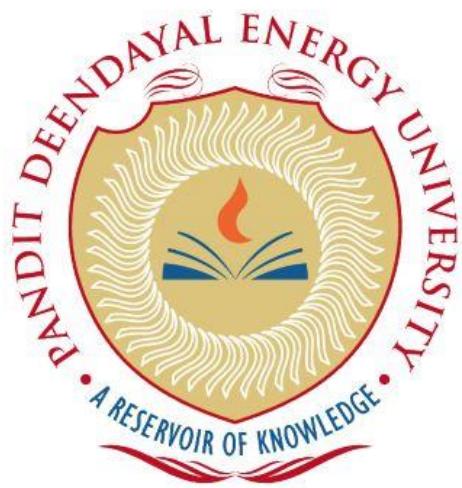


**PANDIT DEENDAYAL ENERGY UNIVERSITY**  
**SCHOOL OF TECHNOLOGY**



**Machine Learning Lab**

**20CP401P**

**LAB MANUAL**

**B.Tech. (Computer Science and Engineering)**

**Semester 7**

**Submitted To:**

Dr. Archana Nigam

**Submitted By:**

HARSH SHAH

21BCP359

G11 Batch

# INDEX

<b>Exp. No.</b>	<b>Assignment Title</b>	<b>Page No.</b>
<b>1</b>	Discuss and analyse different data visualization tools.	1
<b>2</b>	Measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are available.	12
<b>3</b>	Implement simple and multi-linear regression to predict profits for a food truck. Compare the performance of the model on linear and multi-linear regression.	29
<b>4</b>	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs	34
<b>5</b>	For a given set of training data examples stored in a .CSV file, implement and demonstrate various feature selection algorithms and compare the performance of the algorithms.	38
<b>6</b>	Apply Different Machine Learning Approaches for the Classification Task. Compare the Performance of Different ML Approaches in Terms of Accuracy, Precision, and Recall.	44
<b>7</b>	Train a Machine Learning Classifier on an Imbalanced Dataset. Then Balance the Dataset Using Oversampling Techniques. Compare the Model Performance Before and After Oversampling.	49
<b>8</b>	Apply Different Feature Selection Approaches for the Classification/Regression Task. Compare the Performance of Different Feature Selection Approaches.	53
<b>9</b>	Write a Program to Demonstrate the Working of the Decision Tree-Based CART Algorithm. Build the Decision Tree and Classify a New Sample Using a Suitable Dataset. Compare the Performance with That of ID3, C4.5, and CART in Terms of Accuracy, Recall, Precision, and Sensitivity.	58
<b>10</b>	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	65

## LAB ASSIGNMENT 1

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Discuss and analyze different data visualization tools.		

### Objective

The objective of this lab assignment is to explore and analyse various data visualization tools used for representing and understanding complex datasets. Through this assignment, you will gain insights into the strengths, weaknesses, and practical applications of different visualization tools.

Dataset Used: [U.S. International Air Traffic data \(1990-2020\) - Kaggle](#)

### Introduction to Data Visualization

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. It's an essential skill in the data analysis process, helping to make complex data more understandable, insightful, and actionable.

Key Types of Data Visualizations:

- **Charts:** Including bar charts, line charts, pie charts, and histograms, these are used to compare different categories or show changes over time.
- **Graphs:** Such as scatter plots and area graphs, these help in understanding relationships between variables.
- **Maps:** Geographic data can be visualized using maps to highlight spatial patterns and distributions.
- **Infographics:** These combine various visualization elements to tell a story or provide a comprehensive overview of data.
- **Dashboards:** Interactive platforms that allow users to manipulate data and visuals to explore different perspectives and insights.

### Data Visualisation Tools

- **Microsoft Excel:** A widely used tool that offers basic to advanced charting and graphing capabilities.
- **Python Libraries:** Libraries like Matplotlib, Seaborn, and Plotly are popular among data scientists for creating a wide range of static, animated, and interactive visualizations.
- **Tableau:** A powerful tool for creating interactive and shareable dashboards.
- **Power BI:** A business analytics service by Microsoft that provides interactive visualizations and business intelligence capabilities.
- **Google Data Studio:** A free tool that transforms your data into informative, easy-to-read, and shareable dashboards and reports.
- **R/JS Libraries:** Libraries like ggplot2(R), plotly(R), D3.js(JS), plotly(JS) are popular among for creating a wide range of static, animated, and interactive visualizations.

## Brief Overview of Tool Capabilities and Features

### a) Matplotlib (Python)

- Matplotlib is a versatile plotting library in Python that produces publication-quality figures in a variety of formats and interactive environments.
- Wide Range of Plots: Supports line plots, scatter plots, bar charts, histograms, pie charts, error bars, and more.
- Subplots and Axes: Allows creation of complex, multi-plot layouts.
- Use Cases: Scientific Research, Financial Analysis

### b) Seaborn (Python)

- Seaborn is a statistical data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- Complex Plots Simplified: Simplifies the creation of complex visualizations such as heatmaps, time series, violin plots, and pair plots.
- Statistical Plotting: Integrates with pandas Data Frames and supports statistical aggregation and plotting.
- Use Cases: Exploratory Data Analysis (EDA), Statistical Data Visualization

### c) Power BI

- Power BI is a business analytics service by Microsoft that delivers insights to enable fast, informed decisions.
- Interactive Dashboards: Allows creation of interactive and shareable dashboards.
- Data Connectivity: Connects to a wide variety of data sources, including databases, web services, and Excel spreadsheets.
- Data Modelling: Offers robust data modelling and transformation capabilities.
- Real-Time Analytics: Supports real-time data visualization and streaming analytics.
- Use Cases: Business Intelligence and Reporting, Real-Time Data Monitoring

