```
import pandas as pd

import matplotlib.pyplot as plt

from matplotlib.ticker import MaxNLocator

import seaborn as sns

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn import datasets

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report

from sklearn.metrics import accuracy_score, recall_score

from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score

from sklearn.metrics import RocCurveDisplay

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import KNeighborsClassifier


#1. Personas que viven en pobreza estan identificado según de una linea de pobreza. Gente arriba no vive en pobreza, y gente bajo vive en pobreza

# La linea de pobreza se define en relacion con el costo para tener cosas necessario (un lugar para dormir, comida, salud, educación etc)

# INDEC hizo un estudio con mas enfocus en ingresos, aunque el archivo que estoy usando contiene mucha mas estadisticas tambien

#como donde se vive, con cuantas personas se vive, nivel de educacion etc.



excel_file = 'usu_individual_T123.xlsx'


df = pd.read_excel(excel_file)
```

```
gran_buenos_aires = 1 # se declara un variable que es igual de 1 - la numera que corresponde a gente que vive en Buenos Aires

region_filtered = df['REGION'] == gran_buenos_aires


df = df[region_filtered] #nueva dataframe, que es actualizado en respecto con el filtro de region


df = df[df['CH06'] >= 0] # filtro para que valores negativos no existan


varon_mujer = df['CH04'].value_counts()

#print("INDEX", varon_mujer.index, "VALUES", varon_mujer.values)

fig, ax = plt.subplots()


ax.bar(varon_mujer.index, varon_mujer.values) # Composición por sexo (c)

ax.set_title("Numero de Varon y Mujeres")

ax.set_xlabel("Genero")

ax.set_ylabel("Cuenta")

ax.set_xticks(varon_mujer.index)

ax.set_xticklabels(["Varon", "Mujeres"])


#plt.show()

# MATRIZ de CORRELACION (d)

fig, ax = plt.subplots()

datos = ['CH04', 'CH07', 'CH08', 'NIVEL_ED', 'ESTADO', 'CAT_INAC', 'IPCF'] # datos selecionados para crear el matriz

matriz_datos = df[datos]

matriz = matriz_datos.corr()

sns.heatmap(matriz, annot=True, cmap="coolwarm", linewidths=.5, ax=ax)

plt.tight_layout()
```

```
#plt.show() # estaba seguro si deberíamos monstar los grafícos con plt.show o no, pero si descomentas esta linea el grafico aparecerá



#NUMERO DE DESOCUPADOS

desocupados = (df['ESTADO'] == 2).sum() # cuenta de desocupados

inactivos = (df['ESTADO'] == 3).sum() # cuenta de inactivos

print("NUMERO DE DESOCUPADOS ", desocupados, "\nNUMERO DE INACTIVOS ", inactivos)



medio_valor = df['IPCF'].mean() # medio valor de IPCF

#print("MEDIO_VALOR", medio_valor)

#going to use groupby and .mean or something like that for the average



mean_values = df.groupby('ESTADO')['IPCF'].mean() # medio valor, grupado del estado



