



Project Title: Stock Price Prediction

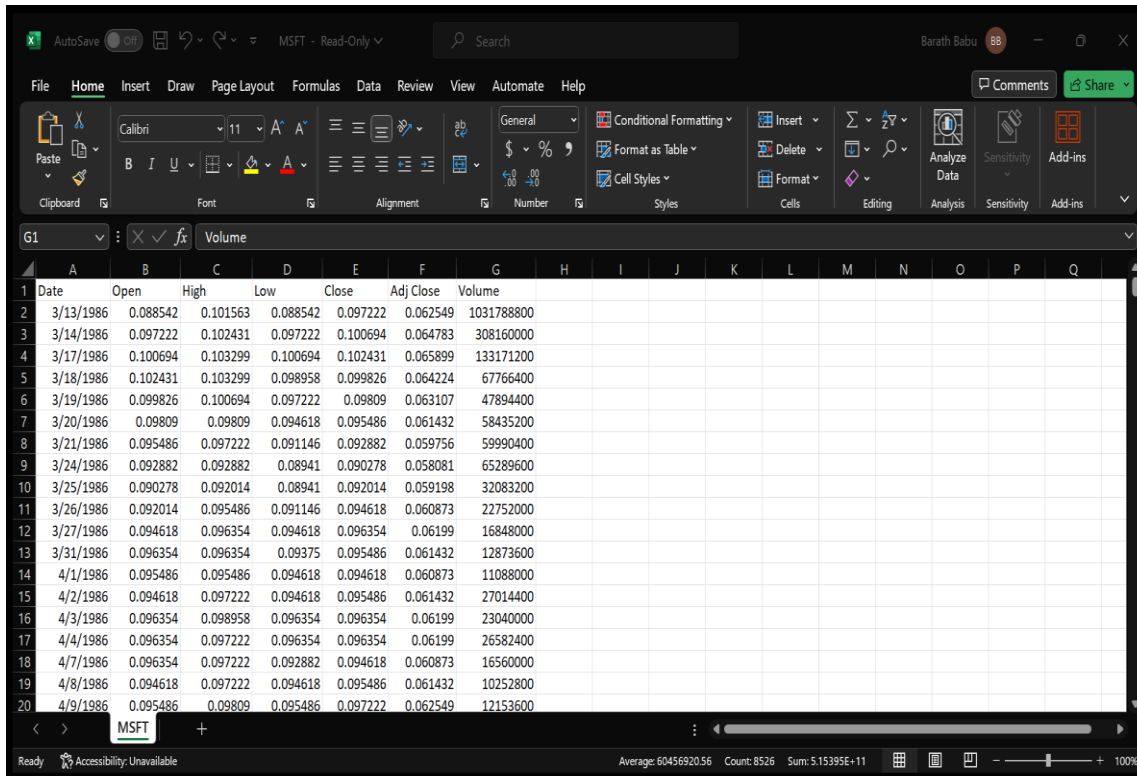
Project Definition:

This PowerPoint contains an In-depth Analysis of Stock Price Prediction to build a predictive model that forecasts stock prices based on historical market data. This project involves data collection, data preprocessing, feature engineering, model selection, training, and evaluation.

Data Collection:

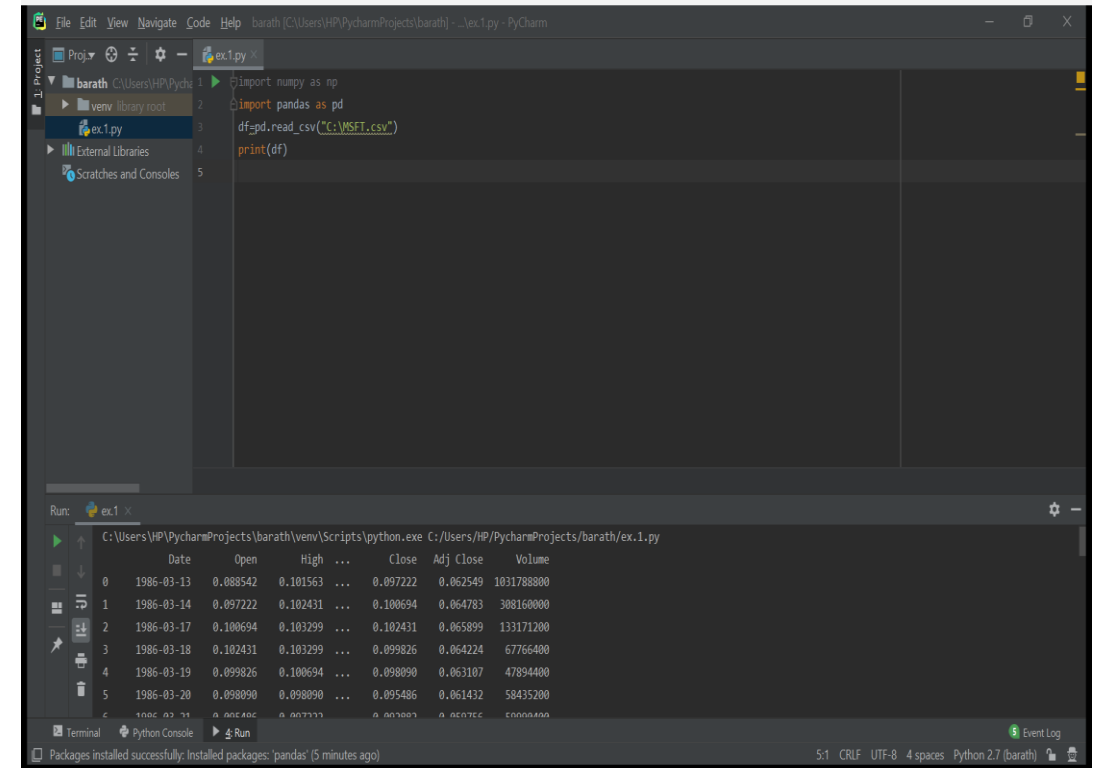
By making use of the link given below we can download the dataset for our project.

