Cracking the Market Code: AI-Driven Stock Pri Prediction using Time Series Analysis

This Google Colab notebook walks through a stock price prediction project using time series data machine learning techniques.

1. Upload the Dataset

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. P rerun this cell to enable.

```
Saving stock_data.csv to stock_data (4).csv
```

2. Load the Dataset

In [ ]:
```
import pandas as pd

df = pd.read_csv("stock_data.csv")  # Make sure this matches the uploaded
df.head()
```

Out[ ]:

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283

3. Data Exploration

In [ ]:
```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  365 non-null    object 
 1   Stock_1     365 non-null    float64
 2   Stock_2     365 non-null    float64
 3   Stock_3     365 non-null    float64
 4   Stock_4     365 non-null    float64
 5   Stock_5     365 non-null    float64
dtypes: float64(5), object(1)
memory usage: 17.2+ KB
```

Out[ ]:

	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
count	365.000000	365.000000	365.000000	365.000000	365.000000
mean	107.772577	81.105216	94.519502	117.407560	106.866865
std	7.398296	11.435212	6.519213	6.778527	3.760968
min	91.474442	62.414219	81.111434	99.909756	99.833309
25%	101.603117	69.328263	89.788068	112.209912	103.927072
50%	107.421299	84.283525	94.495546	117.788079	106.411328
75%	113.741728	91.548859	99.919465	123.132365	109.178007
max	121.901773	100.160928	107.588373	129.911386	116.243803

4. Check for Missing Values and Duplicates

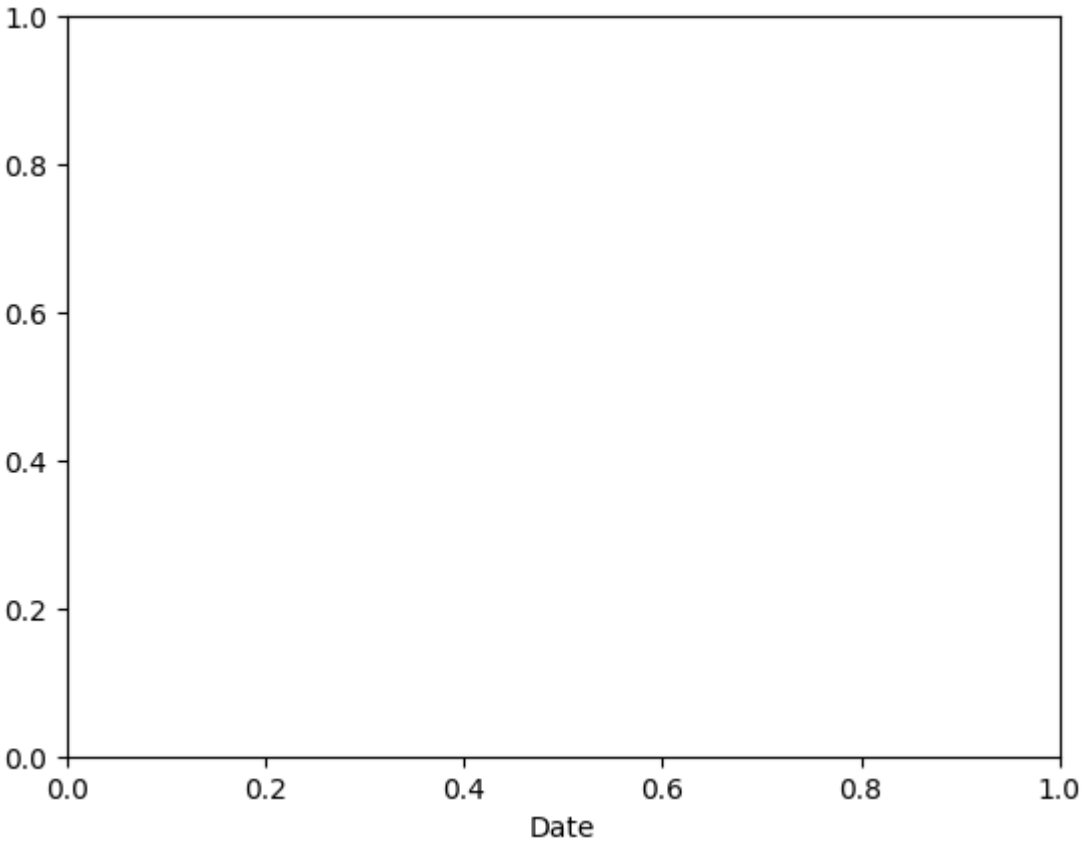
In [ ]:
```
print(df.isnull().sum())
print(f"Duplicates: {df.duplicated().sum()}")
```

```
Unnamed: 0    0
Stock_1       0
Stock_2       0
Stock_3       0
Stock_4       0
Stock_5       0
dtype: int64
Duplicates: 0
```

