وزارة الاتصالات
وتكنولوجيا المعلومات

Digital Egypt Pioneers

Sales Forecasting and Demand Prediction

وزارة الاتصالات
وتكنولوجيا المعلومات

Digital Egypt Pioneers

track : Data science

Group Code :

CAI2_AIS4_S15

Team members :

yousef awad

Losia Aweny

Ali Sharaf

Mai Mamoon

Why Forecasting Matters ?

Problem: Inaccurate sales predictions lead to overstocking, understocking, and lost revenue.

Goal: Help the business make informed inventory and planning decisions.

Project Goal

01. Build a forecasting model for sales and demand

02. Support decision-making with accurate predictions

Dataset Overview

datasets used:
train.csv:sales data
stoers.csv,holidays.csv,oil.csv,transactions.csv
time span: 2013-2017
54 stores, various product families

Data preprocessing

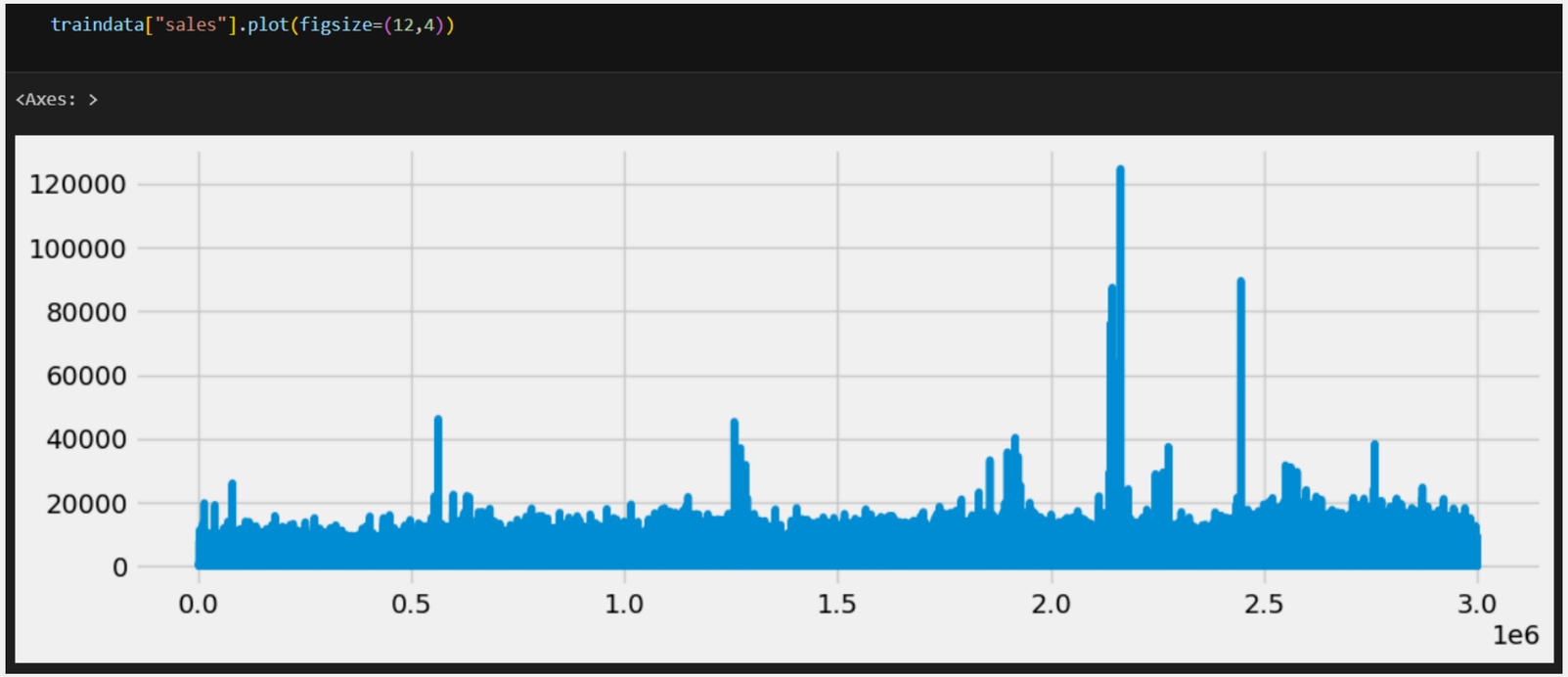
This plot shows us how total daily sales changed over time in the dataset.

x-axis: time

y-axis: total sales for that day

```
traindata["sales"].plot(figsize=(12,4))
```

```
<Axes: >
```



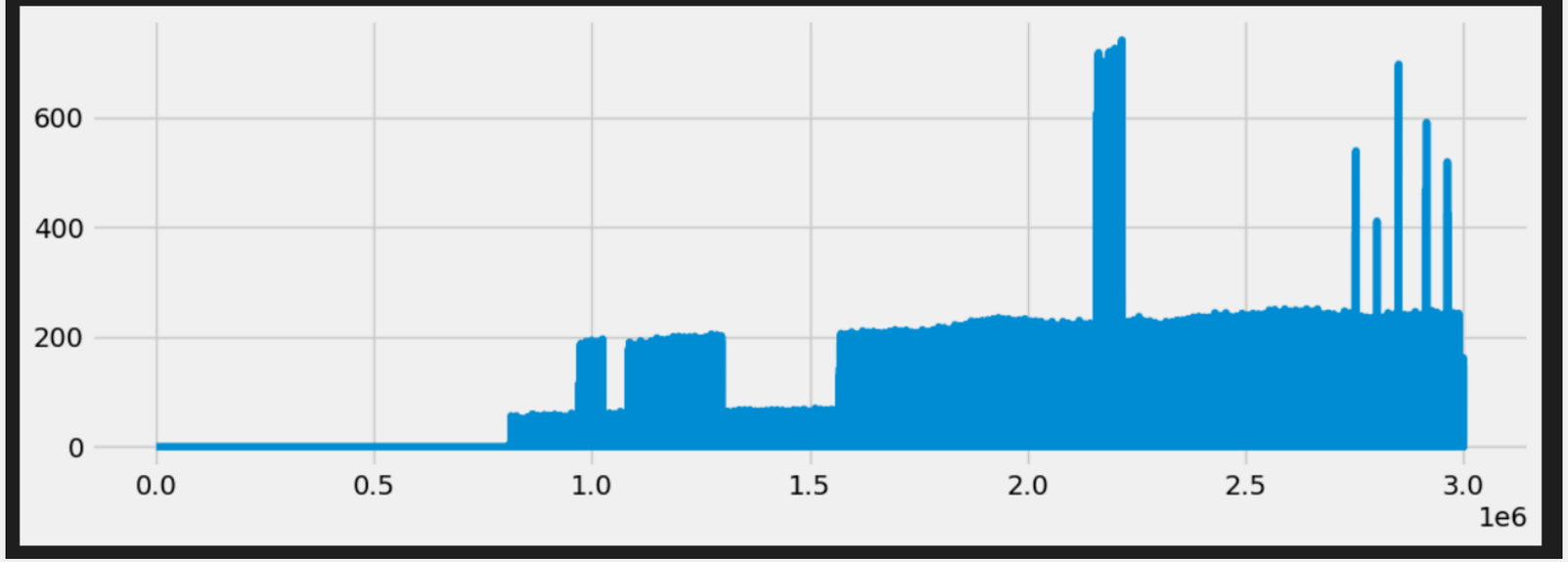
This plot shows us the total sales onpromotion at a given date

x-axis: time

y-axis: total sales for that day

```
# onpromotion gives the total number of items in a product family that were being promoted at a store at a given date.
traindata["onpromotion"].plot(figsize=(12,4))
```

```
<Axes: >
```



Merging Data

```
merged = pd.merge(traindata, storesdata, how='left', on='store_nbr')
```

Merging Data

Adding is_holiday helps your model capture dips/spikes in sales due to holidays, especially for products that are sensitive to seasonality or special events

```
national_holidays = holidaysdata[holidaysdata['locale'] == 'National'][['date']]
```

```
national_holidays['is_holiday'] = 1
merged = pd.merge(merged, national_holidays, how='left', on='date')
```

Merging Data

there is no duplicated data in merged data or any data

```
print(traindata.duplicated().sum())
print(holidaysdata.duplicated().sum())
print(storesdata.duplicated().sum())
print(merged.duplicated().sum())
```

```
duplicates = merged[merged.duplicated()]
duplicates
```

converting date to datetime

```
merged['date'] = pd.to_datetime(merged['date'])
```

