PROJECT TITLE

TESLASTOCKPRICEPREDICTION

PREPARED BY:-

Gajjar Niyati Nayan Kumar

TESLA STOCK PRICE ANALYSIS REPORT

1. Overview :

This report summarizes the analysis performed on Tesla's stock price data from June 2010 to present (1692 trading days). The analysis includes exploratory data analysis, feature engineering, and predictive modeling using both traditional machine learning and deep learning approaches.

2. Data Description

- Source: Tesla.csv
- Time Period: 6/29/2010 to present
- Records: 1,692 trading days
- Columns:
    - Date: Trading date
    - Open: Opening price
    - High: Highest price of the day
    - Low: Lowest price of the day
    - Close: Closing price
    - Volume: Trading volume
    - Adj Close: Adjusted closing price

3. Key Statistics

- Average Closing Price
- Standard Deviation
- Minimum Price
- Maximum Price
- Median Price
- Average Daily Volume

4. Exploratory Data Analysis 4.1 Price Trends

- The closing price plot shows significant growth over time
- Prices remained below 50 until mid-2013

4.2Feature Distributions

- All price features show similar distributions
- Volume has right-skewed distribution

4.3Annual Analysis

- Grouped by year, prices show increasing averages
- 2020-2021 saw strongest price increases

4.4Quarter-End Effects

- Prices higher at quarter-end.
- Volume lower at quarter-end

5. Modeling Approaches

5.1 Classification Models (Price Direction Prediction)

Model	Training AUC	Validation AUC
Logistic Regression	0.519	0.549
SVM (poly kernel)	0.472	0.448
XGBoost	0.964	0.573

Findings:
• XGBoost showed overfitting
• All models performed near random

5.2 LSTM Model (Price Prediction)

Architecture:
• Two LSTM layers (50 units)
• Dropout layers (0.2 rate)
• Dense output layer

Results:
• Achieved low training loss (0.0035)
• Predictions track actual prices reasonably
• Slight lag in predictions observed

6. Conclusions & Recommendations

6.1 Key Findings:
• Significant long-term growth with volatility
• Quarter-end effects present
• LSTM shows promise for trend tracking

6.2 Recommendations:
• Incorporate additional data sources
• Experiment with more complex architectures
• Implement walk-forward validation

Importing a library that is not in Colaboratory

To import a library that's not in Colaboratory by default, you can use `!pip install` or `!apt-get install`.

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

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

```
df = pd.read_csv('/content/Tesla.csv - Tesla.csv.csv')
df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

Next steps: View recommended plots New interactive sheet

