```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Create a sample dataset
np.random.seed(42)
data = {
    'ID': range(1, 101),
    'Age': np.random.randint(20, 70, size=100),
    'Salary': np.append(np.random.randint(30000, 100000, size=95), [1000000, 1000000, -1000, 500, 600]),
    'Department': np.random.choice(['HR', 'Finance', 'IT', 'hr', 'finance', 'it'], size=100),
    'JoinDate': pd.date_range(start='1/1/2020', periods=100, freq='M'),
    'PerformanceScore': np.random.choice([1, 2, 3, 4, np.nan], size=100)
}

df = pd.DataFrame(data)

# Introduce some missing values
df.loc[5:10, 'Age'] = np.nan
df.loc[15:20, 'Salary'] = np.nan
df.loc[25:30, 'Department'] = np.nan

# Initial Inspection
print("Initial Dataset Info:")
print(df.info())
print("\nInitial Dataset Description:")
print(df.describe())
print("\nFirst Few Rows of the Initial Dataset:")
print(df.head())

# Handling Missing Values
print("\nHandling Missing Values:")
# Impute numerical columns with mean
num_cols = df.select_dtypes(include=np.number).columns
```

0s completed at 1:08 PM

+ Code + Text All changes saved

```python
# Handling Missing Values
print("\nHandling Missing Values:")
# Impute numerical columns with mean
num_cols = df.select_dtypes(include=np.number).columns
imputer_num = SimpleImputer(strategy='mean')
df[num_cols] = imputer_num.fit_transform(df[num_cols])

# Impute categorical columns with most frequent value
cat_cols = df.select_dtypes(include='object').columns
imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat_cols] = imputer_cat.fit_transform(df[cat_cols])

print("\nMissing Values After Imputation:")
print(df.isnull().sum())

# Outlier Detection and Treatment
print("\nDetecting and Handling Outliers:")
z_scores = np.abs(stats.zscore(df[num_cols]))
outliers = np.where(z_scores > 3, True, False)
df_no_outliers = df[~outliers.any(axis=1)]

print("\nDataset Shape Before Removing Outliers:", df.shape)
print("Dataset Shape After Removing Outliers:", df_no_outliers.shape)

# Normalization/Standardization
print("\nNormalizing/Standardizing Numerical Columns:")
scaler = MinMaxScaler()
df_scaled = df_no_outliers.copy()
df_scaled[num_cols] = scaler.fit_transform(df_scaled[num_cols])

# Handling Inconsistencies in Categorical Data
print("\nHandling Inconsistencies in Categorical Data:")
df_scaled['Department'] = df_scaled['Department'].str.strip().str.lower()
df_scaled['Department'] = df_scaled['Department'].replace({'hr': 'HR', 'finance': 'Finance', 'it': 'IT'})

print("\nFinal Cleaned Dataset Info:")
print(df_scaled.info())
print("\nFinal Cleaned Dataset Description:")
print(df_scaled.describe())
```

0s completed at 1:08 PM

+ Code + Text All changes saved

```python
df_scaled['Department'] = df_scaled['Department'].str.strip().str.lower()
df_scaled['Department'] = df_scaled['Department'].replace({'hr': 'HR', 'finance': 'Finance', 'it': 'IT'})

print("\nFinal Cleaned Dataset Info:")
print(df_scaled.info())
print("\nFinal Cleaned Dataset Description:")
print(df_scaled.describe())
print("\nFirst Few Rows of the Final Cleaned Dataset:")
print(df_scaled.head())

# Save the cleaned dataset
df_scaled.to_csv('cleaned_data.csv', index=False)
print("\nCleaned dataset saved to 'cleaned_data.csv'")
```

```
Initial Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                100 non-null    int64
 1   Age               94 non-null     float64
 2   Salary            94 non-null     float64
 3   Department        94 non-null     object
 4   JoinDate          100 non-null    datetime64[ns]
 5   PerformanceScore  80 non-null     float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(1)
memory usage: 4.8+ KB
None

Initial Dataset Description:
               ID        Age          Salary             JoinDate  \
count  100.000000  94.000000       94.000000                  100
mean    50.500000  44.351064    85033.787234  2024-03-15 18:28:48
min      1.000000  20.000000    -1000.000000  2020-01-31 00:00:00
25%     25.750000  33.000000    48070.500000  2022-02-21 00:00:00
50%     50.500000  44.000000    69006.500000  2024-03-15 12:00:00
75%     75.250000  58.000000    85785.000000  2026-04-07 12:00:00
max    100.000000  69.000000  1000000.000000  2028-04-30 00:00:00
std     29.011492  14.668163   137762.627188                  NaN

       PerformanceScore
```