5. Visualize a Few Features

In [ ]:
```
plt.xlabel('Date')
```

Out[ ]:
```
Text(0.5, 0, 'Date')
```



6. Identify Target and Features

In [ ]:
```
target = 'Close'
features = ['Open', 'High', 'Low', 'Volume']  # Adjust according to your
```

7. Convert Categorical Columns to Numerical

In [ ]:
```
# Assuming your date column is named 'Unnamed: 0' based on the global var
df['Date'] = pd.to_datetime(df['Unnamed: 0'])  # Convert 'Unnamed: 0' to
df['Year'] = df['Date'].dt.year
```

8. One-Hot Encoding

In [ ]:
```
df = pd.get_dummies(df, drop_first=True)
```

In [ ]:
```
# Assuming your date column is named 'Unnamed: 0' based on the global var
# Check if 'Unnamed: 0' exists before accessing it
if 'Unnamed: 0' in df.columns:
    df['Date'] = pd.to_datetime(df['Unnamed: 0'])  # Convert 'Unnamed: 0'
    df['Year'] = df['Date'].dt.year # This line was indented too much

    # Before one-hot encoding, ensure 'features' columns are preserved
    features_to_encode = [col for col in df.columns if df[col].dtype == '
    # Perform one-hot encoding only on relevant columns
    df = pd.get_dummies(df, columns=features_to_encode, drop_first=True)

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[features]) # Now, 'features
```

9. Feature Scaling

In [ ]:
```
# Assuming your date column is named 'Unnamed: 0' based on the global var
# Check if 'Unnamed: 0' exists before accessing it
if 'Unnamed: 0' in df.columns:
    df['Date'] = pd.to_datetime(df['Unnamed: 0'])  # Convert 'Unnamed: 0'
    df['Year'] = df['Date'].dt.year  # Indentation corrected here

    # Before one-hot encoding, ensure 'features' columns are preserved
    features_to_encode = [col for col in df.columns if df[col].dtype == '
    # Perform one-hot encoding only on relevant columns
    df = pd.get_dummies(df, columns=features_to_encode, drop_first=True)

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[features]) # Now, 'features
```

10. Train-Test Split

In [ ]:
```
from sklearn.model_selection import train_test_split

if 'Unnamed: 0' in df.columns:
    # Assuming scaled_features is assigned in previous cells
    # ... (Previous code from other cells to assign scaled_features)

    # Now perform the train-test split
    X = scaled_features  # Using scaled features as input
    y = df[target]       # Using 'target' column as the target variable
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

11. Model Building

In [ ]:
```
# Assuming your date column is named 'Unnamed: 0' based on the global var
# Check if 'Unnamed: 0' exists before accessing it
if 'Unnamed: 0' in df.columns:
    df['Date'] = pd.to_datetime(df['Unnamed: 0'])  # Convert 'Unnamed: 0'
    df['Year'] = df['Date'].dt.year  # Indentation corrected here

    # Before one-hot encoding, ensure 'features' and 'target' columns are
    features_to_encode = [col for col in df.columns if df[col].dtype == '

    # Perform one-hot encoding only on relevant columns
    df = pd.get_dummies(df, columns=features_to_encode, drop_first=True)