```
df.shape
```

```
(1692, 7)
```

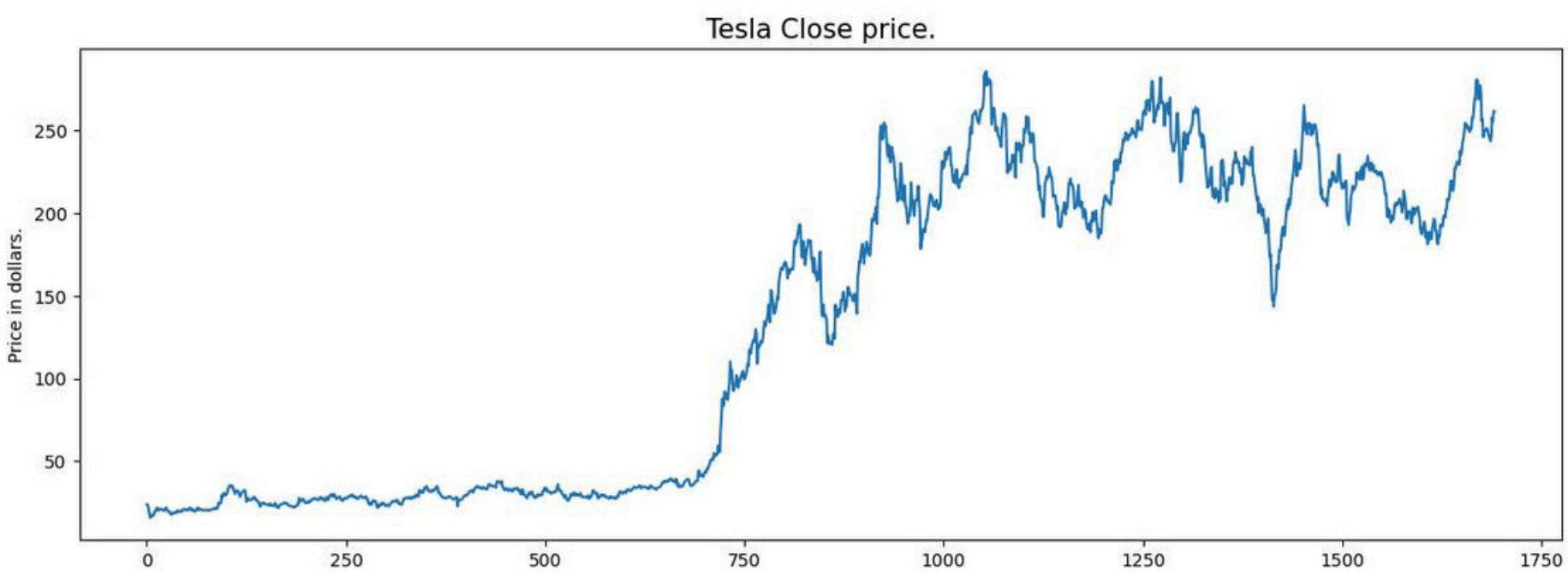
```
df.describe()
```

	Open	High	Low	Close	Volume	Adj Close
count	1692.000000	1692.000000	1692.000000	1692.000000	1.692000e+03	1692.000000
mean	132.441572	134.769698	129.996223	132.428658	4.270741e+06	132.428658
std	94.309923	95.694914	92.855227	94.313187	4.295971e+06	94.313187
min	16.139999	16.629999	14.980000	15.800000	1.185000e+05	15.800000
25%	30.000000	30.650000	29.215000	29.884999	1.194350e+06	29.884999
50%	156.334999	162.370002	153.150002	158.160004	3.180700e+06	158.160004
