Project Title	**Colorado Motor Vehicle Sales Data**
Tools	Visual Studio code / jupyter notebook
Domain	Finance Analyst
Project Difficulties level	Advance

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About Dataset

Colorado Motor Vehicle Sales Data

Description

This dataset contains information on motor vehicle sales in various counties across Colorado, segmented by year and quarter. The data is useful for analyzing trends in vehicle sales, understanding the economic impact of automotive transactions, and making informed decisions in related business or policy planning.

Columns

- **Year**: The calendar year in which the sales data was recorded.
- **Quarter**: The quarter of the year during which the sales were made. The quarters are divided as follows:
    - Q1: January to March
    - Q2: April to June
    - Q3: July to September
    - Q4: October to December
- **County**: The name of the county in Colorado where the sales were recorded.
- **Sales**: The total dollar amount of motor vehicle sales in the specified county and quarter.

Use Cases

- **Economic Analysis**: Track the economic health and trends in the automotive market within Colorado.
- **Market Research**: Identify sales patterns and market demands in different counties.
- **Policy Making**: Inform decisions on automotive industry regulations and infrastructure planning.

File Format

The dataset is available in CSV format, making it easy to import into various data analysis tools and software. The CSV file might be named `colorado_motor_vehicle_sales.csv`.

Colorado Motor Vehicle Sales Data Analysis Project

Project Overview

Objective: To analyze motor vehicle sales data in Colorado to identify trends, forecast future sales, and understand the factors influencing sales.

Steps to Follow:

1. **Define the Scope and Objective**:
    - Identify the key metrics and objectives for the analysis (e.g., monthly sales trends, sales by vehicle type, forecast future sales).
    - Define the time frame for the analysis.
2. **Data Collection**:
    - Gather motor vehicle sales data from reliable sources.
    - For this example, we'll assume a dataset named `colorado_motor_vehicle_sales.csv`.
3. **Data Preparation**:
    - Clean the data to remove any inconsistencies or errors.
    - Prepare the data for analysis using tools like Pandas.
4. **Exploratory Data Analysis (EDA)**:
    - Perform EDA to understand the data distribution and identify patterns.
    - Use visualization tools like Matplotlib and Seaborn to visualize the data.
5. **Statistical Analysis**:
    - Perform statistical analysis to identify correlations and trends.
    - Use tools like Python (Pandas, Statsmodels) for this purpose.
6. **Predictive Modeling**:
    - Build predictive models to forecast future motor vehicle sales.
    - Use machine learning algorithms like Linear Regression, ARIMA, or SARIMA.

7. **Reporting**:
    - Summarize the findings in a comprehensive report.
    - Use visualizations to support the analysis and make the report more engaging.

Example: You can get the basic idea how you can create a project from here

Step-by-Step Implementation

1. **Data Collection**:
    - Assume you have a dataset named `colorado_motor_vehicle_sales.csv` with columns like `Date`, `Vehicle_Type`, `Sales`.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# Load the dataset
data = pd.read_csv('colorado_motor_vehicle_sales.csv')

# Display the first few rows of the dataset
print(data.head())
```

2. **Data Preparation**:

```
# Convert date column to datetime format
```

```
data['Date'] = pd.to_datetime(data['Date'])

# Check for missing values
print(data.isnull().sum())

# Fill or drop missing values if necessary
data.dropna(inplace=True)

# Aggregate sales by month
data.set_index('Date', inplace=True)
monthly_sales = data.resample('M').sum()

# Display the first few rows of the aggregated data
print(monthly_sales.head())
```

3. **Exploratory Data Analysis (EDA)**:

```
# Plot total sales over time
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales.index, monthly_sales['Sales'], label='Total Sales')
plt.title('Total Motor Vehicle Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# Plot sales by vehicle type
```

```
vehicle_sales = data.groupby(['Date', 'Vehicle_Type']).sum().unstack()
vehicle_sales.plot(kind='line', figsize=(12, 6))
plt.title('Motor Vehicle Sales by Type Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend(title='Vehicle Type')
plt.show()
```

4. **Statistical Analysis**:

```
# Compute correlations between sales of different vehicle types
correlation_matrix = vehicle_sales.corr()

