```
import pandas as pd
```

Carga de datos: Subimos el archivo flights.csv a Google Colab y lo cargamos en un DataFrame de pandas.

```
datos = pd.read_csv('/content/drive/MyDrive/A aumentada: previsión de atrasos de vuelos/flights.csv')
datos.shape
```

```
(71175, 11)
```

Exploración inicial: Utilizamos head() para ver las primeras filas y familiarizarnos con las columnas, que incluyen información como la aerolínea, el tipo de aeronave, el origen, los horarios y el retraso.

```
datos.head()
```

	flight_id	airline	aircraft_type	schengen	origin	arrival_time	departure_time	day	year	is_holiday	delay
0	26	MM	Airbus A320	non-schengen	TCY	8.885071	10.885071	0	2010	False	70.205981
1	10	YE	Airbus A320	non-schengen	TCY	8.939996	11.939996	0	2010	False	38.484609
2	3	BZ	Embraer E175	schengen	TZF	18.635384	22.635384	0	2010	False	2.388305
3	28	BZ	Airbus A330	non-schengen	EMM	15.967963	17.967963	0	2010	False	19.138491
4	15	BZ	Airbus A330	non-schengen	FJB	16.571894	19.571894	0	2010	False	15.016271

Conociendo los metadatos

Metadatos con info(): Revisamos los metadatos para conocer el tipo de datos de cada columna y verificar si hay valores nulos (en este caso, no los hay).

```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71175 entries, 0 to 71174
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   flight_id       71175 non-null  int64
 1   airline         71175 non-null  object
 2   aircraft_type   71175 non-null  object
 3   schengen        71175 non-null  object
 4   origin          71175 non-null  object
 5   arrival_time    71175 non-null  float64
 6   departure_time  71175 non-null  float64
 7   day             71175 non-null  int64
 8   year            71175 non-null  int64
 9   is_holiday      71175 non-null  bool
 10  delay           71175 non-null  float64
dtypes: bool(1), float64(3), int64(3), object(4)
memory usage: 5.5+ MB
```

Data columns (total 11 columns): Column Non-Null Count Dtype

---

0 flight_id 71175 non-null int64
1 airline 71175 non-null object 2 aircraft_type 71175 non-null object 3 schengen 71175 non-null object 4 origin 71175 non-null object 5 arrival_time 71175 non-null float64 6 departure_time 71175 non-null float64 7 day 71175 non-null int64
8 year 71175 non-null int64
9 is_holiday 71175 non-null bool
10 delay 71175 non-null float64 dtypes: bool(1), float64(3), int64(3), object(4) memory usage: 5.5+ MB

Estadísticas descriptivas con describe(): Analizamos las estadísticas de las variables numéricas, como el promedio y la desviación estándar del retraso, así como los cuartiles para entender la distribución de los retrasos. También incluimos include="O" para obtener estadísticas de las variables categóricas, como las aerolíneas y los aeropuertos de origen.

```
datos.describe()
```

	flight_id	arrival_time	departure_time	day	year	delay
count	71175.000000	71175.000000	71175.000000	71175.000000	71175.000000	71175.000000
mean	15.465135	13.283159	16.480222	182.000000	2016.000000	12.548378
std	8.649646	4.023380	4.143705	105.366769	3.741684	23.125349
min	1.000000	7.065594	10.065594	0.000000	2010.000000	-41.028033
25%	8.000000	8.939996	12.668655	91.000000	2013.000000	-4.412876
50%	15.000000	14.258911	16.376052	182.000000	2016.000000	9.740454
75%	23.000000	16.909690	20.041281	273.000000	2019.000000	27.650853
max	30.000000	19.341235	23.341235	364.000000	2022.000000	125.632352

tomaremos como referencia, el retraso (delay)

```
datos.describe(include='O')
```

	airline	aircraft_type	schengen	origin
count	71175	71175	71175	71175
unique	3	6	2	10
top	BZ	Airbus A320	schengen	TZF
freq	47598	30778	42569	14162

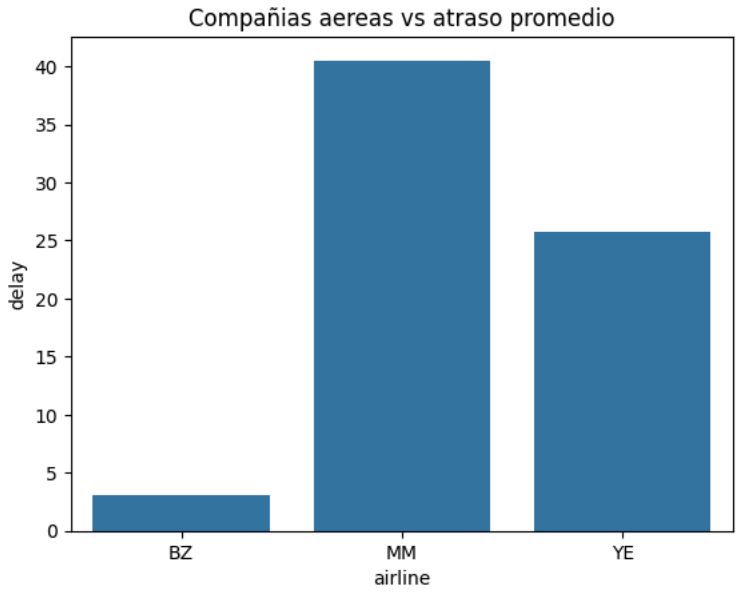
análisis visual del conjunto

se realiza un análisis visual del conjunto de datos para entender mejor los atrasos de vuelos. Se utilizan las bibliotecas seaborn y matplotlib.pyplot para crear gráficos que muestran:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

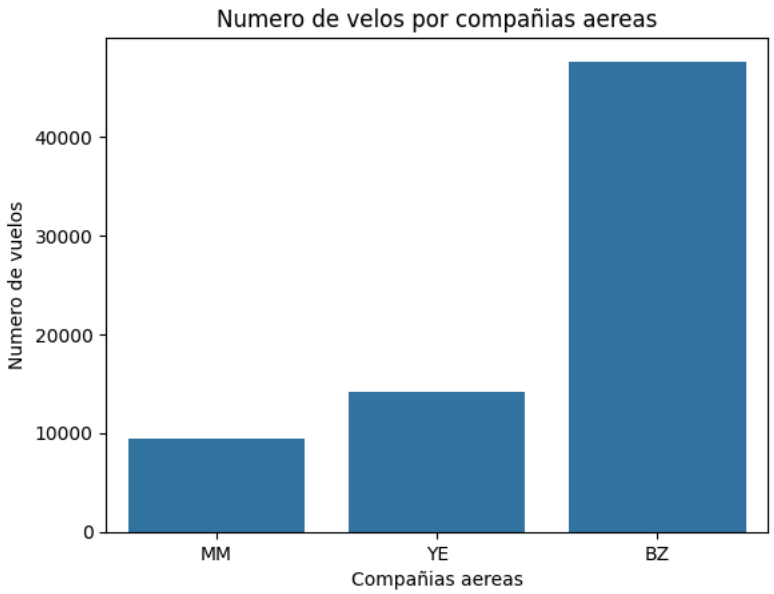
Atraso promedio por aerolínea: Se identifica qué aerolíneas tienen mayores retrasos promedio.

```
avg_delay = datos.groupby('airline')['delay'].mean().reset_index()
sns.barplot(x='airline', y='delay', data=avg_delay)
plt.title('Compañias aereas vs atraso promedio')
plt.show()
```



