```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'stock-price-prediction:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F4571465%2F7806181%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGOOG4-

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading stock-price-prediction, 18005 bytes compressed
[==================================================] 18005 bytes downloaded
Downloaded and uncompressed: stock-price-prediction
Data source import complete.
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/stock-price-prediction/stock_data.csv
```

```
import pandas as pd

df = pd.read_csv('/kaggle/input/stock-price-prediction/stock_data.csv')
df.head()
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283

Next steps: Generate code with df View recommended plots

```
df.rename(columns = {'Unnamed: 0': 'date'}, inplace = True)
df.head()
```

	date	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283

Next steps: Generate code with df View recommended plots

```
df.describe()
```

	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
count	365.000000	365.000000	365.000000	365.000000	365.000000
mean	107.772577	81.105216	94.519502	117.407560	106.866865
std	7.398296	11.435212	6.519213	6.778527	3.760968
min	91.474442	62.414219	81.111434	99.909756	99.833309
25%	101.603117	69.328263	89.788068	112.209912	103.927072
50%	107.421299	84.283525	94.495546	117.788079	106.411328
75%	113.741728	91.548859	99.919465	123.132365	109.178007
max	121.901773	100.160928	107.588373	129.911386	116.243803

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     365 non-null    object 
 1   Stock_1  365 non-null    float64
 2   Stock_2  365 non-null    float64
 3   Stock_3  365 non-null    float64
 4   Stock_4  365 non-null    float64
 5   Stock_5  365 non-null    float64
dtypes: float64(5), object(1)
memory usage: 17.2+ KB
```

```
df.isnull().sum()
```

```
date       0
Stock_1    0
Stock_2    0
Stock_3    0
Stock_4    0
Stock_5    0
dtype: int64
```

EDA

1. Time Series Analysis

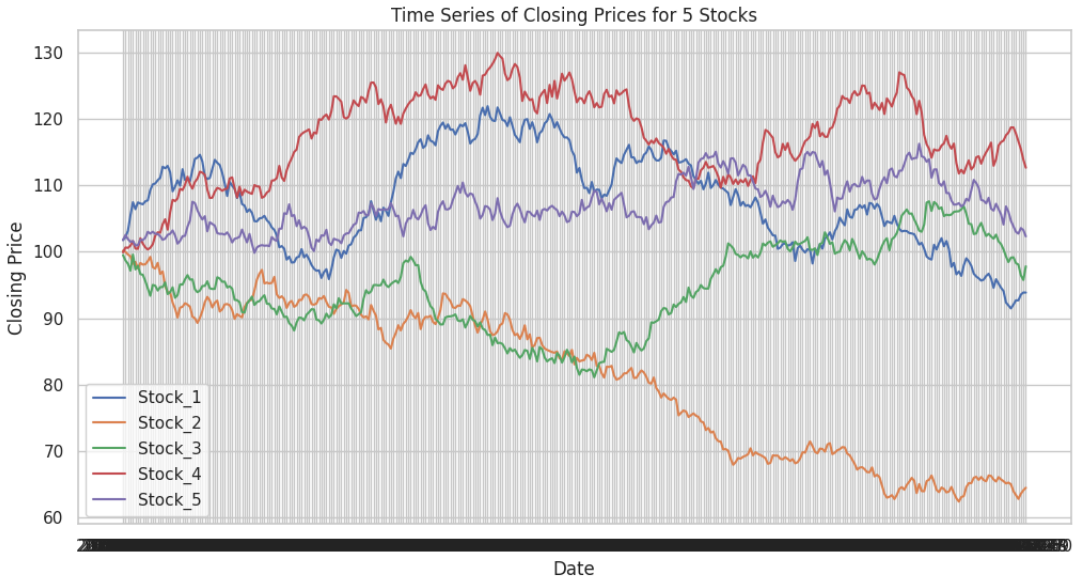
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style = 'whitegrid')

df.set_index('date', inplace=True)


plt.figure(figsize = (12,6))
for column in df.columns:
    plt.plot(df.index, df[column], label = column)

plt.title('Time Series of Closing Prices for 5 Stocks')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```



2. Outlier Detection in Prices

```
df
```

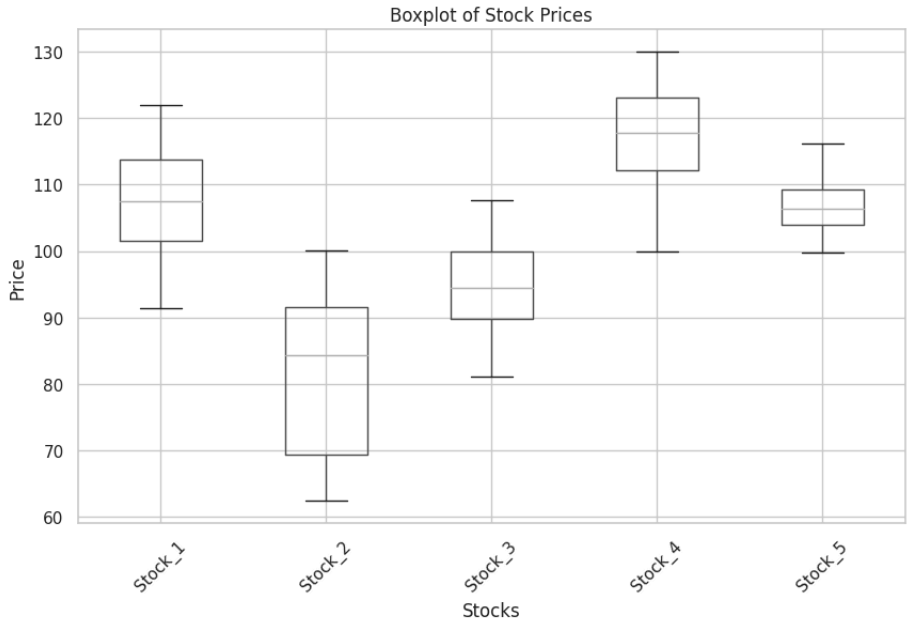
date	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283
...	...	...	...	...	...
2020-12-26	92.684784	63.408103	98.288992	117.788079	102.995720
2020-12-27	92.688279	62.816639	98.061845	116.605106	102.718260
2020-12-28	93.551993	63.597651	96.454800	115.441164	103.566068
2020-12-29	93.870037	64.114492	95.747485	113.856107	103.257107
2020-12-30	93.855317	64.491011	97.805648	112.640418	102.304226

365 rows × 5 columns

Next steps: Generate code with df View recommended plots

```
if 'Date' in df.columns:
    df.drop('Date', axis=1, inplace=True)

plt.figure(figsize=(10, 6))
df.boxplot()
plt.title('Boxplot of Stock Prices')
plt.ylabel('Price')
plt.xlabel('Stocks')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Model Building and Evaluation

Linear Regression

Linear Regression is a linear approach to modeling the relationship between a dependent variable (target) and one or more independent variables (features). It assumes that there is a linear relationship between the features and the target variable, and it tries to fit a straight line to the data that minimizes the sum of the squared residuals.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

df = pd.get_dummies(df)

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

mse = mean_squared_error(y_test, predictions)
rmse = mse ** 0.5
r2 = r2_score(y_test, predictions)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared Score:", r2)

plt.figure(figsize=(10, 6))
plt.plot(y_test.index, y_test.values, label='Actual')
plt.plot(y_test.index, predictions, color='red', label='Predicted')
plt.title('Linear Regression: Actual vs Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 28.92541401358996
Root Mean Squared Error: 5.378235213672786
R-squared Score: 0.3976048110834216
```