filtro = mean_values.loc[[1, 2, 3]]

filtro.index = ['Ocupado', 'Desocupado', 'Inactivo'] # Reemplazo de los nombres de el index

print(filtro)

# PARTE F
```

```
archivo = 'tabla_adulto_equiv.xlsx'


df1 = pd.read_excel(archivo, skiprows = 4, header=0) # se lee la table adulto_equiv, para que podamos usar la clave

df1.columns = ['CH06', 'Mujeres', 'Varones']

df1['CH06'] = df1['CH06'].str.replace(r'\D', '', regex=True) # evitando valores 'strings' de la columna con años

df['CH06'] = df['CH06'].astype(int) # asegurar que los valos se reconoce como numeric




import numpy as np

 # se usa numpy.select para applicar la clave de valores numericos de nuestra base de datos original

conditions_varones = [

    (df['CH06'] == 0) & (df['CH04'] == 1),

    (df['CH06'] == 1) & (df['CH04'] == 1),

    (df['CH06'] == 2) & (df['CH04'] == 1),

    (df['CH06'] == 3) & (df['CH04'] == 1),

    (df['CH06'] == 4) & (df['CH04'] == 1),

    (df['CH06'] == 5) & (df['CH04'] == 1),

    (df['CH06'] == 6) & (df['CH04'] == 1),

    (df['CH06'] == 7) & (df['CH04'] == 1),

    (df['CH06'] == 8) & (df['CH04'] == 1),

    (df['CH06'] == 9) & (df['CH04'] == 1),

    (df['CH06'] == 10) & (df['CH04'] == 1),
```

```
    (df['CH06'] == 11) & (df['CH04'] == 1),
    (df['CH06'] == 12) & (df['CH04'] == 1),
    (df['CH06'] == 13) & (df['CH04'] == 1),
    (df['CH06'] == 14) & (df['CH04'] == 1),
    (df['CH06'] == 15) & (df['CH04'] == 1),
    (df['CH06'] == 16) & (df['CH04'] == 1),
    (df['CH06'] == 17) & (df['CH04'] == 1),
    ((df['CH06'] >= 18) & (df['CH06'] <= 29) & (df['CH04'] == 1)),
    ((df['CH06'] >= 30) & (df['CH06'] <= 45) & (df['CH04'] == 1)),
    ((df['CH06'] >= 46) & (df['CH06'] <= 60) & (df['CH04'] == 1)),
    ((df['CH06'] >= 61) & (df['CH06'] <= 75) & (df['CH04'] == 1)),
    (df['CH06'] > 75) & (df['CH04'] == 1),
] # classificación de valores según de edad y genero

conditions_mujeres = [
    (df['CH06'] == 0) & (df['CH04'] == 2),
    (df['CH06'] == 1) & (df['CH04'] == 2),
    (df['CH06'] == 2) & (df['CH04'] == 2),
    (df['CH06'] == 3) & (df['CH04'] == 2),
    (df['CH06'] == 4) & (df['CH04'] == 2),
    (df['CH06'] == 5) & (df['CH04'] == 2),
    (df['CH06'] == 6) & (df['CH04'] == 2),
    (df['CH06'] == 7) & (df['CH04'] == 2),
    (df['CH06'] == 8) & (df['CH04'] == 2),
    (df['CH06'] == 9) & (df['CH04'] == 2),
    (df['CH06'] == 10) & (df['CH04'] == 2),
    (df['CH06'] == 11) & (df['CH04'] == 2),
    (df['CH06'] == 12) & (df['CH04'] == 2),
    (df['CH06'] == 13) & (df['CH04'] == 2),
```

```
    (df['CH06'] == 14) & (df['CH04'] == 2),
    (df['CH06'] == 15) & (df['CH04'] == 2),
    (df['CH06'] == 16) & (df['CH04'] == 2),
    (df['CH06'] == 17) & (df['CH04'] == 2),
