Memoria de sesiones de Prácticas de Laboratorio (5-8)

Juan Francisco Mier Montoto
Inteligencia de Negocio, EPI Gijón 23-24

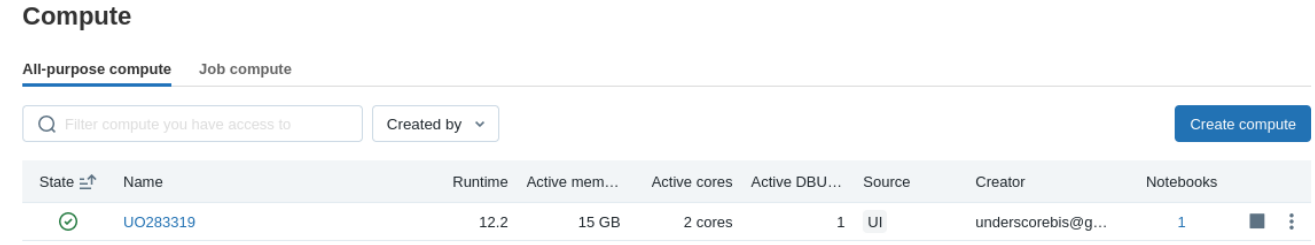
Índice

- **Práctica 5**
- **Práctica 6**
  - Reglas de asociación
    - Código
    - Parámetros
    - Preguntas
  - IA explicativa
- **Práctica 7**
- **Práctica 8**
  - Preparación del código
  - Tareas

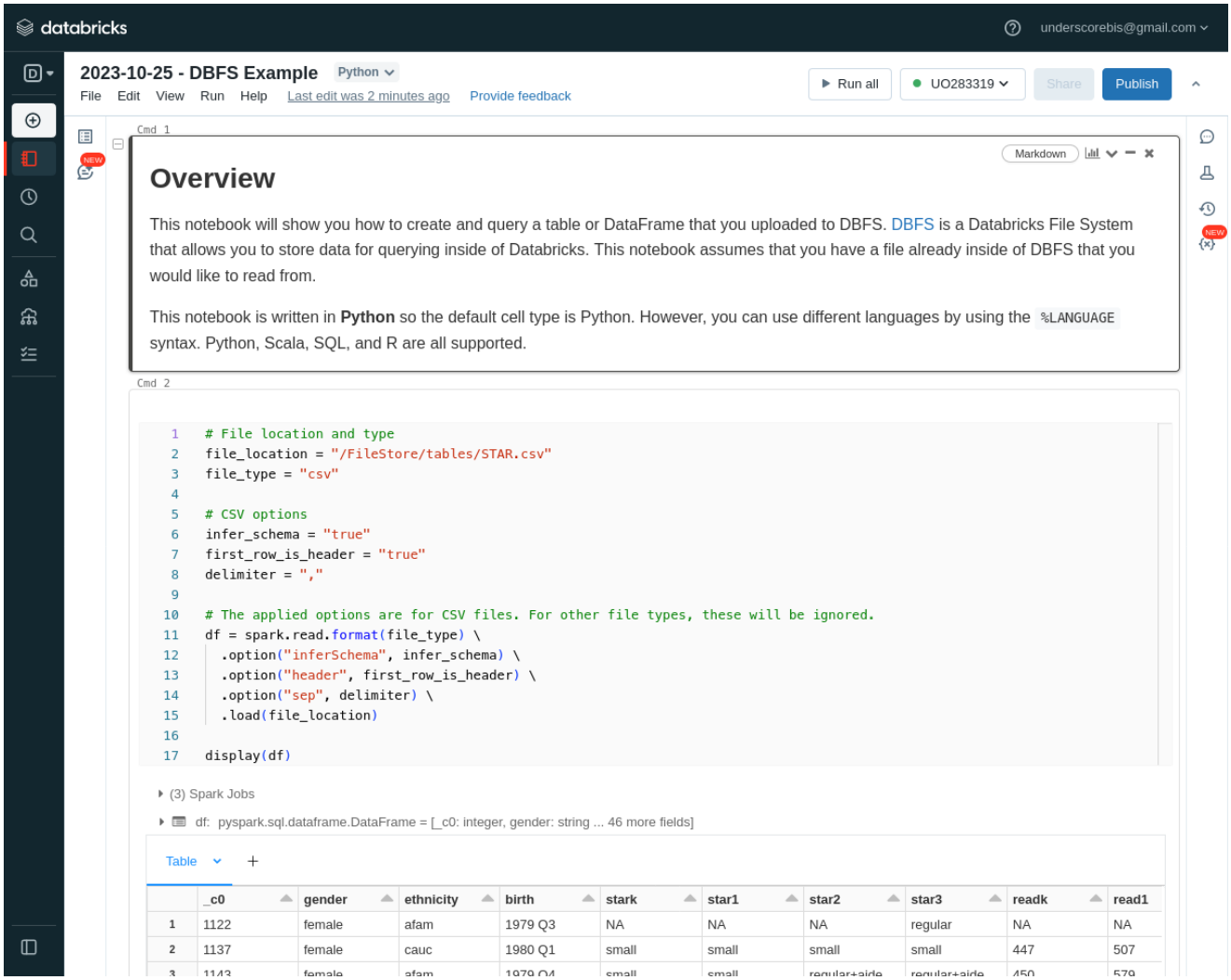
Práctica 5

Para la práctica, se crea una cuenta en la edición *Community* de Databricks como especificado.

Se crea una unidad de *compute*:



Se importa el csv, que genera un Notebook de Python de manera automática (y se comienza a desarrollar):



Durante la práctica, por lo general, se siguen los pasos del enunciado con muy pocas modificaciones. Se hace uso también del tutorial (`tutorial.ipynb`) suministrado para teoría para algunas cosas.

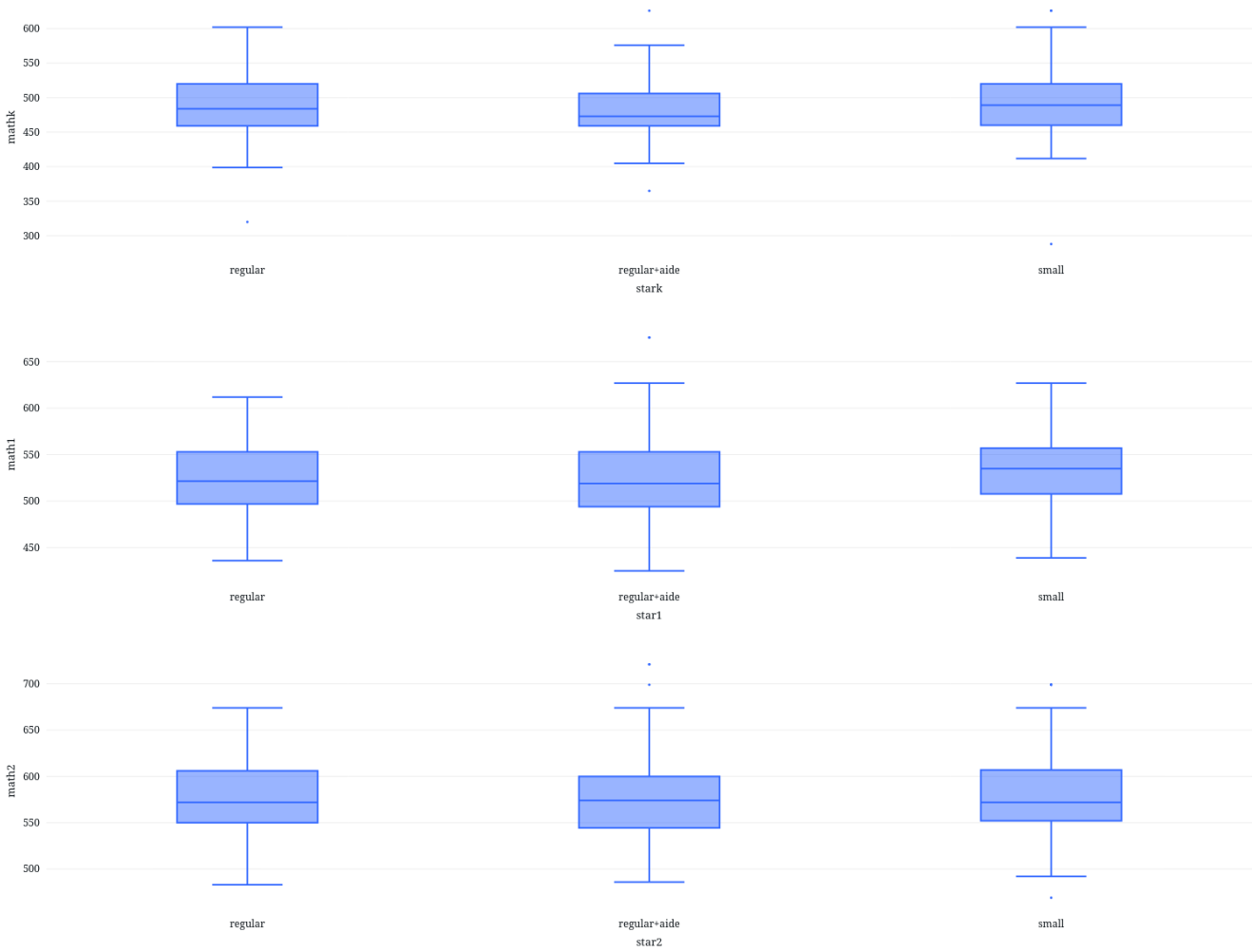
En el paso 8, hay que pasar todas las variables categóricas (incluyendo también primero, segundo y tecero):