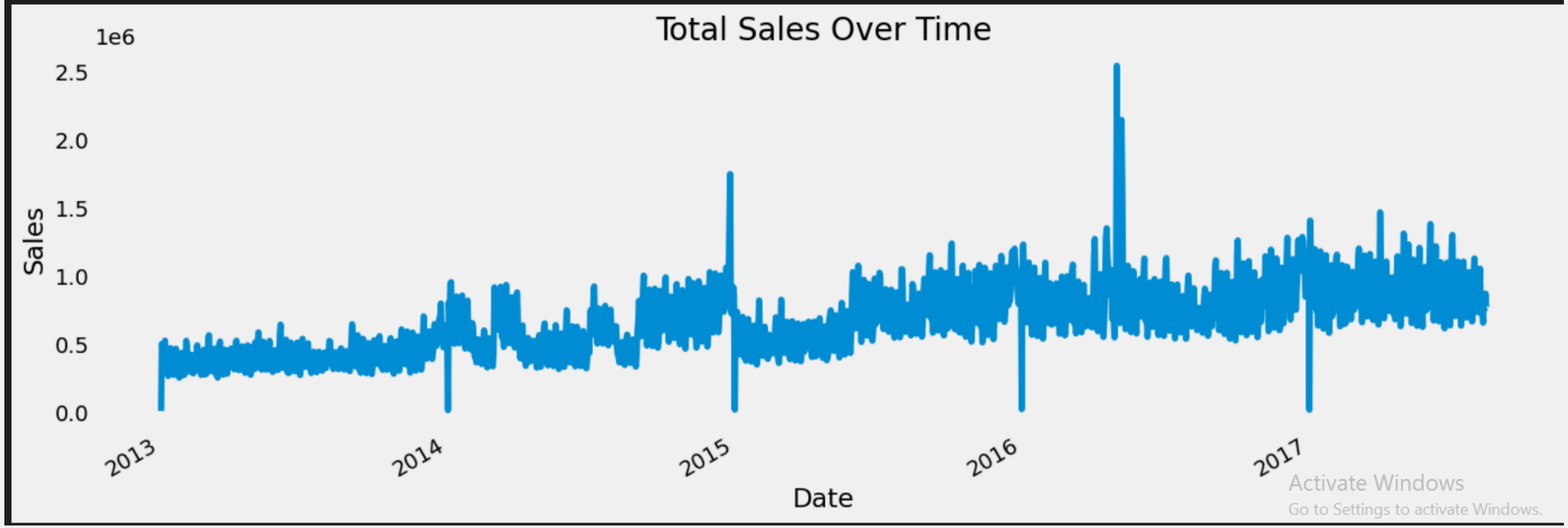
Summary stats for numerical columns

```
print(merged.describe())  # Summary stats for numerical columns
```

```
                 id                           date     store_nbr  \
count  3.008016e+06                        3008016  3.008016e+06
mean   1.501508e+06  2015-04-24 22:50:02.843602688  2.750000e+01
min    0.000000e+00            2013-01-01 00:00:00  1.000000e+00
25%    7.520038e+05            2014-02-27 18:00:00  1.400000e+01
50%    1.502226e+06            2015-04-25 12:00:00  2.750000e+01
75%    2.248883e+06            2016-06-18 06:00:00  4.100000e+01
max    3.000887e+06            2017-08-15 00:00:00  5.400000e+01
std    8.657303e+05                            NaN  1.558579e+01

                sales    onpromotion        cluster     is_holiday
count  3.008016e+06  3.008016e+06  3.008016e+06  3.008016e+06
mean   3.582691e+02  2.609620e+00  8.481481e+00  8.708531e-02
min    0.000000e+00  0.000000e+00  1.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  4.000000e+00  0.000000e+00
50%    1.100000e+01  0.000000e+00  8.500000e+00  0.000000e+00
75%    1.960000e+02  0.000000e+00  1.300000e+01  0.000000e+00
max    1.247170e+05  7.410000e+02  1.700000e+01  1.000000e+00
std    1.103511e+03  1.226308e+01  4.649735e+00  2.819601e-01
```

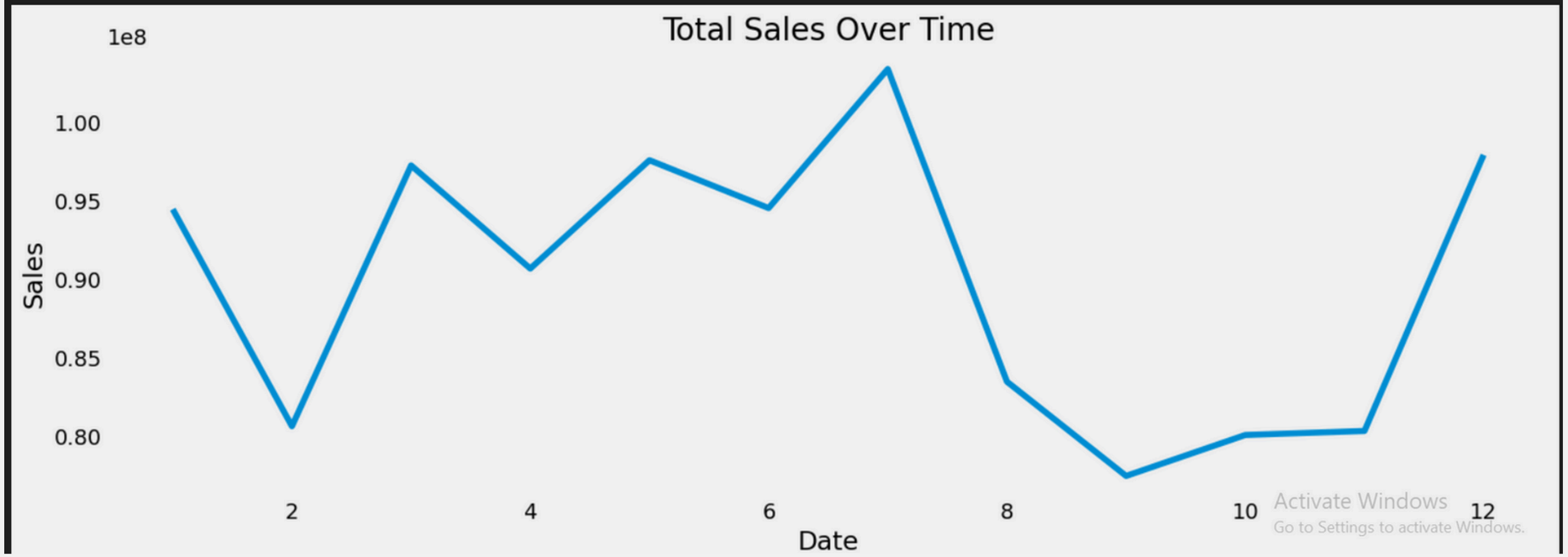
Total sales over time by date

```
daily_sales = merged.groupby('date')['sales'].sum()
daily_sales.plot(figsize=(15,5), title='Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid()
plt.show()
```



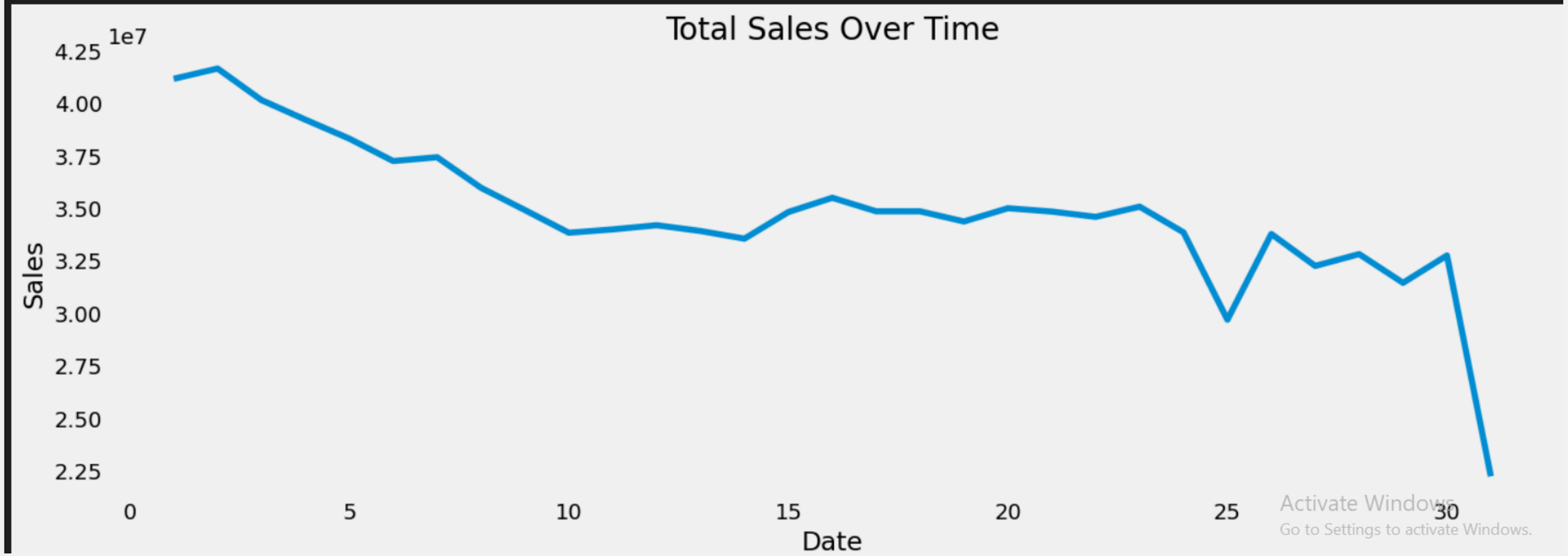
Total sales over time by month

```
daily_sales = merged.groupby('month')['sales'].sum()
daily_sales.plot(figsize=(15,5), title='Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid()
plt.show()
```