<https://www.kaggle.com/prasoonkottarathil/microsoft-lifetime-stocks-dataset>.



The screenshot shows a Microsoft Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Date	Open	High	Low	Close	Adj Close	Volume										
2	3/13/1986	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800										
3	3/14/1986	0.097222	0.102431	0.097222	0.100694	0.064783	308160000										
4	3/17/1986	0.100694	0.103299	0.100694	0.102431	0.065899	133171200										
5	3/18/1986	0.102431	0.103299	0.098958	0.099826	0.064224	67766400										
6	3/19/1986	0.099826	0.100694	0.097222	0.09809	0.063107	47894400										
7	3/20/1986	0.09809	0.09809	0.094618	0.095486	0.061432	58435200										
8	3/21/1986	0.095486	0.097222	0.091146	0.092882	0.059756	59990400										
9	3/24/1986	0.092882	0.092882	0.08941	0.090278	0.058081	65289600										
10	3/25/1986	0.090278	0.092014	0.08941	0.092014	0.059198	32083200										
11	3/26/1986	0.092014	0.095486	0.091146	0.094618	0.060873	22752000										
12	3/27/1986	0.094618	0.096354	0.094618	0.096354	0.06199	16848000										
13	3/31/1986	0.096354	0.096354	0.09375	0.095486	0.061432	12873600										
14	4/1/1986	0.095486	0.095486	0.094618	0.094618	0.060873	11088000										
15	4/2/1986	0.094618	0.097222	0.094618	0.095486	0.061432	27014400										
16	4/3/1986	0.096354	0.098958	0.096354	0.096354	0.06199	23040000										
17	4/4/1986	0.096354	0.097222	0.096354	0.096354	0.06199	26582400										
18	4/7/1986	0.096354	0.097222	0.092882	0.094618	0.060873	16560000										
19	4/8/1986	0.094618	0.097222	0.094618	0.095486	0.061432	10252800										
20	4/9/1986	0.095486	0.09809	0.095486	0.097222	0.062549	12153600										



```
import numpy as np
import pandas as pd
df=pd.read_csv("C:/WSFT.csv")
print(df)
```

Run: ex.1 x

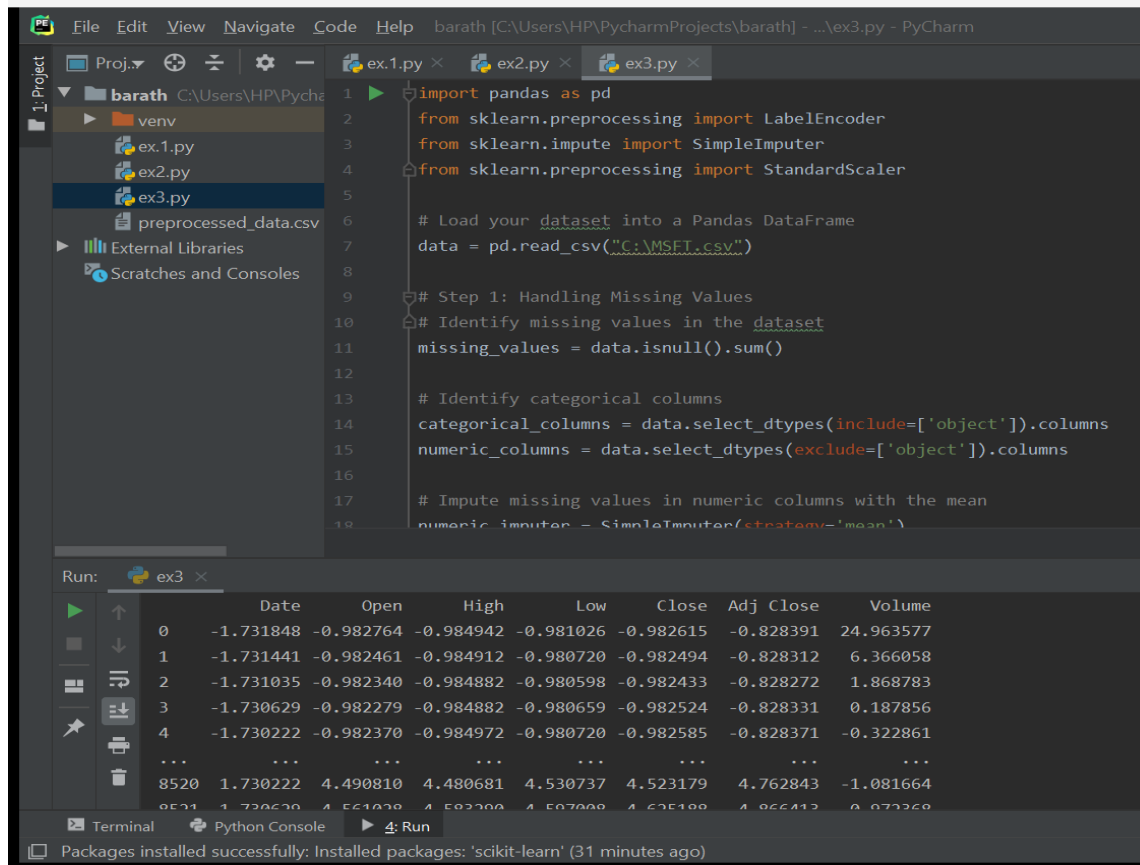
```
C:\Users\HP\PycharmProjects\barath\venv\Scripts\python.exe C:/Users/HP/PycharmProjects/barath/ex.1.py
```

