APPLIED DATA SCIENCE - PHASE 3

STOCK PRICE PREDICTION

Team Leader: Supriya . A(211521104163)

Team Members:

Priyadharshini . Y(211521104119)

Srinidhi . E(211521104157)

Getsy Jacinth .S(211521104043)

Saranya . C(211521104143)

TEAM: TG-06

CONTENTS:

1. Importing Required Libraries

Dataset Link:

https://www.kaggle.com/datasets/prasoonkotta

rathil/microsoft-lifetime-stocks-dataset

2. IMPORT DATA SET:

READ DATASET:

Python with Pandas:

You can use the Pandas library to import datasets in various formats like CSV, Excel, SQL, and more

```
import pandas as pd

data = pd.read_csv('dataset.csv')

print(data)
```

OUTPUT:

```
           Date          Open       High           Low          Close
Adj Close  \
1.     1986-03-13   0.088542   0.101563   0.088542   0.097222
0.062549
2.     1986-03-14   0.097222   0.102431   0.097222   0.100694
0.064783
3.     1986-03-17   0.100694   0.103299   0.100694   0.102431
0.065899
 4.    1986-03-18   0.102431   0.103299   0.098958   0.099826
0.064224
  5.   1986-03-19   0.099826   0.100694   0.097222   0.098090
0.063107
...      ...      ...      ...      ...      ...      ...
8520  2019-12-31  156.770004  157.770004  156.449997  157.699997
157.699997
8521  2020-01-02  158.779999  160.729996  158.330002  160.619995
160.619995
8522  2020-01-03  158.320007  159.949997  158.059998  158.619995
158.619995
8523  2020-01-06  157.080002  159.100006  156.509995  159.029999
159.029999
8524  2020-01-07  159.320007  159.669998  157.330002  157.580002
157.580002
```

```
        Volume
0     1031788800
1      308160000
2      133171200
3       67766400
4       47894400
...          ...
8520    18369400
8521    22622100
8522    21116200
8523    20813700
8524    18017762

[8525 rows x 7 columns]
```

CREATE MATRIX:

-matrix = data.values: After reading the data into the Pandas DataFrame, this line of code extracts the values from the DataFrame and stores them in a NumPy array called matrix. The values attribute of a DataFrame returns a NumPy array containing the data. NumPy is a library for numerical computations in Python and is often used for working with arrays and matrices.

So, by the end of this code, you have successfully imported your dataset, which is originally in CSV format, into a Pandas DataFrame (data) and then converted it into a NumPy array (matrix) for further data analysis or manipulation using the NumPy library.

Code:

```
import numpy as np

import pandas as pd


# Read the dataset into a Pandas DataFrame

data = pd.read_csv('your_dataset.csv')  # Replace 'your_dataset.csv' with your dataset file path
```

Extract the values from the DataFrame and create a NumPy array (matrix)

matrix = data.values

print(matrix)

Now 'matrix' is a NumPy array that represents your dataset

OUTPUT:

```
[['1986-03-13' 0.088542 0.101563 ... 0.097222 0.062549
1031788800]
 ['1986-03-14' 0.097222 0.102431 ... 0.100694 0.064783 308160000]
 ['1986-03-17' 0.100694 0.103299 ... 0.102431 0.065899 133171200]
 ...
 ['2020-01-03' 158.320007 159.949997 ... 158.619995 158.619995
21116200]
 ['2020-01-06' 157.080002 159.100006 ... 159.029999 159.029999
20813700]
 ['2020-01-07' 159.320007 159.669998 ... 157.580002 157.580002
18017762]]
```

DATA CLEANSING:

-sklearn.preprocessing.Imputer class (or SimpleImputer in scikit-learn versions 0.22 and later) to clean missing data in a dataset by replacing missing values with a specified strategy, such as mean, median, most frequent, or a constant value.

- the missing values (NaN) in the dataset are replaced with the mean of the non-missing values in their respective columns. You can change the strategy parameter to 'median', 'most_frequent', or 'constant' depending on your specific data cleaning requirements. If you choose 'constant', you can set the constant value using the fill_value parameter.

