Task 3

Market Price Prediction

Presented by

DIOUF GABRIEL

2024-05-26

Table of CONTENTS

Data Preprocessing

- Loading data and read it from a CSV file into a Data Frame
- Impute Missing values in (quantity, priceMin, priceMax, priceMod
- Convert Categorical Variables(market, state, city) into numerical format
- Convert date column to datetime format

Data Preprocessing

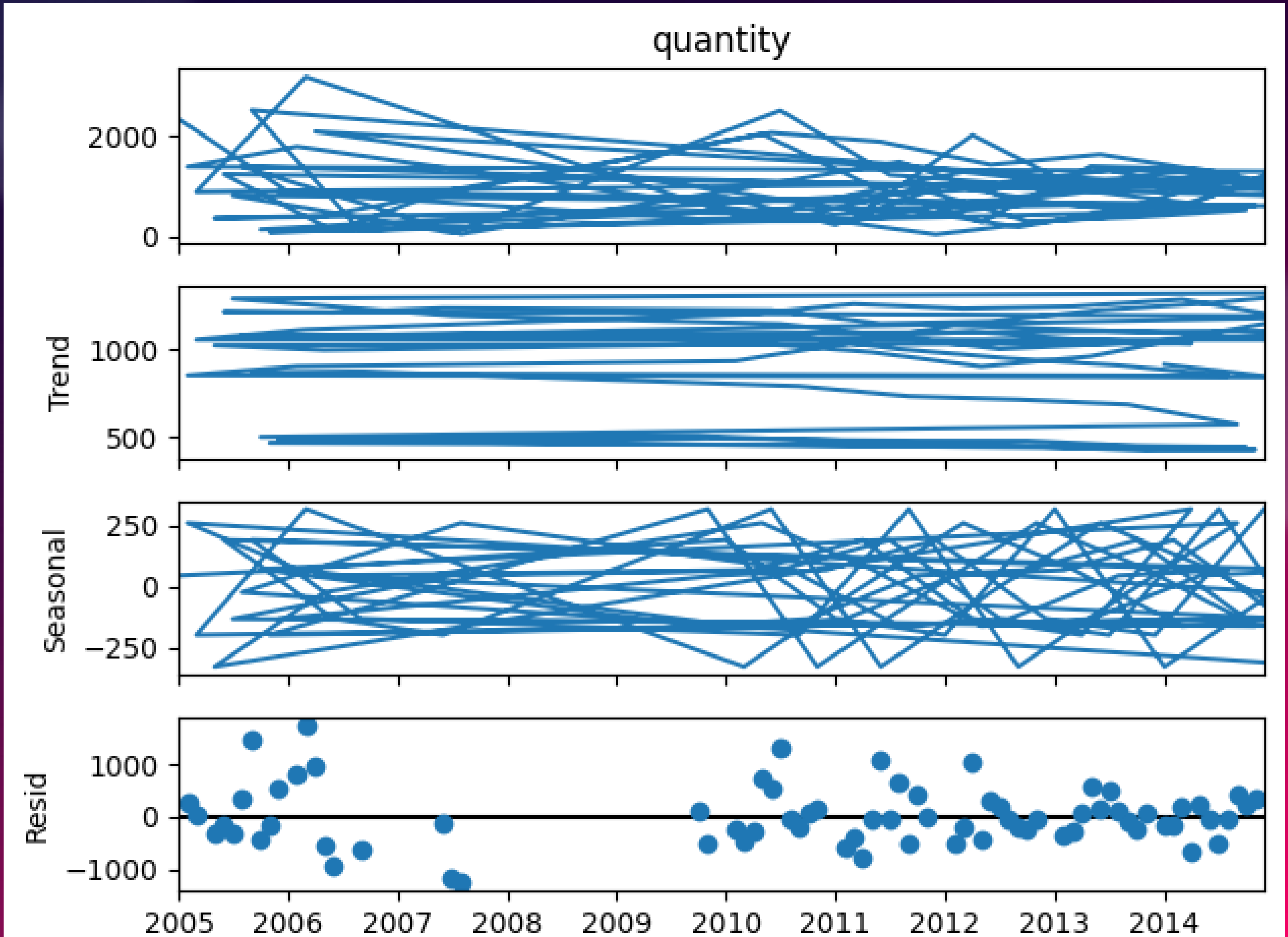
Next steps: View recommended plots

```
# Handle missing values
imputer = SimpleImputer(strategy='mean')
df['quantity'] = imputer.fit_transform(df[['quantity']])
df['priceMin'] = imputer.fit_transform(df[['priceMin']])
df['priceMax'] = imputer.fit_transform(df[['priceMax']])
df['priceMod'] = imputer.fit_transform(df[['priceMod']])
```