# Plot the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

5. **Predictive Modeling**:

```
# Perform seasonal decomposition on total sales
decomposition = seasonal_decompose(monthly_sales['Sales'], model='multiplicative',
period=12)
```

```
decomposition.plot()
plt.show()

# Fit an ARIMA model to the total sales data
model = ARIMA(monthly_sales['Sales'], order=(5, 1, 0))
model_fit = model.fit()
print(model_fit.summary())

# Make predictions
forecast = model_fit.forecast(steps=12)

# Plot the predictions
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales['Sales'], label='Actual Sales')
plt.plot(pd.date_range(start=monthly_sales.index[-1], periods=12, freq='M'), forecast,
label='Forecasted Sales', color='red')
plt.title('Motor Vehicle Sales Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# Evaluate the model
mse = mean_squared_error(monthly_sales['Sales'][-12:], forecast)
print(f'Mean Squared Error: {mse}')
```

6. **Reporting:**

```
# Generate a summary report
report = f"""
Colorado Motor Vehicle Sales Data Analysis Report
================================================

1. Data Overview
----------------
- Time Frame: {data.index.min()} to {data.index.max()}
- Total Sales Data Points: {len(data)}

2. Exploratory Data Analysis
----------------------------
- Total motor vehicle sales were plotted over time, showing general trends and seasonality.
- Sales by vehicle type were plotted to compare different categories.

3. Statistical Analysis
-----------------------
- Seasonal decomposition of total sales showed clear seasonal patterns.
- Correlation analysis showed relationships between sales of different vehicle types.

4. Predictive Modeling
----------------------
- An ARIMA model was used to forecast motor vehicle sales for the next 12 months.
- The model's Mean Squared Error (MSE) was: {mse:.2f}

5. Conclusions
--------------
- The analysis provided insights into the trends and seasonality of motor vehicle sales
```

```
in Colorado.
- The predictive model can be used to forecast future sales, aiding in inventory management and sales strategies.

"""

print(report)
```

Conclusion

This project provides a comprehensive analysis of Colorado motor vehicle sales data, including data collection, preparation, exploratory analysis, statistical analysis, and predictive modeling. The resulting report summarizes key findings and insights, which can be useful for decision-making and strategic planning.

Example: You can get the basic idea how you can create a project from here

Sample code with output

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
```

```
# You can also write temporary files to /kaggle/temp/, but they
won't be saved outside of the current session
```

```
/kaggle/input/colorado-motor-vehicle-sales-data/colorado_motor_
vehicle_sales.csv
```

In [2]:

```
df =
pd.read_csv('/kaggle/input/colorado-motor-vehicle-sales-data/co
lorado_motor_vehicle_sales.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	year	quarter	county	sales
0	20	1	Adams	23160

	08			9000
1	2008	1	Arapahoe	550378000
2	2008	1	Boulder/Bromfield	176771000
3	2008	1	Denver	200103000
4	2008	1	Douglas	93259000

EDA

In [4]:

```
import matplotlib.pyplot as plt
import seaborn as sns


def perform_eda(df):
    # Print the shape of the DataFrame
    print(f"Shape of the DataFrame: {df.shape}\n")
```

```python
# Print the data types of the DataFrame
print(f"Data types:\n{df.dtypes}\n")

# Check for missing values
print(f"Missing values:\n{df.isnull().sum()}\n")

# Summary statistics
print(f"Summary statistics:\n{df.describe()}\n")

# For each column
for column in df.columns:
    # Check if the column is numeric
    if pd.api.types.is_numeric_dtype(df[column]):
        # Plot a histogram
        plt.figure(figsize=(6, 4))
        sns.histplot(data=df, x=column, kde=True)
        plt.title(f"Histogram of {column}")
        plt.show()

    # Check if the column is object type
    elif df[column].dtype == 'object':
        # Plot a bar plot
        plt.figure(figsize=(6, 4))
        sns.countplot(data=df, x=column)
```