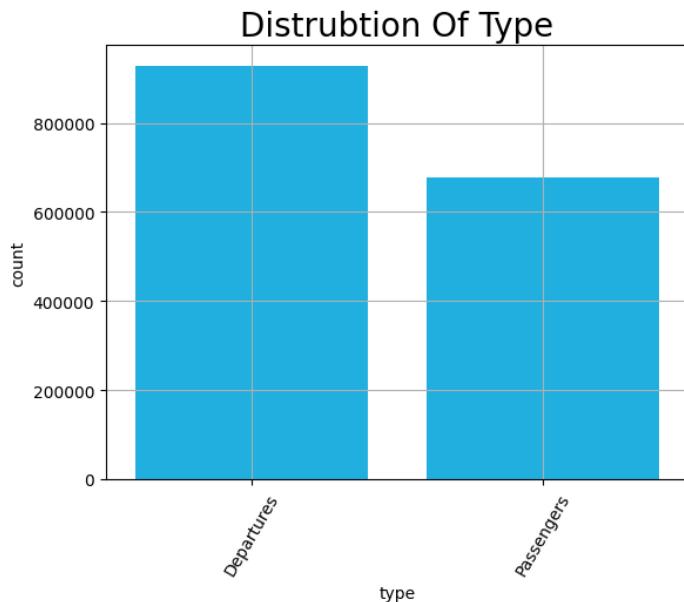
### d) Plotly (Python)

- Plotly is an interactive graphing library for Python, known for its high-quality and interactive plots.
- Interactive Graphs: Produces interactive plots that can be embedded in web applications or dashboards.
- Dash by Plotly: Integrated with Dash, a framework for building interactive web applications with Python.
- Ease of Use: User-friendly API and integration with pandas for quick and easy plotting.
- Cross-Platform: Plots can be viewed in Jupyter Notebooks, standalone HTML files, or web applications.
- Use Cases: Interactive Web Applications, 3D Data Visualization

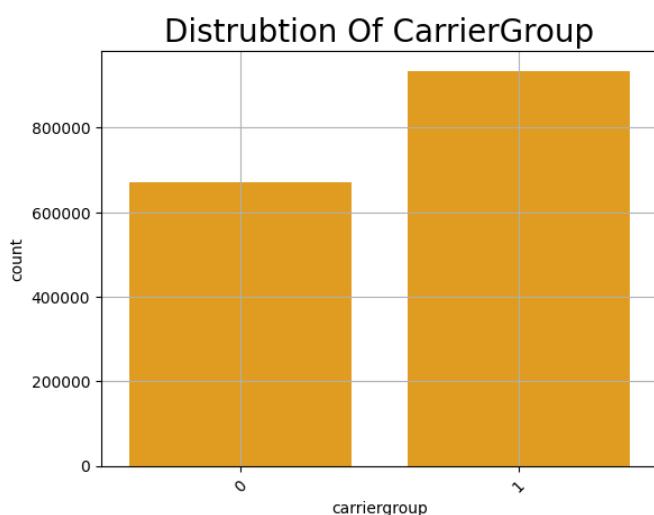
## Code Snippets

### 1. Using Seaborn and Matplotlib

```
sns.countplot(x='type', data=full_data, color="deepskyblue")
plt.title('Distrubtion Of Type', fontsize=20, color="black")
plt.grid(True)
plt.xticks(rotation=60)
plt.show()
```



```
sns.countplot(x='carriergroup', data=full_data, color="orange")
plt.title('Distribution Of CarrierGroup', fontsize=20)
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```

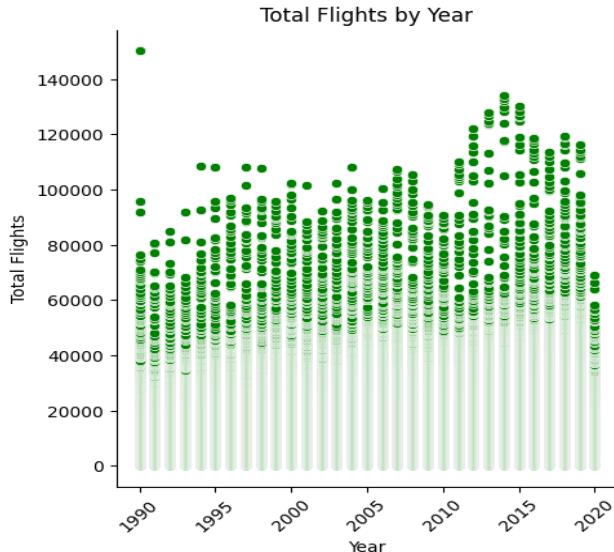


```
plt.figure(figsize=(10, 6))
sns.relplot(x='Year', y='Total', data=full_data, color='deepskyblue')
```

```

plt.xlabel('Year')
plt.ylabel('Total Flights')
plt.title('Total Flights by Year')
plt.xticks(rotation=45)
plt.show()

