#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 26 10:35:01 2023

@author: ali

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

hb = pd.read\_csv('/Users/ali/Downloads/HomeBuyer.csv')

X\_hb = hb.iloc [:,[0,1]].values

y\_hb = hb.iloc [:,2].values

#pd1= pd.read\_csv('/Users/ali/Downloads/BX-Book-Ratings.csv')

try:

bxbook = pd.read\_csv('/Users/ali/Downloads/BX-Book-Ratings.csv', encoding='utf-8')

except UnicodeDecodeError:

# If utf-8 fails, try 'latin1' encoding

try:

bxbook = pd.read\_csv('/Users/ali/Downloads/BX-Book-Ratings.csv', encoding='latin1')

except UnicodeDecodeError:

# If 'latin1' also fails, try 'ISO-8859-1' encoding

bxbook = pd.read\_csv('/Users/ali/Downloads/BX-Book-Ratings.csv', encoding='ISO-8859-1')

bxbook1 = pd.read\_csv('/Users/ali/Downloads/BX-Books.csv', encoding='latin1')

bxbookuser = pd.read\_csv('/Users/ali/Downloads/BX-Users.csv', encoding='latin1')

bxbook.isna().sum()

bxbook1.isna().sum()

bxbookuser.isnull().sum()

bxbook.dtypes

bxbook1.dtypes

bxbookuser.dtypes

bxbookuser['user\_id'].nunique()

bxbook1['book\_title'].nunique()

#bxbook['isbn']= pd.to\_numeric(bxbook['isbn'])

from sklearn.preprocessing import LabelEncoder

labelen = LabelEncoder()

bxbook['isbn'] = labelen.fit\_transform(bxbook['isbn'])

bxbook1['isbn'].dtypes

bxbook1['isbn'] = labelen.fit\_transform(bxbook1['isbn'])

bxbook1.dropna()

bxbook.head()

bxbook\_sorted = bxbook.sort\_values(by = 'isbn', ascending=True)

ordered\_list\_isbn = list(range(len(bxbook['isbn'])))

print(ordered\_list\_isbn)

ordered\_list\_userid = list(range(len(bxbook['user\_id'])))

print(ordered\_list\_userid)

ordered\_list\_combined = list(range(len(bxbook)))

#bxbook1.fillna()

bxbook1\_filled = bxbook1.fillna(bxbook1.mean())

bxbook1\_filled.isnull().sum()

print(bxbook1\_filled.isnull)

com\_merge = pd.merge(bxbook,bxbook1, on='isbn')

bxbook['rating'].unique()

com\_merge.dtypes

com\_merge.drop(com\_merge['year\_of\_publication'], axis=1)

mer\_final.dtypes

bxbook.dtypes

bxbook.isnull().sum()

X=bxbook['isbn']

y=bxbook['rating']

from sklearn.linear\_model import LinearRegression

linearg = LinearRegression()

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=0)

X\_train\_2d = X\_train.values.reshape(-1, 1) # -1 means the size is inferred from the length of the array

y\_train\_2d = y\_train.values.reshape(-1, 1)

#X\_train\_2d = X\_train.reshape(-1, 1)

X\_test\_2d = X\_test.values.reshape(-1,1)

y\_test\_2d = y\_test.values.reshape(-1,1)

linearg.fit(X\_train\_2d, y\_train\_2d)

predictval = linearg.predict(X\_test\_2d)

plt.scatter(X\_train\_2d, y\_train\_2d, color = 'green')

plt.plot (X\_train\_2d, linearg.predict (X\_train\_2d), color = 'red')

plt.title ('compare Training result - isbn/rating')

plt.xlabel('isbn')

plt.ylabel('rating')

plt.show()

plt.scatter(X\_test\_2d, y\_test\_2d, color = 'brown')

plt.plot (X\_train\_2d, linearg.predict (X\_train\_2d), color = 'yellow')

plt.title ('compare Training result - isbn/rating')

plt.xlabel('isbn')

plt.ylabel('rating')

plt.show()

#--------logistic regression-------

X = bxbook.iloc [:,[0,1]].values

y = bxbook.iloc [:,2].values

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(X\_train\_2d, y\_train)

prediclog = logreg.predict(X\_test\_2d)

prediclog

from sklearn.metrics import confusion\_matrix

confuma = confusion\_matrix(y\_test\_2d, prediclog)

print(confuma)

xx, yy = np.mgrid[-5:5:.01, -5:5:.01]

grid = np.c\_[xx.ravel(), yy.ravel()]

#probs = logreg.predict\_proba(grid)[:, 1].reshape(xx.shape)

grid\_1d = grid[:, 0] # Assuming you want to use the first column of the grid

probs = logreg.predict\_proba(grid\_1d.reshape(-1, 1))[:, 1].reshape(xx.shape)

print(probs)

f, ax = plt.subplots(figsize=(8, 6))

contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",

vmin=0, vmax=1)

ax\_c = f.colorbar(contour)

ax\_c.set\_label("$P(y = 1)$")

ax\_c.set\_ticks([0, .25, .5, .75, 1])

ax.scatter(X\_test\_2d[:, 0], X\_test\_2d[:, 1],c = (y\_test\_2d == 1 ), s=50,

cmap="RdBu", vmin=-.2, vmax=1.2,

edgecolor="white", linewidth=1)

ax.set(aspect="equal",

xlim=(-5, 5), ylim=(-5, 5),

xlabel="$X\_1$", ylabel="$X\_2$")

#==============================================================================

# So now let us visualize the Test set

#==============================================================================

plt.show()

print("Number of samples in X\_train\_2d:", len(X\_train\_2d))

print("Number of samples in y\_train:", len(y\_train))

bxbook.dtypes

bxbook.describe()

#---------------------Uber fare assignment---------------

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

uber\_fare = pd.read\_csv('/Users/ali/Downloads/test 3.csv')

uber\_fare.head()

uber\_fare.columns

uber\_fare.dtypes

uber\_fare.isna().sum()

from sklearn.preprocessing import LabelEncoder

laben = LabelEncoder()

uber\_fare['key']=laben.fit\_transform(uber\_fare['key'])

uber\_fare['pickup\_datetime']=laben.fit\_transform(uber\_fare['pickup\_datetime'])

uber\_fare['passenger\_count'].unique()

#X=uber\_fare['pickup\_datetime'] # for linear regression

use1 =["key", "pickup\_datetime"]

used\_features = ['key', 'pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude']

X=uber\_fare[used\_features].values #for logistic regression

y=uber\_fare['passenger\_count']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=11)

from sklearn.linear\_model import LinearRegression

linreg = LinearRegression()