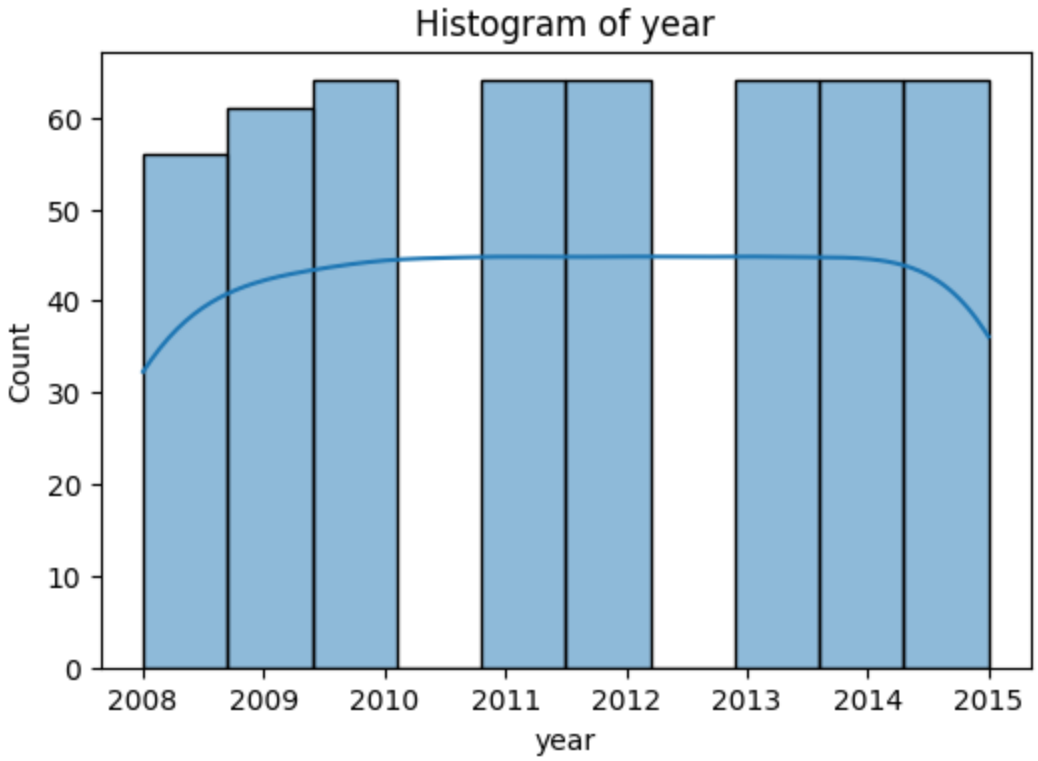
```
            plt.title(f"Bar plot of {column}")
            plt.xticks(rotation=90)
            plt.show()
```

In [5]:

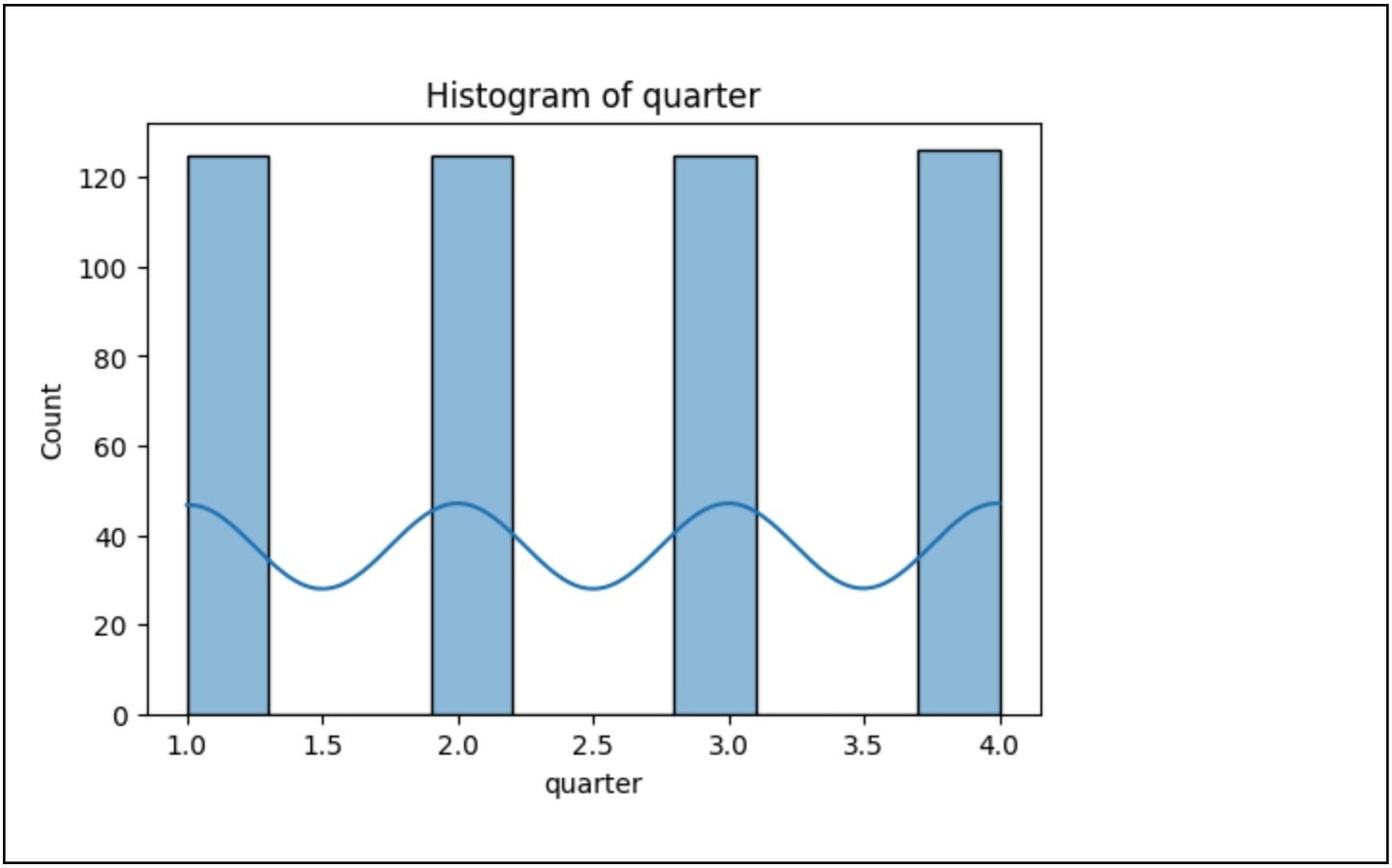
```
perform_eda(df)
```

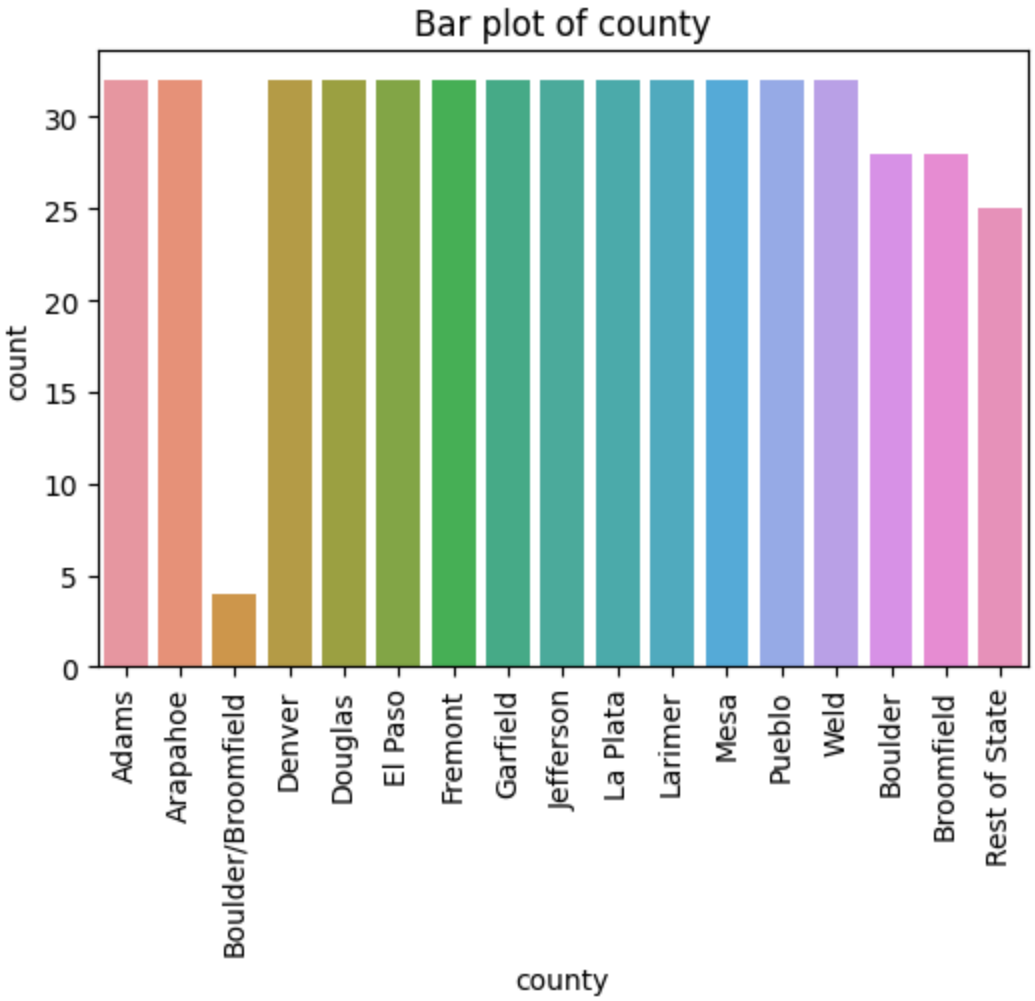
```
Shape of the DataFrame: (501, 4)

Data types:
year         int64
quarter      int64
county      object
sales        int64
dtype: object

Missing values:
year        0
quarter     0
county      0
sales       0
dtype: int64
```

```
Summary statistics:
              year     quarter         sales
count   501.000000  501.000000  5.010000e+02
