Clase PREDICT

[ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import glob
import warnings

# SCIPY
from scipy.stats import poisson, skellam
from tabulate import tabulate
# SCIKIT LEARN
# label encoder
from sklearn.preprocessing import LabelEncoder
# Statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.iolib.summary import Summary

warnings.filterwarnings(action='ignore')

content = '/content/drive'
from google.colab import drive, files
drive.mount(content)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:

```
class predict:
  def __init__(self) -> None:
    self.content = '/content/drive'
    self.folder_data = str
    self.division = str
    self.data = pd.DataFrame

  def look_up(self, folder_data : str, division : str, columns : list, show_info␣
↪= False) -> pd.DataFrame:
    self.folder_data = folder_data
    patron = '*.xls*'
    nombres_archivos = []
    # columnas = ['Div','Date','HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR',␣
↪'HTHG','HTAG','HS', 'AS', 'HST', 'AST']
    for archivo in glob.glob(os.path.join(self.content + '/' + folder_data␣
↪,patron)):
      nombre_archivo = os.path.basename(archivo)
      nombres_archivos.append(nombre_archivo)
```

```
   nombres_archivos.sort()

   dfs = []
   #for i in range(12,len(nombres_archivos)):
   for i in range(len(nombres_archivos)):
     df = pd.read_excel('/content/drive/' + folder_data + nombres_archivos[i],␣
↪sheet_name=division, usecols=columns)
     dfs.append(df)


   data = dfs[0]
   for i in range(1,len(dfs)):
     data = pd.concat([data, dfs[i]], ignore_index=True)

   if show_info:
     print(data.info())

   self.data = data
   return data

 def distribution(self, show_histo = False, show_description = True) -> tuple:
   data = self.data
   data['FTR'].value_counts().plot(kind='bar')
   plt.title('Frecuencia de Victorias (Local, Visita y Empate)')
   plt.xlabel('Resultado')
   plt.ylabel('Frecuencia')

   plt.savefig('/content/histogram.pdf', format='pdf')
   files.download('/content/histogram.pdf')
   if show_histo:
     plt.show()
   #data.value_counts(data['FTR'])
   cat_cols = data.select_dtypes(include=['datetime64[ns]', 'object']).columns
   nums_cols = data.select_dtypes(exclude=['datetime64[ns]', 'object']).columns

   if show_description:
     print(data[nums_cols].describe())
   return nums_cols, cat_cols


 def correlation_plot(self, full_description = False, show_corr = False) ->␣
↪None:
   data = self.data
   data = data.copy()
   for col in data.select_dtypes(include='O'):
     data[col] = LabelEncoder().fit_transform(data[col])
   plt.figure(figsize=(10,8))
   if full_description:
     cmap = sns.diverging_palette(245, 15, as_cmap=True)
     sns.heatmap(data.corr(), cmap=cmap, center=0, annot=True, linewidths=.5)
   else:
     sns.heatmap(data.corr(), cmap='Reds')


 def goals_info(self, title : str, show_histogram = False) -> None:
   spld = self.data[['HomeTeam', 'AwayTeam', 'FTHG', 'FTAG']]
   spld = spld.rename(columns={'FTHG' : 'HomeGoals', 'FTAG' : 'AwayGoals'})

   # para cada valor mdio de goles
   poisson_pred = np.column_stack([[poisson.pmf(i, spld.mean()[j]) for i in␣
↪range(8)] for j in range(2)])

   plt.hist(spld[['HomeGoals', 'AwayGoals']].values, range(9),
        alpha=0.7, label=['Home', 'Away'], density=True, color=["xkcd:carolina␣
↪blue", "xkcd:pistachio"])

   # add lines for the Poisson distributions
   pois1, = plt.plot([i-0.5 for i in range(1,9)], poisson_pred[:,0],
                     linestyle='-', marker='o',label="Home", color = 'Blue')
