2347138_P1

June 26, 2024

[ ]:
```
import pandas as pd

Nifty_data="E:/4TH_sem/AI_ML/STOCKS.csv"

#taking a as dataframe
a=pd.read_csv(Nifty_data)
a.head()
```

[ ]:

	Date	Open	High	Low	Close	Shares Traded	\
0	26-DEC-2023	21365.20	21477.15	21329.45	21441.35	219467748.0	
1	27-DEC-2023	21497.65	21675.75	21495.80	21654.75	256542963.0	
2	28-DEC-2023	21715.00	21801.45	21678.00	21778.70	393080755.0	
3	29-DEC-2023	21737.65	21770.30	21676.90	21731.40	270922276.0	
4	01-JAN-2024	21727.75	21834.35	21680.85	21741.90	153995217.0	

	Turnover (Cr)
0	20081.33
1	23059.25
2	35031.00
3	23697.88
4	14184.09

[ ]:
```
a.describe()
```

[ ]:

	Open	High	Low	Close	Shares Traded	\
count	124.000000	124.000000	124.000000	124.000000	1.230000e+02	
mean	22272.602419	22372.966935	22141.296371	22262.070968	3.336844e+08	
std	557.911182	547.077688	558.735362	554.136642	1.180094e+08	
min	21185.250000	21459.000000	21137.200000	21238.800000	1.906457e+07	
25%	21839.225000	21953.237500	21713.912500	21839.812500	2.670421e+08	
50%	22213.225000	22302.650000	22053.700000	22199.450000	3.172671e+08	
75%	22567.912500	22635.562500	22427.637500	22515.837500	3.742559e+08	
max	23661.150000	23754.150000	23562.050000	23721.300000	1.006105e+09	

	Turnover (Cr)
count	123.000000
mean	32573.542358
std	11896.672621

```
min        1572.770000
25%       25503.345000
50%       30291.600000
75%       37609.590000
max       93786.440000
```

[ ]:
```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 124 entries, 0 to 123
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             124 non-null    object
 1   Open             124 non-null    float64
 2   High             124 non-null    float64
 3   Low              124 non-null    float64
 4   Close            124 non-null    float64
 5   Shares Traded    123 non-null    float64
 6   Turnover (  Cr)  123 non-null    float64
dtypes: float64(6), object(1)
memory usage: 6.9+ KB
```

[ ]:
```
pip install mathplotlib
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Could not find a version that satisfies the requirement mathplotlib (from
versions: none)
ERROR: No matching distribution found for mathplotlib
```

[ ]:
```
pip install pandas matplotlib seaborn mplfinance
```

```
Requirement already satisfied: pandas in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(2.2.1)
Requirement already satisfied: matplotlib in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(3.8.4)
Requirement already satisfied: seaborn in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(0.13.2)
Requirement already satisfied: mplfinance in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(0.12.10b0)
Requirement already satisfied: numpy<2,>=1.26.0 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
```

```
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from matplotlib) (3.1.2)
Requirement already satisfied: six>=1.5 in
c:\users\pratham.m\appdata\local\programs\python\python312\lib\site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
print(a.columns)
```

```
Index(['Date ', 'Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ',
       'Turnover (  Cr)'],
      dtype='object')
```

[ ]:
```
a.rename(columns={'Date ': 'Date', ' Turnover (  Cr)': 'Turnover'},␣
 ↪inplace=True)
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

sns.set(style="whitegrid")
```

```
a['Date']=pd.to_datetime(a['Date'])
a.set_index('Date',inplace=True)
```

```
C:\Users\Pratham.m\AppData\Local\Temp\ipykernel_18020\1787886723.py:8:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  a['Date']=pd.to_datetime(a['Date'])
```