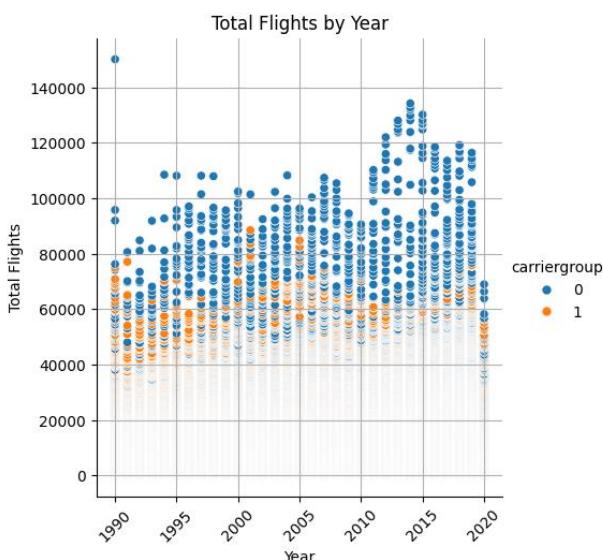
```



```

plt.figure(figsize=(10, 6))
sns.relplot(x='Year', y='Total', hue='carriergroup', data=full_data, color='green')
plt.xlabel('Year')
plt.ylabel('Total Flights')
plt.title('Total Flights by Year')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```



```

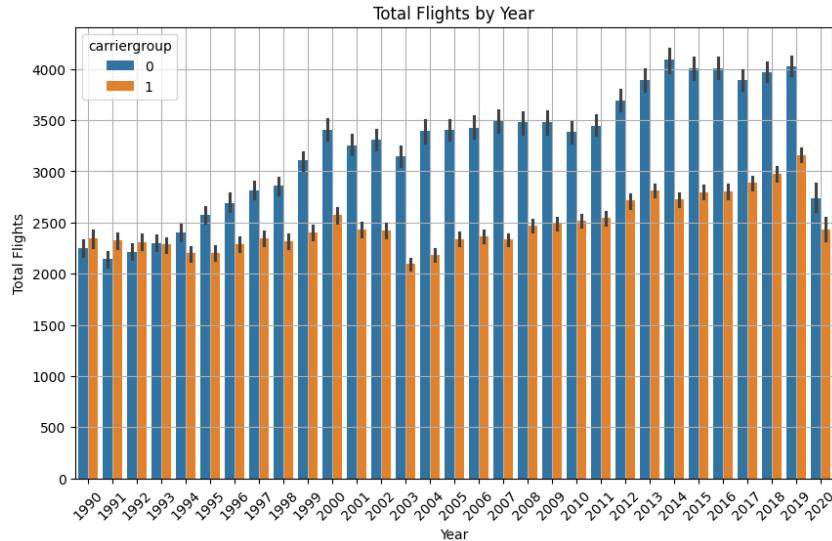
plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Total', hue='carriergroup', data=full_data)
plt.xlabel('Year')

```

```

plt.ylabel('Total Flights')
plt.title('Total Flights by Year')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

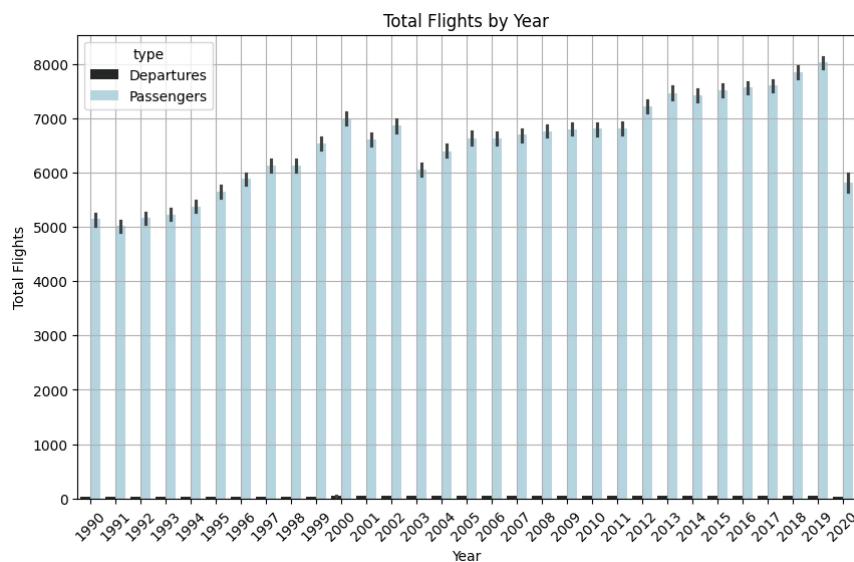
```



```

plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Total', hue='type', data=full_data, color='lightblue' )
plt.xlabel('Year')
plt.ylabel('Total Flights')
plt.title('Total Flights by Year')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

```



```

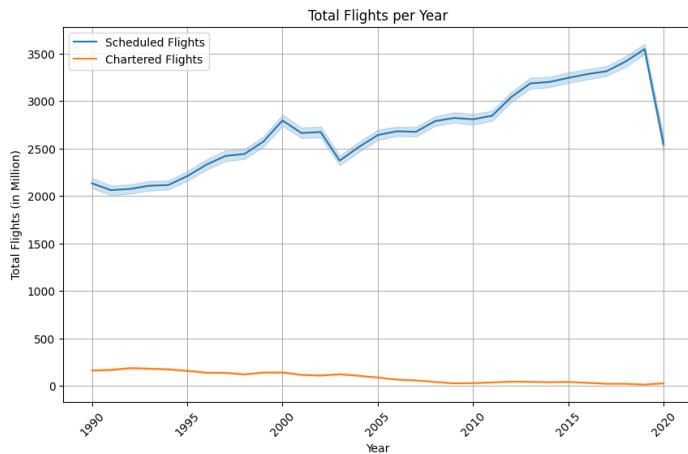
plt.figure(figsize=(10, 6))
sns.lineplot(x='Year', y='Scheduled', data=full_data, label='Scheduled Flights')
sns.lineplot(x='Year', y='Charter', data=full_data, label='Chartered Flights')

```

```

sns.color_palette("flare", as_cmap=True)
plt.title('Total Flights per Year')
plt.xlabel('Year')
plt.ylabel('Total Flights (in Million)')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.show()

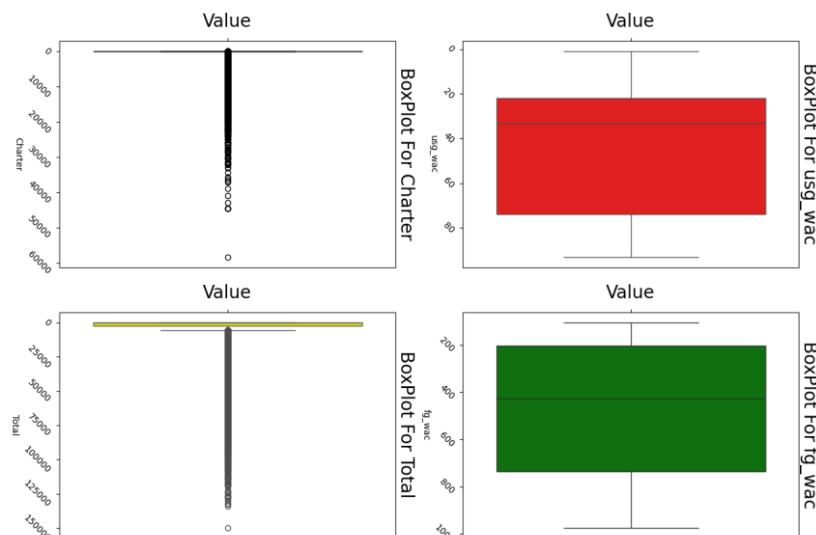
```



```

plt.figure(figsize=(10,15))
x=1
for i,j in zip(['usg_wac','fg_wac','Charter','Total'],['red','green','black','yellow']):
    plt.subplot(2,2,x)
    sns.boxplot(x=i,data=full_data,color=j)
    plt.ylabel('Value',fontsize=20)
    plt.title(f'BoxPlot For {i}',fontsize=20)
    plt.xticks(rotation=45)
    x+=1