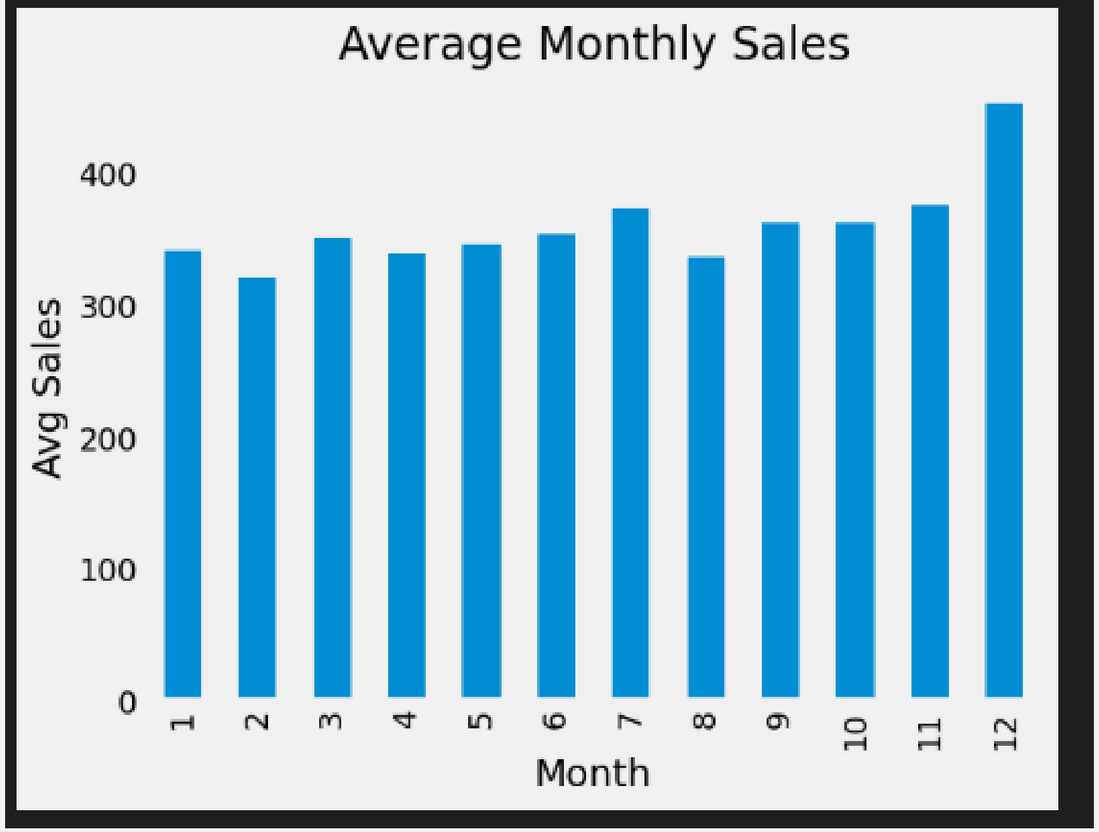
Total sales over time by day

```python
daily_sales = merged.groupby('day')['sales'].sum()
daily_sales.plot(figsize=(15,5), title='Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid()
plt.show()
```



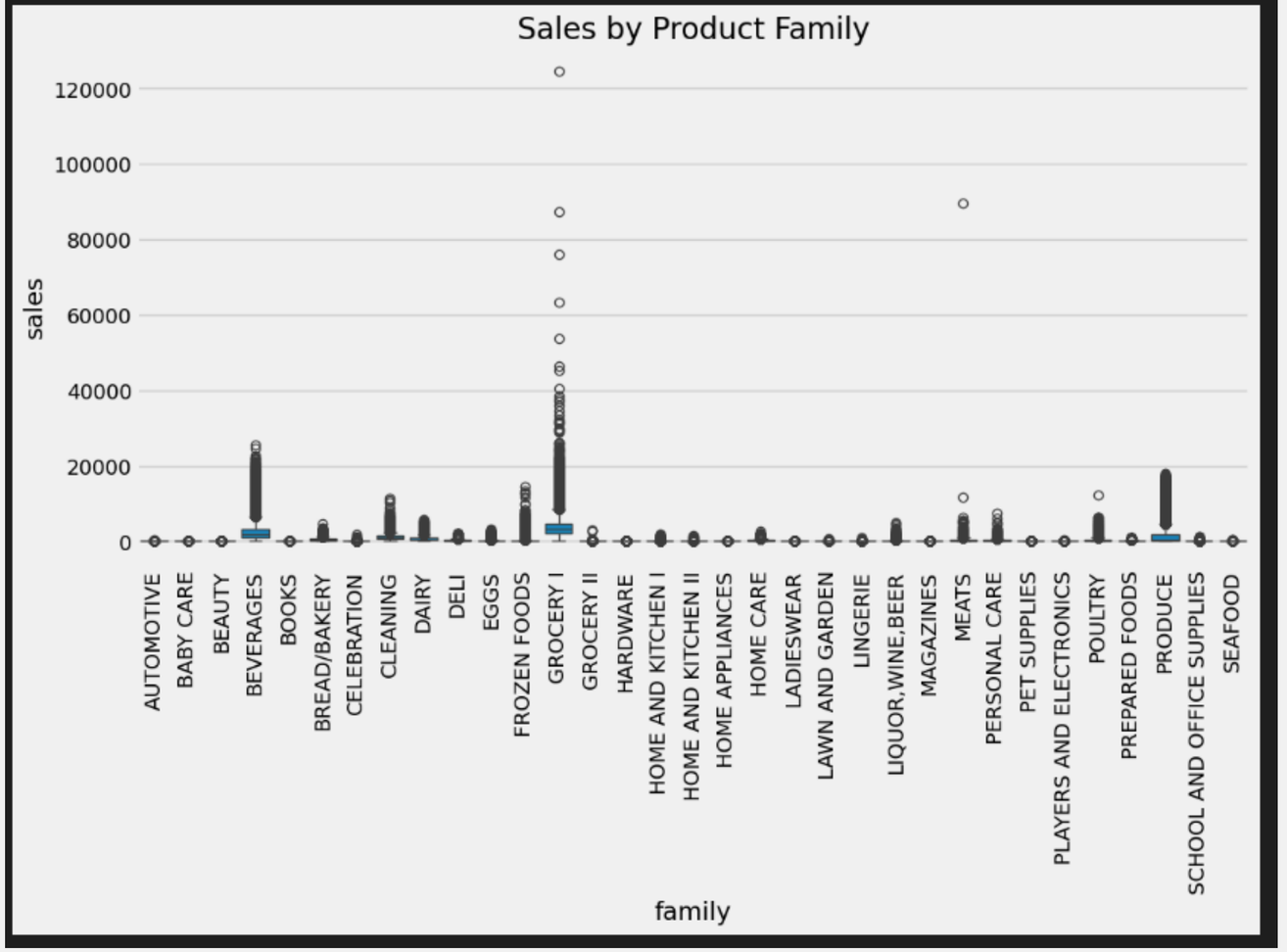
Average monthly sales

```
merged['month'] = merged['date'].dt.month
monthly_avg = merged.groupby('month')['sales'].mean()
monthly_avg.plot(kind='bar', title='Average Monthly Sales')
plt.xlabel('Month')
plt.ylabel('Avg Sales')
plt.grid()
plt.show()
```



This is sales by product family to identitfy which product categories sell more

```
plt.figure(figsize=(12,6))
sns.boxplot(x='family', y='sales', data=merged)
plt.xticks(rotation=90)
plt.title('Sales by Product Family')
plt.show()
```



EDA

Remove unneeded columns&
Add columns we need

```
merged=merged.drop(columns=['cluster'])
```

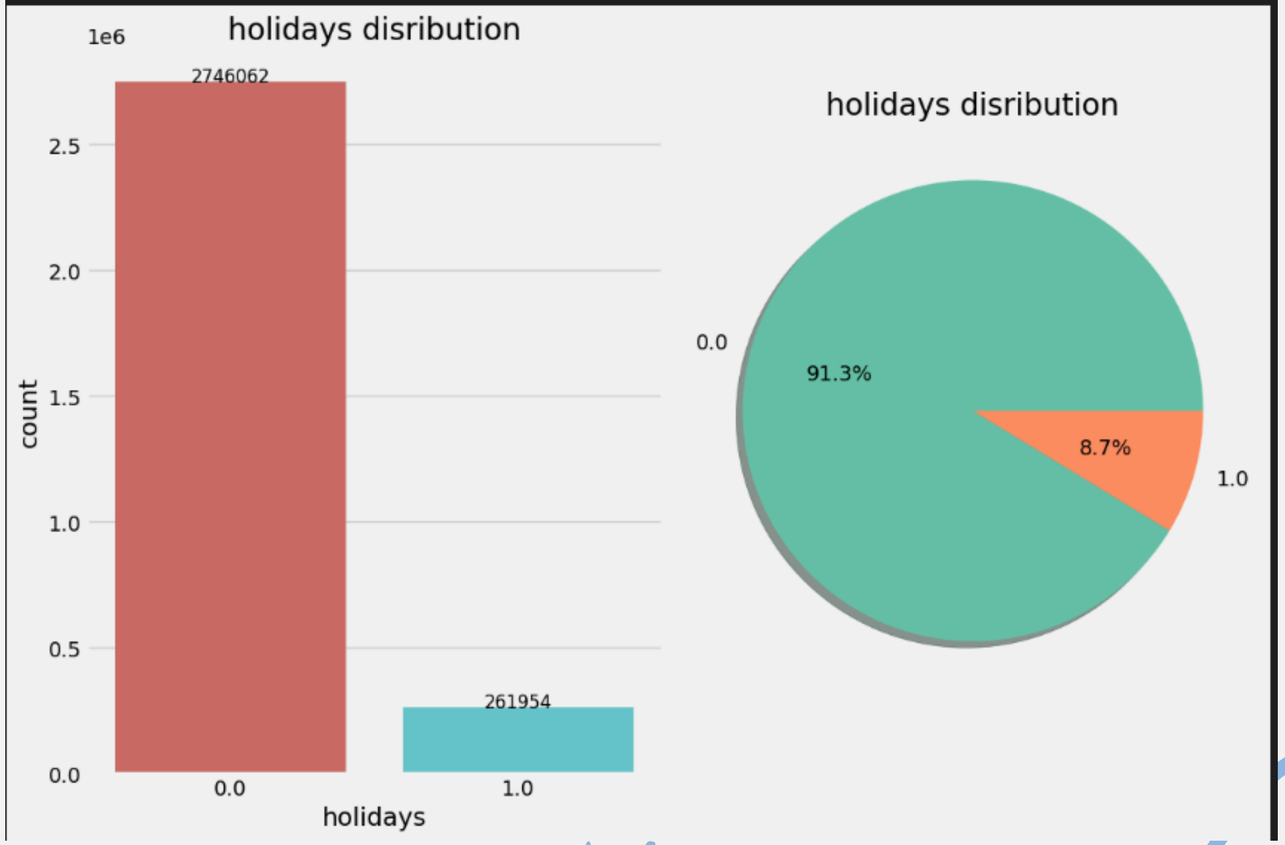
```
merged['day'] = merged['date'].dt.day
merged['week'] = merged['date'].dt.isocalendar().week
merged['weekday'] = merged['date'].dt.weekday
merged['year'] = merged['date'].dt.year
```

```
merged.head()
```

```
merged.info()
```

