```
# -*- coding: utf-8 -*-
"""Analysis_challenge_final_code_Revised.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1-_nCI4x5GldVEJCHO3UwvIbqlTVpbXwx

# ADSP 31014 IP02 Analysis Challenge
"""

import pandas as pd
import matplotlib.pyplot as plt
import Regression
from statsmodels.graphics.tsaplots import plot_acf, plot_ccf
import statsmodels.api as sm
import numpy as np
from scipy.stats import f, poisson, chi2, gaussian_kde
import seaborn as sb
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import (norm, shapiro, anderson, t)
from matplotlib.ticker import (MultipleLocator,
FormatStrFormatter,StrMethodFormatter)
import sys
import seaborn as sns
#import dataframe_image as dfi

"""### 1. Exploratory Data Analysis"""

df = pd.read_excel('UnemploymentDataset.xlsx')

df.head()

df.info()

df['BBKMGDP']=pd.to_numeric(df['BBKMGDP'],errors='coerce')

"""#### Summary Statistics"""

df.describe()

plt.figure(figsize=(12, 10))
sns.heatmap(df[['UNRATE', 'GDPC1', 'BBKMGDP', 'ICT_INVESTMENT', 'IP_INVESTMENT',
'INFLATION_ADJ', 'INFLATION_NOT_ADJ', 'CPIAUCSL', 'FEDFUNDS', 'LFPR']].corr(),
annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()

"""#### Visualizations of the Variables"""

# Set display options for numpy and pandas
np.set_printoptions(precision=10, threshold=sys.maxsize)
np.set_printoptions(linewidth=np.inf)
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)
pd.set_option('max_colwidth', None)
pd.options.display.float_format = '{:,.10f}'.format

# Load the dataset
```

```
df = pd.read_excel('UnemploymentDataset.xlsx')
df['BBKMGDP'] = pd.to_numeric(df['BBKMGDP'], errors='coerce')

# Define predictor names
predictor_name = [
    'ICT_INVESTMENT',
    'IP_INVESTMENT',
    'GDPC1',
    'INFLATION_ADJ',
    'INFLATION_NOT_ADJ',
    'LFPR',
    'FEDFUNDS',
    'LFP_TOTAL',
    'CPIAUCSL',
    'BBKMGDP'
]

# Create histograms for each predictor
for pred in predictor_name:
    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 6), dpi=200)
    values = df[pred].dropna()  # Use df instead of input_data
    counts, bins, patches = ax.hist(values,
                                    bins=20,
                                    color='lightblue',
                                    edgecolor='black',
                                    density=True,
                                    alpha=0.7)
    density = gaussian_kde(values)
    xs = np.linspace(min(bins), max(bins), 200)
    ax.plot(xs, density(xs), color='royalblue', linewidth=2)
    ax.set_xlabel(pred)
    ax.set_ylabel('Number of Observations')
    ax.yaxis.grid(True)
    ax.set_title(f'Distribution of {pred}')
    plt.tight_layout()
    plt.show()

# Plot unemployment rate over time
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 6), dpi=200)
ax.plot(df['DATE'], df['UNRATE'], linewidth=2, color='royalblue')

#COVID period shadings
ax.axvspan(pd.to_datetime('2020-03-01'),
           pd.to_datetime('2020-12-31'),
           alpha = 0.2,
           color = 'red',
           label = 'Early COVID')

ax.axvspan(pd.to_datetime('2021-01-01'),
           pd.to_datetime('2021-12-31'),
           alpha = 0.2,
           color = 'orange',
           label = 'Mid COVID')

ax.axvspan(pd.to_datetime('2022-01-01'),
           pd.to_datetime('2024-01-01'),
           alpha = 0.2,
           color = 'yellow',
           label = 'Late COVID')
```

```
ax.set_xlabel('Calendar Date')
ax.set_ylabel('UNRATE')
ax.yaxis.grid(True)
ax.set_ylim([2, 16])
ax.set_title('U.S. Unemployment Rate Trend (2014-2024) with COVID-19 period
