This is Data Preprocessing and EDA Analysis of the Project, which is developed on Python. Please upload the AU_New_Data_With_EPU.csv file for the analysis before running the code.

For TVP-VAR connectedness and Spillover analysis, please use the New_US_Code_R & New_AU_Code_R notebook instead.

```
# Libraries Loading:
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import json
from datetime import datetime
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

```
# Sample dataframe
data = pd.read_csv("AU_New_Data_With_EPU.csv")

# Convert to dataframe
df = pd.DataFrame(data)
df = df.drop(columns=['WTI'])
df = df.rename(columns={'EPU_News_Based_Policy_Uncert_Index': 'EPU'})
# Convert 'Date' column to datetime (optional but recommended for proper date handling)
df['Date'] = pd.to_datetime(df['Date'])

# Set 'Date' column as the index
df.set_index('Date', inplace=True)

# Display the updated dataframe
print(df)
```

```
              Brent      Gold  ASX50      AUDI         EPU         OPU
Date
2004-01-10  14.177  2.425807   2.38  2.459016   88.764658  300.811176
2004-01-11 -14.386  5.086382   4.23  1.760000   43.547965  242.968242
2004-01-12  -8.493 -2.795545   2.96 -0.628931   43.847222  163.934644
2005-01-01  11.688 -3.677235   1.13  0.474684   34.657753  104.663882
2005-01-02   2.156  3.029687   2.61  1.417323   32.410522  126.600752
...            ...       ...    ...       ...         ...         ...
2023-01-03  -5.168  7.420961  -0.86 -1.791531  190.028779  109.070876
2023-01-04   7.620  1.099176   1.47 -0.829187  194.659277  150.420888
2023-01-05 -11.467 -1.384149  -3.35  0.000000  148.297598   64.430287
2023-01-06  -0.838 -2.201605   1.91  3.177258  113.892480   77.111994
2023-01-07   6.805  2.297874   2.57 -0.648298  149.328944   27.483571

[226 rows x 6 columns]
```

```
# Data Checks And EDA:
df.shape
```

```
(226, 6)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 226 entries, 2004-01-10 to 2023-01-07
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Brent   226 non-null    float64
 1   Gold    226 non-null    float64
 2   ASX50   226 non-null    float64
 3   AUDI    226 non-null    float64
 4   EPU     226 non-null    float64
 5   OPU     226 non-null    float64
dtypes: float64(6)
memory usage: 12.4 KB
```

```
null_percentage = (df.isnull().sum() / len(df)) * 100
null_percentage.sort_values(ascending=True)
```

```
df.describe().T
```

```
# Time series Plots:
plt.figure(figsize=(10, 6))
plt.plot(df.index, df["Brent"], label="Brent Crude Oil")
plt.title("Returns of Brent")
plt.xlabel("Date")
plt.ylabel("Returns")
plt.legend()
plt.grid()
plt.show()


plt.figure(figsize=(10, 6))
plt.plot(df.index, df["Gold"], label="Gold")
plt.xlabel("Index")
plt.ylabel("Returns")
plt.title("Returns of Spot Gold")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(df.index, df["ASX50"], label="ASX50")
plt.xlabel("Index")
plt.ylabel("Returns")
plt.title("Returns of ASX50 Index")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(df.index, df["AUDI"], label="AUD Index")
plt.xlabel("Index")
plt.ylabel("Returns")
plt.title("Returns of AUD Index")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(df.index, df["EPU"], label="EPU Sentiments of Australia")
plt.xlabel("Index")
plt.ylabel("Returns")
plt.title("EPU Sentiments of Australia")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(df.index, df["OPU"], label="Sentiments index of the Oil")
plt.xlabel("Index")
plt.ylabel("Returns")
plt.title("OPU Sentiments of the Crude Oil Price")
```

```
plt.legend()
plt.grid()
plt.show()
```

```
#Combine Time Series Plots of Return Variables:
columns = df.columns[0:4]
plt.figure(figsize=(10, 6))

for var in columns:
    plt.plot(df.index, df[var], label=var)

plt.title("Time Series Plot of Return Variables")
plt.xlabel("Date")
plt.ylabel("")
plt.legend()
plt.grid()
plt.show()
```

```python
#Combine Time Series Plots of Sentiment Index Variables:
columns = df.columns[-2:]
plt.figure(figsize=(10, 6))

for var in columns:
    plt.plot(df.index, df[var], label=var)

plt.title("Time Series Plot of Sentiment Variables")
plt.xlabel("Date")
plt.ylabel("")
plt.legend()
plt.grid()
plt.show()
```

```python
#outlier function
def find_outlier_rows(df, col, level='both'):

  """
    Finds the rows with outliers in a given column of a dataframe.

    This function takes a dataframe and a column as input, and returns the rows
```

```
        with outliers in the given column. Outliers are identified using the
        interquartile range (IQR) formula. The optional level parameter allows the
        caller to specify the level of outliers to return, i.e., lower, upper, or both.

        Args:
            df: The input dataframe.
            col: The name of the column to search for outliers.
            level: The level of outliers to return, i.e., 'lower', 'upper', or 'both'.
                   Defaults to 'both'.