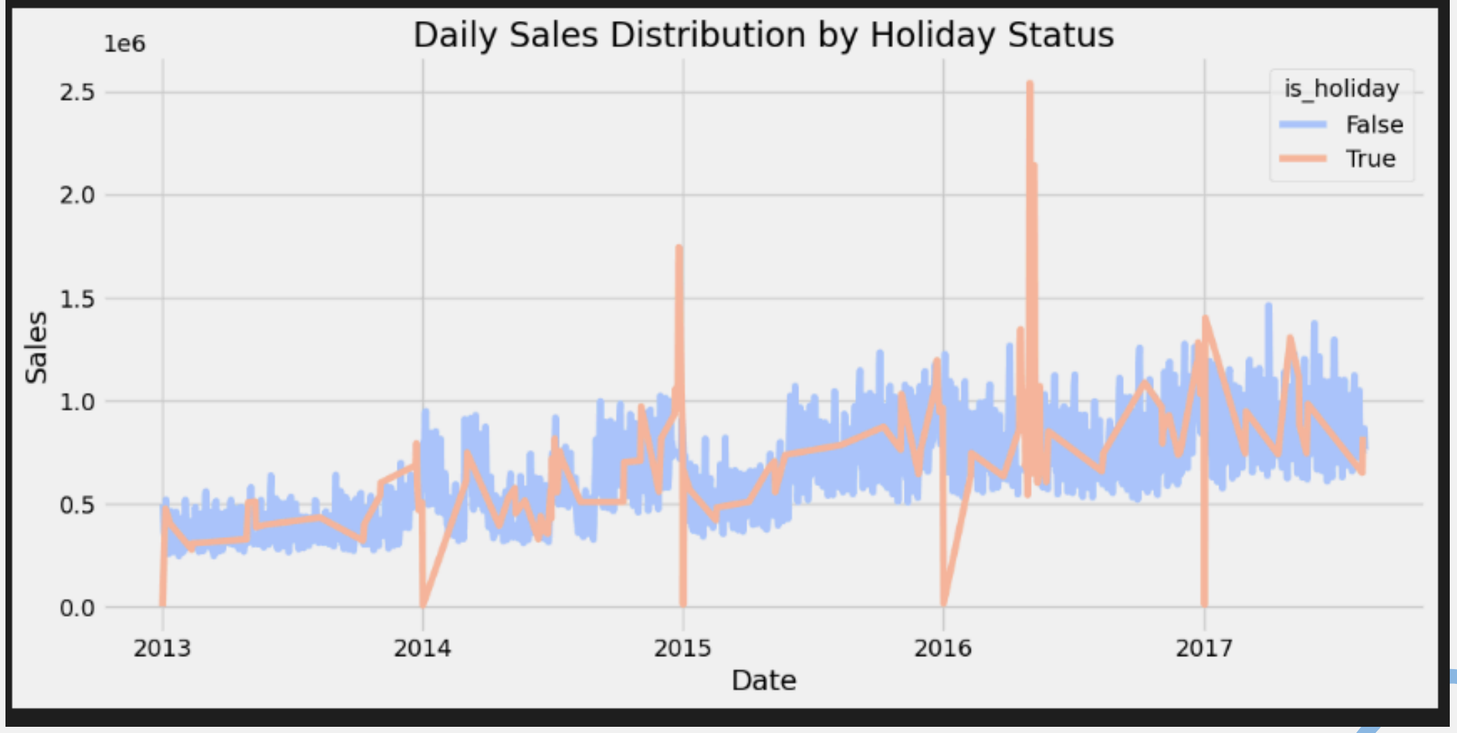
bar pie

```python
fig,axes=plt.subplots(1,2,figsize=(12,8))

geo_count=merged['is_holiday'].value_counts()

sns.barplot(x=geo_count.index,y=geo_count.values,palette='hls',ax=axes[0])
axes[0].set_title("holidays disribution")
axes[0].set_xlabel('holidays')
axes[0].set_ylabel('count')
# Correctly adding values on top of the bars
for i, value in enumerate(geo_count.values):
    axes[0].text(i, value + 5, str(value), ha='center', fontsize=12, color='black')


axes[1].pie(geo_count,labels=geo_count.index,autopct='%1.1f%%',colors=sns.color_palette("Set2"),shadow=True)
axes[1].set_title("holidays disribution")
plt.tight_layout()
plt.show()
```



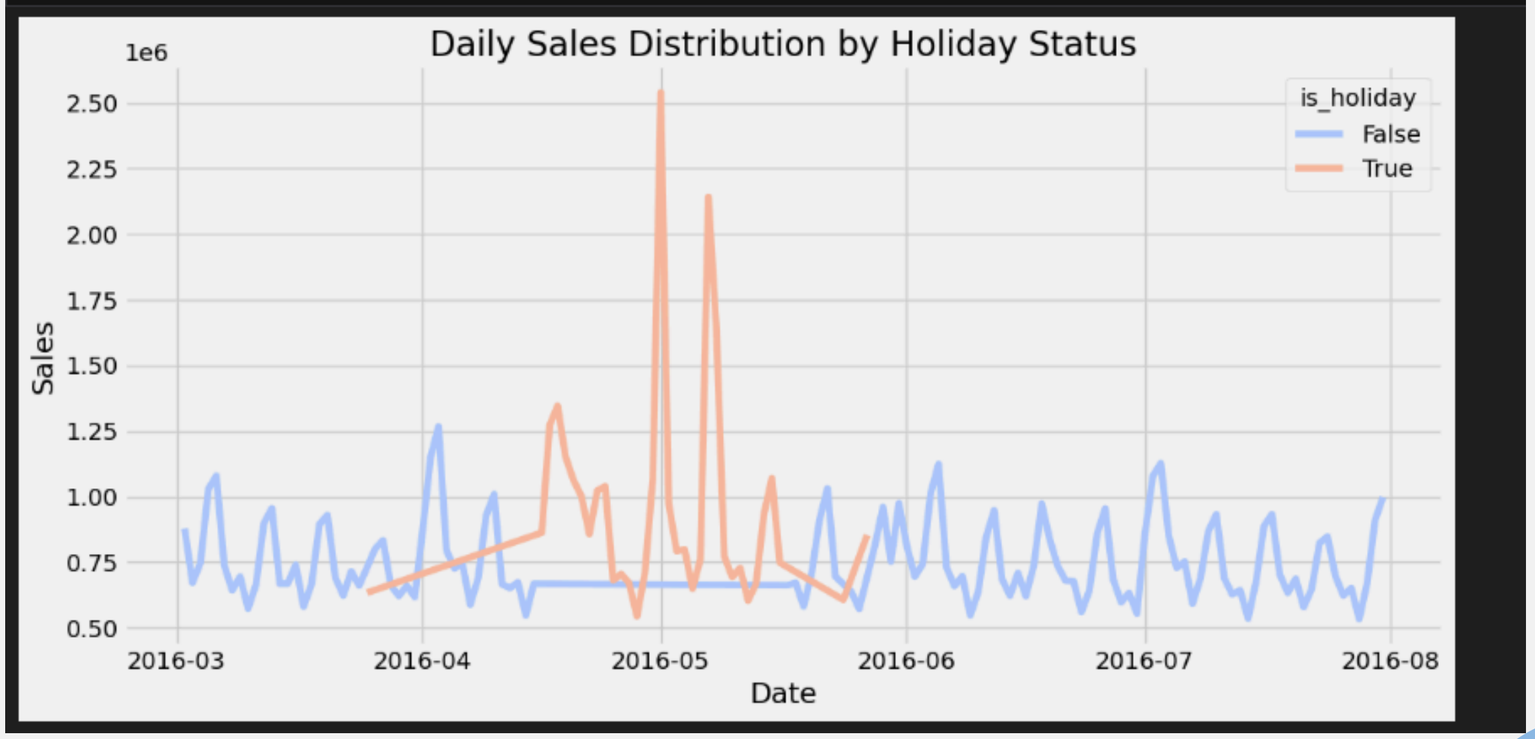
histogram

```python
# Ensure 'is_holiday' is a boolean
merged['is_holiday'] = merged['is_holiday'].astype(bool)

# Aggregating sales data by time and holiday status (daily)
daily_sales = merged.groupby(['date', 'is_holiday']).agg({'sales': 'sum'}).reset_index()

# Plotting a single line plot for holiday vs non-holiday sales
plt.figure(figsize=(12, 6))
sns.lineplot(data=daily_sales, x='date', y='sales', hue='is_holiday', palette='coolwarm')

# Customizing the plot
plt.title('Daily Sales Distribution by Holiday Status')
plt.xlabel('Date')
plt.ylabel('Sales')

# Display the plot
plt.tight_layout()
plt.show()
```



