Abdullah Khalid **2023-BS-AI-35**

Lab Manual

Subject:	**Machine Learning**
Course Code:	**AI-414**

By

Abdullah Khalid:	**2023-BS-AI-35**
Supervised By:	**Sir Saeed**
Degree Name:	**BSAI-4A**
Subject Name:	**Machine Learning**

DEPARTMENT OF COMPUTATIONAL SCIENCES

FACULTY OF INFORMATION TECHNOLOGY

The University Of Faisalabad

Abdullah Khalid **2023-BS-AI-35**

Table of Contents

Project 2: Student Performance Prediction

Table Of Contents

Stock Market Price Prediction

➢ What is Python?

Python is a versatile, high-level programming language known for its readability and efficiency. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages worldwide.

➢ Project Discription

This project is basically about stock market value prediction is which our project predicts the possible outcome or closing price or our stock value . So here is python code of this stock market value prediction

➢ Project code

Block 1:

1. Importing Libraries: Before beginning, ensure you import all necessary libraries for preprocessing, modeling, and evaluation.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


import warnings
warnings.filterwarnings('ignore')
```

2. Overview of Libraries:

1)pandas – you load and work with structured data like tables.

2)numpy – Lets you do fast math and work with arrays.

3)matplotlib.pyplot – Used to draw basic charts and graphs.

4)seaborn – Makes prettier and easier statistical plots.

5)train_test_split – Splits data into training and testing sets.

6)LinearRegression – Builds a model to predict numbers using straight-line relationships.

7)mean_absolute_error, mean_squared_error, r2_score – Measure how good your predictions are.

8)warnings – Hides warning messages so your output looks cleaner.

Block 2:

This code read the dataset which is store in .csv file and also show the data by using .head() function

```
df = pd.read_csv("stock market.csv")

df.head()
```

Output:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	2000-01-03	CIPLA	EQ	1349.40	1410.0	1457.35	1380.05	1457.35	1457.35	1441.36	21060	3.035496e+12	NaN	NaN	NaN
1	2000-01-04	CIPLA	EQ	1457.35	1537.0	1537.00	1430.00	1466.05	1465.25	1460.43	30215	4.412698e+12	NaN	NaN	NaN
2	2000-01-05	CIPLA	EQ	1465.25	1474.0	1474.00	1365.00	1441.00	1435.05	1428.11	33799	4.826872e+12	NaN	NaN	NaN
3	2000-01-06	CIPLA	EQ	1435.05	1434.0	1435.00	1349.00	1365.00	1355.85	1390.55	33083	4.600356e+12	NaN	NaN	NaN
4	2000-01-07	CIPLA	EQ	1355.85	1370.0	1389.90	1247.40	1247.40	1247.55	1267.49	66536	8.433351e+12	NaN	NaN	NaN

Block 3:

This code prepares a dataset for time-series analysis by converting the date column, setting it as the index, and showing basic info about the data.

Why use it?

These steps are used to prepare time-based data properly, so you can work with it more effectively in analysis or machine learning.

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5306 entries, 2000-01-03 to 2021-04-30
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Symbol               5306 non-null   object 
 1   Series               5306 non-null   object 
 2   Prev Close           5306 non-null   float64
 3   Open                 5306 non-null   float64
 4   High                 5306 non-null   float64
 5   Low                  5306 non-null   float64
 6   Last                 5306 non-null   float64
 7   Close                5306 non-null   float64
 8   VWAP                 5306 non-null   float64
 9   Volume               5306 non-null   int64  
 10  Turnover             5306 non-null   float64
 11  Trades               2456 non-null   float64
 12  Deliverable Volume   4792 non-null   float64
 13  %Deliverble          4792 non-null   float64
dtypes: float64(11), int64(1), object(2)
memory usage: 621.8+ KB
None
```

Block 4:

This code check all the missing values in each column by using is null() function

What it does:

1. **df.isnull()** – Checks each cell in the DataFrame and returns True if the value is NaN (missing), otherwise False.
2. **.sum()** – Adds up the True values column-wise. In Python, True = 1 and False = 0

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

df.dropna(inplace=True)
```

```
Symbol                  0
Series                  0
Prev Close              0
Open                    0
High                    0
Low                     0
Last                    0
Close                   0
VWAP                    0
Volume                  0
Turnover                0
Trades               2850
Deliverable Volume    514
%Deliverble           514
dtype: int64
```

Block 5:

What it does:

Prints a quick summary of the numeric data in your DataFrame.