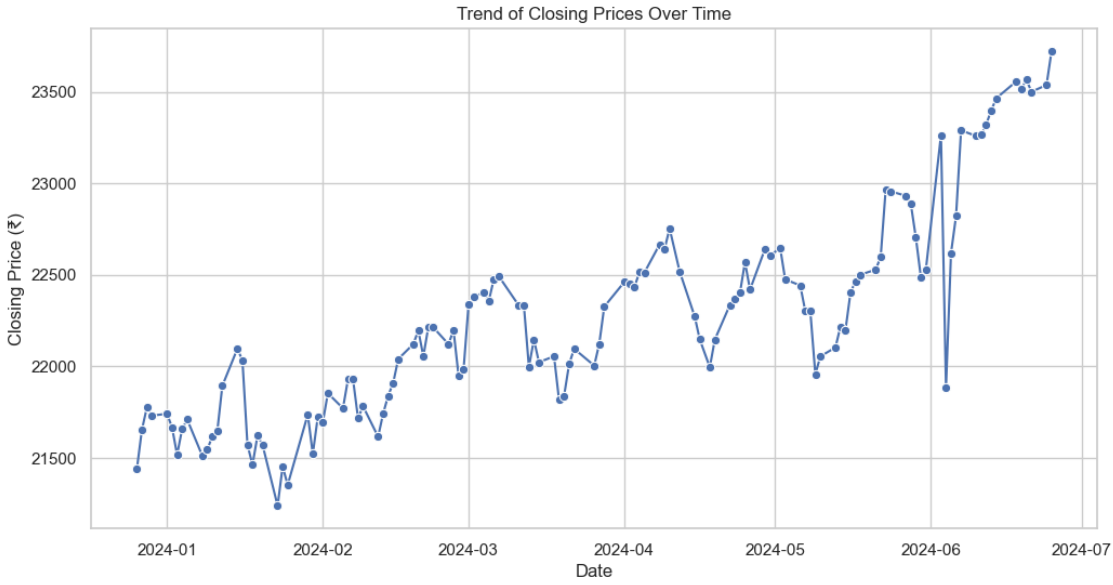
[ ]:
```
# Check column names
print(a.columns)

# Display the first few rows
print(a.head())
```

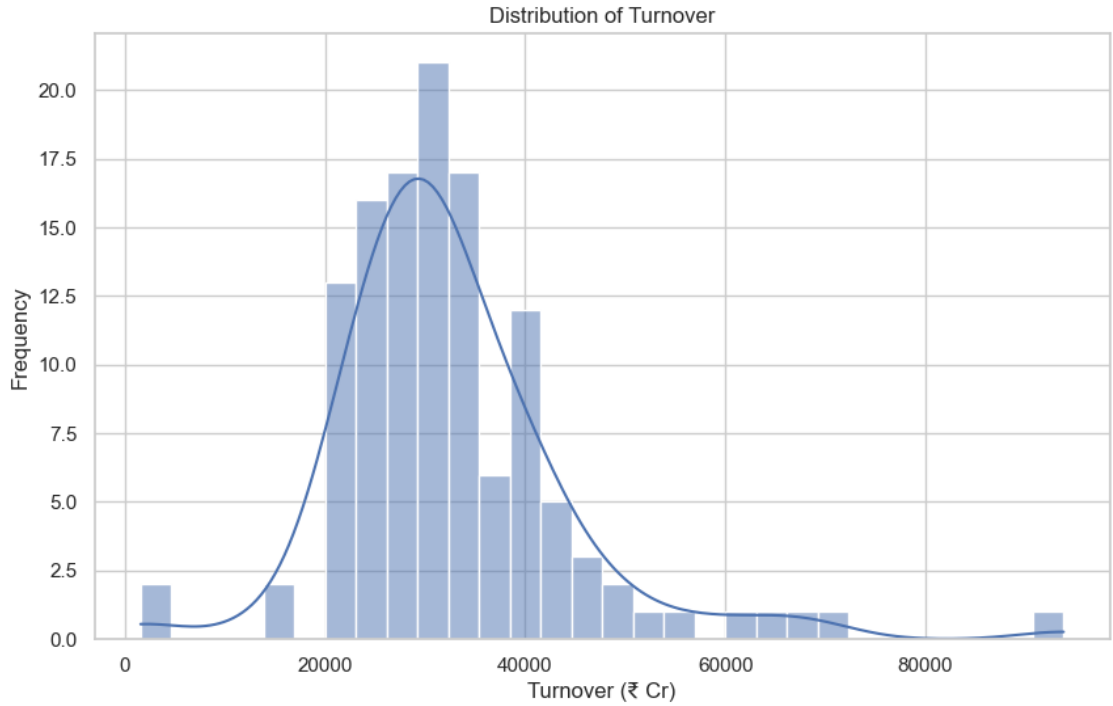
```
Index(['Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ',
       'Turnover (  Cr)'],
      dtype='object')
                Open      High       Low     Close   Shares Traded   \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35     219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75     256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70     393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40     270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90     153995217.0

            Turnover (  Cr)
Date
2023-12-26         20081.33
2023-12-27         23059.25
2023-12-28         35031.00
2023-12-29         23697.88
2024-01-01         14184.09
```