    ((df['CH06'] >= 18) & (df['CH06'] <= 29) & (df['CH04'] == 2)),
    ((df['CH06'] >= 30) & (df['CH06'] <= 45) & (df['CH04'] == 2)),
    ((df['CH06'] >= 46) & (df['CH06'] <= 60) & (df['CH04'] == 2)),
    ((df['CH06'] >= 61) & (df['CH06'] <= 75) & (df['CH04'] == 2)),
    (df['CH06'] > 75) & (df['CH04'] == 2),
]

# Lista de valores numericos, tirado de adulto_equiv
choices_varones = [
    0.35, 0.37, 0.46, 0.51, 0.55, 0.60, 0.64, 0.66, 0.68, 0.69, 0.79,
    0.82, 0.85, 0.90, 0.96, 1.00, 1.03, 1.04, 1.02, 1.00, 1.00, 0.83, 0.74
]

choices_mujeres = [
    0.35, 0.37, 0.46, 0.51, 0.55, 0.60, 0.64, 0.66, 0.68, 0.69, 0.70,
    0.72, 0.74, 0.76, 0.76, 0.77, 0.77, 0.77, 0.76, 0.77, 0.76, 0.67, 0.63
]

# Si no hay un match, usamos 0.0
default_choice = 0.0

# Use numpy.select with the default choice for rows not matching any conditions
df['adulto_equiv'] = np.select(conditions_varones, choices_varones) + np.select(conditions_mujeres,
choices_mujeres, default_choice)
```

```
ad_equiv_hogar1 = df.groupby('CODUSU')['adulto_equiv'].sum().reset_index()


# Rename the aggregated column to 'ad_equiv_hogar1'
#ad_equiv_hogar1.rename(columns={'adulto_equiv': 'ad_equiv_hogar1'}, inplace=True)


# Merge 'ad_equiv_hogar1' back into your original DataFrame
df = df.merge(ad_equiv_hogar1, on='CODUSU', how='left') # anade ad_equiv de nuestra base de datos
df.rename(columns={'adulto_equiv_y': 'ad_equiv_hogar1'}, inplace=True)
df.rename(columns={'adulto_equiv_x': 'adulto_equiv'}, inplace=True) # se renombra las columnas


# Calculate ad_equiv_hogar as a DataFrame
#ad_equiv_hogar = df.groupby('CODUSU')['adulto equiv'].sum().reset_index()


canasta = 57371.05 # se declara canasta
# Filter and create the respondieron DataFrame
respondieron = df[df['ITF'] != 0].copy()


respondieron['ingreso_necesario'] = respondieron['ad_equiv_hogar1'] * canasta
respondieron['pobre'] = (respondieron['ITF'] < respondieron['ingreso_necesario']).astype(int) # se declara la columna pobreza


print("Numero de gente en pobreza: ", respondieron['pobre'].value_counts()[1])
#respondieron_grouped = respondieron.groupby('CODUSU').agg({'adulto_equiv': 'sum', 'ITF': 'first'}).reset_index()
```

```
#column_list = respondieron_grouped.columns.tolist()

#print(column_list)


#print(df[['CODUSU', 'ITF', 'ad_equiv_hogar1']])









#print(count)




df.drop(columns= ['ITF', 'DECIFR', 'IDECIFR', 'RDECIFR', 'GDECIFR', 'PDECIFR', 'ADECIFR',

            'DECCFR', 'IDECCFR','IPCF', 'RDECCFR', 'GDECCFR', 'PDECCFR', 'ADECCFR', 'PONDIH']

    # se elimina los columnas relacionados de ingresos (NOTA IMPORTANTE: No estaba seguro donde estaba el archivo codigos eph.pdf

    # asi que tomé valores según de los identificaron relacionado de ingresos de EPH_registro_1T2023, pero si esto no es correcto,