mean   2011.570858    2.502994  1.760585e+08
std       2.266599    1.120041  1.642055e+08
min    2008.000000    1.000000  6.274000e+06
25%    2010.000000    2.000000  6.148200e+07
50%    2012.000000    3.000000  1.385820e+08
75%    2014.000000    4.000000  2.241580e+08
max    2015.000000    4.000000  9.169100e+08



/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

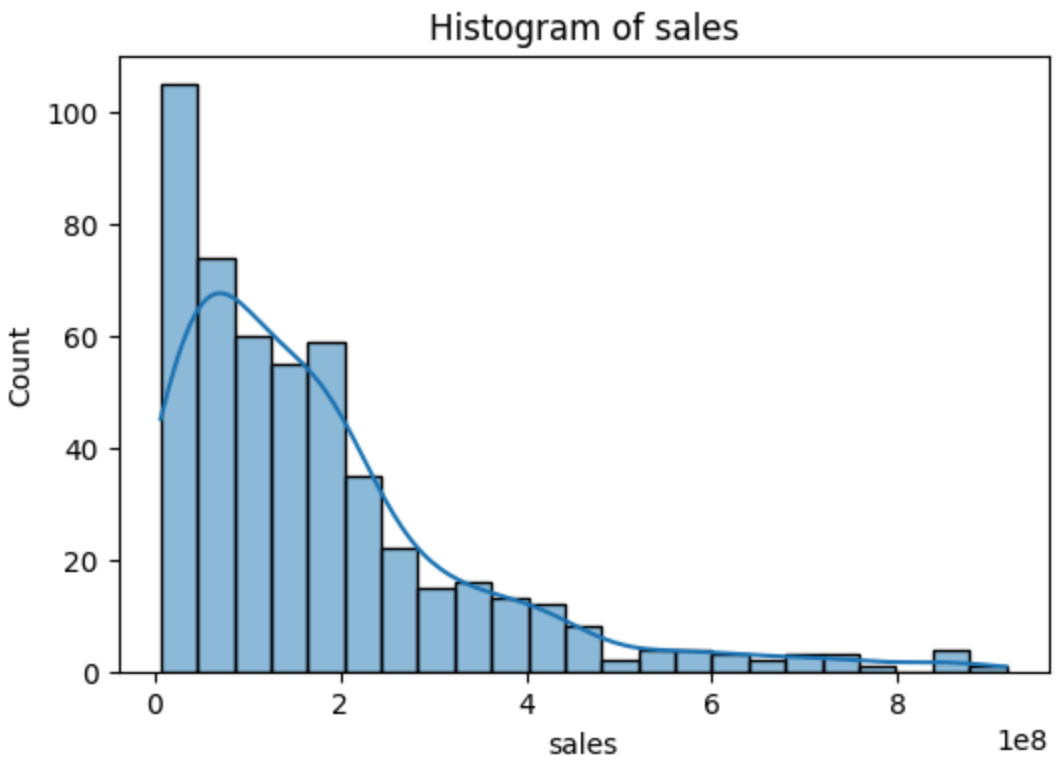