X\_train\_2d = X\_train.values.reshape(-1, 1) # -1 means the size is inferred from the length of the array

y\_train\_2d = y\_train.values.reshape(-1, 1)

#X\_train\_2d = X\_train.reshape(-1, 1)

X\_test\_2d = X\_test.values.reshape(-1,1)

y\_test\_2d = y\_test.values.reshape(-1,1)

linreg.fit(X\_train\_2d, y\_train\_2d)

predicuber = linreg.predict(X\_test\_2d)

print(predicuber)

plt.scatter(X\_train\_2d, y\_train\_2d, color = 'green')

plt.plot (X\_train\_2d, linreg.predict(X\_train\_2d), color = 'red')

plt.title ('compare Training result - pickupdate/passengercount')

plt.xlabel('pickupdate')

plt.ylabel('passenger count')

plt.show()

import seaborn as sns

sns.set(style="white", color\_codes=True)

sns.set(font\_scale=1.5)

plt.figure(figsize=(6,4))

sns.heatmap(uber\_fare.corr())

#---------logistic regression-----------------------------

from sklearn.linear\_model import LogisticRegression

logreg= LogisticRegression()

logreg.fit(X\_train, y\_train)

y\_pred= logreg.predict(X\_test)

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn import metrics

from sklearn.metrics import classification\_report

metrics.confusion\_matrix(y\_test, y\_pred)

metrics.accuracy\_score(y\_test, y\_pred)

print(classification\_report(y\_test, y\_pred))

logreg.predict\_proba(X\_test)

logreg.coef\_

logreg.intercept\_

#------dropping variable key

used\_features = [ 'pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude']

X=uber\_fare[used\_features].values #for logistic regression

#-------------Classification

from sklearn.preprocessing import StandardScaler

sc= StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

from sklearn.linear\_model import Perceptron

ppt= Perceptron(random\_state=11, n\_iter\_no\_change= 4, eta0=0.1)

ppt.fit(X\_train\_std, y\_train)

y\_pred = ppt.predict(X\_test\_std)

print("Misclassified samples : %d" % (y\_test!=y\_pred).sum())

accuracy\_score(y\_test, y\_pred)

ppt.score(X\_test\_std, y\_test)

from matplotlib.colors import ListedColormap

import matplotlib.pyplot as plt

def plot\_decision\_regions(X, y, classifier, test\_idx=None, resolution=0.02):

# setup marker generator and color map

markers = ('s', 'x', 'o', '^', 'v')

colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')

cmap = ListedColormap(colors[:len(np.unique(y))])

# plot the decision surface

x1\_min, x1\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

x2\_min, x2\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1\_min, x1\_max, resolution),

np.arange(x2\_min, x2\_max, resolution))

Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)

Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)

plt.xlim(xx1.min(), xx1.max())

plt.ylim(xx2.min(), xx2.max())

for idx, cl in enumerate(np.unique(y)):

plt.scatter(x=X[y == cl, 0],

y=X[y == cl, 1],

alpha=0.8,

c=colors[idx],

marker=markers[idx],

label=cl,

edgecolor='black')

# highlight test samples

if test\_idx:

# plot all samples

X\_test, y\_test = X[test\_idx, :], y[test\_idx]

plt.scatter(X\_test[:, 0],

X\_test[:, 1],

c='',

edgecolor='black',

alpha=1.0,

linewidth=1,

marker='o',

s=100,

label='test set')

!pip install mlxtend

from mlxtend.plotting import plot\_decision\_regions

X\_combined\_std = np.vstack((X\_train\_std, X\_test\_std))

y\_combined = np.hstack((y\_train, y\_test))

plot\_decision\_regions(X=X\_combined\_std, y=y\_combined,

clf=ppt, legend=2)

plt.xlabel('petal length [standardized]')

plt.ylabel('petal width [standardized]')

plt.legend(loc='upper left')

plt.tight\_layout()

#plt.savefig('images/03\_01.png', dpi=300)

plt.show()

#------------------------MINST Classifier-----------------------

!pip install tensorflow

# example of loading the mnist dataset

from tensorflow.keras.datasets import mnist

from matplotlib import pyplot as plt

# load dataset

(trainX, trainy), (testX, testy) = mnist.load\_data()

# summarize loaded dataset

print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))

print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

# plot first few images

for i in range(9):

# define subplot

plt.subplot(330 + 1 + i)

# plot raw pixel data

plt.imshow(trainX[i], cmap=plt.get\_cmap('gray'))

# show the figure

plt.show()

!pip install --upgrade numpy

!pip show numpy

!pip cache purge

#-------amazon data

import pandas as pd

import numpy as np

data\_ama\_train =pd.read\_csv('/Users/ali/Downloads/train3-amazon.csv')

data\_ama\_test =pd.read\_csv('/Users/ali/Downloads/test-amazon.csv')

data\_ama\_test.isnull().sum()

data\_ama\_test.dtypes

X\_train = data\_ama\_train[used\_features\_amazon].values

y\_train= data\_ama\_train['ACTION']

X\_test = data\_ama\_test[used\_features\_amazon].values

y\_test = data\_ama\_test['ACTION']

data\_ama.columns

data\_ama.dtypes

data\_ama.isnull().sum()

data\_ama.describe()

data\_ama.info()

used\_features\_amazon = ['RESOURCE', 'MGR\_ID', 'ROLE\_ROLLUP\_1', 'ROLE\_ROLLUP\_2',

'ROLE\_DEPTNAME', 'ROLE\_TITLE', 'ROLE\_FAMILY\_DESC', 'ROLE\_FAMILY',

'ROLE\_CODE']

X= data\_ama[used\_features\_amazon].values

y=data\_ama['ACTION']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=.2, random\_state=12)

from sklearn.linear\_model import LogisticRegression

logreg\_amazon = LogisticRegression()

logreg\_amazon.fit(X\_train, y\_train)

y\_pred = logreg\_amazon.predict(X\_test)

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn import metrics

from sklearn.metrics import classification\_report

metrics.confusion\_matrix(y\_test, y\_pred)

metrics.accuracy\_score(y\_test, y\_pred)

print((classification\_report(y\_test, y\_pred)))

logreg\_amazon.coef\_

logreg\_amazon.intercept\_

#-------Assignment-----

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

fluffy = pd.read\_csv('/Users/ali/Downloads/unemployment\_analysis (1).csv')

#string\_value = "7.59 lakhs"

#num\_value = float(string\_value.split()[0])

#print(num\_value)

#fluffy.columns

#fluffy.dtypes

#fluffy['1991'][1]

#fluffy.dtypes

#for i in range(len(fluffy['1991'])):

# string\_value=fluffy['1991'][i]

# num\_value =float(string\_value.split()[0])

# fluffy['1991'][i] = num\_value

#fluffy['1991']= float(fluffy['1991'])

