

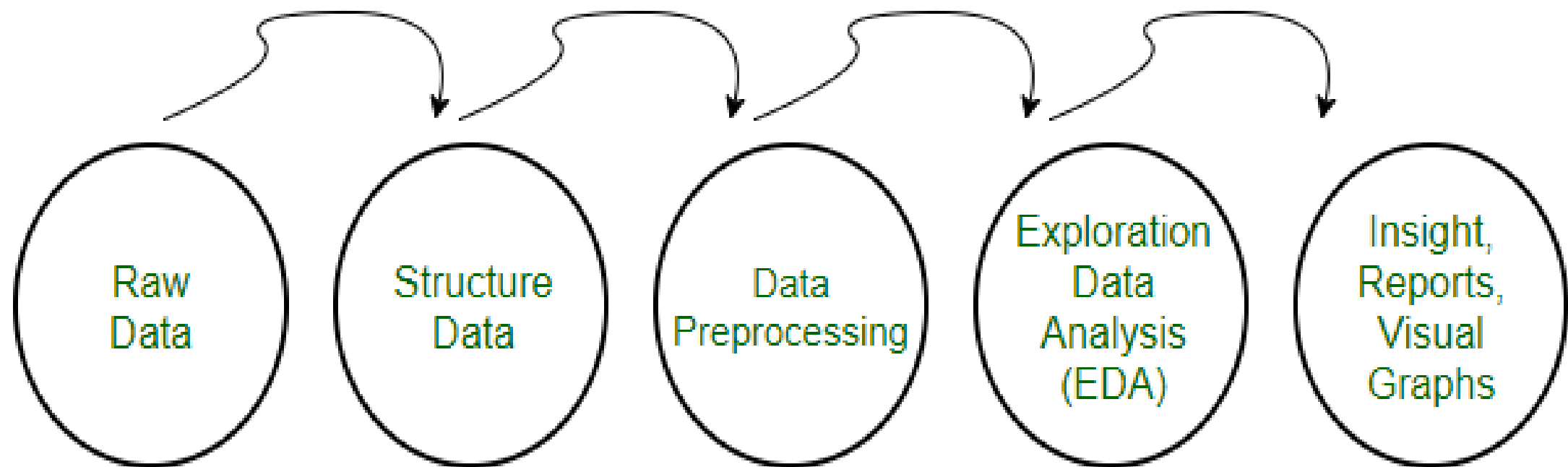
# Stock price prediction

# *Definition:*

**Stock Price Prediction** is the task of forecasting future stock prices based on historical data and various market indicators. It involves using statistical models and machine learning algorithms to analyze financial data and make predictions about the future performance of a stock.

# Data preprocessing

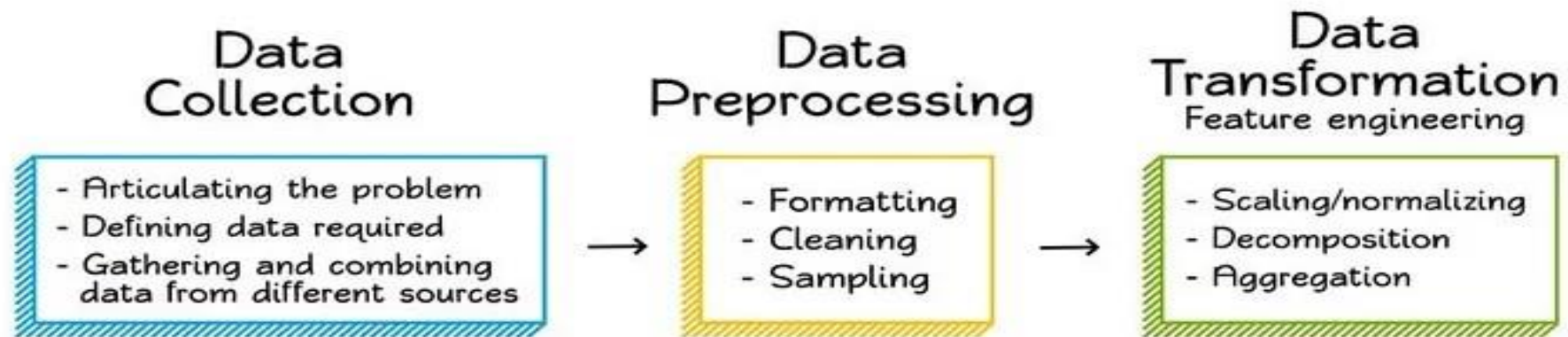
○ Data Preprocessing is the process of making data suitable for use while training a machine learning model. The dataset initially provided for training might not be in a ready-to-use state, for e.g. it might not be formatted properly, or **may contain missing or null values**.



