```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from statsmodels.tsa.seasonal import seasonal_decompose
from plotly.subplots import make_subplots
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
```

```
def generate_sales_dataset(start_year, end_year, num_data_points):
    np.random.seed(42)

    # Generate dates spanning the desired time period
    start_date = pd.to_datetime(f"{start_year}-01-01")
    end_date = pd.to_datetime(f"{end_year}-12-31")
    date_range = pd.date_range(start=start_date, end=end_date, freq="D")

    # Generate sales data
    sales = np.random.randint(10, 100, size=num_data_points)

    # Create the dataset
    data = {
        "Date": np.random.choice(date_range, size=num_data_points, replace=False),
        "Sales": sales
    }

    # Create a DataFrame from the data dictionary
    df = pd.DataFrame(data)

    # Sort the DataFrame by date
    df = df.sort_values(by="Date").reset_index(drop=True)

    return df

# Generate a dataset
dataset = generate_sales_dataset(start_year=2015, end_year=2020, num_data_points=100)
dataset.head()
```

	Date	Sales
0	2015-02-14	35
1	2015-02-26	30
2	2015-04-07	23
3	2015-04-11	69
4	2015-04-22	84

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    100 non-null    datetime64[ns]
 1   Sales   100 non-null    int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 1.7 KB
```

```
dataset.isna().sum()
```

```
Date     0
Sales    0
dtype: int64
```

```
#check for duplicates
duplicates = dataset[dataset.duplicated(keep=False)]
if not duplicates.empty:
  print("Duplicate rows found:")
```

```
  print(duplicates)
else:
  print("No duplicates found.")
```

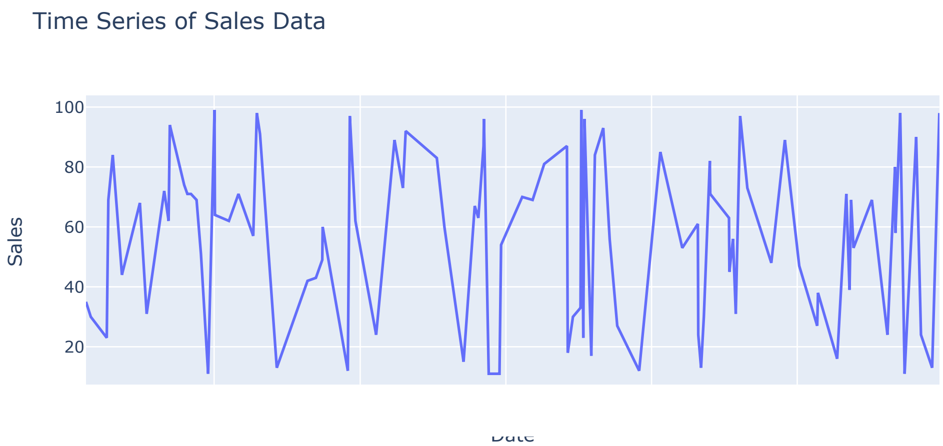
```
No duplicates found.
```