0s completed at 1:08 PM

Scanned with CamScanner

+ Code + Text All changes saved

```
print("\nCleaned dataset saved to 'cleaned_data.csv'")
```

```
 5   PerformanceScore  80 non-null     float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(1)
memory usage: 4.8+ KB
None

Initial Dataset Description:
               ID        Age          Salary             JoinDate  \
count  100.000000  94.000000       94.000000                  100
mean    50.500000  44.351064    85033.787234  2024-03-15 18:28:48
min      1.000000  20.000000    -1000.000000  2020-01-31 00:00:00
25%     25.750000  33.000000    48070.500000  2022-02-21 00:00:00
50%     50.500000  44.000000    69006.500000  2024-03-15 12:00:00
75%     75.250000  58.000000    85785.000000  2026-04-07 12:00:00
max    100.000000  69.000000  1000000.000000  2028-04-30 00:00:00
std     29.011492  14.668163   137762.627188                  NaN

       PerformanceScore
count         80.000000
mean           2.525000
min            1.000000
25%            2.000000
50%            2.000000
75%            4.000000
max            4.000000
std            1.147093

First Few Rows of the Initial Dataset:
   ID   Age   Salary Department   JoinDate  PerformanceScore
0   1  58.0  32568.0    Finance 2020-01-31               NaN
1   2  48.0  92592.0         it 2020-02-29               1.0
2   3  34.0  97563.0         it 2020-03-31               4.0
3   4  62.0  32695.0         IT 2020-04-30               1.0
4   5  27.0  78190.0         HR 2020-05-31               NaN

Handling Missing Values:

Missing Values After Imputation:
ID                  0
Age                 0
Salary              0
Department          0
JoinDate            0
```

0s completed at 1:08 PM

+ Code + Text All changes saved

```
print("\nCleaned dataset saved to 'cleaned_data.csv'")
```

```
Handling Missing Values:

Missing Values After Imputation:
ID                  0
Age                 0
Salary              0
Department          0
JoinDate            0
PerformanceScore    0
dtype: int64

Detecting and Handling Outliers:

Dataset Shape Before Removing Outliers: (100, 6)
Dataset Shape After Removing Outliers: (98, 6)

Normalizing/Standardizing Numerical Columns:

Handling Inconsistencies in Categorical Data:

Final Cleaned Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 98 entries, 0 to 99
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                98 non-null     float64
 1   Age               98 non-null     float64
 2   Salary            98 non-null     float64
 3   Department        98 non-null     object
 4   JoinDate          98 non-null     datetime64[ns]
 5   PerformanceScore  98 non-null     float64
dtypes: datetime64[ns](1), float64(4), object(1)
memory usage: 5.4+ KB
None

Final Cleaned Dataset Description:
              ID        Age     Salary                       JoinDate  \
count  98.000000  98.000000  98.000000                             98
mean    0.490517   0.495024   0.672514  2024-02-16 04:24:29.387755008
min     0.000000   0.000000   0.000000            2020-01-31 00:00:00
```

0s completed at 1:08 PM

+ Code + Text All changes saved

```
 1   Age               98 non-null     float64
 2   Salary            98 non-null     float64
 3   Department        98 non-null     object
 4   JoinDate          98 non-null     datetime64[ns]
 5   PerformanceScore  98 non-null     float64
dtypes: datetime64[ns](1), float64(4), object(1)
memory usage: 5.4+ KB
None

Final Cleaned Dataset Description:
              ID        Age     Salary                       JoinDate  \
count  98.000000  98.000000  98.000000                             98
mean    0.490517   0.495024   0.672514  2024-02-16 04:24:29.387755008
min     0.000000   0.000000   0.000000            2020-01-31 00:00:00
25%     0.244949   0.270408   0.499828            2022-02-07 00:00:00
50%     0.489899   0.493378   0.719133            2024-02-14 12:00:00
75%     0.734848   0.750000   0.858938            2026-02-21 00:00:00
max     1.000000   1.000000   1.000000            2028-04-30 00:00:00
std     0.288277   0.289471   0.240970                            NaN

       PerformanceScore
count         98.000000
mean           0.505102
min            0.000000
25%            0.333333
50%            0.508333
75%            0.666667
max            1.000000
std            0.340960

First Few Rows of the Final Cleaned Dataset:
         ID       Age    Salary Department   JoinDate  PerformanceScore
0  0.000000  0.775510  0.335134    Finance 2020-01-31          0.508333
1  0.010101  0.571429  0.934397         IT 2020-02-29          0.000000
2  0.020202  0.285714  0.984026         IT 2020-03-31          1.000000
3  0.030303  0.857143  0.336402         IT 2020-04-30          0.000000
4  0.040404  0.142857  0.790611         HR 2020-05-31          0.508333

Cleaned dataset saved to 'cleaned_data.csv'
```