#fluffy['1991'] = fluffy['1991'].apply(lambda x: int(x) if x.isdigit() else 0)

#fluffy['1991'] = fluffy['1991'].apply(lambda x: float(x) if isinstance(x, str) and x.isdigit() else 0)

#fluffy['1991'] = fluffy['1991'].astype(float)

#fluffy\_groupby = fluffy[['Country Name', 'Country Code']].groupby(['Country Name', 'Country Code']).sum()

#for i in range(len(fluffy['1991'])):

# string\_value=fluffy['1991'][i]

# num\_value =float(string\_value.split()[0])

# fluffy['1991'][i] = num\_value

#fluffy['1991'] = fluffy['1991'].astype(float)

for i in range(len(fluffy['1991'])):

string\_value=fluffy['1991'][i]

num\_value =float(string\_value.split()[0])

fluffy['1991'][i] = num\_value

fluffy['1991'] = fluffy['1991'].astype(float)

for i in range(len(fluffy['1992'])):

string\_value=fluffy['1992'][i]

num\_value =float(string\_value.split()[0])

fluffy['1992'][i] = num\_value

fluffy['1992'] = fluffy['1992'].astype(float)

for i in range(len(fluffy['1993'])):

string\_value=fluffy['1993'][i]

num\_value =float(string\_value.split()[0])

fluffy['1993'][i] = num\_value

fluffy['1993'] = fluffy['1993'].astype(float)

for i in range(len(fluffy['1994'])):

string\_value=fluffy['1994'][i]

num\_value =float(string\_value.split()[0])

fluffy['1994'][i] = num\_value

fluffy['1994'] = fluffy['1994'].astype(float)

for i in range(len(fluffy['1995'])):

string\_value=fluffy['1995'][i]

num\_value =float(string\_value.split()[0])

fluffy['1995'][i] = num\_value

fluffy['1995'] = fluffy['1995'].astype(float)

for i in range(len(fluffy['1996'])):

string\_value=fluffy['1996'][i]

num\_value =float(string\_value.split()[0])

fluffy['1996'][i] = num\_value

fluffy['1996'] = fluffy['1996'].astype(float)

for i in range(len(fluffy['1997'])):

string\_value=fluffy['1997'][i]

num\_value =float(string\_value.split()[0])

fluffy['1997'][i] = num\_value

fluffy['1997'] = fluffy['1997'].astype(float)

for i in range(len(fluffy['1998'])):

string\_value=fluffy['1998'][i]

num\_value =float(string\_value.split()[0])

fluffy['1998'][i] = num\_value

fluffy['1998'] = fluffy['1998'].astype(float)

for i in range(len(fluffy['1999'])):

string\_value=fluffy['1999'][i]

num\_value =float(string\_value.split()[0])

fluffy['1999'][i] = num\_value

fluffy['1999'] = fluffy['1999'].astype(float)

#for i in range(len(fluffy['2000'])):

# string\_value=fluffy['2000'][i]

# num\_value =float(string\_value.split()[0])

# fluffy['2000'][i] = num\_value

#fluffy['2000'] = pd.to\_numeric(fluffy['2000'], errors='coerce')

#fluffy['2000'] = fluffy['2000'].astype(float)

#print(fluffy['2000'])

for i in range(len(fluffy['2000'])):

value = fluffy['2000'][i]

if isinstance(value, str):

num\_value = float(value.split()[0])

fluffy['2000'][i] = num\_value

#fluffy.dtypes

fluffy['2000'] = fluffy['2000'].astype(float)

for i in range(len(fluffy['2001'])):

string\_value=fluffy['2001'][i]

num\_value =float(string\_value.split()[0])

fluffy['2001'][i] = num\_value

fluffy['2001'] = fluffy['2001'].astype(float)

for i in range(len(fluffy['2002'])):

string\_value=fluffy['2002'][i]

num\_value =float(string\_value.split()[0])

fluffy['2002'][i] = num\_value

fluffy['2002'] = fluffy['2002'].astype(float)

for i in range(len(fluffy['2003'])):

string\_value=fluffy['2003'][i]

num\_value =float(string\_value.split()[0])

fluffy['2003'][i] = num\_value

fluffy['2003'] = fluffy['2003'].astype(float)

for i in range(len(fluffy['2004'])):

string\_value=fluffy['2004'][i]

num\_value =float(string\_value.split()[0])

fluffy['2004'][i] = num\_value

fluffy['2004'] = fluffy['2004'].astype(float)

for i in range(len(fluffy['2005'])):

string\_value=fluffy['2005'][i]

num\_value =float(string\_value.split()[0])

fluffy['2005'][i] = num\_value

fluffy['2005'] = fluffy['2005'].astype(float)

for i in range(len(fluffy['2006'])):

string\_value=fluffy['2006'][i]

num\_value =float(string\_value.split()[0])

fluffy['2006'][i] = num\_value

fluffy['2006'] = fluffy['2006'].astype(float)

for i in range(len(fluffy['2007'])):

string\_value=fluffy['2007'][i]

num\_value =float(string\_value.split()[0])

fluffy['2007'][i] = num\_value

fluffy['2007'] = fluffy['2007'].astype(float)

for i in range(len(fluffy['2008'])):

string\_value=fluffy['2008'][i]

num\_value =float(string\_value.split()[0])

fluffy['2008'][i] = num\_value

fluffy['2008'] = fluffy['2008'].astype(float)

for i in range(len(fluffy['2009'])):

string\_value=fluffy['2009'][i]

num\_value =float(string\_value.split()[0])

fluffy['2009'][i] = num\_value

fluffy['2009'] = fluffy['2009'].astype(float)

for i in range(len(fluffy['2010'])):

string\_value=fluffy['2010'][i]

num\_value =float(string\_value.split()[0])

fluffy['2010'][i] = num\_value

fluffy['2010'] = fluffy['2010'].astype(float)

for i in range(len(fluffy['2011'])):

string\_value=fluffy['2011'][i]

num\_value =float(string\_value.split()[0])

fluffy['2011'][i] = num\_value

fluffy['2011'] = fluffy['2011'].astype(float)

for i in range(len(fluffy['2012'])):

string\_value=fluffy['2012'][i]

num\_value =float(string\_value.split()[0])

fluffy['2012'][i] = num\_value

fluffy['2012'] = fluffy['2012'].astype(float)

for i in range(len(fluffy['2013'])):

string\_value=fluffy['2013'][i]

num\_value =float(string\_value.split()[0])

fluffy['2013'][i] = num\_value

fluffy['2013'] = fluffy['2013'].astype(float)

for i in range(len(fluffy['2014'])):

string\_value=fluffy['2014'][i]