# *Data Preprocessing methods using Python*

- Importing the libraries
- Importing the Dataset
- Handling of Missing Data
- Handling of Categorical Data
- Splitting the dataset into training and testing datasets
- Feature Scaling

# Data Preparation Process



# Step 1: Importing the libraries

- In the beginning, we'll import three basic libraries which are very common in machine learning and will be used every time you train a model
- NumPy:- it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier
- matplotlib:- this library helps in plotting graphs and charts, which are very useful while showing the result of your model
- Pandas:- pandas allows us to import our dataset and also creates a matrix of features containing the dependent and independent variable.



localhost:8889/notebooks/Untitled4.ipynb?kernel\_name=python3

jupyter Untitled4 Last Checkpoint: 32 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [7]: data['day']=data['Date'].apply(lambda x:x.split('-')[0])
data['month']=data['Date'].apply(lambda x:x.split('-')[1])
data['year']=data['Date'].apply(lambda x:x.split('-')[2])
data
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume	day	month	year
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800	1986	03	13
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000	1986	03	14
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200	1986	03	17
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400	1986	03	18
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400	1986	03	19
...	...	...	...	...	...	...	...	...	...	...
8520	2019-12-31	156.770004	157.770004	156.449997	157.699997	157.699997	18369400	2019	12	31
8521	2020-01-02	158.779999	160.729996	158.330002	160.619995	160.619995	22622100	2020	01	02
8522	2020-01-03	158.320007	159.949997	158.059998	158.619995	158.619995	21116200	2020	01	03
8523	2020-01-06	157.080002	159.100006	156.509995	159.029999	159.029999	20813700	2020	01	06
8524	2020-01-07	159.320007	159.669998	157.330002	157.580002	157.580002	18017762	2020	01	07

8525 rows x 10 columns

Windows Taskbar: Type here to search, 32°C, 2:47 PM 10/17/2023



## Step2: Importing the dataset

- There are two types of variables:
- Independent variable
- Dependent variable
- The **independent variable** is the columns that we are going to use to predict the **dependent variable**, or in other words, the independent variable **affects** the dependent variable

localhost:8888/notebooks/Untitled2.ipynb?kernel\_name=python3

jupyter Untitled2 Last Checkpoint: 17 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
stock_price_df=pd.read_csv('C:/Users/Dhinesh PC/Downloads/MSFT.csv')
stock_price_df.shape

Out[4]: (8525, 7)
```

```
In [5]: stock_price_df.columns

Out[5]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
In [6]: stock_price_df.head(8000)
```