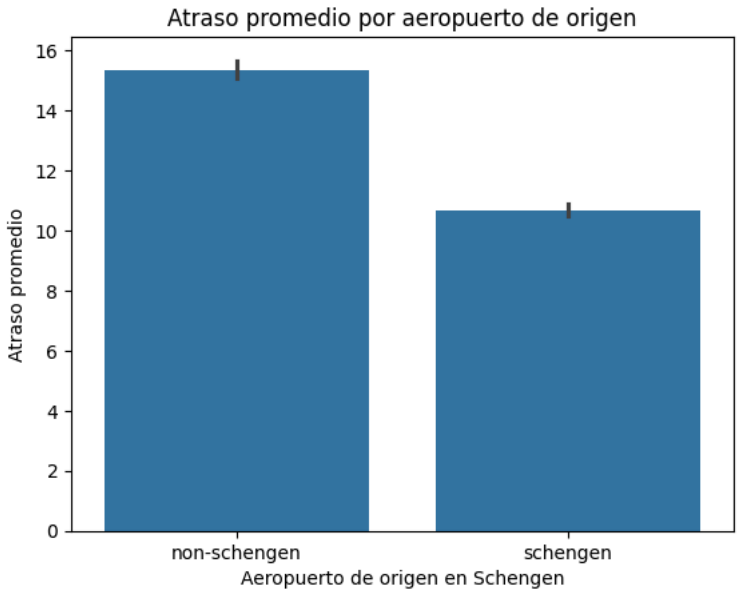
Número de vuelos por aerolínea: Se observa qué aerolíneas operan más vuelos.

```
sns.countplot(x='airline', data=datos)
plt.title('Numero de velos por compañias aereas')
plt.ylabel('Numero de vuelos')
plt.xlabel('Compañias aereas')
plt.show()
```



Atraso promedio según si el aeropuerto de origen está en el espacio Schengen o no: Se compara el retraso entre vuelos Schengen y no Schengen.

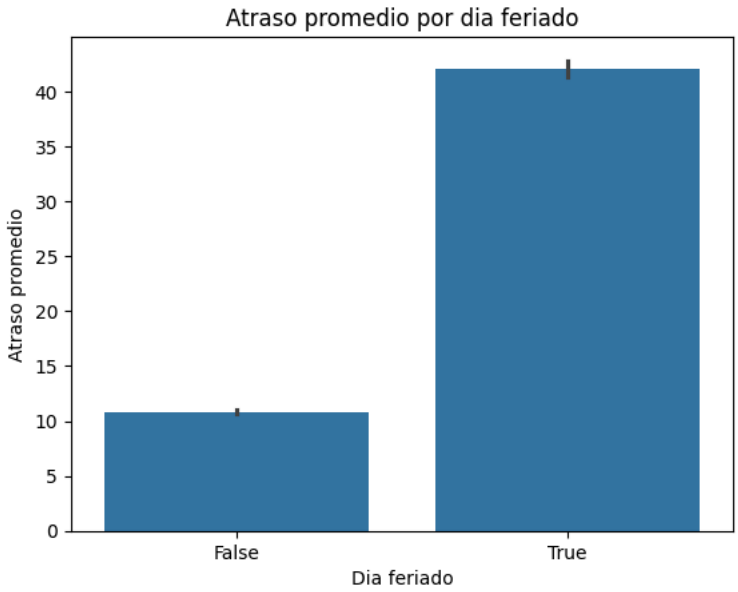
```
sns.barplot(x='schengen', y='delay', data=datos)
plt.xlabel('Aeropuerto de origen en Schengen')
plt.ylabel('Atraso promedio')
plt.title('Atraso promedio por aeropuerto de origen')
plt.show()
```

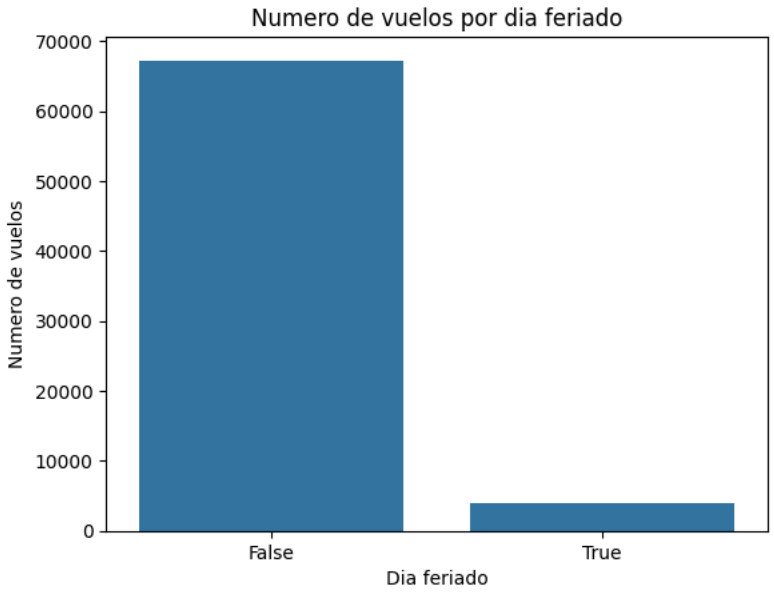


Atraso promedio en días feriados: Se analiza si los días feriados influyen en los retrasos.

Haz doble clic (o pulsa Intro) para editar

```
sns.barplot(x='is_holiday', y='delay', data=datos)
plt.xlabel('Dia feriado')
plt.ylabel('Atraso promedio')
plt.title('Atraso promedio por dia feriado')
plt.show()


sns.countplot(x='is_holiday', data=datos)
plt.xlabel('Dia feriado')
plt.ylabel('Numero de vuelos')
plt.title('Numero de vuelos por dia feriado')
plt.show()
```



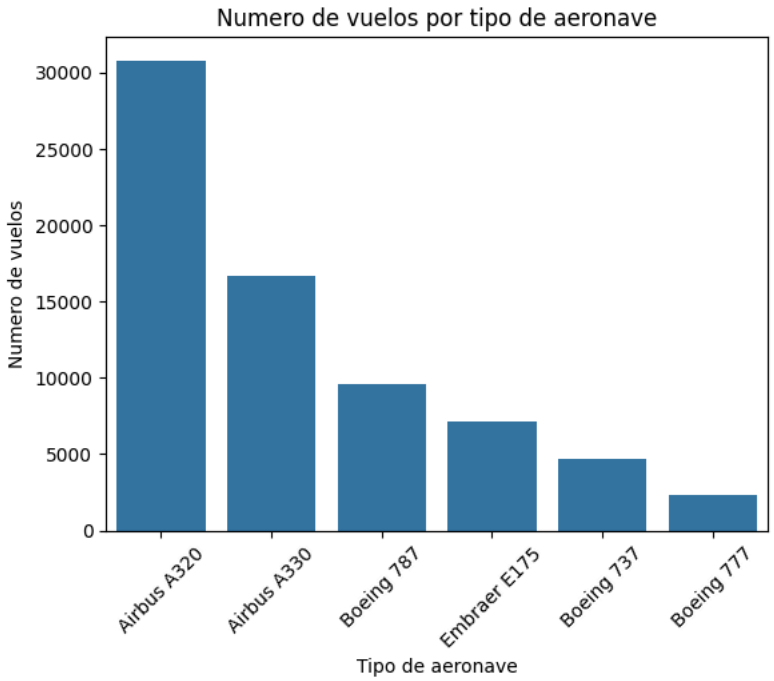


Cantidad de vuelos por tipo de aeronave: Se identifica qué tipos de aeronaves son más comunes en el aeropuerto.

```
order = datos['aircraft_type'].value_counts().index
order
```

```
Index(['Airbus A320', 'Airbus A330', 'Boeing 787', 'Embraer E175',
       'Boeing 737', 'Boeing 777'],
      dtype='object', name='aircraft_type')
```

```
sns.countplot(x='aircraft_type', data=datos, order=order)
plt.xlabel('Tipo de aeronave')
plt.ylabel('Numero de vuelos')
plt.xticks(rotation=45)
plt.title('Numero de vuelos por tipo de aeronave')
plt.show()
```