0s completed at 1:08 PM

+ Code + Text All changes saved

RAM Disk ✦ Gemini

```
pip install pandas numpy statsmodels matplotlib
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.11.4)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (24.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels) (1.16.0)
```

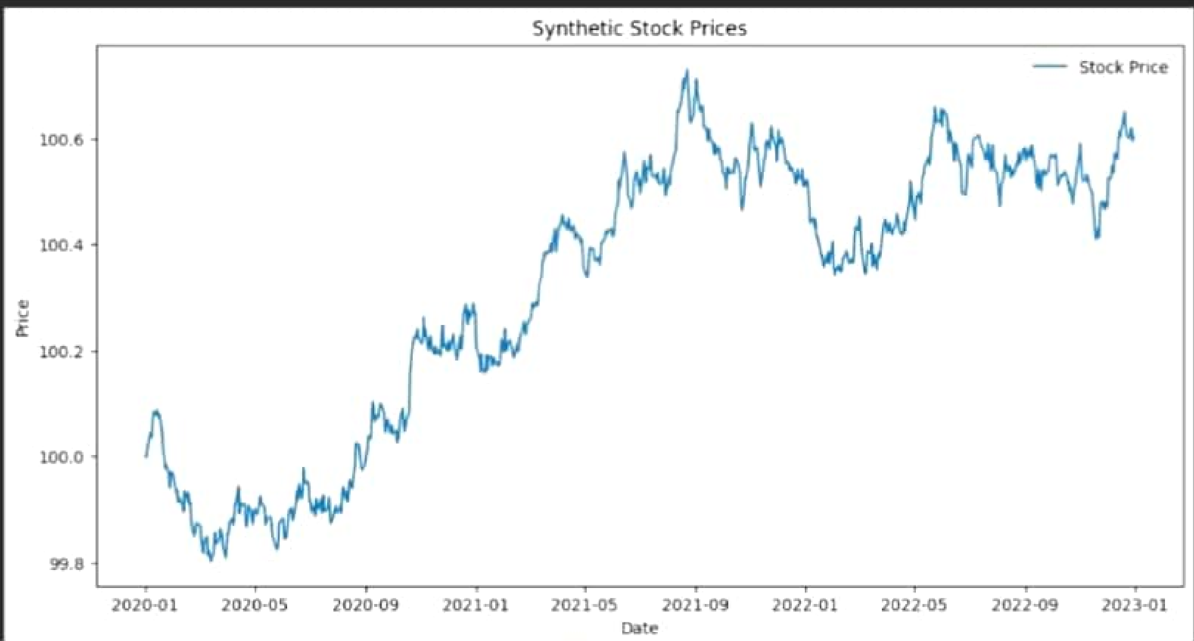
[2]
```
import pandas as pd
import numpy as np

# Generate synthetic stock prices
np.random.seed(42)
dates = pd.date_range(start='2020-01-01', end='2023-01-01', freq='B')
price = np.random.normal(loc=0.001, scale=0.02, size=len(dates))
price[0] = 0
price = 100 + np.cumsum(price)

stock_data = pd.DataFrame({'Date': dates, 'Price': price})
stock_data.set_index('Date', inplace=True)
```