Daily sales Distribution by holiday status

```python
# Ensure 'is_holiday' is a boolean
merged['is_holiday'] = merged['is_holiday'].astype(bool)

# Aggregating sales data by time and holiday status (daily)
daily_sales = merged[(merged['date'] > "2016-03-01") & (merged['date'] < "2016-08-01")].groupby(['date', 'is_holiday']).agg({'sales': 'sum'}).reset_index()

# Plotting a single line plot for holiday vs non-holiday sales
plt.figure(figsize=(12, 6))
sns.lineplot(data=daily_sales, x='date', y='sales', hue='is_holiday', palette='coolwarm')

# Customizing the plot
plt.title('Daily Sales Distribution by Holiday Status')
plt.xlabel('Date')
plt.ylabel('Sales')

# Display the plot
plt.tight_layout()
plt.show()
```



SO WE CAN SUMARIZE THAT CITY WHO CAUSE THE MOST GAIN FROM SALES IS IN QUITO IN STATE Pichincha

```
Click to add a breakpoint d[(merged['date'] > "2016-03-01") & (merged['date'] < "2016-08-01") & (merged['is_holiday'] == True)].groupby(['date', 'city', 'state
daily_sales
```

37]

Python

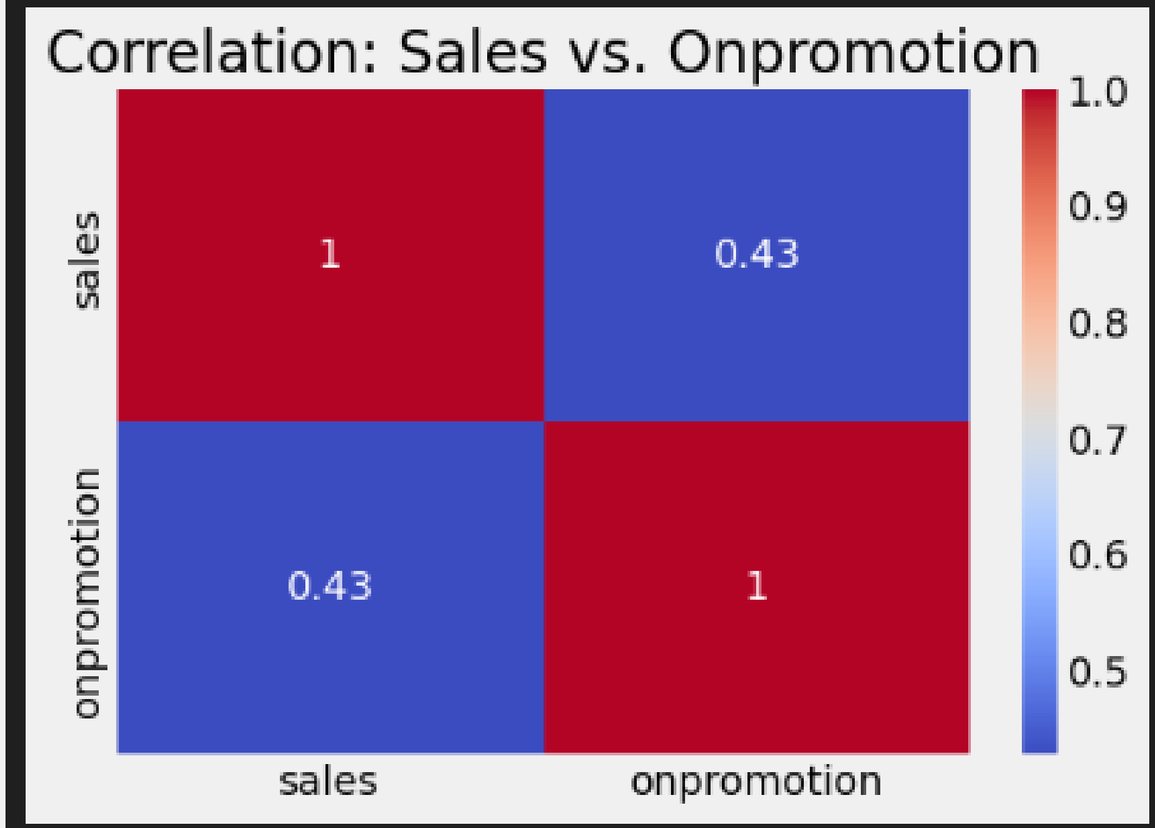
	date	city	state	sales
370	2016-05-01	Quito	Pichincha	1.383253e+06
502	2016-05-07	Quito	Pichincha	1.124058e+06
524	2016-05-08	Quito	Pichincha	8.124316e+05
62	2016-04-17	Quito	Pichincha	7.325722e+05
84	2016-04-18	Quito	Pichincha	7.049882e+05
...	...	...	...	...
724	2016-05-24	Salinas	Santa Elena	3.801368e+03
631	2016-05-13	Playas	Guayas	3.652568e+03
302	2016-04-28	Puyo	Pastaza	3.647044e+03
80	2016-04-18	Manta	Manabi	2.797280e+02
58	2016-04-17	Manta	Manabi	2.571900e+02

748 rows × 4 columns

SO WE CAN SUMARIZE THAT CITY WHO CAUSE THE MOST GAIN FROM SALES IS IN QUITO IN STATE Pichincha

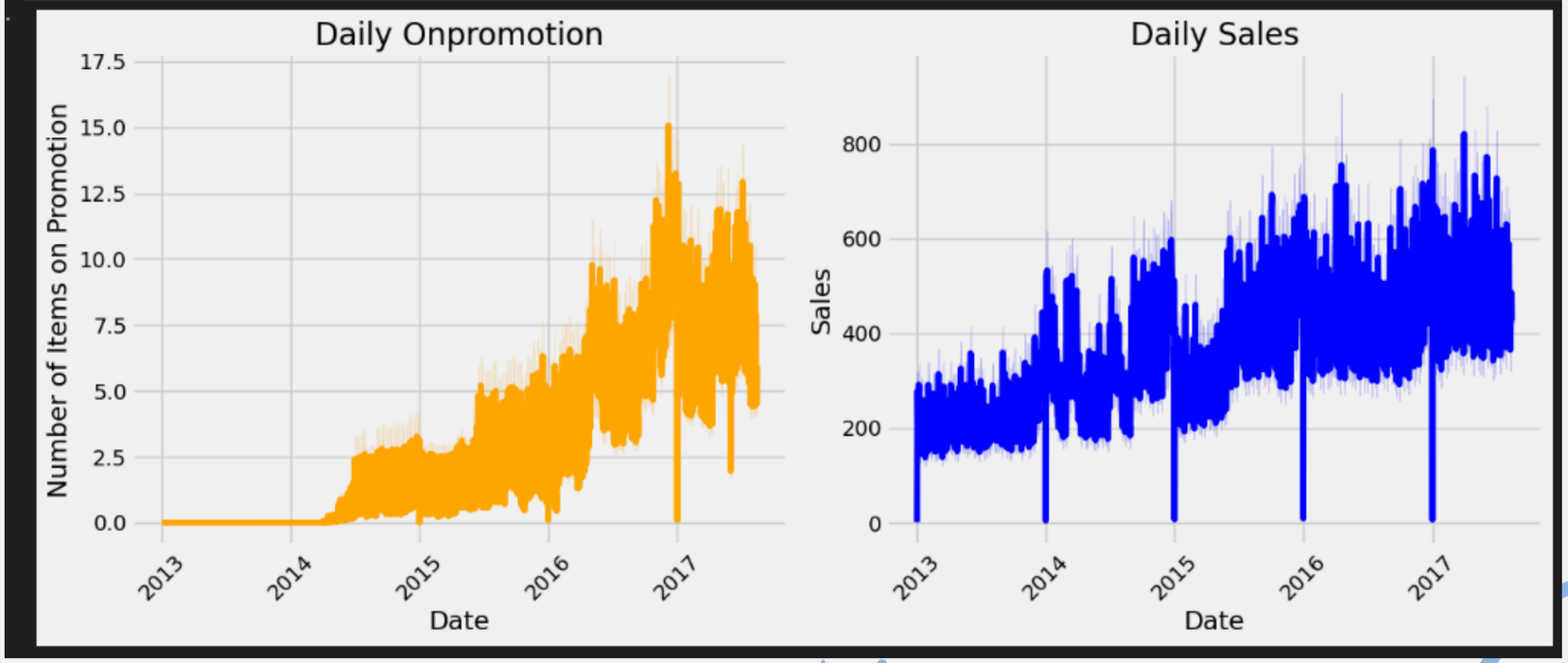
Heat map

```
plt.figure(figsize=(6, 4))
sns.heatmap(merged[['sales', 'onpromotion']].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation: Sales vs. Onpromotion")
plt.show()
```



