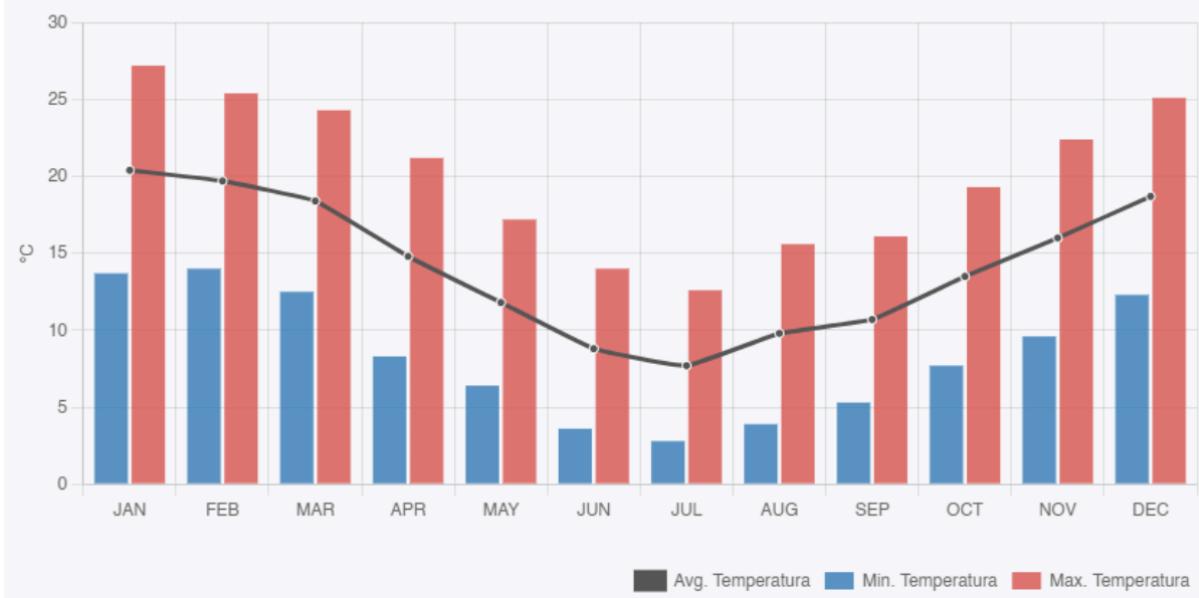
Para resolver esta inquietud decidí descargar el dataset histórico de clima de la web MeteoStat. Mediante web-scraping rápidamente obtuvimos los datos necesarios de todo el periodo para el viento, la temperatura y las precipitaciones:

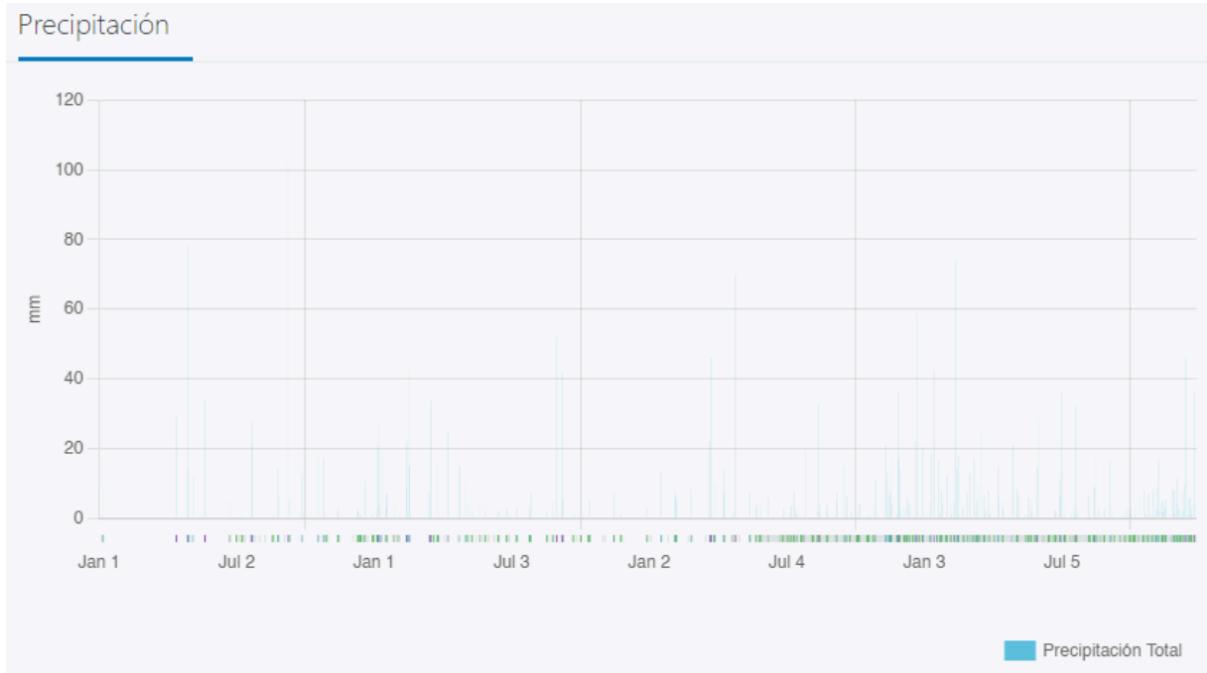
Velocidad del Viento



Temperatura ▾

Más Reciente ▾



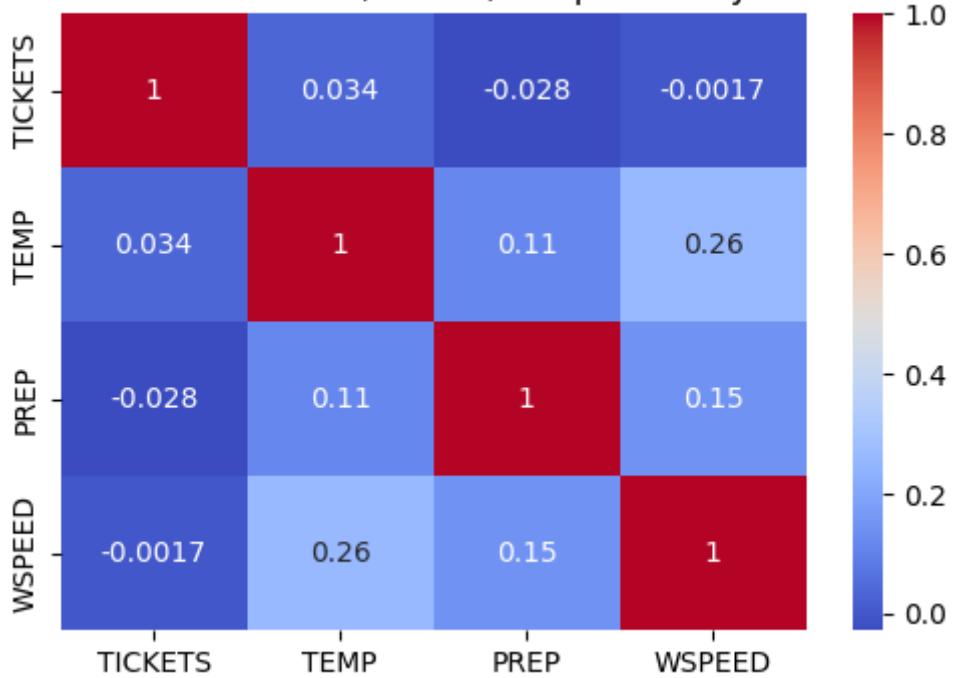


Luego de algunos ajustes, ya disponemos de nuestro dataset con el consumo de transporte diario, donde disponemos de la temperatura promedio, el nivel de precipitaciones y la velocidad del viento:

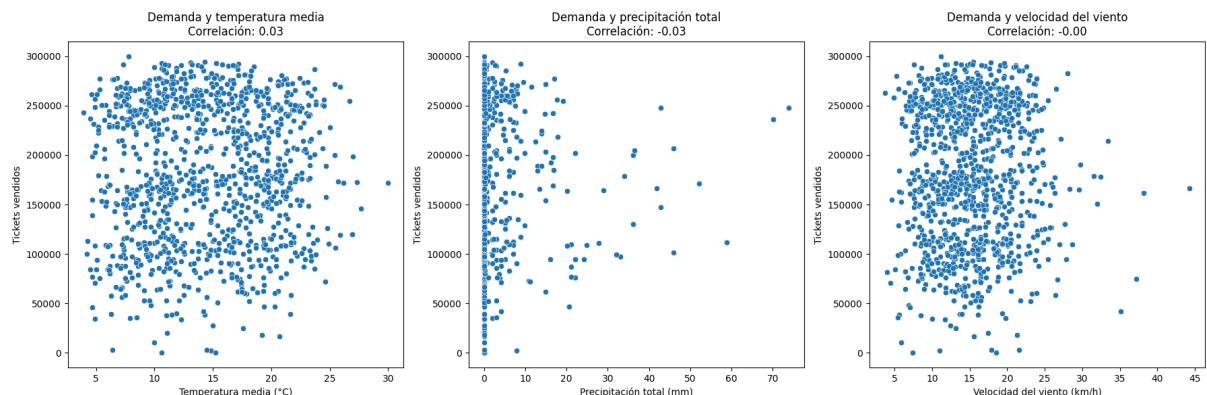
	YEAR	MONTH	DAY	TICKETS	TEMP	PREP	WSPEED	
0	2021		1	1	24794	17.6	0.0	13.2
1	2021		1	2	58561	21.7	0.0	14.1
2	2021		1	3	110429	24.9	0.0	16.8
3	2021		1	4	109282	21.1	0.0	13.6
4	2021		1	5	115442	19.3	3.0	25.4

Lo primero hacer un análisis de correlación entre la demanda y estas variables:

Corr. entre demanda, viento, temp. media y lluvia



Las correlaciones son bajas, indicando que, en este subconjunto de datos, no hay una relación lineal significativa entre el número de tickets y la temperatura o la precipitación.



Temperatura Media (TEMP):

La correlación entre la demanda de boletos y la temperatura media es muy baja (0.034). Esto sugiere que no existe una relación significativa entre la temperatura y la demanda de transporte público. Los datos de dispersión corroboran esta conclusión, mostrando una distribución uniforme de la demanda de boletos a través de diferentes rangos de temperatura.

Precipitación Total (PREP):

La correlación entre la demanda de boletos y la precipitación total es negativa pero muy débil (-0.028). Esta débil correlación negativa indica que la lluvia no tiene un impacto considerable en la demanda de transporte público. El diagrama de dispersión muestra que

la mayoría de los datos se agrupan cerca de niveles bajos de precipitación, sin una clara tendencia observable en la demanda de boletos.

Velocidad del Viento (WSPEED):

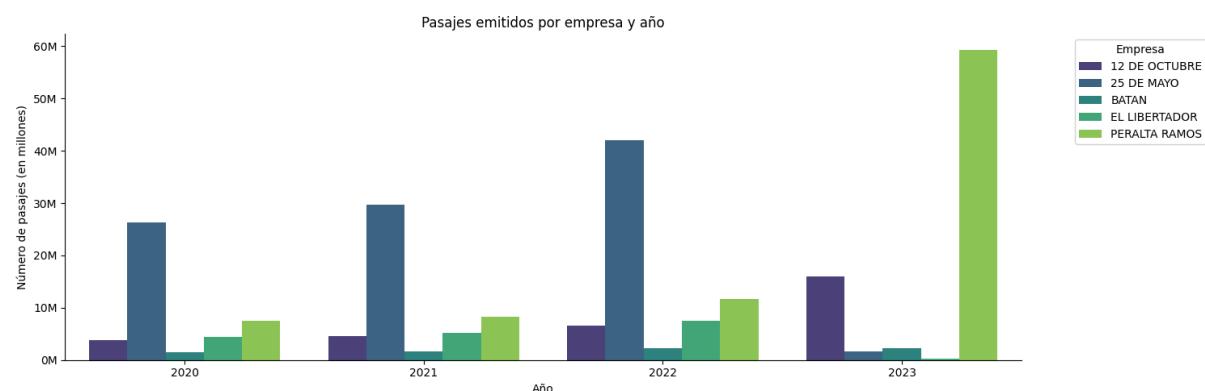
La correlación entre la demanda de boletos y la velocidad del viento es prácticamente nula (-0.0017). Esto indica que la velocidad del viento no afecta significativamente la demanda de transporte público. El diagrama de dispersión respalda esta conclusión, mostrando una dispersión aleatoria de los datos sin una tendencia discernible.

Resultados del experimento:

Las correlaciones bajas o nulas entre las variables climáticas y la demanda sugieren que, en este caso, el clima no juega un papel crucial en la variación de la demanda de transporte público en la ciudad. Esto podría deberse a varios factores, como la prevalencia de hábitos de transporte que no se ven afectados por el clima, o una adaptación de los usuarios al clima local. Los periodos evaluados van del 1-1-2020 al 31-12-2023.