	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	...	0.09809	0.063107	47894400
5	1986-03-20	0.09809	0.09809	...	0.095486	0.061432	58435200

Terminal: Packages installed successfully. Installed packages: 'pandas' (5 minutes ago)

Data Preprocessing:

In Case of this part we will be finding the Total number of missing values, Clean and preprocess the data in the given dataset and Handling the Categorical features into numerical representations.

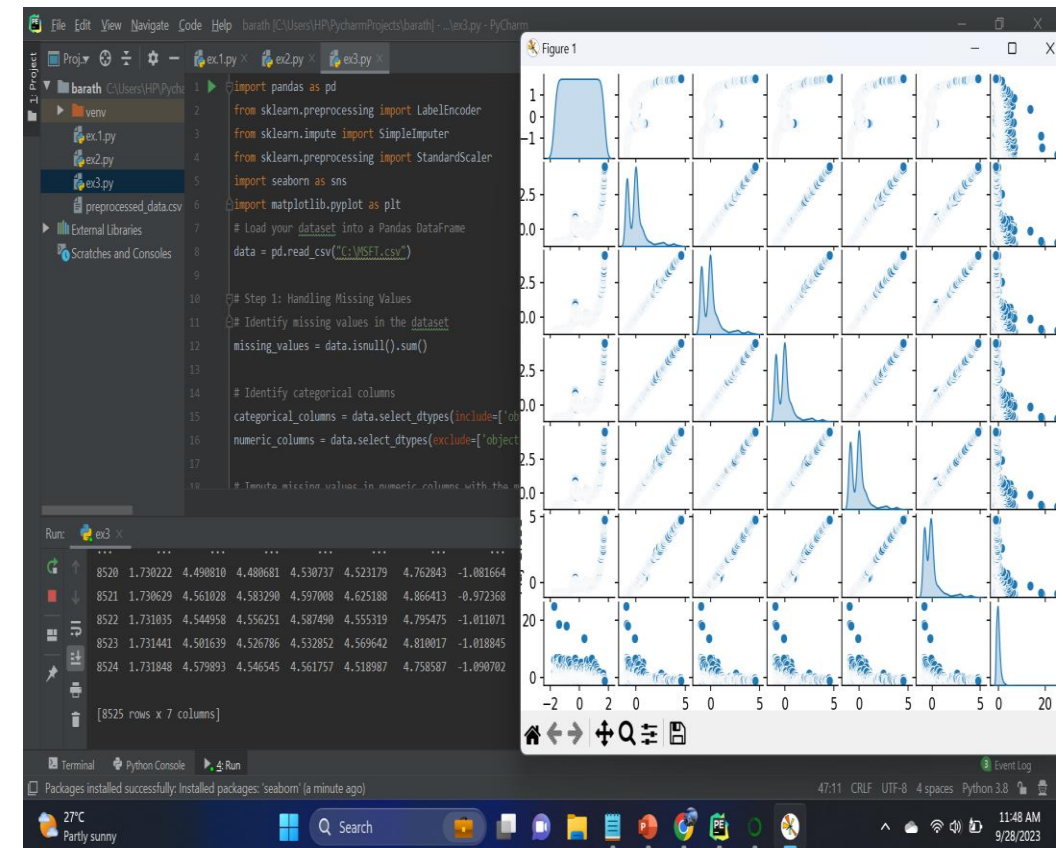


```
1 import pandas as pd
2 from sklearn.preprocessing import LabelEncoder
3 from sklearn.impute import SimpleImputer
4 from sklearn.preprocessing import StandardScaler
5
6 # Load your dataset into a Pandas DataFrame
7 data = pd.read_csv("C:\\MSFT.csv")
8
9 # Step 1: Handling Missing Values
10 # Identify missing values in the dataset
11 missing_values = data.isnull().sum()
12
13 # Identify categorical columns
14 categorical_columns = data.select_dtypes(include=['object']).columns
15 numeric_columns = data.select_dtypes(exclude=['object']).columns
16
17 # Impute missing values in numeric columns with the mean
18 numeric_imputer = SimpleImputer(strategy='mean')
```