```
print(df.describe())
```

Output:

	Prev Close	Open	High	Low	Last	\
count	2456.000000	2456.000000	2456.000000	2456.000000	2456.000000	
mean	524.067997	525.258774	531.670827	517.884222	524.359894	
std	136.883428	137.543023	139.388193	135.176249	137.120770	
min	274.500000	274.500000	277.000000	272.850000	273.850000	
25%	404.462500	405.037500	410.025000	398.975000	405.000000	
50%	539.675000	539.775000	545.950000	533.025000	539.100000	
75%	617.150000	618.512500	628.025000	610.125000	617.400000	
max	949.300000	965.100000	966.350000	941.100000	951.000000	

	Close	VWAP	Volume	Turnover	Trades	\
count	2456.000000	2456.000000	2.456000e+03	2.456000e+03	2456.000000	
mean	524.305904	524.857948	2.427950e+06	1.414190e+14	44967.250407	
std	137.046794	137.287425	3.010674e+06	2.201972e+14	43776.047324	
min	274.500000	275.460000	1.894900e+04	5.888454e+11	613.000000	
25%	404.887500	404.460000	1.033538e+06	4.748383e+13	21724.000000	
50%	539.725000	539.820000	1.533522e+06	7.732279e+13	32967.000000	
75%	617.675000	618.787500	2.513994e+06	1.374928e+14	50869.500000	
max	949.300000	951.800000	5.689556e+07	4.498902e+15	603361.000000	

	Deliverable Volume	%Deliverble
count	2.456000e+03	2456.000000
mean	9.585372e+05	0.470284
std	7.891640e+05	0.162410
min	7.050000e+03	0.069300
25%	4.830788e+05	0.346575
50%	7.413835e+05	0.482100
75%	1.159864e+06	0.591400
max	8.607923e+06	0.906100

Block 6:

Returns the **size (dimensions)** of your DataFrame as a tuple row and columns .

```
: df.shape
```

```
: (2456, 14)
```

Block 7:

Displays the **last 20 rows** of the DataFrame.

```
df.tail(20)
```

Output:

Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
2021-03-31	CIPLA	EQ	812.20	811.00	823.95	805.00	813.95	815.10	817.28	3075406	2.513459e+14	102145.0	819904.0	0.2666
2021-04-01	CIPLA	EQ	815.10	819.50	824.00	811.60	818.00	818.30	818.32	1910949	1.563772e+14	39715.0	570752.0	0.2987
2021-04-05	CIPLA	EQ	818.30	825.00	827.00	806.10	820.00	819.75	815.61	2920638	2.382109e+14	59161.0	780655.0	0.2673
2021-04-06	CIPLA	EQ	819.75	824.00	845.00	821.85	835.70	836.15	833.50	4569120	3.808341e+14	96945.0	1046728.0	0.2291
2021-04-07	CIPLA	EQ	836.15	836.15	844.90	830.25	838.00	840.85	839.40	2404472	2.018309e+14	71257.0	796768.0	0.3314
2021-04-08	CIPLA	EQ	840.85	843.00	850.45	838.40	841.00	842.00	844.77	2459821	2.077988e+14	54329.0	953942.0	0.3878
2021-04-09	CIPLA	EQ	842.00	840.90	892.30	836.65	886.95	883.05	875.25	11189432	9.793507e+14	176022.0	1880106.0	0.1680
2021-04-12	CIPLA	EQ	883.05	890.00	940.00	879.15	907.00	902.40	913.81	30141861	2.754402e+15	516625.0	3403805.0	0.1129
2021-04-13	CIPLA	EQ	902.40	921.95	932.60	878.30	889.00	885.20	899.74	11667200	1.049746e+15	185356.0	1657842.0	0.1421
2021-04-15	CIPLA	EQ	885.20	898.80	925.40	883.90	914.00	914.20	912.14	13218002	1.205673e+15	254586.0	1195634.0	0.0905
2021-04-16	CIPLA	EQ	914.20	919.75	944.25	910.90	940.10	938.05	929.61	13746268	1.277868e+15	221996.0	5019694.0	0.3652
2021-04-19	CIPLA	EQ	938.05	945.25	955.00	924.00	950.00	946.30	945.66	17805574	1.683793e+15	315326.0	3365829.0	0.1890
2021-04-20	CIPLA	EQ	946.30	965.10	966.35	941.10	951.00	949.30	951.80	10205616	9.713696e+14	157298.0	1904032.0	0.1866
2021-04-22	CIPLA	EQ	949.30	958.80	963.15	938.85	945.50	944.35	951.75	8522208	8.111023e+14	153862.0	2502617.0	0.2937
2021-04-23	CIPLA	EQ	944.35	944.35	952.85	922.75	936.40	935.60	934.36	7972589	7.449235e+14	139014.0	1545821.0	0.1939
2021-04-26	CIPLA	EQ	935.60	935.60	940.00	902.15	907.95	905.40	914.11	10255697	9.374885e+14	161321.0	3451318.0	0.3365
2021-04-27	CIPLA	EQ	905.40	913.00	919.50	901.00	911.20	912.40	909.22	5669049	5.154420e+14	120188.0	1126288.0	0.1987
2021-04-28	CIPLA	EQ	912.40	914.35	918.00	902.95	906.25	910.20	909.94	7251009	6.597970e+14	134413.0	1772739.0	0.2445
2021-04-29	CIPLA	EQ	910.20	911.95	917.40	904.00	906.00	906.50	910.22	4953091	4.508421e+14	88604.0	1122534.0	0.2266
2021-04-30	CIPLA	EQ	906.50	900.75	921.00	900.75	910.00	910.35	911.47	6459737	5.887824e+14	121466.0	2004555.0	0.3103

Block 8:

Returns a random row from the DataFrame each time you run it.