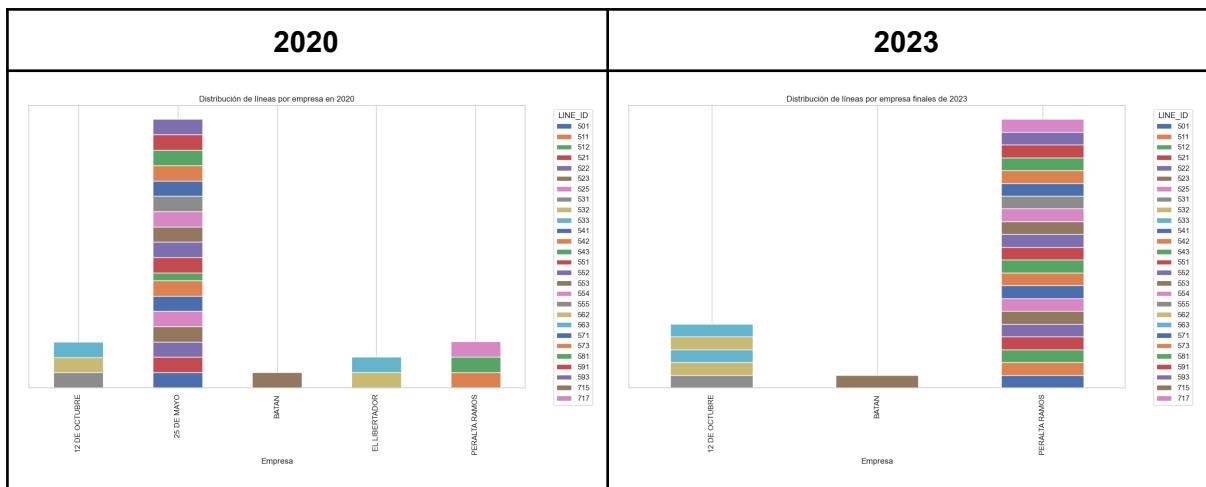
Otras gráficas de valor

Aquí vamos a encontrar algunos gráficos que nos permiten comprender la información brindada de manera clara y simple. Ahora, hagamos un repaso de lo que los datos nos cuentan a nivel empresas:



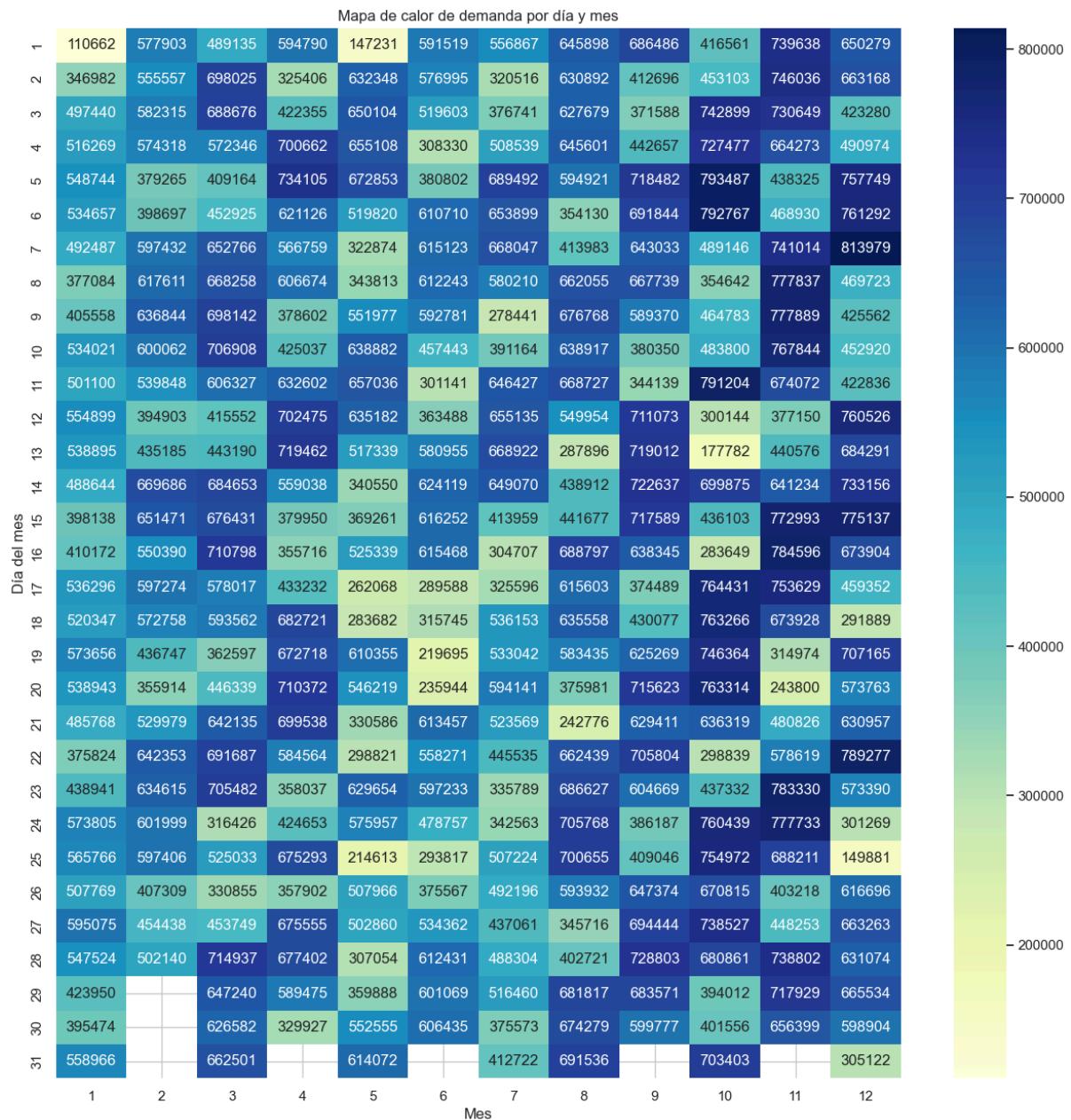
Observando el gráfico podemos apreciar la adquisición de las líneas antiguamente operadas por la empresa 25 de Mayo, que a partir de enero 13 de 2023 pasaron a la gestión de la empresa Peralta Ramos.

Esto también se puede representar de la siguiente manera:



Observamos la desaparición de las empresas 25 de Mayo y El Libertador.

Otra forma de ver el consumo claramente es usando un mapa de calor:



Este mapa de calor es una herramienta valiosa porque permite de una forma visual y rápida identificar patrones, tendencias, días de consumos bajos, etc. Esto podría ser útil para las empresas de transporte, y a responsables de políticas de transportes:

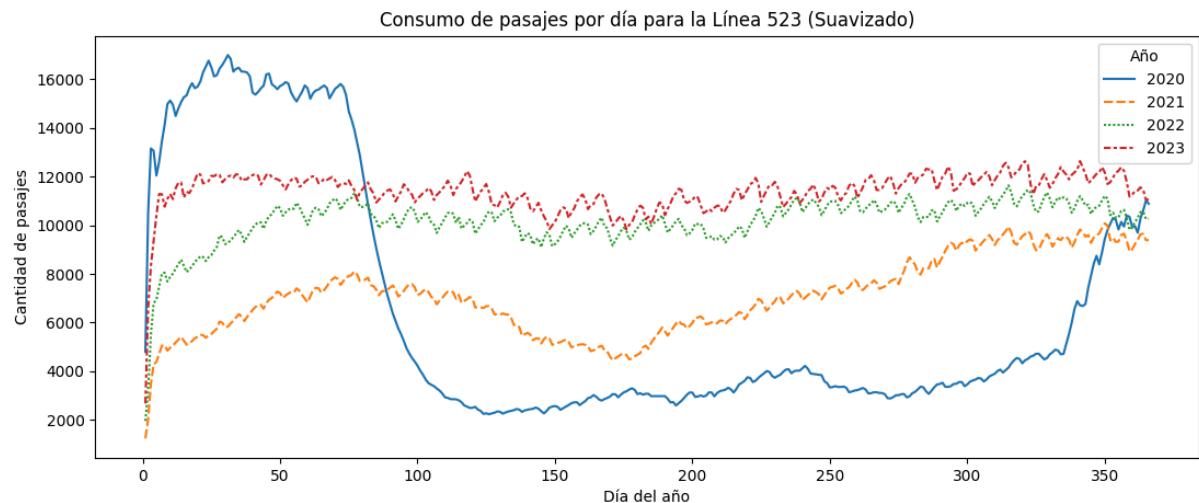
Identificación de patrones: Visualizar patrones de demanda diaria y estacional, facilitando la planificación de servicios de transporte.

Optimización de recursos: Tomar decisiones informadas sobre la asignación de recursos, como la cantidad de vehículos necesarios en diferentes días y meses.

Planificación a largo plazo: Evaluar la efectividad de cambios en el servicio y planificar futuras estrategias de operación basadas en datos históricos.

Análisis de demanda por línea

Otra forma valiosa de contrastar visualmente los datos es generando líneas de consumo suavizadas por cada año de información:



3 - Generando predicciones

Teniendo ya las características generadas por la transformación temporal o de la propia variable objetivo, vamos a evaluar cada una de las características y reflejar su importancia para la predicción de la variable objetivo. Para esto vamos a dividir las características según su naturaleza:

```
● ● ●  
categorical_vars = ['DAY', 'MONTH', 'YEAR', 'COMPANY', 'LINE_ID', 'DAY_OF_WEEK', 'SEASON',  
                    'DAY_OF_YEAR', 'WEEK_OF_YEAR', 'WEEK_OF_MONTH']  
  
binary_vars = ['HOLIDAY', 'WORKDAY', 'WEEKEND', 'DAY_BEFORE_HOLIDAY', 'DAY_AFTER_HOLIDAY']  
  
continuous_vars = ['TICKETS', 'DAY_OF_MONTH_SIN', 'DAY_OF_MONTH_COS', 'DAY_OF_WEEK_SIN',  
                   'DAY_OF_WEEK_COS', 'DAY_OF_YEAR_SIN', 'DAY_OF_YEAR_COS', 'MONTH_SIN', 'MONTH_COS',  
                   'LAG_1', 'LAG_28']
```

Comenzamos con Anova para contrastar nuestras características categóricas contra nuestra variable objetivo continua:

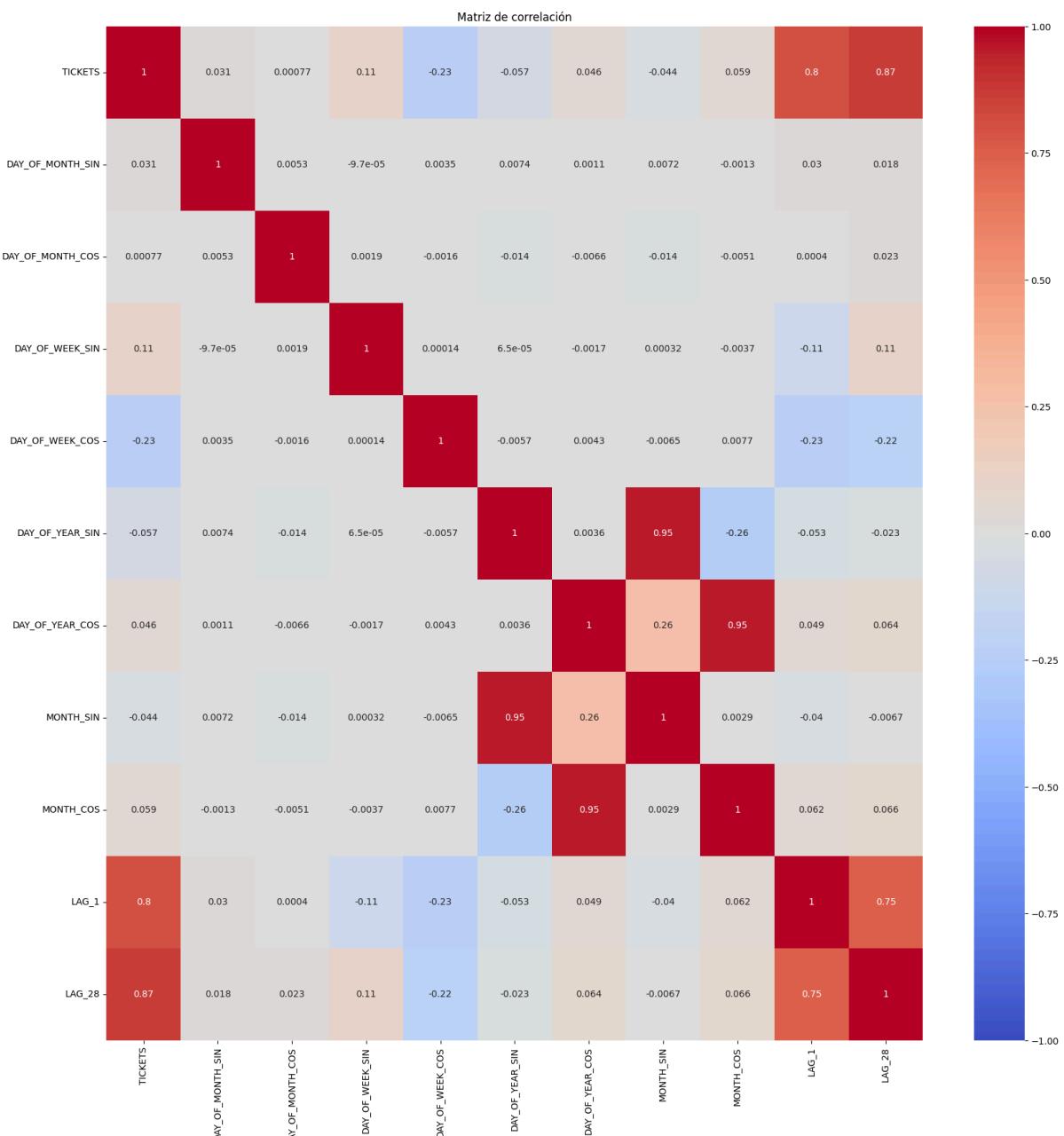
	Variable	Sum of Squares	p-value
0	LINE_ID	4.573907e+11	0.000000e+00
1	DAY_OF_YEAR	9.598867e+10	0.000000e+00
2	DAY_OF_WEEK	6.523319e+10	0.000000e+00
3	COMPANY	4.891038e+10	0.000000e+00
4	YEAR	4.672396e+10	0.000000e+00
5	WEEK_OF_YEAR	1.398184e+10	3.192264e-83
6	MONTH	1.055897e+10	3.618974e-81
7	SEASON	6.190642e+09	7.631839e-53
8	DAY	2.978629e+09	6.474812e-12
9	WEEK_OF_MONTH	6.304079e+08	6.794055e-05

El análisis de varianza (ANOVA) realizado sobre las variables categóricas mostró que la variable más importante es `LINE_ID`, con la mayor suma de cuadrados (4.606005e+11) y un valor p extremadamente significativo (0.000000e+00), indicando que esta variable tiene el mayor impacto en la variabilidad de los tickets. Le siguen `DAY_OF_YEAR` y `DAY_OF_WEEK`, también con altos valores de suma de cuadrados (9.672552e+10 y 6.538796e+10, respectivamente) y valores p significativos (0.000000e+00), lo que muestra

su relevancia en la explicación de la demanda. Otras variables importantes incluyen `COMPANY, YEAR, WEEK_OF_YEAR, y MONTH`, todas con valores p muy bajos y significativos, indicando que estas variables categóricas también pueden contribuir a la variabilidad de los tickets, aunque en menor medida que LINE_ID.