Run: ex3

	Date	Open	High	Low	Close	Adj Close	Volume
0	-1.731848	-0.982764	-0.984942	-0.981026	-0.982615	-0.828391	24.963577
1	-1.731441	-0.982461	-0.984912	-0.980720	-0.982494	-0.828312	6.366058
2	-1.731035	-0.982340	-0.984882	-0.980598	-0.982433	-0.828272	1.868783
3	-1.730629	-0.982279	-0.984882	-0.980659	-0.982524	-0.828331	0.187856
4	-1.730222	-0.982370	-0.984972	-0.980720	-0.982585	-0.828371	-0.322861
...
8520	1.730222	4.490810	4.480681	4.530737	4.523179	4.762843	-1.081664
8521	1.730629	4.561028	4.583290	4.597088	4.625188	4.866413	-0.972368
8522	1.731035	4.544958	4.566251	4.587490	4.555319	4.795475	-1.011871
8523	1.731441	4.501639	4.526786	4.532852	4.569642	4.810017	-1.018845
8524	1.731848	4.579893	4.546545	4.561757	4.518987	4.758587	-1.090702

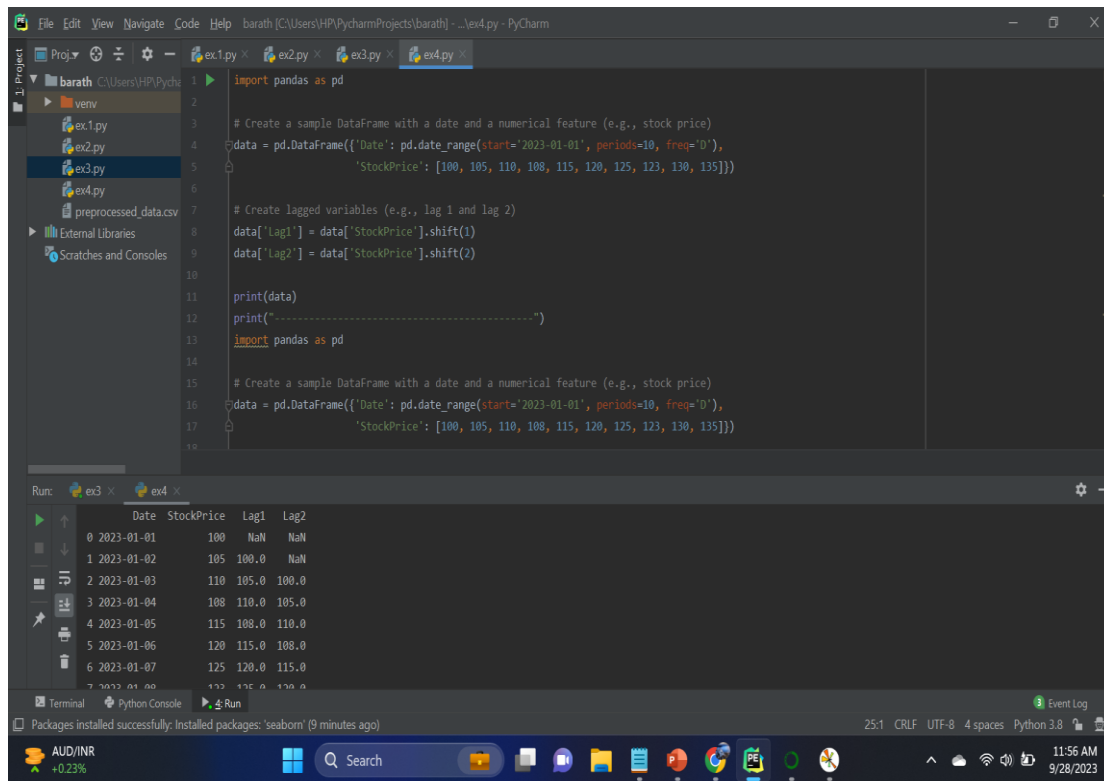
Terminal: Packages installed successfully: 'scikit-learn' (31 minutes ago)



Feature Engineering:

Is the process of creating new features or modifying existing ones to improve the predictive power of a machine learning model. It involves extracting valuable information from the raw data or transforming features in a way that makes them more informative for the task at hand. Here are some common techniques