   pois2, = plt.plot([i-0.5 for i in range(1,9)], poisson_pred[:,1],
                     linestyle='-', marker='o',label="Away", color = '#006400')

   leg=plt.legend(loc='upper right', fontsize=13, ncol=2)
   leg.set_title("Poisson           Actual        ", prop = {'size':'14',␣
↪'weight':'bold'})

   plt.xticks([i-0.5 for i in range(1,9)],[i for i in range(8)])
   plt.xlabel("Goles por Partido",size=13)
   plt.ylabel("Proporcion de Partidos",size=13)
   # title = "Goles por Partido desde la Temporada 2005-06"
   plt.title(title,size=14,fontweight='bold')
   plt.ylim([-0.004, 0.4])
   plt.tight_layout()

   plt.savefig('/content/histogram_poisson.pdf', format='pdf')
   files.download('/content/histogram_poisson.pdf')
   if show_histogram:
     plt.show()
   plt.close()

   # Ahora utilizando la distribución de Skellam
   skellam_pred = [skellam.pmf(i, spld.mean()[0], spld.mean()[1]) for i in␣
↪range(-6,8)]
```

```
   plt.hist(spld[['HomeGoals']].values - spld[['AwayGoals']].values,␣
↪range(-6,8),
        alpha=0.7, label='Actual',density=True)
   plt.plot([i+0.5 for i in range(-6,8)], skellam_pred,
                     linestyle='-', marker='o',label="Skellam", color =␣
↪'#CD5C5C')
   plt.legend(loc='upper right', fontsize=13)
   plt.xticks([i+0.5 for i in range(-6,8)],[i for i in range(-6,8)])
   plt.xlabel("Goles del Equipo Local - Goles del Equipo Visitante",size=13)
   plt.ylabel("Proporcion de Partidos",size=13)
   plt.title("Diferencia de Goles Temporadas␣
↪2016-2024",size=14,fontweight='bold')
   plt.ylim([-0.004, 0.26])
   plt.tight_layout()

   plt.savefig('/content/histogram_skellam.pdf', format='pdf')
   files.download('/content/histogram_skellam.pdf')
   if show_histogram:
     plt.show()
   plt.close()

   self.model_data = spld


 def model_generator(self):
   modelo_goles = pd.concat([self.model_data[['HomeTeam', 'AwayTeam',␣
↪'HomeGoals']].assign(home=1).rename(
             columns={'HomeTeam':'team', 'AwayTeam':'opponent','HomeGoals':
↪'goals'}),
             self.model_data[['AwayTeam','HomeTeam','AwayGoals']].
↪assign(home=0).rename(
             columns={'AwayTeam':'team', 'HomeTeam':'opponent','AwayGoals':
↪'goals'})])

   modelo_poisson = smf.glm(formula="goals ~ home + team + opponent",␣
↪data=modelo_goles,
                           family=sm.families.Poisson()).fit()

   return modelo_poisson

 def simulate_match(self, model, homeTeam : str, awayTeam : str, max_goals=10)␣
↪-> np.array:
   self.homeTeam = homeTeam
   self.awayTeam = awayTeam
   home_goals_avg = model.predict(pd.DataFrame(data={'team': homeTeam,
                                                           'opponent':␣
↪awayTeam,'home':1},
                                                     index=[1])).values[0]
   away_goals_avg = model.predict(pd.DataFrame(data={'team': awayTeam,
                                                         'opponent':␣
↪homeTeam,'home':0},
                                                   index=[1])).values[0]
   team_pred = [[poisson.pmf(i, team_avg) for i in range(0, max_goals+1)] for␣
↪team_avg in [home_goals_avg, away_goals_avg]]
   return(np.outer(np.array(team_pred[0]), np.array(team_pred[1])))

 def probability(self, prob_matrix : np.array) -> pd.DataFrame:
   prob = pd.DataFrame({self.homeTeam : [np.sum(np.tril(prob_matrix, -1))],
                        self.awayTeam : [np.sum(np.triu(prob_matrix, 1))],
                        "Empate" : [np.sum(np.diag(prob_matrix))]})
   return prob
```