https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/542da3e07b3cbe02b366691c69cdabcf/2dc00e19-04be-4ef2-80de-2e6c520cf1f0/2c5673b4.png

```
#step 1 : import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# step 2 : load dataset
df = pd.read_csv('/content/random_stock_market_dataset.csv')

# step 3 : data overview
print(df.info())
print(df.head())

# step 4 : summary statistics
print(df.describe())

# step 5 : check for missing values
print(df.isnull().sum())

# step 6 : outlier detection (IQR method)
numeric = df.drop('Date', axis=1)
q1 = numeric.quantile(0.25)
q3 = numeric.quantile(0.75)
iqr = q3 - q1
outliers = ((numeric < (q1 - 1.5 * iqr)) | (numeric > (q3 + 1.5 * iqr))).sum()
print('outliers per column:\n', outliers)

# step 6 :  Correlation Heatmap
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(numeric.corr(), annot=True)
plt.title('Correlation Heatmap')
plt.show()

# step 7 :  Basic Visualizations
numeric.hist(figsize=(12, 8))
plt.show()

# step 8 : Save any changes (Optional)
df.to_csv('cleaned_stock_market_data.csv', index=False)
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import Binarizer, MinMaxScaler, LabelEncoder

# Load the dataset
file_path = '/content/random_stock_market_dataset.csv'
try:
    df = pd.read_csv(file_path)
except FileNotFoundError:
    print(f"Error: File not found at {file_path}")
    # Exit or raise error if file not found
    exit()
```

```
# Create a copy to store modifications
df_processed = df.copy()

print("--- Original Data Head ---")
print(df_processed.head())
print("\n")

# --- 1. Statistical Measures (Mean, Median, Mode) ---
print("--- 1. Statistical Measures ---")
# Define which columns are numerical for these stats
numerical_cols = ['Open', 'High', 'Low', 'Close', 'Volume']

# Mean
print("Mean:")
print(df_processed[numerical_cols].mean())
print("\n")

# Median
print("Median:")
print(df_processed[numerical_cols].median())
print("\n")

# Mode
print("Mode:")
# Mode can return multiple values (e.g., if two values appear with
# the same highest frequency). We print just the first row of modes.
print(df_processed[numerical_cols].mode().iloc[0])
print("\n")

# --- 2. Imputation ---
print("--- 2. Imputation (Handling Missing Values) ---")
# First, check if there are any missing values in the whole DataFrame
missing_values_count = df_processed.isnull().sum().sum()
if missing_values_count == 0:
    print("No missing values found in the dataset. Imputation is not necessary.")
else:
    print(f"Found {missing_values_count} total missing values. Applying imputation...")
    # This example imputes with the mean for numerical columns.
    # You could choose 'median' or 'most_frequent' as other strategies.
    num_imputer = SimpleImputer(strategy='mean')

    for col in numerical_cols:
        if df_processed[col].isnull().any():
            print(f"Imputing column: {col}")
            # SimpleImputer expects 2D data, so we use [[col]]
            df_processed[col] = num_imputer.fit_transform(df_processed[[col]])

    print("Imputation applied (if any values were missing).")
print("\n")

# --- 3. Binarization ---
print("--- 3. Binarization ---")
# Binarize the 'Volume' column based on its mean
volume_mean = df_processed['Volume'].mean()
print(f"Binarizing 'Volume' column. Threshold (mean) = {volume_mean:.2f}")

# Initialize the Binarizer with the threshold
binarizer = Binarizer(threshold=volume_mean)

# Note: Binarizer (and most sklearn transformers) expects a 2D array.
# .values returns a NumPy array. .reshape(-1, 1) converts it to 2D.
df_processed['Volume_Binarized'] = binarizer.fit_transform(df_processed[['Volume']].values)

print("Binarization complete. New column 'Volume_Binarized' added.")
print(df_processed[['Volume', 'Volume_Binarized']].head())
print("\n")

# --- 4. Normalization (Min-Max Scaling) ---
print("--- 4. Normalization (Min-Max Scaling) ---")
# Normalize the 'Close' column to the [0, 1] range
scaler = MinMaxScaler()
df_processed['Close_Normalized'] = scaler.fit_transform(df_processed[['Close']].values)
```

```
print("Normalization complete. New column 'Close_Normalized' added.")
print(df_processed[['Close', 'Close_Normalized']].head())
print("\n")

# --- 5. & 6. Label Encoding and One-Hot Encoding ---
print("--- 5. & 6. Label and One-Hot Encoding ---")
# These techniques are for categorical data.
# We will create a categorical column from 'Date' to demonstrate.

# Ensure 'Date' is in datetime format
df_processed['Date'] = pd.to_datetime(df_processed['Date'])

# Create a new categorical column: 'Day_of_Week'
df_processed['Day_of_Week'] = df_processed['Date'].dt.day_name()
print("Created new categorical column 'Day_of_Week':")
print(df_processed[['Date', 'Day_of_Week']].head())
print("\n")

# 5. Label Encoding
print("Applying Label Encoding to 'Day_of_Week'...")
le = LabelEncoder()
df_processed['Day_of_Week_LabelEncoded'] = le.fit_transform(df_processed['Day_of_Week'])
print("Label Encoding complete. New column 'Day_of_Week_LabelEncoded' added.")

# Show the mapping created by the LabelEncoder
print("Label Encoder Mapping:")
for i, class_name in enumerate(le.classes_):
    print(f"{class_name} -> {i}")
print("\n")


# 6. One-Hot Encoding
print("Applying One-Hot Encoding to 'Day_of_Week'...")
# pd.get_dummies is a simple way to get one-hot encoding
one_hot_df = pd.get_dummies(df_processed['Day_of_Week'], prefix='Day')

# Join the new one-hot columns back to the main DataFrame
df_processed = pd.concat([df_processed, one_hot_df], axis=1)
print("One-Hot Encoding complete. New 'Day_*' columns added.")
print("\n")


# --- Final Result ---
print("--- Final Processed Data (Head) ---")
# Print the head of the DataFrame to show all new columns
print(df_processed.head())
print("\n")

# Save the processed DataFrame to a new CSV file
output_file = 'stock_data_processed.csv'
df_processed.to_csv(output_file, index=False)
print(f"All processing complete. The new DataFrame has been saved to '{output_file}'")
```

```
--- Final Processed Data (Head) ---
        Date    Open    High     Low   Close  Volume  Volume_Binarized  \
0 2024-01-01  296.45  307.31  293.96  303.72   93133                 0
1 2024-01-02  190.11  193.10  187.21  191.40   64993                 0
2 2024-01-03  197.41  208.64  193.37  205.89   70326                 0
3 2024-01-04  253.13  262.67  248.67  258.95   17358                 0
4 2024-01-05  241.35  253.09  238.99  252.20   20847                 0

   Close_Normalized Day_of_Week  Day_of_Week_LabelEncoded  Day_Friday  \
0          0.481208      Monday                         1       False
1          0.184966     Tuesday                         5       False
2          0.223183   Wednesday                         6       False
3          0.363128    Thursday                         4       False
4          0.345325      Friday                         0        True

   Day_Monday  Day_Saturday  Day_Sunday  Day_Thursday  Day_Tuesday  \
0        True         False       False         False        False
1       False         False       False         False         True
2       False         False       False         False        False
3       False         False       False          True        False
4       False         False       False         False        False

   Day_Wednesday
0          False
1          False
2           True
3          False
4          False


All processing complete. The new DataFrame has been saved to 'stock_data_processed.csv'
```

```
from IPython.display import Image
Image(filename='/content/stock data pairplot.png')
```