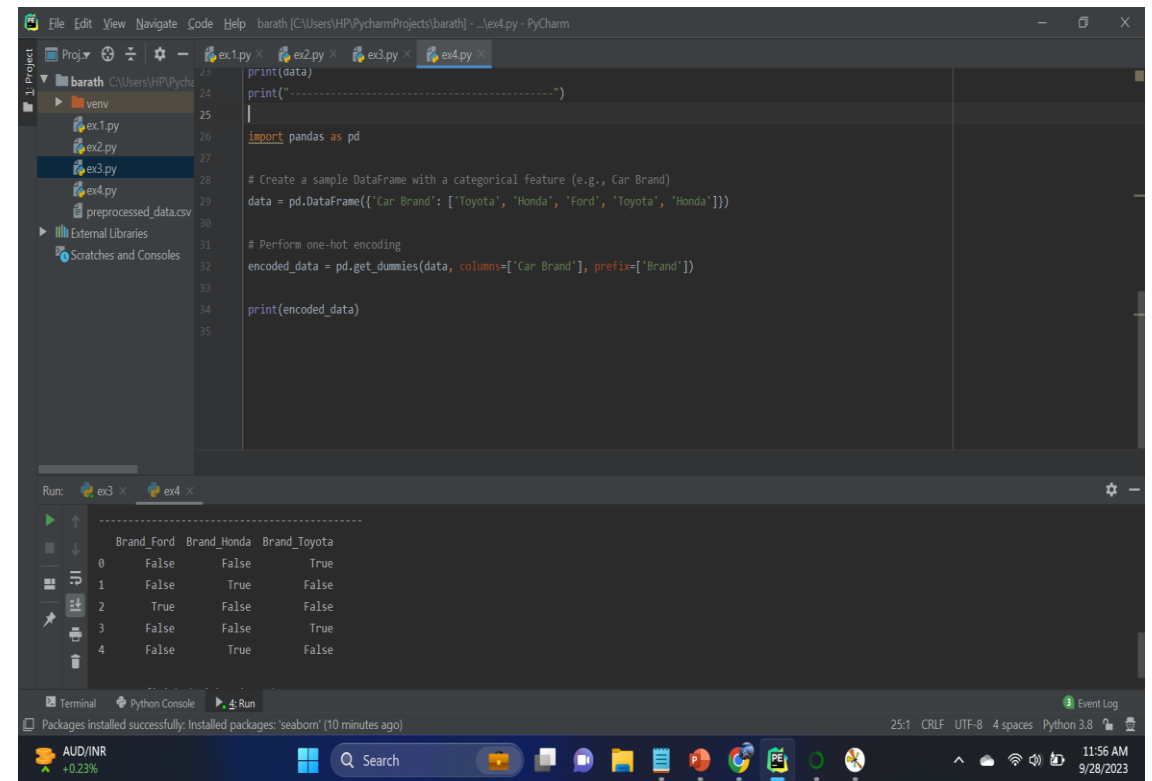
1. Creating Lagged Variable:



```
1 import pandas as pd
2
3 # Create a sample DataFrame with a date and a numerical feature (e.g., stock price)
4 data = pd.DataFrame({'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
5                     'StockPrice': [100, 105, 110, 108, 115, 120, 125, 130, 135]})
6
7 # Create lagged variables (e.g., lag 1 and lag 2)
8 data['Lag1'] = data['StockPrice'].shift(1)
9 data['Lag2'] = data['StockPrice'].shift(2)
10
11 print(data)
12 print("-----")
13 import pandas as pd
14
15 # Create a sample DataFrame with a date and a numerical feature (e.g., stock price)
16 data = pd.DataFrame({'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
17                     'StockPrice': [100, 105, 110, 108, 115, 120, 125, 130, 135]})
```

Date	StockPrice	Lag1	Lag2
0 2023-01-01	100	NaN	NaN
1 2023-01-02	105	100.0	NaN
2 2023-01-03	110	105.0	100.0
3 2023-01-04	108	110.0	105.0
4 2023-01-05	115	108.0	110.0
5 2023-01-06	120	115.0	108.0
6 2023-01-07	125	120.0	115.0
7 2023-01-08	130	125.0	120.0
8 2023-01-09	135	130.0	125.0