        Returns:
            A dataframe containing the rows with outliers in the given column.
      """
      # compute the interquartile range
      iqr = df[col].quantile(0.75) - df[col].quantile(0.25)

      # compute the upper and lower bounds for identifying outliers
      lower_bound = df[col].quantile(0.25) - 1.5 * iqr
      upper_bound = df[col].quantile(0.75) + 1.5 * iqr

      # filter the rows based on the level of outliers to return
      if level == 'lower':
          return df[df[col] < lower_bound]
      elif level == 'upper':
          return df[df[col] > upper_bound]
      else:
          return df[(df[col] > upper_bound) | (df[col] < lower_bound)]


    def count_outliers(df):
      """
      This function takes in a DataFrame and returns a DataFrame containing the count and
      percentage of outliers in each numeric column of the original DataFrame.

      Input:
          df: a Pandas DataFrame containing numeric columns

      Output:
          a Pandas DataFrame containing two columns:
          'outlier_counts': the number of outliers in each numeric column
          'outlier_percent': the percentage of outliers in each numeric column
      """
      # select numeric columns
      df_numeric = df.select_dtypes(include=['int', 'float'])

      # get column names
      columns = df_numeric.columns

      # find the name of all columns with outliers
      outlier_cols = [col for col in columns if len(find_outlier_rows(df_numeric, col)) != 0]

      # dataframe to store the results
      outliers_df = pd.DataFrame(columns=['outlier_counts', 'outlier_percent'])

      # count the outliers and compute the percentage of outliers for each column
      for col in outlier_cols:
          outlier_count = len(find_outlier_rows(df_numeric, col))
          all_entries = len(df[col])
          outlier_percent = round(outlier_count * 100 / all_entries, 2)

          # store the results in the dataframe
          outliers_df.loc[col] = [outlier_count, outlier_percent]

      # return the resulting dataframe
      return outliers_df


    count_outliers(df).sort_values('outlier_counts', ascending=False)
```

```
# Plot histogram for Brent:
plt.subplot(2, 3, 1)
sns.histplot(df['Brent'], bins=20, kde=True, color='blue')
plt.title('Brent Returns')
plt.xlabel('Returns')
plt.ylabel('Frequency')

# Plot histogram for Gold
plt.subplot(2, 3, 2)
sns.histplot(df['Gold'], bins=20, kde=True, color='green')
plt.title('XAU Returns')
plt.xlabel('Returns')
plt.ylabel('')

# Plot histogram for ASX50:
plt.subplot(2, 3, 3)
sns.histplot(df['ASX50'], bins=20, kde=True, color='green')
plt.title('ASX50 Returns')
plt.xlabel('Returns')
plt.ylabel('')

# Plot histogram for AUDI:
plt.subplot(2, 3, 4)
sns.histplot(df['AUDI'], bins=20, kde=True, color='orange')
plt.title('AUDI Returns')
plt.xlabel('Returns')
plt.ylabel('Frequency')

# Plot histogram for EPU:
plt.subplot(2, 3, 5)
sns.histplot(df['EPU'], bins=20, kde=True, color='grey')
plt.title('AU EPU Sentiment')
plt.xlabel('Sentiment Index')
plt.ylabel('')

# Plot histogram for OPU:
plt.subplot(2, 3, 6)
sns.histplot(df['OPU'], bins=20, kde=True, color='indigo')
plt.title('Oil Sentiment')
plt.xlabel('Sentiment Index')
plt.ylabel('')

# Adjust layout
plt.tight_layout()
plt.show()
```

```
# Calculate and visualize correlations
correlation_matrix = df[["Brent", "Gold", "ASX50", "AUDI", "EPU", "OPU"]].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Matrix")
plt.show()
print("Correlation Matrix:\n", correlation_matrix.round(2))
```

```
# PACF plots
plot_pacf(df['Brent'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of Brent')
plt.show()


plot_pacf(df['Gold'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of Gold')
plt.show()

plot_pacf(df['ASX50'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of ASX50')
plt.show()

# PACF plots
plot_pacf(df['AUDI'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of AUDI')
plt.show()


plot_pacf(df['EPU'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of US EPU')
plt.show()

plot_pacf(df['OPU'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) of Oil Sentiment')
plt.show()
```

```
from statsmodels.tsa.stattools import adfuller

for col in df.columns:
  # ADF Test
  result = adfuller(df[col])

  # Print test statistics
  print(f"Test Statistic for {col}: {result[0]:.4f}")
  print(f"p-value for {col}: {result[1]:.4f}")

  # Interpretation (adjust significance level as needed)
  if result[1] < 0.05:
    print(f"{col} is likely stationary (rejects unit root).")
  else:
    print(f"{col} might be non-stationary (fails to reject unit root).")
  print("--------------------------------------")
```

```
Test Statistic for Brent: -10.4260
p-value for Brent: 0.0000
Brent is likely stationary (rejects unit root).
--------------------------------------
Test Statistic for Gold: -16.6353
p-value for Gold: 0.0000
Gold is likely stationary (rejects unit root).
--------------------------------------
Test Statistic for ASX50: -14.3277
p-value for ASX50: 0.0000
ASX50 is likely stationary (rejects unit root).
--------------------------------------
Test Statistic for AUDI: -14.2829
p-value for AUDI: 0.0000
AUDI is likely stationary (rejects unit root).
--------------------------------------
Test Statistic for EPU: -3.9477
p-value for EPU: 0.0017
EPU is likely stationary (rejects unit root).
--------------------------------------
Test Statistic for OPU: -5.7686
p-value for OPU: 0.0000
OPU is likely stationary (rejects unit root).
--------------------------------------
```