```

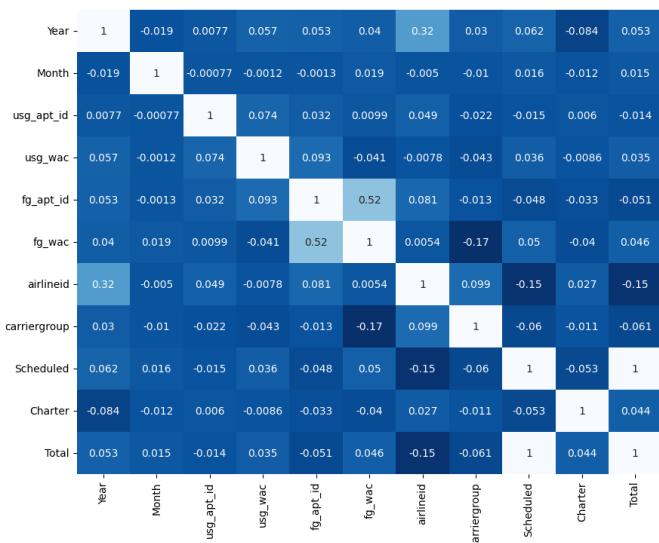


```

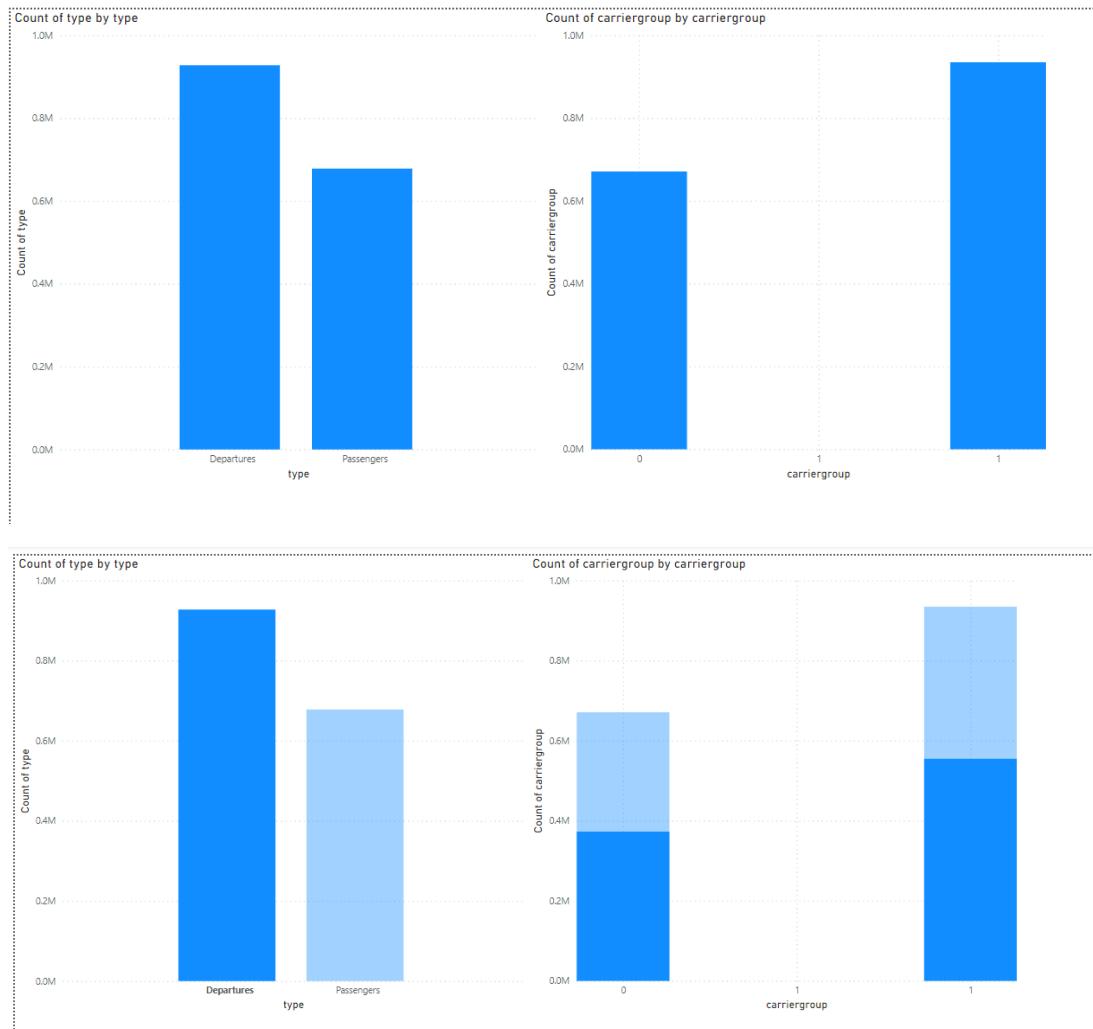
numerical_data = full_data.select_dtypes(include=['number'])
correlation_matrix = numerical_data.corr()
plt.figure(figsize=(10, 8))

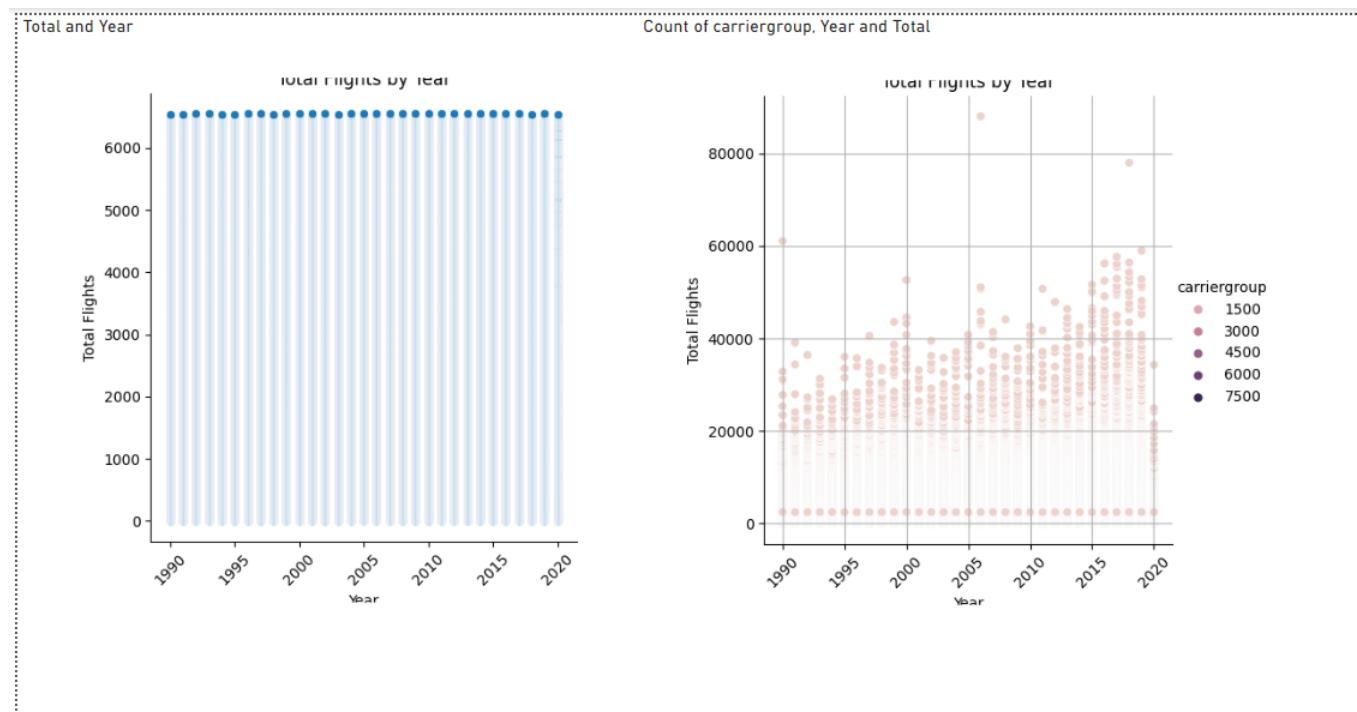
```

```
sns.heatmap(correlation_matrix, cmap='Blues_r', annot=True, cbar=False)
```