2. Moving Averages and One-Hot Encoding



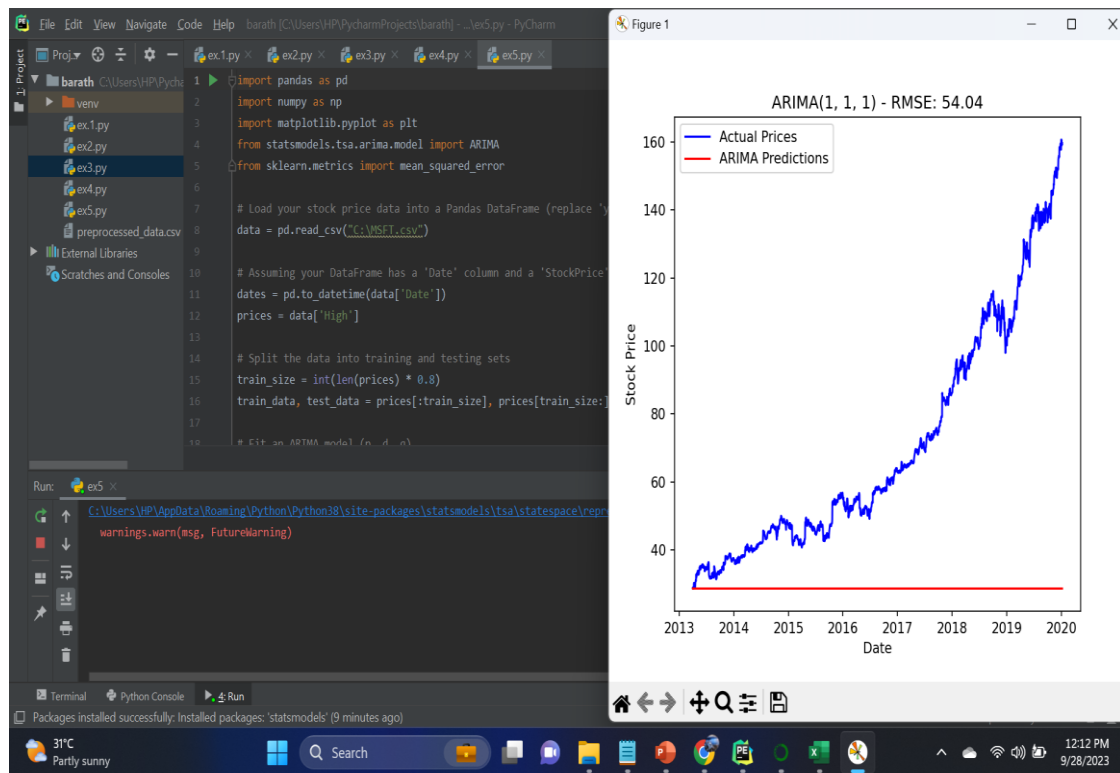
```
1 print(data)
2 print("-----")
3
4 import pandas as pd
5
6 # Create a sample DataFrame with a categorical feature (e.g., Car Brand)
7 data = pd.DataFrame({'Car Brand': ['Toyota', 'Honda', 'Ford', 'Toyota', 'Honda']})
8
9 # Perform one-hot encoding
10 encoded_data = pd.get_dummies(data, columns=['Car Brand'], prefix=['Brand'])
11
12 print(encoded_data)
```

	Brand_Ford	Brand_Honda	Brand_Toyota
0	False	False	True
1	False	True	False
2	True	False	False
3	False	False	True
4	False	True	False

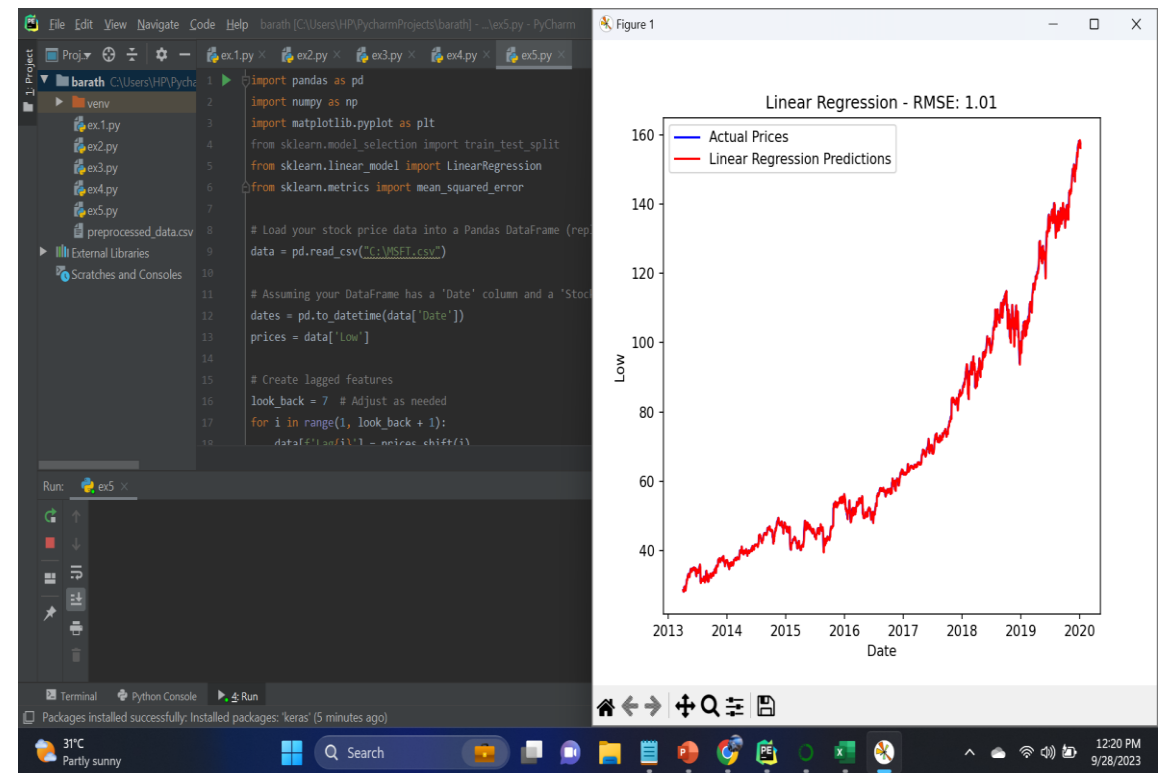
Model selection:

Is a critical step in time series forecasting, and it involves choosing the most suitable algorithm or model to make accurate predictions. When it comes to predicting stock prices, you have various options, including traditional statistical models like ARIMA (AutoRegressive Integrated Moving Average) and machine learning models like Long Short-Term Memory (LSTM) networks.