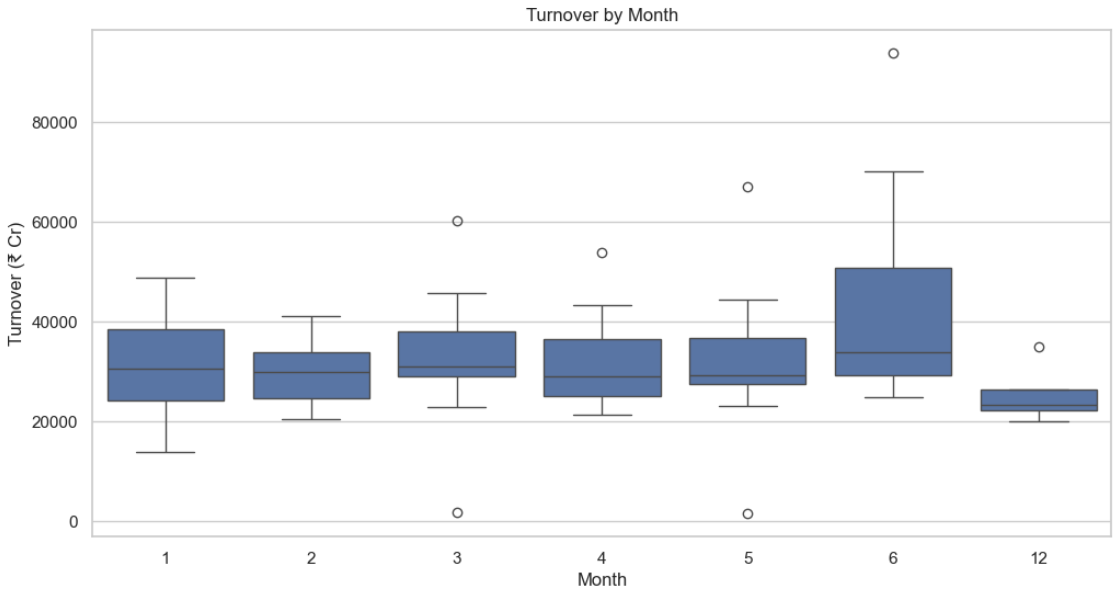
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the CSV file
Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

# Clean column names by stripping any leading/trailing spaces
a.columns = a.columns.str.strip()

# Convert 'Date' column to datetime
```

```
a['Date'] = pd.to_datetime(a['Date'])

# Set 'Date' as the index
a.set_index('Date', inplace=True)

# Adding 'Month' column for monthly analysis
a['Month'] = a.index.month

# Setting style for the plots
sns.set(style="whitegrid")

# 1. Line Plot: Stock Prices Over Time
plt.figure(figsize=(12, 6))
sns.lineplot(x=a.index, y=a['Close'], marker='o')
plt.title('Trend of Closing Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price ( )')
plt.show()

# 2. Histogram: Distribution of Turnover
plt.figure(figsize=(10, 6))
sns.histplot(a['Turnover (  Cr)'], bins=30, kde=True)
plt.title('Distribution of Turnover')
plt.xlabel('Turnover (  Cr)')
plt.ylabel('Frequency')
plt.show()

# 3. Scatter Plot: Opening vs. Closing Prices
plt.figure(figsize=(10, 6))
sns.scatterplot(x=a['Open'], y=a['Close'])
plt.title('Opening Price vs. Closing Price')
plt.xlabel('Opening Price ( )')
plt.ylabel('Closing Price ( )')
plt.show()

# 4. Box Plot: Turnover by Month
plt.figure(figsize=(12, 6))
sns.boxplot(x='Month', y='Turnover (  Cr)', data=a)
plt.title('Turnover by Month')
plt.xlabel('Month')
plt.ylabel('Turnover (  Cr)')
plt.show()
```