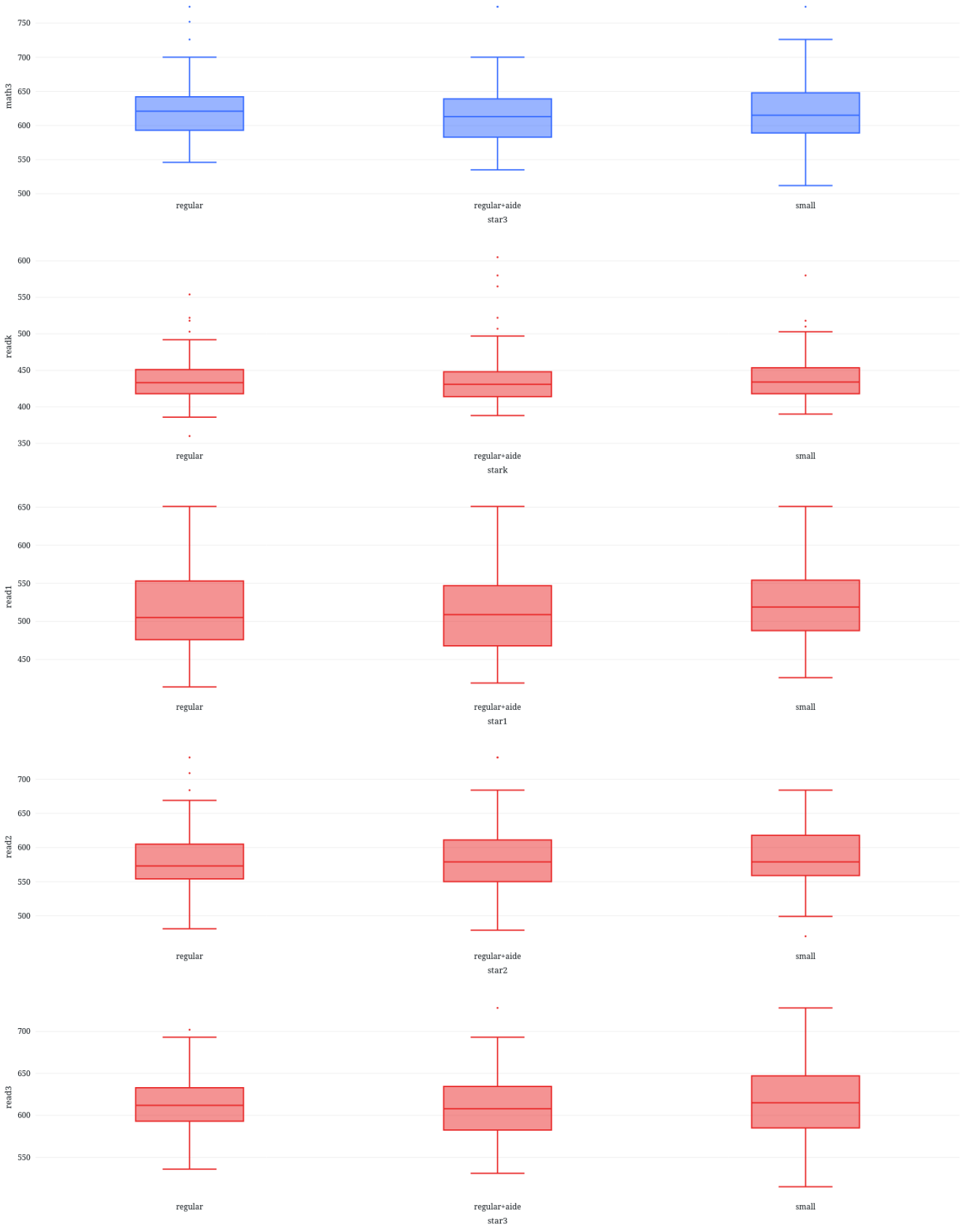
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer,OneHotEncoder

colk =
['gender','ethnicity','birth','stark','star1','star2','star3','lunchk','lunch1'
,'lunch2','lunch3','schoolk','school1','school2','school3','degreek','degree1',
'degree2','degree3','ladderk','ladder1','ladder2','ladder3','tethnicityk','teth
nicity1','tethnicity2','tethnicity3','systemk','system1','system2','system3','s
choolidk','schoolid1','schoolid2','schoolid3']
colkI = [a+'I' for a in colk]
colkE = [a+'E' for a in colk]

indexer = StringIndexer(inputCols=colk, outputCols=colkI)
encoder = OneHotEncoder(inputCols=colkI, outputCols=colkE)
pipeline = Pipeline(stages=[indexer,encoder])

df = pipeline.fit(df).transform(df)
```

Copy

Para visualizar los datos en los pasos 9 y 10, se ejecuta `display(df)` y se generan visualizaciones nuevas con los parámetros deseados.

Los gráficos son los siguientes (en orden $k \to 1 \to 2 \to 3$ y $math \to read$)





Para el resto de tareas, se crea un método que admita varios parámetros con el objetivo de reducir el código que se repite y se pueda ejecutar en serie de manera más sencilla:

```
from pyspark.ml.feature import Imputer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
```

```
rcolinputk =
['genderI','ethnicityI','birthI','starkI','lunchkI','schoolkI','degreekI','ladderkI','tethnicitykI']
# Dividir con randomSplit en 75% y 25%
train, test = df.randomSplit([0.75, 0.25])

# Crear un pipeline con tres etapas y aplicársela al conjunto de entrenamiento
def execute_read(var, k, regressor):
        # Etapa 1. Con pyspark.ml.feature.Imputer, rellena los valores perdidos de la
        # variable de salida (strategy="mean") en otra variable.
        im = Imputer(inputCol=f"read{k}", outputCol=f"my{var}{k}_imputado", strategy="mean")

        # Etapa 2. Con pyspark.ml.feature.VectorAssembler, combina todas las variables
        # de entrada en una columna llamada "myfeatures".
        assembler = VectorAssembler(inputCols=colinputk, outputCol="myfeatures")

        # Etapa 3. Con pyspark.ml.regression.LinearRegression, ajusta un modelo a las
        # variables de entrada "myfeatures" y de salida "myreadk_imputado".
        lr = regressor(featuresCol="myfeatures", labelCol=f"my{var}{k}_imputado")

        # Crear un pipeline con las tres etapas anteriores
        df = Pipeline(stages=[im, assembler, lr]).fit(train).transform(test)

        # Por último, calcula el error cuadrático medio y R2 en conjunto de test.
        evaluator = RegressionEvaluator(labelCol=f"my{var}{k}_imputado", predictionCol="prediction", metricName="rmse")
        rmse = evaluator.evaluate(df)
        print(f"{var}{k}: RMSE = {rmse}")
        evaluator = RegressionEvaluator(labelCol=f"my{var}{k}_imputado", predictionCol="prediction", metricName="r2")
        r2 = evaluator.evaluate(df)
        print(f"{var}{k}: R2 = {r2}")


vars = ['read', 'math']
ks = ['k', '1', '2', '3']
regressors = [LinearRegression, RandomForestRegressor]
for regressor in regressors:
        print(f"--- {regressor} ---")
        for var in vars:
                for k in ks:
                        execute_read(var, k, regressor)
        print()
```

Resultados

LinearRegression

```
--- <class 'pyspark.ml.regression.LinearRegression'> ---
readk: RMSE = 21.619245485356064
readk: R2 = 0.08485149876036313
read1: RMSE = 39.316696546831395
read1: R2 = 0.0892636113699351
read2: RMSE = 31.540031330499513
read2: R2 = 0.09259473771094673
read3: RMSE = 26.55601286849753
read3: R2 = 0.06826478739037134
mathk: RMSE = 21.619245485356064
mathk: R2 = 0.08485149876036313
math1: RMSE = 39.316696546831395
math1: R2 = 0.0892636113699351
math2: RMSE = 31.540031330499513
math2: R2 = 0.09259473771094673
math3: RMSE = 26.55601286849753
math3: R2 = 0.06826478739037134
```

RandomForestRegressor

```
--- <class 'pyspark.ml.regression.RandomForestRegressor'> ---
readk: RMSE = 21.299212748352044
readk: R2 = 0.11174510395260862
read1: RMSE = 38.45600871402194
read1: R2 = 0.128701305781043
read2: RMSE = 30.827644354682175
read2: R2 = 0.13312249603964366
read3: RMSE = 25.909975707199827
read3: R2 = 0.11304664745658033
mathk: RMSE = 21.299212748352044
mathk: R2 = 0.11174510395260862
math1: RMSE = 38.45600871402194
math1: R2 = 0.128701305781043
math2: RMSE = 30.827644354682175
math2: R2 = 0.13312249603964366
math3: RMSE = 25.909975707199827
math3: R2 = 0.11304664745658033
```

Análisis

A primera vista, el regresor de *RandomForest* funciona de manera ligeramente mejor que el regresor lineal, con r^2 ligeramente superiores y con $RMSE$ ligeramente inferiores. Independientemente de esto, los resultados son, por lo general, malos, con valores de r^2 inválidos para cualquier análisis serio fuera de demostraciones de juguete como estas.

Práctica 6

Reglas de asociación

Código

Antes de realizar análisis, se introduce de manera breve el código que permite realizarlo en primer lugar:

```
# Importar librerías
%matplotlib ipympl
import seaborn as sns
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
from IPython.display import display, Markdown
import ipywidgets as widgets
```

```
data = pd.read_csv('Grocery Products Purchase.csv') # se lee el fichero de datos
data = data.iloc[:, 0:8] # selección de las primeras tres columnas
data = data.dropna() # se eliminan las compras que tengan 'nan'

for a in data.columns: # parseo
        data[a] = data[a].astype(str)
```

```
# Selección de algunas columnas y transformación a formato de transacción
transactions = data.iloc[:, 0:8].values.tolist()

# Transformación con One-Hot Encoding
te = TransactionEncoder()
oht = te.fit(transactions).transform(transactions)
df = pd.DataFrame(oht, columns=te.columns_)
```

```
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True) # Minería de ítems frecuentes
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=2) # Minería de reglas de asociación
```

```
# método que añade interactividad a los gráficos
def onclick(event):
    # Obtener el índice del punto más cercano al clic
    idx = event.ind[0]

    # Extraer la regla correspondiente a ese índice
```

```
    rule = rules.iloc[idx]
    antecedents = ', '.join(list(rule['antecedents']))
    consequents = ', '.join(list(rule['consequents']))

    # Mostrar la regla en el notebook
    display(Markdown(f"**Regla:** {antecedents} -> {consequents}"))
    display(Markdown(f"**Lift:** {rule['lift']:.2f}, **Confianza:**
{rule['confidence']:.2f}"))
```

Copy

```
fig, ax = plt.subplots(figsize=(10, 6))
sns.scatterplot(data=rules, x="lift", y="confidence", alpha=0.6,
size="support", sizes=(20, 200), ax=ax, picker=4)
plt.title("Reglas de Asociación: Lift vs Confianza")