ARIMA Model Example

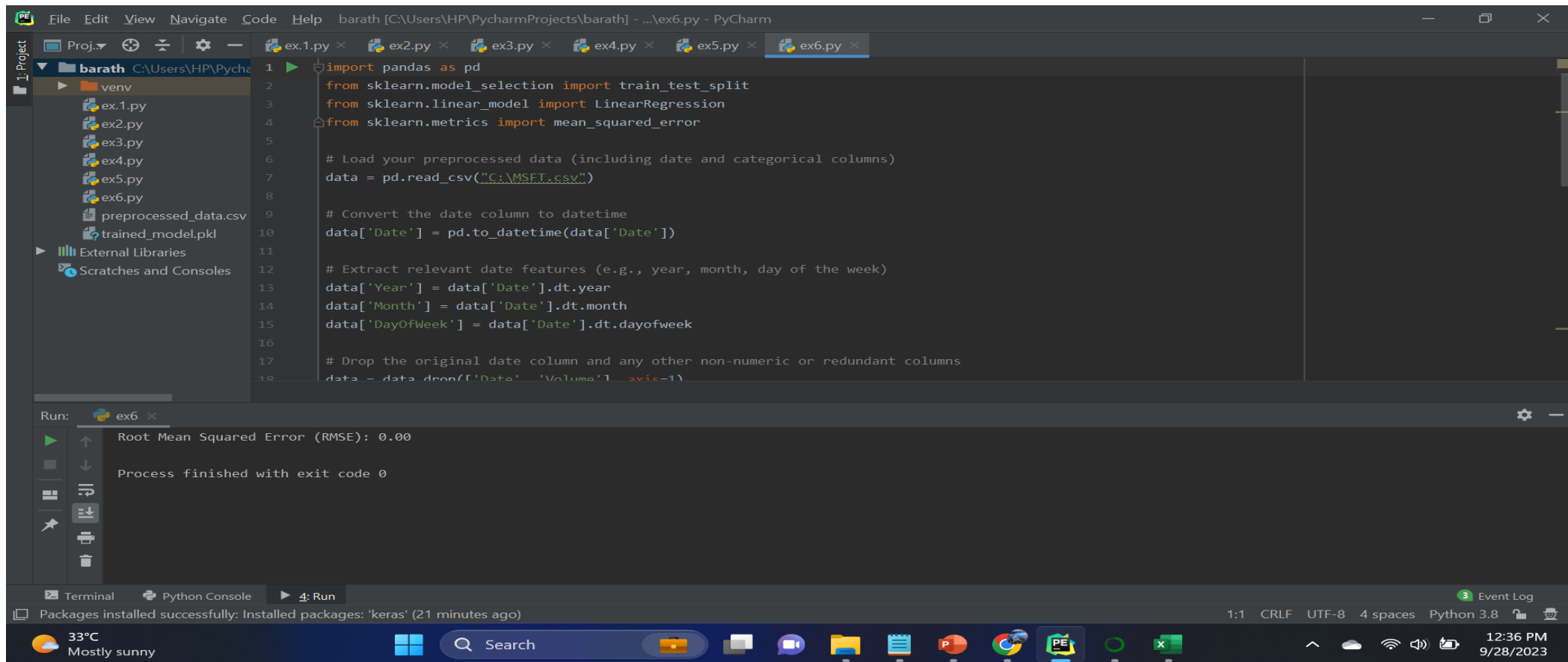


LSTM Model Example



Model training :

Is the process of using a machine learning algorithm or model to learn patterns and relationships within your preprocessed data. During this phase, the model adjusts its internal parameters to minimize the difference between its predictions and the actual target values in your training dataset. Here's an overview of the model training process, along with Python code examples using scikit-learn for a simple linear regression model



```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LinearRegression
4 from sklearn.metrics import mean_squared_error
5
6 # Load your preprocessed data (including date and categorical columns)
7 data = pd.read_csv("C:\\MSFT.csv")
8
9 # Convert the date column to datetime
10 data['Date'] = pd.to_datetime(data['Date'])
11
12 # Extract relevant date features (e.g., year, month, day of the week)
13 data['Year'] = data['Date'].dt.year
14 data['Month'] = data['Date'].dt.month
15 data['DayOfWeek'] = data['Date'].dt.dayofweek
16
17 # Drop the original date column and any other non-numeric or redundant columns
18 data = data.drop(['Date', 'Volume', 'Adj-1'])
```