```
dataset.describe()
```

	Sales
count	100.000000
mean	56.680000
std	27.281503
min	11.000000
25%	31.000000
50%	60.500000
75%	80.250000
max	99.000000

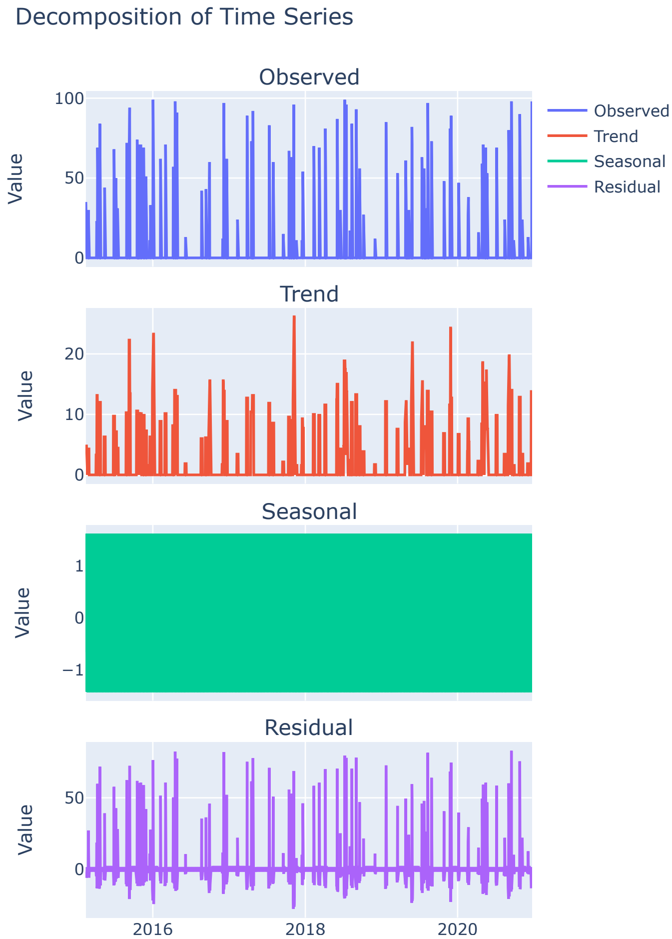
```
# Check for outliers using a box plot
plt.boxplot(dataset['Sales'])
plt.title('Sales Distribution')
plt.show()
```


```
# date column to datetime format
dataset['Date'] = pd.to_datetime(dataset['Date'])

# Set index
dataset.set_index('Date', inplace=True)


#line plot of time series
fig = px.line(dataset, x=dataset.index, y='Sales', title='Time Series of Sales Data')
fig.update_layout(
    xaxis_title='Date',
    yaxis_title='Sales',
    width=800,
    height=400,
)
# Show the plot
fig.show()
```



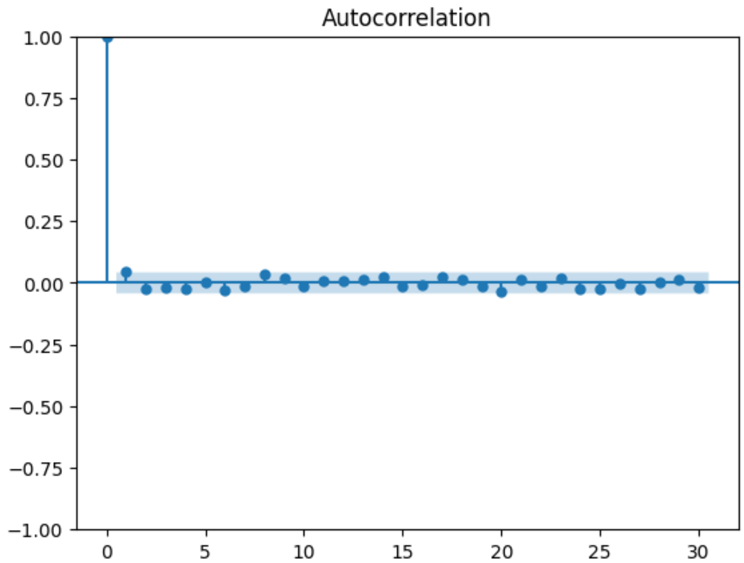

```
# Resample the data to a fixed frequency to daily
dataset_resampled = dataset.resample('D').sum()

# Decompose the time series into trend, seasonal, and residual components
decomposition = seasonal_decompose(dataset_resampled['Sales'], model='additive')

# Create a subplot grid
fig = make_subplots(rows=4, cols=1, shared_xaxes=True, vertical_spacing=0.05,
                    subplot_titles=['Observed', 'Trend', 'Seasonal', 'Residual'])

# Add traces to each subplot
fig.add_trace(go.Scatter(x=dataset_resampled.index, y=decomposition.observed, mode='lines', name='Observed'), row=1, col=1)
fig.add_trace(go.Scatter(x=dataset_resampled.index, y=decomposition.trend, mode='lines', name='Trend'), row=2, col=1)
fig.add_trace(go.Scatter(x=dataset_resampled.index, y=decomposition.seasonal, mode='lines', name='Seasonal'), row=3, col=1)
fig.add_trace(go.Scatter(x=dataset_resampled.index, y=decomposition.resid, mode='lines', name='Residual'), row=4, col=1)