highlighted')
ax.legend(ncol = 1, loc = 'center left', bbox_to_anchor = (1.0, 0.5))

plt.tight_layout()
plt.show()

plt.hist(df['UNRATE'])

plt.scatter(df['UNRATE'], df['GDPC1'])

plt.scatter(df[df['DATE'].dt.year <=2019]['UNRATE'], df[df['DATE'].dt.year <=2019]
['GDPC1'])

plt.scatter(df[df['DATE'].dt.year <=2019]['UNRATE'], df[df['DATE'].dt.year <=2019]
['IP_INVESTMENT'])

plt.scatter(df['UNRATE'], df['INFLATION_NOT_ADJ'])

plt.scatter(df[df['DATE'].dt.year <=2019]['UNRATE'], df[df['DATE'].dt.year <=2019]
['INFLATION_NOT_ADJ'])

plot_acf(df['UNRATE'], lags=24)

#Checking cross correlation between Unemployment rate and lagged predictor
variables
for x in df.iloc[:,5:18].columns:
    plot_ccf(df['UNRATE'], df[x], title=f'Cross correlation Unemployment rate and
{x}')

"""### 2. Data Preprocessing and Feature Selection
#### Box-Cox Transformation
"""

def BoxCoxTransform (y, p):
    if (p == 0):
        y1 = np.log(y)
    else:
        y1 = (np.power(y,p) - 1.0) / p
    return (y1)

y=df['UNRATE']
y_summary = y.describe()
# Custom histogram
custom_bins = np.arange(2, 15, 0.5)
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (16,8), dpi = 200)
ax.hist(y, bins = custom_bins, align = 'mid', color = 'green')
ax.ticklabel_format(axis = 'both', style = 'plain')
# ax.xaxis.set_major_locator(MultipleLocator(base = 5))
# ax.xaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}%'))
# ax.xaxis.set_minor_locator(MultipleLocator(base = 1))
ax.yaxis.set_major_locator(MultipleLocator(base = 15))
ax.set_xlabel('Unemployment Rate')
```

```
ax.set_ylabel('Count')
ax.grid(axis = 'both', linestyle = 'dashed')
plt.show()
# Sort the values in ascending order
y_sorted = np.sort(y)
# Calculate the hypothetical quantiles
n_sample = y_summary['count']
y_mean = y_summary['mean']
y_stddev = y_summary['std']
n_sample=len(y)
empirical_prob = np.arange(1.0, (n_sample+1.0)) / (n_sample + 0.5)
z_quantile = norm.ppf(empirical_prob, loc = y_mean, scale = y_stddev)
empirical_prob = np.arange(1.0, (n_sample+1.0)) / (n_sample + 0.5)

result_list = []

for power_lambda in np.arange(-2.0, 2.1, 0.5):
    y1 = y.apply(BoxCoxTransform, p = power_lambda)
    y1_sorted = np.sort(y1)
    y1_mean = np.mean(y1)
    y1_stddev = np.std(y1, ddof = 1)
    z_quantile = norm.ppf(empirical_prob, loc = y1_mean, scale = y1_stddev)
    pearson_corr = np.corrcoef(y1_sorted, z_quantile)[0,1]
    dist_ideal = np.mean(np.abs(y1_sorted - z_quantile)) / y1_stddev
    shapiro_test = shapiro(y1)
    anderson_test = anderson(y1, dist = 'norm')
    result_list.append([power_lambda, pearson_corr, dist_ideal, shapiro_test[0],
    anderson_test[0]])
result_df = pd.DataFrame(result_list, columns = ['Power', 'Corr', 'MAD','Shapiro',
'Anderson'])

power_lambda =  result_df.iloc[result_df['MAD'].idxmin()]['Power']
y1 = y.apply(BoxCoxTransform, p = power_lambda)
y1_sorted = np.sort(y1)
y1_mean = np.mean(y1)
y1_stddev = np.std(y1, ddof = 1)
z_quantile = norm.ppf(empirical_prob, loc = y1_mean, scale = y1_stddev)

custom_bins = np.arange(0.45,0.5,0.005)
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (16,8), dpi = 200)
ax.hist(y1, align='mid', color='green')  # Use custom bins
ax.ticklabel_format(axis='both', style='plain')

ax.xaxis.set_major_locator(MultipleLocator(base=0.01))
ax.xaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}'))
ax.xaxis.set_minor_locator(MultipleLocator(base=0.01))

ax.yaxis.set_major_locator(MultipleLocator(base=10))
ax.set_title('Lambda = ' + str(power_lambda))
ax.set_xlabel('Transformed UNRATE')