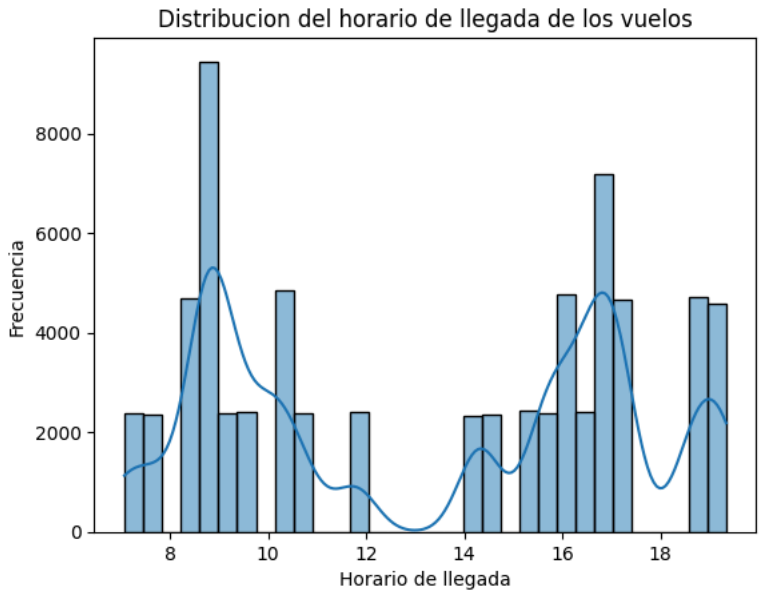


Histogramas:

Se utilizan para analizar la frecuencia de diferentes valores, como el horario de llegada y salida de los vuelos, y principalmente el atraso en minutos. Se crea un histograma utilizando sns.histplot con una curva de densidad (kde=True). Se introduce la regla de Friedman-Diaconis para establecer el ancho adecuado de los bins en un histograma. Se aplica la función para calcular el ancho de los bins y se compara con el uso manual de un número fijo de bins.

Scott's rule

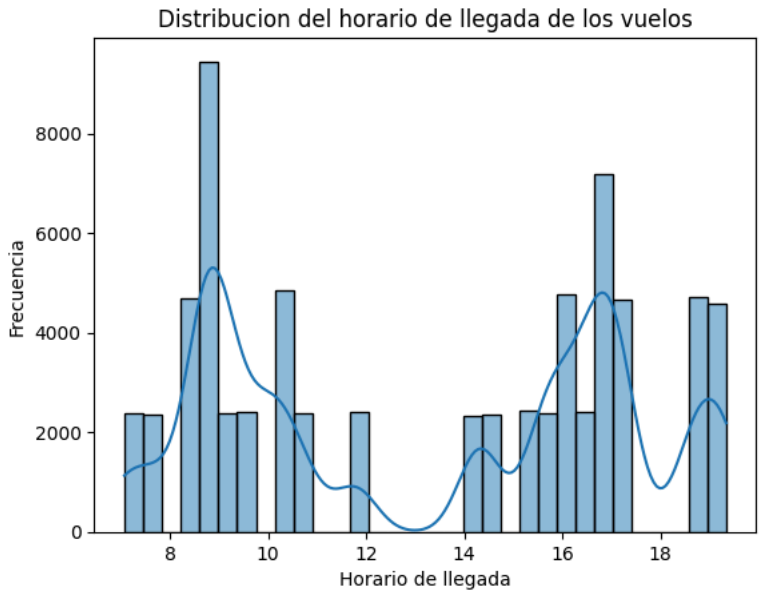
```
sns.histplot(datos['arrival_time'], kde=True)
plt.xlabel('Horario de llegada')
plt.ylabel('Frecuencia')
plt.title('Distribucion del horario de llegada de los vuelos')
plt.show()
```




```
import numpy as np
def ancho_bin(df,columna):
  q75,q25 = np.percentile(df[columna],[75,25])
```

```
    iqr = q75 - q25
    n = len(df[columna])
    return 2*iqr*(n**(-1/3))
```

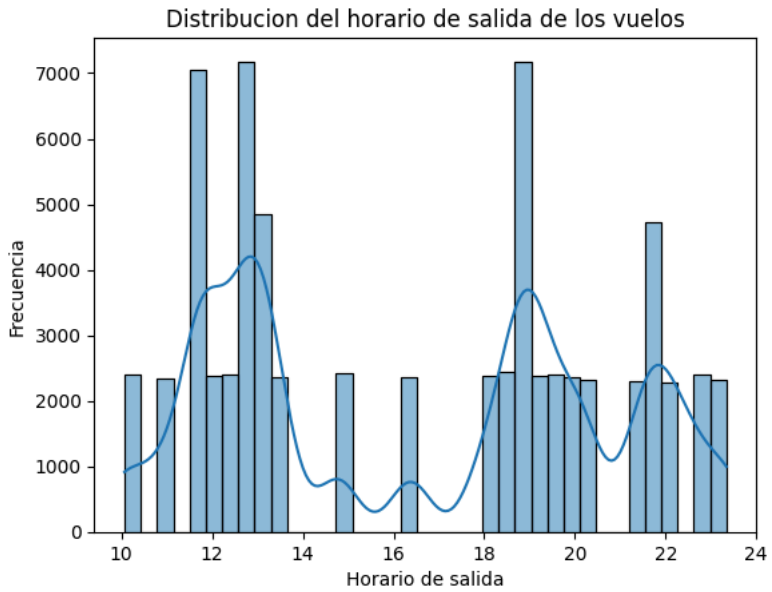
```
bin_width = ancho_bin(datos,'arrival_time')
sns.histplot(datos['arrival_time'], kde=True, binwidth=bin_width )
plt.xlabel('Horario de llegada')
plt.ylabel('Frecuencia')
plt.title('Distribucion del horario de llegada de los vuelos')
plt.show()
```



Análisis de variables:

Se aplica el histograma al horario de llegada y salida de los vuelos, observando las diferencias en la distribución. Se analiza la distribución de la variable respuesta, que son los retrasos en los vuelos, calculando el promedio y la mediana del atraso.

```
bin_width = ancho_bin(datos,'departure_time')
sns.histplot(datos['departure_time'], kde=True, binwidth=bin_width )
plt.xlabel('Horario de salida')
plt.ylabel('Frecuencia')
plt.title('Distribucion del horario de salida de los vuelos')
plt.show()
```