fig.canvas.mpl_connect('pick_event', onclick) # Conectar el evento de clic con
la función onclick
plt.show()
```

Parámetros

Para llegar a estos valores, se ha jugado bastante con las variables de `min_support` y `min_threshold`:

- Se encuentra que variando los valores de `min_support`, se consiguen diferentes valores medios de *Lift* (la variable que se está buscando maximizar). En concreto, la relación es inversa, por lo que se reduce `min_support` a un valor lo más pequeño posible que obtiene resultados lógicos.
- Tras ajustar el `min_support`, se observa que el gráfico representa demasiadas reglas de asociación (fuera del rango buscado de entre 10 y 100), por lo que se aumenta el umbral mínimo para tratar de reducir la cantidad de valores que se representan.

Preguntas

1. ¿Cuáles son las asociaciones más relevantes entre los productos?

Para realizar esta tarea, se inyecta un poco de código que muestra (formateado) las diez reglas más relevantes.

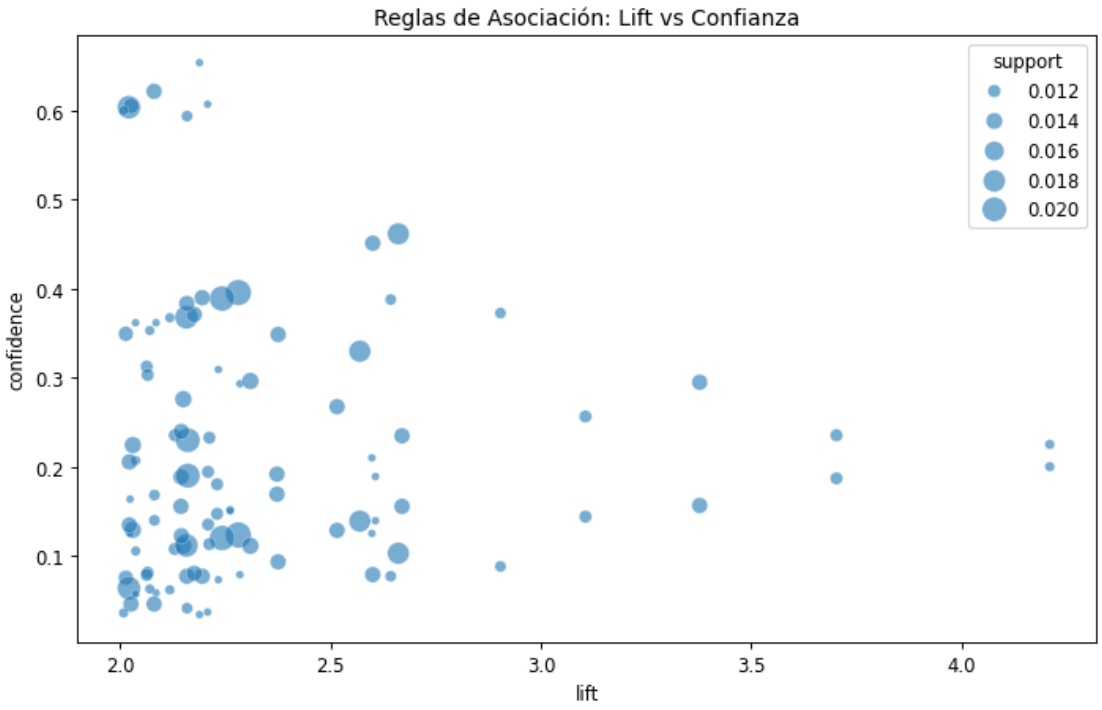
```
# mostrar las asociaciones más relevantes
display(Markdown('### Reglas de asociación'))
display(rules.sort_values(by=['lift'], ascending=False).head(10))
```

Copy

id	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
10	(chocolate)	(waffles)	0.053444	0.047506	0.010689	0.200000	4.210
11	(waffles)	(chocolate)	0.047506	0.053444	0.010689	0.225000	4.210
15	(flour)	(sugar)	0.050475	0.063539	0.011876	0.235294	3.703

id	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
14	(sugar)	(flour)	0.063539	0.050475	0.011876	0.186916	3.703
24	(processed cheese)	(white bread)	0.046318	0.087292	0.013658	0.294872	3.377
25	(white bread)	(processed cheese)	0.087292	0.046318	0.013658	0.156463	3.377
16	(ham)	(processed cheese)	0.082542	0.046318	0.011876	0.143885	3.106
17	(processed cheese)	(ham)	0.046318	0.082542	0.011876	0.256410	3.106
4	(misc. beverages)	(bottled water)	0.030285	0.128266	0.011283	0.372549	2.904
5	(bottled water)	(misc. beverages)	0.128266	0.030285	0.011283	0.087963	2.904

2. Visualice un grafo con las reglas más relevantes



El gráfico es interactivo: al hacer click encima de alguna de las reglas, se imprime información en el notebook.

```
Regla: bottled beer -> soda
Lift: 2.28, Confianza: 0.40
Regla: herbs -> whole milk, root vegetables
Lift: 2.16, Confianza: 0.37
Regla: herbs -> whole milk, root vegetables
Lift: 2.16, Confianza: 0.37
Regla: soda, yogurt -> bottled water
Lift: 2.31, Confianza: 0.30
Regla: soda, tropical fruit -> bottled water
Lift: 2.29, Confianza: 0.29
Regla: root vegetables, whole milk, tropical fruit -> other vegetables, citrus fruit
Lift: 2.52, Confianza: 0.27
Regla: other vegetables, citrus fruit -> root vegetables, whole milk, tropical fruit
Lift: 2.52, Confianza: 0.13
```

3. Dé un ejemplo de cómo usaría esta información para decidir sobre las bajadas y subidas de precio

Teniendo en cuenta la tabla del primer punto, por lo general sería buena idea rebajar y subirle el precio a alguna de las siguientes parejas de items:

- Chocolate y gofres
- Azúcar y harina
- Pan de molde y lonchas de queso
- Agua y otras bevidas

Por ejemplo, podría hacerse una oferta de harina muy agresiva pero por otro lado subir el precio del azúcar.

IA explicativa

Lo primero de todo es cargar el dataset y preprocesarlo:

```
data = pd.read_csv('bsd/breast-cancer-wisconsin.data', header=None)
data.columns = [
    'Id number',
    'Clump Thickness',
    'Uniformity of Cell Size',
    'Uniformity of Cell Shape',
    'Marginal Adhesion',
    'Single Epithelial Cell Size',
    'Bare Nuclei',
    'Bland Chromatin',
    'Normal Nucleoli',
```

```
    'Mitoses',
    'Class'
]
data = data.apply(pd.to_numeric, errors='coerce')
data = data.dropna()
```

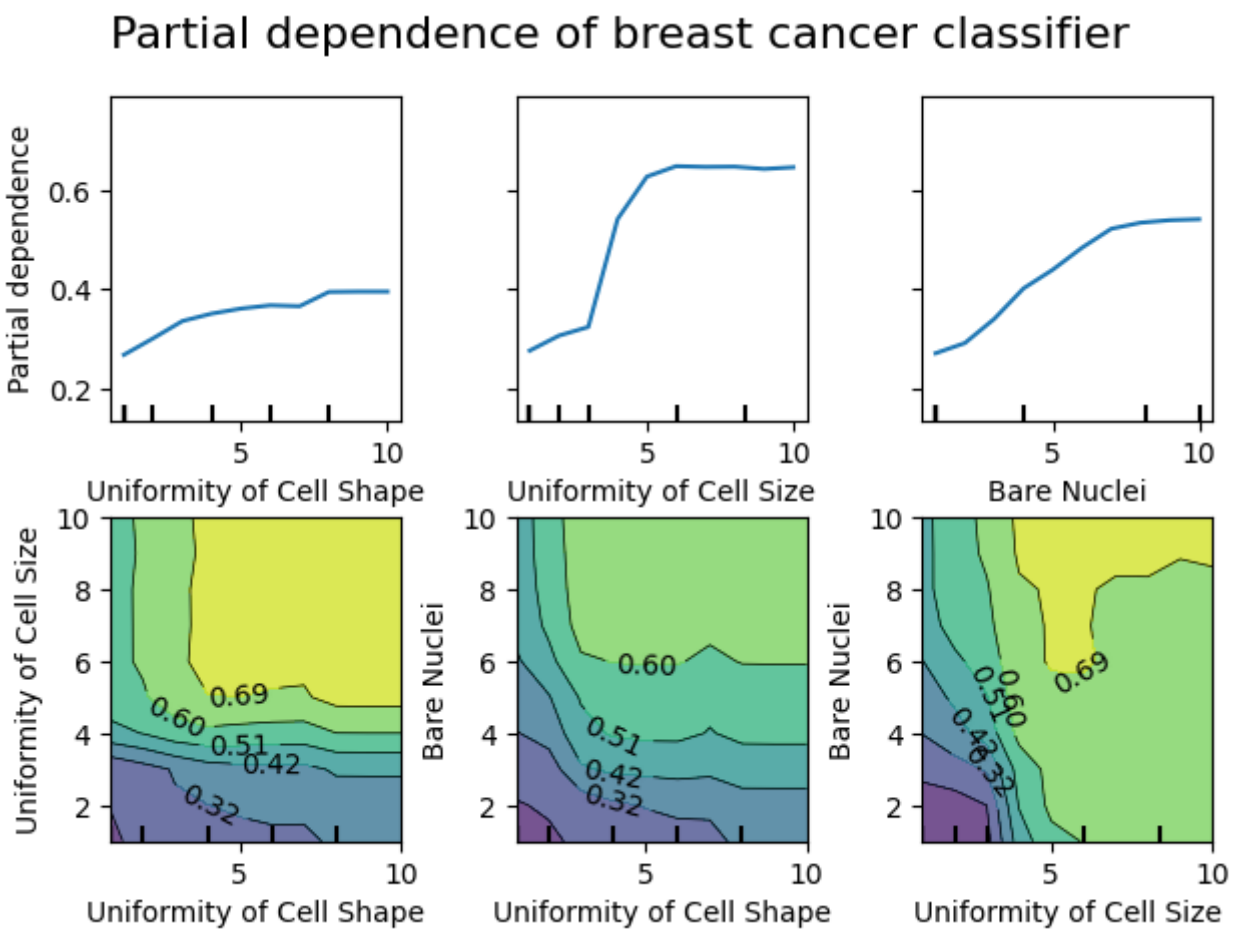
Se entrena un modelo de `GradientBoostingClassifier`:

```
# train test split
y = data['Class']
X = data.drop(columns=['Id number', 'Class'])
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=7401)

# fit model by default
model = GradientBoostingClassifier(random_state=7401)
model.fit(X_train, y_train)

# evaluate
score = model.score(X_test, y_test)
print(score)
```

... lo que devuelve 0.9532163742690059.

Para las dos primeras partes, se crea un *Partial Dependence Plot* y se crean los gráficos que se piden:

```
# partial dependence plot
pdp = PartialDependenceDisplay.from_estimator(
        model,
        X_train,
        features=['Uniformity of Cell Shape', 'Uniformity of Cell Size', 'Bare
Nuclei',
                ('Uniformity of Cell Shape', 'Uniformity of Cell Size'),
                ('Uniformity of Cell Shape', 'Bare Nuclei'),
                ('Uniformity of Cell Size', 'Bare Nuclei')
        ],
        grid_resolution=50
)

pdp.figure_.subplots_adjust(wspace=0.4, hspace=0.3)
plt.suptitle('Partial dependence of breast cancer classifier', fontsize=16)
plt.tight_layout()
plt.show()
```



Del gráfico anterior, la fila superior son los gráficos que muestran la influencia de las variables individuales y la fila inferior muestra la dependencia del diagnóstico con las combinaciones dos a dos.

Para el tercer punto, se realiza una "explicación LIME" de las 20 primeras instancias del conjunto de test:

```
from lime import lime_tabular
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=7401, n_estimators=100)
model.fit(X_train, y_train)

explainer = lime_tabular.LimeTabularExplainer(
        X_train.values,
        feature_names=X_train.columns,
        class_names=['Benign', 'Malignant'],
        mode='classification'
)

instancias = 20 # <- se piden las 20 primeras instancias!
exp = explainer.explain_instance(
        X_test.iloc[instancias],
        model.predict_proba,
        num_features=5
)
```

```
for i in range(instancias):
        exp = explainer.explain_instance(
                X_test.iloc[i],
                model.predict_proba,
                num_features=5
        )
        exp.show_in_notebook(show_table=True, show_all=False)
```

El código anterior resulta en los siguientes gráficos:


```
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(
```


```
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(
```


```
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(
```


```
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(
```

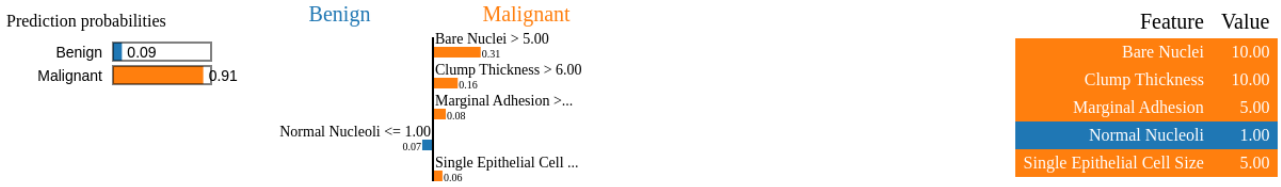
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
warnings.warn(

/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
warnings.warn(

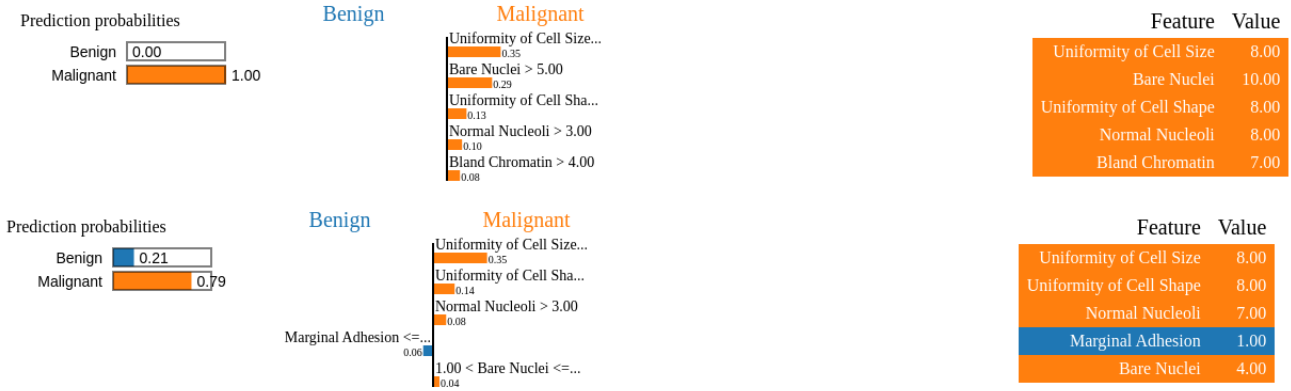
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
warnings.warn(

/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
warnings.warn(

/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(



/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(



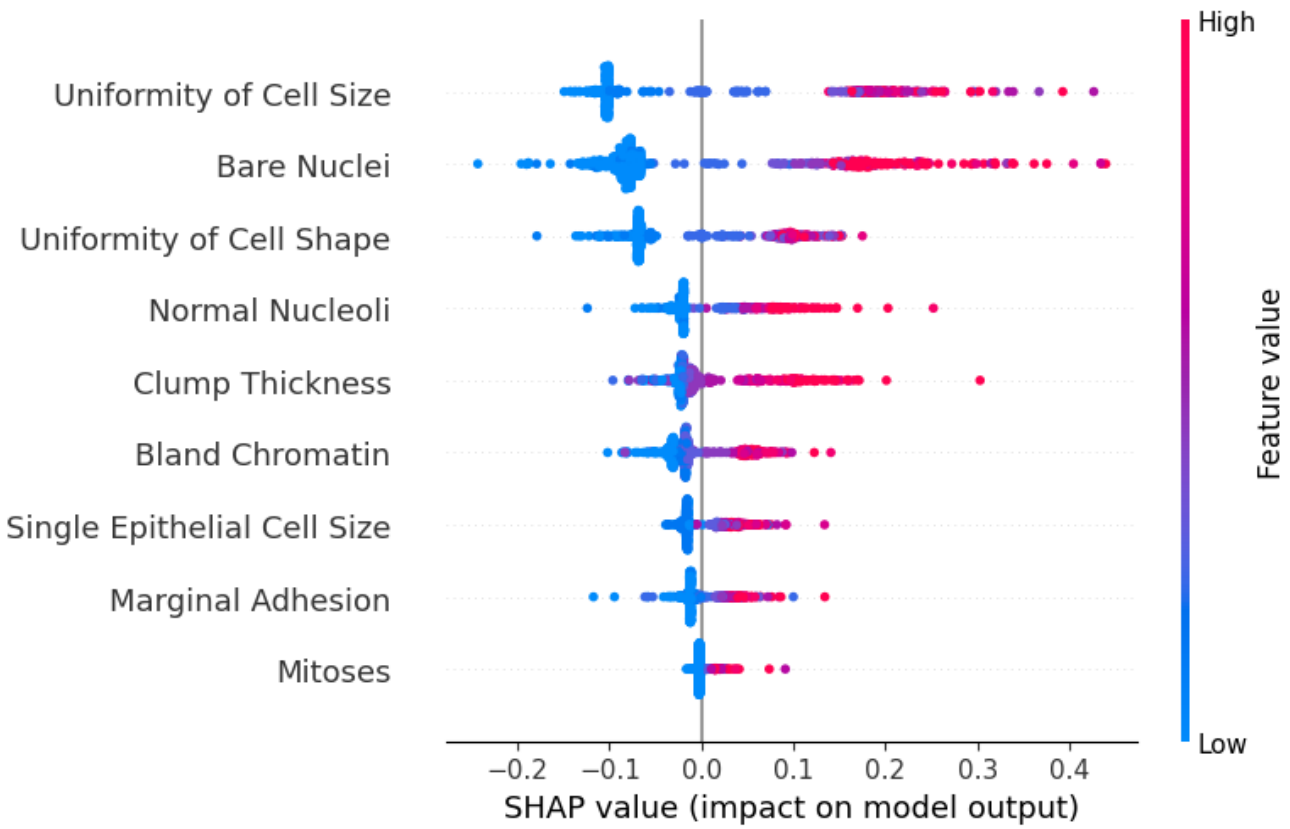
/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature r
  warnings.warn(

/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature r
  warnings.warn(

/home/mier/.local/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
  warnings.warn(

Para la parte opcional, se genera un gráfico SHAP resumiendo la importancia de cada variable para el diagnóstico:

```
import shap
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_train)
shap.summary_plot(shap_values[1], X_train)
```



Para el siguietne apartado opcional, se prepara este código:

```
shap.initjs()

# Instancia benigna
shap.force_plot(
        explainer.expected_value[1],
        shap_values[1][instancias],
        X_test.iloc[instancias],
        feature_names=X_test.columns
)

# Instancia maligna
shap.force_plot(
        explainer.expected_value[1],
        shap_values[1][instancias+1],
        X_test.iloc[instancias+1],
        feature_names=X_test.columns
)
```

Pero no funciona debido a un error de compatibilidad desconocido entre la librería, el Javascript y el Jupyter Notebook en el que se desarrolló esta práctica :(
Puesto que no se puede completar el desarrollo de esta parte opcional, tampoco se consigue la siguiente, que debería ser muy similar.

Práctica 7

Esta práctica se completa tomando como base el notebook de teoría, tal y como se dice en el enunciado de las prácticas.

Lectura inicial del dataset

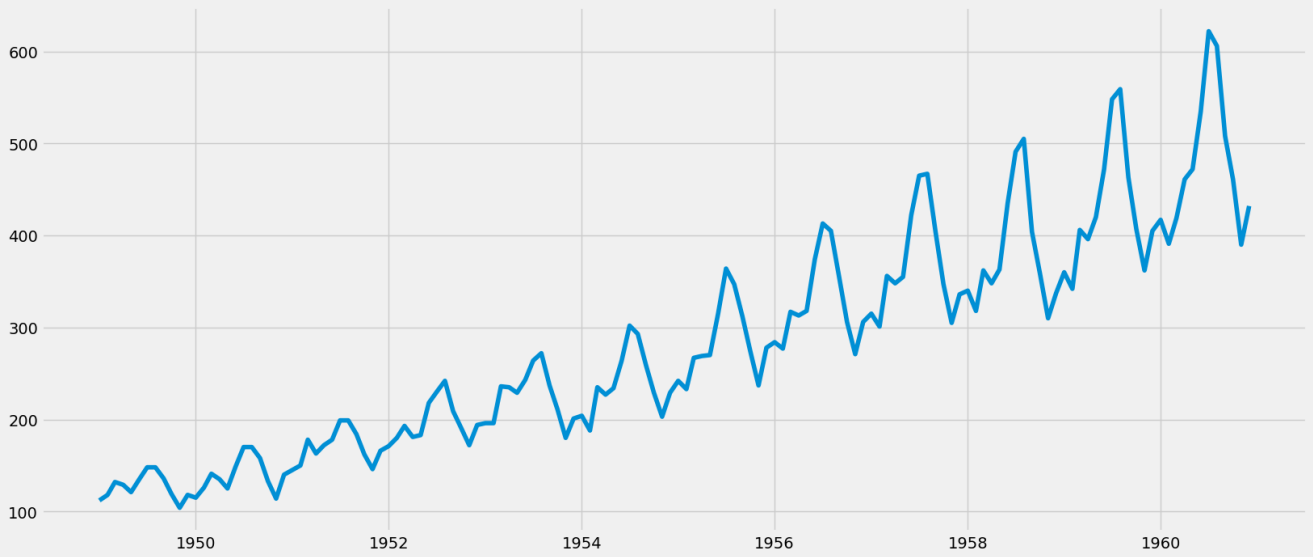
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
# Lectura sin formato
data = pd.read_csv('AirPassengers.csv')
print(data.head())
print('\n Data Types:')
print(data.dtypes)
```

Copy

```
     Month  #Passengers
0  1949-01          112
1  1949-02          118
2  1949-03          132
3  1949-04          129
4  1949-05          121

 Data Types:
Month           object
#Passengers      int64
dtype: object
```

```
import datetime
dateparse = lambda dates: datetime.datetime.strptime(dates, '%Y-%m')
data = pd.read_csv('AirPassengers.csv', parse_dates=['Month'],
index_col='Month',date_parser=dateparse)
ts = data['#Passengers'] # Serie temporal
plt.style.use('fivethirtyeight')
plt.plot(ts)
plt.show()
```

Copy



1. Determinar los parámetros del modelo ARIMA con los que se obtenga el mejor ajuste en la serie `AirPassengers`

```
import itertools
import statsmodels.api as sm

endTrain = '1957-12-31'
startTest = '1958-01-01'
y_train = ts[:endTrain]
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,
q))]
```