num\_value =float(string\_value.split()[0])

fluffy['2014'][i] = num\_value

fluffy['2014'] = fluffy['2014'].astype(float)

for i in range(len(fluffy['2015'])):

value = fluffy['2015'][i]

if isinstance(value, str):

num\_value = float(value.split()[0])

fluffy['2015'][i] = num\_value

fluffy['2015'] = fluffy['2015'].astype(float)

for i in range(len(fluffy['2016'])):

string\_value=fluffy['2016'][i]

num\_value =float(string\_value.split()[0])

fluffy['2016'][i] = num\_value

fluffy['2016'] = fluffy['2016'].astype(float)

for i in range(len(fluffy['2017'])):

string\_value=fluffy['2017'][i]

num\_value =float(string\_value.split()[0])

fluffy['2017'][i] = num\_value

fluffy['2017'] = fluffy['2017'].astype(float)

for i in range(len(fluffy['2018'])):

string\_value=fluffy['2018'][i]

num\_value =float(string\_value.split()[0])

fluffy['2018'][i] = num\_value

fluffy['2018'] = fluffy['2018'].astype(float)

for i in range(len(fluffy['2019'])):

string\_value=fluffy['2019'][i]

num\_value =float(string\_value.split()[0])

fluffy['2019'][i] = num\_value

fluffy['2019'] = fluffy['2019'].astype(float)

for i in range(len(fluffy['2020'])):

string\_value=fluffy['2020'][i]

num\_value =float(string\_value.split()[0])

fluffy['2020'][i] = num\_value

fluffy['2020'] = fluffy['2020'].astype(float)

for i in range(len(fluffy['2021'])):

string\_value=fluffy['2021'][i]

num\_value =float(string\_value.split()[0])

fluffy['2021'][i] = num\_value

fluffy['2021'] = fluffy['2021'].astype(float)

fluffy.dtypes

fluffy.columns

fgroup1.index

fgroup = fluffy.groupby('Country Code').sum()

fgroup1 = fluffy.groupby('Country Name').sum()

used\_features =['1991', '1992', '1993', '1994', '1995',

'1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004',

'2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',

'2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021']

#fgroup2 = fgroup1[used\_features, Index= 'Country Name']

fgroup1['Sum'] = fgroup1.sum(axis=1)

if fluffy.isnull() is True:

print("nathi la kai pan")

else:

print("kai to chhe")

fluffy.isnull().sum()

def data\_cleaning(df):

# Display missing values in the DataFrame

if df.isnull().any().any():

print("Missing values before cleaning:")

print(df[df.isnull().any(axis=1)])

# Replace missing values with the average of the corresponding country

df\_cleaned = df.groupby('Country Name').transform(lambda x: x.fillna(x.mean()))

# Display missing values after cleaning

print("\nMissing values after cleaning:")

print(df\_cleaned[df\_cleaned.isnull().any(axis=1)])

return df\_cleaned

else:

print("No missing values in the DataFrame.")

return df

# Example usage:

# Assuming 'fluffy' is your DataFrame

# Call the function with your DataFrame

cleaned\_fluffy = data\_cleaning(fluffy)

cleaned\_fluffy.isnull().sum()

def data\_cleaning(df):

# Display missing values in the DataFrame

if df.isnull().any().any():

print("Missing values before cleaning:")

print(df[df.isnull().any(axis=1)])

# Replace missing values with the average of the corresponding country

#df\_cleaned = df.groupby('Country Name').transform(lambda x: x.fillna(x.mean()))

df\_cleaned = df.apply(lambda x: x.fillna(x.mean()) if pd.api.types.is\_numeric\_dtype(x) else x, axis=0)

# Display missing values after cleaning

print("\nMissing values after cleaning:")

print(df\_cleaned[df\_cleaned.isnull().any(axis=1)])

else:

print("No missing values in the DataFrame.")

# Handle outliers for specific countries: 'Benin', 'Bahrain'

countries\_to\_handle\_outliers = ['Benin', 'Bahrain']

for country in countries\_to\_handle\_outliers:

country\_data = df\_cleaned[df\_cleaned['Country Name'] == country]

# Use a method to handle outliers, e.g., replace with mean/median/mode

# For example, replacing outliers with the median

median\_value = country\_data['Your\_Column\_Name'].median()

df\_cleaned.loc[df\_cleaned['Country Name'] == country, 'Your\_Column\_Name'] = df\_cleaned.loc[df\_cleaned['Country Name'] == country, 'Your\_Column\_Name'].apply(lambda x: median\_value if x > median\_value \* 3 else x)

# Create new columns: "Year" and "No. of unemployed"

df\_cleaned['Year'] = df\_cleaned.columns[4:] # Assuming year columns start from the 5th column

df\_cleaned['No. of unemployed'] = df\_cleaned.iloc[:, 4:].sum(axis=1)

# Rename columns

df\_cleaned = df\_cleaned.rename(columns={'Country Name': 'Country\_name', 'Country Code': 'Country\_code'})

return df\_cleaned

def data\_cleaning(df):

# Display missing values in the DataFrame

if df.isnull().any().any():

print("Missing values before cleaning:")

print(df[df.isnull().any(axis=1)])

# Replace missing values with the average of the corresponding country

df\_cleaned = df.groupby('Country Name').apply(lambda x: x.fillna(x.mean()))

df\_cleaned = df.apply(lambda x: x.fillna(x.mean()) if pd.api.types.is\_numeric\_dtype(x) else x, axis=0)

#df\_cleaned = df.groupby('Country Name').apply(lambda x: x.fillna(x.mean()))

# Display missing values after cleaning

print("\nMissing values after cleaning:")

print(df\_cleaned[df\_cleaned.isnull().any(axis=1)])

return df\_cleaned

else:

print("No missing values in the DataFrame.")

return df

cleaned\_fluffy = data\_cleaning(fluffy)

fluffy.dtypes

def data\_cleaning(df):

# Display missing values in the DataFrame

if df.isnull().any().any():

print("Missing values before cleaning:")

print(df[df.isnull().any(axis=1)])

# Replace missing values with the average of the corresponding country

df\_cleaned = df.apply(lambda x: x.fillna(x.mean()) if pd.api.types.is\_numeric\_dtype(x) else x, axis=0)

# Display missing values after cleaning

print("\nMissing values after cleaning:")

print(df\_cleaned[df\_cleaned.isnull().any(axis=1)])

return df\_cleaned

else:

print("No missing values in the DataFrame.")

return df

# Example usage:

# Assuming 'fluffy' is your DataFrame

# Call the function with your DataFrame

cleaned\_fluffy = data\_cleaning(fluffy)

def data\_cleaning(df):