ax.set_ylabel('Number of Observation')
ax.grid(axis='both', linestyle='dashed')
plt.show()

pearson_corr = np.corrcoef(y1_sorted, z_quantile)[0,1]
dist_ideal = np.mean(np.abs(y1_sorted - z_quantile))
shapiro_test = shapiro(y1)
anderson_test = anderson(y1, dist = 'norm')
```

```
print(pearson_corr)
print(dist_ideal)
print(shapiro_test)
print(anderson_test)

df['UNRATE']=y1.astype('float')

"""#### Creating dummy variables for the year"""

#dummies=pd.get_dummies(df['DATE'].dt.year.astype('category'),dtype='float')
dummies = pd.get_dummies(df['DATE'].dt.year.astype('category'), dtype='float')
reference_year = 2024
dummies = dummies.drop(columns=[reference_year], errors='ignore')
df = pd.concat([df, dummies], axis=1)
df.columns = df.columns.astype(str)

"""#### Set up the dataset for the model"""

X_train=df.iloc[0:127,3:30].drop(columns=
['CPIAUCSL','LFP_TOTAL','LFP_MEN_20YEARSANDOLDER','LFP_WOMEN_20YEARSANDOLDER','LFP_
16TO19YEARSOLD',

'LFP_WHITE','LFP_BLACKORAFRICANAMERICAN','LFP_ASIAN','LFP_HISPANICORLATINO'])
X_train.insert(0,'Intercept',1.0)
#Predictors are 2 period lagged compared to the unemployment rate

y_train=df['UNRATE'].iloc[2:129].reset_index().drop(columns='index')['UNRATE']
y_train

"""#### Backward Selection"""

def takeFSig(s):
    return s[6]
remove_threshold = 0.1
q_show_diary = True
step_diary=[]
var_in_model = X_train.columns.tolist()
candidate_name = df.iloc[:,3:18].columns.tolist()
candidate_count=len(X_train.columns.tolist())-1
X_train_column = X_train.columns
n_sample=126
result_list = Regression.LinearRegression(X_train, y_train)
m1 = len(result_list[5])
# residual_variance = result_list[2] and residual_df = result_list[3]
SSE1 = result_list[2] * result_list[3]
step_diary.append([0, 'None', SSE1, m1] + 4 * [np.nan])
# Backward Selection Steps
for iStep in range(candidate_count):
    FTest = []
    for pred in candidate_name:
        drop_cols = [col for col in X_train_column if pred in col]
        X = X_train.drop(columns = drop_cols)
        result_list = Regression.LinearRegression(X, y_train)
        m0 = len(result_list[5])
        SSE0 = result_list[2] * result_list[3]
        df_numer = m1 - m0
        df_denom = n_sample - m1
        if (df_numer > 0 and df_denom > 0):
            FStat = ((SSE0 - SSE1) / df_numer) / (SSE1 / df_denom)
```

```
            FSig = f.sf(FStat, df_numer, df_denom)
            FTest.append([pred, SSE0, m0, FStat, df_numer, df_denom, FSig])
# Show F Test results for the current step
    if (q_show_diary):
        print('\n===== F Test Results for the Current Backward Step =====')
        print('Step Number: ', iStep)
        print('Step Diary:')
        print('[Variable Candidate | Residual Sum of Squares | N Non-Aliased
Parameters | F Stat | F DF1 | F DF2 | F Sig]')
        for row in FTest:
            print(row)
    FTest.sort(key = takeFSig, reverse = True)
    FSig = takeFSig(FTest[0])
    if (FSig >= remove_threshold):
        remove_var = FTest[0][0]
        SSE1 = FTest[0][1]
        m1 = FTest[0][2]
        step_diary.append([iStep+1] + FTest[0])
        drop_cols = [col for col in X_train_column if remove_var in col]
        X_train = X_train.drop(columns = drop_cols)
        X_train_column = X_train.columns
        print(remove_var)
        var_in_model.remove(remove_var)
        candidate_name.remove(remove_var)
    else:
        break
backward_summary = pd.DataFrame(step_diary, columns = ['Step', 'Variable Removed',
'Residual Sum of Squares', 'N Non-Aliased Parameters', 'F Stat', 'F DF1',