```
C:\Users\Pratham.m\AppData\Local\Temp\ipykernel_18020\347481218.py:13:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  a['Date'] = pd.to_datetime(a['Date'])
```


Trend of Closing Prices Over Time
Closing Price (₹)
23500
23000
22500
22000
21500
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
Date


Distribution of Turnover
Frequency
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20000
40000
60000
80000
Turnover (₹ Cr)



Understanding data

2.1 Display number of samples (row) and number of attributes(columns) in your data set

2.2 Display all columns names in your data set

2.3 Display the structure of the data frame

2.4 Display the statistical information about the dataset

2.5 Display no of samples based on particular coloumn

[ ]:
```
row=len(a)
print("Number of samples row",row)

columns=len(a.columns)
print("Number of attributes columns",columns)
```

```
Number of samples row 124
Number of attributes columns 7
```

[ ]:
```
columns=a.columns.tolist()
for column in columns:
    print(column)
```

```
Open
High
Low
Close
Shares Traded
Turnover (  Cr)
Month
```

[ ]:
```
print("DataFrame Structure",a.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 124 entries, 2023-12-26 to 2024-06-25
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Open             124 non-null    float64
 1   High             124 non-null    float64
 2   Low              124 non-null    float64
 3   Close            124 non-null    float64
 4   Shares Traded    123 non-null    float64
 5   Turnover (  Cr)  123 non-null    float64
 6   Month            124 non-null    int32
dtypes: float64(6), int32(1)
memory usage: 7.3 KB
DataFrame Structure None
```

[ ]:
```
# Display statistical information about the dataset
print("Statistical information:")
print(a.describe())
```

```
Statistical information:
                Open          High           Low         Close  Shares Traded  \
```

```
count    124.000000    124.000000    124.000000    124.000000   1.230000e+02
mean   22272.602419  22372.966935  22141.296371  22262.070968   3.336844e+08
std      557.911182    547.077688    558.735362    554.136642   1.180094e+08
min    21185.250000  21459.000000  21137.200000  21238.800000   1.906457e+07
25%    21839.225000  21953.237500  21713.912500  21839.812500   2.670421e+08
50%    22213.225000  22302.650000  22053.700000  22199.450000   3.172671e+08
75%    22567.912500  22635.562500  22427.637500  22515.837500   3.742559e+08
max    23661.150000  23754.150000  23562.050000  23721.300000   1.006105e+09

       Turnover (  Cr)       Month
count       123.000000  124.000000
mean      32573.542358    3.669355
std       11896.672621    2.262173
min        1572.770000    1.000000
25%       25503.345000    2.000000
50%       30291.600000    3.500000
75%       37609.590000    5.000000
max       93786.440000   12.000000
```

[ ]:
```
# Display number of samples based on a particular column
samples_per_month = a['Month'].value_counts()
print("Number of samples based on Month:")
print(samples_per_month)
```

```
Number of samples based on Month:
Month
1     22
5     22
2     21
4     20
3     19
6     16
12     4
Name: count, dtype: int64
```

Extracting independent variables and dependent variables in your dataset and interpret the difference between independent and dependent variables.

[ ]:
```
# Assuming 'a' is your DataFrame with cleaned data
# Independent variables
independent_vars = a[['Open', 'High', 'Low', 'Shares Traded', 'Turnover (␣
↪Cr)']]

# Dependent variable (assuming 'Close' is the dependent variable)
dependent_var = a['Close']

# Print shape and first few rows for verification
print("Independent Variables:")
```

```
print(independent_vars.head())
print(f"Shape: {independent_vars.shape}\n")

print("Dependent Variable:")
print(dependent_var.head())
print(f"Shape: {dependent_var.shape}")
```

```
Independent Variables:
                Open      High       Low  Shares Traded  Turnover (  Cr)
Date
2023-12-26  21365.20  21477.15  21329.45    219467748.0          20081.33
2023-12-27  21497.65  21675.75  21495.80    256542963.0          23059.25
2023-12-28  21715.00  21801.45  21678.00    393080755.0          35031.00
2023-12-29  21737.65  21770.30  21676.90    270922276.0          23697.88
2024-01-01  21727.75  21834.35  21680.85    153995217.0          14184.09
Shape: (124, 5)