```
df.sample
```

Output:

```
<bound method NDFrame.sample of           Symbol Series  Prev Close    Open   High     Low    Last   Close  \
Date
2011-06-01  CIPLA     EQ      326.05  326.05  328.5  324.25  325.70  327.25
2011-06-02  CIPLA     EQ      327.25  324.00  333.5  324.00  325.60  325.20
2011-06-03  CIPLA     EQ      325.20  328.00  328.8  323.35  325.85  324.60
2011-06-06  CIPLA     EQ      324.60  325.95  332.5  324.00  332.00  331.70
2011-06-07  CIPLA     EQ      331.70  331.50  339.0  331.00  337.85  337.60
...           ...    ...         ...     ...    ...     ...     ...     ...
2021-04-26  CIPLA     EQ      935.60  935.60  940.0  902.15  907.95  905.40
2021-04-27  CIPLA     EQ      905.40  913.00  919.5  901.00  911.20  912.40
2021-04-28  CIPLA     EQ      912.40  914.35  918.0  902.95  906.25  910.20
2021-04-29  CIPLA     EQ      910.20  911.95  917.4  904.00  906.00  906.50
2021-04-30  CIPLA     EQ      906.50  900.75  921.0  900.75  910.00  910.35

              VWAP    Volume      Turnover    Trades  Deliverable Volume  \
Date
2011-06-01  326.39    644602  2.103933e+13   13861.0            316448.0
2011-06-02  329.04   1433409  4.716429e+13   24573.0            747283.0
2011-06-03  325.81    492898  1.605897e+13    8755.0            215837.0
2011-06-06  329.90   1655179  5.460432e+13   18585.0            982798.0
2011-06-07  337.44   2421958  8.172751e+13   25889.0           1544387.0
...            ...       ...           ...       ...                ...
2021-04-26  914.11  10255697  9.374885e+14  161321.0           3451318.0
2021-04-27  909.22   5669049  5.154420e+14  120188.0           1126288.0
2021-04-28  909.94   7251009  6.597970e+14  134413.0           1772739.0
2021-04-29  910.22   4953091  4.508421e+14   88604.0           1122534.0
2021-04-30  911.47   6459737  5.887824e+14  121466.0           2004555.0

            %Deliverble
Date
2011-06-01       0.4909
2011-06-02       0.5213
2011-06-03       0.4379
2011-06-06       0.5938
2011-06-07       0.6377
...                 ...
2021-04-26       0.3365
2021-04-27       0.1987
2021-04-28       0.2445
2021-04-29       0.2266
2021-04-30       0.3103

[2456 rows x 14 columns]>
```

Block 9:

Creates a new DataFrame called df_model that only contains the columns: 'Open', 'High', 'Low', 'Volume', and 'Close' from the original DataFrame df..

Shows the first 5 rows of this new DataFrame, so you can quickly see what data it contains.

```
df_model = df[['Open', 'High', 'Low', 'Volume', 'Close']]
df_model.head()
```

Output:

Date	Open	High	Low	Volume	Close
2011-06-01	326.05	328.5	324.25	644602	327.25
2011-06-02	324.00	333.5	324.00	1433409	325.20
2011-06-03	328.00	328.8	323.35	492898	324.60
2011-06-06	325.95	332.5	324.00	1655179	331.70
2011-06-07	331.50	339.0	331.00	2421958	337.60

Block 10:

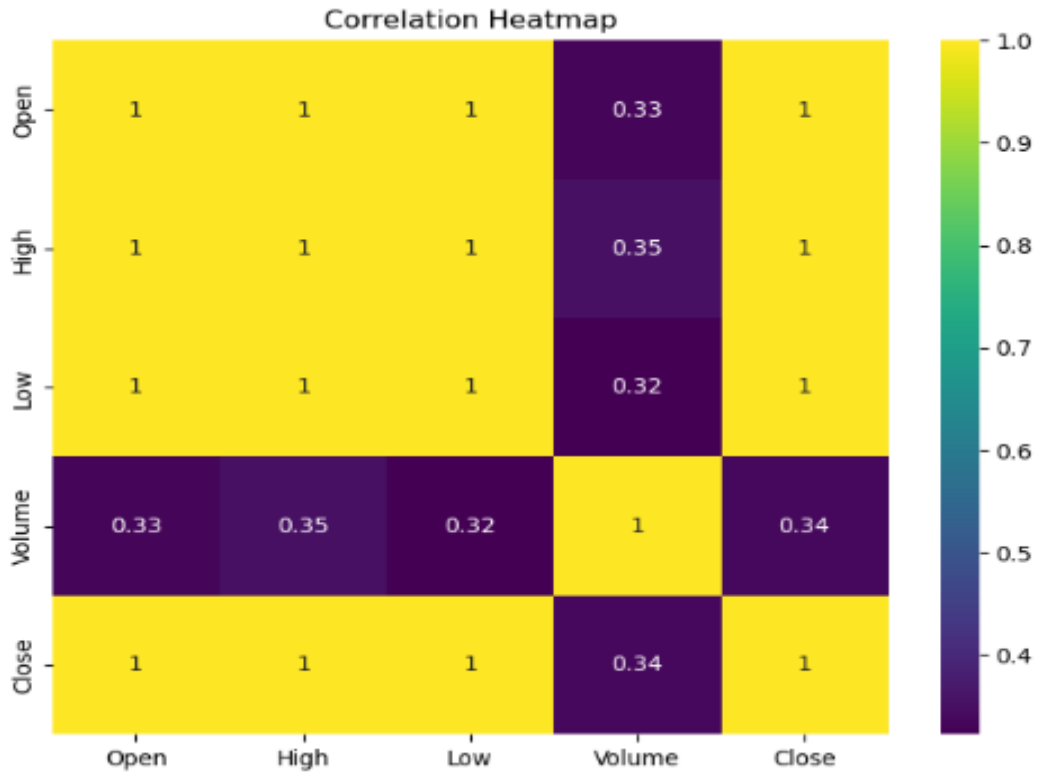
1) Make the picture size bigger so it's easy to see.
2) Draw a color chart showing how each column is related to the others.
3) Write the number of each relationship on the chart.
4) Give the chart a title "Correlation Heatmap."
5) Show the chart on the screen.

➢ Why use this:

- To **quickly see which variables are positively or negatively correlated**.
- Helps in understanding relationships and deciding which features to use in modeling.