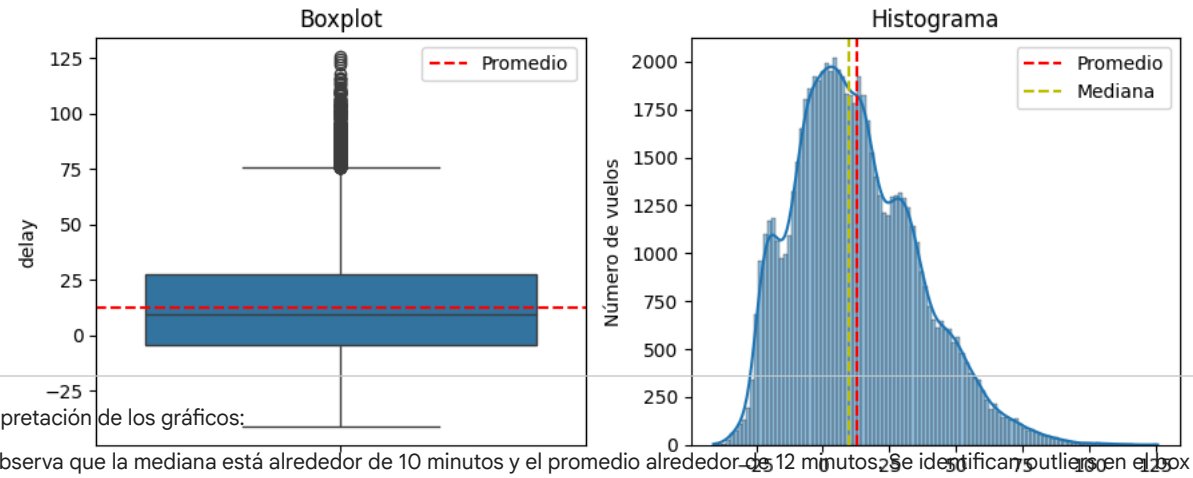
```
atraso_promedio = datos['delay'].mean()
atraso_mediana = datos['delay'].median()
print(f"Atraso promedio: {atraso_promedio}")
print(f"Atraso mediana: {atraso_mediana}")
```

```
Atraso promedio: 12.548378015698628
Atraso mediana: 9.740453855590491
```

Box plots e histogramas:

Se crean dos gráficos lado a lado: un box plot y un histograma, para visualizar la distribución de los retrasos. El box plot muestra el rango intercuartil, la mediana, los valores mínimo y máximo, y las observaciones atípicas (outliers). El histograma muestra cómo están distribuidos los atrasos y se ajusta el ancho de los bins utilizando el valor calculado previamente. Se añaden líneas verticales para indicar el promedio y la mediana en el histograma.

```
atraso_promedio = datos['delay'].mean()
atraso_mediana = datos['delay'].median()

fig, ax = plt.subplots(1,2,figsize=(9,4))

sns.boxplot(data=datos, y='delay',ax=ax[0])
ax[0].set_title('Boxplot')
ax[0].axhline(y=atraso_promedio, color='r', linestyle='--', label='Promedio')
ax[0].legend()

binwidth = ancho_bin(datos, 'delay')
sns.histplot(data=datos, x='delay', ax=ax[1], kde=True, binwidth=binwidth)
plt.ylabel('Número de vuelos')
plt.grid(False)
ax[1].set_title('Histograma')
ax[1].axvline(x=atraso_promedio, color='r', linestyle='--', label='Promedio')
ax[1].axvline(x=atraso_mediana, color='y', linestyle='--', label='Mediana')
ax[1].legend()

plt.tight_layout()

plt.show()
```



Interpretación de los gráficos:

Se observa que la mediana está alrededor de 10 minutos y el promedio alrededor de 12 minutos. Se identifican outliers en el box plot, con un valor máximo de 125 minutos de atraso. Se concluye que la distribución de los retrasos es cercana a la normal, lo que permite avanzar al modelado con machine learning.

Clase 2

```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71175 entries, 0 to 71174
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   flight_id       71175 non-null  int64
```

```
 1   airline         71175 non-null  object
 2   aircraft_type   71175 non-null  object
 3   schengen        71175 non-null  object
 4   origin          71175 non-null  object
 5   arrival_time    71175 non-null  float64
 6   departure_time  71175 non-null  float64
 7   day             71175 non-null  int64
 8   year            71175 non-null  int64
 9   is_holiday      71175 non-null  bool
 10  delay           71175 non-null  float64
dtypes: bool(1), float64(3), int64(3), object(4)
memory usage: 5.5+ MB
```

```
datos.describe()
```

	flight_id	arrival_time	departure_time	day	year	delay
count	71175.000000	71175.000000	71175.000000	71175.000000	71175.000000	71175.000000
mean	15.465135	13.283159	16.480222	182.000000	2016.000000	12.548378
std	8.649646	4.023380	4.143705	105.366769	3.741684	23.125349
min	1.000000	7.065594	10.065594	0.000000	2010.000000	-41.028033
25%	8.000000	8.939996	12.668655	91.000000	2013.000000	-4.412876
50%	15.000000	14.258911	16.376052	182.000000	2016.000000	9.740454
75%	23.000000	16.909690	20.041281	273.000000	2019.000000	27.650853
max	30.000000	19.341235	23.341235	364.000000	2022.000000	125.632352

```
datos.columns
```

```
Index(['flight_id', 'airline', 'aircraft_type', 'schengen', 'origin',
       'arrival_time', 'departure_time', 'day', 'year', 'is_holiday', 'delay'],
      dtype='object')
```

Creación de la columna "Fecha":

Combinaron las columnas "Day" y "Year" para crear una nueva columna llamada "date". Convirtieron los valores de año a tipo string y los concatenaron con el día. Utilizaron el método to_datetime de pandas para convertir la columna "date" al formato de fecha correcto (%Y-%j).

```
datos['date'] = datos['year'].astype(str) + '-' + (datos['day']+1).astype(str)
datos['date'] = pd.to_datetime(datos['date'], format='%Y-%j')
datos.tail(2)
```

	flight_id	airline	aircraft_type	schengen	origin	arrival_time	departure_time	day	year	is_holiday	delay	date
71173	5	BZ	Airbus A320	schengen	ZQO	9.344097	12.344097	364	2022	True	56.758844	2022-12-31

Identificación de fines de semana:

Crearon una columna llamada "isWeekend" para indicar si una fecha corresponde a un fin de semana (sábado o domingo). Utilizaron el método weekday de datetime para obtener el día de la semana y verificaron si está dentro del sexto o séptimo día (sábado y domingo).

```
datos['is_weekend'] = datos['date'].dt.weekday.isin([5,6])
datos.tail(2)
```

	flight_id	airline	aircraft_type	schengen	origin	arrival_time	departure_time	day	year	is_holiday	delay	date
71173	5	BZ	Airbus A320	schengen	ZQO	9.344097	12.344097	364	2022	True	56.758844	2022-12-31

Nombre del día de la semana:

Generaron una columna llamada "dayName" utilizando el método day_name de datetime para obtener el nombre del día de la semana correspondiente a cada fecha.

```
datos['day_name'] = datos['date'].dt.day_name()
datos.tail(2)
```

	flight_id	airline	aircraft_type	schengen	origin	arrival_time	departure_time	day	year	is_holiday	delay	date
71173	5	BZ	Airbus A320	schengen	ZQO	9.344097	12.344097	364	2022	True	56.758844	2022-12-31

En resumen, agregaron nuevas columnas al DataFrame para enriquecerlo y prepararlo para el modelado con machine learning.

```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71175 entries, 0 to 71174
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   flight_id       71175 non-null  int64
 1   airline         71175 non-null  object
 2   aircraft_type   71175 non-null  object
 3   schengen        71175 non-null  object
 4   origin          71175 non-null  object
 5   arrival_time    71175 non-null  float64
 6   departure_time  71175 non-null  float64
 7   day             71175 non-null  int64
 8   year            71175 non-null  int64
 9   is_holiday      71175 non-null  bool
 10  delay           71175 non-null  float64
 11  date            71175 non-null  datetime64[ns]
 12  is_weekend      71175 non-null  bool
 13  day_name        71175 non-null  object
dtypes: bool(2), datetime64[ns](1), float64(3), int64(3), object(5)
memory usage: 6.7+ MB
```