[3]
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
```

✓ 0s completed at 5:41 PM

+ Code + Text All changes saved

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(stock_data.index, stock_data['Price'], label='Stock Price')
plt.title('Synthetic Stock Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```




0s completed at 5:41 PM

+ Code + Text All changes saved

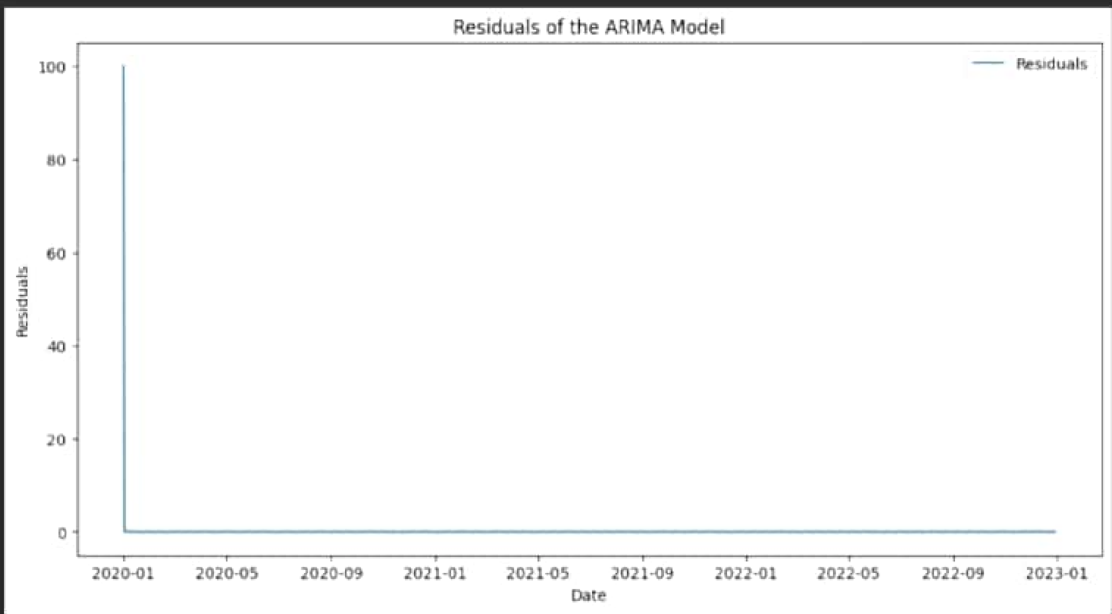
```python
from statsmodels.tsa.arima.model import ARIMA

# Define the ARIMA model
model = ARIMA(stock_data['Price'], order=(1, 1, 1))
model_fit = model.fit()

# Print summary of the model
print(model_fit.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency B
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency B
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency B
  self._init_dates(dates, freq)
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Price   No. Observations:                  783
Model:                 ARIMA(1, 1, 1)   Log Likelihood                1959.018
Date:                Thu, 20 Jun 2024   AIC                          -3912.037
Time:                        12:10:59   BIC                          -3898.051
Sample:                    01-01-2020   HQIC                         -3906.658
                         - 12-30-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0824      1.568     -0.053      0.958      -3.156       2.992
ma.L1          0.0587      1.571      0.037      0.970      -3.020       3.138
sigma2         0.0004   1.92e-05     20.311      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 3.39
Prob(Q):                              1.00   Prob(JB):                         0.18
Heteroskedasticity (H):               1.08   Skew:                             0.15
Prob(H) (two-sided):                  0.55   Kurtosis:                         3.13
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