```
Out[6]:
```

	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
...	...	...	...	...	...	...	...
7995	2017-11-28	84.070000	85.059998	84.019997	84.879997	82.219917	21926000
7996	2017-11-29	84.700000	84.810000	82.180000	82.220000	80.720155	27281100



Type here to search



localhost:8888/notebooks/Untitled2.ipynb?kernel\_name=python3

jupyter Untitled2 Last Checkpoint: 17 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [5]: `stock_price_df.columns`

Out[5]: `Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')`

In [6]: `stock_price_df.head(8000)`

Out[6]:

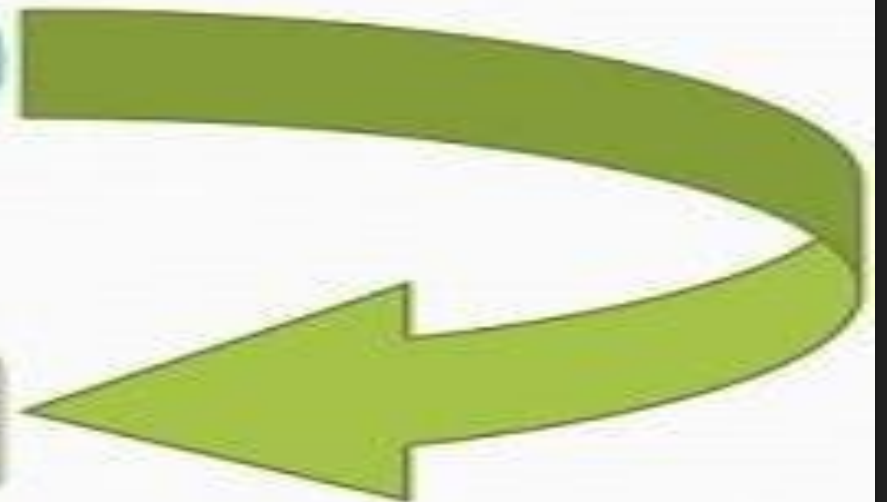
	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400
...	...	...	...	...	...	...	...
7995	2017-11-28	84.070000	85.059998	84.019997	84.879997	82.219917	21926000
7996	2017-11-29	84.709999	84.919998	83.180000	83.339996	80.728165	27381100
7997	2017-11-30	83.510002	84.519997	83.339996	84.169998	81.532158	33054600
7998	2017-12-01	83.599998	84.809998	83.220001	84.260002	81.619354	29532100
7999	2017-12-04	84.419998	84.430000	80.699997	81.080002	78.539001	39094900

8000 rows x 7 columns

Independent  
Variable

Affects  
**CHANGE**  
in the

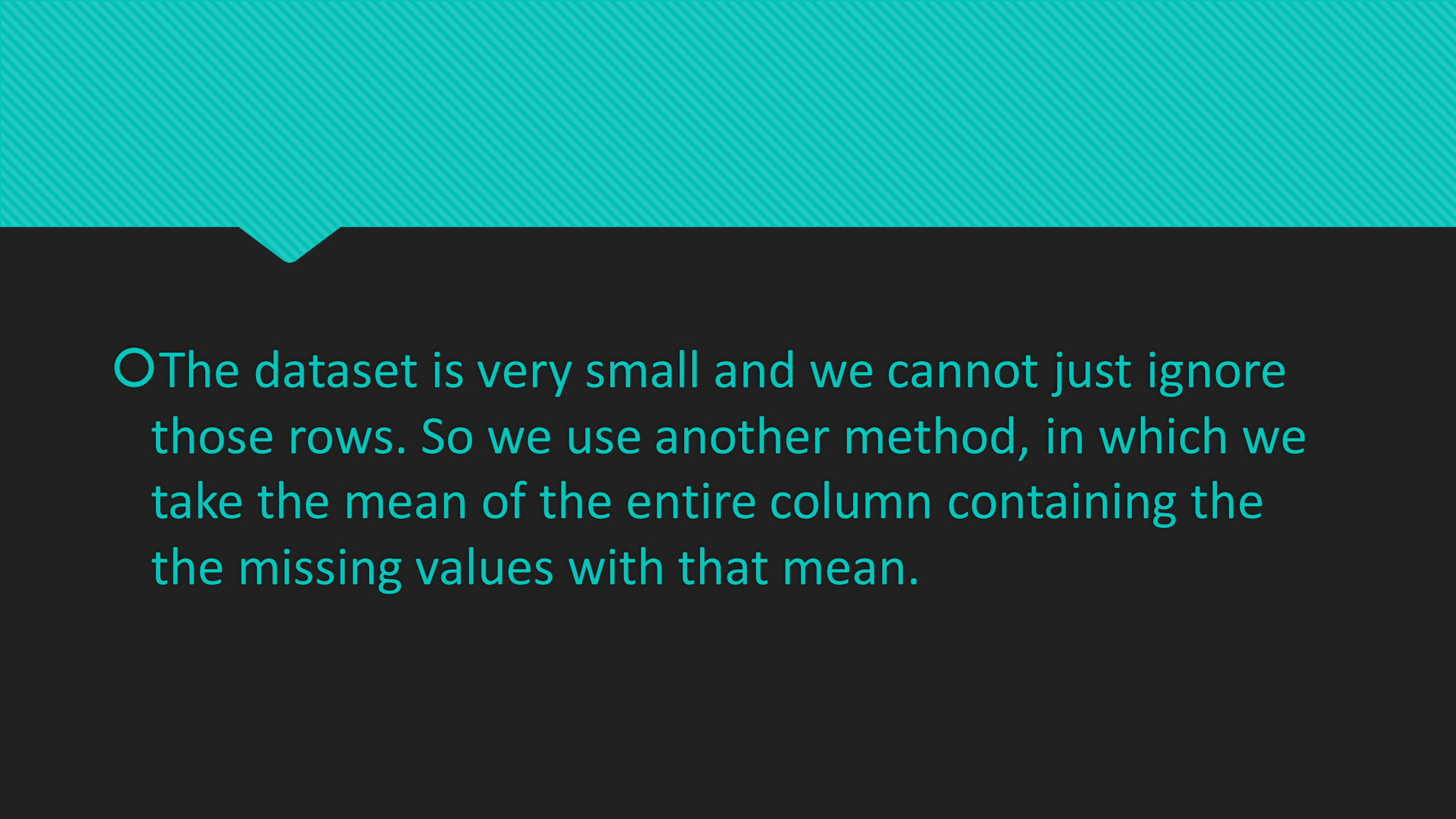
Dependent  
Variable



## Step3: Handling the missing data

○ To handle missing values, one of them is to ignore them and delete the entire entry/row, this is commonly done in datasets containing a very large number of entries, where the missing values only constitute 0.1% of the total data. Thus they affect the model negligibly and can be removed.





○The dataset is very small and we cannot just ignore those rows. So we use another method, in which we take the mean of the entire column containing the missing values with that mean.

# Step4: Encoding categorical variable

○ We have two categorical columns, the *country* column, and the *purchased* column.

## ○ OneHot Encoding

○ In the *country* column, we have three different categories: France, Germany, Spain. We can simply label France as 0, Germany as 1, and Spain as 2 but doing this might lead our machine learning model to interpret that there is some correlation between these numbers and the outcome.



localhost:8888/notebooks/Untitled2.ipynb?kernel\_name=python3#

Import favorites | Gmail | YouTube | Maps | News | Other favorites

jupyter Untitled2 Last Checkpoint: 37 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [33]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[0])],remainder='passthrough')
data=pd.DataFrame(ct.fit_transform(data))
data
```