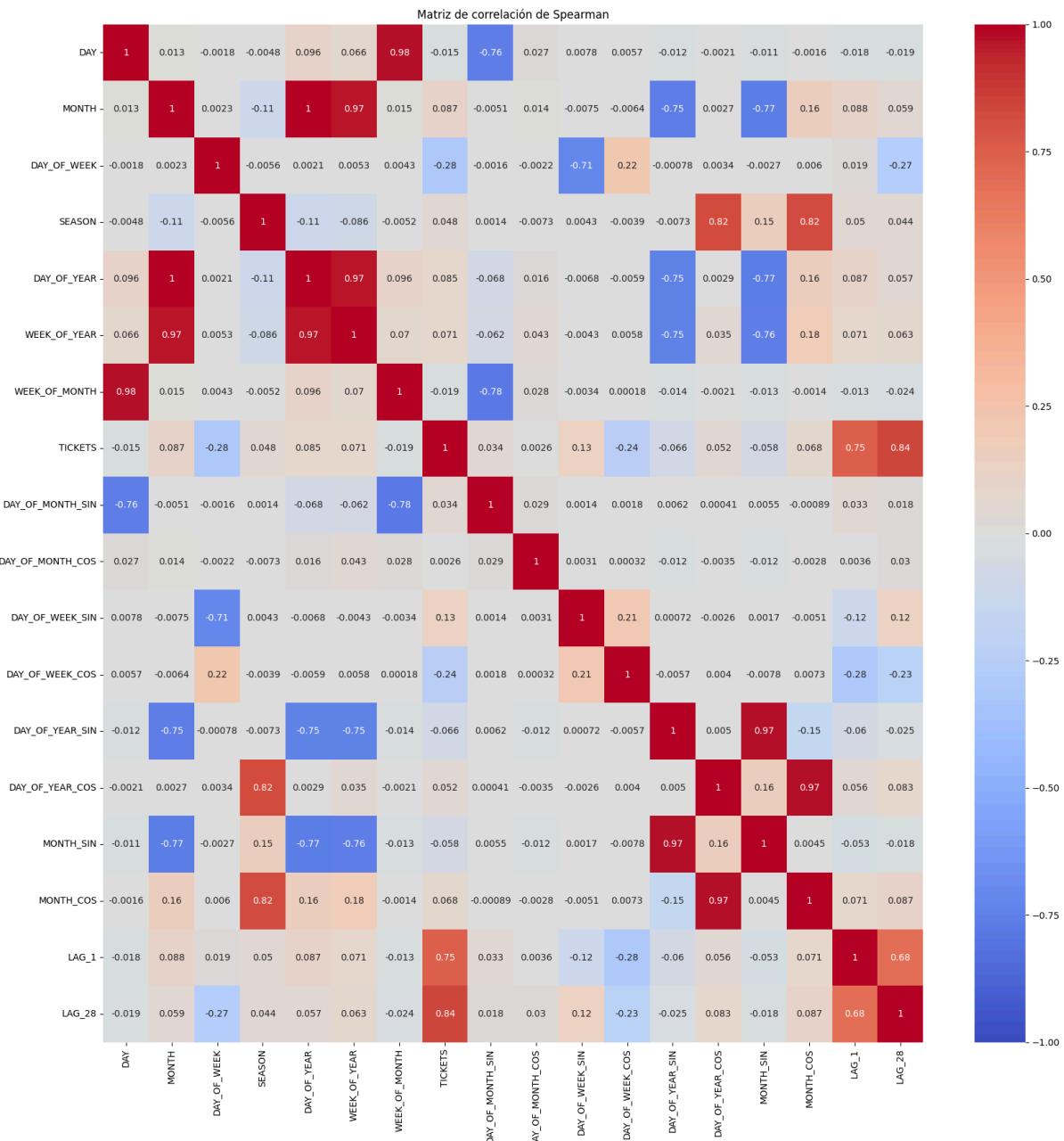
Aquí me gustaría hacer una salvedad ya que `COMPANY` presenta discrepancias en sus datos, esto se debe a que en 2023 una empresa compró varias líneas de otra empresa, esto nos puede introducir valores erróneos, tal como vimos en gráficos anteriores. He probado algunas transformaciones con esta variable pero ninguna aportó mejoras.

Ahora evaluamos todas las características continuas con una matriz de correlación de Pearson:



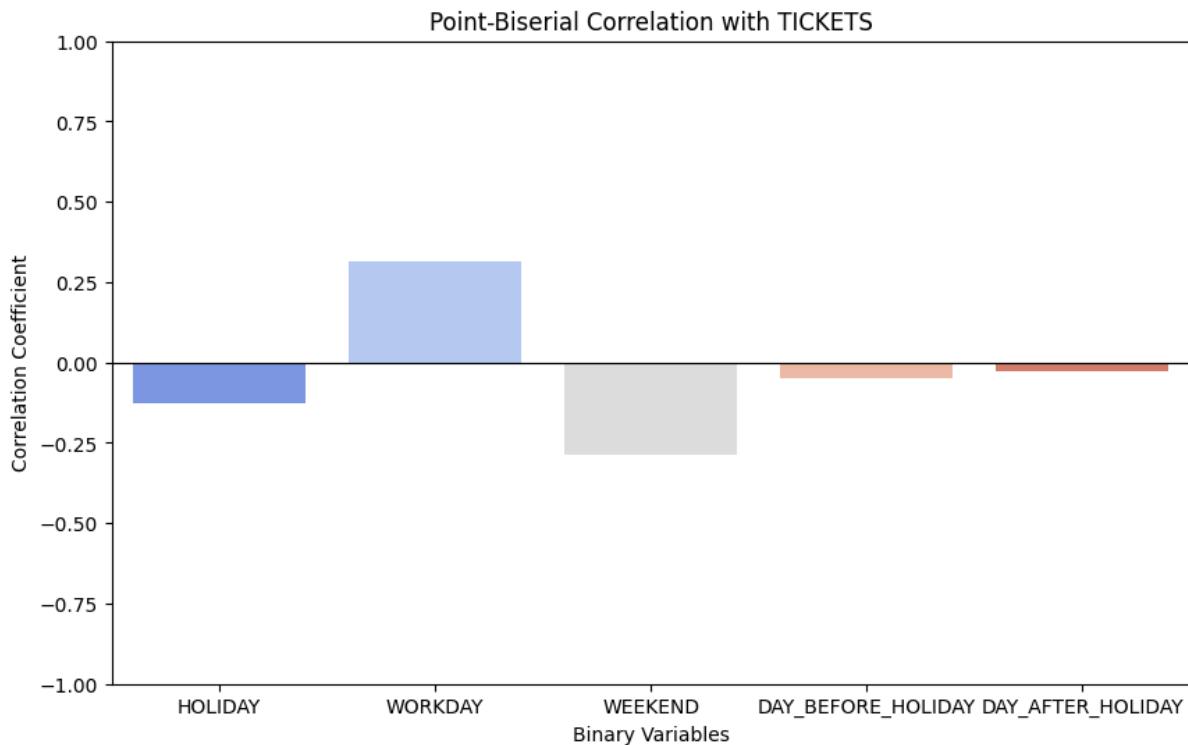
Observando la matriz podemos afirmar que las características `LAG_1` u `LAG_28` son las que mejor explican nuestra variable objetivo, aunque para evitar redundancia solo podríamos optar por una. Por último, y en menor medida, tendríamos `DAY_OF_WEEK_SIN` y `DAY_OF_WEEK_COS`, ambas tienen una correlación baja, y no se correlacionan entre sí, aunque la que menos correlación tiene con `LAG_28` es `DAY_OF_WEEK_SIN`

Ahora vamos a buscar relaciones no lineales con la matriz de Spearman, aquí podemos sumar las variables ordinales:



Observamos con algo de correlación a `LAG_1` que podría ir con `DAY_OF_WEEK`. Podríamos probarla en lugar de `LAG_28` junto a `DAY_OF_YEAR_SIN`, pero no juntas.

Por último evaluamos nuestras características binarias con una correlación biserial:



De aquí observamos que `WORKDAY` y `WEEKEND` serían un buen punto de partida, aunque al ser mutuamente excluyentes, nos quedamos con `WORKDAY`.

Modelo

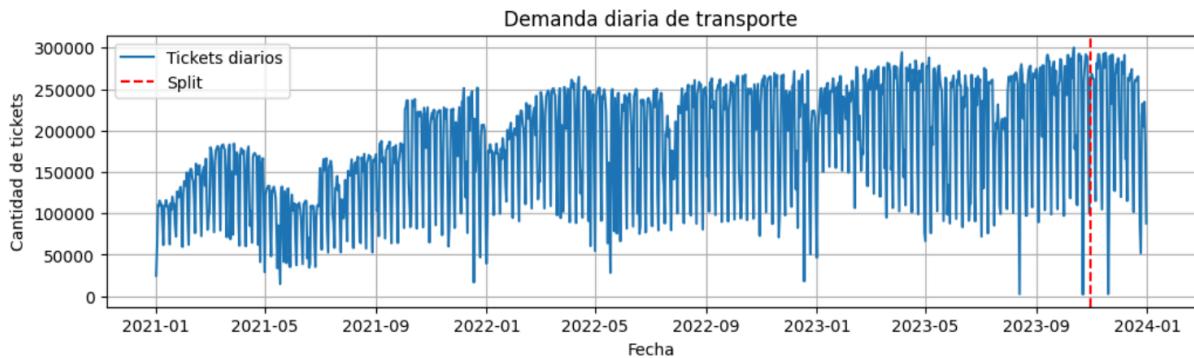
Elegí XGBoost para predecir la demanda de transporte debido a varias ventajas que considero importantes. En primer lugar, XGBoost es notablemente resistente a los outliers, una característica esencial dado que los datos de transporte contienen demandas significativas, más que nada en verano, que podrían distorsionar los resultados en otros modelos. Además, XGBoost maneja eficazmente los valores faltantes, imputándolos durante el proceso de modelado. Otro aspecto destacable es su bajo costo computacional, ya que está diseñado para ser eficiente en términos de tiempo y recursos, permitiendo entrenar modelos complejos sin requerir un hardware excesivamente potente. La configuración inicial es relativamente sencilla y el algoritmo proporciona un equilibrio robusto entre rendimiento y facilidad de uso, permitiendo obtener resultados precisos sin necesidad de una extensa optimización de hiperparámetros.

Las características elegidas son:

```
● ● ●  
# Selección de características  
X = df[['DAY_OF_WEEK_SIN', 'LINE_ID', 'WORKDAY', 'LAG_28', 'YEAR', 'DAY_OF_YEAR_SIN']].apply(pd.to_numeric)  
y = df['TICKETS']
```

Entrenamiento

La primera consta de dividir los datos en entrenamiento / test por medio de una fecha. En este caso vamos a entrenar con el rango de datos desde el 1/1/2021 hasta el 31/10/2023, y testear con los datos restantes de 2023. De esta manera evalúo el comportamiento del modelo con datos más recientes, ya que no olvidemos nuestra misión es predecir 2024:



Ahora si, podemos pasar al entrenamiento:

```
● ● ●

# Dividimos los datos en conjuntos de entrenamiento y prueba por fecha
train_start_date = '2022-01-01'
split = '2023-10-31'

# Creamos conjuntos de entrenamiento y prueba
X_train = X[(df['DATE'] >= train_start_date) & (df['DATE'] < split)]
X_test = X[df['DATE'] >= split]
y_train = y[(df['DATE'] >= train_start_date) & (df['DATE'] < split)]
y_test = y[df['DATE'] >= split]

# =====

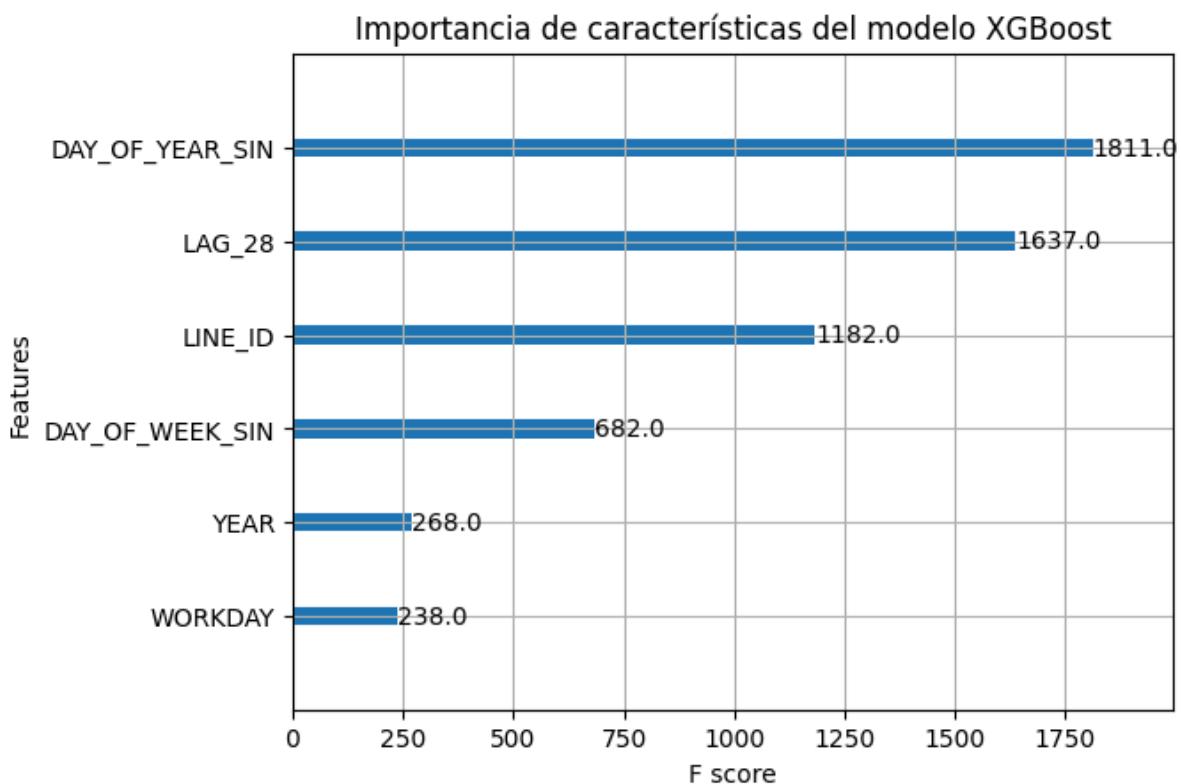
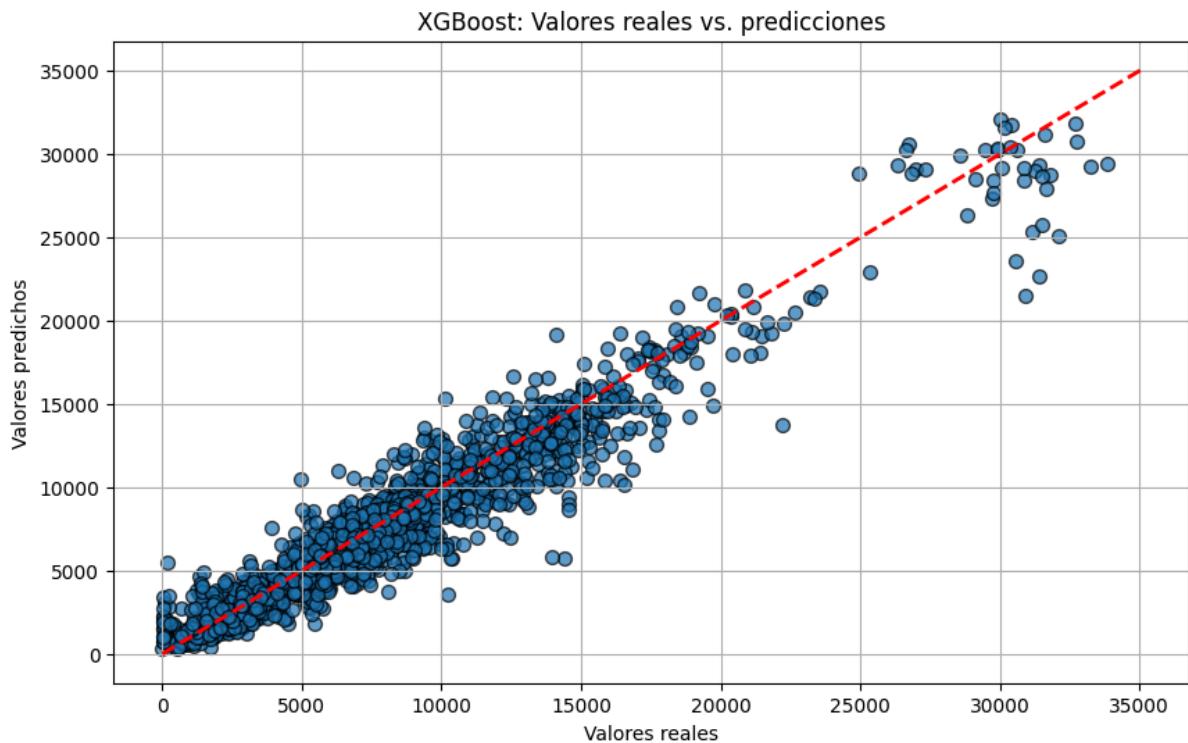
y_train = np.log1p(y_train)

# Definimos el modelo
xgb_model_a = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
xgb_model_a.fit(X_train, y_train)

# Predecir en el conjunto de prueba
y_pred = xgb_model_a.predict(X_test)
y_pred = np.expm1(y_pred)
```

Aquí los resultados fueron:

Mean Squared Error (XGBoost): 2670241.365722042
R^2 Score (XGBoost): 0.9190491437911987
Mean Absolute Error (XGBoost): 1162.7118050451063
Root Mean Squared Error (XGBoost): 1634.0873188792702
Explained Variance Score (XGBoost): 0.923362658358684



Los resultados obtenidos son muy aceptables, con un Mean Squared Error (MSE) de 2,670,241.37, se observa que el modelo tiene un error cuadrático medio relativamente bajo, lo que indica una buena precisión en sus predicciones. El R² Score de 0.919 sugiere que el modelo explica el 91.9% de la variabilidad en los datos de respuesta, evidenciando una alta capacidad predictiva. Además, el Mean Absolute Error (MAE) de 1,162.71 reafirma que las predicciones del modelo son consistentes y cercanas a los valores reales. El Root Mean Squared Error (RMSE) de 1,634.09, aunque más alto que el MAE, sigue siendo aceptable y muestra una dispersión baja en las predicciones. Finalmente, el Explained Variance Score de 0.923 indica que el modelo captura la varianza en los datos de manera efectiva.

Las series temporales presentan características notablemente diferentes a problemas de clasificación o regresión sin dependencia temporal.

Ahora, teniendo en cuenta que queremos predecir 2024, el modelo se va a ver beneficiado de un entrenamiento con las fechas más recientes, que en la prueba anterior fueron utilizadas de testeo. Las siguientes predicciones se realizan con el modelo entrenado con **todo** el conjunto de datos.

```
● ● ●  
# Definimos el modelo  
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)  
xgb_model.fit(X, y)
```

Generando 2024

Para generar los datos de 2024, primero debemos recrear las condiciones o características que ingresaron el modelo, es decir que por cada fechas vamos a tener que contar con el seno del día de la semana, si es laborable, su línea, su retraso de 28 días y el seno del día del año. Para esto vamos a utilizar una serie de funciones que nos ayuden a tal fin:

```

def get_lag_28_value(date, df_line):
    lag_28_date = date - timedelta(days=28)
    lag_28_value = df_line[df_line['DATE'] == lag_28_date]['TICKETS']
    if lag_28_value.empty:
        return None
    return lag_28_value.values[0]

# Obtener feriados Argentina
ar_holidays = holidays.Argentina(years=[2024])

# Agregamos las fechas que faltan
manual_holidays = {
    '2024-03-28': "Jueves santo"
}
ar_holidays.update(manual_holidays)

for date, holiday in ar_holidays.items():
    print(f'{date}: {holiday}')

# Definir la función que determina si una fecha es feriado
def is_holiday(date):
    return int(date in ar_holidays)

# Definir la función que determina si una fecha es un día hábil (workday)
def is_workday(date, day_of_week):
    return 1 if day_of_week in range(1, 6) and not is_holiday(date) else 0

```

En esta sección obtenemos los feriados de 2024, donde manualmente agregamos Jueves Santo como fecha faltante, además la función que determina si es feriado, y si es día laboral.

En la siguiente imagen viene la función principal, esta se encarga de generar predicciones diarias de la cantidad de pasajes utilizados por cada línea de colectivos existente durante el último periodo de 2023. Inicialmente, filtro el DataFrame original para obtener los registros correspondientes a la línea de interés, y convierto las fechas a un formato adecuado. Posteriormente, inicializo listas para almacenar tanto las predicciones generadas como las fechas omitidas, en caso de que existan, debido a la falta de datos históricos necesarios. Para cada fecha en el rango especificado, calculo el día de la semana y el día del año, determino si es un día laboral, y obtengo el valor del retraso de 28 días. Si el retraso está disponible, realizo transformaciones sinusoidales para las características temporales, construyo un DataFrame con las características necesarias y genero la predicción utilizando el modelo entrenado con toda la serie temporal (De 2021 a 2023). La predicción es ajustada

y almacenada junto con la información de la fecha. Además, las fechas para las cuales no se pudo obtener el retraso de 28 días son registradas para su revisión.

```
● ● ●

def generate_date_range(year):
    start_date = datetime(year, 1, 1)
    end_date = datetime(year, 12, 31)
    return pd.date_range(start_date, end_date).date

# =====

def generate_predictions(df, line_id, company, model, date_range):

    # Filtramos para la linea especificada
    df_line = df[df['LINE_ID'] == line_id].copy()
    df_line['DATE'] = pd.to_datetime(df_line['DATE']).dt.date
    df_line = df_line.sort_values(by='DATE')

    # Inicializamos una lista para almacenar las predicciones
    predictions = []
    omitions = []

    # Generamos la predicción para cada día
    for date in date_range:
        day_of_week = date.weekday() + 1
        day_of_year = date.timetuple().tm_yday

        workday = is_workday(date, day_of_week)
        lag_28, lag_28_date = get_lag_28_value(date, df_line)

        # Verificamos si LAG_28 existe
        if (lag_28 is not None):

            if lag_28 < 0:
                print(f'Lag_28 menor: [{lag_28}] - Fecha: {lag_28_date} - Linea: {line_id}')

            day_of_week_sin = np.sin(2 * np.pi * day_of_week / 7)
            day_of_year_sin = np.sin(2 * np.pi * day_of_year / 366) # Solo para 2024

            data = pd.DataFrame({
                'DAY_OF_WEEK_SIN': [day_of_week_sin],
                'LINE_ID': [line_id],
                'WORKDAY': [workday],
                'LAG_28': [lag_28],
                'YEAR': 2024,
                'DAY_OF_YEAR_SIN': [day_of_year_sin]
            })

            prediction = model.predict(data)
            prediction = prediction[0]
            prediction = round(prediction)

            if prediction >= 0:
                # Almacenamos la predicción con la fecha correspondiente
                predictions.append({
                    'DATE': date,
                    'DAY': date.day,
                    'MONTH': date.month,
                    'YEAR': date.year,
                    'COMPANY': company,
                    'LINE_ID': line_id,
                    'TICKETS': prediction
                })

            # Agregamos la predicción al DataFrame df_line
            new_row = pd.DataFrame({'DATE': [date], 'TICKETS': [prediction], 'LINE_ID': [line_id]})
            df_line = pd.concat([df_line, new_row], ignore_index=True)

        else:
            print(f'Prediccion negativa: [{prediction}] - Fecha: {date} | Lag: {lag_28} - Fecha lag: {lag_28_date} - Linea: {line_id}')
            print(f'Lag_28 nulo: [{lag_28}] - Fecha: {lag_28_date} - Linea: {line_id}')

    # Convertimos las predicciones en DataFrame
    predictions_df = pd.DataFrame(predictions)

return predictions_df
```

Esta función se encarga de generar las variables necesarias para la predicción, evaluar si hay datos para generar una predicción satisfactoria y por último evalúa el valor de predicción. Para hacer uso de la función antes mencionada, primero buscamos las líneas que operaron sobre el fin de 2023 y hacemos la predicción de cada fecha:

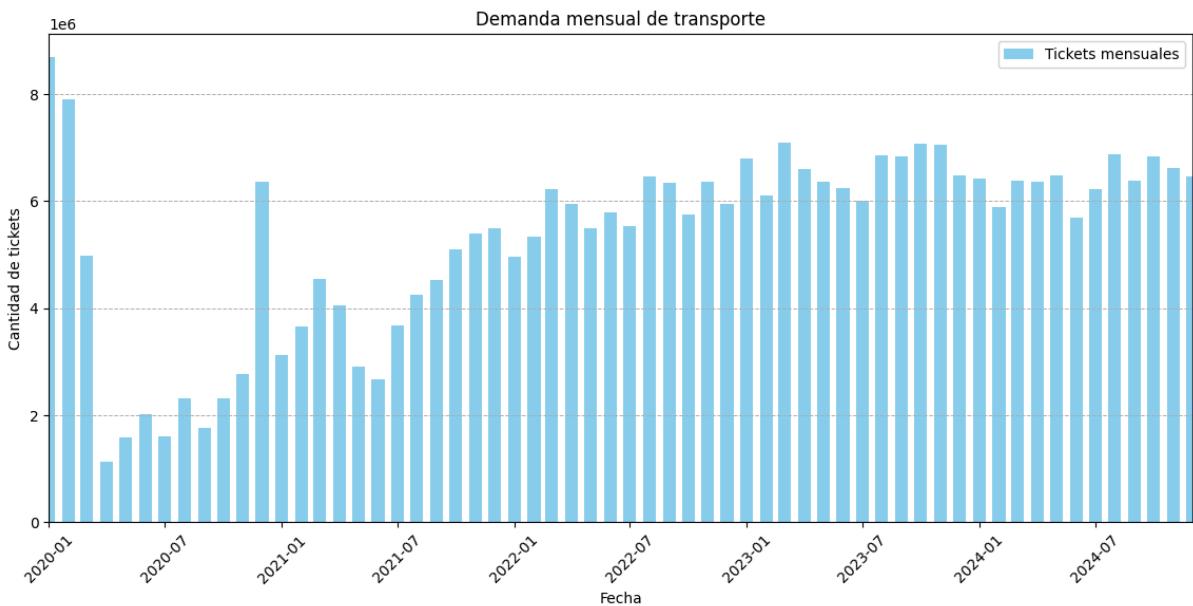
```
# Para todas las líneas
all_predictions = []

# Tomamos las líneas posteriores al 1/4 por la nueva organización de líneas
df_new_lines = df[df['DATE'] > '2023-04-01']
unique_line_company = df_new_lines[['LINE_ID', 'COMPANY']].drop_duplicates()

date_range = generate_date_range(2024)

for _, row in unique_line_company.iterrows():
    line_id = row['LINE_ID']
    company = row['COMPANY']
    pred_df = generate_predictions(df, line_id, company, 2024, xgb_model, date_range)
    if pred_df is not None:
        all_predictions.append(pred_df)
```