'F DF2', 'F Sig'])
backward_summary

"""### 3. Model
#### Linear Regression Results
"""

result_list = Regression.LinearRegression(X_train, y_train)

result_list[0]

"""#### Model Diagnostics"""

# Calculate Diagnostic Values
y_predicted = X_train.dot(result_list[0]['Estimate'])
corr_y_prediction = Regression.PearsonCorrelation(y_train, y_predicted)
R_Square = np.square(corr_y_prediction)
print(R_Square)

cols =
['IP_INVESTMENT','BBKMGDP','INFLATION_NOT_ADJ','2014','2015','2016','2017','2018','
2019','2020','2021','2022','2023']

#Checking for multicollinearity
vif_data = pd.DataFrame()
vif_data["feature"] = cols
vif_data["VIF"] = [variance_inflation_factor(X_train[cols].values, i) for i in
range(len(cols))]

vif_data
```

```
covb=result_list[1]
residual_variance = result_list[2]
residual_df = result_list[3]
model_df = len(result_list[5])
XtX_ginv = covb / residual_variance
H_matrix = X_train.dot(XtX_ginv).dot(X_train.transpose())
leverage = pd.Series(np.diag(H_matrix), index = H_matrix.index, name ='leverage')

y_residual = y_train - y_predicted
y_std_residual = y_residual / np.sqrt(residual_variance * (1.0 - leverage))
y_residual_deleted = y_residual / (1.0 - leverage)
y_residual_deleted_direct = []
residual_variance_deleted = []
for i in range(len(y_train)):
    X_drop_i = X_train.drop(index = i)
    y_drop_i = y_train.drop(index = i)
    result_list = Regression.LinearRegression(X_drop_i, y_drop_i)
    residual_variance_deleted.append(result_list[2])
    y_prediction_i = X_train.loc[i].dot(result_list[0]['Estimate'])
    y_residual_deleted_direct.append(y.loc[i] - y_prediction_i)
y_student_residual = y_residual / np.sqrt(residual_variance_deleted * (1.0 -
leverage))
diagnostic_df = pd.DataFrame({'Response': y_train,
                                  'Prediction': y_predicted,
                                  'Leverage': leverage,
                                  'Simple Residual': y_residual,
                                  'Standardized Residual': y_std_residual,
                                  'Deleted Residual': y_residual_deleted,
                                  'Studentized Residual': y_student_residual})

rmse = np.sqrt(1/n_sample*sum((y_train-y_predicted)**2))
rmse

"""#### Plot Different Residuals and Leverage"""

stats = diagnostic_df.columns
fig, axs = plt.subplots(1, 4, figsize = (12,8), sharey = True, dpi = 200)
plt.subplots_adjust(wspace = 0.15)
for j in range(4):
    col = stats[j+3]
    ax = axs[j]
    ax.boxplot(diagnostic_df[col], vert = True)
    ax.axhline(0.0, color = 'red', linestyle = ':')
    # ax.set_yticks(np.arange(-0.02,0.03,0.01))
    ax.set_xlabel(col)
    ax.grid(axis = 'y')
plt.suptitle('Box plots for 4 different types of residuals ')
plt.show()

#plot
fig, (ax0, ax1) = plt.subplots(nrows = 2, ncols = 1, dpi = 200, sharex =
True,figsize = (12,6))
ax0.hist(leverage, bins = np.arange(leverage.min(), leverage.max(), 0.05), color =
'royalblue')
ax0.axvline(model_df / n_sample, color = 'orangered', linestyle = ':')
ax0.set_xlabel('')
ax0.set_ylabel('Number of Observations')
ax0.yaxis.grid(True)
ax1.boxplot(leverage, vert = False)
```

```
ax1.axvline(model_df / n_sample, color = 'orangered', linestyle = ':')
ax1.set_xlabel('Leverage')
ax1.set_ylabel('')
ax1.set_xticks(np.arange(0, 0.6, 0.05))
ax1.xaxis.set_major_locator(MultipleLocator(base = 0.05))
ax1.xaxis.set_minor_locator(MultipleLocator(base = 0.01))
ax1.yaxis.set_major_locator(MultipleLocator(base = 1.0))
ax1.xaxis.set_major_formatter(FormatStrFormatter('%.2f'))