Dependent Variable:
Date
2023-12-26    21441.35
2023-12-27    21654.75
2023-12-28    21778.70
2023-12-29    21731.40
2024-01-01    21741.90
Name: Close, dtype: float64
Shape: (124,)
```

Find unique values from a column in a Data frame & count the unique values from a column in a Data frame.

[ ]:
```
unique_values = a['Month'].unique()

print("Unique values in the 'Month' column:")
print(unique_values)
```

```
Unique values in the 'Month' column:
[12  1  2  3  4  5  6]
```

[ ]:
```
# Counting unique values in the 'Month' column
value_counts = a['Month'].value_counts()

print("Count of unique values in the 'Month' column:")
print(value_counts)
```

```
Count of unique values in the 'Month' column:
Month
1     22
5     22
2     21
```

```
4     20
3     19
6     16
12     4
Name: count, dtype: int64
```

Find the missing values and non-missing values in your dataset.

[ ]:
```
# Check for missing values in the entire DataFrame
missing_values = a.isnull().sum()

print("Missing values in the dataset:")
print(missing_values)
```

```
Missing values in the dataset:
Open                0
High                0
Low                 0
Close               0
Shares Traded       1
Turnover (  Cr)     1
Month               0
dtype: int64
```

[ ]:
```
# Check for non-missing values in the entire DataFrame
non_missing_values = a.notnull().sum()

print("\nNon-missing values in the dataset:")
print(non_missing_values)
```

```
Non-missing values in the dataset:
Open                124
High                124
Low                 124
Close               124
Shares Traded       123
Turnover (  Cr)     123
Month               124
dtype: int64
```

[ ]:
```
# Drop rows with any NaN values
a_dropna = a.dropna()

# Print the cleaned DataFrame
print("DataFrame after dropping rows with any NaN values:")
print(a_dropna)
```

```
DataFrame after dropping rows with any NaN values:
                Open      High       Low     Close  Shares Traded  \
```

```
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…              …         …         …         …             …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover (  Cr)  Month
Date
2023-12-26         20081.33     12
2023-12-27         23059.25     12
2023-12-28         35031.00     12
2023-12-29         23697.88     12
2024-01-01         14184.09      1
…                       …    …
2024-06-19         39430.39      6
2024-06-20         33390.07      6
2024-06-21         70062.97      6
2024-06-24         24862.37      6
2024-06-25         34263.36      6

[123 rows x 7 columns]
```

[ ]:
```
# Replace NaN values with mean of the column
a_filled_mean = a.fillna(a.mean())

# Print the DataFrame with NaN replaced
print("DataFrame after replacing NaN with mean:")
print(a_filled_mean)
```

```
DataFrame after replacing NaN with mean:
                Open      High       Low     Close  Shares Traded  \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…              …         …         …         …             …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
```

```
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover (  Cr)  Month
Date
2023-12-26         20081.33     12
2023-12-27         23059.25     12
2023-12-28         35031.00     12
2023-12-29         23697.88     12
2024-01-01         14184.09      1
…                       …    …
2024-06-19         39430.39      6
2024-06-20         33390.07      6
2024-06-21         70062.97      6
2024-06-24         24862.37      6
2024-06-25         34263.36      6

[124 rows x 7 columns]
```

[ ]:
```
# Check for missing values in the entire DataFrame after replacing NaN with mean
missing_values_after_fillna = a_filled_mean.isnull().sum()

print("Missing values in the dataset after replacing NaN with mean:")
print(missing_values_after_fillna)
```

```
Missing values in the dataset after replacing NaN with mean:
Open               0
High               0
Low                0
Close              0
Shares Traded      0
Turnover (  Cr)    0
Month              0
dtype: int64
```

[ ]:
```
# Consider first value as unique and rest as duplicates
a_unique_first = a.drop_duplicates(keep='first')
a_unique_first
```

[ ]:
```
                Open      High       Low     Close  Shares Traded  \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…                …         …         …         …             …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
```

```
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover ( Cr)  Month
Date
2023-12-26         20081.33     12
2023-12-27         23059.25     12
2023-12-28         35031.00     12
2023-12-29         23697.88     12
2024-01-01         14184.09      1
…                       …    …
2024-06-19         39430.39      6
2024-06-20         33390.07      6
2024-06-21         70062.97      6
2024-06-24         24862.37      6
2024-06-25         34263.36      6

