```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from google.colab import files
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn import metrics

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

```
#upload the file
uploaded= files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
df=pd.read_csv('tesla.csv')
```

```
df.shape
```

```
(1692, 7)
```

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

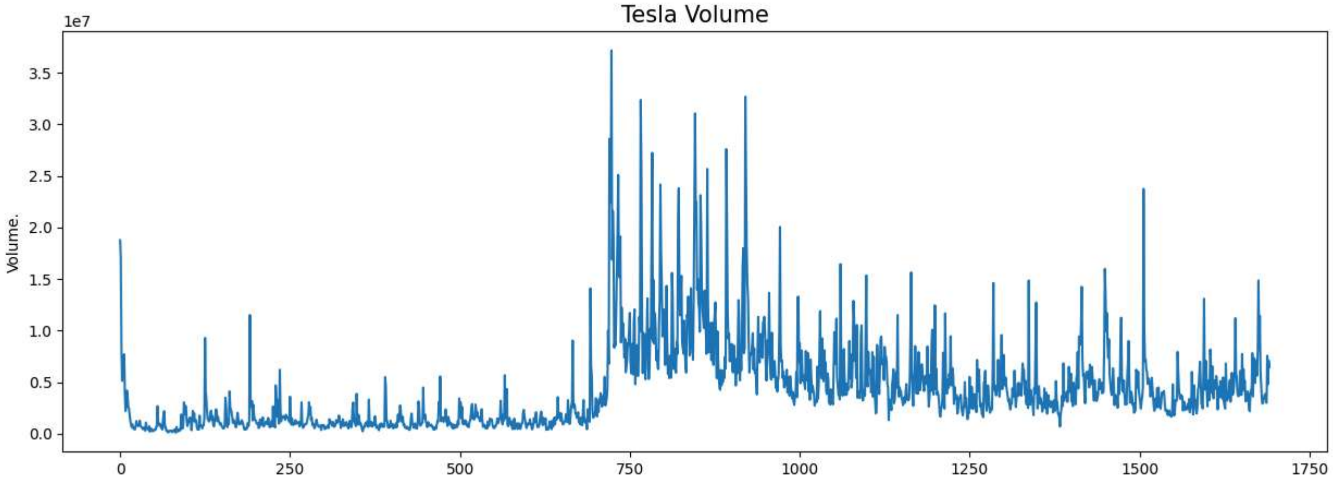
```
          0
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
Adj Close 0

dtype: int64
```

```
df.head()
```

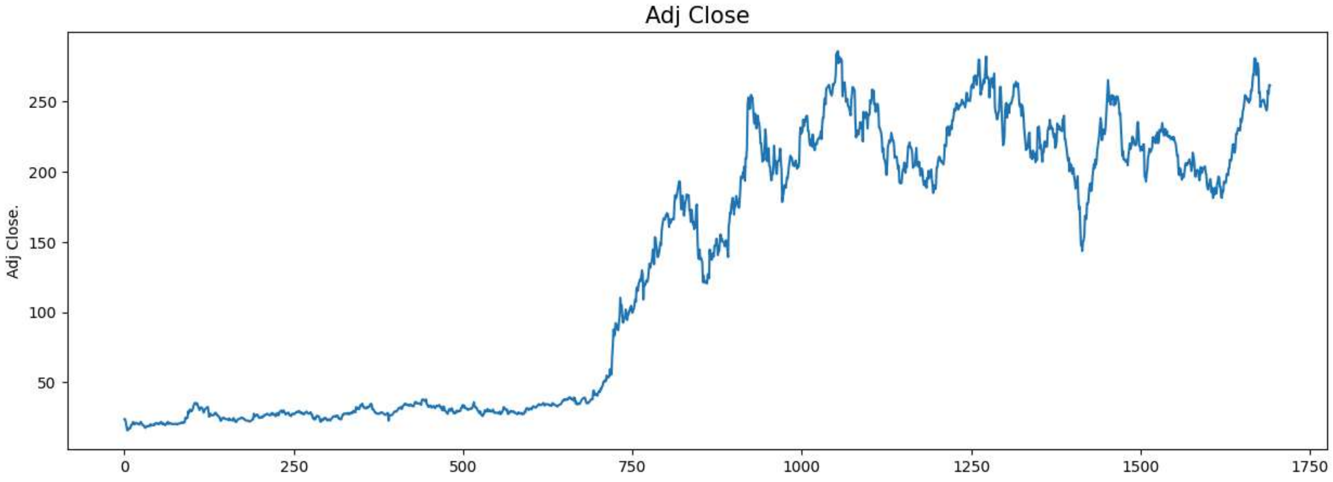
```
       Date       Open   High        Low      Close    Volume  Adj Close
0 6/29/2010  19.000000  25.00  17.540001  23.889999  18766300  23.889999
1 6/30/2010  25.790001  30.42  23.299999  23.830000  17187100  23.830000
2  7/1/2010  25.000000  25.92  20.270000  21.959999   8218800  21.959999
3  7/2/2010  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: Generate code with df View recommended plots New interactive sheet

```
plt.figure(figsize=(15,5))
plt.plot(df['Volume'])
plt.title('Tesla Volume', fontsize=15)
plt.ylabel('Volume.')
plt.show()
```



```
plt.figure(figsize=(15,5))
plt.plot(df['Adj Close'])
plt.title('Adj Close', fontsize=15)
plt.ylabel('Adj Close.')
plt.show()
```



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create a target variable indicating if the price will increase the next day
df['Next_Close'] = df['Close'].shift(-1)
df['Price_Change'] = (df['Next_Close'] > df['Close']).astype(int)  # 1 if price increases, 0 otherwise

# Drop the last row as it will have NaN in 'Next_Close'
df = df.dropna()

# Features and target variable
X = df[['Close']]  # You can add more features if available
y = df['Price_Change']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the Decision Tree classifier
```

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the classification report
print(f'Accuracy: {accuracy:.2f}')
print('\nClassification Report:')
print(classification_report(y_test, y_pred))

# Make a single prediction for the next day (for example, using the last available close price)
last_close_price = df[['Close']].iloc[-1].values.reshape(1, -1)
next_day_prediction = clf.predict(last_close_price)
print(f'\nPredicted price change for the next day: {"Increase" if next_day_prediction[0] == 1 else "Decrease"}')
```

```
Accuracy: 0.48

Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.51      0.49       251
           1       0.49      0.46      0.47       257

    accuracy                           0.48       508
   macro avg       0.48      0.48      0.48       508
weighted avg       0.48      0.48      0.48       508


Predicted price change for the next day: Decrease
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create and train the Random Forest classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

# Make predictions on the test set
rf_y_pred = rf_clf.predict(X_test)

# Calculate accuracy
rf_accuracy = accuracy_score(y_test, rf_y_pred)

# Print the classification report
print(f'Random Forest Accuracy: {rf_accuracy:.2f}')
print('\nRandom Forest Classification Report:')
print(classification_report(y_test, rf_y_pred))
```

```
Random Forest Accuracy: 0.47

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.47      0.47       251
           1       0.48      0.47      0.47       257

    accuracy                           0.47       508
   macro avg       0.47      0.47      0.47       508
weighted avg       0.47      0.47      0.47       508
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Create and train the SVM classifier
svm_clf = SVC(kernel='rbf', random_state=42)  # You can also use 'linear', 'poly', etc.
svm_clf.fit(X_train, y_train)

# Make predictions on the test set
svm_y_pred = svm_clf.predict(X_test)

# Calculate accuracy
svm_accuracy = accuracy_score(y_test, svm_y_pred)
```

```
# Print the classification report
print(f'SVM Accuracy: {svm_accuracy:.2f}')
print('\nSVM Classification Report:')
print(classification_report(y_test, svm_y_pred))
```

```
SVM Accuracy: 0.51

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.14      0.22       251
           1       0.51      0.88      0.64       257

    accuracy                           0.51       508
   macro avg       0.52      0.51      0.43       508
weighted avg       0.52      0.51      0.43       508
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create and train the Gradient Boosting classifier
gb_clf = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_clf.fit(X_train, y_train)

# Make predictions on the test set
gb_y_pred = gb_clf.predict(X_test)

# Calculate accuracy
gb_accuracy = accuracy_score(y_test, gb_y_pred)

# Print the classification report
print(f'Gradient Boosting Accuracy: {gb_accuracy:.2f}')
print('\nGradient Boosting Classification Report:')
print(classification_report(y_test, gb_y_pred))
```

```
Gradient Boosting Accuracy: 0.48

Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.45      0.26      0.33       251
           1       0.49      0.70      0.58       257

    accuracy                           0.48       508
   macro avg       0.47      0.48      0.45       508
weighted avg       0.47      0.48      0.45       508
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Assuming df is your DataFrame and it includes a 'Close' column with Tesla's closing prices

# Create a target variable indicating if the price will increase the next day
df['Next_Close'] = df['Close'].shift(-1)
df['Price_Change'] = (df['Next_Close'] > df['Close']).astype(int)  # 1 if price increases, 0 otherwise

# Drop the last row as it will have NaN in 'Next_Close'
df = df.dropna()

# Features and target variable
X = df[['Close']]  # You can add more features if available
y = df['Price_Change']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the Logistic Regression classifier
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the classification report
print(f'Logistic Regression Accuracy: {accuracy:.2f}')
print('\nClassification Report:')
print(classification_report(y_test, y_pred))
```

```
Logistic Regression Accuracy: 0.51

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.13      0.21       248
           1       0.51      0.87      0.65       259

    accuracy                           0.51       507
   macro avg       0.51      0.50      0.43       507
weighted avg       0.51      0.51      0.43       507
```

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report

# Create and train the XGBoost classifier
xgb_clf = xgb.XGBClassifier(n_estimators=100, random_state=42)
xgb_clf.fit(X_train, y_train)

# Make predictions on the test set
xgb_y_pred = xgb_clf.predict(X_test)

# Calculate accuracy
xgb_accuracy = accuracy_score(y_test, xgb_y_pred)

# Print the classification report
print(f'XGBoost Accuracy: {xgb_accuracy:.2f}')
print('\nXGBoost Classification Report:')
print(classification_report(y_test, xgb_y_pred))
```

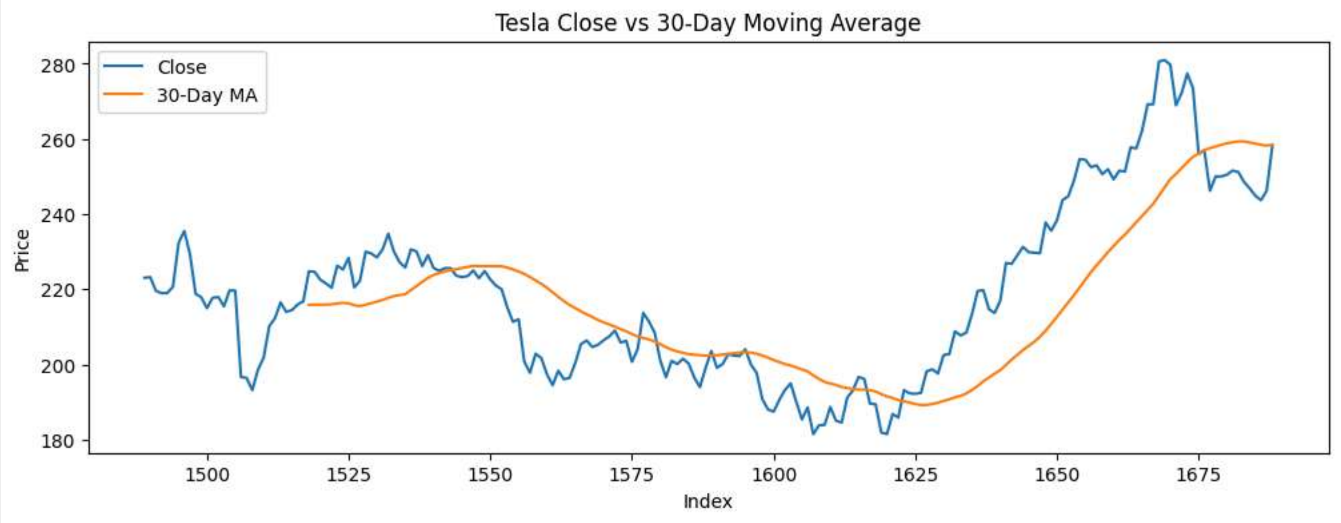
```
XGBoost Accuracy: 0.51

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.48      0.49       248
           1       0.52      0.54      0.53       259

    accuracy                           0.51       507
   macro avg       0.51      0.51      0.51       507
weighted avg       0.51      0.51      0.51       507
```

```
# === Tiny GUI for Next-Day Up/Down Prediction ===
import warnings, math
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

# XGBoost is optional; we'll include it only if available
try:
    import xgboost as xgb
    HAS_XGB = True
except Exception:
    HAS_XGB = False

import ipywidgets as widgets
```

```
from IPython.display import display, clear_output

# --- Prep data (reuses your df) ---
df_gui = df.copy()

# Create target: 1 if tomorrow's Close > today's Close else 0
df_gui['Next_Close'] = df_gui['Close'].shift(-1)
df_gui['Price_Change'] = (df_gui['Next_Close'] > df_gui['Close']).astype(int)

# Simple features (you can extend later)
# Here we add a couple of handy signals without making it complex
df_gui['Ret_1']   = df_gui['Close'].pct_change(1)
df_gui['MA_5']    = df_gui['Close'].rolling(5).mean()
df_gui['MA_10']   = df_gui['Close'].rolling(10).mean()
df_gui['MA_gap']  = df_gui['Close'] - df_gui['MA_5']  # price vs short MA
df_gui['Vol_5']   = df_gui['Close'].pct_change().rolling(5).std()

df_gui = df_gui.dropna().copy()

feature_cols = ['Close', 'Ret_1', 'MA_5', 'MA_10', 'MA_gap', 'Vol_5']
X = df_gui[feature_cols]
y = df_gui['Price_Change']

# Use last 30% as test to avoid time leakage (finance-friendly split)
split_idx = int(len(df_gui) * 0.7)
X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]

# --- Model zoo via pipelines (scale where it helps) ---
models = {
    "Decision Tree": Pipeline([("clf", DecisionTreeClassifier(random_state=42))]),
    "Random Forest": Pipeline([("clf", RandomForestClassifier(n_estimators=200, random_state=42))]),
    "Gradient Boosting": Pipeline([("clf", GradientBoostingClassifier(random_state=42))]),
    "SVM (RBF)": Pipeline([("scaler", StandardScaler()), ("clf", SVC(kernel="rbf", probability=True, random_state=42))]),
    "Logistic Regression": Pipeline([("scaler", StandardScaler()), ("clf", LogisticRegression(max_iter=1000, random_state=42))]),
}
if HAS_XGB:
    models["XGBoost"] = Pipeline([("clf", xgb.XGBClassifier(n_estimators=200, random_state=42, eval_metric="logloss"))])

# --- Widgets ---
model_dd   = widgets.Dropdown(options=list(models.keys()), value=list(models.keys())[0], description="Model:")
train_btn  = widgets.Button(description="Train", button_style="primary")
close_in   = widgets.FloatText(description="Enter Close:", value=float(X.iloc[-1]['Close']))
pred_btn   = widgets.Button(description="Predict", button_style="")
out        = widgets.Output(layout={'border': '1px solid #ddd'})
chart_out  = widgets.Output(layout={'border': '1px solid #ddd'})

# --- State ---
fitted = {"pipe": None, "acc": None, "report": None}

def plot_close_ma():
    with chart_out:
        clear_output(wait=True)
        # Plot last 200 points for readability
        tail = df_gui.tail(200).copy()
        tail['MA_30'] = tail['Close'].rolling(30).mean()
        plt.figure(figsize=(12,4))
        plt.plot(tail['Close'], label="Close")
        plt.plot(tail['MA_30'], label="30-Day MA")
        plt.title("Tesla Close vs 30-Day Moving Average")
        plt.xlabel("Index")
        plt.ylabel("Price")
        plt.legend()
        plt.show()

def on_train_clicked(_):
    with out:
        clear_output(wait=True)
        name = model_dd.value
        pipe = models[name]
        pipe.fit(X_train, y_train)
        y_pred = pipe.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        rep = classification_report(y_test, y_pred, digits=3)

        fitted["pipe"] = pipe
        fitted["acc"] = acc
        fitted["report"] = rep
        fitted["report"] = rep

        print(f"✅ Trained: {name}")
        print(f"Test Accuracy: {acc:.3f}\n")
        print("Classification Report:")
        print(rep)

def on_predict_clicked(_):
    with out:
        if fitted["pipe"] is None:
            print("Please click Train first.")
            return

        # 1) Predict using the last real row (today) → tomorrow's direction
        last_row = X.iloc[[-1]].copy()

        # 2) Predict using user-entered hypothetical Close
        #    Recompute simple signals with the custom Close injected
        custom_close = float(close_in.value)
        # Take the last 10 rows (to rebuild rolling features consistently)
        context = df_gui.iloc[-10:].copy()
        context.iloc[-1, context.columns.get_loc('Close')] = custom_close
        context['Ret_1']  = context['Close'].pct_change(1)
        context['MA_5']   = context['Close'].rolling(5).mean()
        context['MA_10']  = context['Close'].rolling(10).mean()
        context['MA_gap'] = context['Close'] - context['MA_5']
        context['Vol_5']  = context['Close'].pct_change().rolling(5).std()
        custom_row = context[feature_cols].iloc[[-1]]

        # Predictions + (if available) probabilities
        pipe = fitted["pipe"]
        pred_last   = pipe.predict(last_row)[0]
        pred_custom = pipe.predict(custom_row)[0]

        def prob_of_increase(row):
            try:
                proba = pipe.predict_proba(row)[0]
                # proba[1] is probability of class "1" (increase)
                return proba[1]
            except Exception:
                return None

        p_last   = prob_of_increase(last_row)
        p_custom = prob_of_increase(custom_row)

        print("— Prediction (latest data row) —")
        print(f"Next-day: {'Increase ↑' if pred_last==1 else 'Decrease ↓'}" +
              (f" | P(↑)={p_last:.2f}" if p_last is not None else ""))

        print("\n— Prediction (your Close input) —")
        print(f"Close entered: {custom_close}")
        print(f"Next-day: {'Increase ↑' if pred_custom==1 else 'Decrease ↓'}" +
              (f" | P(↑)={p_custom:.2f}" if p_custom is not None else ""))

# Wire up
train_btn.on_click(on_train_clicked)
pred_btn.on_click(on_predict_clicked)

# Show UI
ui = widgets.VBox([
    widgets.HBox([model_dd, train_btn]),
    widgets.HBox([close_in, pred_btn]),
    chart_out,
    out
])
display(ui)

# Initial chart
plot_close_ma()
```

Model: Decision Tree

Enter Close: 258



```
✅ Trained: Decision Tree
Test Accuracy: 0.502

Classification Report:
              precision    recall  f1-score   support

           0      0.491     0.581     0.533       246
           1      0.516     0.426     0.467       258

    accuracy                          0.502       504
   macro avg      0.504     0.504     0.500       504
weighted avg      0.504     0.502     0.499       504

— Prediction (latest data row) —
Next-day: Decrease ↓ | P(↑)=0.00

— Prediction (your Close input) —
Close entered: 258.0
Next-day: Decrease ↓ | P(↑)=0.00
```