Run: ex6 ×

Root Mean Squared Error (RMSE): 0.00

Process finished with exit code 0

Terminal Python Console 4: Run

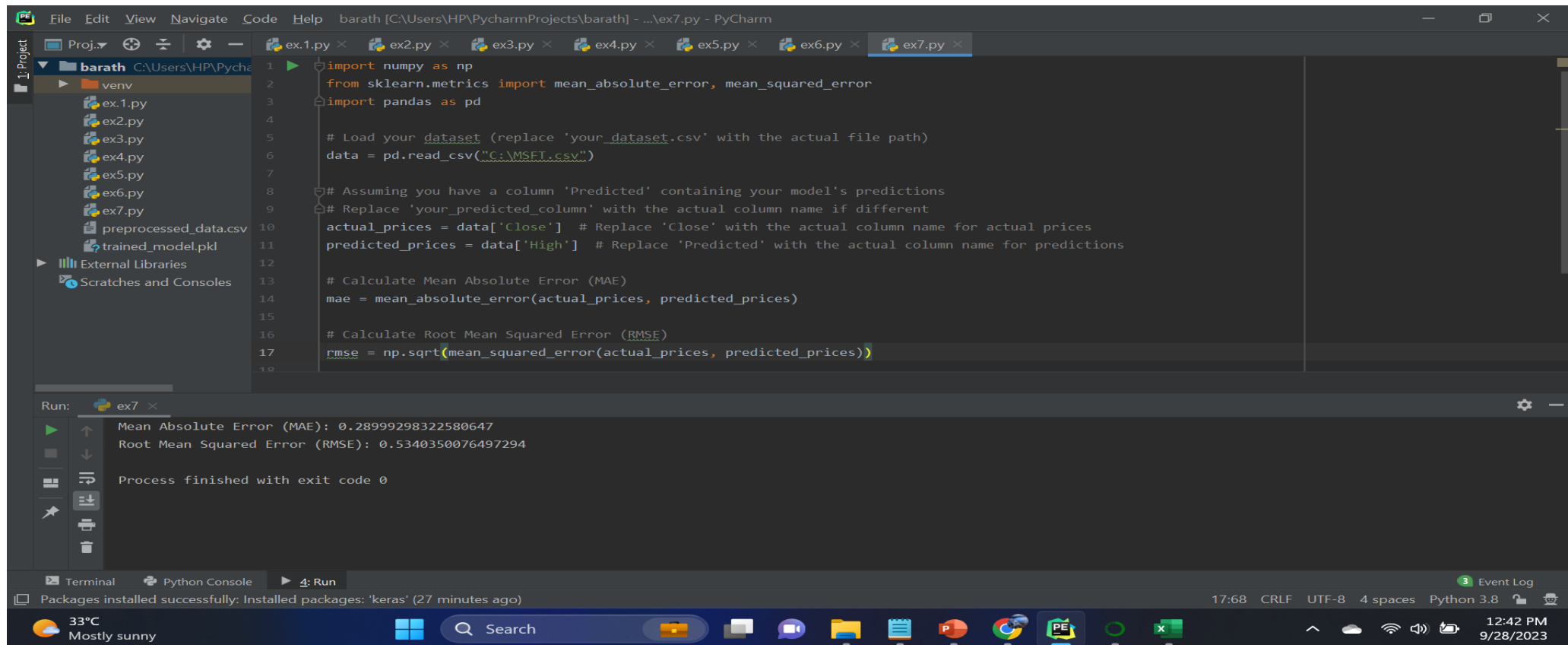
Packages installed successfully: Installed packages: 'keras' (21 minutes ago)

1:1 CRLF UTF-8 4 spaces Python 3.8

33°C Mostly sunny 12:36 PM 9/28/2023

Evaluation:

To evaluate the performance of your time series forecasting model, you can use various metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and others. These metrics provide insights into how well your model's predictions align with the actual data. Here's how to compute these metrics using Python.



```
1 import numpy as np
2 from sklearn.metrics import mean_absolute_error, mean_squared_error
3 import pandas as pd
4
5 # Load your dataset (replace 'your_dataset.csv' with the actual file path)
6 data = pd.read_csv("C:\\MSFT.csv")
7
8 # Assuming you have a column 'Predicted' containing your model's predictions
9 # Replace 'your_predicted_column' with the actual column name if different
10 actual_prices = data['Close'] # Replace 'Close' with the actual column name for actual prices
11 predicted_prices = data['High'] # Replace 'Predicted' with the actual column name for predictions
12
13 # Calculate Mean Absolute Error (MAE)
14 mae = mean_absolute_error(actual_prices, predicted_prices)
15
16 # Calculate Root Mean Squared Error (RMSE)
17 rmse = np.sqrt(mean_squared_error(actual_prices, predicted_prices))
18
19
```

Run: ex7 ×

```
Mean Absolute Error (MAE): 0.28999298322580647
Root Mean Squared Error (RMSE): 0.5340350076497294
Process finished with exit code 0
```

Terminal Python Console Run

Packages installed successfully: Installed packages: 'keras' (27 minutes ago)

17:68 CRLF UTF-8 4 spaces Python 3.8

33°C Mostly sunny 12:42 PM 9/28/2023