# Display missing values in the DataFrame

if df.isnull().any().any():

print("Missing values before cleaning:")

print(df[df.isnull().any(axis=1)])

# Replace missing values with the average of the corresponding country

#df\_cleaned = df.groupby('Country Name').transform(lambda x: x.fillna(x.mean()))

df\_cleaned = df.apply(lambda x: x.fillna(x.mean()) if pd.api.types.is\_numeric\_dtype(x) else x, axis=0)

# Display missing values after cleaning

print("\nMissing values after cleaning:")

print(df\_cleaned[df\_cleaned.isnull().any(axis=1)])

# Find and replace outliers for specific countries (Benin, Bahrain)

outlier\_countries = ['Benin', 'Bahrain']

for country\_name in outlier\_countries:

country\_data = df\_cleaned[df\_cleaned['Country Name'] == country\_name]

for column in df.columns[2:]:

q1 = country\_data[column].quantile(0.25)

q3 = country\_data[column].quantile(0.75)

iqr = q3 - q1

lower\_bound = q1 - 1.5 \* iqr

upper\_bound = q3 + 1.5 \* iqr

# Replace outliers with mean/median/mode

df\_cleaned.loc[df\_cleaned['Country Name'] == country\_name, column] = df\_cleaned.loc[df\_cleaned['Country Name'] == country\_name, column].apply(lambda x: country\_data[column].mean() if lower\_bound <= x <= upper\_bound else x)

# Create a new column 'Year' and 'No. of unemployed'

#df\_cleaned['Year'] = df\_cleaned.columns[2:].astype(int).to\_list() \* (len(df\_cleaned) // len(df\_cleaned.columns[2:]))

#df\_cleaned['Year'] = sorted(df\_cleaned.columns[2:].astype(int).to\_list() \* (len(df\_cleaned) // len(df\_cleaned.columns[2:])))

#df\_cleaned['Year'] = df\_cleaned.columns[2:].astype(int).to\_list() \* (len(df\_cleaned) // len(df\_cleaned.columns[2:]))

#df\_cleaned['Year'] = df\_cleaned['Year'].iloc[:len(df\_cleaned)]

#df\_cleaned['No. of unemployed'] = df\_cleaned.iloc[:, 2:].sum(axis=1)

# Change column names

df\_cleaned = df\_cleaned.rename(columns={'Country Name': 'Country\_name', 'Country Code': 'Country\_code'})

return df\_cleaned

else:

print("No missing values in the DataFrame.")

return df

# Example usage:

# Assuming 'fluffy' is your DataFrame

# Call the function with your DataFrame

df\_pivot = data\_cleaning(fluffy)

df\_pivot.dtypes

def descriptive\_stats(df, country\_code):

# Filter data for the specified country\_code

country\_data = df[df['Country\_code'] == country\_code]

# Calculate mean, median, mode, and standard deviation for each year

stats = country\_data.describe().transpose()[['mean', '50%', 'std']].reset\_index()

stats = stats.rename(columns={'50%': 'median'})

# Find the year with minimum and maximum unemployment

min\_year = country\_data.iloc[:, 2:].idxmin(axis=1).str.extract('(\d+)').astype(int).iloc[0]

max\_year = country\_data.iloc[:, 2:].idxmax(axis=1).str.extract('(\d+)').astype(int).iloc[0]

# Find the top 5 countries with maximum unemployment in 2021

top5\_2021 = df.nlargest(5, '2021')[['Country\_name', '2021']]

# Find the top 3 countries with unemployment greater than 5 lacs in 2021

top3\_2021\_gt\_5lacs = df[df['2021'] > 5].nlargest(3, '2021')[['Country\_name', '2021']]

# Calculate the percentage change in unemployment from 1991 to 2021

change\_percentage = ((df['2021'] - df['1991']) / df['1991']) \* 100

return stats, min\_year, max\_year, top5\_2021, top3\_2021\_gt\_5lacs, change\_percentage

# Example usage:

# Assuming 'df\_pivot' is your cleaned DataFrame

# Call the function with the DataFrame and the desired country\_code

country\_code = 'WSM'

descriptive\_stats\_result = descriptive\_stats(df\_pivot, country\_code)

print(descriptive\_stats\_result)

df\_pivot1 = fluffy[fluffy['Country Code']=='AFG']

stats1 = df\_pivot1.describe().transpose()[['mean', '50%', 'std']].reset\_index()

stats1 = stats1.rename(columns={'50%': 'median'})

stats1 = stats1.rename(columns={'index': 'year'})

df2 = pd.DataFrame({'consumption': [10.51, 103.11, 55.48], 'co2\_emissions': [37.2, 19.66, 1712]},index=['Pork', 'Wheat Products', 'Beef'])

df3 = df2.idxmin()

df4 = df2.idxmin(axis=1)

df5 = fluffy.iloc[:,2:].idxmin(axis=1)

cd = df\_pivot[df\_pivot['Country\_code']=='WSM']

dey = cd.iloc[:,2:].idxmin(axis=1)

mey = cd.iloc[:,2:].idxmin(axis=1).str.extract('(\d+)').astype(int)

mxy = cd.iloc[:,2:].idxmin(axis=1).str.extract('(\d+)').astype(int).iloc[0]

myd = cd.iloc[:,2:].idxmax(axis=1).str.extract('(\d+)').astype(int).iloc[0]

top\_5 = df\_pivot.nlargest(5,'2021')[['Country\_name','2021']]

top3= df\_pivot[df\_pivot['2021']>5].nlargest(3,'2021')[['Country\_name','2021','2001']]

cp = ((df\_pivot['2021']-df\_pivot['1991'])/df\_pivot['1991']).astype(float)

cp.dtypes

minyear\_BGR = df\_pivot[df\_pivot['Country\_code']=='BGR'].iloc[:,2:].idxmin(axis=1)

minyear\_BGR1 = df\_pivot[df\_pivot['Country\_code']=='BGR'].iloc[:,2:]

df1= df\_pivot

df1['cp'] = (((df1['2021']-df1['1991'])/df1['1991'])\*100).astype(float)

df2= df1[['Country\_name', 'Country\_code', 'cp']]

cj = df\_pivot[df\_pivot['Country\_code']=='JPN']

minyear\_japan= int(cj.iloc[:,2:].idxmin(axis=1).str.extract('(\d+)').astype(int).iloc[0])

ck = df\_pivot[df\_pivot['Country\_code']=='MDA' | df\_pivot['Country\_code']=='NAC' | df\_pivot['Country\_code']=='PAN' | df\_pivot['Country\_code']=='PAK' | df\_pivot['Country\_code']=='UGA']