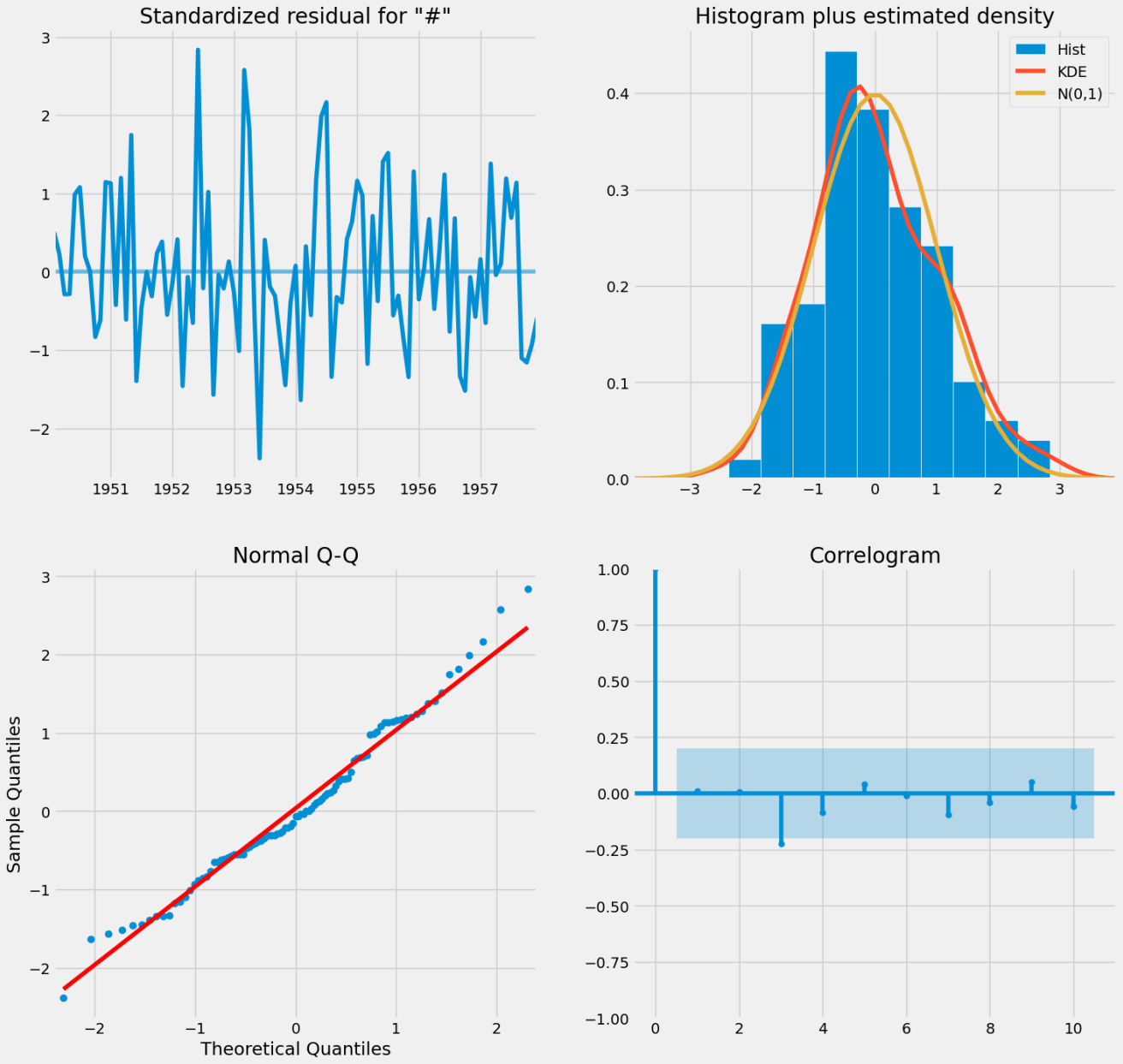
Para ajustar los hiperparámetros:

```
# Ajuste de hiperparámetros
mejor = np.inf
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y_train,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_invertibility=False)
            results = mod.fit(disp=False)
            if results.aic < mejor:
                mejor = results.aic
                mejores_parametros = [param, param_seasonal]
        except:
            continue
```

Se ajusta el modelo:

```
# Ajuste del modelo
mod = sm.tsa.statespace.SARIMAX(y_train,
                                order=mejores_parametros[0],
                                seasonal_order=mejores_parametros[1],
                                enforce_invertibility=False)
results = mod.fit()
results.plot_diagnostics(figsize=(16, 16))
plt.show()
```

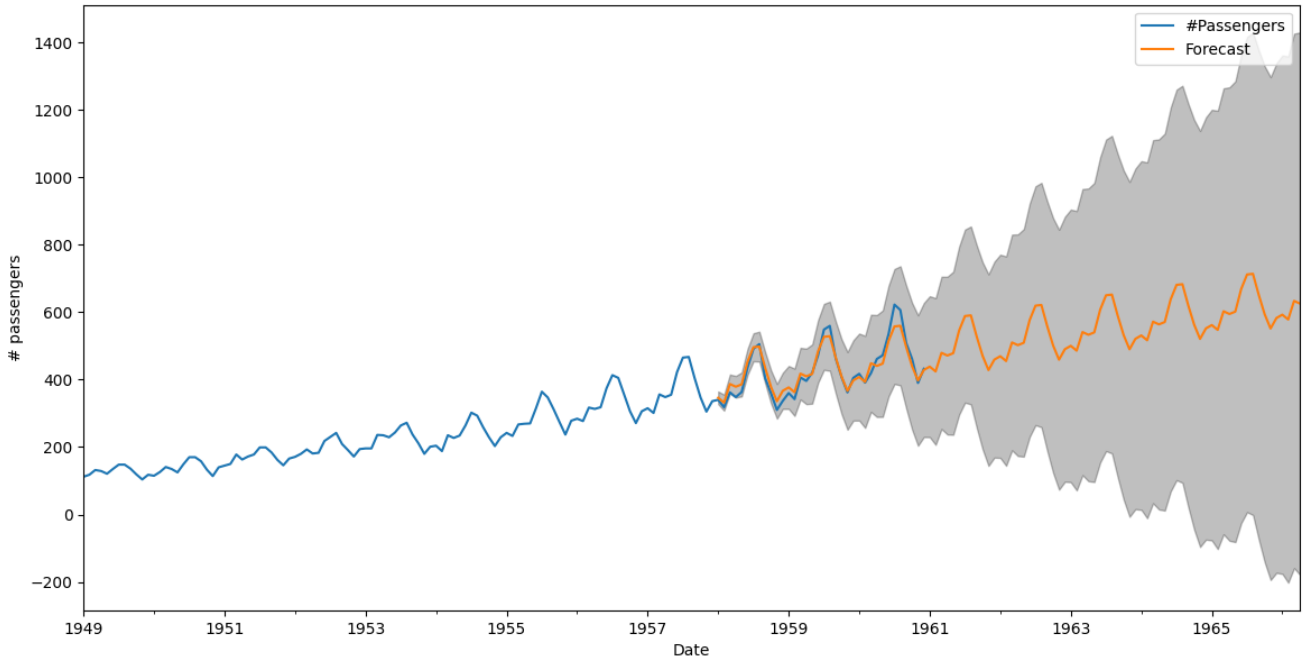
Copy



Se predicen los valores de la serie:

```
# Predicción a múltiples pasos
pred_uc = results.get_forecast(steps=100)
pred_ci = pred_uc.conf_int()
ax = data.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
```

```
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('# passengers')
plt.legend()
plt.show()
```



Se evalúan los resultados de la predicción:

```
# Evaluación de la predicción                                    Copy
y_truth = data[startTest:]['#Passengers']
# print(y_truth)
predicciones_arima = pred_uc.predicted_mean[y_truth.index]

mse = ((predicciones_arima - y_truth) ** 2).mean()
rele = (np.abs(predicciones_arima - y_truth)/y_truth*100).mean()
print('Error cuadrático medio ARIMA {}'.format(round(mse, 2)))
print('Raíz cuadrada de ECM ARIMA {}'.format(round(np.sqrt(mse), 2)))
print('Error porcentual medio ARIMA {}'.format(round(rele, 2)))

predicciones_arima = pred_uc.predicted_mean[y_truth.index]
```

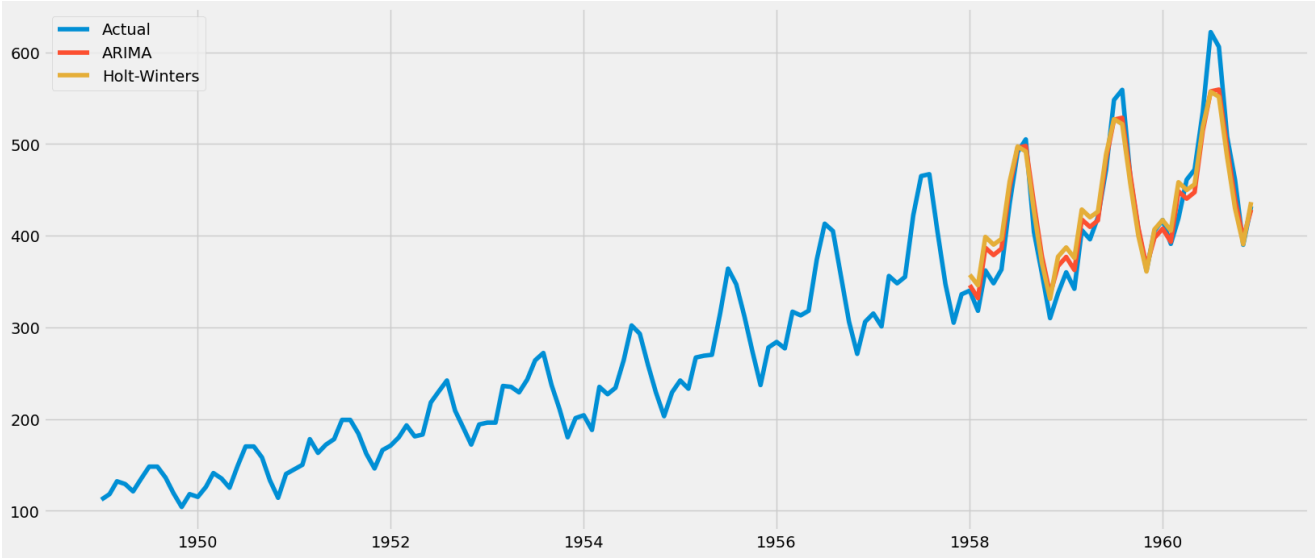
```
Error cuadrático medio ARIMA 489.84
Raíz cuadrada de ECM ARIMA 22.13
Error porcentual medio ARIMA 4.15
```

2. Ajustar el modelo Holt-Winters a esta serie y comparar sus resultados

```
# Holt-Winters
from pylab import rcParams
```

```
rcParams['figure.figsize'] = 18, 8

from statsmodels.tsa.api import ExponentialSmoothing
hw_model = ExponentialSmoothing(
    y_train, trend='add', seasonal='add', seasonal_periods=12).fit()
predicciones_hw = hw_model.forecast(12*3)

mse = ((predicciones_hw - y_truth) ** 2).mean()
rele = (np.abs(predicciones_hw - y_truth)/y_truth*100).mean()
print('Error cuadrático medio HW {}'.format(round(mse, 2)))
print('Raíz cuadrada de ECM HW {}'.format(round(np.sqrt(mse), 2)))
print('Error porcentual medio HW {}'.format(round(rele, 2)))

plt.plot(ts, label="Actual")
plt.plot(predicciones_arima, label="ARIMA")
plt.plot(predicciones_hw, label="Holt-Winters")
plt.legend()
plt.show()
```

```
Error cuadrático medio HW 695.68
Raíz cuadrada de ECM HW 26.38
Error porcentual medio HW 5.11
```

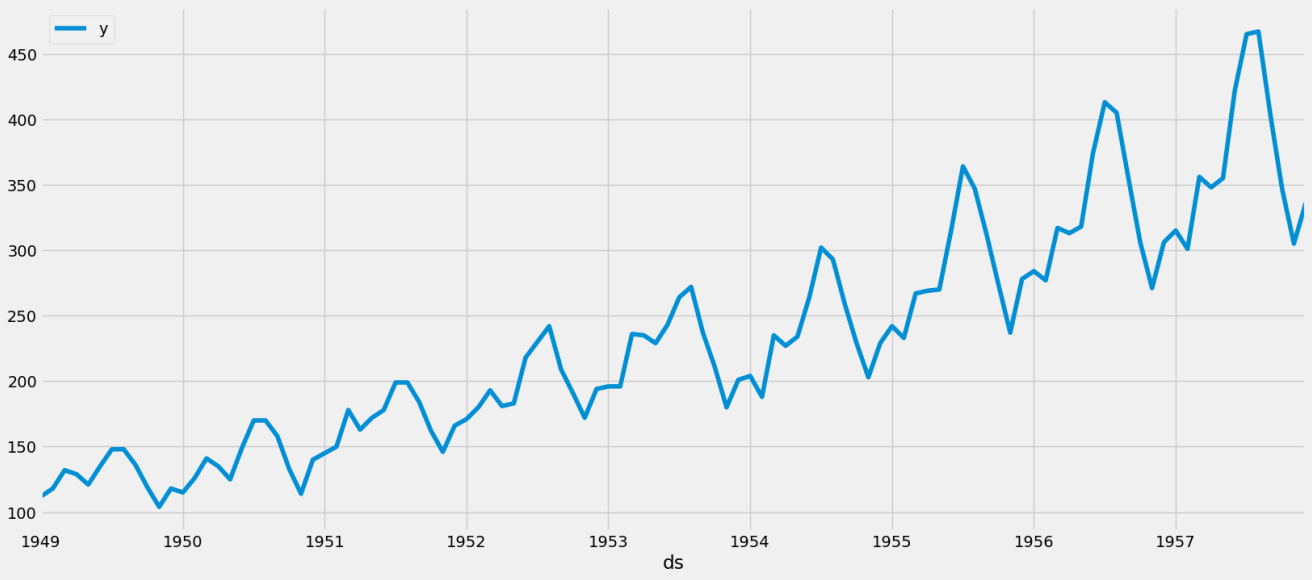


3. Ajustar Prophet a los mismos datos y comparar los resultados

Primero, se ajustan los datos que se van a utilizar como entrenamiento:

```
y_train = data[:endTrain]
ejemplo = pd.DataFrame({"ds": list(y_train.index), "y": [ v[0] for v in
y_train.values ]})
ejemplo.plot(x="ds", y="y") # show training data
```