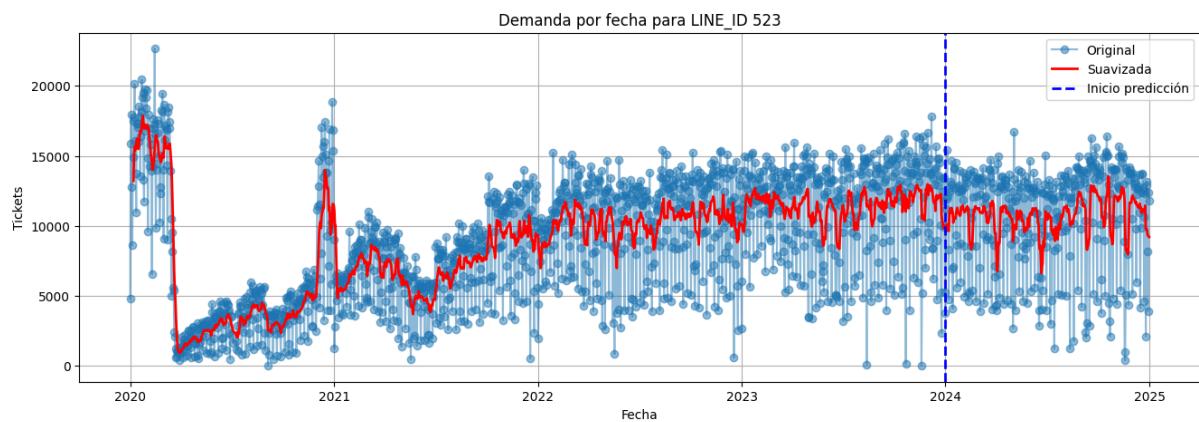
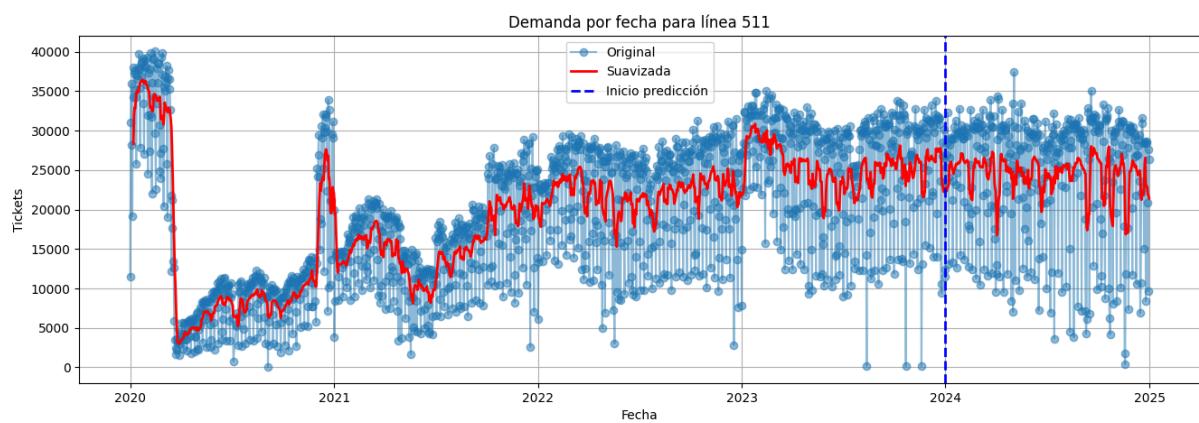
A partir de las predicciones de demanda, he generado una serie de visualizaciones que proporcionan una visión integral sobre la evolución de los boletos de colectivo a lo largo de los últimos años y las nuevas proyecciones para el año 2024.



En esta gráfica, observamos la demanda mensual desde enero de 2020 hasta el fin del corriente año. Como mencionamos anteriormente, hay una notable fluctuación en la

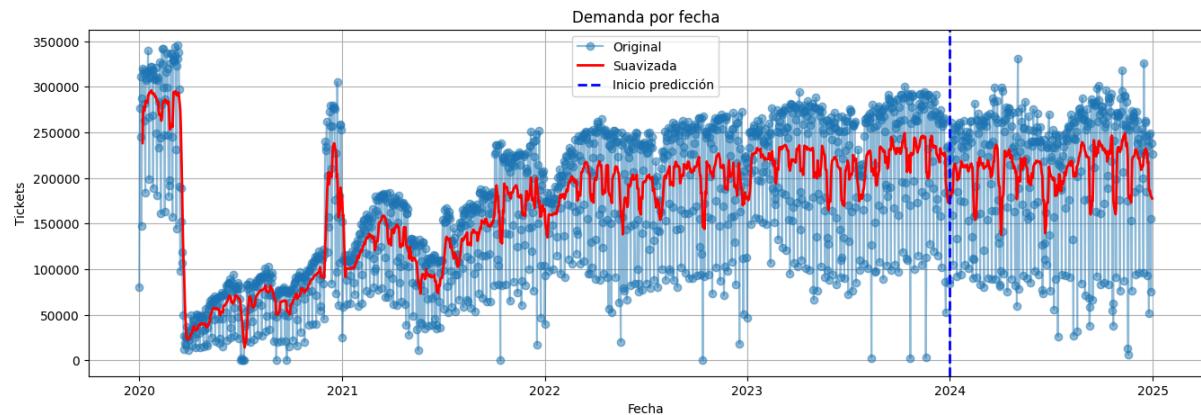
cantidad de tickets mensuales emitidos, con picos significativos a principios de 2020, y luego una caída abrupta por las restricciones de movilidad de la pandemia. A medida que avanzamos hacia 2021 y 2022, se puede notar una recuperación gradual en la cantidad de tickets mensuales, alcanzando una cierta estabilidad en 2023 y en la predicción de 2024. Este comportamiento sugiere un retorno a la normalidad en términos de movilidad urbana, aunque con algunas fluctuaciones mensuales típicas de la estacionalidad. Si observamos en detalle la demanda de enero de 2020 nos obliga a preguntarnos si quizás el home-office, que se implementó masivamente durante la pandemia, ha tenido un impacto duradero en la forma en que las personas se desplazan.

Ahora veamos en detalle algunas líneas de transporte en particular:



En el caso de la línea 523, se puede observar que la demanda experimentó un fuerte descenso en 2020, seguido de una recuperación gradual durante 2021 y 2022. Las predicciones para 2024 indican una tendencia de estabilidad con ligeras variaciones estacionales. Para la línea 511, el patrón es similar, con una caída pronunciada en 2020, una recuperación en los años siguientes, y una proyección para 2024 que sugiere estabilidad con algunas fluctuaciones estacionales.

Conclusiones



El hecho de trabajar con series temporales le da un plus de dificultad, sin embargo, se aprende bastante sobre transformaciones de características y cómo estas afectan al modelo. Generar la predicción por día para cada línea de transporte supone un desafío significativo, ya que implica considerar múltiples variables, tanto estacionales como eventuales, que pueden influir en la demanda.

La capacidad de prever la demanda diaria no solo permite optimizar la asignación de recursos y mejorar la planificación operativa, sino que también proporciona una herramienta valiosa para anticipar y mitigar posibles problemas en el sistema de transporte.

Esta experiencia resalta la importancia de una adecuada selección y transformación de características, así como el valor de integrar múltiples enfoques metodológicos para abordar la variabilidad inherente en las series temporales.

4 - Análisis espacial de demanda

El objetivo durante este proceso es tener un mapa de la ciudad donde se vea claramente el contraste de la demanda de transporte mes a mes durante el periodo analizado.

Para esto obtuve los puntos de las paradas, también de la página web de transporte. Primero traemos nuestros datos originales, incluidas las predicciones para 2024:

	DATE	DAY	MONTH	YEAR	COMPANY	LINE_ID	TICKETS
0	2020-01-01	1	1	2020	25 DE MAYO	501	315
1	2020-01-01	1	1	2020	PERALTA RAMOS	511	11463
2	2020-01-01	1	1	2020	PERALTA RAMOS	512	1889
3	2020-01-01	1	1	2020	25 DE MAYO	521	2729
4	2020-01-01	1	1	2020	25 DE MAYO	522	4010
...
48657	2024-12-31	31	12	2024	PERALTA RAMOS	581	6728
48658	2024-12-31	31	12	2024	PERALTA RAMOS	591	7552
48659	2024-12-31	31	12	2024	PERALTA RAMOS	593	4728
48660	2024-12-31	31	12	2024	BATAN	715	6894
48661	2024-12-31	31	12	2024	PERALTA RAMOS	717	5857

48662 rows × 7 columns

Posteriormente cargamos los datos de las paradas:

	cartodb_id	linea	geometry
0	1	720	POINT (-57.54245 -38.00059)
1	2	720	POINT (-57.54245 -38.00059)
2	3	720	POINT (-57.54245 -38.00059)
3	4	720	POINT (-57.54245 -38.00059)
4	5	720	POINT (-57.54245 -38.00059)
...
10076	10077	717	POINT (-57.65495 -37.93991)
10077	10078	717	POINT (-57.65220 -37.94155)
10078	10079	717	POINT (-57.65220 -37.94155)
10079	10080	717	POINT (-57.64052 -37.94983)
10080	10081	717	POINT (-57.64052 -37.94983)
10081 rows × 3 columns			

Luego de algunos ajustes, comparamos la información de mapas, contra la información de líneas y notamos:

```
Valores únicos de LINE_ID en df: 27
Valores únicos de LINE_ID en gdf: 26
LINE_IDs en df pero no en gdf: {554, 715}
LINE_IDs en gdf pero no en df: {720}
```

No poseemos información de las rutas de las líneas 715 y 554. Y, por otro lado, tenemos una ruta de la cual no tenemos datos de consumo, la 720, que fue eliminada del análisis por la completa ausencia de información de consumo.

Esto no supone un problema, ya que:

La línea 554 utiliza la misma ruta que la línea 552 y 551, podríamos hacer una copia de alguna y renombrar la ruta con este id. En tanto la línea 715, utiliza la misma ruta que la 720, podríamos simplemente renombrarla.

```

# Copiar los datos de la línea 551 para crear la nueva línea 554
line_554 = gdf[gdf['LINE_ID'] == 551].copy()
line_554['LINE_ID'] = 554

# Añadimos la nueva línea al GeoDataFrame
gdf = pd.concat([gdf, line_554], ignore_index=True)
gdf = gdf.sort_values(by='LINE_ID').reset_index(drop=True)
gdf[gdf['LINE_ID'] == 554]

```

	cartodb_id	LINE_ID	geometry
5267	6344	554	POINT (-57.55984 -37.96250)
5268	6352	554	POINT (-57.54483 -37.97014)
5269	6351	554	POINT (-57.54579 -37.96964)
5270	6350	554	POINT (-57.54773 -37.96862)
5271	6349	554	POINT (-57.54979 -37.96765)
...
5490	6415	554	POINT (-57.55808 -38.04439)
5491	6445	554	POINT (-57.56040 -38.01634)
5492	6379	554	POINT (-57.56538 -38.02052)
5493	6510	554	POINT (-57.56410 -38.04362)
5494	6509	554	POINT (-57.60635 -37.93903)
228 rows × 3 columns			

Ya tenemos la línea 554 con su ruta correspondiente. Vamos a trabajar la línea 715:



```
# Reemplazar el valor de LINE_ID 720 por 715
gdf.loc[gdf['LINE_ID'] == 720, 'LINE_ID'] = 715
gdf[gdf['LINE_ID'] == 715]
```

cartodb_id	LINE_ID	geometry
0	1	POINT (-57.54245 -38.00059)
1	2	POINT (-57.54245 -38.00059)
2	3	POINT (-57.54245 -38.00059)
3	4	POINT (-57.54245 -38.00059)
4	5	POINT (-57.54245 -38.00059)
...
1975	1976	POINT (-57.70437 -38.02560)
1976	1977	POINT (-57.70437 -38.02560)
1977	1978	POINT (-57.72600 -38.03202)
1978	1979	POINT (-57.72600 -38.03202)
1979	1980	POINT (-57.72600 -38.03202)
1980 rows × 3 columns		

Bien, ya tenemos consistencia entre las rutas y todas las líneas de transporte de nuestro análisis. Vamos a mensualizar la demanda de cada línea:



```
# Creamos un nuevo DataFrame agrupado por LINE_ID, MONTH y YEAR, sumando los TICKETS
df_agrupado = df.groupby(['LINE_ID', 'MONTH', 'YEAR'], as_index=False)[['TICKETS']].sum()

df_agrupado = df_agrupado.sort_values(by=['MONTH', 'YEAR', 'LINE_ID'])
order = ['MONTH', 'YEAR', 'LINE_ID', 'TICKETS']
df_agrupado = df_agrupado[order]
df_agrupado
```