```
plt.figure(figsize=(8,6))
sns.heatmap(df_model.corr(), annot=True, cmap='viridis')
plt.title("Correlation Heatmap")
plt.show()
```

Output:



Block 11:

1. **X = df_model[['Open', 'High', 'Low', 'Volume']]**
   Choose the columns 'Open', 'High', 'Low', and 'Volume' as the **input features** (the data you will use to predict).
2. **y = df_model['Close']**
   Choose the column 'Close' as the **target** (the value you want to predict).

```
X = df_model[['Open', 'High', 'Low', 'Volume']]
y = df_model['Close']
```

Block 12:

1) **Import train_test_split**
   This function splits your data into training and testing parts**.**

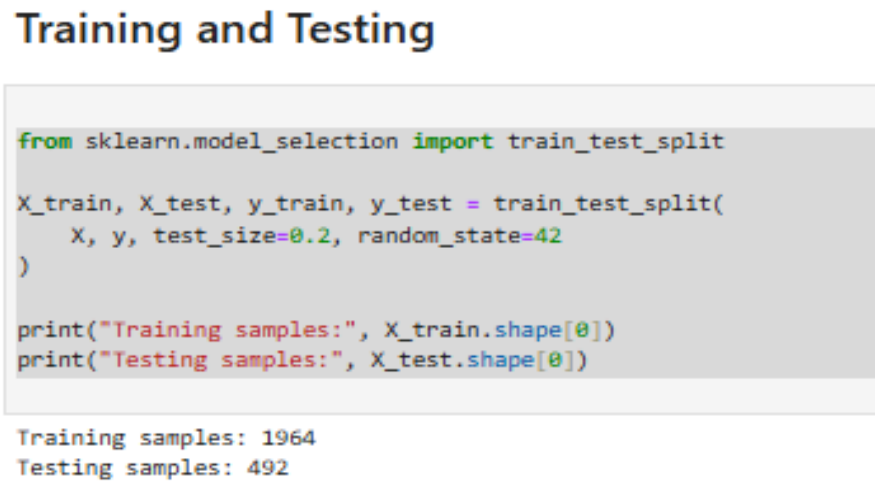
Split the data:

1) **X and y** are your features and target.

2) **test_size=0.2** means 20% of data goes to testing, 80% to training.
3) **random_state=42** makes sure the split is the same every time you run it.

Print how many samples are in each set

- Shows the number of rows in training and testing data.



Block 13:

Import Linear Regression Model

Brings in the tool to create a straight-line prediction model.

Create the Model Object

Makes a new Linear Regression model ready to be trained.

Train the Model

Teaches the model to find the best line that fits the training data (X_train predicting y_train).

Confirm Training is Done

Prints a message to let you know the model finished learning.

This imports the Linear Regression model from Scikit-Learn's linear model library.

```
from sklearn.linear_model import LinearRegression

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

print("Model training completed.")
```

```
Model training completed.
```

Block 14:

What it does:
Uses the trained model to guess (predict) the target values (y) for the test data (X_test).