Code:

```
import numpy as np
import pandas as pd
df = pd.read_csv('MSFT.csv')
print(df.head())
df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values('Date')
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Close'] = scaler.fit_transform(df['Close'].values.reshape(-1, 1))
def create_sequences(data, sequence_length):
    sequences = []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i+sequence_length])
    return np.array(sequences)

sequence_length = 10  # Choose an appropriate value
X = create_sequences(df['Close'], sequence_length)
train_size = int(0.7 * len(X))
valid_size = int(0.2 * len(X))

X_train = X[:train_size]
X_valid = X[train_size:train_size+valid_size]
```

```
X_test = X[train_size+valid_size:]
y_train = df['Close'].iloc[sequence_length:train_size + sequence_length].values
y_valid = df['Close'].iloc[train_size + sequence_length:train_size + valid_size + sequence_length].values
y_test = df['Close'].iloc[train_size + valid_size + sequence_length:].values
X_train = X_train.reshape(-1, sequence_length, 1)
X_valid = X_valid.reshape(-1, sequence_length, 1)
X_test = X_test.reshape(-1, sequence_length, 1)
```

<u>OUTPUT:</u>

	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
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

ONE- HOT ENCODING:

One-hot encoding is a data preprocessing technique used in machine learning and data analysis to convert categorical data into a binary format, making it easier for algorithms to work with such data. It is particularly useful for dealing with categorical variables that have no intrinsic ordinal relationship or numeric meaning.

In one-hot encoding:

1.Each unique category or label in a categorical feature is transformed into a binary vector.

2.For each unique category, a new binary feature (or column) is created.

3.If a data point belongs to a specific category, the corresponding binary feature is set to 1; otherwise, it's set to 0.

4.The result is a binary matrix where each column represents a category, and each row corresponds to an individual data point.

It can increase the dimensionality of the data, potentially leading to a sparse matrix, which may require additional techniques for dimensionality reduction or handling.

One-hot encoding is widely supported in libraries like Pandas in Python, making it easy to apply in data preprocessing pipelines.

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Sample dataset with categorical data
data = pd.read_csv('MSFT.csv')

# Create a OneHotEncoder instance
encoder = OneHotEncoder()

# Fit the encoder to the data and transform it
encoded_data = encoder.fit_transform(data)

# The result is a sparse matrix, so you can convert it to a dense array
if needed
dense_encoded_data = encoded_data.toarray()

print("Original Data:")
print(data)
print("\nOne-Hot Encoded Data (Sparse Matrix):")
print(encoded_data)
print("\nOne-Hot Encoded Data (Dense Array):")
print(dense_encoded_data)
```

OUTPUT:

Original Data:

```
        Date      Open      High  ...     Close    Adj Close
Volume
0    1986-03-13   0.088542   0.101563  ...   0.097222   0.062549
1031788800
1    1986-03-14   0.097222   0.102431  ...   0.100694   0.064783
308160000
2    1986-03-17   0.100694   0.103299  ...   0.102431   0.065899
133171200
3    1986-03-18   0.102431   0.103299  ...   0.099826   0.064224
67766400
4    1986-03-19   0.099826   0.100694  ...   0.098090   0.063107
47894400
...        ...        ...        ...  ...        ...        ...        ...
8520  2019-12-31  156.770004  157.770004  ...  157.699997
157.699997   18369400
8521  2020-01-02  158.779999  160.729996  ...  160.619995
160.619995   22622100
8522  2020-01-03  158.320007  159.949997  ...  158.619995
158.619995   21116200
8523  2020-01-06  157.080002  159.100006  ...  159.029999
159.029999   20813700
8524  2020-01-07  159.320007  159.669998  ...  157.580002
157.580002   18017762

[8525 rows x 7 columns]
```

One-Hot Encoded Data (Sparse Matrix):

```
(0, 0)	1.0
(0, 8525)	1.0
(0, 13202)	1.0
(0, 17819)	1.0
(0, 22463)	1.0
(0, 27286)	1.0
(0, 42042)	1.0
(1, 1)	1.0
(1, 8533)	1.0
(1, 13203)	1.0
(1, 17829)	1.0
(1, 22468)	1.0
(1, 27291)	1.0
(1, 42031)	1.0
(2, 2)	1.0
(2, 8538)	1.0
(2, 13204)	1.0
(2, 17833)	1.0
(2, 22470)	1.0
(2, 27293)	1.0
(2, 41716)	1.0
(3, 3)	1.0
(3, 8540)	1.0
```

(3, 13204)	1.0
(3, 17831)	1.0
:	:
(8521, 22454)	1.0
(8521, 27277)	1.0
(8521, 33705)	1.0
(8521, 34354)	1.0
(8522, 8522)	1.0
(8522, 13188)	1.0
(8522, 17817)	1.0
(8522, 22452)	1.0
(8522, 27273)	1.0
(8522, 33701)	1.0
(8522, 34227)	1.0
(8523, 8523)	1.0
(8523, 13183)	1.0
(8523, 17814)	1.0
(8523, 22446)	1.0
(8523, 27276)	1.0
(8523, 33704)	1.0
(8523, 34208)	1.0
(8524, 8524)	1.0
(8524, 13191)	1.0
(8524, 17816)	1.0
(8524, 22450)	1.0