#titles and axis labels
fig.update_layout(title='Decomposition of Time Series', xaxis_title='Date', height=800)
fig.update_yaxes(title_text='Value', row=1, col=1)
fig.update_yaxes(title_text='Value', row=2, col=1)
fig.update_yaxes(title_text='Value', row=3, col=1)
fig.update_yaxes(title_text='Value', row=4, col=1)

# Show the plot
fig.show()
```

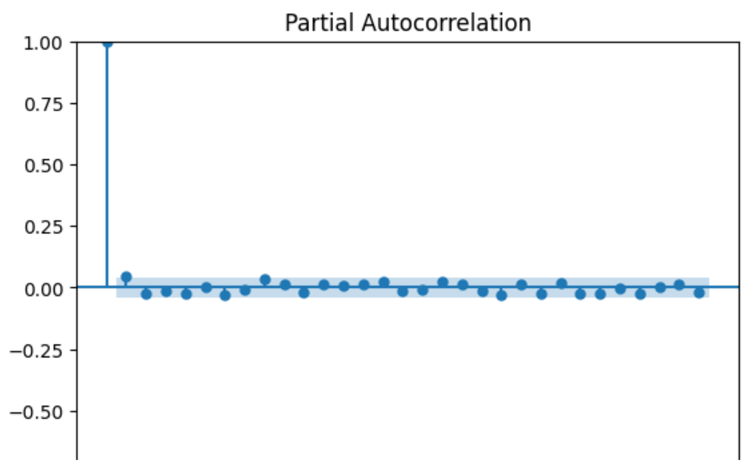
Date




```
# Conduct autocorrelation and partial autocorrelation analysis
plt.figure(figsize=(12, 6))

# Autocorrelation plot
plot_acf(dataset_resampled['Sales'], lags=30, alpha=0.05)

# Partial autocorrelation plot
plot_pacf(dataset_resampled['Sales'], lags=30, alpha=0.05)

plt.show()
```

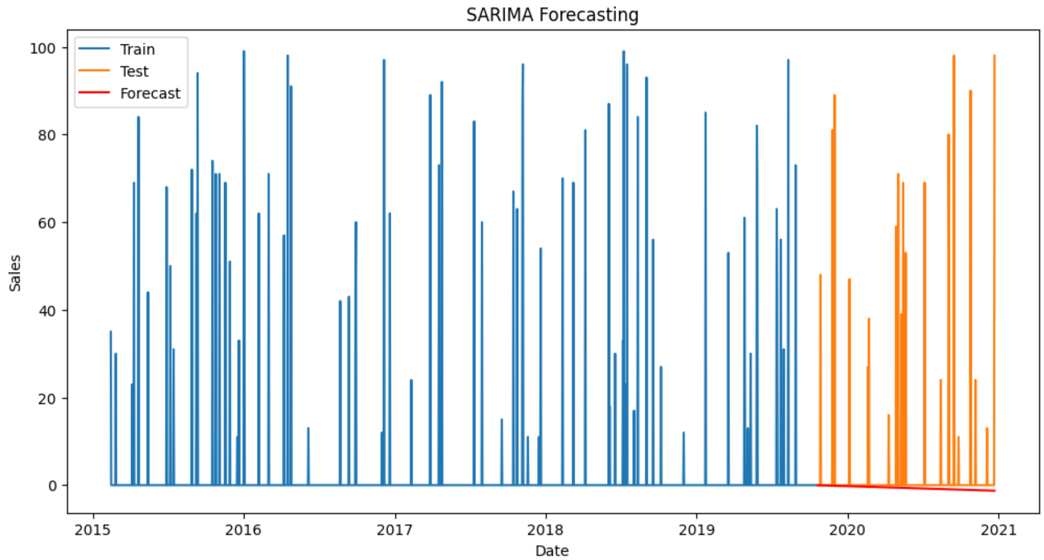
```
<Figure size 1200x600 with 0 Axes>
```





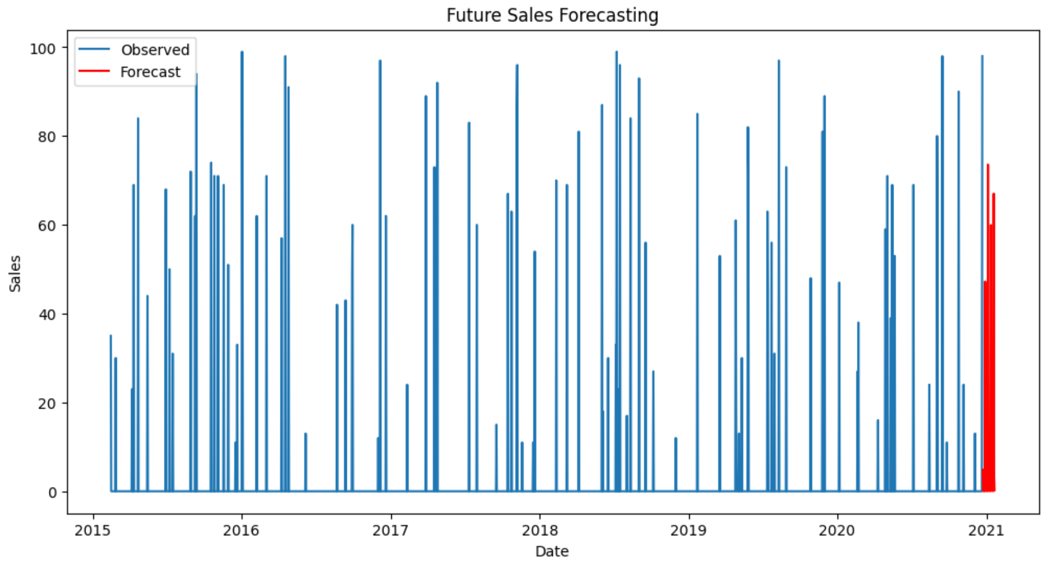