    # por favor avisamé y lo cambiaré!

        )


respondieron.drop(columns=['ITF', 'DECIFR', 'IDECIFR', 'RDECIFR', 'GDECIFR', 'PDECIFR', 'ADECIFR',

            'DECCFR', 'IDECCFR','IPCF', 'RDECCFR', 'GDECCFR', 'PDECCFR', 'ADECCFR', 'PONDIH',

                'adulto_equiv', 'ad_equiv_hogar1',

                'ingreso_necesario', 'ITF'])


y = respondieron['pobre'] # se identifica la columna de el target variable
```

```
# Assuming 'respondieron' contains your independent variables

X = respondieron[['CH04', 'CH07', 'CH08', 'NIVEL_ED', 'ESTADO', 'CAT_INAC']].copy()

# otra vez los instucciones no me queda tan claro (tal vez un resultado de mi nivel de Español),

# pero tomé los variables que usé por el confusion matrix en el primer paso, obvio hay mas variables que pueden ser considerados


X['intercept'] = 1 # declaración de intercepto


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=201) # train_test_split con test_size y semilla indicado


logit_model = LogisticRegression()

logit_model.fit(X_train, y_train)

logit_predictions = logit_model.predict(X_test) # usa logit


logit_confusion = confusion_matrix(y_test, logit_predictions)

logit_probs = logit_model.predict_proba(X_test)[:, 1]

fpr_logit, tpr_logit, _ = roc_curve(y_test, logit_probs)

roc_auc_logit = roc_auc_score(y_test, logit_probs)

logit_auc = roc_auc_score(y_test, logit_model.predict_proba(X_test)[:,1])

logit_accuracy = accuracy_score(y_test, logit_predictions)

print("La precisión del Logit: %.2f" %logit_accuracy)

print(f"Matriz de Confusion:\n{logit_confusion}")


knn = KNeighborsClassifier(n_neighbors=3) # knn, con vecinos indicado
```

```
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

knn_probs = knn.predict_proba(X_test)[:, 1]

fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_probs)

roc_auc_knn = roc_auc_score(y_test, knn_probs)


accuracy = accuracy_score(y_test, y_pred)

confusion = confusion_matrix(y_test, y_pred)


print(f"La precisión del KNN: {accuracy}") # Creo que en este caso KNN funciona mejor, porque

# los AUC scores son mas o menos iguales, pero la differencia en precisión de KNN es mas substantivo de los otros algoritmos. Tal vez para aumentar mas el valor de precisión,

# se puede escalar los datos.

print(f"Matriz de Confusion:\n{confusion}")


lda = LinearDiscriminantAnalysis(n_components=1) #lda algoritmo aplicada (no estaba tan seguro en como eligo n_components)

lda = lda.fit(X_train, y_train)

X_r = lda.transform(X_train)

y_test_pred_lda = lda.predict(X_test)

accuracy_lda = accuracy_score(y_test, y_test_pred_lda)

lda_probs = lda.predict_proba(X_test)[:, 1]

fpr_lda, tpr_lda, _ = roc_curve(y_test, lda_probs)

roc_auc_lda = roc_auc_score(y_test, lda_probs) # roc score que incluyé en los graficos por cada algorithmo

lda_confusion = confusion_matrix(y_test, y_test_pred_lda)


# Print the confusion matrix

print("Matriz de Confusion de LDA:")

print(lda_confusion)
```

```
print("La precisión del LDA: %.2f" %accuracy_lda)
```

```
plt.plot(fpr_logit, tpr_logit, color='orange', lw=2, label=f'Logistic Regression (AUC = {roc_auc_logit:.2f})')
# graficar todos de las curvas de ROC

plt.plot(fpr_knn, tpr_knn, color='green', lw=2, label=f'K-Nearest Neighbors (AUC = {roc_auc_knn:.2f})')

plt.plot(fpr_lda, tpr_lda, color='blue', lw=2, label=f'Linear Discriminant Analysis (AUC = {roc_auc_lda:.2f}')


plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('Falso Positivo')

plt.ylabel('Verdadero Positivo')

plt.title('Receiver Operating Characteristic (ROC) Curva')

plt.legend(loc='lower right') # grafico para comparar todos de las curvas de ROC

#plt.show()
```

```
norespondieron = df[df['ITF'] == 0]['ITF'] # aca tenemos norespondieron, pero no me queda claro los instrucciónes en relacion con como hacemos classificacion con norespondieron

# Pobre es una columna que no existe en norespondieron, porque norespondieron no tiene una columna de ITF. Si me falta algún detalle por favor avisamé,

#porque quero hacer este parte pero no entiendo exactamente como hacemos el classificaci´´n.


#6. En relacion con los variables, como ya mencioné no me queda tan claro si estoy usando las variable que supongamos a usar

# Los variable que usé me parece applicable, pero hay mucha mas variables que pordría ser interesante para usar
```

cuando entienda que variables deberíamos usar, puedo cambiar las variables y hacer este parte con el eliminacíon de variables que no son tan relevantes.

Me gustaría revisar los partes que no entendí, por favor avísame y lo apradecería mucho

#gracias otra vez