Codificación de variables: Los modelos de Machine Learning funcionan mejor con datos numéricos.

```
datos['schengen'] = datos['schengen'].replace({'schengen': 1, 'non-schengen': 0})
datos['is_holiday'] = datos['is_holiday'].replace({True: 1, False: 0})
datos['is_weekend'] = datos['is_weekend'].replace({True: 1, False: 0})

datos.sample(2)
```

```
/tmp/ipython-input-27-1097227084.py:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a f
  datos['schengen'] = datos['schengen'].replace({'schengen': 1, 'non-schengen': 0})
/tmp/ipython-input-27-1097227084.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a f
  datos['is_holiday'] = datos['is_holiday'].replace({True: 1, False: 0})
/tmp/ipython-input-27-1097227084.py:3: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a f
  datos['is_weekend'] = datos['is_weekend'].replace({True: 1, False: 0})
```

	flight_id	airline	aircraft_type	schengen	origin	arrival_time	departure_time	day	year	is_holiday	delay	date
65473	25	BZ	Boeing 777	0	CSF	19.341235	23.341235	349	2021	0	46.199276	2021-12-16
12406	23	YE	Embraer E175	1	TCY	18.801235	21.801235	97	2012	0	35.333705	2012-

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

```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71175 entries, 0 to 71174
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   flight_id       71175 non-null  int64
```

```
 1   airline         71175 non-null  object
 2   aircraft_type   71175 non-null  object
 3   schengen        71175 non-null  int64
 4   origin          71175 non-null  object
 5   arrival_time    71175 non-null  float64
 6   departure_time  71175 non-null  float64
 7   day             71175 non-null  int64
 8   year            71175 non-null  int64
 9   is_holiday      71175 non-null  int64
 10  delay           71175 non-null  float64
 11  date            71175 non-null  datetime64[ns]
 12  is_weekend      71175 non-null  int64
 13  day_name        71175 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(6), object(4)
memory usage: 7.6+ MB
```

Variables categóricas y booleanas: Se identificaron columnas como "Aerolínea", "Tipo de aeronave", "Espacio Schengen", "Origen", IsWeekend e IsHoliday para codificar.

Método replace: Se utilizó para transformar variables con dos categorías (Schengen/no Schengen, True/False) en 0 y 1.

```
categoricas = ['airline','aircraft_type','origin','day_name']
pd.get_dummies(data=datos,columns=categoricas,dtype=int).head()
```

	flight_id	schengen	arrival_time	departure_time	day	year	is_holiday	delay	date	is_weekend	...	origin_TCY	origin
0	26	0	8.885071	10.885071	0	2010	0	70.205981	2010-01-01	0	...	1	
1	10	0	8.939996	11.939996	0	2010	0	38.484609	2010-01-01	0	...	1	
2	3	1	18.635384	22.635384	0	2010	0	2.388305	2010-01-01	0	...	0	
3	28	0	15.967963	17.967963	0	2010	0	19.138491	2010-01-01	0	...	0	
4	15	0	16.571894	19.571894	0	2010	0	15.016271	2010-01-01	0	...	0	

5 rows × 36 columns

Función get_dummies: Se usó pd.get_dummies para codificar variables categóricas como "Airline", "Aircraft Type", "Origin" y "DayName" en múltiples columnas binarias (0 y 1).

```
datos_codificados = pd.get_dummies(data=datos,columns=categoricas,dtype=int)
datos_codificados.sample(5)
```

	flight_id	schengen	arrival_time	departure_time	day	year	is_holiday	delay	date	is_weekend	...	origin_TCY	or
68685	22	0	8.794147	11.794147	199	2022	0	44.265054	2022-07-19	0	...	0	
36230	30	1	8.923441	12.923441	225	2016	0	-8.623620	2016-08-13	1	...	0	
53471	9	0	10.733469	12.733469	279	2019	0	0.805070	2019-10-07	0	...	0	
34551	21	1	7.065594	10.065594	113	2016	0	13.085434	2016-04-23	1	...	0	
5543	7	1	8.564949	13.564949	4	2011	0	16.623981	2011-01-05	0	...	0	

5 rows × 36 columns

Advertencias: Se mostró cómo ignorar advertencias sobre futuras modificaciones en Python usando la biblioteca warnings.

```
Empieza a programar o a crear código con IA.
```

Finalmente, se decide eliminar algunas columnas del DataFrame, como flightID, DepartureTime, día, año y fecha, justificando por qué no son necesarias para el modelo en este caso. El objetivo es preparar el DataFrame para la siguiente clase, donde se iniciará el modelado con Machine Learning.

```
datos[['arrival_time','departure_time']].corr()DecisionTreeClassifier
```

	arrival_time	departure_time
arrival_time	1.000000	0.973797
departure_time	0.973797	1.000000

```
datos_codificados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71175 entries, 0 to 71174
Data columns (total 36 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   flight_id                   71175 non-null  int64
 1   schengen                    71175 non-null  int64
 2   arrival_time                71175 non-null  float64
 3   departure_time              71175 non-null  float64
 4   day                         71175 non-null  int64
 5   year                        71175 non-null  int64
 6   is_holiday                  71175 non-null  int64
 7   delay                       71175 non-null  float64
 8   date                        71175 non-null  datetime64[ns]
 9   is_weekend                  71175 non-null  int64
 10  airline_BZ                  71175 non-null  int64
 11  airline_MM                  71175 non-null  int64
 12  airline_YE                  71175 non-null  int64
 13  aircraft_type_Airbus A320   71175 non-null  int64
 14  aircraft_type_Airbus A330   71175 non-null  int64
 15  aircraft_type_Boeing 737    71175 non-null  int64
 16  aircraft_type_Boeing 777    71175 non-null  int64
 17  aircraft_type_Boeing 787    71175 non-null  int64
 18  aircraft_type_Embraer E175  71175 non-null  int64
 19  origin_AUZ                  71175 non-null  int64
 20  origin_CNU                  71175 non-null  int64
 21  origin_CSF                  71175 non-null  int64
 22  origin_EMM                  71175 non-null  int64
 23  origin_FJB                  71175 non-null  int64
 24  origin_MWL                  71175 non-null  int64
 25  origin_PUA                  71175 non-null  int64
 26  origin_TCY                  71175 non-null  int64
 27  origin_TZF                  71175 non-null  int64
 28  origin_ZQO                  71175 non-null  int64
 29  day_name_Friday             71175 non-null  int64
 30  day_name_Monday             71175 non-null  int64
 31  day_name_Saturday           71175 non-null  int64
 32  day_name_Sunday             71175 non-null  int64
 33  day_name_Thursday           71175 non-null  int64
 34  day_name_Tuesday            71175 non-null  int64
 35  day_name_Wednesday          71175 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(32)
memory usage: 19.5 MB
```

```
df = datos_codificados.drop(columns=['flight_id','departure_time','day','year','date'])
df.sample(5)
```

	schengen	arrival_time	is_holiday	delay	is_weekend	airline_BZ	airline_MM	airline_YE	aircraft_type_Airbus A320	aircr
28439	0	10.177197	0	28.225794	0	1	0	0	1	
51315	1	9.344097	0	-4.538551	0	1	0	0	1	
11363	1	7.065594	0	45.107808	1	1	0	0	0	
43216	1	17.313731	0	0.717612	0	0	0	1	0	
11195	0	19.341235	0	8.631429	0	1	0	0	0	

5 rows × 31 columns

Preparación de Datos:

Se divide el DataFrame en datos de entrenamiento y prueba utilizando train_test_split. Las variables independientes (X) se definen como todas las columnas excepto la columna delay, que es la variable de respuesta (Y).

```
from re import X
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import train_test_split

X = df.drop(columns=['delay'])
y = df['delay']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
baseline = DummyRegressor()
baseline.fit(X_train, y_train)
```

```
▾ DummyRegressor
DummyRegressor()
```

```
Empieza a programar o a crear código con IA.
```

Creación del Modelo Baseline: Se crea un modelo Baseline utilizando DummyRegressor y se ajusta a los datos de entrenamiento (X_train y Y_train).

Evaluación del Modelo: Se evalúa el modelo utilizando métricas como el error cuadrático medio (mean_squared_error), el error absoluto medio (mean_absolute_error) y el coeficiente de determinación (r2_score).

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Función calcularRegresión: Se define una función para calcular y formatear las métricas de evaluación.

```
y_pred_dummy = baseline.predict(X_test)
def calcularRegresion(y_test,y_pred):
  rmse = mean_squared_error(y_test,y_pred)
  mae = mean_absolute_error(y_test,y_pred)
  r2 = r2_score(y_test,y_pred)
  metricas = {
      'RMSE':(round(rmse**(1/2),4)),
      'MAE':round(mae,4),
      'R2':round(r2,4)
  }
  return metricas
```

Resultados: Se obtienen los resultados del modelo Baseline, que muestran un error cuadrático medio de 23.16 minutos, un error absoluto medio de 18.5 minutos y un coeficiente de determinación de cero, lo que indica que el modelo no está generalizando correctamente.El objetivo es mejorar estos resultados en las próximas clases utilizando modelos más adecuados.

```
resultados_baseline = calcularRegresion(y_test,y_pred_dummy)
resultados_baseline
```

```
{'RMSE': 23.1612, 'MAE': 18.5646, 'R2': -0.0}
```

{'RMSE': 23.1612, 'MAE': 18.5646, 'R2': -0.0}

- resultados no tan bueno

```
Empieza a programar o a crear código con IA.
```

Importación y Configuración del Modelo:

Se importó RandomForestRegressor de sklearn.ensemble. Se creó un modelo llamado modelo utilizando RandomForestRegressor con una profundidad máxima de 5 (max_depth=5) y un estado de aleatoriedad fijo (random_state=42).

```
from sklearn.ensemble import RandomForestRegressor
```

```
modelo = RandomForestRegressor(max_depth=5,random_state=42)
modelo.fit(X_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=5, random_state=42)
```

```
ypred =modelo.predict(X_test)
resultados_rf = calcularRegresion(y_test,ypred)
resultados_rf
```

```
{'RMSE': 13.7479, 'MAE': 11.0262, 'R2': 0.6477}
```

{'RMSE': 13.7479, 'MAE': 11.0262, 'R2': 0.6477}

```
Empieza a programar o a crear código con IA.
```

Haz doble clic (o pulsa Intro) para editar

Importación y Configuración del Modelo:

Se importó RandomForestRegressor de sklearn.ensemble. Se creó un modelo llamado modelo utilizando RandomForestRegressor con una profundidad máxima de 5 (max_depth=5) y un estado de aleatoriedad fijo (random_state=42).

Evaluación del Modelo:

Se calculó la regresión con calcularRegresión para evaluar el resultado del modelo. Se observó una mejora significativa en comparación con la línea base, con un Root Mean Square Error de 13 minutos y un coeficiente de determinación de casi 65%.

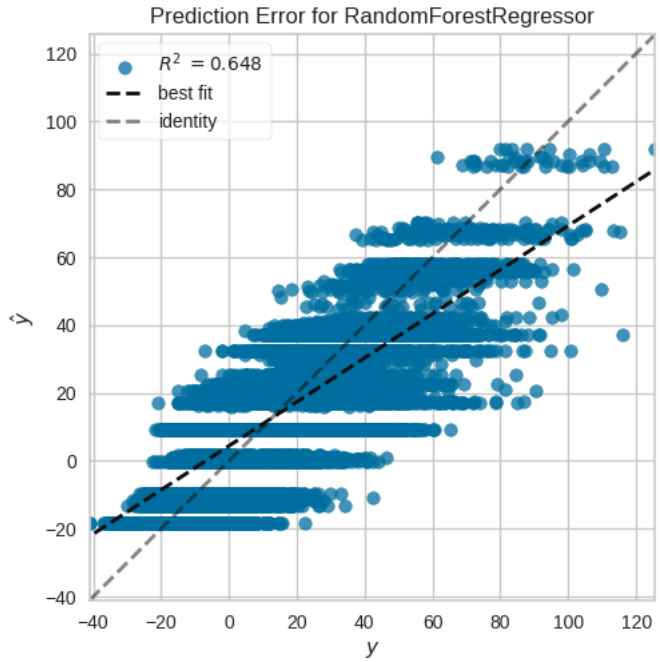
Evaluación del Modelo:

Se calculó la regresión con calcularRegresión para evaluar el resultado del modelo. Se observó una mejora significativa en comparación con la línea base, con un Root Mean Square Error de 13 minutos y un coeficiente de determinación de casi 65%.

Análisis de Residuos:

Se explicó que los residuos son la diferencia entre las observaciones y la línea trazada por el modelo. Se observó que el comportamiento de prueba fue muy cercano al de entrenamiento, indicando que el modelo está generalizando relativamente bien.

```
from yellowbrick.regressor import PredictionError
visualizer = PredictionError(modelo)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



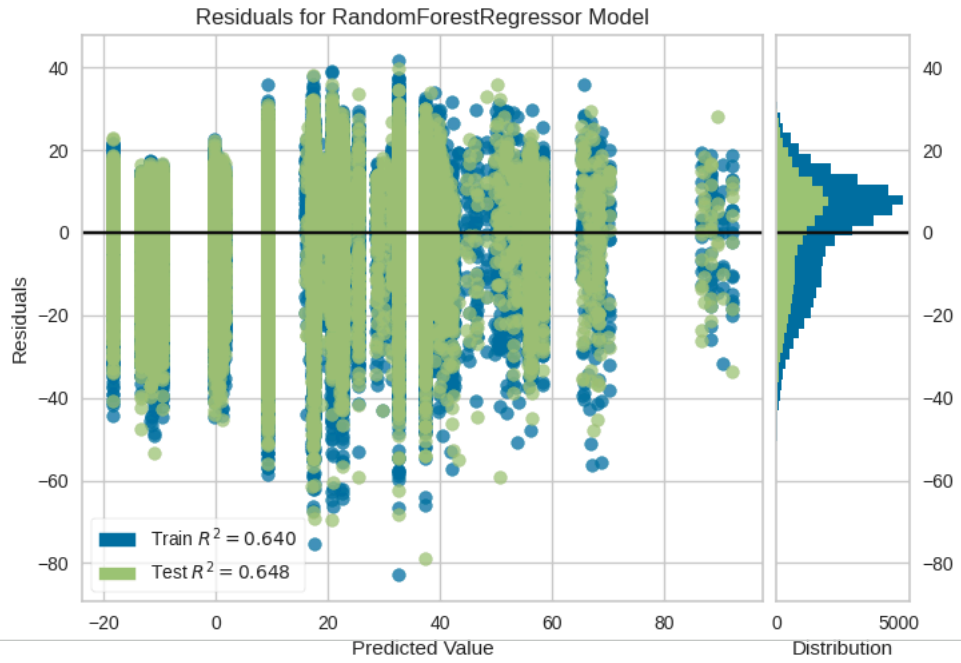

```
<Axes: title={'center': 'Prediction Error for RandomForestRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>
```

Haz doble clic (o pulsa Intro) para editar

Análisis de Residuos:

Se explicó que los residuos son la diferencia entre las observaciones y la línea trazada por el modelo. Se observó que el comportamiento de prueba fue muy cercano al de entrenamiento, indicando que el modelo está generalizando relativamente bien.