Copy

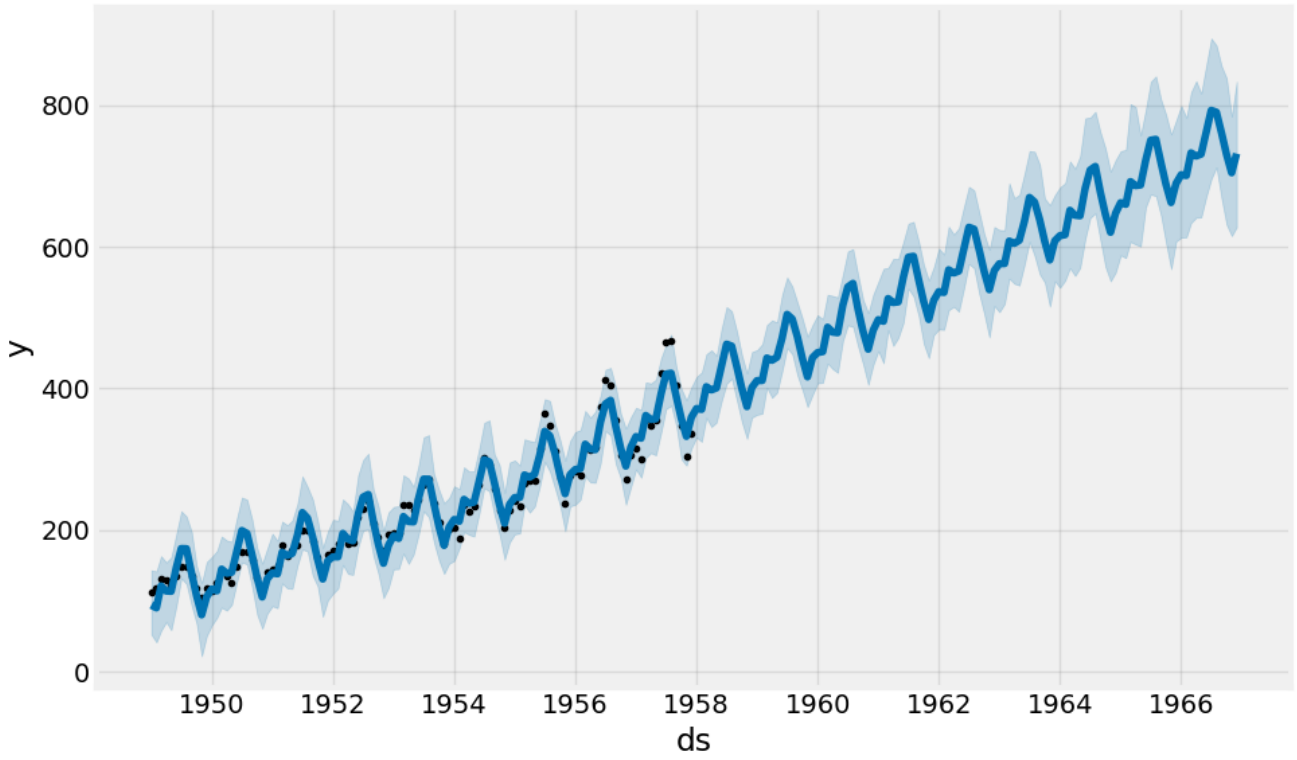


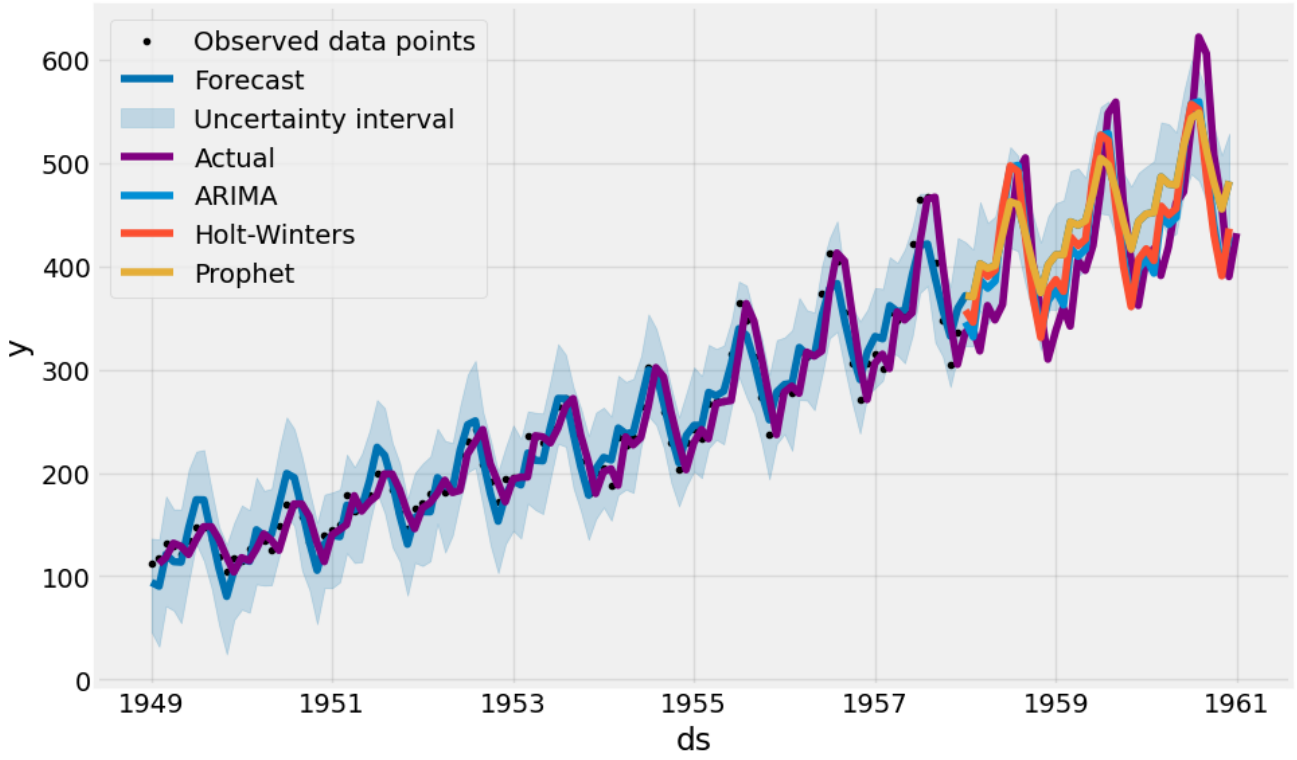
Después, se realizan las predicciones y se guardan en la variable correspondiente:

```
from prophet import Prophet
m = Prophet(interval_width=1)
m.fit(ejemplo)
forecast = m.predict(
        m.make_future_dataframe(periods=length, freq='MS')
)

fig1 = m.plot(forecast[
    ['ds', 'yhat', 'yhat_lower', 'yhat_upper']
])

print('Error cuadrático medio PROPHET {}'.format(round(mse, 2)))
print('Raíz cuadrada de ECM PROPHET {}'.format(round(np.sqrt(mse), 2)))
print('Error porcentual medio PROPHET {}'.format(round(rele, 2)))

forecast = forecast.set_index('ds')
predicciones_prophet = forecast.loc[[str(value).split("T")[0].strip() for value
in y_truth.index.values], 'yhat']

plt.plot(ts, label="Actual", color="purple")
plt.plot(predicciones_arima, label="ARIMA")
plt.plot(predicciones_hw, label="Holt-Winters")
plt.plot(predicciones_prophet, label="Prophet")
plt.legend()
plt.show()
```

Se obtienen los siguientes resultados:

```
Error cuadrático medio PROPHET 695.68
Raíz cuadrada de ECM PROPHET 26.38
Error porcentual medio PROPHET 5.11
```



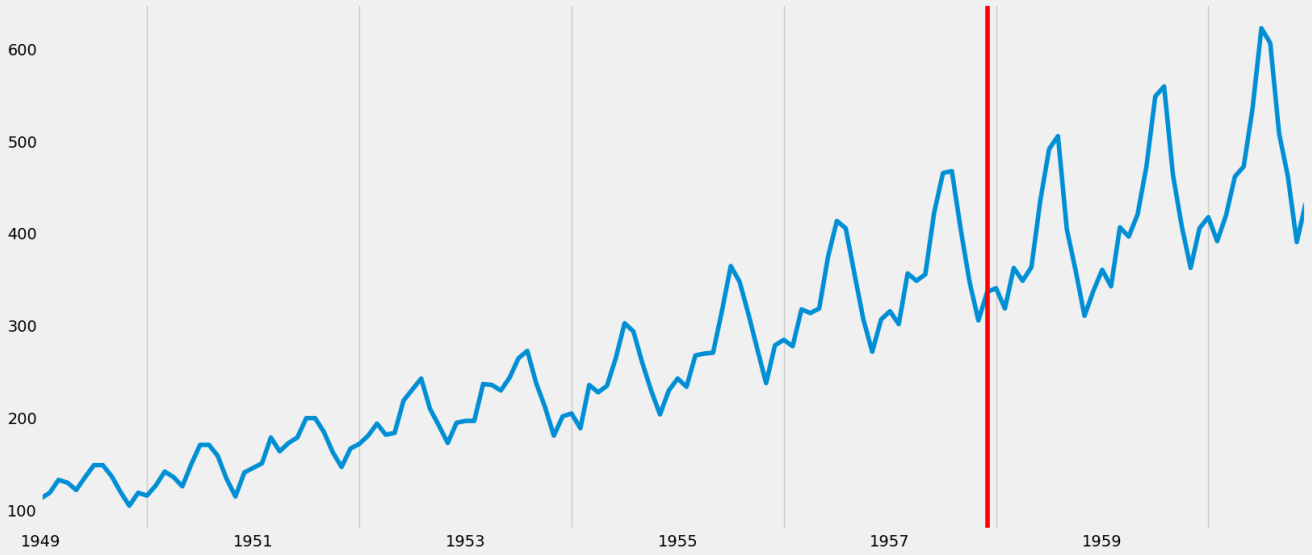


4. Ajustar DeepAR a los mismos datos y comparar los resultados

Primero, se ajustan los datos a DeepAR y se muestran en un gráfico, destacando el límite entre el conjunto de entrenamiento y el de testing:

```
from gluonts.dataset.common import ListDataset
from gluonts.dataset.util import to_pandas
from gluonts.torch.model.deepar import DeepAREstimator
from gluonts.evaluation.backtest import make_evaluation_predictions
```