75%	220.557495	224.099999	217.119999	220.022503	5.662100e+06	220.022503
max	287.670013	291.420013	280.399994	286.040009	3.716390e+07	286.040009

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1692 entries, 0 to 1691
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1692 non-null   object
 1   Open       1692 non-null   float64
 2   High       1692 non-null   float64
 3   Low        1692 non-null   float64
 4   Close      1692 non-null   float64
 5   Volume     1692 non-null   int64
 6   Adj Close  1692 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 92.7+ KB
```

```
plt.figure(figsize=(15,5))
plt.plot(df['Close'])
plt.title('Tesla Close price.', fontsize=15)
plt.ylabel('Price in dollars.')
plt.show()
```




```
df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

Next steps: View recommended plots New interactive sheet

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

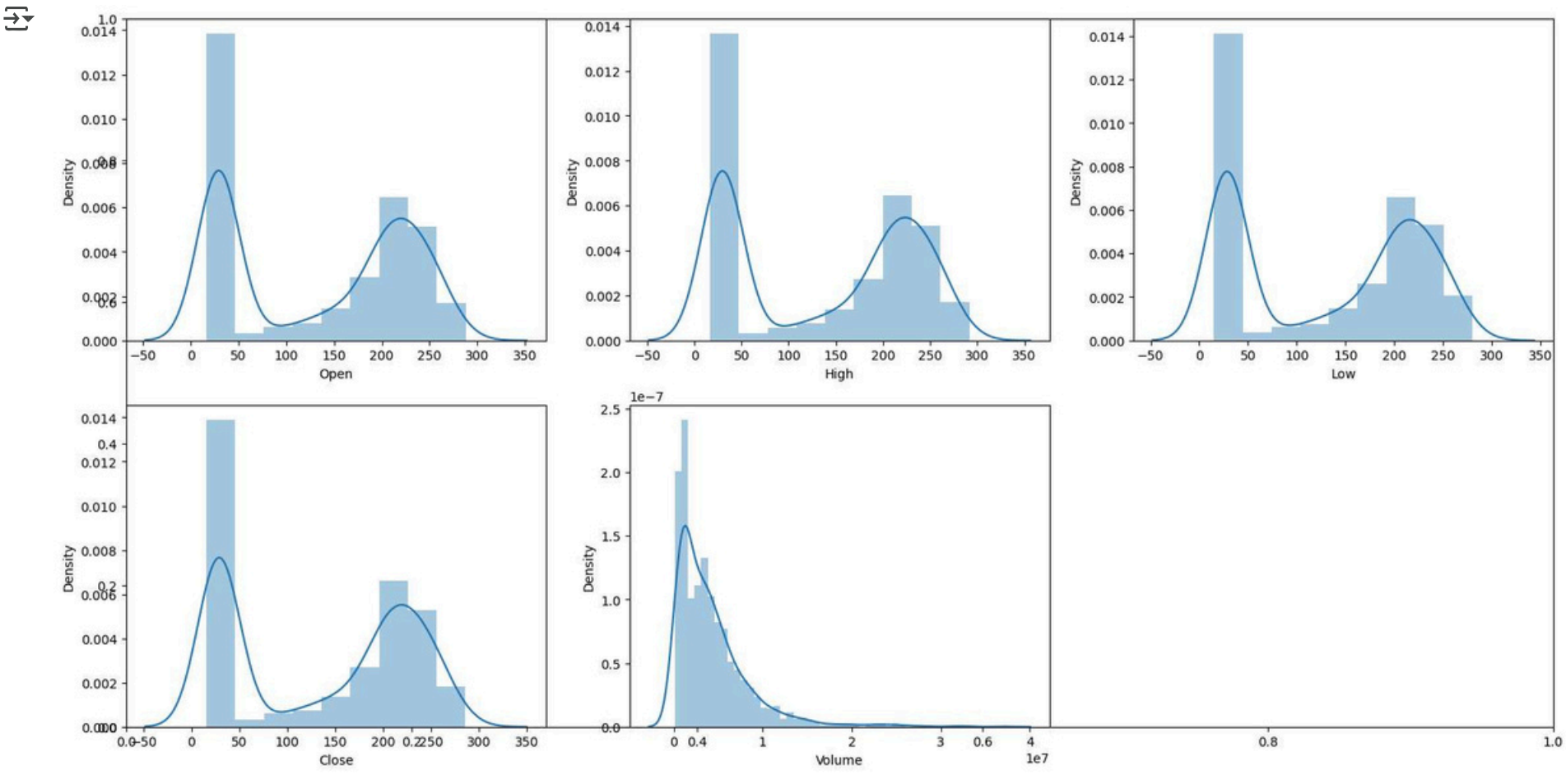
	0
Date	0
Open	0
High	0
Low	0
Close	0
Volume	0
Adj Close	0

dtype: int64

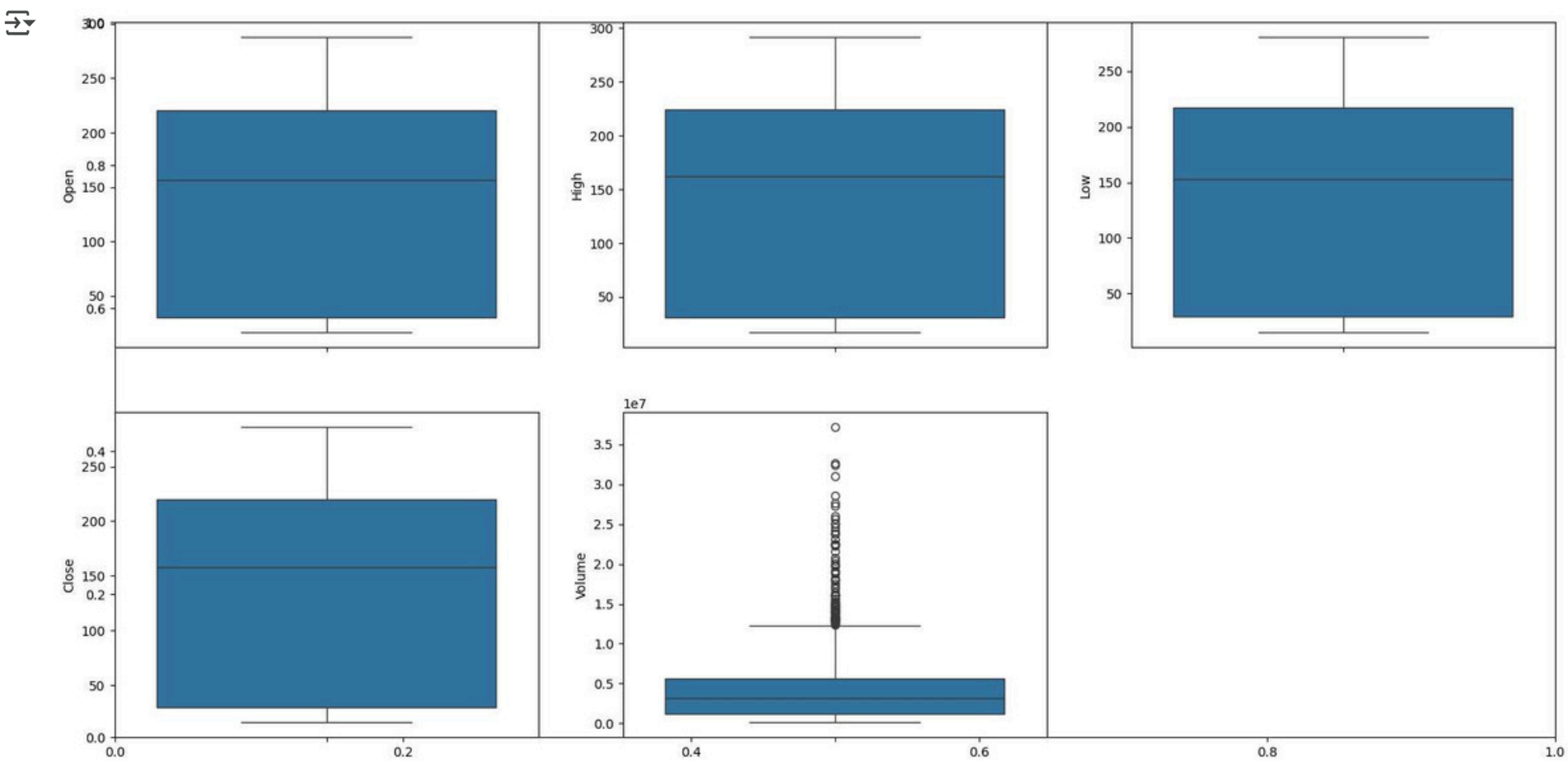
```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sb.distplot(df[col])
plt.show()
```




```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sb.boxplot(df[col])
plt.show()
```




```
splitted = df['Date'].str.split('/', expand=True)

df['day'] = splitted[1].astype('int')
df['month'] = splitted[0].astype('int')
df['year'] = splitted[2].astype('int')

df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close	day	month	year
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999	29	6	2010
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000	30	6	2010
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999	1	7	2010
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001	2	7	2010
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001	6	7	2010

Next steps: View recommended plots | New interactive sheet

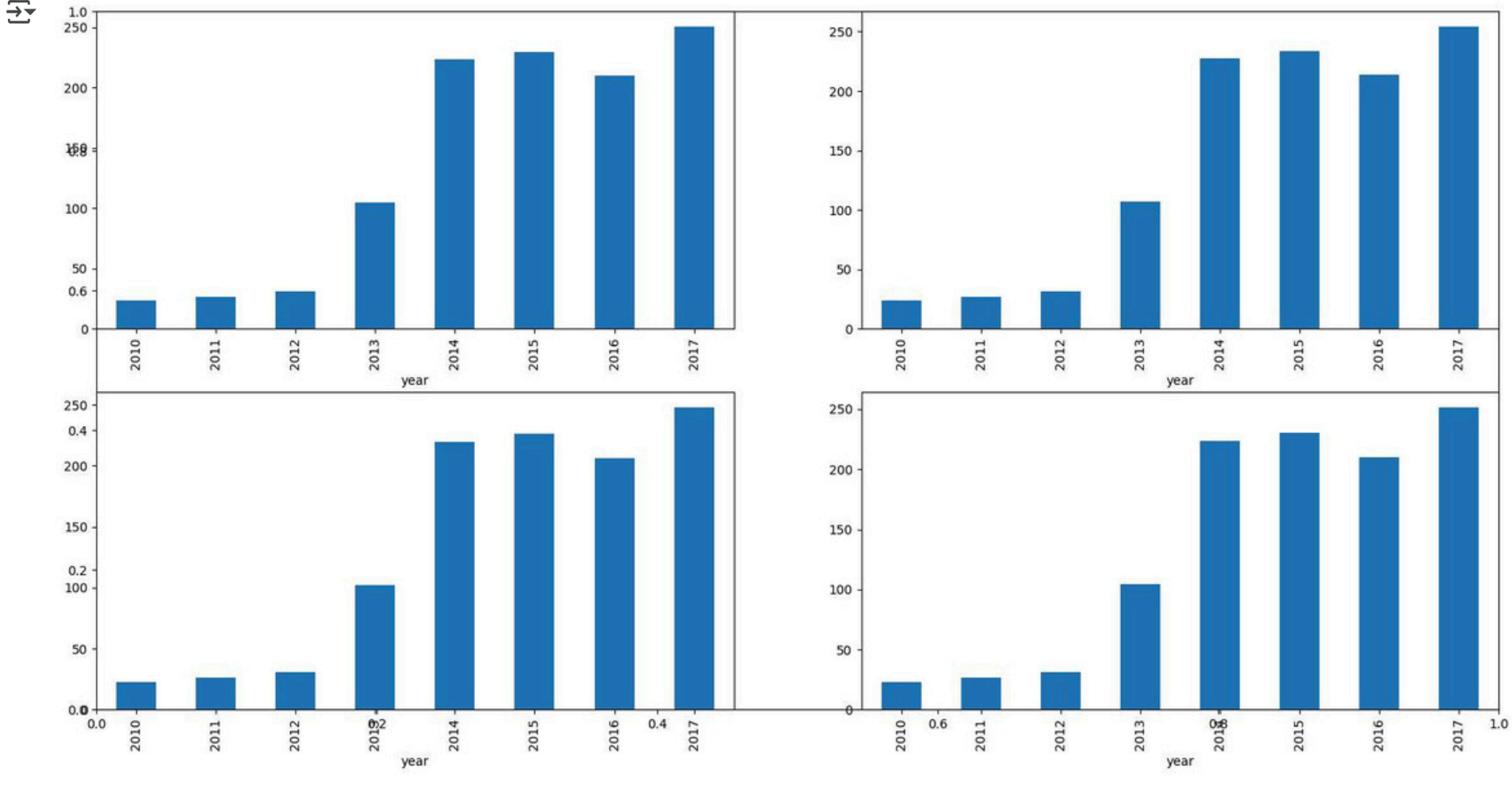
```
df['is_quarter_end'] = np.where(df['month']%3==0,1,0)
df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close	day	month	year	is_quarter_end
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999	29	6	2010	1
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000	30	6	2010	1
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999	1	7	2010	0
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001	2	7	2010	0
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001	6	7	2010	0

Next steps: View recommended plots New interactive sheet

```
data_grouped = df.drop('Date', axis=1).groupby('year').mean()
plt.subplots(figsize=(20,10))

for i, col in enumerate(['Open', 'High', 'Low', 'Close']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
plt.show()
```




```
df.drop('Date', axis=1).groupby('is_quarter_end').mean()
```

is_quarter_end	Open	High	Low	Close	Volume	Adj Close	day	month	year
0	130.813739	133.182620	128.257229	130.797709	4.461581e+06	130.797709	15.686501	6.141208	2013.353464
1	135.679982	137.927032	133.455777	135.673269	3.891084e+06	135.673269	15.657244	7.584806	2013.314488

```
df['open-close']  = df['Open'] - df['Close']
df['low-high']  = df['Low'] - df['High']
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

```
features = df[['open-close', 'low-high', 'is_quarter_end']]
target = df['target']

scaler = StandardScaler()
features = scaler.fit_transform(features)

X_train, X_valid, Y_train, Y_valid = train_test_split(
    features, target, test_size=0.1, random_state=2022)
print(X_train.shape, X_valid.shape)
```

```
(1522, 3) (170, 3)
```

```
models = [LogisticRegression(), SVC(
  kernel='poly', probability=True), XGBClassifier()]

for i in range(3):
  models[i].fit(X_train, Y_train)

  print(f'{models[i]} : ')
  print('Training Accuracy : ', metrics.roc_auc_score(
    Y_train, models[i].predict_proba(X_train)[:,1]))
  print('Validation Accuracy : ', metrics.roc_auc_score(
    Y_valid, models[i].predict_proba(X_valid)[:,1]))
  print()
```

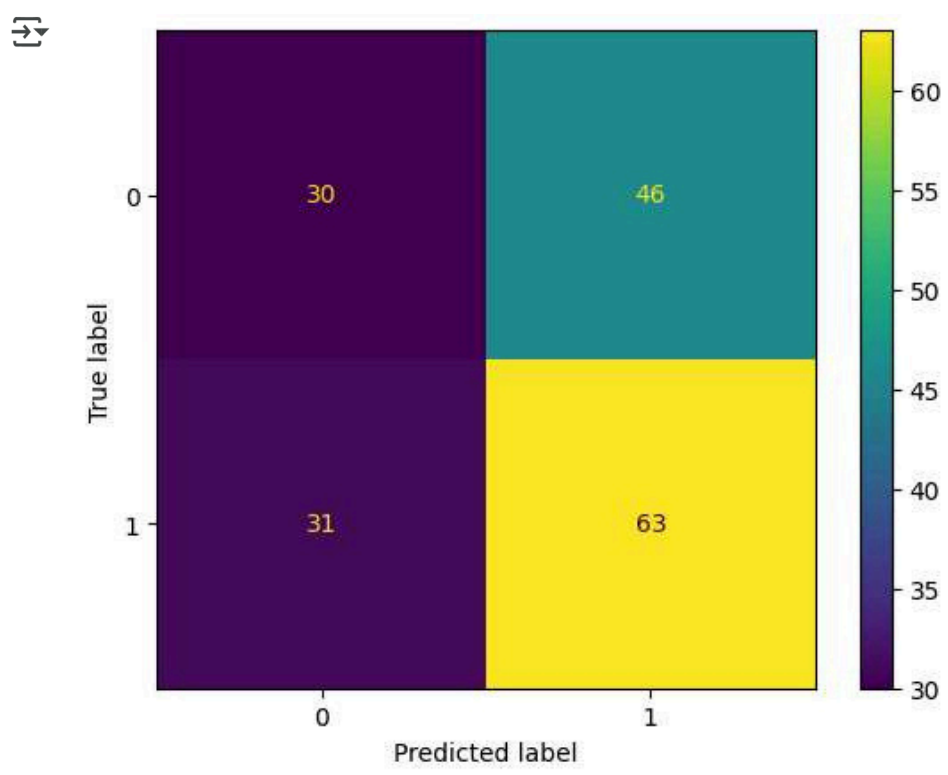
```
LogisticRegression() :
Training Accuracy :  0.5191606217616581
Validation Accuracy :  0.5436730123180291

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.4721053540587219
Validation Accuracy :  0.44820828667413215

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...) :
Training Accuracy :  0.9644602763385148
Validation Accuracy :  0.572998320268757
```

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(models[0], X_valid, Y_valid)
plt.show()
```




```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Scale the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1,1))

# Create sequences
def create_dataset(data, time_step=60):
    X, y = [], []
    for i in range(time_step, len(data)):
        X.append(data[i-time_step:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

X, y = create_dataset(scaled_data)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))


from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout

model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(Dropout(0.2))
```

```
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))  # output layer

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - loss: 0.0779
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - loss: 0.0057
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - loss: 0.0038
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0043
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - loss: 0.0035
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - loss: 0.0038
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - loss: 0.0036
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - loss: 0.0036
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0034
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - loss: 0.0035
<keras.src.callbacks.history.History at 0x7cd96e25c750>
```

```
predictions = model.predict(X)
predictions = scaler.inverse_transform(predictions)

# Plot
import matplotlib.pyplot as plt

plt.plot(scaler.inverse_transform(y.reshape(-1, 1)), label='Actual Price')
plt.plot(predictions, label='LSTM Predictions')
plt.legend()
plt.show()
```

```
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```