```
from yellowbrick.regressor import ResidualsPlot
visualizer = ResidualsPlot(modelo)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```




```
<Axes: title={'center': 'Residuals for RandomForestRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>
```

importancia de la validación cruzada para asegurarnos de que el modelo de Random Forest Regressor generalice bien en diferentes combinaciones de datos. Se utilizó la función KFold de Scikit-Learn para dividir el conjunto de datos en varias partes, entrenando y probando el modelo con diferentes combinaciones.

Además, se empleó la función cross_validate para realizar la validación cruzada, utilizando métricas de utilidad como neg_root_mean_squared_error, neg_mean_absolute_error y r2 para evaluar el rendimiento del modelo en cada combinación. Los resultados mostraron consistencia en las métricas, lo que indica que el modelo generaliza bien.

Validación cruzada

3 - Validación cruzada

Por último, vamos a realizar la validación cruzada para evaluar el modelo de manera más asertiva.

Haz doble clic (o pulsa Intro) para editar

```
from sklearn.model_selection import KFold, cross_validate

scoring = {
    'RMSE': 'neg_root_mean_squared_error',
    'MAE': 'neg_mean_absolute_error',
    'R2': 'r2'
}
cv = KFold(n_splits=5,shuffle=True,random_state=42)
cv_resultados = cross_validate(modelo,X_train,y_train, scoring=scoring, cv=cv)
cv_resultados
```

```
{'fit_time': array([1.98281264, 2.36270428, 2.33271194, 2.2031126 , 2.11878252]),
 'score_time': array([0.03711438, 0.04220319, 0.03815985, 0.0356133 , 0.03747153]),
 'test_RMSE': array([-13.76989748, -13.86928373, -13.96807668, -13.85347402,
        -13.98953528]),
 'test_MAE': array([-11.03252135, -11.13534968, -11.2094657 , -11.14503717,
        -11.25775565]),
 'test_R2': array([0.64559466, 0.63745663, 0.63523823, 0.64089302, 0.63441267])}
```

Clase 4

Análisis de la importancia de las features:

Se examina la importancia de cada columna (feature) en el DataFrame para determinar su relevancia en el modelo. Se utiliza el atributo feature_importances_ para obtener un array con los porcentajes de importancia de cada feature. Se crea un DataFrame llamado FeatureImportances para visualizar y ordenar las features según su importancia.

```
len(df.columns)
```

```
31
```

```
modelo.feature_importances_
```

```
array([3.39814918e-06, 3.60417588e-02, 1.45895991e-01, 6.10861947e-05,
       5.30319673e-01, 1.36989042e-05, 2.14417900e-05, 1.00746574e-01,
       5.76173558e-02, 5.18974843e-04, 8.83162236e-05, 2.64390542e-02,
       4.58830752e-02, 1.01922847e-04, 1.71891675e-04, 1.59736760e-02,
       3.57450957e-05, 9.21269997e-05, 5.42552087e-04, 1.03212932e-02,
       2.09099647e-02, 6.63707069e-03, 2.64070312e-04, 8.68597522e-04,
       1.00052465e-04, 2.79146627e-05, 7.17673254e-05, 3.72705034e-05,
       1.06499638e-04, 8.71862374e-05])
```

Selección de la cantidad óptima de features:

Se crea un DataFrame llamado Resultados para comparar el rendimiento del modelo con diferentes cantidades de features. Se itera sobre diferentes cantidades de features (1, 5, 10, 15, 20, 25 y 30) y se entrena un modelo Random Forest con cada subconjunto de features. Se evalúa el rendimiento del modelo utilizando métricas como RMSE, MAE y R^2. Se observa que el rendimiento del modelo se estabiliza alrededor de 15 features, lo que indica que añadir más features no mejora significativamente el modelo.

```
importances = modelo.feature_importances_
```

```
feature_importances = pd.DataFrame({'Features' :X.columns, 'Importances' :(importances*100).round(2)}).sort_values('Importances
feature_importances
```

	Features	Importances
4	airline_BZ	53.03
2	is_holiday	14.59
7	aircraft_type_Airbus A320	10.07
8	aircraft_type_Airbus A330	5.76
12	aircraft_type_Embraer E175	4.59
1	arrival_time	3.60
11	aircraft_type_Boeing 787	2.64
20	origin_TCY	2.09
15	origin_CSF	1.60
19	origin_PUA	1.03
21	origin_TZF	0.66
23	day_name_Friday	0.09
18	origin_MWL	0.05
9	aircraft_type_Boeing 737	0.05
22	origin_ZQO	0.03
14	origin_CNU	0.02
29	day_name_Wednesday	0.01
17	origin_FJB	0.01
10	aircraft_type_Boeing 777	0.01
3	is_weekend	0.01
24	day_name_Monday	0.01
13	origin_AUZ	0.01
28	day_name_Tuesday	0.01
26	day_name_Sunday	0.01
0	schengen	0.00
5	airline_MM	0.00
6	airline_YE	0.00
16	origin_EMM	0.00
25	day_name_Saturday	0.00
27	day_name_Thursday	0.00

Ajuste fino de la cantidad de features:

Se modifica el código para explorar un rango más estrecho de cantidades de features (10 a 15) y encontrar el número óptimo. Se observa que el mejor rendimiento se obtiene con 13 features.

```
resultados = pd.DataFrame(index=['RMSE','MAE','R2'])
model_features = RandomForestRegressor(max_depth=5,random_state=42)
ct_features = [i if i != 0 else 1 for i in range(0,35,5)]
for i in ct_features:
  selected_features = feature_importances['Features'][:i]
  X_train_selected = X_train[selected_features]
  X_test_selected = X_test[selected_features]

  model_features.fit(X_train_selected,y_train)
  y_pred_selected = model_features.predict(X_test_selected)
  metricas = calcularRegresion(y_test,y_pred_selected)
  resultados[i] = metricas.values()
resultados
```

	1	5	10	15	20	25	30
RMSE	18.8331	15.1564	13.7587	13.7497	13.7498	13.7477	13.7479
MAE	14.9620	12.0605	11.0422	11.0285	11.0280	11.0261	11.0262
R2	0.3388	0.5718	0.6471	0.6476	0.6476	0.6477	0.6477

Preparación de los datos para el ajuste de hiperparámetros:

Se seleccionan las 13 features más importantes y se crea un nuevo DataFrame con solo estas features. Se divide el DataFrame en conjuntos de entrenamiento y prueba utilizando la función train_test_split.

```
resultados = pd.DataFrame(index=['RMSE','MAE','R2'])
model_features = RandomForestRegressor(max_depth=5,random_state=42)
ct_features = range(10,16)
for i in ct_features:
  selected_features = feature_importances['Features'][:i]
  X_train_selected = X_train[selected_features]
  X_test_selected = X_test[selected_features]

  model_features.fit(X_train_selected,y_train)
  y_pred_selected = model_features.predict(X_test_selected)
  metricas = calcularRegresion(y_test,y_pred_selected)
  resultados[i] = metricas.values()
resultados
```