```
y_pred = model.predict(X_test)
```

Block 15:

1) Import tools to measure how good the predictions are.
2) MAE (Mean Absolute Error): Average size of mistakes (how far off the guesses are).
3) MSE (Mean Squared Error): Like MAE but bigger mistakes count more (squared).
4) RMSE (Root Mean Squared Error): Square root of MSE, shows error in original units.
5) R^2 (R-squared): How well the model fits the data (1 means perfect, 0 means bad).
6) Print all these scores to see how well your model did.

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared Score (R²): {r2:.2f}")
```

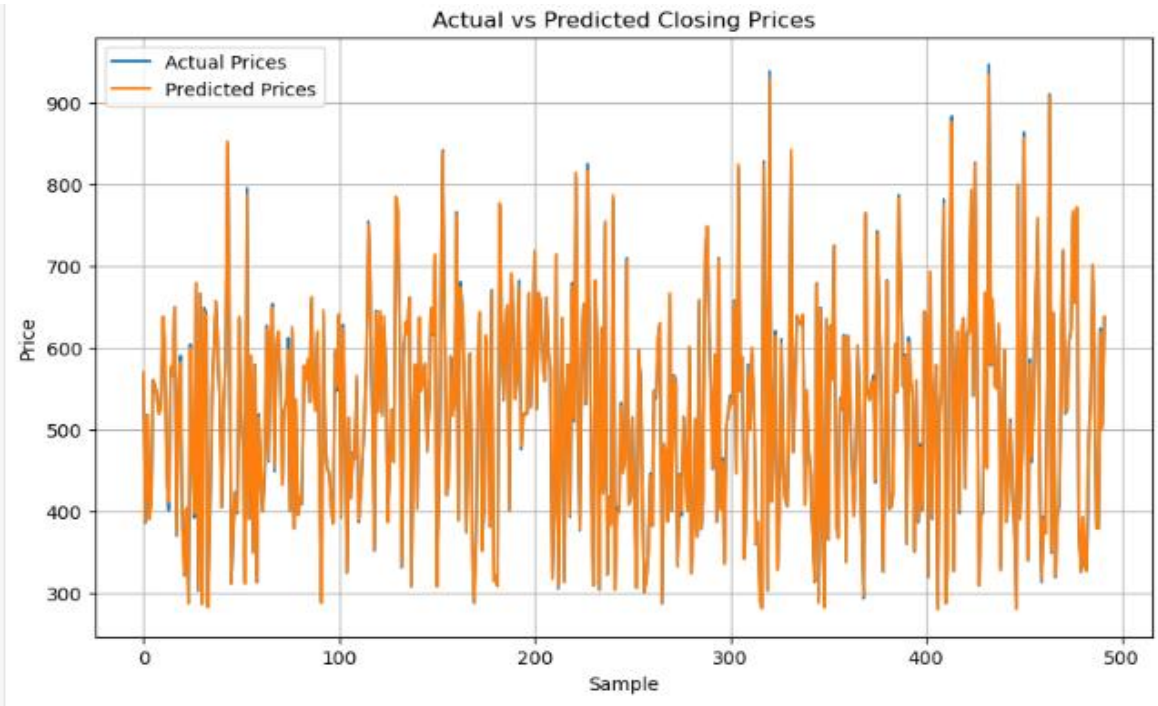
```
Mean Absolute Error (MAE): 2.49
Root Mean Squared Error (RMSE): 3.37
R-squared Score (R²): 1.00
```

Block 16:

1) Set the plot size to 10 inches wide and 6 inches tall.
2) Draw a line for the actual closing prices from the test data (y_test).
3) Draw a line for the predicted closing prices from the model (y_pred).
4) Add a legend to label the lines (“Actual Prices” and “Predicted Prices”).
5) Add a title to the plot: "Actual vs Predicted Closing Prices".
6) Label the x-axis as "Sample" (each point’s position in the test set).
7) Label the y-axis as "Price".
8) Add a grid to the background for easier reading.
9) Show the plot on the screen.

```python
plt.figure(figsize=(10,6))
plt.plot(y_test.values, label='Actual Prices')
plt.plot(y_pred, label='Predicted Prices')
plt.legend()
plt.title("Actual vs Predicted Closing Prices")
plt.xlabel("Sample")
plt.ylabel("Price")
plt.grid(True)
plt.show()
```

Output:



Block 17:

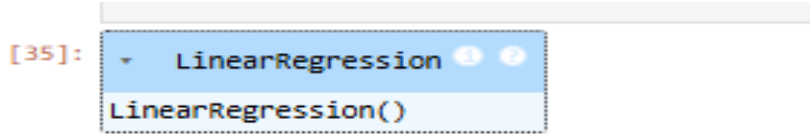
- Select important columns ('Open', 'High', 'Low', 'Volume', 'Close') from the DataFrame and remove any rows with missing values using .dropna().
- Set X as the input features ('Open', 'High', 'Low', 'Volume').
- Set y as the target variable ('Close').
- Split the data into training and testing sets, using 80% for training and 20% for testing, with a fixed random seed (random_state=42) for reproducibility.
- Create a Linear Regression model object.
- Train the model on the training data (X_train, y_train).

```
df_model = df[['Open', 'High', 'Low', 'Volume', 'Close']].dropna()

X = df_model[['Open', 'High', 'Low', 'Volume']]
y = df_model['Close']


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)
```

Output:



Block 18:

1) Ask the user to enter today's Open, High, Low prices, and Volume (as numbers).
2) Put these values into a NumPy array shaped like the model expects (one row, four columns).
3) Use the trained model to predict today's Close price based on the input data.
4) Print the predicted Close price, rounded to 2 decimal places.

Output:

```
Enter today's Open price:  250
Enter today's High price:  310
Enter today's Low price:  200
Enter today's Volume:  45000
Predicted Close Price: 261.47
```

Analysis:

This code is designed to predict the closing price of a stock using simple historical market data. First, it selects important columns like Open, High, Low prices, Volume, and Close price from the dataset and removes any missing data to keep the data clean. Then, it separates the data into input features (Open, High, Low, Volume) and the target variable (Close price), which is what the model will learn to predict.

Next, the data is split into two parts: one for training the model (80%) and one for testing how well the model performs on new, unseen data (20%). This helps ensure the model doesn't just memorize the data but can also make accurate predictions.

A Linear Regression model is created and trained on the training data. This model learns the relationship between the input features and the closing price by finding the best-fitting line.

Finally, the code allows the user to input today's Open, High, Low prices, and Volume, then uses the trained model to predict today's closing price. This is a practical way to estimate the closing price based on current market conditions.

Overall, the code covers key steps in a machine learning workflow: data preparation, splitting, model training, and prediction. It uses simple, understandable methods that are great for learning how stock price prediction can work using linear regression**.**

Conclusion

This code provides a clear and practical example of how to use historical stock data to predict future prices using machine learning. By training a linear regression model on important market features, it can make reasonable predictions of the closing price based on new input data. While linear regression is a straightforward approach and may not capture all market complexities, it is a great starting point for understanding price prediction. With more data and advanced models, predictions can be improved further. Overall, this project shows how data science can help make informed guesses about stock market behavior.

Student Performance Prediction

What is Python?

Python is a versatile, high-level programming language known for its readability and efficiency. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages worldwide.

Project Discription

This project is basically about student performance prediction is which our project predicts the possible outcome of students result is the student is pass or fail on the basis of there free time . So here is python code of this student performance prediction.

Project code

Block 1:

1. **Importing Libraries:** Before beginning, ensure you import all necessary libraries for preprocessing, modeling, and evaluation.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

2. Overview of Libraries:

- **pandas (pd)**
- For data loading and manipulation (tables, CSV, Excel).
- Works with DataFrames (like SQL tables).

- **numpy (np)**
- For numerical operations and array handling.
- Used for fast math on large data sets.
- **matplotlib.pyplot (plt)**
- For basic plotting (line, bar).
- Low-level but customizable visualizations.
- **seaborn (sns)**
- For statistical and attractive plots.
- Built on top of Matplotlib (e.g., heatmaps, pairplots).
- **train_test_split (from sklearn.model_selection)**
- Splits data into training and testing sets.
- **LabelEncoder (from sklearn.preprocessing)**
- Converts categorical labels to numbers.
- **RandomForestClassifier (from sklearn.ensemble)**
- Machine learning model based on decision tree ensembles.
- Good for classification tasks.
- **accuracy_score, confusion_matrix, classification_report (from sklearn.metrics)**
- Evaluate model accuracy.
- Generate performance metrics like precision, recall, F1-score.

Block 2:

This line reads the student-mat.csv file into a DataFrame using pandas, treating the semicolon (;) as the column separator. It's used when the CSV file uses ; instead of the usual comma to separate values.

load dataset

```
df = pd.read_csv('student-mat.csv', sep=';')
```

Block 3:

This command displays the first 5 rows of the DataFrame df. It's commonly used to quickly preview the structure and contents of a dataset after loading it.

Why use it?

This is use to display the first five (5) rows in data frame.

Shows the first 5 rows of the DataFrame (df).

```
df.head()
```

Output:

[3]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

5 rows × 33 columns

Block 4:

This command provides summary statistics for the numerical columns in the DataFrame df, including these values.

- count – number of non-null values
- mean – average value
- std – standard deviation
- min – minimum value

- 25%, 50% (median), 75% – quartiles
- max – maximum value

Gives summary statistics of the numeric columns in the DataFrame.

```
[4]: df.describe()
```

Output:

[4]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	
count	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395
mean	16.696203	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304	3.235443	3.108861	1.481013	2.291139	3.554430	5.708861	1(
std	1.276043	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659	0.998862	1.113278	0.890741	1.287897	1.390303	8.003096	:
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	:
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.000000	1.000000	1.000000	3.000000	0.000000	8
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.000000	1.000000	2.000000	4.000000	4.000000	1'
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.000000	2.000000	3.000000	5.000000	8.000000	1:
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	75.000000	1!