Out[33]:

	0
0	(0, 0)\t1.0\n (0, 8525)\t0.088542\n (0, 85...
1	(0, 1)\t1.0\n (0, 8525)\t0.097222\n (0, 85...
2	(0, 2)\t1.0\n (0, 8525)\t0.100694\n (0, 85...
3	(0, 3)\t1.0\n (0, 8525)\t0.102431\n (0, 85...
4	(0, 4)\t1.0\n (0, 8525)\t0.099826\n (0, 85...
...	...
8520	(0, 8520)\t1.0\n (0, 8525)\t156.770004\n (...)
8521	(0, 8521)\t1.0\n (0, 8525)\t158.779999\n (...)
8522	(0, 8522)\t1.0\n (0, 8525)\t158.320007\n (...)
8523	(0, 8523)\t1.0\n (0, 8525)\t157.080002\n (...)
8524	(0, 8524)\t1.0\n (0, 8525)\t159.320007\n (...)

8525 rows x 1 columns

In [ ]:

Windows Taskbar: Type here to search | 27°C | 12:34 PM 10/17/2023

localhost:8889/notebooks/Untitled4.ipynb?kernel\_name=python3

jupyter Untitled4 Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [33]: import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Sample data
data = pd.read_csv('C:/Users/Dhinesh PC/Documents/MSFT.csv')
data
# Label Encoding (for ordinal data)
label_encoder = LabelEncoder()
data['Category_LabelEncoded'] = label_encoder.fit_transform(data['Adj Close'])

# One-Hot Encoding (for nominal data)
one_hot_encoder = OneHotEncoder(sparse=False)
encoded_categories = one_hot_encoder.fit_transform(data[['Adj Close']])
encoded_categories_df = pd.DataFrame(encoded_categories, columns=one_hot_encoder.get_feature_names(['Adj Close']))

print(data)
print(encoded_categories_df)
```

	Volume	Category_LabelEncoded
0	1031788800	8
1	308160000	13
2	133171200	15
3	67766400	11
4	47894400	9
...	...	...
8520	18369400	6422
8521	22622100	6427
8522	21116200	6423
8523	20813700	6426

Type here to search 32°C 3:22 PM 10/17/2023

localhost:8889/notebooks/Untitled4.ipynb?kernel\_name=python3

Import favorites | Gmail | YouTube | Maps | News | Other favorites

jupyter Untitled4 Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3 (ipykernel)

In [33]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Sample data
data = pd.read_csv('C:/Users/Dhinesh PC/Documents/MSFT.csv')
data
# Label Encoding (for ordinal data)
label_encoder = LabelEncoder()
data['Category_LabelEncoded'] = label_encoder.fit_transform(data['Adj Close'])

# One-Hot Encoding (for nominal data)
one_hot_encoder = OneHotEncoder(sparse=False)
encoded_categories = one_hot_encoder.fit_transform(data[['Adj Close']])
encoded_categories_df = pd.DataFrame(encoded_categories, columns=one_hot_encoder.get_feature_names(['Adj Close']))

print(data)
print(encoded_categories_df)
```