[124 rows x 7 columns]
```

[ ]:
```
# Consider last value as unique and rest as duplicates
a_unique_last = a.drop_duplicates(keep='last')
a_unique_last
```

[ ]:
```
                Open      High       Low     Close  Shares Traded  \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…              …       …       …       …              …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover ( Cr)  Month
Date
2023-12-26         20081.33     12
2023-12-27         23059.25     12
2023-12-28         35031.00     12
2023-12-29         23697.88     12
2024-01-01         14184.09      1
…                       …    …
```

```
2024-06-19          39430.39      6
2024-06-20          33390.07      6
2024-06-21          70062.97      6
2024-06-24          24862.37      6
2024-06-25          34263.36      6

[124 rows x 7 columns]
```

[ ]:
```
# Consider all of the same values as duplicates
a_unique_all = a.drop_duplicates(keep=False)
a_unique_all
```

[ ]:
```
                Open      High       Low     Close  Shares Traded  \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…              …         …         …         …             …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover (  Cr)  Month
Date
2023-12-26          20081.33     12
2023-12-27          23059.25     12
2023-12-28          35031.00     12
2023-12-29          23697.88     12
2024-01-01          14184.09      1
…                      …    …
2024-06-19          39430.39      6
2024-06-20          33390.07      6
2024-06-21          70062.97      6
2024-06-24          24862.37      6
2024-06-25          34263.36      6

[124 rows x 7 columns]
```

[ ]:
```
# Drop columns with NaN values exceeding a threshold (e.g., 30%)
threshold = len(a) * 0.3  # Adjust the threshold percentage as needed
a_dropna_columns = a.dropna(thresh=threshold, axis=1)

# Print the DataFrame after dropping columns with NaN values
```

```
print("DataFrame after dropping columns with NaN values exceeding threshold:")
print(a_dropna_columns)
```

```
DataFrame after dropping columns with NaN values exceeding threshold:
                Open      High       Low     Close  Shares Traded  \
Date
2023-12-26  21365.20  21477.15  21329.45  21441.35    219467748.0
2023-12-27  21497.65  21675.75  21495.80  21654.75    256542963.0
2023-12-28  21715.00  21801.45  21678.00  21778.70    393080755.0
2023-12-29  21737.65  21770.30  21676.90  21731.40    270922276.0
2024-01-01  21727.75  21834.35  21680.85  21741.90    153995217.0
…              …         …         …         …             …
2024-06-19  23629.85  23664.00  23412.90  23516.00    328811255.0
2024-06-20  23586.15  23624.00  23442.60  23567.00    280336970.0
2024-06-21  23661.15  23667.10  23398.20  23501.10    609877803.0
2024-06-24  23382.30  23558.10  23350.00  23537.85    239358460.0
2024-06-25  23577.10  23754.15  23562.05  23721.30    298111025.0

            Turnover (  Cr)  Month
Date
2023-12-26         20081.33     12
2023-12-27         23059.25     12
2023-12-28         35031.00     12
2023-12-29         23697.88     12
2024-01-01         14184.09      1
…                       …    …
2024-06-19         39430.39      6
2024-06-20         33390.07      6
2024-06-21         70062.97      6
2024-06-24         24862.37      6
2024-06-25         34263.36      6

[124 rows x 7 columns]
```

Encode the Categorical data in your dataset using LabelEncoder

[ ]:
```
import pandas as pd

Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

print(a.columns)
```

```
Index(['Date ', 'Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ',
       'Turnover (  Cr)'],
      dtype='object')
```

[ ]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Example DataFrame with stock market data
Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

# Assuming 'Sector' is a categorical column in your DataFrame
categorical_columns = ['Sector']  # Replace with actual categorical columns in␣
 ↪your dataset

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
for col in categorical_columns:
    if col in a.columns:
        a[col + '_encoded'] = label_encoder.fit_transform(a[col])

# Print the updated DataFrame
print("Encoded DataFrame:")
print(a.head())
```

```
Encoded DataFrame:
         Date      Open      High       Low     Close   Shares Traded   \
0  26-DEC-2023  21365.20  21477.15  21329.45  21441.35      219467748.0
1  27-DEC-2023  21497.65  21675.75  21495.80  21654.75      256542963.0
2  28-DEC-2023  21715.00  21801.45  21678.00  21778.70      393080755.0
3  29-DEC-2023  21737.65  21770.30  21676.90  21731.40      270922276.0
4  01-JAN-2024  21727.75  21834.35  21680.85  21741.90      153995217.0

   Turnover (  Cr)
0         20081.33
1         23059.25
2         35031.00
3         23697.88
4         14184.09
```