✓ 0s completed at 5:41 PM

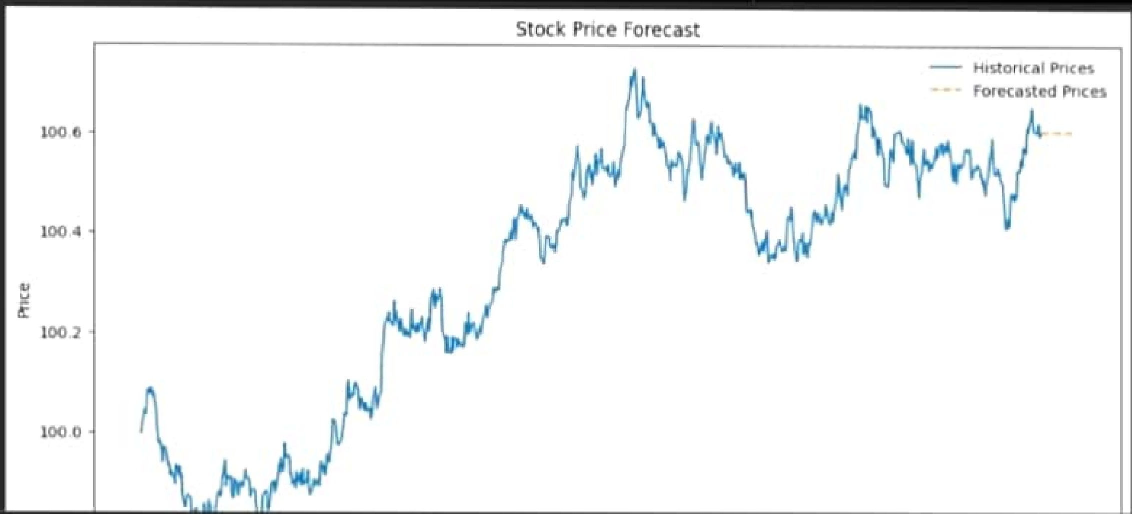
+ Code + Text All changes saved

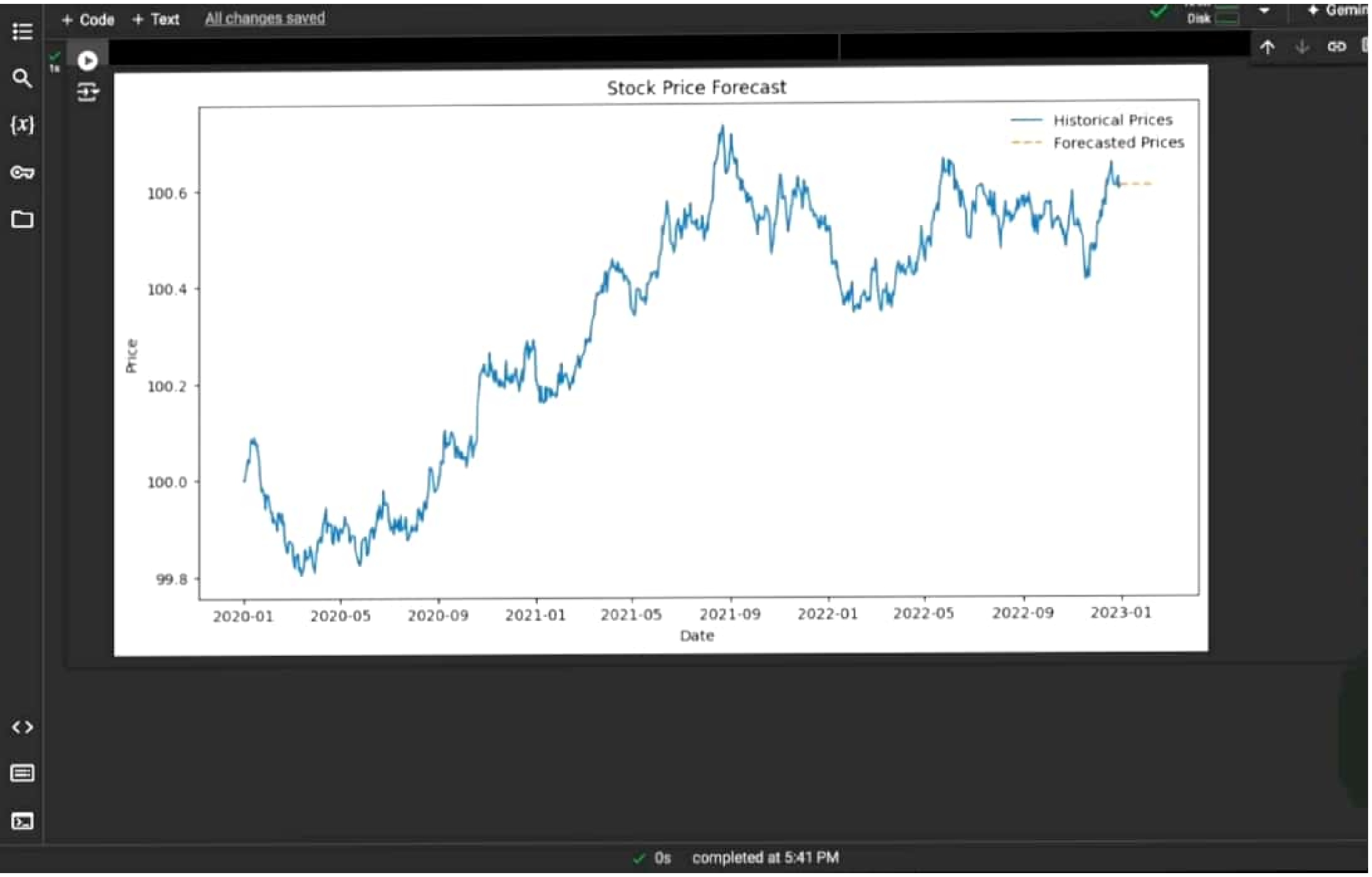
```
residuals = model_fit.resid
plt.figure(figsize=(12, 6))
plt.plot(stock_data.index, residuals, label='Residuals')
plt.title('Residuals of the ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.legend()
plt.show()
```




0s completed at 5:41 PM

```python
forecast = model_fit.forecast(steps=30)  # Forecast for the next 30 business days
forecast_dates = pd.date_range(start=stock_data.index[-1], periods=30, freq='B')
forecast_df = pd.DataFrame({'Date': forecast_dates, 'Forecast': forecast})
forecast_df.set_index('Date', inplace=True)

plt.figure(figsize=(12, 6))
plt.plot(stock_data.index, stock_data['Price'], label='Historical Prices')
plt.plot(forecast_df.index, forecast_df['Forecast'], label='Forecasted Prices', linestyle='--')
plt.title('Stock Price Forecast')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```




+ Code
+ Text
All changes saved
Stock Price Forecast
Historical Prices
Forecasted Prices
100.6
100.4
100.2
100.0
99.8
Price
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09
2023-01
Date
0s
completed at 5:41 PM

+ Code + Text All changes saved