ax1.xaxis.grid(True)
plt.suptitle('Histogram and Boxplot of Leverage Values')
plt.title('')
plt.show()

#Identify high leverage observations
high_leverage_indx = diagnostic_df[diagnostic_df['Leverage']>=0.25].index
high_leverage=X_train.iloc[high_leverage_indx]
y_predicted[high_leverage_indx]
high_leverage

#Identify Outliers
outlier_indx =diagnostic_df[abs(diagnostic_df['Standardized Residual'])>3].index
X_train.iloc[outlier_indx]

"""#### Drop the highly influential observations and retrain the final model"""

#Drop outliers and retrain model
X_train=X_train.drop(outlier_indx,axis=0).drop(high_leverage_indx, axis=0)
y_train=y_train.drop(outlier_indx,axis=0).drop(high_leverage_indx, axis=0)

result_list = Regression.LinearRegression(X_train, y_train)

#dfi.export(result_list[0], 'Model_Results.png')
#result_list[0]

y_predicted = X_train.dot(result_list[0]['Estimate'])
y_residual=y_train-y_predicted

plt.figure(figsize = (6,4), dpi = 200)
plt.scatter(y_train, y_residual, c = 'green', s = 20, marker = 'o')
plt.xlabel('Transformed Unemployment Rate')
plt.ylabel('Simple Residual')
plt.xticks(np.arange(0.45,0.5,0.01))
plt.yticks(np.arange(-0.02,0.03,0.01))
plt.grid(axis = 'both', linestyle = 'dotted')
plt.show()

anderson(y_residual, dist = 'norm')

corr_y_prediction = Regression.PearsonCorrelation(y_train, y_predicted)
R_Square = np.square(corr_y_prediction)
print(R_Square, corr_y_prediction)

"""#### 4. Final Prediction of the Unemployment Rate of December 2024"""

X_test=df.loc[128:129,cols]
X_test.insert(0,'Intercept',1)
# X_test.insert(0,'Intercept',1)
y_12 = X_test.iloc[0,:].dot(result_list[0]['Estimate'])
print('The predicted unemployment rate for December 2024 is:')
```

```
final_pred = (np.exp(np.log(power_lambda*y_12+1)/power_lambda))
final_pred

##calculating the standard errors using given the matrix formula
X_test=X_test.iloc[0,:].to_numpy()
resid_var = result_list[2]
resid_df = result_list[3]
X_mat = X_train
X_test

# Define the input data
X_test = df.loc[128:129, cols]
X_test.insert(0, 'Intercept', 1)

# Calculate y_12
y_12 = X_test.iloc[0, :].dot(result_list[0]['Estimate'])
print('The predicted unemployment rate for December 2024 is:')
final_pred = (np.exp(np.log(power_lambda*y_12+1)/power_lambda))
print(final_pred)

# Calculate the standard errors using the given matrix formula
X_test = X_test.iloc[0, :].to_numpy()
resid_var = result_list[2]
resid_df = result_list[3]
X_mat = X_train

pred_std_err = np.sqrt(resid_var) * np.sqrt(X_test @
(np.linalg.inv(X_train.T.dot(X_train))) @ X_test.T)
t_critical = t.ppf(0.975, resid_df)
pred_95_CI_lower = y_12 - t_critical * pred_std_err
pred_95_CI_upper = y_12 + t_critical * pred_std_err

# Revert Box-Cox transformation for the confidence interval
pred_95_CI_lower = (np.exp(np.log(power_lambda*pred_95_CI_lower+1)/power_lambda))
pred_95_CI_upper = (np.exp(np.log(power_lambda*pred_95_CI_upper+1)/power_lambda))

# Print the 95% confidence interval
print(f"95% Confidence Interval for Unemployment Rate is: [{pred_95_CI_lower},
{pred_95_CI_upper}]")

# Calculate R-squared
y_predicted = X_train.dot(result_list[0]['Estimate'])
corr_y_prediction = Regression.PearsonCorrelation(y_train, y_predicted)
R_Square = np.square(corr_y_prediction)
print(f"R-squared: {R_Square}")

# Calculate MSE
mse = np.mean((y_train - y_predicted)**2)
print(f"MSE: {mse}")

# Calculate RMSE (already calculated above as rmse)
print(f"RMSE: {rmse}")

# Calculate MAE
mae = np.mean(np.abs(y_train - y_predicted))
print(f"MAE: {mae}")
```