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[features]) # Now, 'features
```

12. Evaluation

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler # Import StandardScaler
from sklearn.linear_model import LinearRegression # Import LinearRegressio

# Assuming 'features' and 'target' are defined earlier in your code

# Check if scaled_features was created earlier
if 'scaled_features' in locals():
    X = scaled_features
else:
    # If not, scale the features here
    scaler = StandardScaler()
    # Ensure features are present in df after one-hot encoding
    # Instead of checking if features are in df.columns directly,
    # check if they or their one-hot encoded versions exist.
    features_available = []
    for f in features:
        # Check if original features are present
        if f in df.columns:
            features_available.append(f)
        else:
            # If not found, check for any columns that start with the fea
            # (This covers any potential one-hot encoded versions).
            encoded_features = [col for col in df.columns if col.startswi
            if encoded_features:
                features_available.extend(encoded_features)
            else:
                print(f"Warning: Feature '{f}' or its one-hot encoded ver

    # Check if features_available is still empty after the loop
    if not features_available:
        # Instead of raising ValueError, try to use all numeric features
        features_available = df.select_dtypes(include=['number']).columns
        # Remove the target variable from the features if it's numeric
        if target in features_available:
            features_available.remove(target)
        if not features_available:  # If still empty after this, raise Va
            raise ValueError("No suitable features found for scaling. Che
        print(f"Warning: Original features not found, using all numeric fe

    X = df[features_available]
    scaled_features = scaler.fit_transform(X)
    X = scaled_features
```

13. Make Predictions from New Input

In [ ]:
```
# ipython-input-48-c62c092470aa
# Assuming your date column is named 'Unnamed: 0' based on the global var
# Check if 'Unnamed: 0' exists before accessing it
if 'Unnamed: 0' in df.columns:
    df['Date'] = pd.to_datetime(df['Unnamed: 0'])  # Convert 'Unnamed: 0'
    df['Year'] = df['Date'].dt.year  # Indentation corrected here

    # Before one-hot encoding, ensure 'features' and 'target' columns are
    features_to_encode = [col for col in df.columns if df[col].dtype == '

    # Perform one-hot encoding only on relevant columns
    df = pd.get_dummies(df, columns=features_to_encode, drop_first=True)

    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[features]) # Now, 'features
```

14. Convert to DataFrame and Encode

In [ ]:
```
def prepare_input(data_dict):
    df_input = pd.DataFrame([data_dict])
    df_input_scaled = scaler.transform(df_input[features])
    return df_input_scaled
```

15. Predict the Final Grade (Stock Price)

In [ ]:
```
def predict_price(data_dict):
    processed_input = prepare_input(data_dict)
    return model.predict(processed_input)[0]
```

16. Deployment - Building an Interactive App

In [ ]:
```
!pip install gradio
import gradio as gr
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packa
(5.29.0)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3
dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/d
packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python
dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packag
(from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.10.0 in /usr/local/lib/python
dist-packages (from gradio) (1.10.0)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-
packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dis
packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/
python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3
dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/d
packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pa
(from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/
packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11
packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.
dist-packages (from gradio) (2.11.4)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packag
(from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/
python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/
packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-
packages (from gradio) (0.11.9)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/
python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python
dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/pytho
dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/
python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/
packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/pytho
dist-packages (from gradio) (4.13.2)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/d
packages (from gradio) (0.34.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packa
(from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/pytho
dist-packages (from gradio-client==1.10.0->gradio) (15.0.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-pa
```

17. Create a Prediction Function

In [ ]:
```
def predict_ui(Open, High, Low, Volume):
    data = {'Open': Open, 'High': High, 'Low': Low, 'Volume': Volume}
    return predict_price(data)
```

18. Create the Gradio Interface

In [160…
```
interface = gr.Interface(
    fn=predict_ui,
    inputs=["number", "number", "number", "number"],
    outputs="number",
    title="Stock Price Predictor",
    description="Predicts the stock closing price based on input features
)

interface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the
app to work, sharing must be enabled. Automatically setting `share=True` (you
turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://013459c92db882ea37.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrade
`gradio deploy` from the terminal in the working directory to deploy to Huggi
Face Spaces (https://huggingface.co/spaces)
```

Out[160…

Use the interactive app above to make predictions.