```
# Divide the dataset into training and testing sets
train_size = int(0.8 * len(dataset_resampled))
train_data, test_data = dataset_resampled[:train_size], dataset_resampled[train_size:]

# Initialize and fit the SARIMA model
order = (1, 1, 1)
seasonal_order = (1, 1, 0, 7)
sarima_model = SARIMAX(train_data['Sales'], order=order, seasonal_order=seasonal_order)
sarima_results = sarima_model.fit()

# Forecast using the trained model
forecast = sarima_results.get_forecast(steps=len(test_data))
forecast_mean = forecast.predicted_mean

# Evaluation using RMSE
rmse = np.sqrt(mean_squared_error(test_data['Sales'], forecast_mean))
print("Root Mean Squared Error (RMSE):", rmse)

# Plot the observed vs. predicted values for the testing period
plt.figure(figsize=(12, 6))
plt.plot(train_data.index, train_data['Sales'], label='Train')
plt.plot(test_data.index, test_data['Sales'], label='Test')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red')
plt.title('SARIMA Forecasting')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()
```

```
Root Mean Squared Error (RMSE): 14.025002128844342
```




```
# Train the SARIMA model using the entire dataset
order = (1, 1, 1)
seasonal_order = (1, 1, 0, 7)
sarima_model = SARIMAX(dataset_resampled['Sales'], order=order, seasonal_order=seasonal_order)
sarima_results = sarima_model.fit()

# Forecast future sales
forecast_steps = 30
forecast = sarima_results.get_forecast(steps=forecast_steps)
forecast_mean = forecast.predicted_mean
forecast_index = pd.date_range(start=dataset_resampled.index[-1], periods=forecast_steps + 1, closed='right')

# Plot the observed and forecasted sales for the future period
plt.figure(figsize=(12, 6))
plt.plot(dataset_resampled.index, dataset_resampled['Sales'], label='Observed')
plt.plot(forecast_index, forecast_mean, label='Forecast', color='red')
plt.title('Future Sales Forecasting')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()
```

```
<ipython-input-47-8e4b218b97f6>:11: FutureWarning:

Argument `closed` is deprecated in favor of `inclusive`.
```



0s completed at 5:21 PM