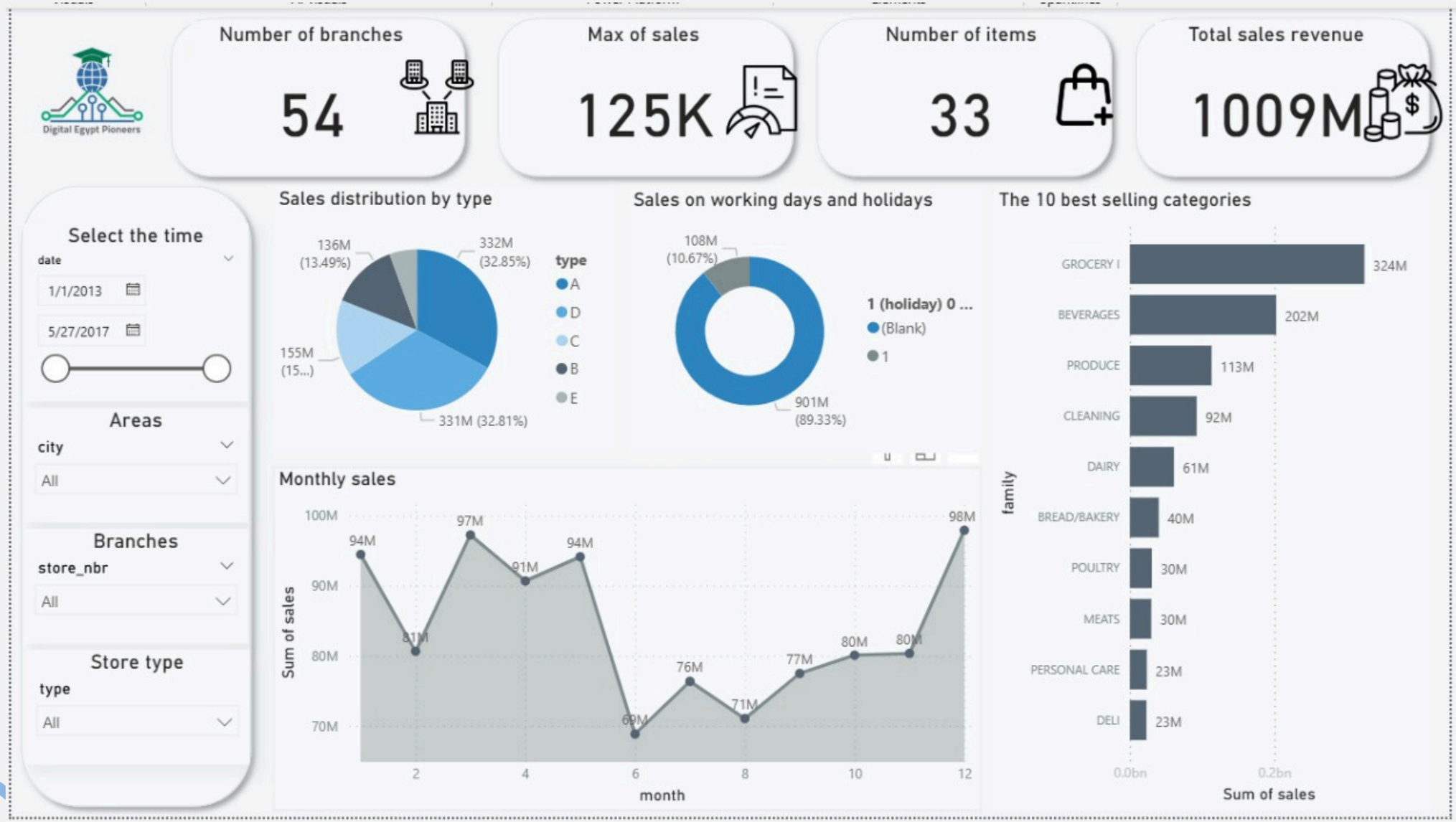
This is comparison betwen daily sales on promotion over the time and daily sales over the time

```
# Set figure size and style
plt.figure(figsize=(14, 6))

# Subplot 1: Onpromotion over time
plt.subplot(1, 2, 1)
sns.lineplot(data=merged, x='date', y='onpromotion', color='orange')
plt.title('Daily Onpromotion')
plt.xlabel('Date')
plt.ylabel('Number of Items on Promotion')
plt.xticks(rotation=45)

# Subplot 2: Sales over time
plt.subplot(1, 2, 2)
sns.lineplot(data=merged, x='date', y='sales', color='blue')
plt.title('Daily Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)

# Layout adjustment
plt.tight_layout()
plt.show()
```



Power Bi charts

Power Bi charts



Power Bi charts

Digital Egypt Pioneers

Select the time

date

1/1/2013

5/27/2017

Branches

store_nbr

All

Store type

type

All

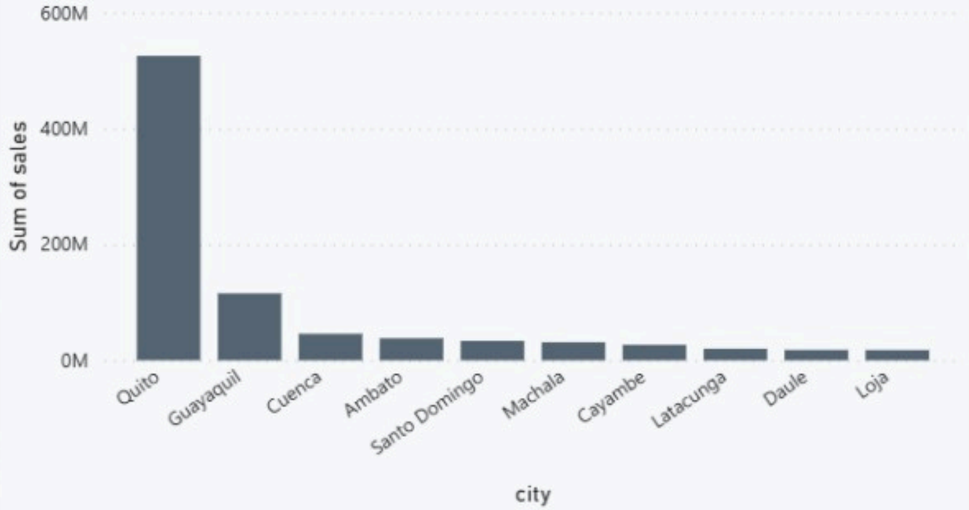
Areas

city

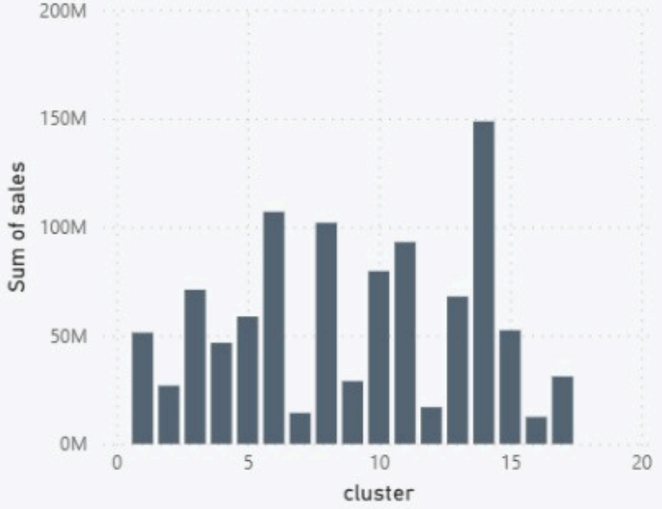
All

store_nbr	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DAIRY	DELI	EGGS	FROZEN FOOD
1	5K	0K	4K	2507K	0K	544K	16K	1029K	1002K	197K	226K	208K
2	9K	0K	7K	3912K	0K	874K	12K	1553K	1370K	394K	301K	262K
3	15K	1K	15K	10666K	1K	1842K	45K	3316K	3268K	635K	840K	729K
4	6K	0K	7K	3459K	0K	653K	10K	1425K	1268K	348K	278K	246K
5	9K	0K	8K	2388K	0K	616K	24K	1596K	829K	398K	184K	263K
6	9K	0K	9K	4774K	0K	875K	15K	1980K	1531K	548K	331K	367K
7	7K	0K	6K	5268K	0K	1027K	18K	1685K	1960K	270K	457K	415K
8	9K	0K	11K	5978K	0K	1203K	23K	2025K	2258K	446K	524K	387K
9	22K	0K	9K	4346K	0K	914K	15K	2772K	1190K	940K	309K	202K
10	5K	0K	2K	1410K	0K	184K	4K	1252K	399K	400K	79K	71K
11	11K	0K	8K	5787K	0K	969K	16K	3126K	812K	685K	248K	165K
12	8K	0K	2K	2055K	0K	300K	9K	1231K	531K	313K	121K	77K
13	9K	0K	2K	2053K	0K	267K	6K	1251K	432K	254K	84K	53K
Total	525K	9K	314K	202282K	6K	39881K	707K	92310K	60833K	22820K	14765K	13554

Top 10 cities in terms of sales



Sum of sales by cluster



modelling

LSTM Model

```
[ ]  # Normalize sales
     scaler = MinMaxScaler()
     scaled_sales = scaler.fit_transform(daily_sales[['sales']])

     # Create sequences
     def create_sequences(data, window_size=30):
         X, y = [], []
         for i in range(len(data) - window_size):
             X.append(data[i:i+window_size])
             y.append(data[i+window_size])
         return np.array(X), np.array(y)

     window_size = 30
     X, y = create_sequences(scaled_sales, window_size)

     # Train/validation split
     split = int(len(X) * 0.8)
     X_train, X_val = X[:split], X[split:]
     y_train, y_val = y[:split], y[split:]

     # Reshape for LSTM
     X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
     X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))
```

LSTM results

```
[ ]  from tensorflow.keras.optimizers import Adam
```

```
[ ]  model = Sequential()
     model.add(LSTM(16, activation='relu', input_shape=(window_size, 1)))
     # model.add(LSTM(16, activation='relu', ))
     #
     model.add(Dense(1))
     model.compile(optimizer=Adam(learning_rate=0.0001), loss='mse')

     history = model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_val, y_val))
```