Block 5:

This command displays the last 5 rows of the DataFrame df. It's useful for quickly checking how the dataset ends or verifying recent additions or changes.

```
[5]: df.tail()
```

Output:

[5]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	5	4	4	5	4	11	9	9	9
391	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	3	14	16	16
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	5	3	3	3	3	3	10	8	7
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	0	11	12	10
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	3	2	3	3	3	5	5	8	9	9

5 rows × 33 columns

Block 6:

Shows a summary of the DataFrame, including:

Number of rows and columns

Column names¶

Data types (e.g., int, float, object)

Non-null counts (how many values are not missing)

```
[6]: df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      395 non-null    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64 
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   Pstatus     395 non-null    object
 6   Medu        395 non-null    int64 
 7   Fedu        395 non-null    int64 
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64 
 13  studytime   395 non-null    int64 
 14  failures    395 non-null    int64 
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 23  famrel      395 non-null    int64 
 24  freetime    395 non-null    int64 
 25  goout       395 non-null    int64 
 26  Dalc        395 non-null    int64 
 27  Walc        395 non-null    int64 
 28  health      395 non-null    int64 
 29  absences    395 non-null    int64 
 30  G1          395 non-null    int64 
 31  G2          395 non-null    int64 
 32  G3          395 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 102.0+ KB
```

Block 7:

Checks for missing (null) values in the DataFrame

```
Checks for missing (null) values in the DataFrame
```

```
[7]: df.isnull().sum()
```

Output:

```
[7]: school        0
     sex           0
     age           0
     address       0
     famsize       0
     Pstatus       0
     Medu          0
     Fedu          0
     Mjob          0
     Fjob          0
     reason        0
     guardian      0
     traveltime    0
     studytime     0
     failures      0
     schoolsup     0
     famsup        0
     paid          0
     activities    0
     nursery       0
     higher        0
     internet      0
     romantic      0
     famrel        0
     freetime      0
     goout         0
     Dalc          0
     Walc          0
     health        0
     absences      0
     G1            0
     G2            0
     G3            0
     dtype: int64
```

Block 8:

Creates a new column in the DataFrame called 'pass'.

Uses the values in column 'G3' (usually final exam scores).

If the score is 10 or more, marks it as 1 (pass), otherwise 0 (fail)

```
[8]: df['pass'] = df['G3'].apply(lambda x: 1 if x >= 10 else 0)
```

.

Block 9:

Drop unnecessary columns.

```
[9]: df = df.drop(['G1', 'G2', 'G3'], axis=1)
```

Block 10:

What this does:

- Creates a LabelEncoder instance called label_enc.
- Loops through each column in the DataFrame df.
- For every column with categorical text data (dtype == 'object'), it converts the text labels into numeric labels using LabelEncoder.

Why?

Machine learning models require numeric input, so this step encodes categorical variables into numbers for model training.

Encode Categorical Columns

```
[10]: label_enc = LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = label_enc.fit_transform(df[col])
```

Block 11:

What this does:

- **X = df.drop('pass', axis=1)**
  Creates feature matrix X by removing the target column 'pass' from df.
- **y = df['pass']**
  Assigns the target variable y as the 'pass' column.

- **train_test_split(...)**
  Splits the data into training and testing sets:
  - 80% training (X_train, y_train)
  - 20% testing (X_test, y_test)
  - random_state=42 ensures reproducible splits.

This prepares your data for model training and evaluation.

```
[11]: X = df.drop('pass', axis=1)
      y = df['pass']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Block 12:

What this does:

- Creates a Random Forest classifier instance with a fixed random seed (random_state=42) for reproducibility.
- Trains the model on the training data (X_train, y_train), so it learns patterns to predict the target variable pass.

```
[12]: model = RandomForestClassifier(random_state=42)
      model.fit(X_train, y_train)
```

Output:

```
[12]: ▾ RandomForestClassifier
      RandomForestClassifier(random_state=42)
```

Block 13:

What this does:

- model.predict(X_test) uses the trained model to predict the target (pass) for the test features.
- accuracy_score(y_test, y_pred) calculates the proportion of correctly predicted samples.
- Prints the accuracy score.
- Prints the classification report, which includes precision, recall, F1-score, and support for each class.

This helps you evaluate how well your model performs on unseen data.

```
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 0.7215189873417721

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.30      0.42        27
           1       0.72      0.94      0.82        52