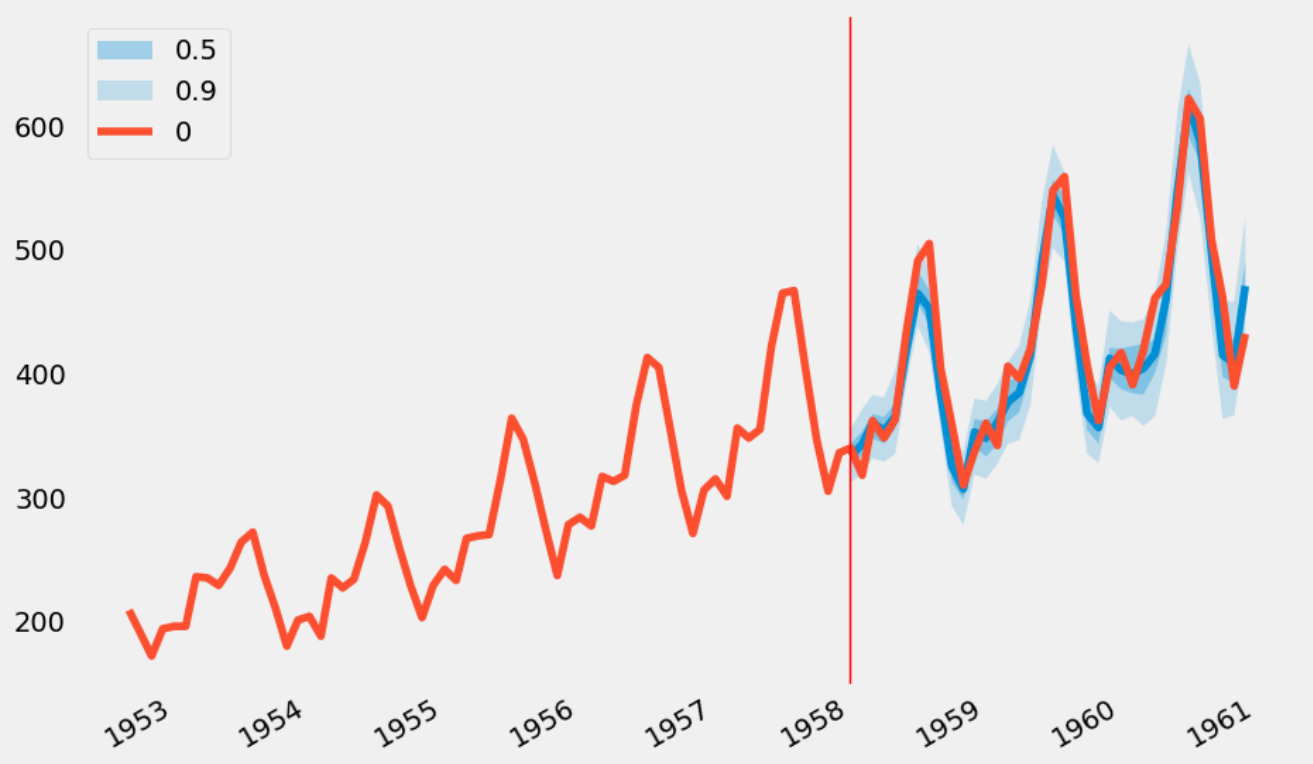
```
# Una variable "target", una fecha "start" y una frecuencia
training_data = ListDataset(
    [{"start": ts.index[0], "target": y_train['#Passengers']}], freq="M"
)

# Train + test
test_data = ListDataset(
    [{"start": ts.index[0], "target": ts}], freq="M"
)

to_pandas(test_data[0]).plot()
plt.axvline(y_train.index[-1], color='r')
plt.grid(which="both")
plt.show()
```




```
# prediction length: diferencia entre el final del train y el final del dataset
prediction_length = len(ts) - len(y_train)
estimator = DeepAREstimator(
        freq="M",
        prediction_length=prediction_length,
        cardinality=[1],
        trainer_kwargs={"max_epochs": 50, "accelerator": "cpu"}
)
predictor = estimator.train(training_data=training_data)
```

```
forecasts_it, ts_it = make_evaluation_predictions(
    dataset=test_data,  # test dataset
    predictor=predictor,  # predictor
    num_samples=100,  # number of sample paths we want for evaluation
)
forecasts = list(forecasts_it)[0]
ts = list(ts_it)[0]

print(f"Number of sample paths: {forecasts.num_samples}")
```

```
print(f"Dimension of samples: {forecasts.samples.shape}")
print(f"Start date of the forecast window: {forecasts.start_date}")
print(f"Frequency of the time series: {forecasts.freq}")

plot_prob_forecasts(ts, forecasts, 100, endTrain)
plt.show()
predicciones_deepar = forecasts.mean
```



DeepAR se ajusta MUY bien a todos los cambios en el número de pasajeros, tanto globales como estacionales, superando por mucho al resto de modelos.

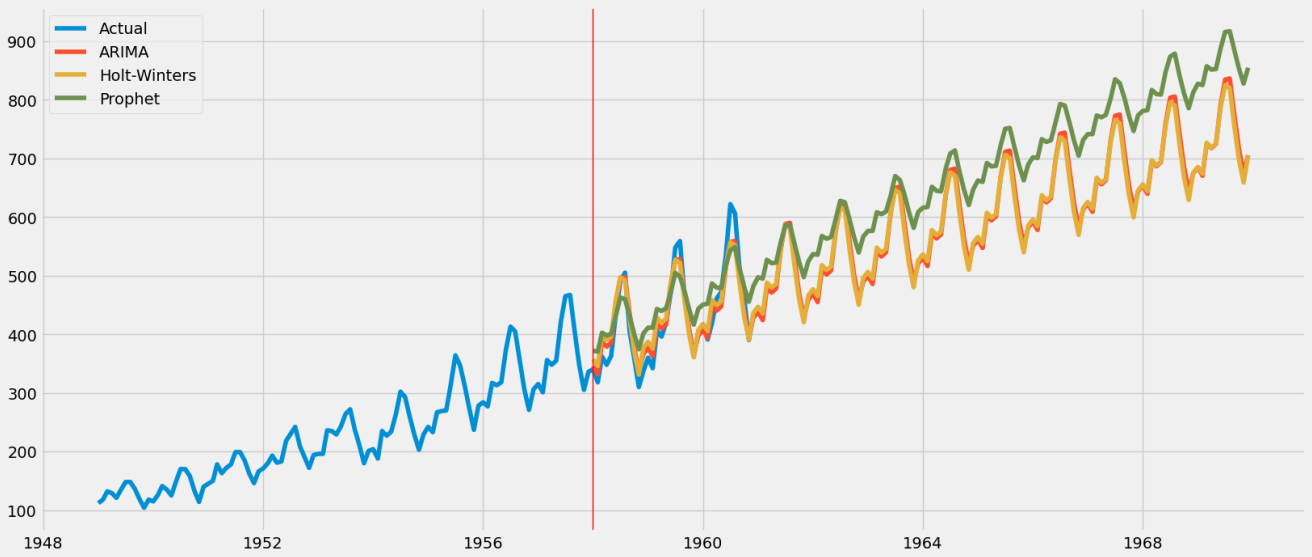
5. Comparar entre sí las predicciones de los modelos (long term)

Resultados

Para compararar todos los modelos, se representan en un gráfico simultáneamente:

Copy

```
plt.plot(ts_copy, label="Actual")
plt.plot(predicciones_arima, label="ARIMA")
plt.plot(predicciones_hw, label="Holt-Winters")
plt.plot(predicciones_prophet, label="Prophet")
plt.axvline(pd.to_datetime(endTrain), lw=1, color='r')
plt.legend()
plt.show()
```



Análisis

De la gráfica anterior, podemos realizar un escueto análisis de las predicciones:

- ARIMA y Holt-Winters realizan predicciones muy similares y además bastante conservadoras, ignorando el trend del número de pasajeros.
- Prophet analiza mejor el trend global pero se adapta peor a los cambios estacionales normales del conjunto.

Práctica 8

Preparación del código

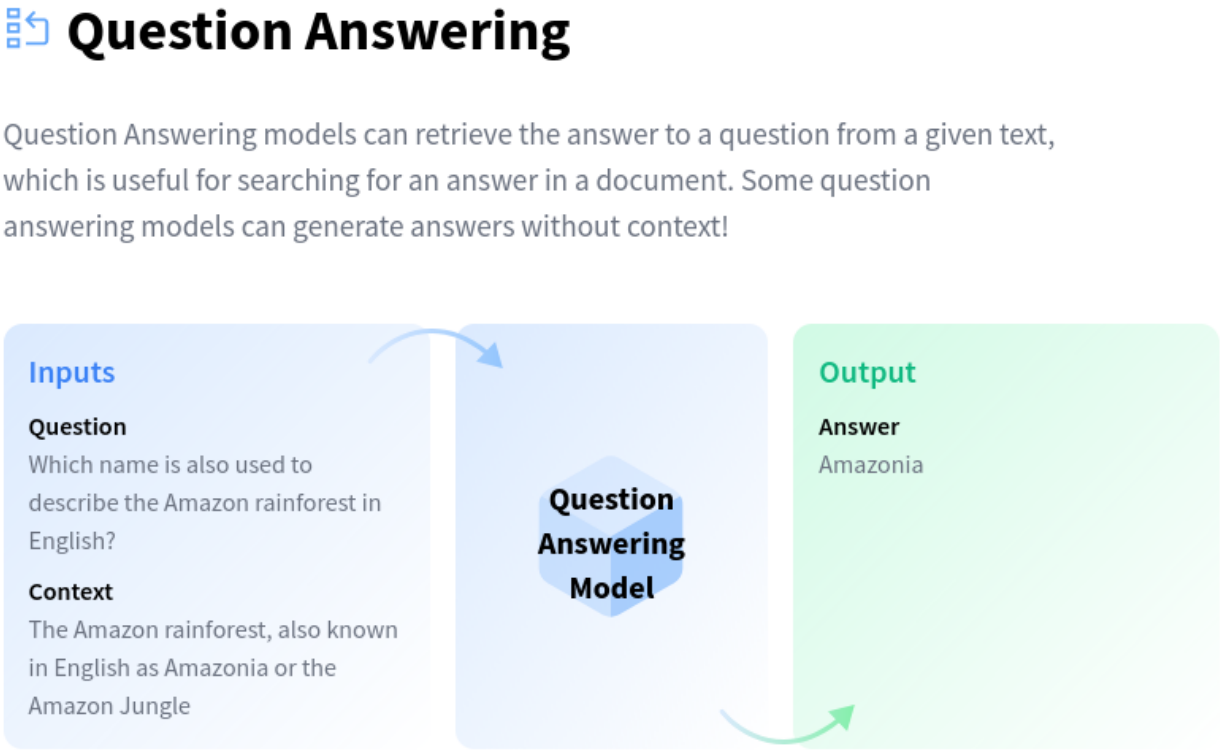
- Para la resolución de esta práctica, se escoge Google Colab como entorno de desarrollo, lo que permite ejecuciones con GPU de manera rápida y gratuita.
- Obviamente, a la hora de preparar el código, se utilizan ejemplos y código de HuggingFace, como Transformers.
- Para facilitar la ejecución, se utilizan diccionarios de Python para almacenar tanto los modelos como los datasets, para poder probar con mayor facilidad.