ck = df\_pivot[(df\_pivot['Country\_code'] == 'MDA') | (df\_pivot['Country\_code'] == 'NAC') | (df\_pivot['Country\_code'] == 'PAN') | (df\_pivot['Country\_code'] == 'PAK') | (df\_pivot['Country\_code'] == 'UGA')]

ck['cmp\_year'] = (ck['2021']-ck['2019'])/ck['2019']\*100

sig\_changes = ck[(ck['cmp\_year']>20) | (ck['cmp\_year']< -20) ]

min\_cmp= ck['cmp\_year'].idxmin

ds= minyear\_japan-1

dx= minyear\_japan+1

japn\_mod= cj[['Country\_name', str(dx) ,str(minyear\_japan)]]

import pandas as pd

import matplotlib.pyplot as plt

# Assuming df\_pivot is the cleaned DataFrame from the previous questions

# 1. Minimum Unemployment for Country 'BGR'

min\_unemployment\_BGR = df\_pivot[df\_pivot['Country Code'] == 'BGR'].iloc[:, 2:].min().idxmin()

print(f"Year of minimum unemployment for 'BGR': {min\_unemployment\_BGR}")

# 2. Create a New DataFrame with Country Names and Country Codes

df\_percentage\_change = df\_pivot[['Country Name', 'Country Code']].copy()

df\_percentage\_change['Percentage Change'] = ((df\_pivot['2021'] - df\_pivot['1991']) / df\_pivot['1991']) \* 100

# 3. Compare Minimum Unemployment for 'Japan' with Previous and Next Years

cj = df\_pivot[df\_pivot['Country Code'] == 'JPN']

minyear\_japan = int(cj.iloc[:, 2:].idxmin(axis=1).str.extract('(\d+)').iloc[0])

previous\_year = minyear\_japan - 1

next\_year = minyear\_japan + 1

comparison\_years = cj[['Country Name', str(previous\_year), str(minyear\_japan), str(next\_year)]]

print(comparison\_years)

# 4. Countries with Huge Jump in Unemployment from 2019 to 2021

countries\_of\_interest = ['MDA', 'NAC', 'PAN', 'PAK', 'UGA']

df\_interest = df\_pivot[df\_pivot['Country Code'].isin(countries\_of\_interest)]

change\_2019\_2021 = ((df\_interest['2021'] - df\_interest['2019']) / df\_interest['2019']) \* 100

significant\_changes = df\_interest[(change\_2019\_2021 > 20) | (change\_2019\_2021 < -20)]

print("Countries with significant changes from 2019 to 2021:")

print(significant\_changes)

# 5. Change in Percentage for Country 'LSO' from 1991 to 2021

country\_LSO = df\_pivot[df\_pivot['Country Code'] == 'LSO']

change\_percentage\_LSO = ((country\_LSO['2021'] - country\_LSO['1991']) / country\_LSO['1991']) \* 100

# Plotting the unemployment numbers for 'LSO' over the years

years = country\_LSO.columns[2:].astype(int)

plt.plot(years, country\_LSO.iloc[:, 2:].values.flatten(), marker='o')

plt.xlabel('Year')

plt.ylabel('Unemployment Rate')

plt.title('Unemployment Rate Over the Years for LSO')

plt.show()

print("Change in percentage for 'LSO' from 1991 to 2021:", change\_percentage\_LSO)

ck = df\_pivot[df\_pivot['Country\_code']=='AFW'].iloc[:,2:]

ck\_tran = ck.T

years = ck.columns[2:].astype(int)

years\_col = ck.iloc[:,2:].values.flatten()

alag =[[years,years\_col]]

alag1 = pd.DataFrame(alag)

df2 = pd.DataFrame({'20': [10.51], '21': [37.2]})

year1 = df2.columns[:].astype(int)

year2 = df2.iloc[:,:].values.flatten()

plt.plot(year1, year2, marker=0)

plt.xlabel('year1')

plt.ylabel('year2')

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

# Generate some sample data

data = {'Date': pd.date\_range(start='2023-01-01', periods=20),

'Value': [23, 45, 32, 56, 34, 21, 40, 55, 28, 42, 37, 48, 65, 31, 47, 50, 39, 44, 52, 29]}

df = pd.DataFrame(data)

df.set\_index('Date', inplace=True)

# Plot the original data

plt.figure(figsize=(10, 6))

plt.plot(df.index, df['Value'], label='Original Data')

# Calculate and plot the 3-point moving average

ma\_window = 3

ma\_values = df['Value'].rolling(window=ma\_window).mean()

plt.plot(df.index, ma\_values, label=f'{ma\_window}-Point Moving Average', linestyle='dashed')

plt.title('Moving Average Forecasting')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

# Extend the time index for future predictions

future\_dates = pd.date\_range(start='2023-02-01', periods=5)

df\_extended = pd.DataFrame(index=df.index.union(future\_dates))

df\_extended['Value'] = df['Value']

# Calculate and plot the 3-point moving average for the extended data

ma\_values\_extended = df\_extended['Value'].rolling(window=ma\_window).mean()

plt.figure(figsize=(10, 6))

plt.plot(df\_extended.index, ma\_values\_extended, label=f'{ma\_window}-Point Moving Average (Extended)', linestyle='dashed')

plt.scatter(future\_dates, ma\_values\_extended.loc[future\_dates], color='red', label='Future Predictions')

plt.title('Extended Moving Average Forecasting')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

js = ck\_tran.columns[0]

column\_map = {'valu'}

ck\_tran.rename(columns=column\_map, inplace=True)

print(js)

#moving average assignment-----------------------------

import matplotlib.pyplot as plt

ck = df\_pivot[df\_pivot['Country\_code']=='AFW'].iloc[:,2:]

ck\_tran = ck.T

ck\_tran.columns = ['Year']

extended\_index = pd.RangeIndex(start=0, stop=len(ck\_tran) + 2)

padded\_values = np.pad(ck\_tran['Year'].values, (0, len(extended\_index) - len(ck\_tran['Year'].values)), mode='constant', constant\_values=np.nan)

ck\_traun\_extended = pd.DataFrame(index=extended\_index, data=padded\_values, columns=['Year'])

plt.figure(figsize=(10,6))

plt.plot(ck\_tran.index, ck\_tran['Year'], label='chalse')

ma\_win = 3

ma\_val= ck\_tran['Year'].rolling(window=ma\_win).mean()

plt.plot(ck\_tran.index, ma\_val, label=f'{ma\_win}-Point Moving Average', linestyle='dashed')

plt.title('Moving Average Forecasting')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

ma\_val\_ext = ck\_traun\_extended['Year'].rolling(window=ma\_win).mean()

plt.figure(figsize=(10,6))

plt.plot(ck\_traun\_extended.index, ma\_val\_ext, label=f'{ma\_win}-Point Moving Average (Extended)', linestyle='dashed', color='brown')

plt.title('Extended Moving Average Forecasting')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