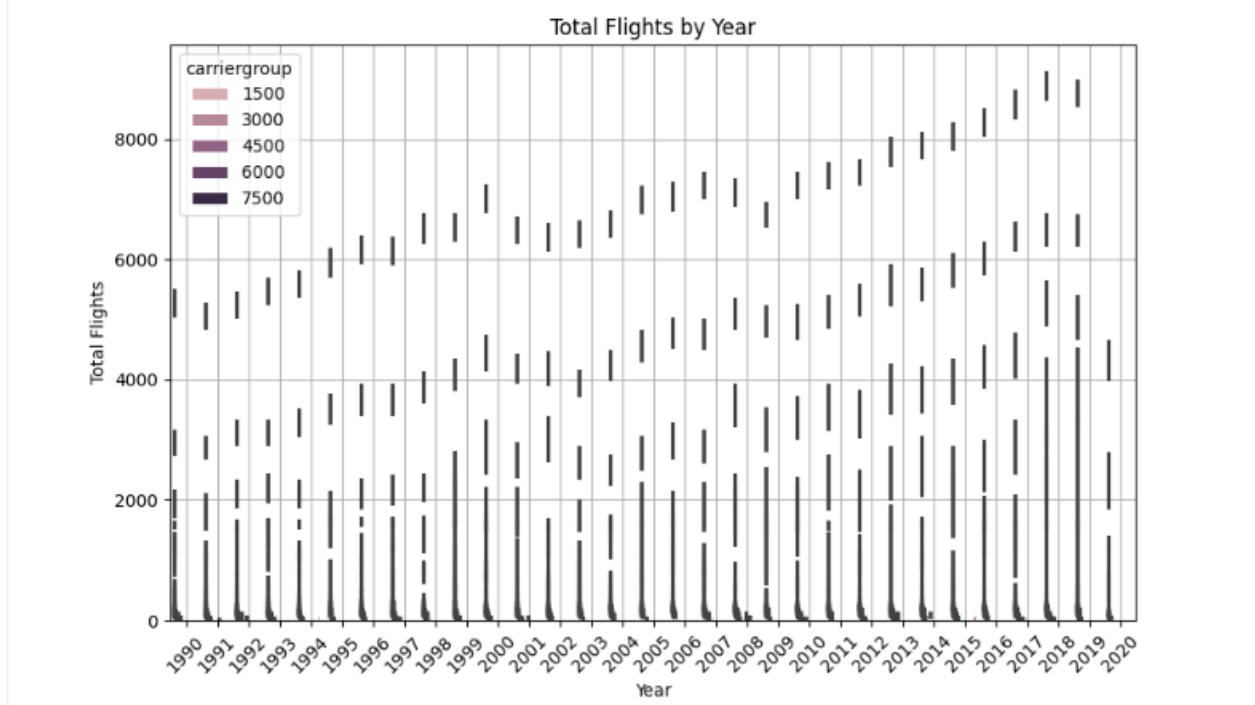


## 2. Using POWERBI



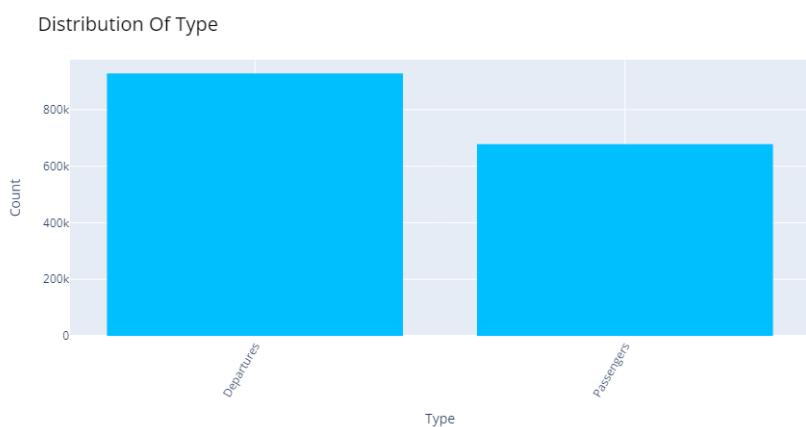


Count of carriergroup, Total and Year



### 3. Using Plotly (Python)

```
fig = px.histogram(full_data, x='type', color_discrete_sequence=['deepskyblue'])
fig.update_layout(
    title='Distribution Of Type',
    title_font_size=20,
    title_font_color='black',
    xaxis_title='Type',
    yaxis_title='Count',
    xaxis_tickangle=-60,
    xaxis_showgrid=True,
    yaxis_showgrid=True
)
fig.show()
```

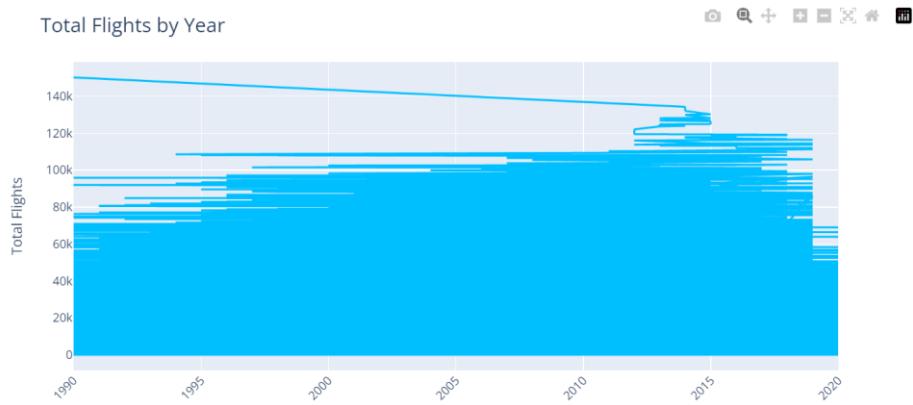


```
fig = px.histogram(full_data, x='carriergroup', color_discrete_sequence=['orange'])
fig.update_layout(
    title='Distribution Of CarrierGroup',
    title_font_size=20,
    xaxis_title='CarrierGroup',
    yaxis_title='Count',
    xaxis_tickangle=-45,
    xaxis_showgrid=True,
    yaxis_showgrid=True
)