Tareas

Para la práctica 8, existen 5 tareas:

1. Escoger una tarea dentro de *Natural Language Processing* (NLP)

Se escoge *Question Answering*.



2. Elegir un dataset asociado a dicha tarea:

A la hora de escoger datasets, hay que tener en cuenta dos puntos clave:

- Se necesitan conjuntos que cuenten con un esquema básico de pregunta, contexto y respuesta válida.
- Puesto que la mayoría de modelos están entrenados sobre datasets derivados de *squad*, sería adecuado escoger un conjunto derivado de *squad* y otro que no lo sea, para corroborar

que el modelo es capaz de encontrar las respuestas adecuadas en el mayor número de circunstancias.

Para este análisis se escoge el dataset oficial de y el dataset de Databricks .

```
from datasets import load_dataset
squad = load_dataset("squad_v2")
dolly = load_dataset("databricks/databricks-dolly-15k")
dolly = dolly.filter(lambda x: len(x['context']) > 0)
```

Como nota adicional, se filtran y se eliminan todos los elementos que no contienen contexto.

3. Elegir al menos dos modelos para resolver la tarea

Se escogen tres modelos diferentes para contrastar resultados:

- El modelo por defecto (`distilbert-base-cased-distilled-squad`), entrenado sobre *squad*.
- Otro modelo *bert* (Bidirectional Encoder Representations from Transformers) entrenado también sobre *Squad v2*, más actualizado.
- Un último modelo totalmente diferente, que no utiliza ninguna derivación de *bert*/*robert*/*berta* ni está entrenado sobre *squad* ni sus derivados.

```
default = pipeline("question-answering") # modelo por defecto, fine-tuned para squad v2
deberta = pipeline("question-answering", model="deepset/deberta-v3-base-squad2")
splinter = pipeline("question-answering", model="tau/splinter-base")

models = [
        {'model': default, 'name': "distilbert-distilled"},
        {'model': deberta, 'name': 'deberta-v3'},
        {'model': splinter, 'name': "splinter"}
]
```

4. Evaluar sobre el dataset elegido y hacer una comparativa de los modelos

Evaluación

A la hora de evaluar, primero se ejecuta una batería de tests compuestas de muestras aleatorias de los conjuntos escogidos frente a todos los modelos.

```
import random

def addQuestions(questions):
        from_squad = squad['train'][random.randint(0, len(squad['train']))]
        from_dolly = dolly['train'][random.randint(0, len(dolly['train']))]
```

```
        questions.append({'question': from_squad['question'], 'context':
from_squad['context'], 'answer': from_squad['answers']['text'][0], 'name':
'SQUAD'})
        questions.append({'question': from_dolly['instruction'], 'context':
from_dolly['context'], 'answer': from_dolly['response'], 'name': 'DOLLY'})

questions = []
for _ in range(3):
        addQuestions(questions)

for question in questions:
        print(f"{question['name']} - Question: {question['question']}")
```

```
SQUAD - Question: What Supreme Court case ruled that a treaty provision that conflicts with the US Constitution is null and void under US law?
DOLLY - Question: What is the play Esther?
SQUAD - Question: Who was the inaugural president of the St. Luke Penny Savings Bank?
DOLLY - Question: What is the Dollarama?
SQUAD - Question: The War On Terrorism was caused by what event?
DOLLY - Question: Who is George Lucas?
```

Con las preguntas resumidas en un array, se prueban los modelos:

```
for question in questions:                                                    Copy
        print(f"{question['name']} - Expected answer: {question['answer']}")
        for model in models:
                model['result'] = model['model'](question =
question['question'], context = question['context'])
                print(f"{question['name']} - {model['name']} result:
{model['result']['answer']} (score: {model['result']['score']})")
        print()
```

```
SQUAD - Expected answer: Reid v. Covert
SQUAD - distilbert-distilled result: Reid v. Covert (score: 0.9981474876403809)
SQUAD - deberta-v3 result:  Reid v. Covert (score: 0.9999759197235107)
SQUAD - splinter result: "treaties" do not have a (score: 0.03492017835378647)

DOLLY - Expected answer: Esther is a play written in 1689 by Jean Racine, a French dramatist. It premiered on January 26, 1689 and is about the biblical Book of Esther.
DOLLY - distilbert-distilled result: a play in three acts (score: 0.13538317382335663)
DOLLY - deberta-v3 result:  a play in three acts (score: 0.8509674072265625)
DOLLY - splinter result: It was premièred (score: 1.0806551244968432e-06)

SQUAD - Expected answer: Maggie L. Walker
SQUAD - distilbert-distilled result: Maggie L. Walker (score: 0.9991340041160583)
SQUAD - deberta-v3 result:  Maggie L. Walker (score: 0.9999907016754l5)
SQUAD - splinter result: 's historic Jackson Ward became known (score: 0.02055794559419155)

DOLLY - Expected answer: Dollarama is a Canadian dollar store retail chain headquartered in Montreal. Since 2009 it is Canada's biggest retailer of items for five dollars or less.
DOLLY - distilbert-distilled result: a Canadian dollar store retail chain (score: 0.373991996049881)
DOLLY - deberta-v3 result:  Canadian dollar store retail chain (score: 0.46547165513038635)
DOLLY - splinter result: with Rossy Michael, a similar chain founded (score: 0.0002518128603696823)

SQUAD - Expected answer: the September 11, 2001 attacks
SQUAD - distilbert-distilled result: September 11, 2001 attacks (score: 0.6055676937103271)
SQUAD - deberta-v3 result:  the September 11, 2001 attacks (score: 0.9883590340614319)
SQUAD - splinter result: ) and ensure that countries considered (score: 0.0007651983760297298)

DOLLY - Expected answer: George Walton Lucas Jr. (born May 14, 1944) is an American filmmaker. Lucas is best known for creating the Star Wars and Indiana Jones franchises and founding Lucasf

After graduating from the University of Southern California in 1967, Lucas co-founded American Zoetrope with filmmaker Francis Ford Coppola. Lucas wrote and directed THX 1138 (1971), based o

Lucas's next film, the epic space opera Star Wars (1977), had a troubled production but was a surprise hit, becoming the highest-grossing film at the time, winning six Academy Awards and spa

In 1997, Lucas re-released the original Star Wars trilogy as part of a Special Edition featuring several modifications; home media versions with further changes were released in 2004 and 201
DOLLY - distilbert-distilled result: an American filmmaker (score: 0.10614709556102753)
DOLLY - deberta-v3 result:  American filmmaker. (score: 0.15211142599582672)
DOLLY - splinter result: His films are among the 100 highest-grossing movies at the North American (score: 0.005577832460403442)
```

Comparativa

Lo primero de todo y lo más obvio, es que el modelo `splinter-base` no sirve para las tareas (de la manera en la que estamos evaluando).
Lo segundo, es que el conjunto *Dolly 15k* espera respuestas que, por lo general, son demasiado largas para que los modelos escojan la respuesta correcta.

Analizando los dos modelos tipo *BERT* sobre el dataset tradicional *Squad*, sus respuestas son muy similares (si no iguales) la mayor parte de las veces:

```
SQUAD - Expected answer: Spanish
SQUAD - distilbert-distilled result: Spanish (score: 0.9637588262557983)
SQUAD - deberta-v3 result:  Spanish (score: 0.9983365535736084)

SQUAD - Expected answer: CBBC
SQUAD - distilbert-distilled result: CBBC (score: 0.9925686717033386)
SQUAD - deberta-v3 result:  CBBC. (score: 0.997228741645813)

SQUAD - Expected answer: Geilenkirchen, Germany
SQUAD - distilbert-distilled result: Geilenkirchen, Germany (score: 0.18866463005542755)
SQUAD - deberta-v3 result:  Geilenkirchen, Germany. (score: 0.9707450270652771)
```

En algunos casos, sin embargo, `deberta-v3` devuelve respuestas más acertadas y con mayor `confidence` que el modelo por defecto:

```
SQUAD - Expected answer: David Jamieson
SQUAD - distilbert-distilled result: David Jamieson (score: 0.8238303661346436)
SQUAD - deberta-v3 result:  David Jamieson (score: 0.9979233741760254)

SQUAD - Expected answer: the United States-based WWE
SQUAD - distilbert-distilled result: WWE (score: 0.6753526926040649)
SQUAD - deberta-v3 result:  the United States-based WWE, (score: 0.47005128860473633)

SQUAD - Expected answer: Detection of prey
SQUAD - distilbert-distilled result: Detection of prey, attack, capture and finally consumption (score: 0.5044756531715393)
SQUAD - deberta-v3 result:  Detection of prey, (score: 0.9998793601989746)
```

De este análisis, se deduce, que el segundo modelo, `deberta-v3-base-squad2`, es el mejor *para la tarea que se está evaluando*.

5. Elegir el mejor modelo y crear una demo para desplegarlo

Después de escoger el mejor modelo, se crea una demo sencilla haciendo uso de la librería *Gradio*.

```python
import gradio as gr

def run(question, context):
        return deberta(question = question, context = context)

demo = gr.Interface(
        fn=run,
        inputs=["text", "text"],
        outputs=["text"],
)

demo.launch(share=True)
```

A partir de la sencilla demostración anterior, se obtiene una interfaz que permite interactuar libremente con el modelo escogido:

question

Who is Thomas Jefferson?

context

Thomas Jefferson (April 13, 1743 – July 4, 1826) was an American statesman, diplomat, lawyer, architect, philosopher, and Founding Father who served as the third president of the United States from 1801 to 1809. Among the Committee of Five charged by the Second Continental Congress with authoring the Declaration of Independence, Jefferson was the Declaration's primary author. Following the American Revolutionary War and prior to becoming the nation's third president in 1801, Jefferson was the first United States secretary of state under George Washington and then the nation's second vice president under John Adams.\n\nAmong the nation's Founding Fathers, Jefferson is considered unmatched in his intellectual depth and breadth. His passionate writings and advocacy for human rights, including freedom of thought, speech, and religion, were a leading inspiration behind the American Revolution, which ultimately gave rise to the American Revolutionary War, American independence, and the United States Constitution. Jefferson's ideas were globally influential in shaping and inspiring the Age of Enlightenment, which proved transformational in the late 17th and 18th centuries. He was a leading proponent of democracy, republicanism, and individual rights, and produced formative documents and decisions at the state, national, and international levels.\n\nDuring the American Revolution, Jefferson represented Virginia in the Second Continental Congress in Philadelphia, which adopted the Declaration of Independence on July 4, 1776. As a Virginia legislator, he drafted a state law for religious freedom. He served as the second Governor of Virginia from 1779 to 1781, during the Revolutionary War. In 1785, Jefferson was appointed the United States Minister to France, and subsequently, the nation's first secretary of state under President George Washington from 1790 to 1793. Jefferson and James Madison organized the Democratic-Republican Party to oppose the Federalist Party during the formation of the First Party System. With Madison, he anonymously wrote the Kentucky and Virginia Resolutions in 1798 and 1799, which sought to strengthen states' rights by nullifying the federal Alien and Sedition Acts.\n\nJefferson and Federalist John Adams became friends as well as political rivals, serving in the Continental Congress and drafting the Declaration of Independence together. In the 1796

Clear

Submit

output

{'score': 0.4891199469566345, 'start': 3723, 'end': 3767, 'answer': ' a plantation owner, lawyer, and politician,'}

Flag