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Example DataFrame with stock market data
Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

# Assuming 'Close' is the target variable (dependent variable)
```

```
X = a[['Open ', 'High ', 'Low ', 'Shares Traded ', 'Turnover ( Cr)']]  # Independent variables
y = a['Close ']  # Dependent variable

# Split dataset into training (75%) and test (25%) sets with random_state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Print sizes of training and test sets
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
```

```
Training set size: 93 samples
Test set size: 31 samples
```

[ ]:
```
#Role of random_state:
#Deterministic Split: The random_state parameter controls the randomness of the data split.
#When you set random_state to a specific number (e.g., 42),
# the data split will always be the same whenever you run the script with the same dataset.
# This is crucial for reproducibility and debugging purposes.
```

[ ]:
```
#Without random_state:
# If you omit random_state, the split will vary each time you run the script,
# which can be useful for exploring how different splits affect model performance.
```

Perform feature scaling by means of Standardization or Z score technique and interpret how each standardized value is computed.

[ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Example DataFrame with stock market data
Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

# Print columns to verify exact names and check for any issues
print("Columns in the DataFrame:")
print(a.columns)

# Assuming these are the correct column names after verifying
columns_to_scale = ['Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ', 'Turnover ( Cr)']

# Initialize StandardScaler
scaler = StandardScaler()
```

```python
# Fit and transform the specified columns
a[columns_to_scale] = scaler.fit_transform(a[columns_to_scale])

# Print the standardized DataFrame
print("\nStandardized DataFrame:")
print(a.head())

# Explanation of Standardization:
print("\nExplanation:")
for column in columns_to_scale:
    mean_value = a[column].mean()
    std_value = a[column].std()
    print(f"Column: {column}")
    print(f"Mean ( ): {mean_value}")
    print(f"Standard Deviation ( ): {std_value}")
    print(f"First few standardized values (z):")
    for i in range(5):
        original_value = a.loc[i, column]
        standardized_value = (original_value - mean_value) / std_value
        print(f"Original: {original_value}, Standardized: {standardized_value}")
    print("\n")
```

```
Columns in the DataFrame:
Index(['Date ', 'Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ',
       'Turnover (  Cr)'],
      dtype='object')

Standardized DataFrame:
         Date      Open      High       Low     Close   Shares Traded   \
0  26-DEC-2023 -1.633026 -1.644101 -1.458902 -1.487089       -0.971819
1  27-DEC-2023 -1.394660 -1.279609 -1.159968 -1.100423       -0.656363
2  28-DEC-2023 -1.003501 -1.048910 -0.832551 -0.875834        0.505377
3  29-DEC-2023 -0.962739 -1.106080 -0.834528 -0.961539       -0.534016
4  01-JAN-2024 -0.980555 -0.988529 -0.827430 -0.942513       -1.528896

   Turnover (  Cr)
0        -1.054354
1        -0.803015
2         0.207412
3        -0.749114
4        -1.552087

Explanation:
Column: Open
Mean ( ): 3.986058794863869e-15
Standard Deviation ( ): 1.0040568117894588
First few standardized values (z):
```

```
Original: -1.6330262057462506, Standardized: -1.6264280930835262
Original: -1.3946597254090285, Standardized: -1.3890247135751512
Original: -1.0035011410277936, Standardized: -0.9994465743819109
Original: -0.9627385832351294, Standardized: -0.9588487144659804
Original: -0.9805553303498054, Standardized: -0.9765934744292363


Column: High
Mean ( ): 2.3207242579261336e-15
Standard Deviation ( ): 1.004056811789459
First few standardized values (z):
Original: -1.6441012244035305, Standardized: -1.637458364007678
Original: -1.2796087815472539, Standardized: -1.2744386239128247
Original: -1.048910392488671, Standardized: -1.0446723533694022
Original: -1.1060802796738352, Standardized: -1.10161125016676
Original: -0.9885287138886165, Standardized: -0.9845346421452332


Column: Low
Mean ( ): -7.377611066863944e-15
Standard Deviation ( ): 1.0040568117894588
First few standardized values (z):
Original: -1.4589015366252824, Standardized: -1.4530069608563074
Original: -1.1599677983019567, Standardized: -1.1552810405564815
Original: -0.8325513443042973, Standardized: -0.8291874867324421
Original: -0.8345280627422608, Standardized: -0.8311562183965802
Original: -0.827429846533206, Standardized: -0.8240866819662619


Column: Close
Mean ( ): -1.4683594841816587e-15
Standard Deviation ( ): 1.004056811789459
First few standardized values (z):
Original: -1.4870889511437109, Standardized: -1.481080486365484
Original: -1.100423087047189, Standardized: -1.0959769149775316
Original: -0.8758343623294577, Standardized: -0.872295623161521
Original: -0.9615386517941488, Standardized: -0.9576536312526632
Original: -0.9425133866909929, Standardized: -0.9387052362218598


Column: Shares Traded
Mean ( ): 1.6066642144981941e-16
Standard Deviation ( ): 1.0040899966195638
First few standardized values (z):
Original: -0.9718194737300155, Standardized: -0.9678609258152235
Original: -0.6563627829552504, Standardized: -0.6536891963519258
Original: 0.5053772104611922, Standardized: 0.5033186389294074
Original: -0.5340155357553472, Standardized: -0.5318403106825081
```

```
Original: -1.5288964310964994, Standardized: -1.5226687211741818


Column: Turnover (  Cr)
Mean ( ): -5.1675014764057084e-17
Standard Deviation ( ): 1.0040899966195636
First few standardized values (z):
Original: -1.054354092444897, Standardized: -1.0500593532398053
Original: -0.8030149341706408, Standardized: -0.7997439839796479
Original: 0.20741166159748908, Standardized: 0.20656680406714048
Original: -0.7491139809386236, Standardized: -0.7460625874778564
Original: -1.552086515660187, Standardized: -1.5457643447156582
```