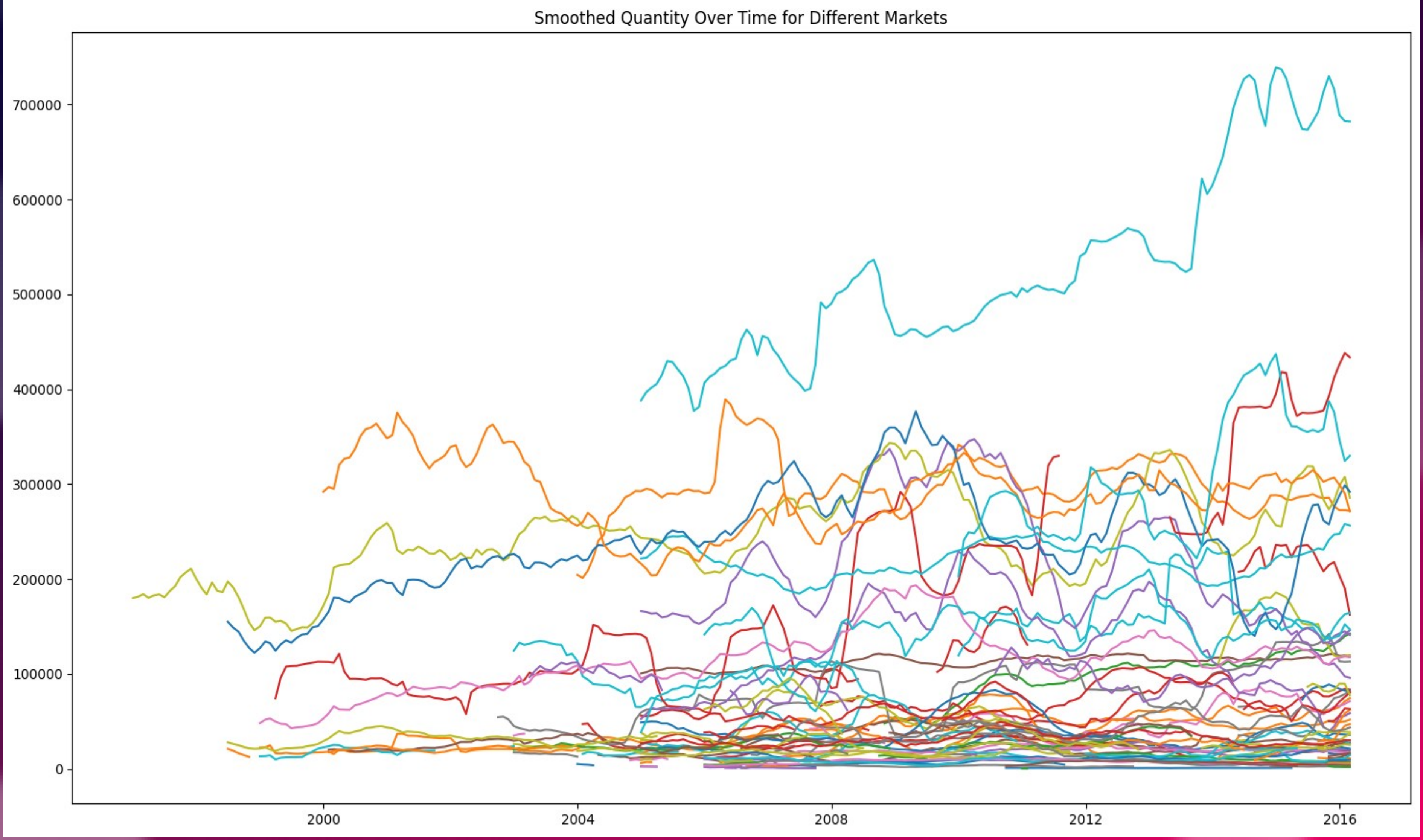
+ Code + Text

```
[61] # Encode categorical variables
le_market = LabelEncoder()
df['market'] = le_market.fit_transform(df['market'])
le_state = LabelEncoder()
df['state'] = le_state.fit_transform(df['state'])
le_city = LabelEncoder()
df['city'] = le_city.fit_transform(df['city'])
```

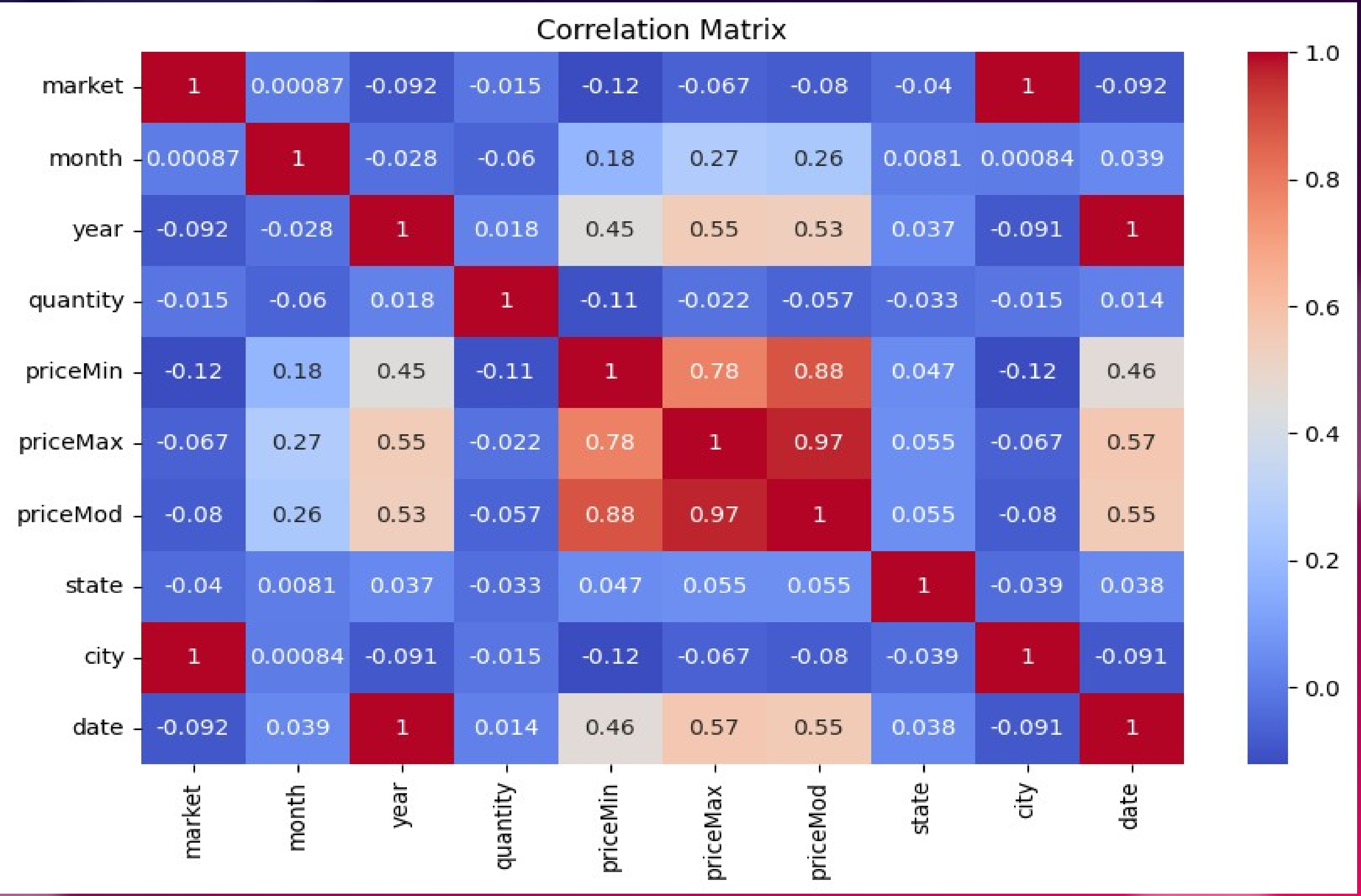
- Exploratory Data Analysis



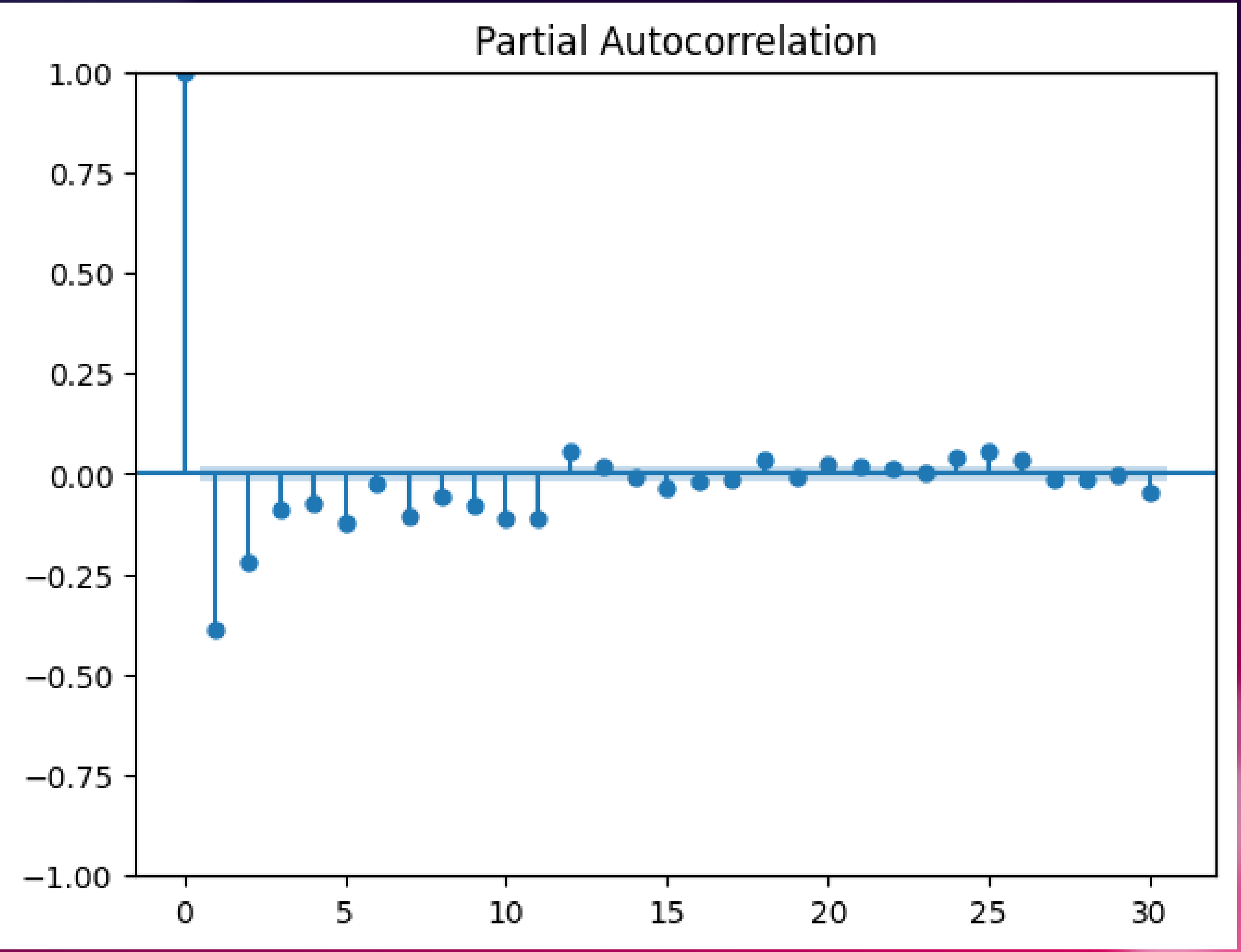
- Exploratory Data Analysis



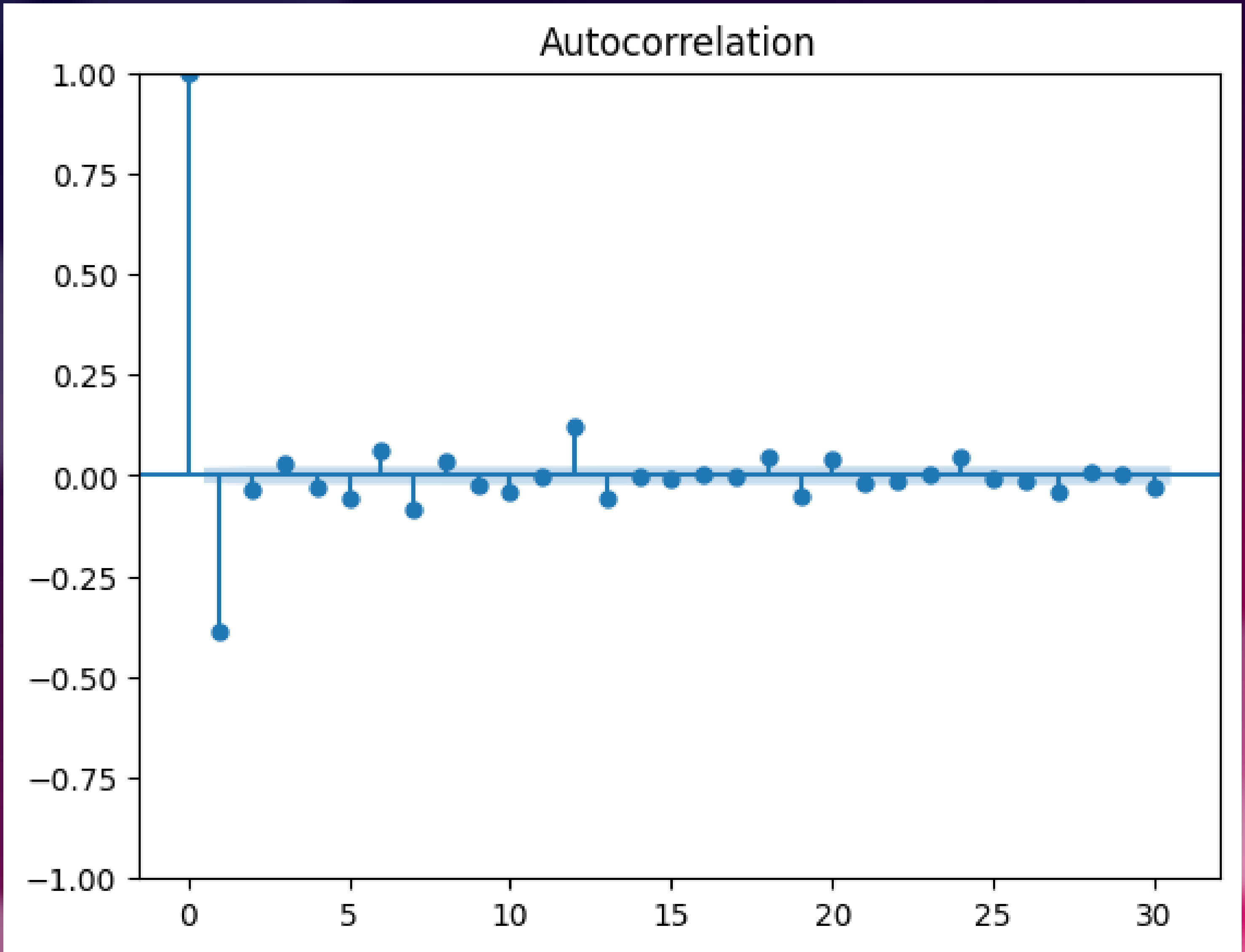
• Exploratory Data Analysis



• Exploratory Data Analysis



- Exploratory Data Analysis



Feature Engineering

- Create lagged feature
- Creat rolling statistics
- Creat seasonal Indicators
- Drop rows with NaN values

Feature Engineering

```
[14] # Create lagged features
     df['quantity_lag1'] = df['quantity'].shift(1)
     df['quantity_lag2'] = df['quantity'].shift(2)
```

```
[15] # Create rolling statistics
     df['quantity_roll_mean'] = df['quantity'].rolling(window=3).mean()
     df['quantity_roll_std'] = df['quantity'].rolling(window=3).std()
```

```
     # Create seasonal indicators
     df = pd.get_dummies(df, columns=['month'], drop_first=True)
```

+ Code + Text

```
[17] # Drop rows with NaN values created by shifting
     df = df.dropna()
     print(df.head())
```

```
   market  year  quantity  priceMin  priceMax  priceMod  state  city  \
2       0  2010     790.0    1283.0    1592.0    1460.0     16     0
3       0  2011     245.0    3067.0    3750.0    3433.0     16     0
4       0  2012    1035.0     523.0     686.0     605.0     16     0
```

Connected to Python 3 Google Compute Engine backend

- Build and Training Model

- we chose for the forecasts:

  Seasonal Autoregressive integrate Moving Average(SARIMA) and Long Short-Term Memory(LSTM)

• Model Evaluation

Evaluate

```python
# Evaluate SARIMA
sarima_pred = forecast.predicted_mean
mae_sarima = mean_absolute_error(test['quantity'], sarima_pred)
mse_sarima = mean_squared_error(test['quantity'], sarima_pred)
rmse_sarima = np.sqrt(mse_sarima)
```

MinMaxScaler: scaler

sklearn.preprocessing._data.MinMaxScaler instance

```python
# Evaluate LSTM
mae_lstm = mean_absolute_error(scaler.inverse_transform(y_test.reshape(-1, 1)), predictions)
mse_lstm = mean_squared_error(scaler.inverse_transform(y_test.reshape(-1, 1)), predictions)
rmse_lstm = np.sqrt(mse_lstm)
```

```python
[40] print(f'SARIMA - MAE: {mae_sarima}, MSE: {mse_sarima}, RMSE: {rmse_sarima}')
     print(f'LSTM - MAE: {mae_lstm}, MSE: {mse_lstm}, RMSE: {rmse_lstm}')
```

```
SARIMA - MAE: 55908.64657722915, MSE: 10762350304.686823, RMSE: 103741.74812816113
LSTM - MAE: 26842.70438071693, MSE: 2840505904.4968004, RMSE: 53296.396730893546
```

The best Model

For SARIMA Performance

- MAE: 55908.64657722915
- MSE: 10762350304.686823
- RMSE: 103741.74812816113

For LSTM Performance

- MAE: 26842.70438071693
- MSE: 2840505904.4968004
- RMSE: 53296.396730893546

The best Model

MAE in LSM is smaller than MAE in SARIMA

- 26842.70438071693 < 55908.64657722915

MSE in LSM is smaller than MSE in SARIMA

53296.396730893546

- 2840505904.4968004 < 10762350304.686823

RMSE in LSM is smaller than RMSE inSARIMA

- 53296.396730893546 < 103741.74812816113

Conclusion

After Comparing performances

- The best performing model is LSTM based on all three metrics.

THANK YOU FOR YOUR ATTENTION!!!!!!!!!