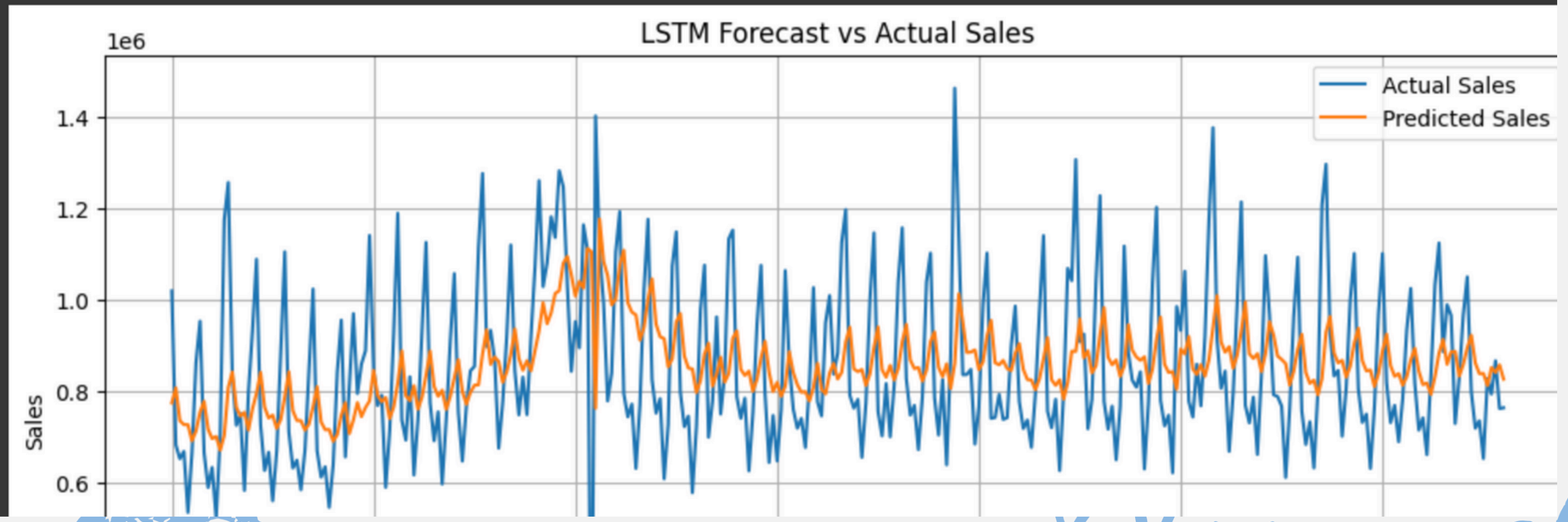
```
Epoch 1/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.1317 - val_loss: 0.1063
Epoch 2/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0309 - val_loss: 0.0172
Epoch 3/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0104 - val_loss: 0.0187
Epoch 4/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0102 - val_loss: 0.0169
Epoch 5/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0097 - val_loss: 0.0169
Epoch 6/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0103 - val_loss: 0.0168
Epoch 7/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0096 - val_loss: 0.0167
Epoch 8/50
166/166 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0104 - val_loss: 0.0167
Epoch 9/50
```

```
[ ]  y_pred = model.predict(X_val)
     y_pred_inv = scaler.inverse_transform(y_pred)
     y_val_inv = scaler.inverse_transform(y_val)

     mse = mean_squared_error(y_val_inv, y_pred_inv)
     mae = mean_absolute_error(y_val_inv, y_pred_inv)

     print(f"Mean Squared Error: {mse:.2f}")
     print(f"Mean Absolute Error: {mae:.2f}")
```

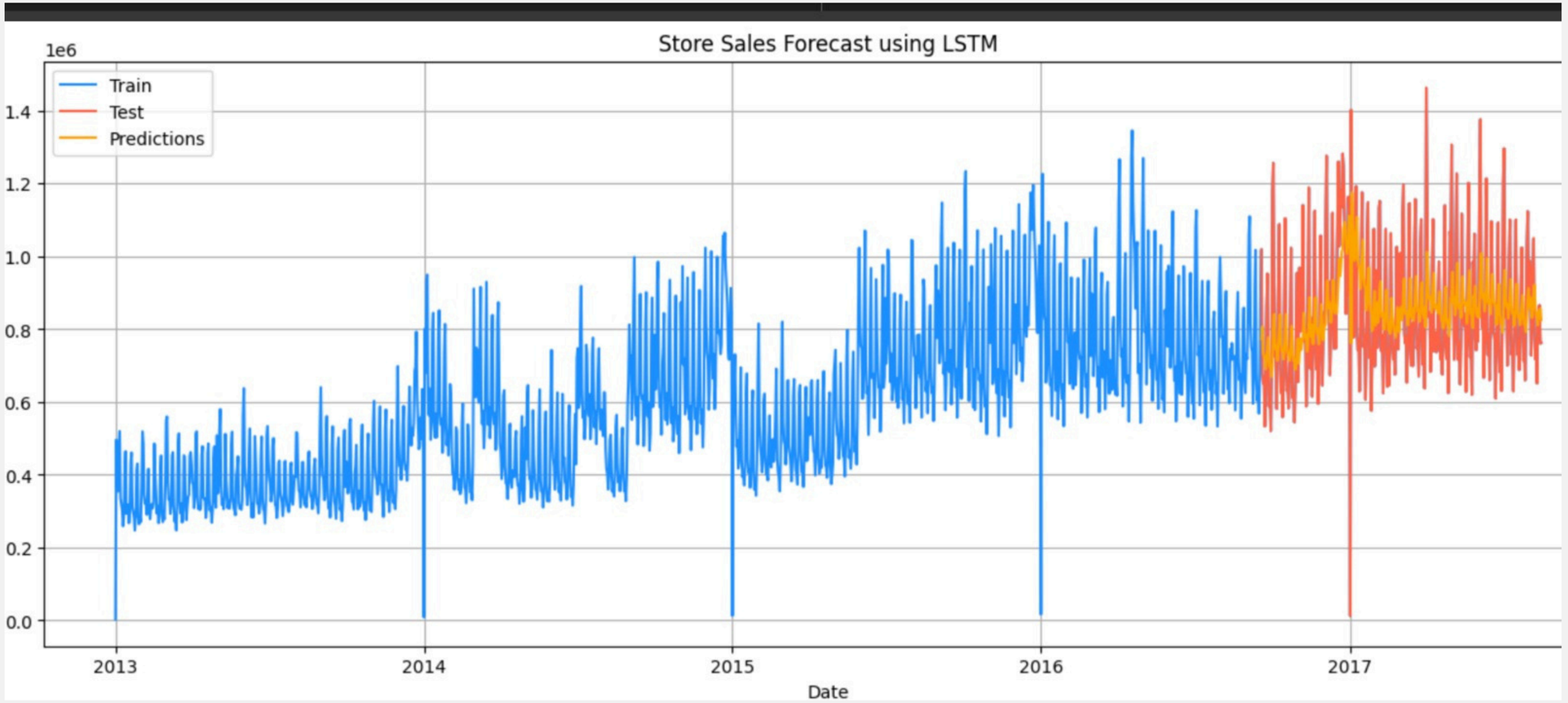
```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Mean Squared Error: 32235446686.02
Mean Absolute Error: 142038.31
```

Store sales prediction using LSTM

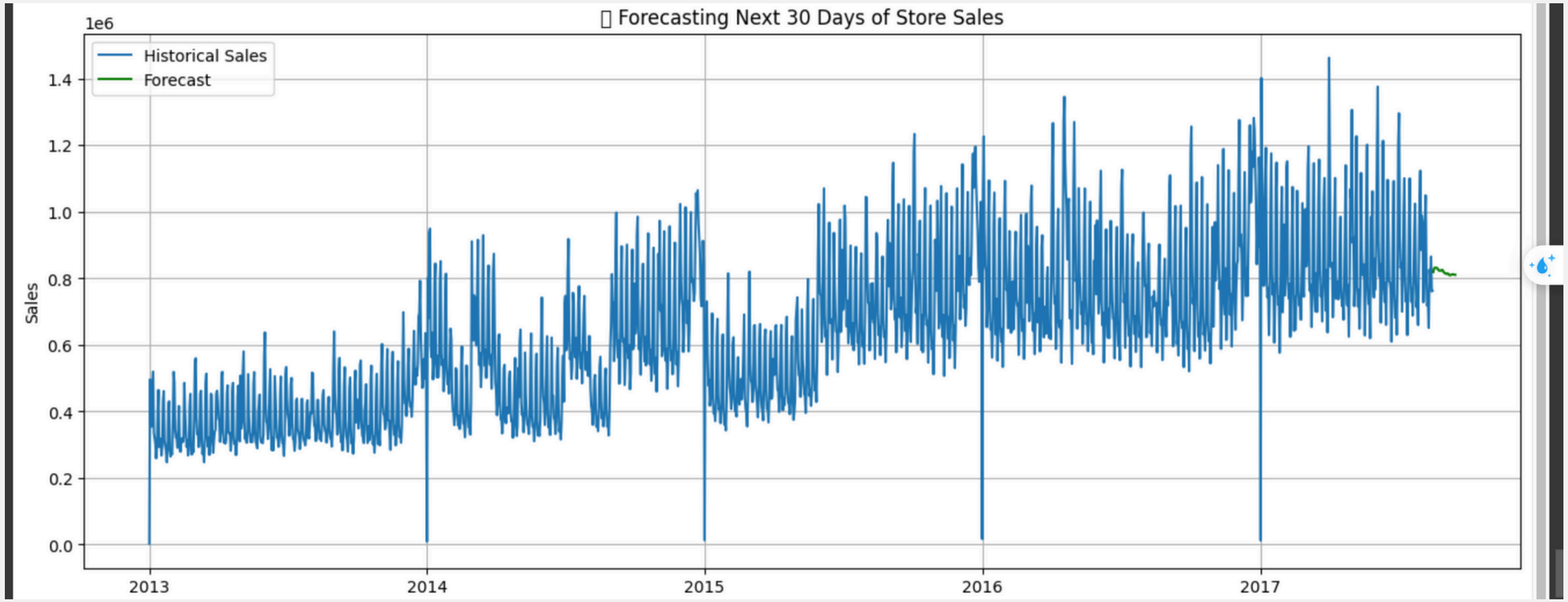
```
[ ]  plt.figure(figsize=(12,6))
     plt.plot(y_val_inv, label='Actual Sales')
     plt.plot(y_pred_inv, label='Predicted Sales')
     plt.title("LSTM Forecast vs Actual Sales")
     plt.xlabel("Days")
     plt.ylabel("Sales")
     plt.legend()
     plt.grid()
     plt.show()
```