	10	11	12	13	14	15
RMSE	13.7587	13.7564	13.7526	13.7526	13.7497	13.7497
MAE	11.0422	11.0425	11.0312	11.0314	11.0285	11.0285
R2	0.6471	0.6472	0.6474	0.6474	0.6476	0.6476

```
Empieza a programar o a crear código con IA.
```

de 10 a 12 features hay mejora paulatina y en 13 features se estanca o estabiliza

```
selected_features = feature_importances['Features'][:13]
X_selected_features = X[selected_features]
X_selected_features
```

	airline_BZ	is_holiday	aircraft_type_Airbus A320	aircraft_type_Airbus A330	aircraft_type_Embraer E175	arrival_time	aircraft_type_Bo
0	0	0	1	0	0	8.885071	
1	0	0	1	0	0	8.939996	
2	1	0	0	0	1	18.635384	
3	1	0	0	1	0	15.967963	
4	1	0	0	1	0	16.571894	
...	...	...	...	...	...	...	
71170	1	1	0	0	1	18.635384	
71171	1	1	0	1	0	16.718722	
71172	0	1	0	0	0	8.564949	
71173	1	1	1	0	0	9.344097	
71174	1	1	0	0	0	8.591208	

71175 rows × 13 columns

```
X_train, X_test, y_train, y_test = train_test_split(X_selected_features, y, random_state=42)
```

Ajuste de Hiperparámetros:

Selección de las 13 características más importantes del modelo. Revisión de la documentación del estimador de Random Forest y el regresor para identificar los hiperparámetros ajustables.

```
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3]
}
```

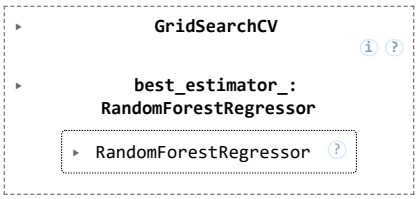
```
from sklearn.model_selection import GridSearchCV
```

```
Empieza a programar o a crear código con IA.
```

Uso de GridSearchCV:

Importación de la clase GridSearchCV de sklearn.model_selection. Implementación de la validación cruzada con KFold, especificando 5 divisiones y un estado de aleatoriedad.

```
cv = KFold(n_splits=5,shuffle=True,random_state=42)
model_grid = GridSearchCV(RandomForestRegressor(random_state=42),
                          param_grid=param_grid ,
                          cv=cv,scoring='r2')
model_grid.fit(X_train,y_train)
```



Definición del Model Grid:

Creación de un estimador RandomForestRegressor con un estado de aleatoriedad fijo.

Definición de los parámetros a probar en el param_grid: profundidad máxima, mínimo de muestras de hojas, mínimo de muestras para las divisiones y número de estimadores.

Entrenamiento y Predicción:

Ajuste del modelo con model_grid.fit utilizando los datos de entrenamiento. Predicción con los mejores parámetros encontrados por GridSearchCV utilizando X_test.

```
model_grid.best_params_
```

```
{'max_depth': 10,
 'min_samples_leaf': 3,
 'min_samples_split': 2,
 'n_estimators': 200}
```

Evaluación de Métricas:

Cálculo de métricas como el coeficiente de determinación, el error absoluto medio y el Root Mean Square Error. Comparación de las métricas del modelo ajustado con las métricas anteriores para evaluar la mejora.

```
y_pred_grid = model_grid.predict(X_test)
metrica_model_grid = calcularRegresion(y_test,y_pred_grid)
metrica_model_grid
```

```
{'RMSE': 13.2308, 'MAE': 10.6296, 'R2': 0.6754}
```

```
resultados['modelo_grid'] = list(metrica_model_grid.values())
resultados
```

	10	11	12	13	14	15	modelo_grid
RMSE	13.7587	13.7564	13.7526	13.7526	13.7497	13.7497	13.2308
MAE	11.0422	11.0425	11.0312	11.0314	11.0285	11.0285	10.6296
R2	0.6471	0.6472	0.6474	0.6474	0.6476	0.6476	0.6754

```
Empieza a programar o a crear código con IA.
```

El RandomForestRegressor es un modelo de conjunto que combina varios árboles de decisión para hacer predicciones más robustas y precisas a través de la técnica de Bagging.

Los hiperparámetros son configuraciones que puedes ajustar para controlar el comportamiento del RandomForestRegressor. A continuación, se presenta una lista de los principales hiperparámetros:

1 - n_estimators:

Este hiperparámetro especifica el número de árboles de decisión que se crearán en el bosque aleatorio. Cuanto mayor sea el número de estimadores, mayor será la capacidad del modelo para ajustarse a los datos. Sin embargo, un número muy alto puede llevar a un aumento en el tiempo de entrenamiento. 2 - criterion:

El hiperparámetro criterion determina la función de medición de la calidad de una división durante la construcción de los árboles. Para regresión, el valor predeterminado es "mse" (Error Cuadrático Medio), que calcula la media de los cuadrados de los errores. Otra opción es "mae" (Error Absoluto Medio), que utiliza la media de los valores absolutos de los errores. 3 - max_depth:

Este hiperparámetro controla la profundidad máxima de los árboles de decisión en el bosque. Limitar la profundidad puede ayudar a evitar el sobreajuste, ya que impide que los árboles se ajusten demasiado a los datos de entrenamiento y no puedan hacer buenas predicciones para nuevos datos. 4 - min_samples_split:

El min_samples_split determina el número mínimo de muestras necesarias para dividir un nodo interno del árbol. Esto ayuda a controlar el crecimiento de los árboles y evita divisiones que llevan a nodos con pocas muestras. 5 - min_samples_leaf:

Este hiperparámetro define el número mínimo de muestras requeridas en una hoja (nodo terminal) del árbol. Esto ayuda a controlar la granularidad del árbol y puede impedir que las hojas contengan muy pocas muestras. 6 - max_features:

max_features especifica el número máximo de características a considerar al buscar la mejor división en cada nodo. Los valores comunes incluyen "auto" (sqrt(n_features)), "sqrt" (también sqrt(n_features)), "log2" (log2(n_features)), o un número entero que representa la cantidad exacta de características a considerar. 7 - random_state:

Este hiperparámetro define una semilla para el generador de números aleatorios utilizado para crear el bosque aleatorio. Definir un valor fijo para random_state garantiza que el modelo sea reproducible. 8 - n_jobs:

n_jobs especifica el número de núcleos de CPU a utilizar para el entrenamiento en paralelo. Si se define como -1, se utilizarán todos los núcleos disponibles. Estos son algunos de los principales hiperparámetros del RandomForestRegressor en scikit-learn. La elección adecuada de estos hiperparámetros puede afectar significativamente el rendimiento y la capacidad de generalización del modelo para tareas de regresión. Para consultar los otros hiperparámetros faltantes, puedes consultar la documentación oficial de scikit-learn.

Aquí están los pasos clave:

Importar pickle:

Este módulo permite serializar y deserializar objetos de Python.

```
from sklearn.ensemble import RandomForestRegressor
import pickle
```

```
Empieza a programar o a crear código con IA.
```

Abrir un archivo binario:

Se utiliza la función open para crear un archivo con extensión .pkl en modo de escritura binaria ('wb'). Este archivo almacenará el modelo