#extended\_index = pd.RangeIndex(start=0, stop=len(ck\_tran) + 2)

#ck\_traun\_extended = pd.DataFrame(index=extended\_index, data=ck\_tran['Year'].values, columns=['Year'])

#----------Assignment ends here----------

# Display the original and extended DataFrames

print("Original DataFrame:")

print(ck\_tran)

print("\nExtended DataFrame:")

print(ck\_traun\_extended)

#-----ARIMA

from statsmodels.tsa.arima.model import ARIMA

forecast = ARIMA(ck\_tran['Year']).fit().get\_forecast(steps=21)

plt.plot(ck\_tran.index, ck\_tran['Year'], label ='Oringal Datat', color='green')

plt.plot(forecast.index, forecast.predicted\_mean, label ='badlayelo', color='red')

plt.legend()

plt.show()

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data', color='green')

plt.plot(forecast.index.to\_numpy(), forecast.predicted\_mean, label='Forecast', color='red')

plt.legend()

plt.show()

from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model

model = ARIMA(ck\_tran['Year']).fit()

# Get forecast and confidence intervals

forecast = model.get\_forecast(steps=21)

forecast\_index = forecast.index

forecast\_mean = forecast.predicted\_mean

forecast\_ci = forecast.conf\_int()

# Plot original data

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data', color='green')

# Plot forecast

plt.plot(forecast\_index, forecast\_mean, label='Forecast', color='red')

# Plot confidence intervals

plt.fill\_between(forecast\_index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='red', alpha=0.2, label='Confidence Intervals')

# Add legend and show plot

plt.legend()

plt.show()

from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model

model = ARIMA(ck\_tran['Year']).fit()

# Get forecast and confidence intervals

forecast = model.get\_forecast(steps=21)

forecast\_index = pd.date\_range(start=ck\_tran.index[-1], periods=22, freq='AS-JAN')[1:] # Assuming annual frequency starting from the last date in ck\_tran

forecast\_mean = forecast.predicted\_mean

forecast\_ci = forecast.conf\_int()

# Plot original data

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data', color='green')

# Plot forecast

plt.plot(forecast\_index, forecast\_mean, label='Forecast', color='red')

# Plot confidence intervals

plt.fill\_between(forecast\_index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='red', alpha=0.2, label='Confidence Intervals')

# Add legend and show plot

plt.legend()

plt.show()

from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model

model = ARIMA(ck\_tran['Year']).fit()

# Get forecast and confidence intervals

forecast = model.get\_forecast(steps=21)

forecast\_index = pd.date\_range(start=ck\_tran.index[-1], periods=22, freq='AS-JAN')[1:] # Assuming annual frequency starting from the last date in ck\_tran

forecast\_mean = forecast.predicted\_mean

forecast\_ci = forecast.conf\_int()

# Plot original data

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data', color='green')

# Plot forecast

plt.plot(forecast\_index, forecast\_mean, label='Forecast', color='red')

# Plot confidence intervals

plt.fill\_between(forecast\_index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='red', alpha=0.2, label='Confidence Intervals')

# Add legend and show plot

plt.legend()

plt.show()

import pandas as pd

from statsmodels.tsa.arima.model import ARIMA

import matplotlib.pyplot as plt

# Assuming ck\_tran is your time series data

# Example: ck\_tran = pd.read\_csv('your\_data.csv', index\_col='Date', parse\_dates=True)

# Check and adjust time zone information

ck\_tran.index = ck\_tran.index.tz\_localize(None) # Remove time zone information if present

# Fit ARIMA model

model = ARIMA(ck\_tran['Year']).fit()

# Get forecast and confidence intervals

forecast = model.get\_forecast(steps=21)

forecast\_index = pd.date\_range(start=ck\_tran.index[-1], periods=22, freq='AS-JAN', tz=None)[1:] # Assuming annual frequency starting from the last date in ck\_tran

forecast\_mean = forecast.predicted\_mean

forecast\_ci = forecast.conf\_int()

# Plot original data

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data', color='green')

# Plot forecast

plt.plot(forecast\_index, forecast\_mean, label='Forecast', color='red')

# Plot confidence intervals

plt.fill\_between(forecast\_index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='red', alpha=0.2, label='Confidence Intervals')

# Add legend and show plot

plt.legend()

plt.show()

#---- to learn ARIMA----

from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

#series = read\_csv('/Users/ali/Downloads/shampoo-sales.csv', header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser)

#print(series.head())

#series.plot()

#pyplot.show()

import pandas as pd

# Specify the correct file path and encoding

file\_path = '/Users/ali/Downloads/shampoo-sales.csv'

encoding = 'latin1'

# Read the CSV file

series = pd.read\_csv(file\_path, header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser, encoding=encoding)

def custom\_date\_parser(date\_string):

return pd.to\_datetime(date\_string, format='%Y-%m-%d')

#series = pd.read\_csv(file\_path, header=0, parse\_dates=[0],skiprows=12, sep=',' index\_col=0, date\_parser=custom\_date\_parser, encoding=encoding)

series = pd.read\_csv(file\_path, header=0, parse\_dates=[0], skiprows=12, sep=',', index\_col=0, date\_parser=custom\_date\_parser, encoding=encoding)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

# Create a simple time series with a trend and seasonality

np.random.seed(42)

time = pd.date\_range(start='2022-01-01', periods=365, freq='D')

trend = 0.05 \* np.arange(365)

seasonality = 10 \* np.sin(2 \* np.pi \* np.arange(365) / 30)

noise = np.random.normal(0, 2, size=365)

ts\_data = pd.Series(trend + seasonality + noise, index=time)

# Plot the generated time series

plt.plot(ts\_data, label='Generated Data')

plt.title('Generated Time Series Data with Trend and Seasonality')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

# Fit ARIMA model

model = ARIMA(ts\_data, order=(1, 1, 1)) # ARIMA(p, d, q) order

result = model.fit()

# Forecast future values

forecast\_steps = 30

forecast = result.get\_forecast(steps=forecast\_steps)

# Plot results

plt.plot(ts\_data, label='Original Data')

plt.plot(result.fittedvalues, color='red', label='Fitted Values')

plt.plot(forecast.predicted\_mean, color='green', label='Forecast')

plt.fill\_between(forecast.conf\_int().index, forecast.conf\_int().iloc[:, 0], forecast.conf\_int().iloc[:, 1], color='green', alpha=0.2, label='Confidence Intervals')

plt.title('ARIMA Model on Generated Time Series Data')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

model1 = ARIMA(ck\_tran, order=(1, 1, 1)) # ARIMA(p, d, q) order

result1 = model1.fit()