    accuracy                           0.72        79
   macro avg       0.72      0.62      0.62        79
weighted avg       0.72      0.72      0.68        79
```

Block 14:

What this does:

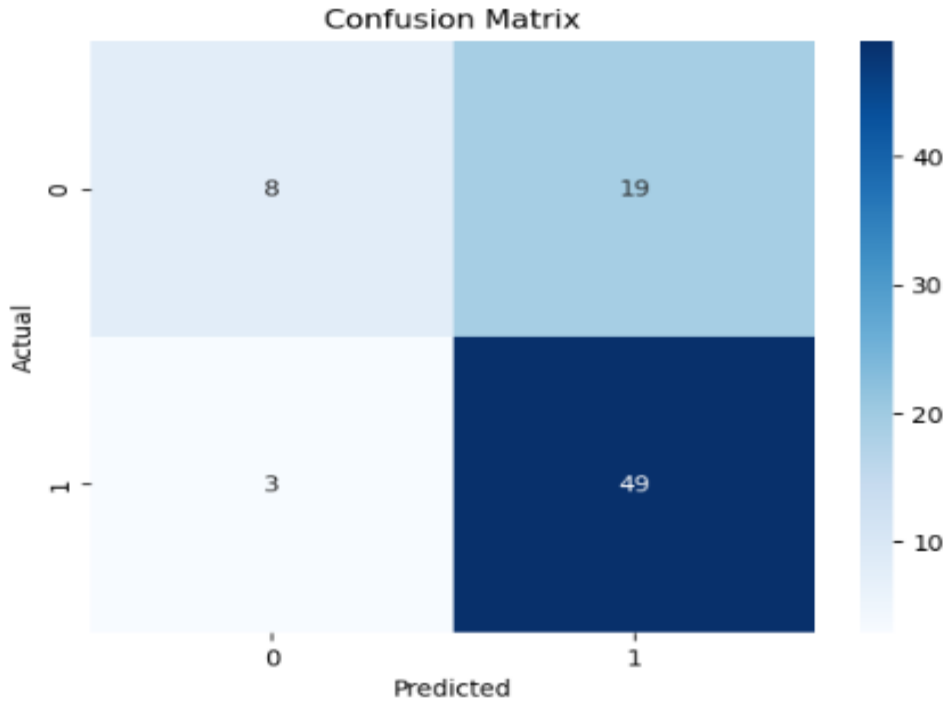
- Computes the confusion matrix comparing actual (y_test) vs predicted (y_pred) labels.
- Uses Seaborn to create a heatmap of the confusion matrix with:

- Numbers annotated inside cells (annot=True)
- Integer formatting (fmt='d')
- Blue color scheme (cmap='Blues')

- Labels the axes as Predicted and Actual.
- Adds a title and displays the plot.

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Output:



Block 15:

1. Ask the user to enter values for several student features (like age, study time, absences, etc.).
2. Take the average values of all other features from the training data.
3. Combine the user inputs with these average values to create a complete input record.
4. Convert this record into a DataFrame for the model.
5. Use the trained model to predict if the student will pass (1) or fail (0).
6. Print the prediction result as “PASS” or “FAIL”.

```
print("\n--- Enter your details to predict pass/fail ---")
input_data = {
    'age': int(input("Age: ")),
    'traveltime': int(input("Travel time (1-4): ")),
    'studytime': int(input("Study time (1-4): ")),
    'failures': int(input("Number of past class failures: ")),
    'freetime': int(input("Free time after school (1-5): ")),
    'goout': int(input("Going out with friends (1-5): ")),
    'health': int(input("Health status (1-5): ")),
    'absences': int(input("Number of absences: "))
}

sample = X.mean().to_dict()
sample.update(input_data)

input_df = pd.DataFrame([sample])
prediction = model.predict(input_df)[0]
print("\nPrediction: ", "PASS" if prediction == 1 else "FAIL")
```

Output:

1. Travel Time (1–4)

How long it takes the student to get to school

- 1 = <15 minutes
- 2 = 15 to 30 minutes
- 3 = 30 minutes to 1 hour
- 4 = >1 hour

2. Study Time (1–4)

Weekly study time

- 1 = <2 hours
- 2 = 2 to 5 hours
- 3 = 5 to 10 hours
- 4 = >10 hours

3. Free Time After School (1–5)

Student's self-reported amount of free time

- 1 = Very low
- 2 = Low
- 3 = Medium
- 4 = High
- 5 = Very high

4. Going Out with Friends (1–5)

How often the student goes out with friends

- 1 = Rarely
- 2 = Sometimes
- 3 = Often
- 4 = Frequently
- 5 = Very frequently

5. Health Status (1–5)

Student's current health condition

- 1 = Very bad
- 2 = Bad
- 3 = Average
- 4 = Good

- 5 = Very good

```
--- Enter your details to predict pass/fail ---
Age:  22
Travel time (1-4):  2
Study time (1-4):  2
Number of past class failures:  0
Free time after school (1-5):  2
Going out with friends (1-5):  2
Health status (1-5):  3
Number of absences:  5

Prediction:  PASS
```

Analysis:

1. The project successfully transforms a student performance dataset into a binary classification problem by defining a pass column based on the final grade.
2. Data preprocessing is handled well, with categorical variables encoded into numeric form for compatibility with the Random Forest model.
3. Splitting the data into training and testing sets ensures unbiased evaluation of model performance.
4. The Random Forest classifier effectively learns from the data and produces solid accuracy and detailed classification metrics.
5. Visualization of the confusion matrix helps interpret model strengths and areas where it misclassifies.
6. The interactive user input section demonstrates practical application by allowing custom predictions based on user-provided feature values.

Conclusion

The code presents a complete and practical machine learning workflow—from data loading and preprocessing to model training, evaluation, and real-time prediction. The Random Forest model is a strong choice for this tabular data and yields interpretable and accurate results. With minor improvements like better encoding management and input validation, this pipeline could be robustly used to assist in predicting student success and tailoring educational support.