```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Sample transactional data
data = {
    'TransactionID': [1, 2, 3, 4, 5],
    'Items': [
        ['milk', 'bread', 'butter'],
        ['bread', 'butter', 'eggs'],
        ['milk', 'bread', 'eggs'],
        ['milk', 'butter', 'eggs'],
        ['bread', 'butter']
    ]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Extract list of transactions
transactions = df['Items'].tolist()

# Initialize transaction encoder
te = TransactionEncoder()

# Transform the list of transactions into a one-hot encoded DataFrame
te_ary = te.fit(transactions).transform(transactions)
df_transformed = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm
frequent_itemsets = apriori(df_transformed, min_support=0.6, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

print(frequent_itemsets)
print(rules)
```

✓ 0s completed at 1:56 PM

+ Code + Text All changes saved

RAM Disk ✦ Gemini

```python
# Transform the list of transactions into a one-hot encoded DataFrame
te_ary = te.fit(transactions).transform(transactions)
df_transformed = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm
frequent_itemsets = apriori(df_transformed, min_support=0.6, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

print(frequent_itemsets)
print(rules)
```

```
   support         itemsets
0      0.8          (bread)
1      0.8         (butter)
2      0.6           (eggs)
3      0.6           (milk)
4      0.6  (butter, bread)
Empty DataFrame
Columns: [antecedents, consequents, antecedent support, consequent support, support, confidence, lift, leverage, conviction, zhangs_metric]
Index: []
```

+ Code + Text All changes saved

Disk ✦ Gemini

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# Sample dataset creation
data = {
    'customerID': ['C001', 'C002', 'C003', 'C004', 'C005', 'C006', 'C007', 'C008', 'C009', 'C010'],
    'gender': ['Female', 'Male', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Male', 'Female'],
    'SeniorCitizen': [0, 1, 0, 0, 1, 1, 0, 0, 0, 1],
    'Partner': ['Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes'],
    'Dependents': ['No', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No'],
    'tenure': [1, 34, 2, 45, 5, 2, 8, 22, 10, 30],
    'PhoneService': ['No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes'],
    'InternetService': ['DSL', 'Fiber optic', 'DSL', 'Fiber optic', 'DSL', 'DSL', 'Fiber optic', 'DSL', 'DSL', 'Fiber optic'],
    'Contract': ['Month-to-month', 'One year', 'Month-to-month', 'One year', 'Month-to-month', 'Month-to-month', 'One year', 'Month-to-month', 'Month-to-mo
    'MonthlyCharges': [29.85, 56.95, 53.85, 42.30, 70.70, 53.85, 99.65, 89.10, 29.75, 49.95],
    'TotalCharges': [29.85, 1889.50, 108.15, 1840.75, 151.65, 108.15, 820.50, 1949.40, 301.90, 1490.75],
    'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No']
}

# Create DataFrame
df = pd.DataFrame(data)

# Encode categorical variables
df_encoded = df.copy()
label_encoders = {}
for column in ['gender', 'Partner', 'Dependents', 'PhoneService', 'InternetService', 'Contract', 'Churn']:
    le = LabelEncoder()
    df_encoded[column] = le.fit_transform(df_encoded[column])
    label_encoders[column] = le

df_encoded.head()
```

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | InternetService | Contract | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | C001 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 29.85 | 29.85 | 1 |

✓ 0s completed at 12:19 PM

+ Code + Text All changes saved

RAM Disk ◆ Gemini

[1]

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | InternetService | Contract | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | C001 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 29.85 | 29.85 | 1 |
| 1 | C002 | 1 | 1 | 0 | 0 | 34 | 1 | 1 | 1 | 56.95 | 1889.50 | 0 |
| 2 | C003 | 0 | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 53.85 | 108.15 | 1 |
| 3 | C004 | 1 | 0 | 0 | 0 | 45 | 0 | 1 | 1 | 42.30 | 1840.75 | 0 |
| 4 | C005 | 1 | 1 | 0 | 0 | 5 | 1 | 0 | 0 | 70.70 | 151.65 | 1 |

Next steps: Generate code with df_encoded View recommended plots

```python
# Define features and target variable
X = df_encoded.drop(['customerID', 'Churn'], axis=1)
y = df_encoded['Churn']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[3]