# Forecast future values

forecast\_steps = 3

forecast1 = result1.get\_forecast(steps=forecast\_steps)

# Plot results

plt.plot(ck\_tran, label='Original Data')

plt.plot(result1.fittedvalues, color='red', label='Fitted Values')

plt.plot(forecast1.predicted\_mean, color='green', label='Forecast')

plt.fill\_between(forecast1.conf\_int().index, forecast1.conf\_int().iloc[:, 0], forecast1.conf\_int().iloc[:, 1], color='green', alpha=0.2, label='Confidence Intervals')

plt.title('ARIMA Model on Generated Time Series Data')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()

#--------------------ARIMA Model Assignment--------------

from statsmodels.tsa.arima.model import ARIMA

import matplotlib.pyplot as plt

import matplotlib.dates as mdates # Add this import statement

# Assuming ck\_tran is a DataFrame with a 'Year' column

# Convert the 'Year' column to a datetime index

ck\_tran.index = pd.to\_datetime(ck\_tran.index, format='%Y')

# Fit ARIMA model

model\_ck\_tran = ARIMA(ck\_tran['Year'], order=(1, 1, 1))

result\_ck\_tran = model\_ck\_tran.fit()

# Forecast future values

forecast\_steps\_ck\_tran = 3 # Adjust the number of forecast steps as needed

forecast\_ck\_tran = result\_ck\_tran.get\_forecast(steps=forecast\_steps\_ck\_tran)

# Plot results for ck\_tran

plt.plot(ck\_tran.index, ck\_tran['Year'], label='Original Data')

plt.plot(result\_ck\_tran.fittedvalues.index, result\_ck\_tran.fittedvalues, color='red', label='Fitted Values')

plt.plot(forecast\_ck\_tran.predicted\_mean.index, forecast\_ck\_tran.predicted\_mean, color='green', label='Forecast')

plt.fill\_between(forecast\_ck\_tran.conf\_int().index, forecast\_ck\_tran.conf\_int().iloc[:, 0], forecast\_ck\_tran.conf\_int().iloc[:, 1], color='green', alpha=0.2, label='Confidence Intervals')

# Specify the date format for better visualization

#plt.gca().xaxis.set\_major\_formatter(mdates.DateFormatter('%Y'))

# Rotate x-axis labels for better readability (optional)

plt.gcf().autofmt\_xdate()

plt.title('ARIMA Model on ck\_tran Data')

plt.xlabel('Year')

plt.ylabel('Value')

plt.legend()

plt.show()

##----------------------Assignment ends here---------------

##---------------comparing mape of two models--------------------------------------

forecast\_steps\_ck\_tran = 10

forecast\_ck\_tran = result\_ck\_tran.get\_forecast(steps=forecast\_steps\_ck\_tran)

# Concatenate fitted values and forecasted values

predicted\_values\_arima = np.concatenate([result\_ck\_tran.fittedvalues.values, forecast\_ck\_tran.predicted\_mean.values])

# Trim or pad the actual values to match the length of predicted\_values\_arima

actual\_values\_arima = ck\_tran['Year'].values[-len(predicted\_values\_arima):]

# Trim or pad the predicted values to match the length of actual\_values\_arima

predicted\_values\_arima = predicted\_values\_arima[:len(actual\_values\_arima)]

# Calculate MAPE for ARIMA model

mape\_arima = np.mean(np.abs((actual\_values\_arima - predicted\_values\_arima) / actual\_values\_arima)) \* 100

# Calculate MAPE for Moving Average model

mape\_ma = np.mean(np.abs((ck\_tran['Year'].values - predicted\_values\_arima) / ck\_tran['Year'].values)) \* 100

# Print MAPE values for comparison

print(f'MAPE for ARIMA model: {mape\_arima:.2f}%')

print(f'MAPE for Moving Average model: {mape\_ma:.2f}%')

# Compare MAPE values

if mape\_arima < mape\_ma:

print('ARIMA model has a lower MAPE, indicating better accuracy.')

elif mape\_arima > mape\_ma:

print('Moving Average model has a lower MAPE, indicating better accuracy.')

else:

print('Both models have similar MAPE values.')

#by=result\_ck\_tran.fittedvalues.values

#jy= forecast\_ck\_tran.predicted\_mean.values

chy= [10, 15, 20, 25, 30, 35, 40]

ty = [18, 22, 27, 31, 36]

ay = chy[-len(ty):]

ay

ty=ty[:len(ay)]

#---------- visualization assignment starts here----------------

#---------Question 1------------------

#data0= pd.DataFrame({'Country':['India', 'Aus','Pak','SA','Zim', 'BAN'],'Year':[22,23,54,85,45,72]})

data1 = df\_pivot[['Country\_name', '2021']]

data1\_sorted = data1.sort\_values(by='2021', ascending=False)

data1\_values = data1\_sorted['2021'].head(10)

data1\_con = data1\_sorted['Country\_name'].head(10)

plt.bar(data1\_con, data1\_values, align='edge', color=['green','brown'])

plt.xlabel('Countries')

plt.ylabel('Unemployment')

plt.gcf().autofmt\_xdate()

#plt.gcf().autofmt\_ydate()

plt.title('Top 10 countries in 2021')

#-----Question 2----

data2= df\_pivot

data2.columns

colums\_to\_be\_dropped= ['Country\_code', '1991', '1992', '1993', '1994', '1995',

'1996', '1997', '1998','2020', ]

data3 = data2.drop(columns=colums\_to\_be\_dropped, axis=1)

data3.columns

data3\_sorted = data3.sort\_values(by='2021', ascending=False)

data3\_values = data3\_sorted[['1999', '2000', '2001', '2002', '2003', '2004', '2005',

'2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014',

'2015', '2016', '2017', '2018', '2019']].head(5)

data3\_con = data3\_sorted['Country\_name'].head(5)

plt.plot(data3\_values, data3\_con, marker='o', linestyle='-', color='b', label='Line Maro')

plt.xlabel('Values')

plt.ylabel('countries')

plt.show()

data3\_values

data3\_con

import matplotlib.pyplot as plt

# Transpose the data3\_values DataFrame

data3\_values\_transposed = data3\_values.T

# Plotting the line graph

for country in data3\_values\_transposed.columns:

plt.plot(data3\_values\_transposed.index, data3\_values\_transposed[country], marker='o', linestyle='-', label=country)

plt.gca().xaxis.set\_major\_formatter(mdates.DateFormatter('%Y'))

plt.gcf().autofmt\_xdate

# Adding labels and title

plt.xlabel('Year')

plt.ylabel('Unemployement')

plt.title('Line Graph Example')

# Adding a legend

#plt.legend()

# Display the plot

plt.show()