```
  (8524, 27270)	1.0
  (8524, 33698)	1.0
  (8524, 33997)	1.0

One-Hot Encoded Data (Dense Array):
[[1. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

Process finished with exit code 0
```

INFERENCES FROM CODE:

In this code:

We import the pandas library, which is commonly used for data manipulation.

We load the Stock Price Prediction dataset into the program for data manipulation.

We create a DataFrame df from the data dictionary.

Finally, we print the resulting DataFrame, df_encoded, which now contains the one-hot encoded features.

This code will create a new DataFrame with one-hot encoded columns for the 'Stock Price Prediction' feature. We can use this DataFrame for further analysis and stock price prediction tasks, including training machine learning models.

5.SPLITTING THE DATA INTO TEST SET AND TRAINING SET:

Divide the data into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate its performance

a. Training Data: This portion of the data is used to train your stock price prediction model. It typically consists of historical stock price and volume data. The training data should cover a significant portion of your historical data, for example, 70% to 80% of the total data.

b. Testing Data: This is a smaller portion of your data, often around 20% to 30%, used to evaluate the model's performance. It typically consists of more recent data that the model has not seen during training.

CODE:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import MinMaxScaler
# Load your stock price data into a Pandas DataFrame
df = pd.read_csv('MSFT.csv')  # Replace with your data file
# Select the feature(s) you want to use for prediction
X = df[['Close']].values
# Normalize the data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
# Define the sequence length
seq_length = 10
# Create sequences from the data
sequences = []
labels = []
for i in range(len(X) - seq_length):
  sequences.append(X[i:i+seq_length])
labels.append(X[i+seq_length])
X = np.array(sequences)
y = np.array(labels)
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Reshape the data for LSTM input
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
# Verify the shapes of the split data
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

OUTPUT:

```
X_train shape: (6812, 10, 1)
y_train shape: (6812, 1)
X_test shape: (1703, 10, 1)
y_test shape: (1703, 1)
(6812, 10, 1) (6812, 1) (1703, 10, 1) (1703, 1)
```

In this code:

- We load the stock price data and select the 'Close' feature as an example.
- Normalize the data using Min-Max scaling.
- Define the sequence length (e.g., 10 in this case) and create sequences from the data.
- Use train_test_split to split the data into training and test sets, with a specified test size.
- Reshape the data to meet the input requirements of an LSTM model (samples, time steps, features).
- Verify the shapes of the training and test sets.
- X is your input data (in this example, it's the sequences of stock prices).
- y is your target data (the corresponding labels or next stock price).

- test_size is set to 0.2, indicating that you want to allocate 20% of the data to the test set, and 80% to the training set.
- random_state is used for seed value to ensure reproducibility.
- This code will split your data into training and test sets, and it will print the shapes of the resulting arrays for verification.

6.<u>FEATURE SCALING:</u>

- Feature scaling is an essential data preprocessing step in many machine learning algorithms. It ensures that all your features (variables) have similar scales or ranges, which can be particularly important for algorithms that are sensitive to the magnitude of the input variables, like distance-based algorithms (e.g., k-means clustering) and gradient descent-based algorithms (e.g., support vector machines and neural networks). Two common methods for feature scaling are Min-Max scaling (also known as normalization) and Standardization (Z-score scaling). You can use scikit-learn to perform feature scaling

Code:

```
#Set Target Variable
output_var = PD.DataFrame(df[‘Adj Close’])
#Selecting the Features
features = [‘Open’, ‘High’, ‘Low’, ‘Volume’]

#Scaling
scaler = MinMaxScaler()
feature_transform = scaler.fit_transform(df[features])
```

```
feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)
feature_transform.head()
```

output :

Date	Open	High	Low	Volume
1990-01-02	0.000129	0.000105	0.000129	0.064837
1990-01-03	0.000265	0.000195	0.000273	0.144673
1990-01-04	0.000249	0.000300	0.000288	0.160404
1990-01-05	0.000386	0.000300	0.000334	0.086566
1990-01-08	0.000265	0.000240	0.000273	0.072656