Perform Normalization using min max scalar and by using normalize method and interpret the difference between Standardization and Normalization.

[ ]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Example DataFrame with stock market data
Nifty_data = "E:/4TH_sem/AI_ML/STOCKS.csv"
a = pd.read_csv(Nifty_data)

# Assuming these are the correct column names after verifying
columns_to_normalize = ['Open ', 'High ', 'Low ', 'Close ', 'Shares Traded ',␣
 ↪'Turnover (  Cr)']

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the specified columns
a[columns_to_normalize] = scaler.fit_transform(a[columns_to_normalize])

# Print the normalized DataFrame
print("\nNormalized DataFrame:")
print(a.head())
```

```
Normalized DataFrame:
         Date      Open      High       Low     Close   Shares Traded   \
0  26-DEC-2023  0.072681  0.007908  0.079283  0.081591        0.203034
1  27-DEC-2023  0.126176  0.094438  0.147885  0.167553        0.240596
2  28-DEC-2023  0.213963  0.149206  0.223024  0.217482        0.378927
3  29-DEC-2023  0.223111  0.135634  0.222570  0.198429        0.255165
4  01-JAN-2024  0.219112  0.163541  0.224199  0.202659        0.136702
```

```
   Turnover (  Cr)
0         0.200714
1         0.233008
2         0.362834
3         0.239933
4         0.136762
```

Differences and Interpretation:

Range: Standardization does not bound values to a specific range, resulting in values that can be positive or negative and can vary widely based on the data's distribution. Min-Max normalization scales values to a fixed range ([0, 1]), making it useful for algorithms that require values to be on the same scale.

Effect on Distribution: Standardization maintains the shape of the original distribution (mean of 0 and variance of 1), whereas Min-Max normalization transforms data to a uniform range, potentially altering the distribution's shape.

Use Cases:

Use Standardization when dealing with algorithms like PCA (Principal Component Analysis) or clustering algorithms that require standardized data. Use Min-Max normalization when working with neural networks, image processing, or algorithms sensitive to the scale of data.

Interpretation: Normalization Process: Min-Max normalization scales each feature to a range of [0, 1] by subtracting the minimum value and dividing by the range (difference between maximum and minimum values).

Application: This ensures that all features have the same scale, making it suitable for algorithms that require all inputs to be on the same scale, such as neural networks.

Standardization (Z-score normalization): Method: Standardization transforms data such that it has a mean of 0 and a standard deviation of 1.

Min-Max Normalization: Method: Min-Max normalization scales data to a fixed range, typically [0, 1].

[ ]: