Data Preparation and Exploration

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the data
df = pd.read_csv('ORCL.csv', parse_dates=['Date'], index_col='Date')

# Display basic info
print(df.info())
print(df.head())
```

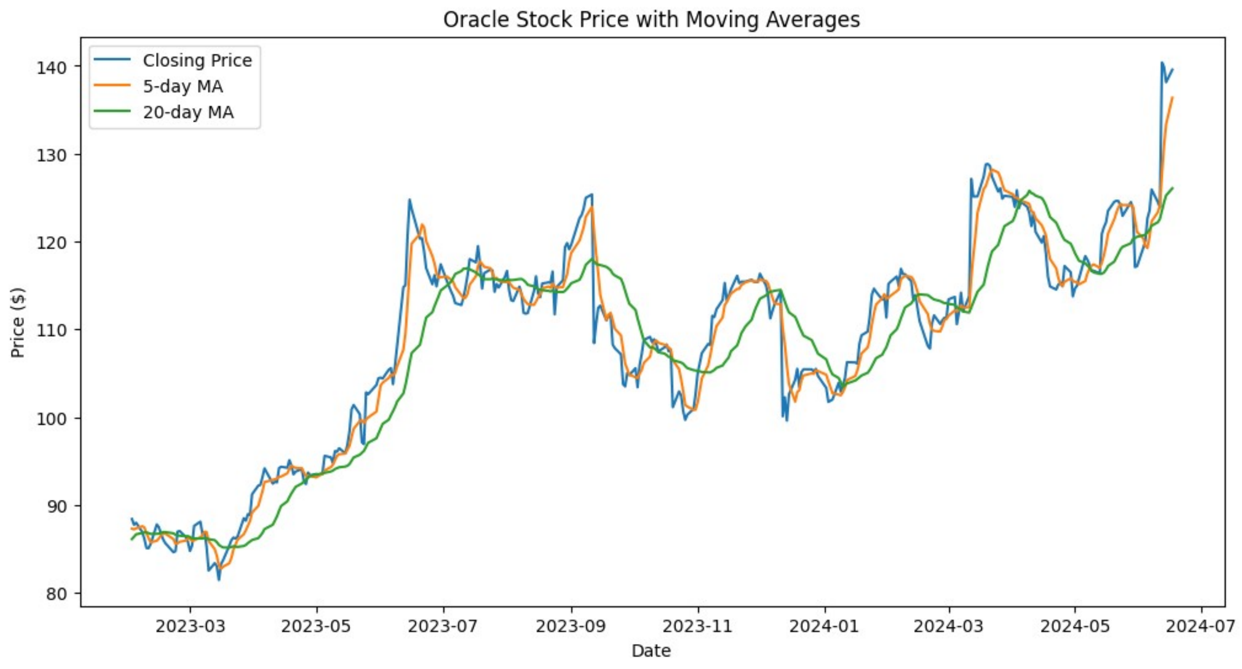
```
<class 'pandas.core.frame.DataFrame'>
Index: 366 entries, 2023-01-03 00:00:00-05:00 to 2024-06-17 00:00:00-
04:00
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          366 non-null    float64
 1   High          366 non-null    float64
 2   Low           366 non-null    float64
 3   Close         366 non-null    float64
 4   Volume        366 non-null    int64
 5   Dividends     366 non-null    float64
 6   Stock Splits  366 non-null    int64
dtypes: float64(5), int64(2)
memory usage: 22.9+ KB
None
                                Open       High        Low      Close
\
Date

2023-01-03 00:00:00-05:00  80.662024  82.001985  80.456629  81.884621

2023-01-04 00:00:00-05:00  82.354092  83.302828  81.806372  82.627960

2023-01-05 00:00:00-05:00  83.136553  83.527785  81.395577  82.461678

2023-01-06 00:00:00-05:00  82.882260  84.447182  81.982431  83.782097

2023-01-09 00:00:00-05:00  85.254708  85.991016  84.587128  84.842377


                            Volume  Dividends  Stock Splits
```

```
Date
2023-01-03 00:00:00-05:00  8997500       0.00             0
2023-01-04 00:00:00-05:00  7836200       0.00             0
2023-01-05 00:00:00-05:00  7643800       0.00             0
2023-01-06 00:00:00-05:00  8641600       0.00             0
2023-01-09 00:00:00-05:00  7519700       0.32             0
```

Data Cleaning and Feature Engineering

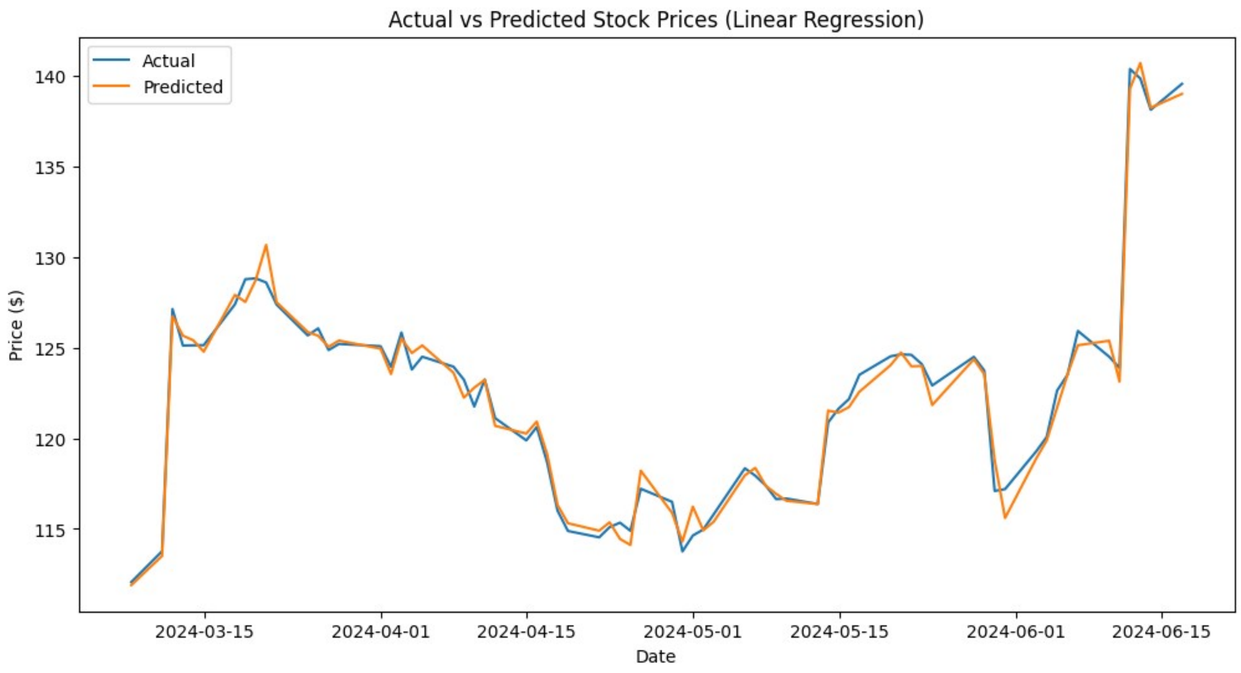
```
# Check for missing values
print(df.isnull().sum())

# Create additional features
df['Daily_Return'] = df['Close'].pct_change()
df['Log_Return'] = np.log(df['Close']/df['Close'].shift(1))
df['MA_5'] = df['Close'].rolling(window=5).mean()
df['MA_20'] = df['Close'].rolling(window=20).mean()
df['Volatility'] = df['Daily_Return'].rolling(window=20).std() *
np.sqrt(252)

# Drop rows with NaN values from rolling calculations
df = df.dropna()

# Visualize the closing price and moving averages
plt.figure(figsize=(12,6))
plt.plot(df['Close'], label='Closing Price')
plt.plot(df['MA_5'], label='5-day MA')
plt.plot(df['MA_20'], label='20-day MA')
plt.title('Oracle Stock Price with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```

```
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```



1. Stock Price Prediction Models Linear Regression Model

```
# Prepare data for linear regression
X = df[['Open', 'High', 'Low', 'Volume']]
y = df['Close']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, shuffle=False)

# Train the model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions
y_pred = lr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Linear Regression MSE: {mse:.2f}, R2: {r2:.2f}")

# Plot actual vs predicted
plt.figure(figsize=(12,6))
plt.plot(y_test.index, y_test, label='Actual')
plt.plot(y_test.index, y_pred, label='Predicted')
plt.title('Actual vs Predicted Stock Prices (Linear Regression)')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.show()

Linear Regression MSE: 0.46, R2: 0.99
```

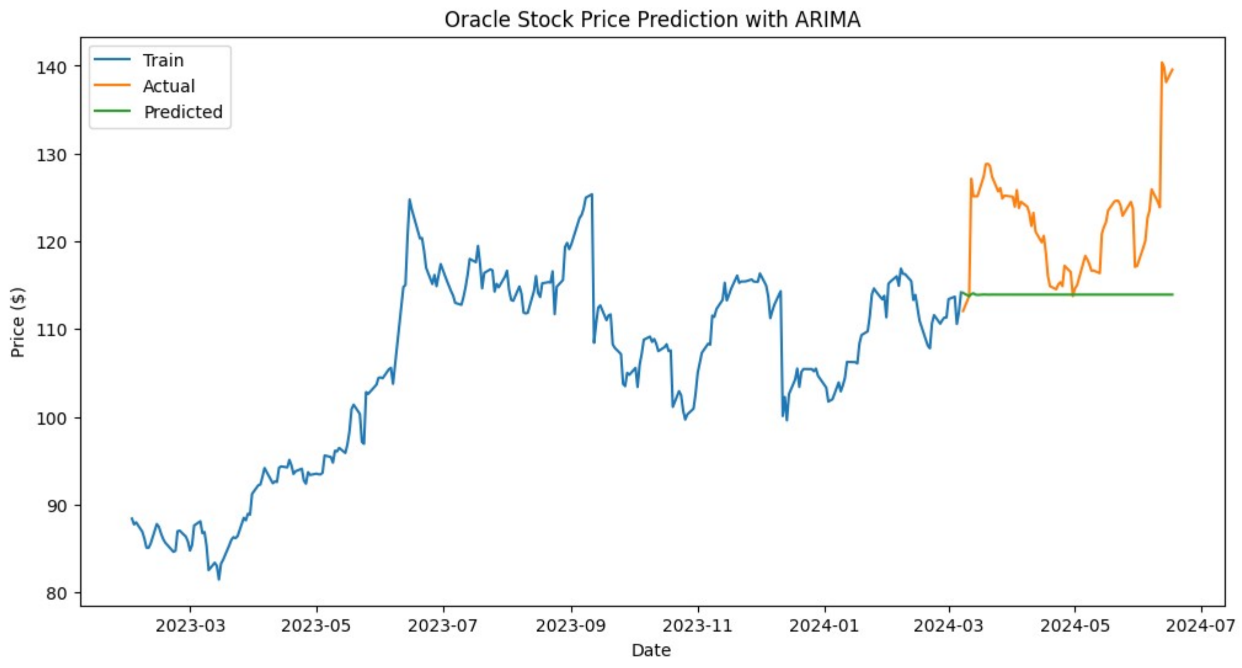


ARIMA Model

```
# Prepare data for ARIMA
ts_data = df['Close']

# Split into train and test
train_size = int(len(ts_data) * 0.8)
train, test = ts_data[:train_size], ts_data[train_size:]

# Fit ARIMA model
model = ARIMA(train, order=(5,1,0))
model_fit = model.fit()

# Make predictions
predictions = model_fit.forecast(steps=len(test))

# Evaluate
mse = mean_squared_error(test, predictions)
print(f"ARIMA MSE: {mse:.2f}")

# Plot results
plt.figure(figsize=(12,6))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Actual')
```

```python
plt.plot(test.index, predictions, label='Predicted')
plt.title('Oracle Stock Price Prediction with ARIMA')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```

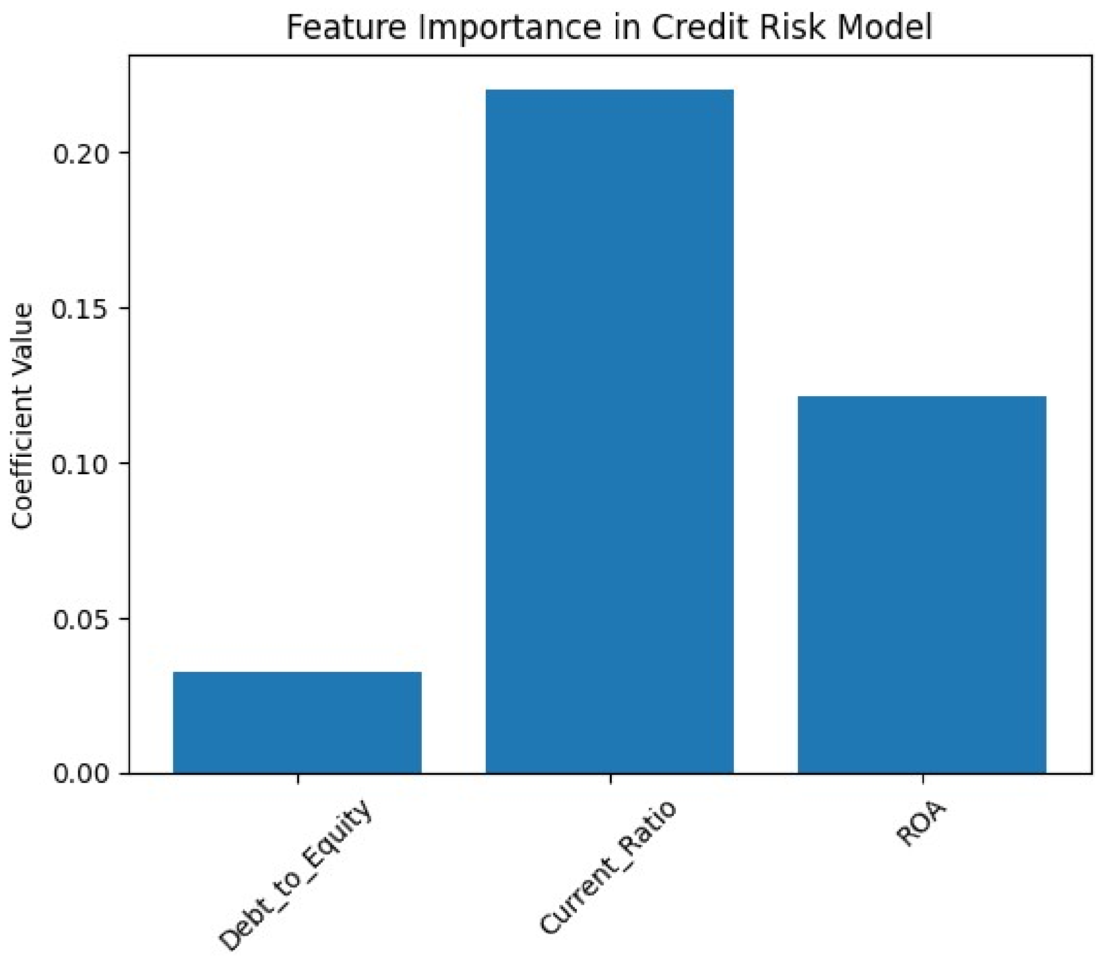
```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/
tsa_model.py:473: ValueWarning: An unsupported index was provided. As
a result, forecasts cannot be generated. To use the model for
forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: An unsupported index was provided. As a result,
forecasts cannot be generated. To use the model for forecasting, use
one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: An unsupported index was provided. As a result,
forecasts cannot be generated. To use the model for forecasting, use
one of the supported classes of index.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:837: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:837: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(

ARIMA MSE: 107.04
```



1. Credit Risk Modeling (Simulated Data)

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Simulate credit risk data (hypothetical)
np.random.seed(42)
n_samples = 1000
credit_data = pd.DataFrame({
    'Debt_to_Equity': np.random.normal(1.5, 0.5, n_samples),
    'Current_Ratio': np.random.normal(2.0, 0.8, n_samples),
    'ROA': np.random.normal(0.08, 0.03, n_samples),
    'Default': np.random.binomial(1, 0.15, n_samples)
})

# Split data
X = credit_data[['Debt_to_Equity', 'Current_Ratio', 'ROA']]
y = credit_data['Default']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)

# Train logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions
y_pred = logreg.predict(X_test)

# Evaluation
print("Credit Risk Model Evaluation:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plot feature importance
importance = logreg.coef_[0]
plt.bar(X.columns, importance)
plt.title('Feature Importance in Credit Risk Model')
plt.ylabel('Coefficient Value')
plt.xticks(rotation=45)
plt.show()
```

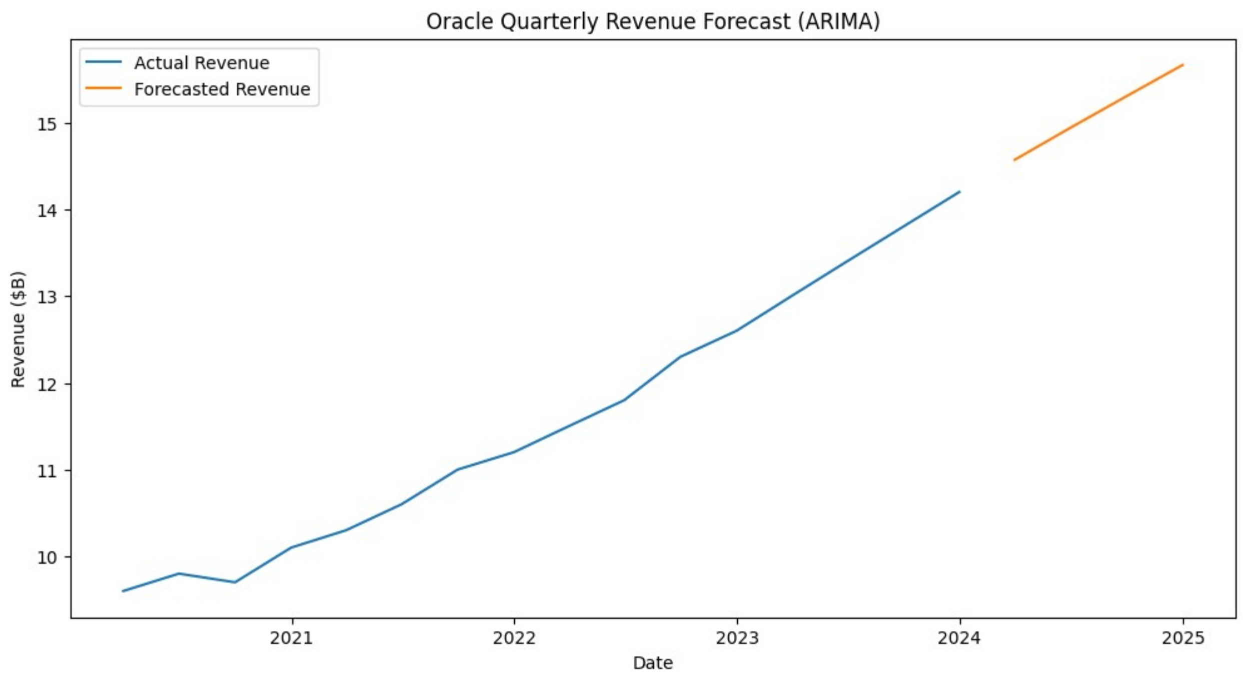
```
Credit Risk Model Evaluation:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       269
           1       0.00      0.00      0.00        31

    accuracy                           0.90       300
   macro avg       0.45      0.50      0.47       300
weighted avg       0.80      0.90      0.85       300

Confusion Matrix:
[[269   0]
 [ 31   0]]

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```



1. Revenue Forecasting

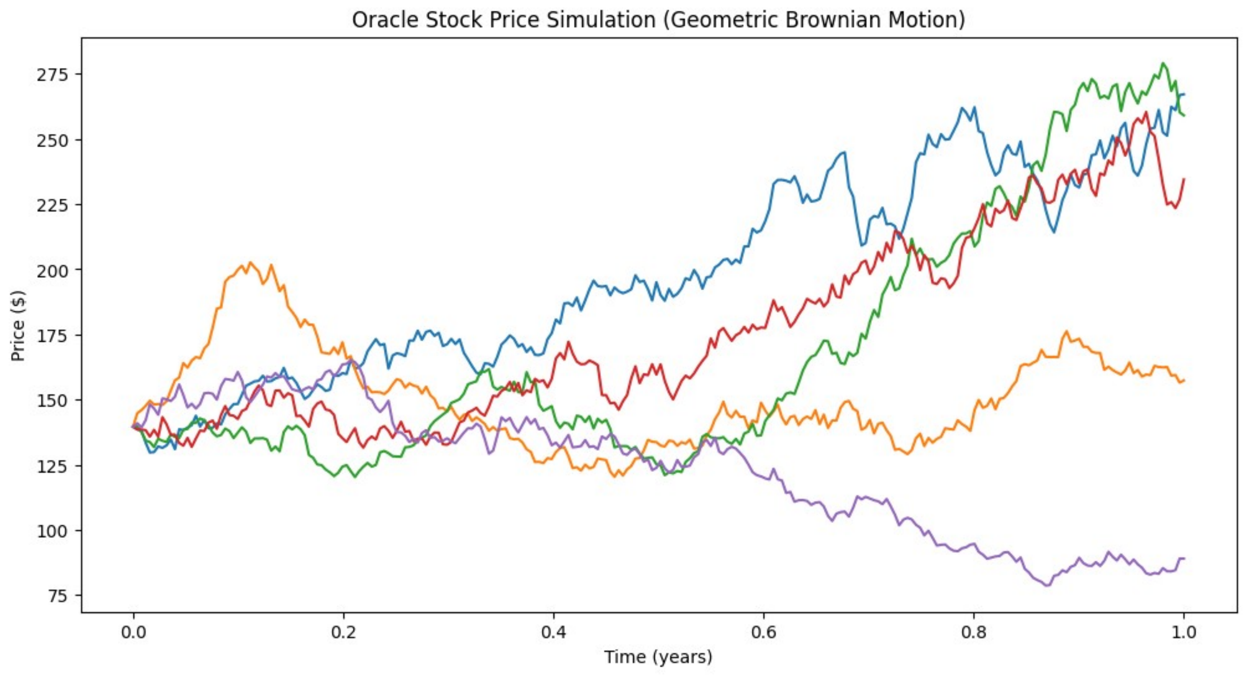
```python
# Simulate quarterly revenue data
dates = pd.date_range(start='2020-01-01', periods=16, freq='Q')
revenue = np.array([9.6, 9.8, 9.7, 10.1, 10.3, 10.6, 11.0, 11.2, 11.5,
11.8, 12.3, 12.6, 13.0, 13.4, 13.8, 14.2])  # in billions
revenue_df = pd.DataFrame({'Date': dates, 'Revenue': revenue})

# Prepare data for time series forecasting
revenue_df.set_index('Date', inplace=True)

# Fit ARIMA model
model = ARIMA(revenue_df, order=(1,1,1))
model_fit = model.fit()

# Forecast next 4 quarters
forecast = model_fit.forecast(steps=4)

# Plot results
plt.figure(figsize=(12,6))
plt.plot(revenue_df.index, revenue_df['Revenue'], label='Actual
Revenue')
plt.plot(forecast.index, forecast, label='Forecasted Revenue')
plt.title('Oracle Quarterly Revenue Forecast (ARIMA)')
plt.xlabel('Date')
plt.ylabel('Revenue ($B)')
plt.legend()
plt.show()
```

```
<ipython-input-6-5be0a54e314d>:2: FutureWarning: 'Q' is deprecated and
will be removed in a future version, please use 'QE' instead.
  dates = pd.date_range(start='2020-01-01', periods=16, freq='Q')
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: No frequency information was provided, so
inferred frequency QE-DEC will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: No frequency information was provided, so
inferred frequency QE-DEC will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: No frequency information was provided, so
inferred frequency QE-DEC will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/
sarimax.py:966: UserWarning: Non-stationary starting autoregressive
parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
```



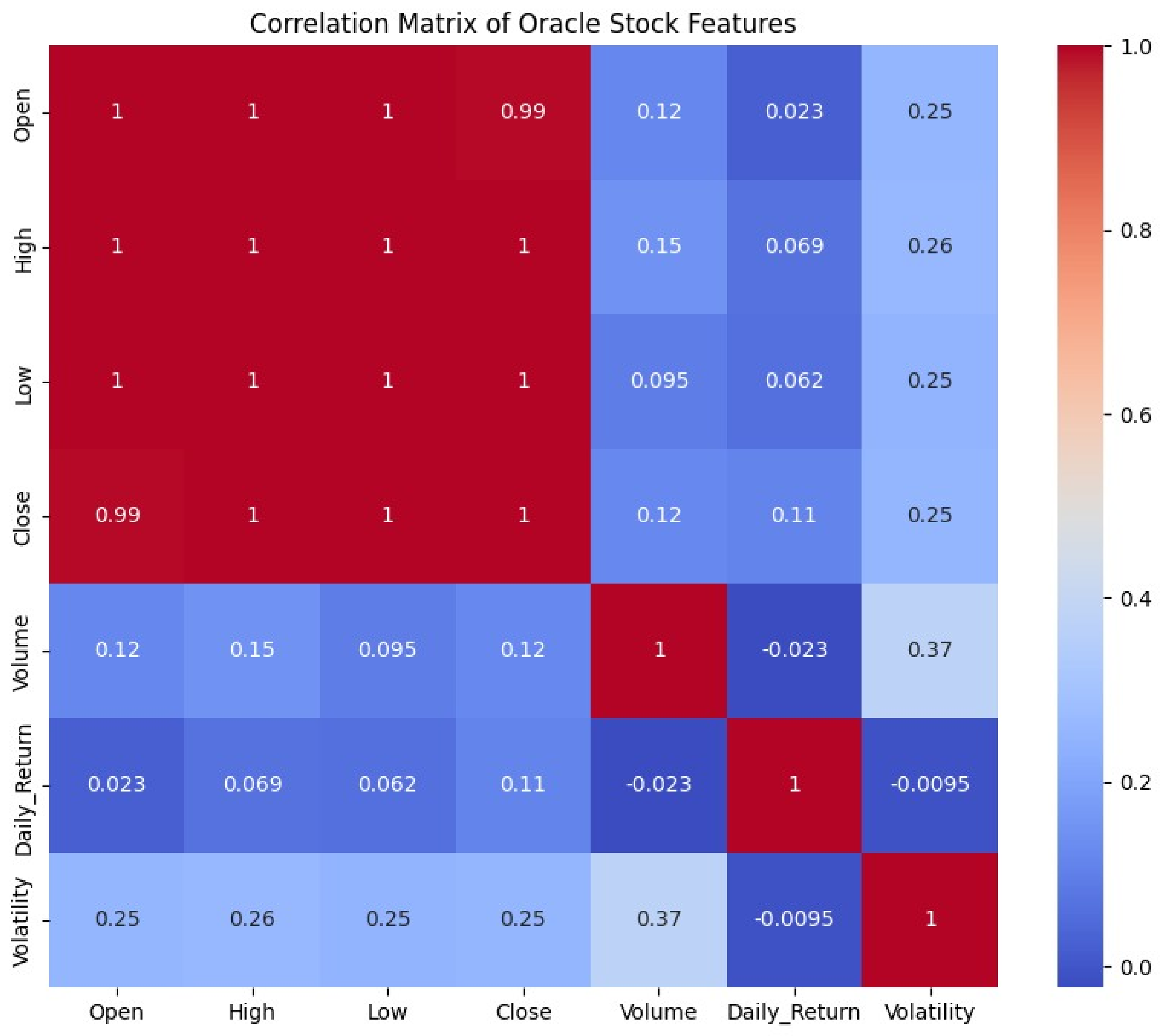
1. Stochastic Process - Geometric Brownian Motion Simulation

```python
def gbm_simulation(S0, mu, sigma, T=1, N=252, n_sim=5):
    dt = T/N
    t = np.linspace(0, T, N)
    S = np.zeros((N, n_sim))
    S[0] = S0

    for i in range(1, N):
        z = np.random.normal(0, 1, n_sim)
        S[i] = S[i-1] * np.exp((mu - 0.5 * sigma**2) * dt + sigma *
np.sqrt(dt) * z)

    return t, S

# Parameters
S0 = df['Close'].iloc[-1]  # Last closing price
mu = df['Daily_Return'].mean() * 252  # Annualized return
sigma = df['Daily_Return'].std() * np.sqrt(252)  # Annualized
volatility

# Run simulation
t, S = gbm_simulation(S0, mu, sigma, T=1, N=252, n_sim=5)

# Plot simulations
plt.figure(figsize=(12,6))
for i in range(5):
    plt.plot(t, S[:,i])
plt.title('Oracle Stock Price Simulation (Geometric Brownian Motion)')
plt.xlabel('Time (years)')
plt.ylabel('Price ($)')
plt.show()
```

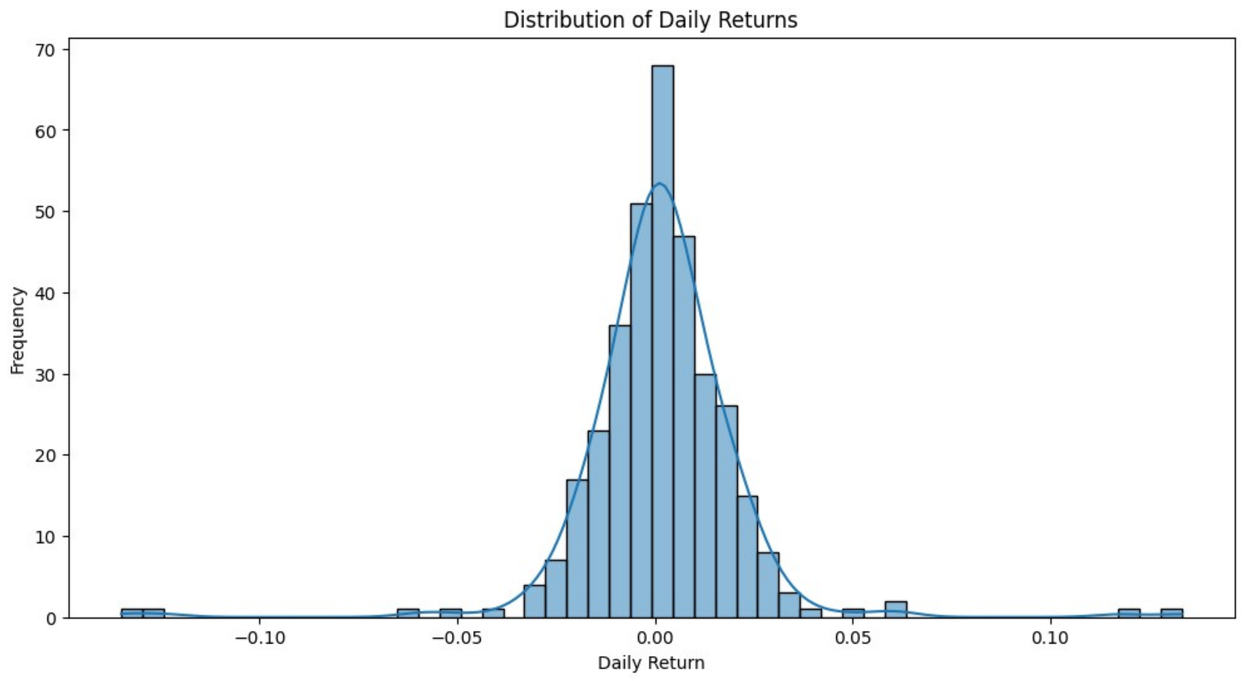


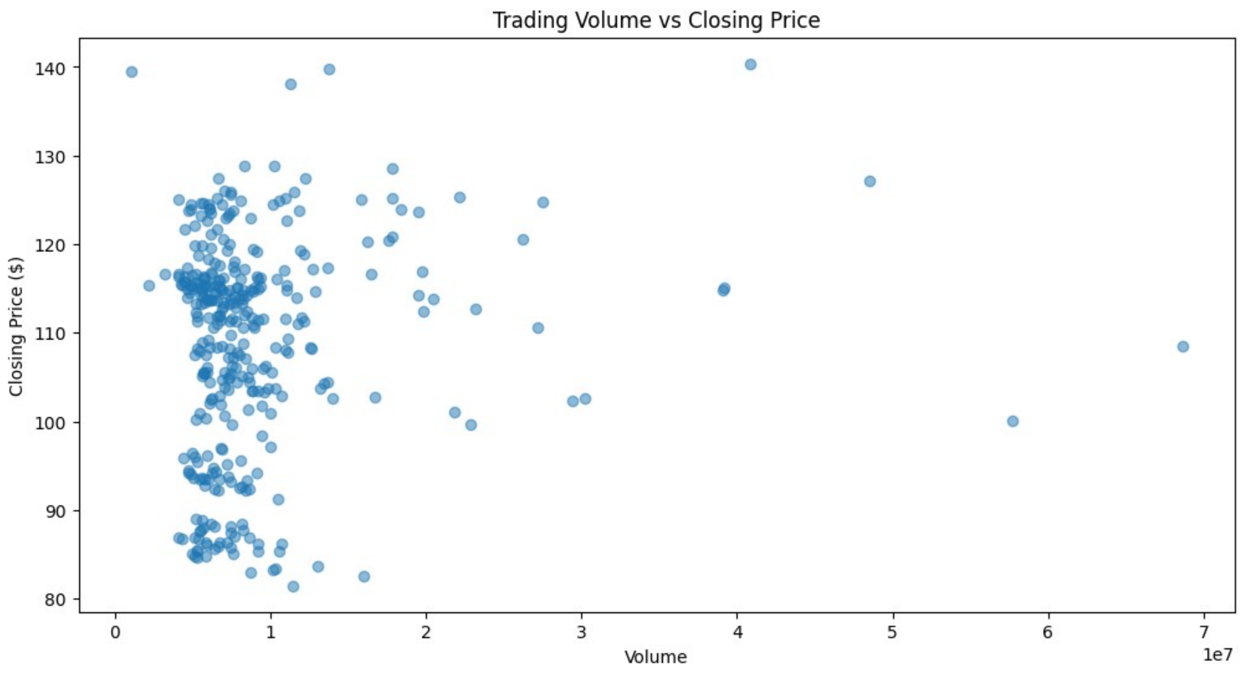
1. Exploratory Data Analysis

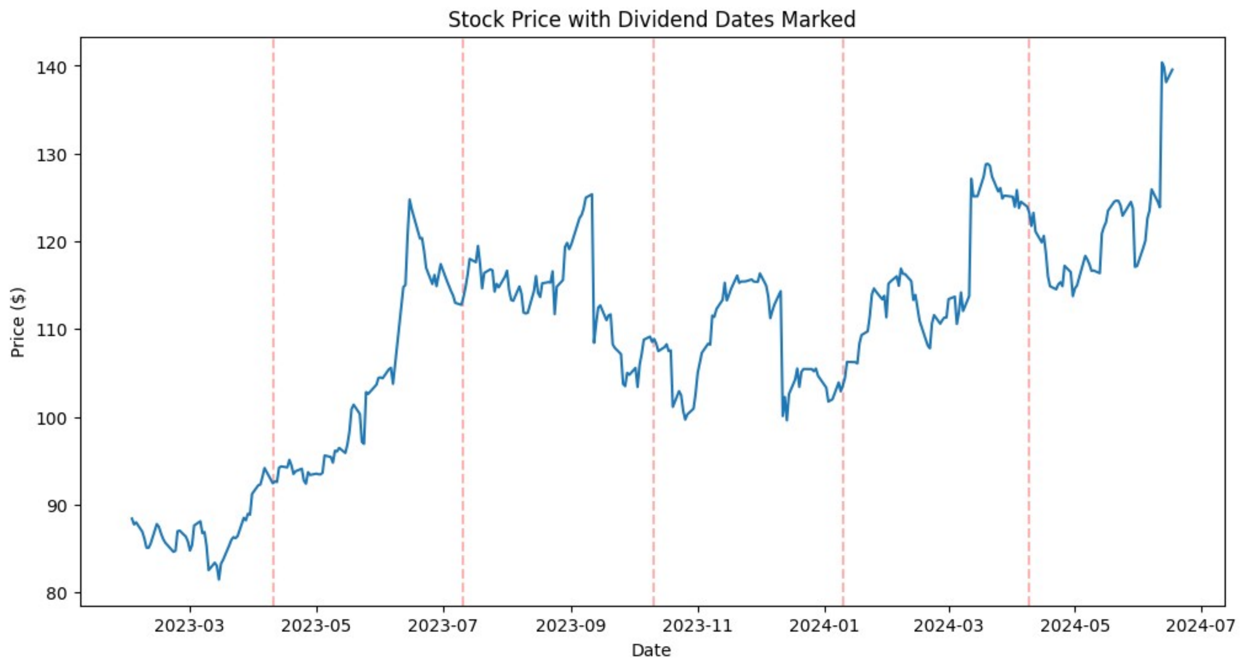
```
# Correlation matrix
corr = df[['Open', 'High', 'Low', 'Close', 'Volume', 'Daily_Return', 
'Volatility']].corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Oracle Stock Features')
plt.show()

# Daily returns distribution
plt.figure(figsize=(12,6))
sns.histplot(df['Daily_Return'], kde=True, bins=50)
plt.title('Distribution of Daily Returns')
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.show()

# Volume vs Price
plt.figure(figsize=(12,6))
plt.scatter(df['Volume'], df['Close'], alpha=0.5)
plt.title('Trading Volume vs Closing Price')
plt.xlabel('Volume')
plt.ylabel('Closing Price ($)')
plt.show()

# Dividend analysis (when dividends occur)
dividend_dates = df[df['Dividends'] > 0].index
if len(dividend_dates) > 0:
    plt.figure(figsize=(12,6))
    for date in dividend_dates:
        plt.axvline(date, color='r', linestyle='--', alpha=0.3)
    plt.plot(df['Close'])
    plt.title('Stock Price with Dividend Dates Marked')
    plt.xlabel('Date')
    plt.ylabel('Price ($)')
    plt.show()
```



	Open	High	Low	Close	Volume	Daily_Return	Volatility
Open	1	1	1	0.99	0.12	0.023	0.25
High	1	1	1	1	0.15	0.069	0.26
Low	1	1	1	1	0.095	0.062	0.25
Close	0.99	1	1	1	0.12	0.11	0.25
Volume	0.12	0.15	0.095	0.12	1	-0.023	0.37
Daily_Return	0.023	0.069	0.062	0.11	-0.023	1	-0.0095
Volatility	0.25	0.26	0.25	0.25	0.37	-0.0095	1


Distribution of Daily Returns
Frequency
Daily Return
0
10
20
30
40
50
60
70
−0.10
−0.05
0.00
0.05
0.10


Trading Volume vs Closing Price
Closing Price ($)
Volume
80
90
100
110
120
130
140
0
1
2
3
4
5
6
7
1e7



Interpretation of Oracle Stock and Financial Analysis

1. Data Preparation and Feature Engineering

The dataset contains Oracle's stock prices and related attributes (e.g., Open, High, Low, Close, Volume, Dividends) from January 2023 to June 2024. Additional financial features such as daily return, log return, moving averages (5-day and 20-day), and volatility were engineered to capture trends and risk.

The rolling features helped visualize patterns over time. For instance, moving averages smooth the price trend, while volatility quantifies the stock's risk.

---

2. Stock Price Prediction

Linear Regression Model:

- Inputs: Open, High, Low, Volume
- Output: Close
- Performance:
  - **MSE = 0.46**, **R^2 = 0.99**
  - Interpretation: The model performs exceptionally well, suggesting that the closing price is highly predictable from the other price-related features. However, this could also indicate overfitting due to potential multicollinearity and lack of non-linear patterns.

ARIMA Model:

- Time series model using past closing prices.
- Performance:
  - **MSE = 107.04**
  - Interpretation: ARIMA performed poorly compared to linear regression. A key reason could be the unsuitability of ARIMA for this dataset's index type (as warnings indicated), and possibly the volatile nature of stock data that ARIMA struggles to model without adjustments.

---

3. Credit Risk Modeling (Simulated Data)

A logistic regression was used to classify defaults using simulated financial ratios.

- **Precision/Recall for Class 1 (Defaults): 0.00**
- Interpretation:
  - The model completely failed to predict any defaults, possibly due to class imbalance (only 15% default rate).
  - This is a classic case of an imbalanced classification problem where most predictions favored the majority class (non-default), rendering the model ineffective for risk detection.

---

4. Revenue Forecasting

Using an ARIMA model, Oracle's simulated quarterly revenue from 2020 to 2024 was projected for four future quarters.

- Interpretation:
  - The revenue shows a steady upward trend.
  - The ARIMA model seems to follow this trend well in forecasting, but lacks uncertainty intervals, which are crucial in real-world financial forecasting.

---

5. Geometric Brownian Motion (GBM) Simulation

This stochastic model simulated five possible future stock price paths over one year.

- Parameters:
  - Starting price = Latest close
  - μ = Mean daily return (annualized)
  - σ = Daily volatility (annualized)
- Interpretation:
  - GBM captures the randomness and volatility of future stock prices.

   - It’s useful for visualizing uncertainty in price movements and for options pricing, but it assumes constant volatility and drift, which may not hold in real markets.

---

6. Exploratory Data Analysis (EDA)

- **Correlation Matrix:**
   - Strong positive correlations among price-related features (Open, High, Low, Close).
   - Daily return has low correlation with most features, indicating it captures different market dynamics.
- **Return Distribution:**
   - Slightly skewed distribution with heavy tails, reflecting occasional extreme gains or losses.
- **Volume vs. Price:**
   - No clear pattern, suggesting that volume alone doesn’t predict closing price effectively.
- **Dividend Dates:**
   - Highlighted on price trends to observe market response. Some correlation between dividends and price stability or spikes is possible.

Conclusion

The analysis successfully integrates classical models (Linear Regression, ARIMA, Logistic Regression) with modern financial metrics and simulations (GBM). While the Linear Regression model excelled in stock price prediction, the ARIMA model struggled without preprocessing for time series index support. The credit risk model highlights the dangers of unbalanced datasets, and simulations like GBM and ARIMA offer insights into price and revenue forecasting under uncertainty.