	Adj Close_0.061711	Adj Close_0.06199	Adj Close_0.062549
0	0.0	0.0	1.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
...	...	...	...
8520	0.0	0.0	0.0
8521	0.0	0.0	0.0
8522	0.0	0.0	0.0
8523	0.0	0.0	0.0
8524	0.0	0.0	0.0

Type here to search

32°C 3:22 PM 10/17/2023

## ○ Label Encoding

○ In the last column, i.e. the purchased column, the data is in binary form meaning that there are only two outcomes either Yes or No. Therefore here we need to perform Label Encoding.

## Step5: Normalizing the dataset

○ Feature scaling is bringing all of the features on the dataset to the same scale, this is necessary while training a machine learning model because in some cases the **dominant features become so dominant that the other ordinary features are not even considered by the model.**



## Step6: Splitting the dataset

- Before we begin training our model there is one final step to go, which is splitting of the testing and training dataset. In machine learning, a **larger part** of the dataset is used to train the model, and a small part is used to test the trained model for finding out the accuracy and the efficiency of the model.
- Now before we begin splitting the dataset we need to separate the dependent and independent variables

- The last (*purchased*) column is the dependent variable and the rest are independent variables, so we'll store the dependent variable in 'y' and the independent variables in 'X'.
- Another important part we need to remember is that while training the model accepts data as arrays so it is necessary that **we convert the data to arrays**. We do that while separating the dependent and independent variables by adding *.values* while storing data in 'X' and 'y'.



# Transformation variable:

- "Transformation variable" is not a standard concept in Python. It's possible that you're referring to a variable used in transformations or operations on data.
- In Python, a variable is a name that refers to a value. You can use variables to store and manipulate data. Transformation variables, in this context, could be any variable you use to hold intermediate or final results during a transformation

localhost:8889/notebooks/Untitled4.ipynb?kernel\_name=python3

jupyter Untitled4 Last Checkpoint: 32 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [7]: data['day']=data['Date'].apply(lambda x:x.split('-')[0])
data['month']=data['Date'].apply(lambda x:x.split('-')[1])
data['year']=data['Date'].apply(lambda x:x.split('-')[2])
data
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume	day	month	year
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800	1986	03	13
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000	1986	03	14
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200	1986	03	17
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400	1986	03	18
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400	1986	03	19
...	...	...	...	...	...	...	...	...	...	...
8520	2019-12-31	156.770004	157.770004	156.449997	157.699997	157.699997	18369400	2019	12	31
8521	2020-01-02	158.779999	160.729996	158.330002	160.619995	160.619995	22622100	2020	01	02
8522	2020-01-03	158.320007	159.949997	158.059998	158.619995	158.619995	21116200	2020	01	03
8523	2020-01-06	157.080002	159.100006	156.509995	159.029999	159.029999	20813700	2020	01	06
8524	2020-01-07	159.320007	159.669998	157.330002	157.580002	157.580002	18017762	2020	01	07

8525 rows x 10 columns

localhost:8889/notebooks/IBM.ipynb#

jupyter IBM Last Checkpoint: 3 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [39]: import numpy as np
data['Log_Transformed'] = np.log(data['Volume'])
data['Sqrt_Transformed'] = np.sqrt(data['Volume'])
from scipy import stats
data['BoxCox_Transformed'], _ = stats.boxcox(data['Volume'])
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data['Standardized'] = scaler.fit_transform(data[['Volume']])
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data['Normalized'] = scaler.fit_transform(data[['Volume']])
print(data)
```

	Date	Open	High	Low	Close	Adj Close
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224
4	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	Category_LabelEncoded	Log_Transformed	Sqrt_Transformed
0	1031788800	8	20.754560	32121.469456
1	308160000	13	19.546130	17554.486606
2	133171200	15	18.707146	11539.982669
3	67766400	11	18.021577	9222.024085

Type here to search

32°C 3:31 PM 10/17/2023

localhost:8889/notebooks/IBM.ipynb#

jupyter IBM Last Checkpoint: 3 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Notebook saved Trusted Python 3 (ipykernel)

```
In [39]: import numpy as np
data['Log_Transformed'] = np.log(data['Volume'])
data['Sqrt_Transformed'] = np.sqrt(data['Volume'])
from scipy import stats
data['BoxCox_Transformed'], _ = stats.boxcox(data['Volume'])
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data['Standardized'] = scaler.fit_transform(data[['Volume']])
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data['Normalized'] = scaler.fit_transform(data[['Volume']])
print(data)
```

	Date	Open	High	Low	Close	Adj Close
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224
4	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	Category_LabelEncoded	Log_Transformed	Sqrt_Transformed		
0	1031788800	8	20.754560	32121.469456		
1	308160000	13	19.546130	17554.486606		
2	133171200	15	18.707146	11539.982669		
3	67766400	11	18.021577	9222.924095		



Type here to search



3:31 PM

10/17/2023





localhost:8889/notebooks/IBM.ipynb#

Import favoritesGmailYouTubeMapsNewsOther favorites

jupyter IBM Last Checkpoint: 3 minutes ago (autosaved)Logout

FileEditViewInsertCellKernelWidgetsHelpTrustedPython 3 (ipykernel)

RunCode

	Volume	Category_LabelEncoded	Log_Transformed	Sqrt_Transformed	\
0	1031788800	8	20.754560	32121.469456	
1	308160000	13	19.546130	17554.486606	
2	133171200	15	18.707146	11539.982669	
3	67766400	11	18.031577	8232.034985	
4	47894400	9	17.684509	6920.578011	
...	...	...	...	...	
8520	18369400	6422	16.726197	4285.953803	
8521	22622100	6427	16.934438	4756.269547	
8522	21116200	6423	16.865551	4595.236664	
8523	20813700	6426	16.851122	4562.203415	
8524	18017762	6420	16.706869	4244.733443	

	BoxCox_Transformed	Standardized	Normalized
0	157.341204	24.963577	1.000000
1	129.177315	6.366058	0.297096
2	112.525895	1.868783	0.127119
3	100.616460	0.187856	0.063588
4	94.969637	-0.322861	0.044285
...	...	...	...
8520	80.877527	-1.081664	0.015605
8521	83.762953	-0.972368	0.019736
8522	82.798038	-1.011071	0.018273
8523	82.597238	-1.018845	0.017980
8524	80.614441	-1.090702	0.015264

[8525 rows x 13 columns]

In [ ]:

Type here to search

32°C

3:32 PM 10/17/2023

# Outliers detection and treatment

- Detecting and treating outliers in Python typically involves using statistical methods and data visualization techniques to identify data points that significantly deviate from the rest of the dataset. Once identified, you can choose to handle outliers in various ways, such as removing them, transforming them, or imputing them

localhost:8889/notebooks/IBM.ipynb#

jupyter IBM Last Checkpoint: 11 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [41]: import pandas as pd

data=pd.read_csv('C:/Users/Dhinesh PC/Documents/MSFT.csv')
Q1 = data['High'].quantile(0.25)
Q3 = data['Low'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = data[(data['High'] < lower_bound) | (data['Low'] > upper_bound)]
print(outliers)
```

	Date	Open	High	Low	Close	Adj Close
7974	2017-10-27	84.370003	86.199997	83.610001	83.809998	80.777763
7975	2017-10-30	83.699997	84.330002	83.110001	83.889999	80.854881
7976	2017-10-31	84.360001	84.360001	83.110001	83.180000	80.170578
7977	2017-11-01	83.680000	83.760002	82.879997	83.180000	80.170578
7978	2017-11-02	83.349998	84.459999	83.120003	84.050003	81.009094
...	...	...	...	...	...	...
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					
7974	71066700					



Type here to search



32°C



3:39 PM  
10/17/2023





localhost:8889/notebooks/IBM.ipynb#

jupyter IBM Last Checkpoint: 11 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
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
7974 71066700
7975 31756700
7976 27086600
7977 22307400
7978 23992900
...
8520 18369400
8521 22622100
8522 21116200
8523 20813700
8524 18017762

[551 rows x 7 columns]
```

In [ ]:

32°C 3:39 PM 10/17/2023

# Conclusion

○ Evaluate the model's performance on the testing data using appropriate metrics (e.g., accuracy, mean squared error, etc.). This step helps you understand how well your model is likely to perform on new, unseen data