fig.show()
```

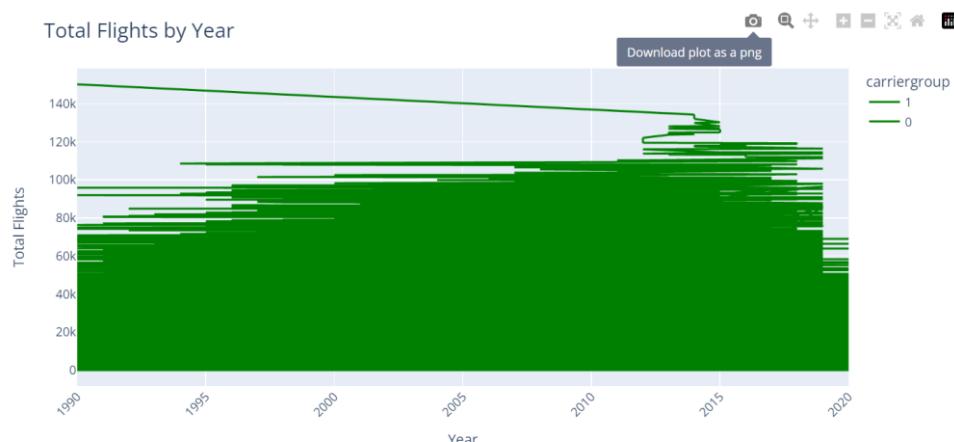


```
fig = px.line(full_data, x='Year', y='Total', line_shape='linear', color_discrete_sequence=['deepskyblue'])
```

```
fig.update_layout(
    title='Total Flights by Year',
    title_font_size=20,
    xaxis_title='Year',
    yaxis_title='Total Flights',
    xaxis_tickangle=-45
)
fig.show()
```



```
fig = px.line(full_data, x='Year', y='Total', color='carriergroup', line_shape='linear',
color_discrete_sequence=['green'])
fig.update_layout(
    title='Total Flights by Year',
    title_font_size=20,
    xaxis_title='Year',
    yaxis_title='Total Flights',
    xaxis_tickangle=-45,
    xaxis_showgrid=True,
    yaxis_showgrid=True
)
fig.show()
```



```
# Compute the correlation matrix
numerical_data = full_data.select_dtypes(include=['number'])
```

```
correlation_matrix = numerical_data.corr()
```

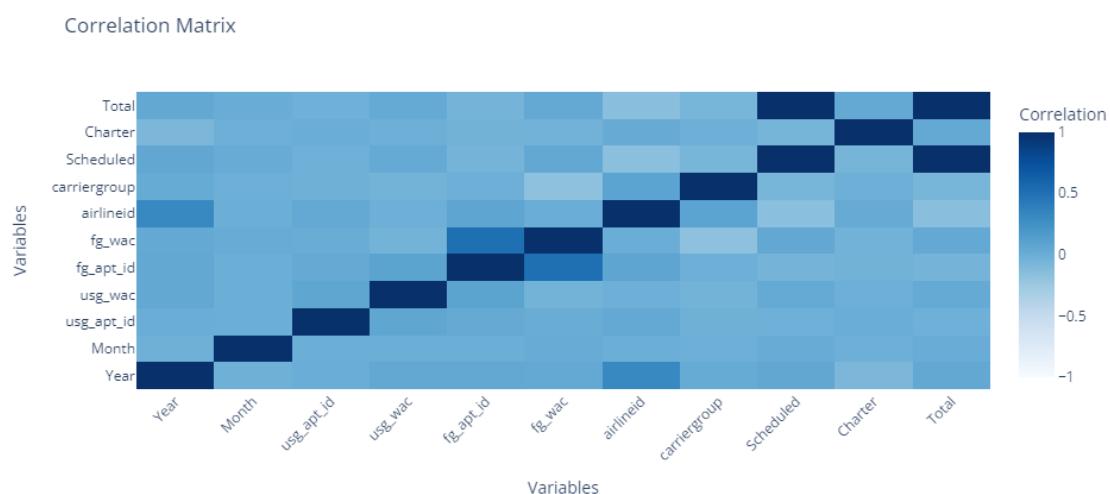
# Create the heatmap

```
fig = go.Figure(data=go.Heatmap(
    z=correlation_matrix.values,
    x=correlation_matrix.columns,
    y=correlation_matrix.columns,
    colorscale='Blues',
    colorbar_title='Correlation',
    zmid=0
))
```

# Update layout

```
fig.update_layout(
    title='Correlation Matrix',
    xaxis_title='Variables',
    yaxis_title='Variables',
    xaxis_nticks=len(correlation_matrix.columns),
    yaxis_nticks=len(correlation_matrix.columns),
    xaxis_tickangle=-45
)
```

```
fig.show()
```



## LAB ASSIGNMENT 2

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are available.		

### Objective

The objective of this lab assignment is to explore and analyse a dataset containing measurements of electric power consumption in a household over a period of almost 4 years. You will perform various data visualization tasks to gain insights into electrical quantities, sub-metering values, and overall trends.

Dataset: <https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption>

### Task – 1: Load the data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv( ./data/household_power_consumption.txt', sep=';', _values=['nan', '?'] )
df.index = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True)
df.index.name = 'dt'
df = df.drop(columns = ['Date', 'Time'])
```

### # Data Cleaning

```
df.isna().sum()
```

Global_active_power	25979
Global_reactive_power	25979
Voltage	25979
Global_intensity	25979
Sub_metering_1	25979
Sub_metering_2	25979
Sub_metering_3	25979
<b>dtype: int64</b>	

```
df.dropna(inplace=True)
```

```
df.isna().sum()
```

```
Global_active_power      0  
Global_reactive_power    0  
Voltage                  0  
Global_intensity         0  
Sub_metering_1            0  
Sub_metering_2            0  
Sub_metering_3            0  
dtype: int64
```

### Task – 2: Subset the data from the given dates (December 2006 and November 2009)

```
start_date = pd.Timestamp('2006-12-01')
```

```
end_date = pd.Timestamp('2009-11-30')
```

```
newdf = df.loc[start_date:end_date]
```

### Task – 3: Create a histogram

```
plt.figure(figsize=(14, 10))
```

```
for i, column in enumerate(newdf.columns, 1):
```

```
    plt.subplot(3, 3, i)
```

```
    sns.histplot(newdf[column], kde=True, bins=10)
```

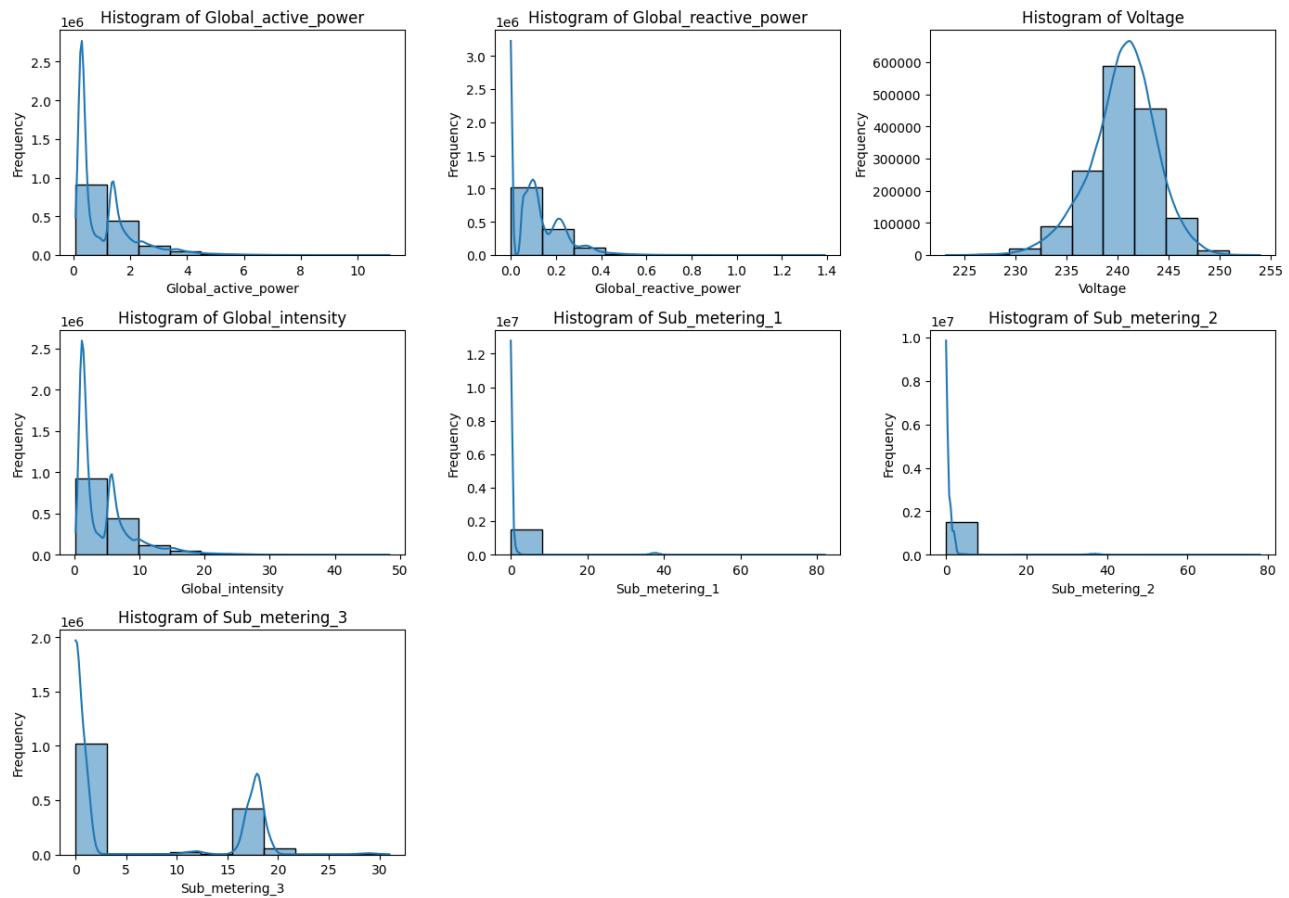
```
    plt.title(f'Histogram of {column}')
```