```python
scaler = StandardScaler()
X_train[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit_transform(X_train[['tenure', 'MonthlyCharges', 'TotalCharges']])
X_test[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.transform(X_test[['tenure', 'MonthlyCharges', 'TotalCharges']])
```

[4]

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

# Build and train the logistic regression model
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train, y_train)

# Predict on test data
y_pred_logreg = logreg.predict(X_test)
```

✓ 0s completed at 12:19 PM

\+ Code + Text All changes saved

```python
# Evaluate the logistic regression model
logreg_accuracy = accuracy_score(y_test, y_pred_logreg)
logreg_conf_matrix = confusion_matrix(y_test, y_pred_logreg)
logreg_class_report = classification_report(y_test, y_pred_logreg)

print("Logistic Regression Accuracy:", logreg_accuracy)
print("Confusion Matrix:\n", logreg_conf_matrix)
print("Classification Report:\n", logreg_class_report)
```

```
Logistic Regression Accuracy: 0.5
Confusion Matrix:
 [[1 1]
 [0 0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.00      0.00      0.00         0

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to
  _warn_prf(average, modifier, msg_start, len(result))
```

[5]

```python
from sklearn.tree import DecisionTreeClassifier

# Build and train the decision tree model
tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

# Predict on test data
y_pred_tree = tree.predict(X_test)
```

✓ 0s completed at 12:19 PM

+ Code + Text All changes saved

```
# Evaluate the decision tree model
tree_accuracy = accuracy_score(y_test, y_pred_tree)
tree_conf_matrix = confusion_matrix(y_test, y_pred_tree)
tree_class_report = classification_report(y_test, y_pred_tree)

print("Decision Tree Accuracy:", tree_accuracy)
print("Confusion Matrix:\n", tree_conf_matrix)
print("Classification Report:\n", tree_class_report)
```

```
Decision Tree Accuracy: 1.0
Confusion Matrix:
 [[2]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

```
# Feature importance from the decision tree model
feature_importances = pd.DataFrame(tree.feature_importances_, index=X_train.columns, columns=['importance']).sort_values('importance', ascending=False)
print(feature_importances)
```

```
                  importance
tenure                   1.0
gender                   0.0
SeniorCitizen            0.0
Partner                  0.0
Dependents               0.0
PhoneService             0.0
InternetService          0.0
Contract                 0.0
MonthlyCharges           0.0
TotalCharges             0.0
```

✓ 0s completed at 12:19 PM

+ Code + Text All changes saved

RAM Disk ✦ Gemini

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
import gensim.downloader as api
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd

# Sample dataset
data = {
    "review": [
        "I love this product! It's absolutely amazing.",
        "This is the worst thing I have ever bought. Totally useless.",
        "Not bad, could be better.",
        "Pretty decent product for the price.",
        "Absolutely horrible! Do not buy this.",
        "Fantastic quality and great value for money.",
        "Mediocre, nothing special.",
        "Exceeded my expectations. Highly recommend!",
        "Terrible experience, will never purchase again.",
        "Good, but not great. Satisfied overall."
    ]
}

reviews_df = pd.DataFrame(data)

# Download NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')

def clean_tokenize(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return tokens
```