```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Histogram of quarter
Count
quarter
0
20
40
60
80
100
120
1.0
1.5
2.0
2.5
3.0
3.5
4.0




```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

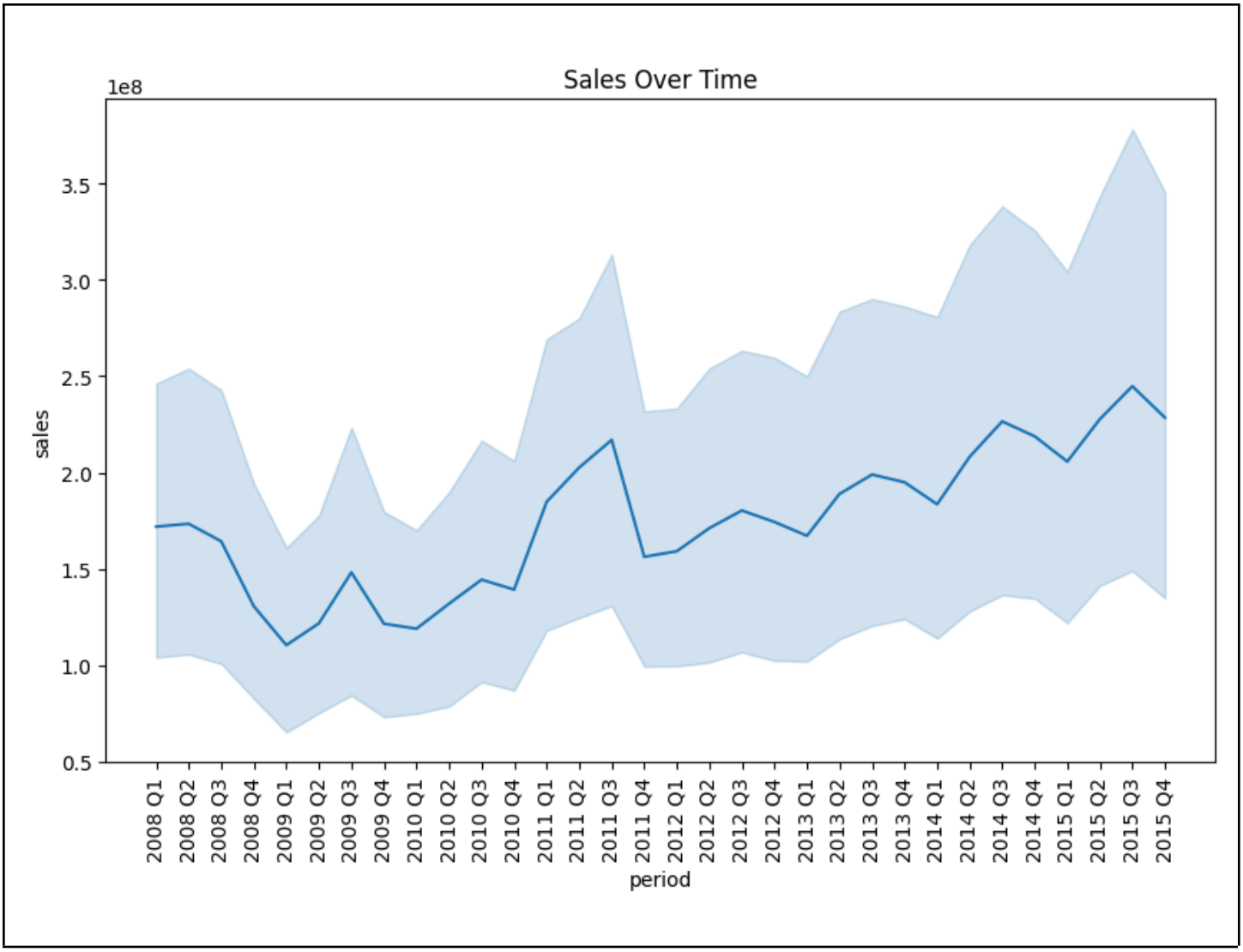


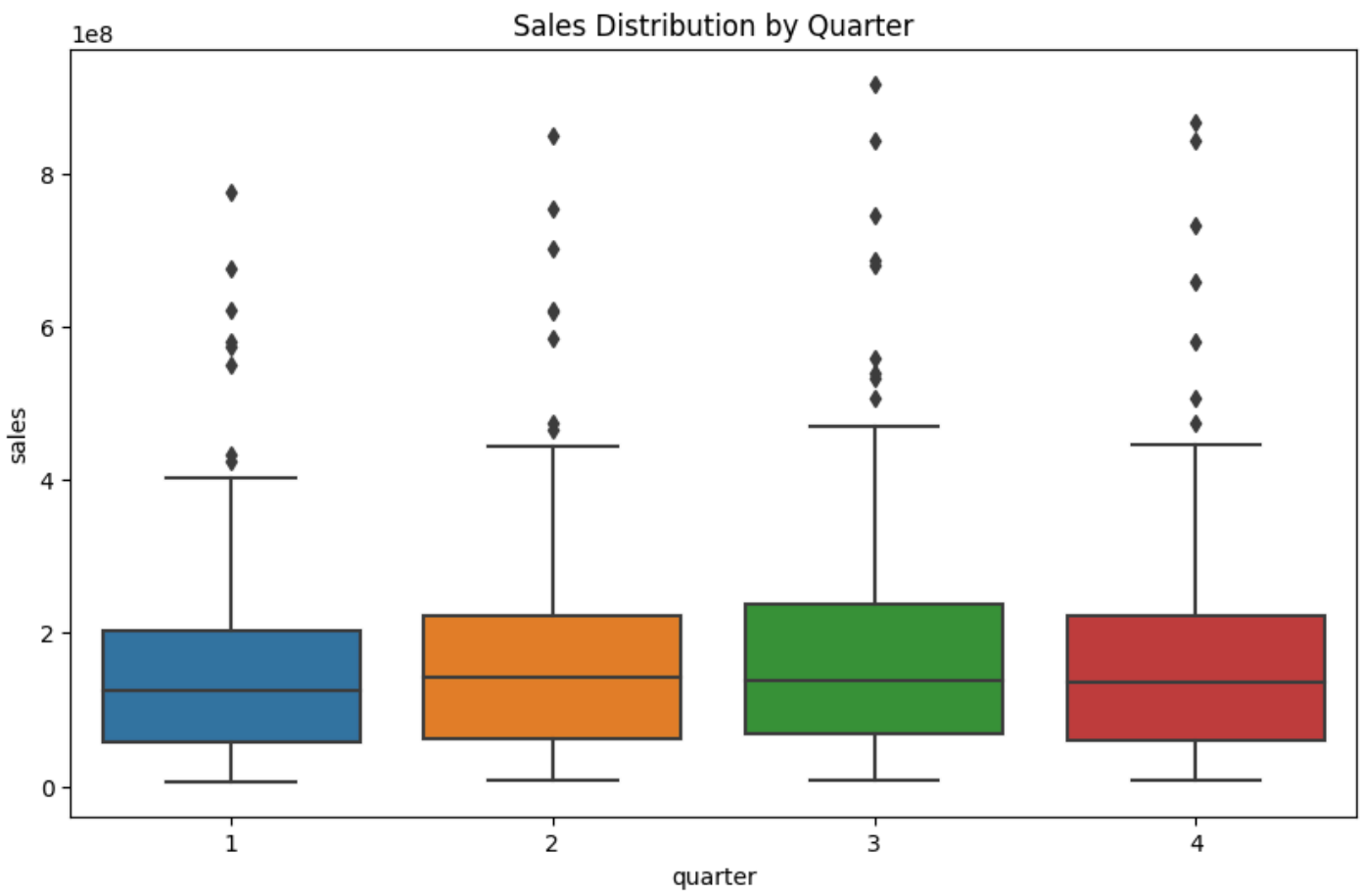
In [6]:

```
# Create a new column that represents the year and quarter
df['period'] = df['year'].astype(str) + ' Q' +
df['quarter'].astype(str)

# Time series plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='period', y='sales')
plt.title('Sales Over Time')
plt.xticks(rotation=90)
plt.show()

# Box plot by quarter
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='quarter', y='sales')
plt.title('Sales Distribution by Quarter')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Sales Over Time
1e8
3.5
3.0
2.5
2.0
1.5
1.0
0.5
sales
2008 Q1
2008 Q2
2008 Q3
2008 Q4
2009 Q1
2009 Q2
2009 Q3
2009 Q4
2010 Q1
2010 Q2
2010 Q3
2010 Q4
2011 Q1
2011 Q2
2011 Q3
2011 Q4
2012 Q1
2012 Q2
2012 Q3
2012 Q4
2013 Q1
2013 Q2
2013 Q3
2013 Q4
2014 Q1
2014 Q2
2014 Q3
2014 Q4
2015 Q1
2015 Q2
2015 Q3
2015 Q4
period



In [7]:

```
import ipywidgets as widgets

def plot_sales_by_county(df, year, quarter):
    # Filter the DataFrame for the selected year and quarter
    filtered_df = df[(df['year'] == year) & (df['quarter'] == quarter)]

    # Group the data by county and sum the sales
    county_sales = filtered_df.groupby('county')['sales'].sum().reset_index()
```

```
    # Sort the counties by sales
    county_sales_sorted = county_sales.sort_values('sales',
ascending=False)

    # Create the bar plot
    plt.figure(figsize=(12, 6))
    sns.barplot(data=county_sales_sorted, x='county',
y='sales', palette='viridis')
    plt.title(f'Sales by County for {year} Q{quarter}')
    plt.xticks(rotation=90)
    plt.ylabel('Total Sales')
    plt.xlabel('County')
    plt.show()

# Create widgets for year and quarter selection
year_widget = widgets.IntSlider(min=df['year'].min(),
max=df['year'].max(), step=1, description='Year:')
quarter_widget = widgets.IntSlider(min=df['quarter'].min(),
max=df['quarter'].max(), step=1, description='Quarter:')

# Use the interact function to create the interactive plot
widgets.interact(lambda year, quarter: plot_sales_by_county(df,
year, quarter), year=year_widget, quarter=quarter_widget)
```

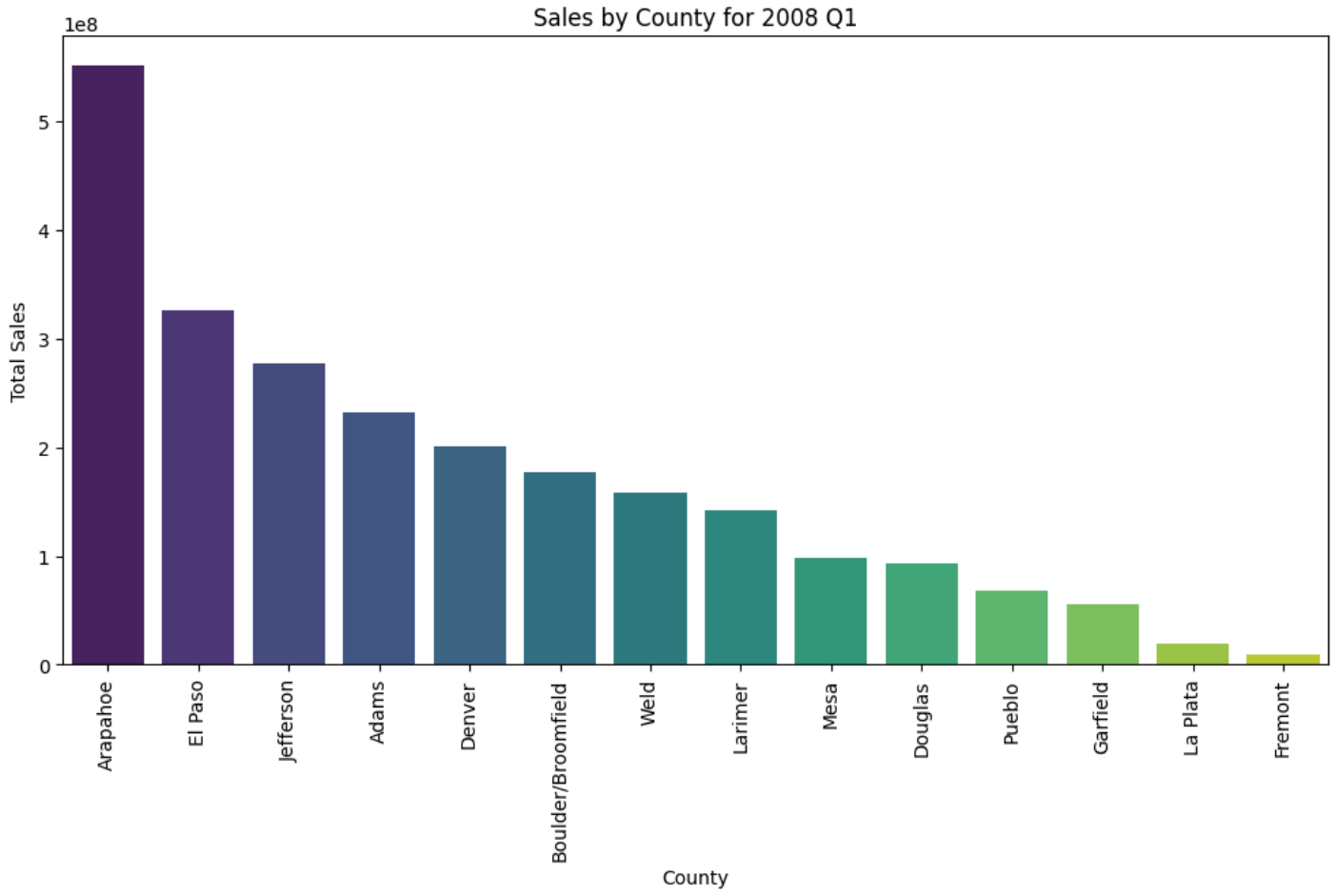
```
```

Year:

2008

Quarter:

1



Out[7]:

<function __main__.<lambda>(year, quarter)>

This code creates two sliders that allow you to select the year and quarter. When you adjust these sliders, the function is called with the selected year and quarter, and it plots the sales for each county for that period.

Machine Learning

DISCLAIMER: That this is just for demonstartion purpose.

The number of features are not enough to make a trustworthy model.

In [8]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Convert 'county' column to categorical
df['county'] = df['county'].astype('category').cat.codes

# Define features and target
X = df[['year', 'quarter', 'county']]
y = df['sales']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the model
model = RandomForestRegressor(n_estimators=100,
```

```
random_state=42)

# Fit the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"RMSE: {rmse}")
```

```
RMSE: 20402876.97387048
```

In [9]:

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
```

```
    'min_samples_split': [2, 5, 10]
}

# Initialize the model
model = RandomForestRegressor(random_state=42)

# Initialize the grid search
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_root_mean_squared_error')

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

print(f"Best parameters: {best_params}")

# Fit the model with the best parameters
model = RandomForestRegressor(**best_params, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"RMSE: {rmse}")
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2,
'n_estimators': 200}

RMSE: 19981856.206187755
```

Reference link