STOCK PRICE PREDICTION

Abstract:

Title: Predictive Modeling of Stock Prices Using Machine Learning

Module 1: Data Collection and Preprocessing

Module 1 of our research project focuses on the crucial task of collecting and preprocessing historical stock price data. This module encompasses the following key steps:

1.1 Data Gathering:

We employ various data sources, including financial databases and APIs, to gather historical stock price data for a selected set of companies. This dataset serves as the foundation for our predictive models.

1.2 Data Cleaning:

To ensure data quality and consistency, we perform data cleaning procedures such as handling missing values, outlier detection, and data alignment across multiple sources.

1.3 Feature Engineering:

We engineer relevant financial features, including moving averages, technical indicators, and sentiment scores from news and social media, to provide valuable input to our prediction models.

Module 2: Model Selection and Training

In Module 2, we focus on the selection and training of machine learning models for stock price prediction. The following steps are involved in this module:

2.1 Model Selection: We explore a range of predictive models, including linear regression, decision trees, support vector machines, and deep neural networks, to determine the most suitable approach for our dataset.

2.2 Data Splitting: We split the historical data into training, validation, and test sets to facilitate model training and evaluation.

2.3 Model Training and Tuning: We train the selected models on the training data and fine-tune their hyperparameters to optimize their predictive performance.

Module 3: Model Evaluation and Validation

Module 3 is dedicated to evaluating the predictive accuracy and robustness of our stock price prediction models. This module includes the following steps:

3.1 Model Evaluation Metrics: We employ standard evaluation metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2) to assess the models' performance.

3.2 Cross-Validation: We use cross-validation techniques to ensure the generalizability of our models and mitigate overfitting.

3.3 Model Interpretability: We analyze the feature importance and model interpretability to gain insights into the factors driving stock price predictions.

Module 4: Deployment and Real-time Prediction

In the final module, we discuss the deployment of our predictive models for real-time stock price prediction. This includes setting up a pipeline to ingest and preprocess incoming data and making predictions available via APIs or web interfaces.

Our research project aims to provide a comprehensive framework for stock price prediction, leveraging machine learning techniques and financial data. By following this systematic approach, we contribute to the ongoing efforts to enhance investment decision-making and risk management in the financial industry.

Introduction

This is an automatically-generated kernel with starter code demonstrating how to read in the data and begin exploring. If you're inspired to dig deeper, click the blue "Fork Notebook" button at the top of this kernel to begin editing.

Exploratory Analysis

To begin this exploratory analysis, first import libraries and define functions for plotting the data using `matplotlib`. Depending on the data, not all plots will be made.

In [1]:
```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O
```

There is 1 csv file in the current version of the dataset:

In [2]:
```
for dirname, _, filenames in os.walk(' input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
input/MSFT.csv
```

.

unfold_less

In [3]:
```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col]
< 50]] # For displaying purposes, pick columns that have between 1 a
nd 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRo
w), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
```

```
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

unfold_less

In [4]:

```
# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep col
umns where there are more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or
constant columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, f
acecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
```

unfold_less

In [5]:

```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerica
l columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep col
umns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for mat
rix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSiz
e, plotSize], diagonal='kde')
```

```
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0
.2), xycoords='axes fraction', ha='center', va='center', size=textSi
ze)
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

Now you're ready to read in the data and use the plotting functions to visualize the data.

Let's check 1st file: /input/MSFT.csv

In [6]:

```
nRowsRead = 1000 # specify 'None' if want to read whole file
# MSFT.csv may have more rows in reality, but we are only loading/pr
eviewing the first 1000 rows
df1 = pd.read_csv('/kaggle/input/MSFT.csv', delimiter=',', nrows = n
RowsRead)
df1.dataframeName = 'MSFT.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 1000 rows and 7 columns
```

Let's take a quick look at what the data looks like:

In [7]:

```
df1.head(5)
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400

	Date	Open	High	Low	Close	Adj Close	Volume
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

Distribution graphs (histogram/bar graph) of sampled columns:

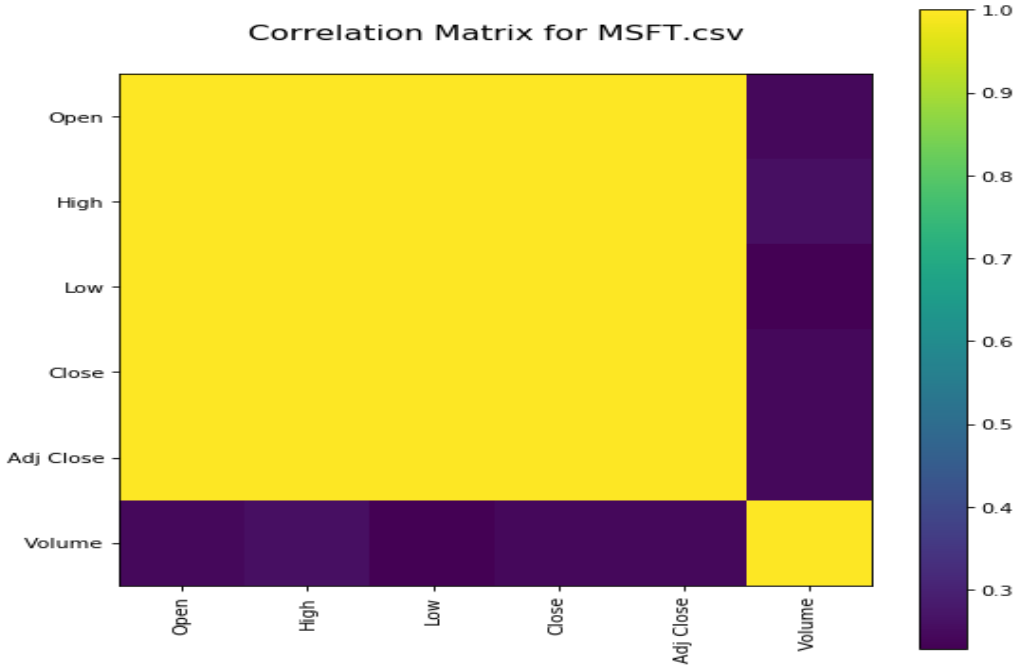
In [8]:
```
plotPerColumnDistribution(df1, 10, 5)
```

```
<Figure size 2400x512 with 0 Axes>
```

Correlation matrix:

In [9]:
```
plotCorrelationMatrix(df1, 8)
```

Scatter and density plots:



In [10]:
```
plotScatterMatrix(df1, 18, 10)
```

Scatter and Density Plot