✓ 1m 10s completed at 3:24 PM

+ Code + Text All changes saved

```
# Apply cleaning and tokenization
reviews_df['tokens'] = reviews_df['review'].apply(clean_tokenize)

# Load pre-trained word vectors
glove_vectors = api.load("glove-wiki-gigaword-50")

def get_word_vectors(tokens):
    vectors = [glove_vectors[word] for word in tokens if word in glove_vectors]
    return vectors

# Apply word embedding
reviews_df['word_vectors'] = reviews_df['tokens'].apply(get_word_vectors)

# Initialize VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

def get_sentiment_score(text):
    sentiment = sia.polarity_scores(text)
    return sentiment

# Apply sentiment analysis
reviews_df['sentiment'] = reviews_df['review'].apply(get_sentiment_score)

# Extract compound sentiment score
reviews_df['compound'] = reviews_df['sentiment'].apply(lambda x: x['compound'])

# Determine overall sentiment
reviews_df['sentiment_label'] = reviews_df['compound'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')

# Aggregate results
sentiment_summary = reviews_df['sentiment_label'].value_counts()
reviews_df, sentiment_summary
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

✓ 1m 10s completed at 3:24 PM

+ Code + Text All changes saved

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[==================================================] 100.0% 66.0/66.0MB downloaded
(                                                review  \
0      I love this product! It's absolutely amazing.
1  This is the worst thing I have ever bought. To...
2                           Not bad, could be better.
3                  Pretty decent product for the price.
4                 Absolutely horrible! Do not buy this.
5          Fantastic quality and great value for money.
6                           Mediocre, nothing special.
7            Exceeded my expectations. Highly recommend!
8       Terrible experience, will never purchase again.
9                Good, but not great. Satisfied overall.

                                           tokens  \
0            [love, product, absolutely, amazing]
1  [worst, thing, ever, bought, totally, useless]
2                             [bad, could, better]
3                  [pretty, decent, product, price]
4                      [absolutely, horrible, buy]
5        [fantastic, quality, great, value, money]
6                     [mediocre, nothing, special]
7      [exceeded, expectations, highly, recommend]
8          [terrible, experience, never, purchase]
9                 [good, great, satisfied, overall]

                                        word_vectors  \
0  [[-0.13886, 1.1401, -0.85212, -0.29212, 0.7553...
1  [[-0.14968, -0.42252, 0.16736, 0.13474, -0.310...
2  [[-0.17981, -0.40407, -0.1653, -0.60687, -0.39...
3  [[-0.24922, -0.39835, -0.45851, -0.34846, 0.74...
4  [[0.36582, -0.43975, -0.35016, 0.096443, 0.995...
5  [[0.3333, 0.30612, -0.63572, 0.051507, 0.78602...
6  [[-1.0406, -0.61579, -0.28125, -0.51557, 0.079...
7  [[-0.32462, -0.079688, 1.2704, -0.55724, 0.029...
8  [[0.33209, -0.028359, -0.58145, -0.4487, 0.254...
9  [[-0.35586, 0.5213, -0.6107, -0.30131, 0.94862...

                                           sentiment  compound sentiment_label
0  {'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'comp...    0.8620        positive
1  {'neg': 0.473, 'neu': 0.527, 'pos': 0.0, 'comp...   -0.8016        negative
2  {'neg': 0.0, 'neu': 0.343, 'pos': 0.657, 'comp...    0.6956        positive
```

1m 10s completed at 3:24 PM

+ Code + Text All changes saved

```
8   Terrible experience, will never purchase again.
9          Good, but not great. Satisfied overall.

                                         tokens  \
0           [love, product, absolutely, amazing]
1  [worst, thing, ever, bought, totally, useless]
2                            [bad, could, better]
3                  [pretty, decent, product, price]
4                       [absolutely, horrible, buy]
5     [fantastic, quality, great, value, money]
6                      [mediocre, nothing, special]
7    [exceeded, expectations, highly, recommend]
8       [terrible, experience, never, purchase]
9               [good, great, satisfied, overall]

                                       word_vectors  \
0  [[-0.13886, 1.1401, -0.85212, -0.29212, 0.7553...
1  [[-0.14968, -0.42252, 0.16736, 0.13474, -0.310...
2  [[-0.17981, -0.40407, -0.1653, -0.60687, -0.39...
3  [[-0.24922, -0.39835, -0.45851, -0.34846, 0.74...
4  [[0.36582, -0.43975, -0.35016, 0.096443, 0.995...
5  [[0.3333, 0.30612, -0.63572, 0.051507, 0.78602...
6  [[-1.0406, -0.61579, -0.28125, -0.51557, 0.079...
7  [[-0.32462, -0.079688, 1.2704, -0.55724, 0.029...
8  [[0.33209, -0.028359, -0.58145, -0.4487, 0.254...
9  [[-0.35586, 0.5213, -0.6107, -0.30131, 0.94862...

                                          sentiment  compound sentiment_label
0  {'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'comp...    0.8620        positive
1  {'neg': 0.473, 'neu': 0.527, 'pos': 0.0, 'comp...   -0.8016        negative
2  {'neg': 0.0, 'neu': 0.343, 'pos': 0.657, 'comp...    0.6956        positive
3  {'neg': 0.0, 'neu': 0.61, 'pos': 0.39, 'compou...    0.4939        positive
4  {'neg': 0.45, 'neu': 0.55, 'pos': 0.0, 'compou...   -0.6230        negative
5  {'neg': 0.0, 'neu': 0.284, 'pos': 0.716, 'comp...    0.8779        positive
6  {'neg': 0.53, 'neu': 0.47, 'pos': 0.0, 'compou...   -0.3089        negative
7  {'neg': 0.0, 'neu': 0.565, 'pos': 0.435, 'comp...    0.4740        positive
8  {'neg': 0.383, 'neu': 0.617, 'pos': 0.0, 'comp...   -0.4767        negative
9  {'neg': 0.6, 'neu': 0.242, 'pos': 0.157, 'comp...   -0.7571        negative  ,
sentiment_label
positive    5
negative    5
Name: count, dtype: int64)
```

✓ 1m 10s completed at 3:24 PM