	MONTH	YEAR	LINE_ID	TICKETS
0	1	2020	501	24866
60	1	2020	511	1062951
120	1	2020	512	176840
180	1	2020	521	269986
240	1	2020	522	389117
...
1379	12	2024	581	180718
1439	12	2024	591	209634
1499	12	2024	593	130151
1559	12	2024	715	193362
1619	12	2024	717	165540

1620 rows × 4 columns

Bien, para poder trabajar con el mapa, necesitamos extraer latitud y longitud en columnas separadas:

```
● ● ●

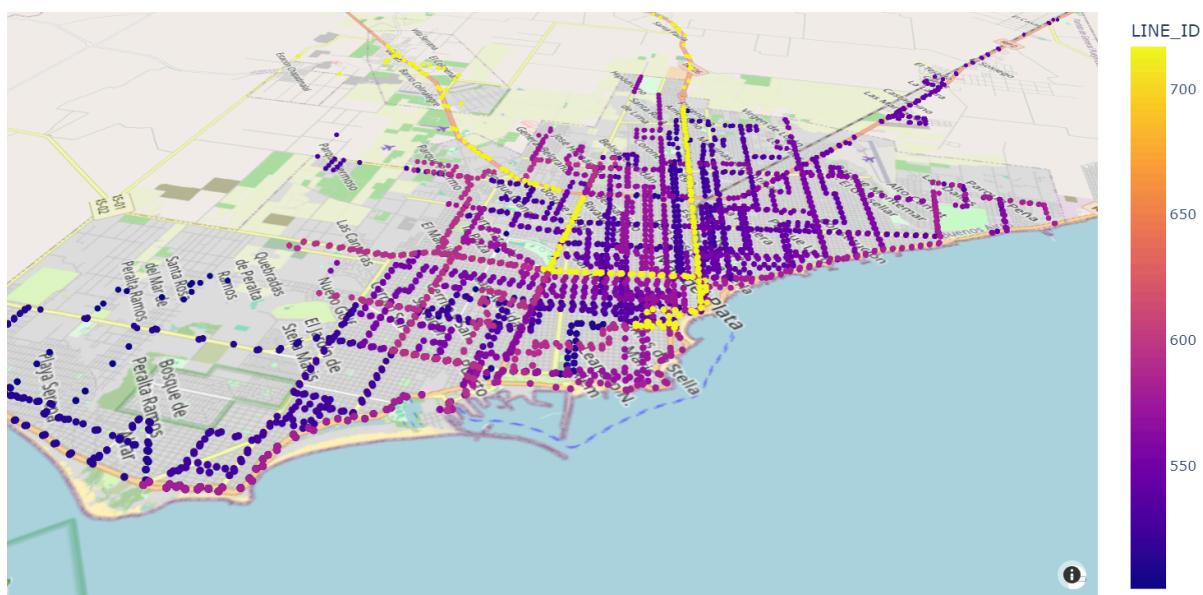
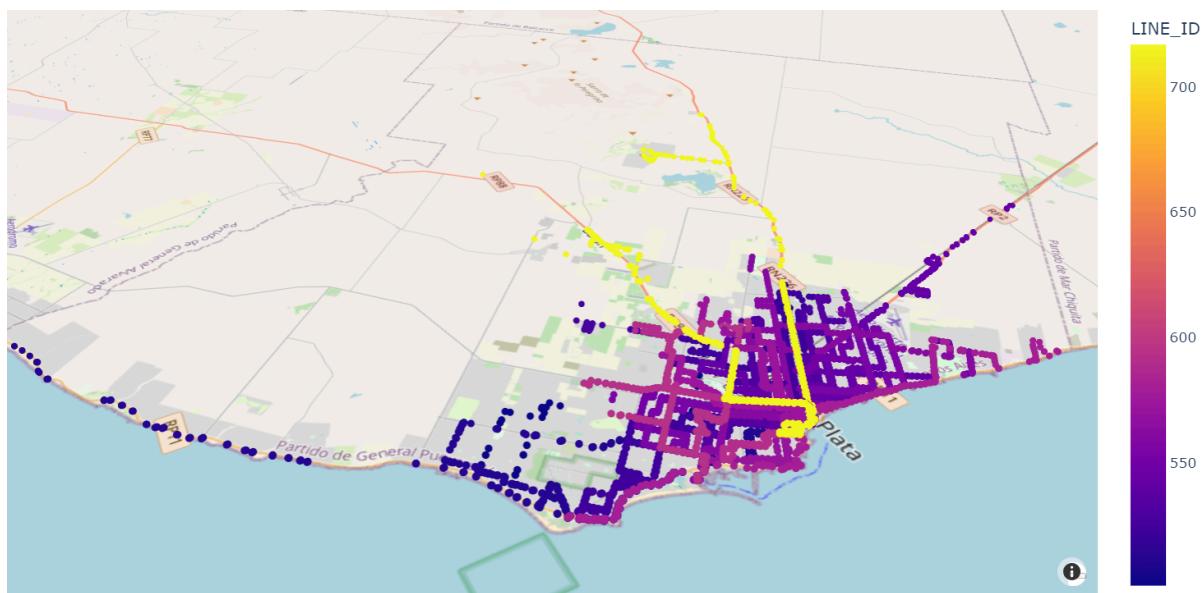
# Convertimos el GeoDF a formato compatible con Plotly
gdf['lon'] = gdf.geometry.x
gdf['lat'] = gdf.geometry.y

# Creamos el mapa usando Plotly
fig = px.scatter_mapbox(gdf, lat="lat", lon="lon", hover_name="LINE_ID", color="LINE_ID", # Usar una columna existente
                        zoom=12, height=500)

fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

fig.show()
```

Hecho esto, ya tenemos las rutas graficables en el mapa, veamos algunas muestras:



Lo que observamos es simplemente las paradas de transporte, diferenciadas por un tono de color de acuerdo a su LINE_ID.

Para tener un mapa claro y representativo de la demanda, el objetivo aquí es superponer la demanda en los recorridos, generando líneas y puntos de demanda más claros visualmente. Esto se logró homogeneizando puntos dentro de radios de 100 metros. Ejemplo: Av. Pedro Luro, tiene paradas sobre ambas manos, esos waypoints deben convertirse en un único waypoint. Este, se encargará de contener en una lista, todas las líneas únicas que por allí transiten.

```

● ● ●

gdf_copy = gdf.copy()

# Reproyectar a un CRS proyectado adecuado (por ejemplo, EPSG:3857)
gdf_copy = gdf_copy.to_crs(epsg=3857)

# Creamos un buffer de 100 metros alrededor de cada punto
gdf_copy['buffer'] = gdf_copy.geometry.buffer(100) # 100 metros

# Unimos los buffers superpuestos
unary_union_buffers = unary_union(gdf_copy['buffer'].tolist())

# Extraemos los centroides de los buffers unidos
centroids = [geom.centroid for geom in unary_union_buffers.geoms]

# Creamos nuevo GeoDF con los centroides
centroids_gdf = gpd.GeoDataFrame(geometry=centroids, crs=gdf_copy.crs)

# Inicializar listas para almacenar los puntos combinados y los LINE_ID que pasan por allí
combined_points = []
combined_line_ids = []

# Iteraremos sobre los centroides y agrupamos los puntos originales
for centroid in centroids_gdf.geometry:
    # Seleccionamos los puntos que están dentro del buffer de 100 metros del centroide
    points_within_buffer = gdf_copy[gdf_copy.geometry.within(centroid.buffer(100))]
    # Obtenemos los LINE_ID que pasan por el buffer y asegurarse de que sean únicos
    line_ids = list(set(points_within_buffer['LINE_ID'].tolist()))
    combined_points.append(centroid)
    combined_line_ids.append(line_ids)

# Creamos un nuevo GeoDF con los puntos combinados y los LINE_ID asociados
combined_gdf = gpd.GeoDataFrame({'geometry': combined_points, 'LINE_ID': combined_line_ids}, crs=gdf_copy.crs)

# Reproyectar de nuevo al CRS original
combined_gdf = combined_gdf.to_crs(epsg=4326)

# Imprimir y evaluamos la estructura del nuevo GeoDF
print(combined_gdf.head())

# Creamos un DF plano para Plotly
plot_data = []
for idx, row in combined_gdf.iterrows():
    plot_data.append({
        'lon': row['geometry'].x,
        'lat': row['geometry'].y,
        # Convertimos la lista de LINE_ID a una cadena separada por comas
        'LINE_ID': ', '.join(map(str, row['LINE_ID']))
    })

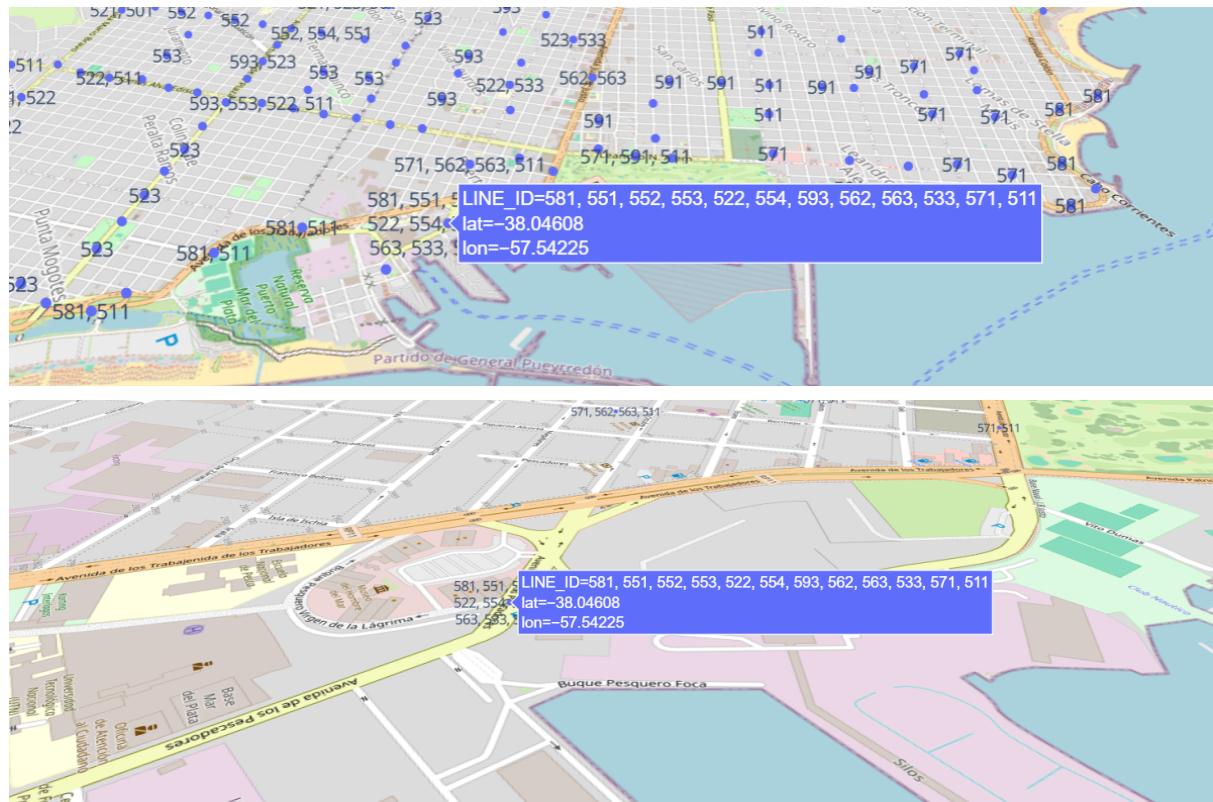
plot_df = pd.DataFrame(plot_data)

# Crear el mapa usando Plotly
fig = px.scatter_mapbox(plot_df, lat="lat", lon="lon", zoom=11, height=500, text='LINE_ID')

fig.update_layout(
    mapbox_style="open-street-map",
    mapbox_center={"lat": -38.077591762587275, "lon": -57.59587897840447},
    mapbox_zoom=11,
    mapbox_bearing=-61.600000000000136,
    mapbox_pitch=58.49999999999982,
    margin={"r":0, "t":0, "l":0, "b":0}
)
fig.show()

```

El resultado es el esperado, tenemos los puntos, que fueron unificados por cercanía, y contienen todas las líneas que por allí pasan:



Ahora nos resta generar las imágenes, para esto vamos sumar los consumos en los puntos combinados a fin de que puedan ser representados. Cada mes va a tener una imagen, que luego se grabará a un video:

```
# Inicializar la lista para almacenar las imágenes
images = []

# Obtener el valor máximo de tickets considerando todas las superposiciones
max_tickets = 4000000

# Configuración del mapa
map_config = {
    'mapbox.center': {'lon': -57.56430778285193, 'lat': -37.997453317564485},
    'mapbox.zoom': 11.5,
    'mapbox.bearing': -65.60000000000026,
    'mapbox.pitch': 0,
    'mapbox._derived': {
        'coordinates': [
            [-57.731682295106765, -38.12437902923855],
            [-57.55326248178028, -37.814306569090036],
            [-57.39693327059726, -37.870307567383705],
            [-57.57535308392373, -38.180143858646154]
        ]
    }
}
```

```

● ● ●

# Iterar sobre cada año y mes
for year in range(2020, 2025): # [y, y+1]
    for month in range(1, 13):
        # Filtrar los datos para el año y mes actual
        df_month = df_agrupado[(df_agrupado['YEAR'] == year) & (df_agrupado['MONTH'] == month)]

        # Crear una copia del GeoDataFrame combinado para cada mes
        combined_gdf_copy = combined_gdf.copy()

        # Inicializar una lista para almacenar los tickets sumados para cada punto
        combined_gdf_copy['TICKETS'] = 0

        # Sumar los tickets en los puntos combinados
        for idx, row in combined_gdf_copy.iterrows():
            total_tickets = df_month[df_month['LINE_ID'].isin(row['LINE_ID'])]['TICKETS'].sum()
            combined_gdf_copy.at[idx, 'TICKETS'] = total_tickets

        # Crear un DataFrame plano para Plotly
        plot_data = []
        for idx, row in combined_gdf_copy.iterrows():
            plot_data.append({
                'lon': row['geometry'].x,
                'lat': row['geometry'].y,
                'tickets': row['TICKETS']
            })

        plot_df = pd.DataFrame(plot_data)

        # Crear el mapa usando Plotly con la configuración obtenida
        fig = px.scatter_mapbox(plot_df, lat="lat", lon="lon", hover_name="tickets", color="tickets", size="tickets", # Definir el tamaño de los puntos
                               size_max=37, # Ajustar el tamaño máximo de los puntos en pixeles
                               color_continuous_scale="Portland", zoom=map_config['mapbox.zoom'], height=1080, width=1920,
                               range_color=[0, max_tickets])

        fig.update_layout(
            mapbox_style="open-street-map",
            mapbox_center=map_config['mapbox.center'],
            mapbox_zoom=map_config['mapbox.zoom'],
            mapbox_bearing=map_config['mapbox.bearing'],
            mapbox_pitch=map_config['mapbox.pitch'],
            margin={"": 0, "t": 0, "l": 0, "b": 0},
            coloraxis_colorbar=dict(
                thickness=10, # Cambiar el grosor del colorbar
                len=0.3, # Cambiar la longitud del colorbar
                anchor="top",
                y=1,
                xanchor="left",
                x=0,
                outlineWidth=0, # Eliminar el borde del colorbar
                bgcolor='rgba(0,0,0,0)' # Hacer el fondo del colorbar transparente
            ),
            annotations=[{
                "x": 0.5,
                "y": 0.1,
                "xref": "paper",
                "yref": "paper",
                "text": f'{pd.to_datetime(year*10000 + month*100 + 1, format='%Y%m%d').strftime('%B %Y')}',
                "showarrow": False,
                "font": {"size": 20}
            }]
        )

        # Guardar la imagen del mapa en un archivo temporal
        image_filename = f'mapas/map_{year}_{month}.png'
        fig.write_image(image_filename)

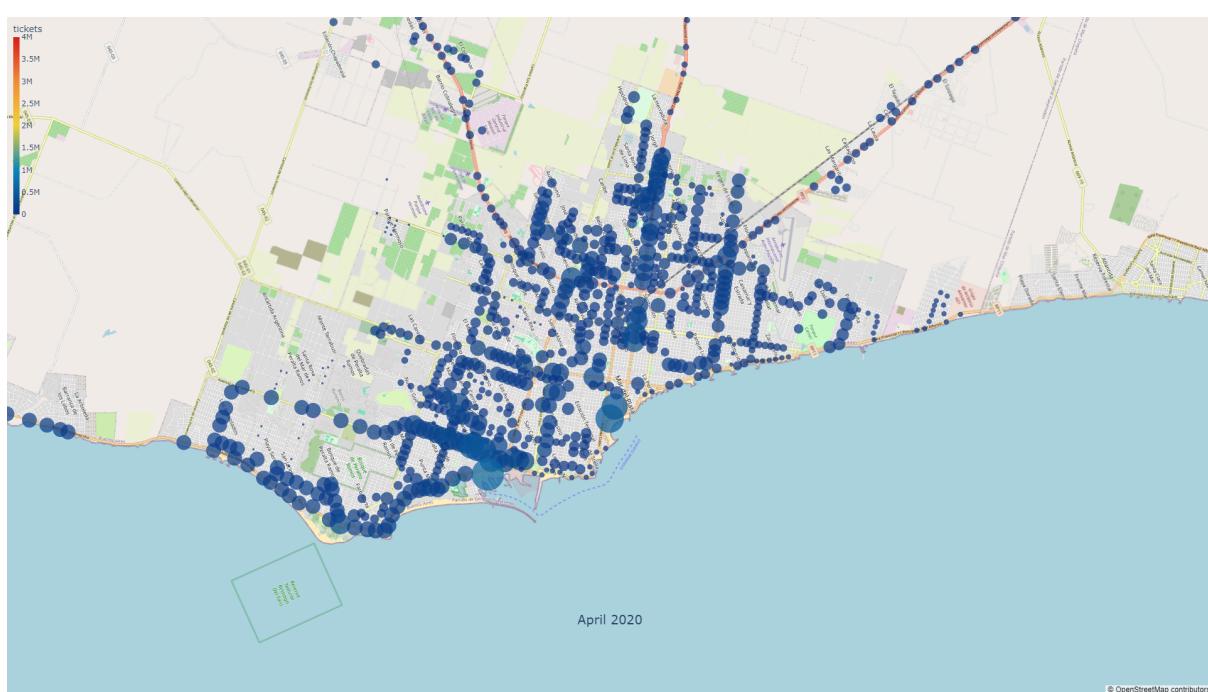
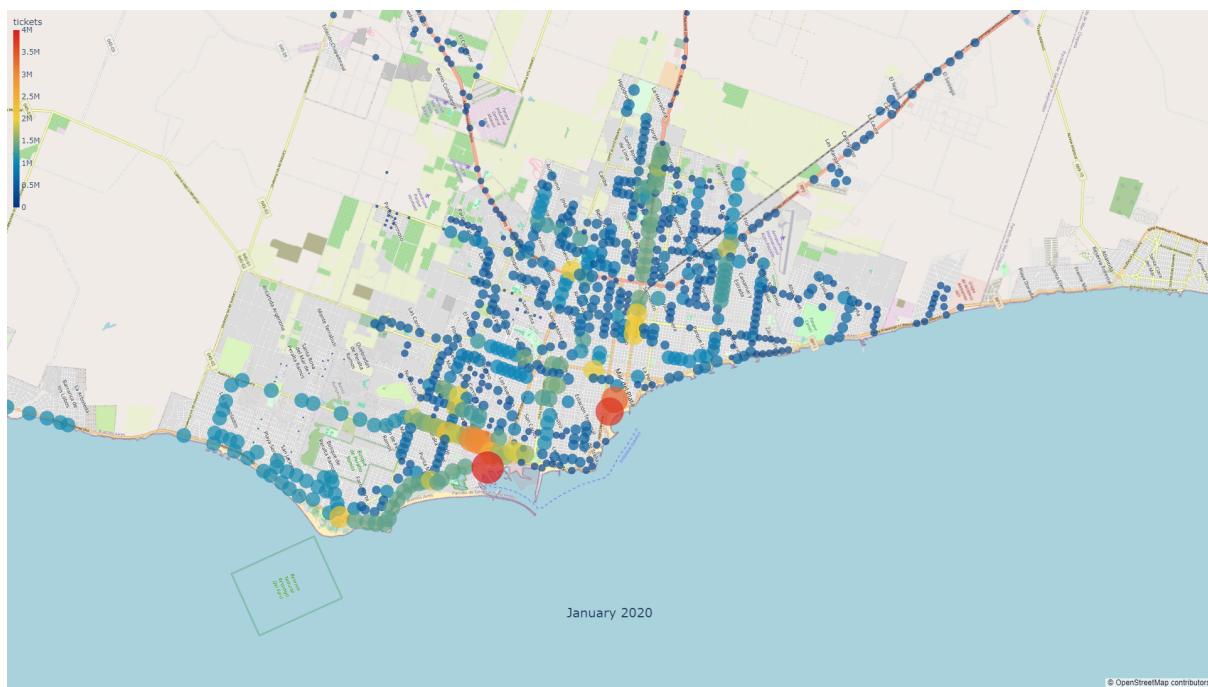
        # Leer la imagen y agregarla a la lista de imágenes
        images.append(imageio.imread(image_filename))

    # Crear el video a partir de las imágenes
    imageio.mimsave("mapa.mp4", images, fps=2, codec="libx264")

    print("Video generado: mapa.mp4")

```

El código va a generar las imágenes para cada mes, tal como observamos:



Como podemos observar en las imágenes, ahora podemos apreciar la demanda por zonas específicas. El video que incluye todo el análisis se encuentra en la carpeta del proyecto.

Análisis y conclusiones

Patrón estacional de demanda: El gráfico revela un patrón estacional evidente en todos los años a lo largo de los años. La demanda es más alta al comienzo del año (enero y febrero) y nuevamente hacia el final del año (noviembre y diciembre). Esta tendencia confirma que los factores estacionales, como las vacaciones de verano y el turismo, y las fiestas de fin de

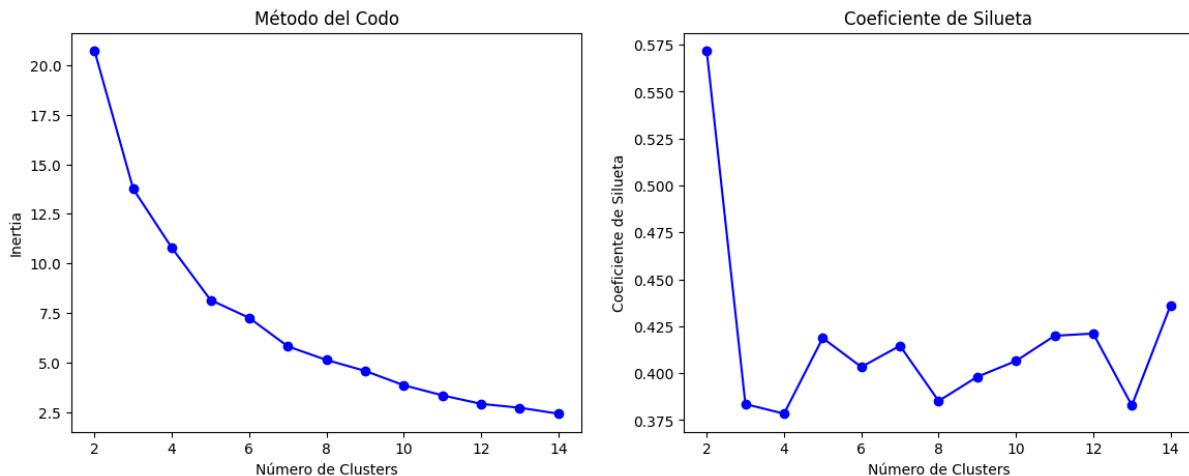
año, influyen significativamente en el uso del transporte público. Cabe aclarar nuevamente, el impacto que tuvo la pandemia sobre marzo de 2020 en adelante. Inclusive el nivel más alto registrado se da justo en enero de 2020, que hasta el momento nunca se ha igualado.

Potenciales implicaciones y recomendaciones

Planificación de rutas y recursos: La alta demanda en las dos zonas céntricas podrían indicarnos que son las zonas más demandantes durante todo el año. Aunque en verano vemos como las zonas de la costa (desde el centro hacia el sur) tienden a escalar. Las zonas alejadas de Mar del Plata (Sierra de los padres) mantienen un consumo constante, que no se eleva en verano.

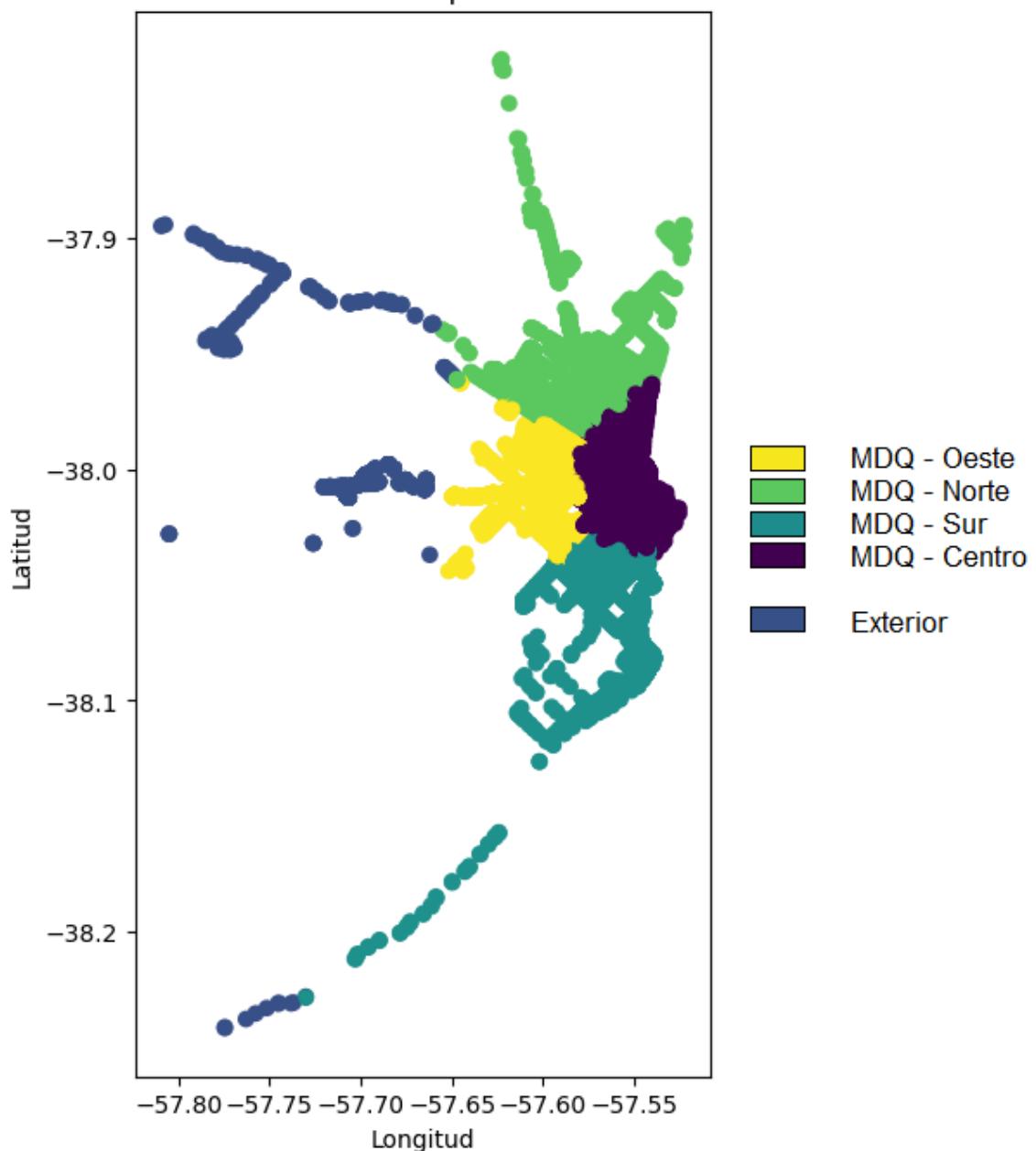
Investigación adicional

Como último paso, propongo evaluar estas conclusiones haciendo un agrupamiento por zonas, para esto tomamos el datasets con los puntos y hacemos una clusterización, observamos que podría estar bien explicado en 4 o 5 zonas:

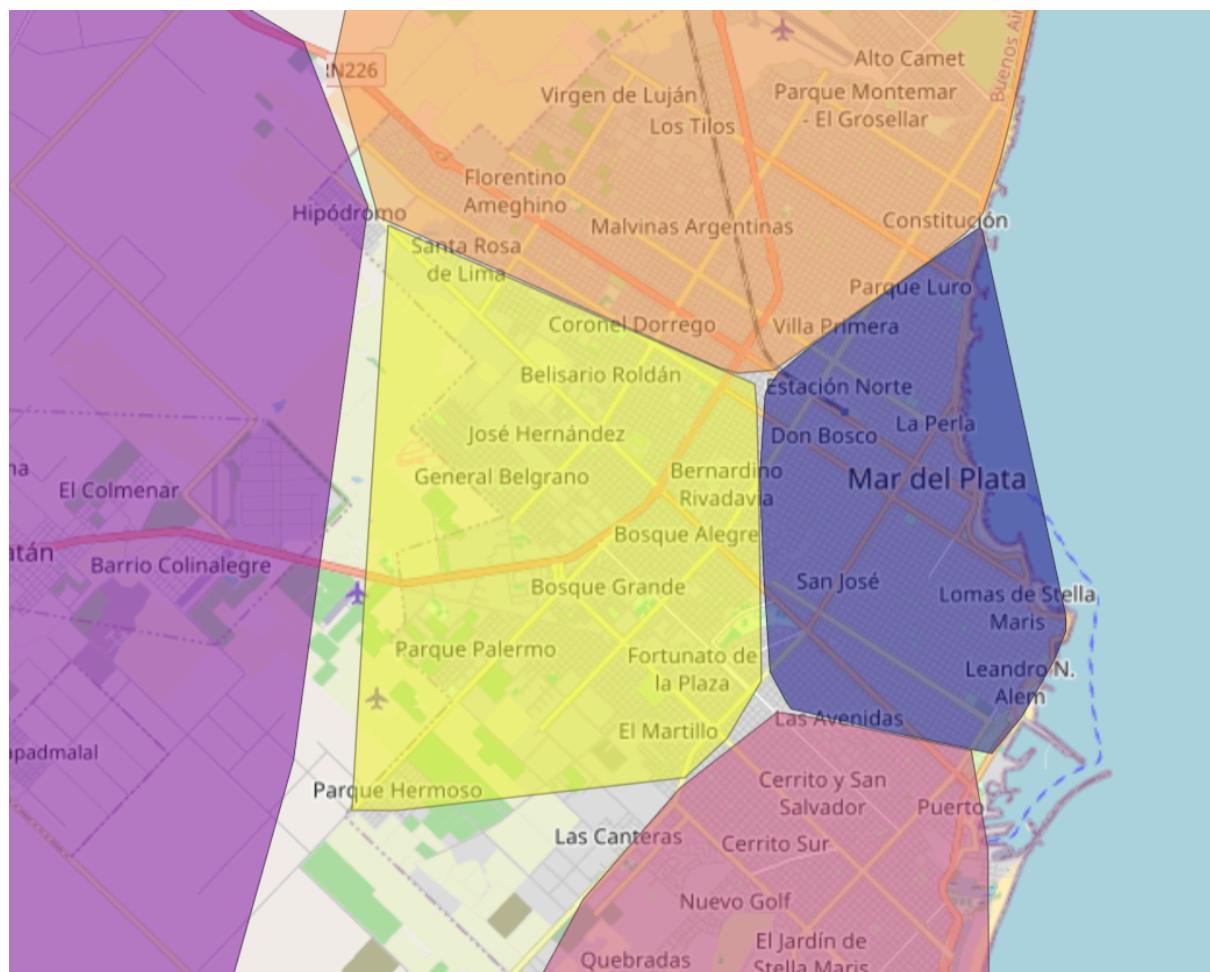


Por la naturaleza del problema vamos a optar por 5 zonas principales:

MDP - Zonas de transporte - Clusters: 5

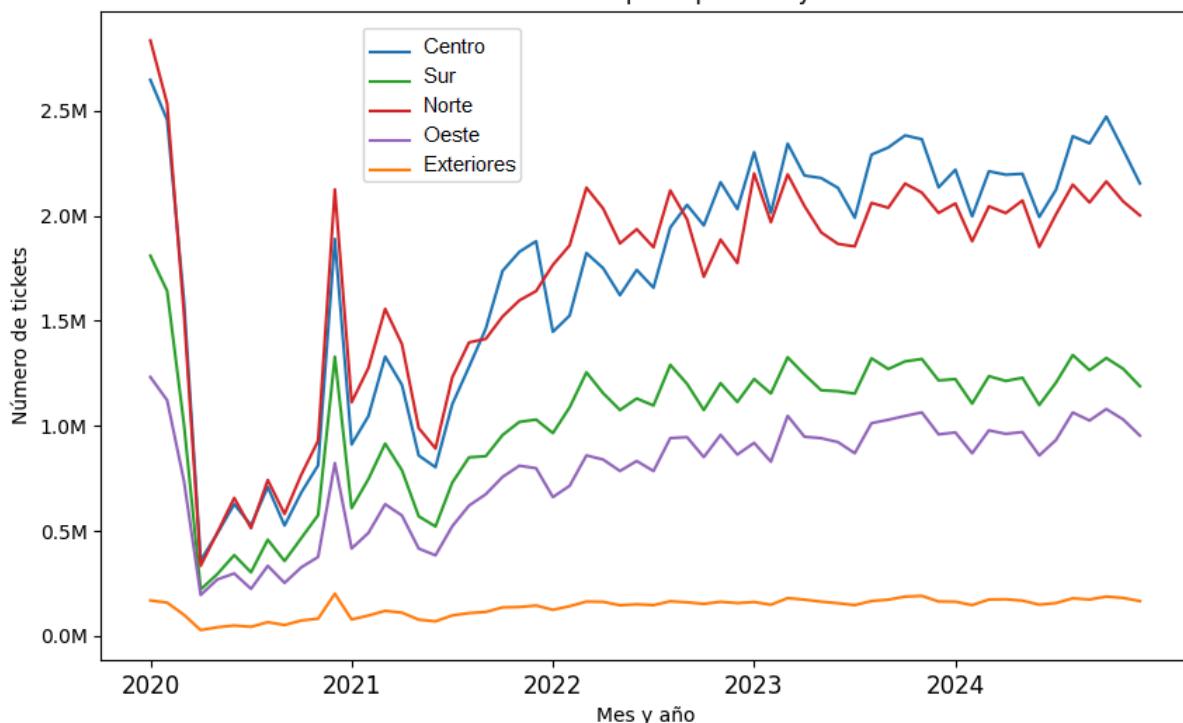


Para comprender más de cerca las divisiones veamos un mapa con polígonos delimitando los clusters:



Ya tenemos las zonas, ahora vamos a contrastar la demanda y veamos el gráfico resultante:

Análisis de demanda de transporte por mes y año - Clusters: 5



Evolución temporal de la demanda

Impacto de la pandemia y recuperación post-pandemia:

Se observa una caída significativa en la demanda en todas las zonas al inicio de 2020, coincidiendo con el comienzo de las restricciones.

La recuperación de la demanda ha sido lenta y desigual, sin llegar a alcanzar los niveles pre-pandemia hasta el momento en las zonas comprendidas en Mar del Plata, no así en zonas exteriores.

Estacionalidad de la demanda:

Verano: La demanda incrementa notablemente durante los meses de verano (Mayormente en diciembre y febrero), especialmente en las zonas Centro y Norte, reflejando la afluencia turística. Cabe aclarar que la mayoría de las playas están situadas sobre estas áreas.

Ciclo lectivo escolar: Notamos un periodo de alza en el consumo sobre marzo, que podría estar relacionado con el comienzo del periodo escolar.

Zonas de mayor demanda:

Centro: Muestra la demanda más alta y constante, indicativo de la importancia del área para actividades comerciales y administrativas.

Norte: También presenta altos niveles de demanda, seguramente asociado a su atractivo turístico.

Sur y oeste: Demuestran una demanda moderada, reflejando su naturaleza residencial e industrial.

Exteriores: Menor demanda pero consistente en el tiempo, reflejando su carácter periférico y posiblemente menor densidad poblacional.

Conclusión

El análisis de la demanda de transporte en Mar del Plata muestra una recuperación gradual post-pandemia, aunque los niveles de consumo aún no han alcanzado los valores pre-2020. Las zonas Centro y Norte representan los principales focos de demanda, impulsados por la actividad comercial y turística, mientras que las zonas Sur y Oeste tienen un comportamiento más estable, con una demanda moderada. Las áreas periféricas (Exteriores) presentan una menor demanda pero más estable estacionalmente hablando.

Este análisis sugiere que para optimizar el servicio de transporte público, es crucial considerar la estacionalidad y las características específicas de cada zona, implementando mejoras en infraestructura y políticas que respondan a las necesidades cambiantes de los ciudadanos.

Próximos pasos

Sería beneficioso realizar estudios adicionales para comprender mejor los factores específicos que influyen en la variabilidad de la demanda en cada zona.

Factores como la ubicación de escuelas, universidades, áreas comerciales, o eventos específicos podrían proporcionar insights valiosos para un análisis más granular y efectivo del transporte.

Además sería interesante poder contar con información de otras líneas como la 221, que no opera con el sistema SUBE y funciona solo con efectivo, pero que su recorrido traslada a los marplatenses de un extremo a otro por la línea costera.

Y por último estudiar y entender cómo ha cambiado la vida laboral, estudiantil, y recreativa de los ciudadanos para comprender por qué no hemos recuperado los niveles de demanda de transporte ubicados previamente a la pandemia de 2020.

Muchas gracias,
Braian A. D'Aleo

Referencias

Sobre el origen de los datos:

Datos de la Ciudad de Mar del Plata. (n.d.). Sube - Transacciones diarias. Datos Mar del Plata. Recuperado de

<https://datos.mardelplata.gob.ar/?q=dataset/sube-transacciones-diarias>

Datos de la Ciudad de Mar del Plata. (n.d.). Pasajeros de transporte urbano. Datos Mar del Plata. Recuperado de

<https://datos.mardelplata.gob.ar/?q=dataset/transporte-p%C3%BAblico-de-pasajeros>

Datos de la Ciudad de Mar del Plata. (2021). Pasajes urbanos. Datos Mar del Plata. Recuperado de

<https://datos.mardelplata.gob.ar/?q=dataset/transporte-p%C3%BAblico-de-pasajeros/resource/dce1e822-c9a3-4829-ba3a-43194ad4332e>

Datos de la Ciudad de Mar del Plata. (n.d.). Paradas de colectivo. Datos Mar del Plata.

Recuperado de <https://datos.mardelplata.gob.ar/?q=dataset/paradas-de-colectivo>

Secretaría de Transporte. (n.d.). Uso de SUBE. Datos Transporte. Recuperado de

<https://datos.transporte.gob.ar/dataset/sube-cantidad-de-transacciones-usos-por-fecha>

Datos abiertos, Argentina. (n.d.). Cantidad de transacciones usos por fecha. Datos Argentina. Recuperado de

<https://datos.gob.ar/ro/dataset/transporte-sube---cantidad-transacciones-usos-por-fecha>

Referencias adicionales:

MeteoStat. (n.d.). Dataset de clima. MeteoStat. Recuperado de

<https://meteostat.net/es/place/ar/mar-del-plata?s=87692&t=2020-01-01/2023-12-31>

Mulla, R. (n.d.). Time series forecasting with machine learning. Kaggle. Recuperado de

<https://www.kaggle.com/code/robikscube/time-series-forecasting-with-machine-learning-yt>

Kanaries. (n.d.). XGBoost: la potencia de los algoritmos de aprendizaje automático.

Kanaries Docs. Recuperado de <https://docs.kanaries.net/es/topics/Python/xgboost>

XGBoost Developers. (n.d.). XGBoost documentation. XGBoost. Recuperado de

<https://xgboost.readthedocs.io/en/stable/>

Databricks. (2022, enero 18). Introduction to time series forecasting [Video]. YouTube.

https://www.youtube.com/watch?v=rcdDI8qf0ZA&ab_channel=Databricks

PyData. (2021, septiembre 29). Forecasting time series with scikit-learn and XGBoost [Video]. YouTube.

https://www.youtube.com/watch?v=9QtL7m3YS9I&t=2101s&ab_channel=PyData

Codebasics. (2021, octubre 20). Time series forecasting using XGBoost | Kaggle tutorial [Video]. YouTube.

https://www.youtube.com/watch?v=0lsmdNLNorY&ab_channel=codebasics