Store sales forcasting using LSTM



Store sales forcasting next 30 days using LSTM



ARIMA MODEL

```
[ ]  from statsmodels.tsa.arima.model import ARIMA
     from sklearn.metrics import mean_squared_log_error
     import matplotlib.pyplot as plt

     # Assuming your data is already loaded as:
     # X (features), y (target), with datetime index

     # Split into train/test (last 30 days for validation)
     y_train, y_valid = y.iloc[:-30], y.iloc[-30:]

     # Fit ARIMA model (you may need to tune these parameters)
     order = (5,1,0)  # Example order (p,d,q)
     model = ARIMA(y_train, order=order)
     model_fit = model.fit()

     # Generate predictions
     y_fit = model_fit.predict(start=y_train.index[0], end=y_train.index[-1], typ='levels').clip(0.0)
     y_pred = model_fit.forecast(steps=len(y_valid)).clip(0.0)

     # Calculate RMSLE
     rmsle_train = mean_squared_log_error(y_train, y_fit) ** 0.5
     rmsle_valid = mean_squared_log_error(y_valid, y_pred) ** 0.5
     print(f'Training RMSLE: {rmsle_train:.5f}')
     print(f'Validation RMSLE: {rmsle_valid:.5f}')

     # Enhanced visualization
     plt.figure(figsize=(14, 7))
     plt.style.use('seaborn-v0_8-darkgrid')
```

RMSE result

```
[ ]  import numpy as np
     forecast = forecast.fillna(0)
     rmse = np.sqrt(mean_squared_error(test, forecast))
     print(f'RMSE: {rmse}')
```

```
RMSE: 867959.2676517024
```

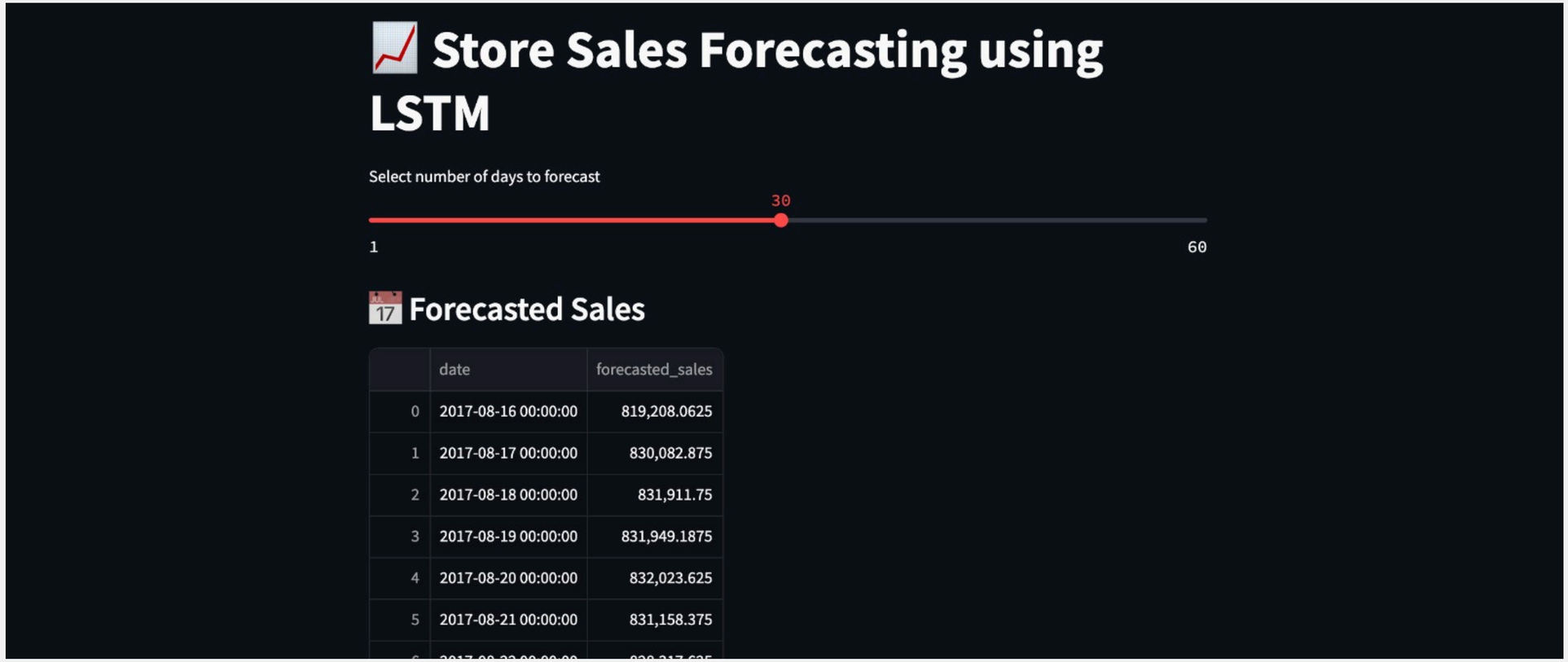
Random Forest Model

```python
y_pred_valid_log = model.predict(X_valid, num_iteration=model.best_iteration)
y_pred_valid = np.expm1(y_pred_valid_log)  # Inverse transform

def rmsle(y_true, y_pred):
    return np.sqrt(mean_squared_log_error(y_true, np.maximum(0, y_pred)))

print(f'Validation RMSLE: {rmsle(y_valid, y_pred_valid):.5f}')

# --- Predict on Test Set ---
X_test = test[features]
y_test_pred_log = model.predict(X_test, num_iteration=model.best_iteration)
y_test_pred = np.expm1(y_test_pred_log)  # Inverse transform
y_test_pred = np.maximum(0, y_test_pred)

# --- Submission ---
submission = pd.read_csv('/kaggle/input/store-sales-time-series-forecasting/sample_submission.csv')
submission['sales'] = y_test_pred
submission.to_csv('submission.csv', index=False)

print("✅ Submission file created successfully brother (Optimized)!")
```

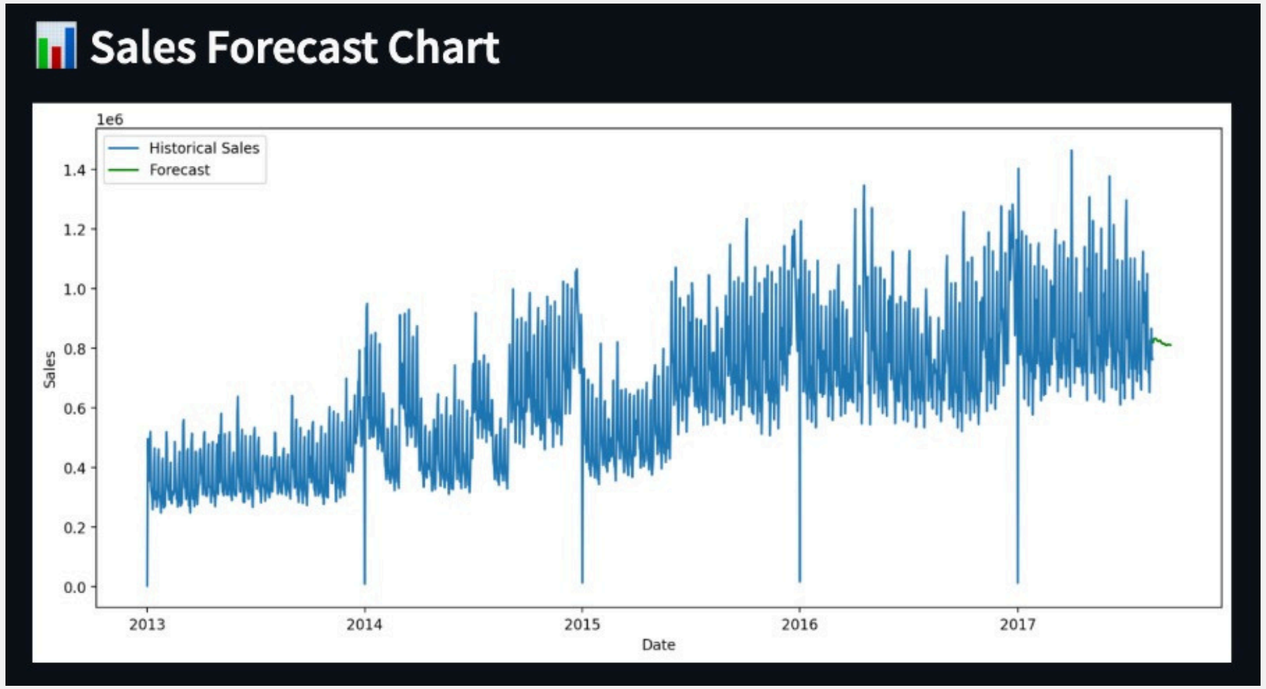
```
[2500]  training's rmse: 0.346521       valid_1's rmse: 0.369552
[3000]  training's rmse: 0.343396       valid_1's rmse: 0.369444
[3500]  training's rmse: 0.340539       valid_1's rmse: 0.369198
[4000]  training's rmse: 0.337966       valid_1's rmse: 0.369017
[4500]  training's rmse: 0.335471       valid_1's rmse: 0.36897
Early stopping, best iteration is:
[4295]  training's rmse: 0.336486       valid_1's rmse: 0.3689
Validation RMSLE: 0.36889
✅ Submission file created successfully brother (Optimized)!
```

Deployment

Store sales forcasting using LSTM



sales forecast chart



Any Questions?

Thank you very much!