```
    plt.xlabel(column)
```

```
    plt.ylabel('Frequency')
```

```
    plt.tight_layout()
```

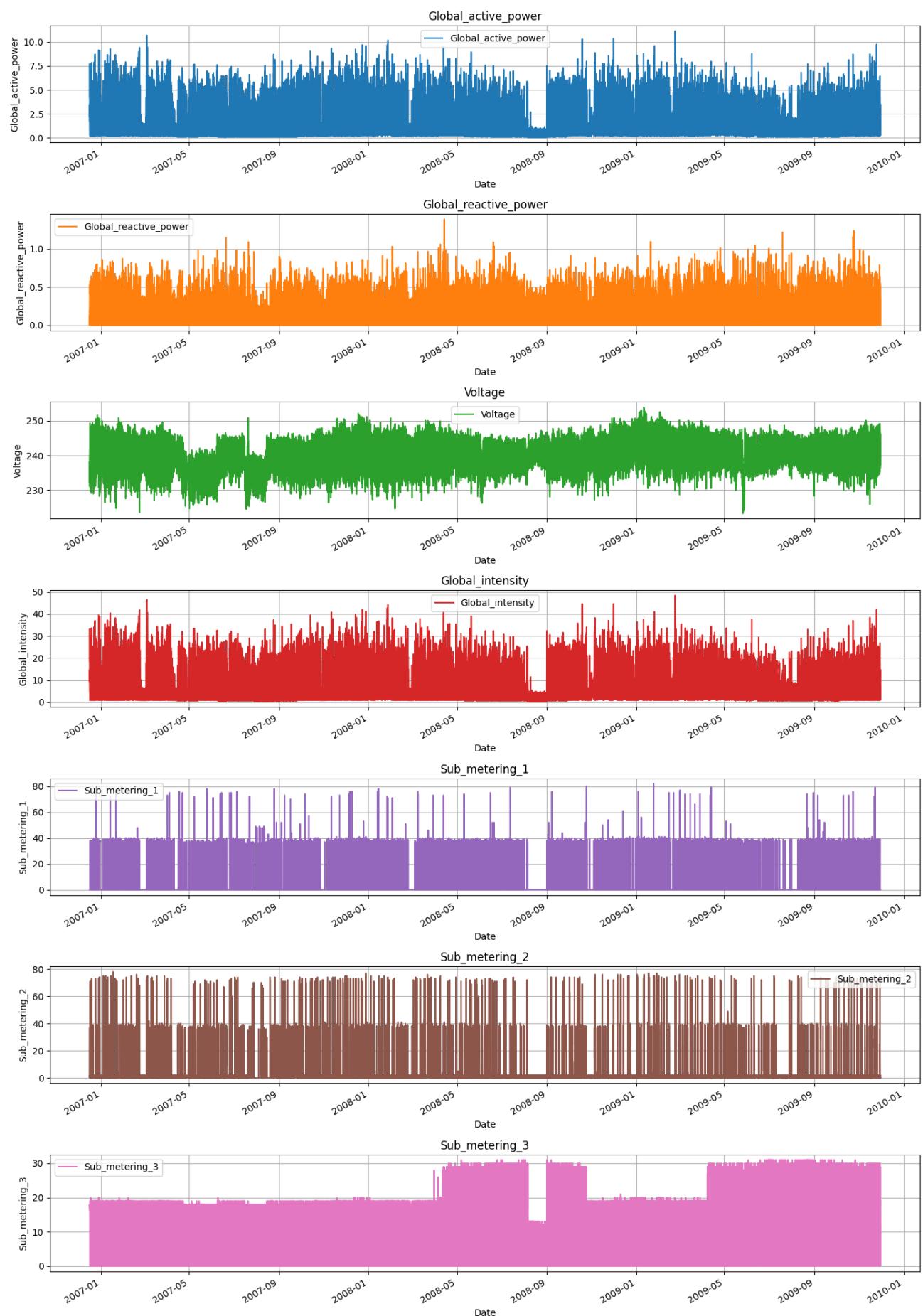
```
    plt.show()
```



#### Task – 4: Create a Time series

```
plt.figure(figsize=(14, 20))

for i, column in enumerate(newdf.columns, 1):
    plt.subplot(7, 1, i)
    newdf[column].plot(title=column, xlabel='Date', ylabel=column, legend=True)
    plt.grid(True)
plt.tight_layout()
plt.show()
```



### Task – 5: Create a plot for sub metering

```
df_melted = newdf.reset_index().melt(id_vars='dt', value_vars=['Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3'])

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

sns.lineplot(data=df_melted, x='dt', y='value', hue='variable')

plt.title('Sub Metering Time Series')

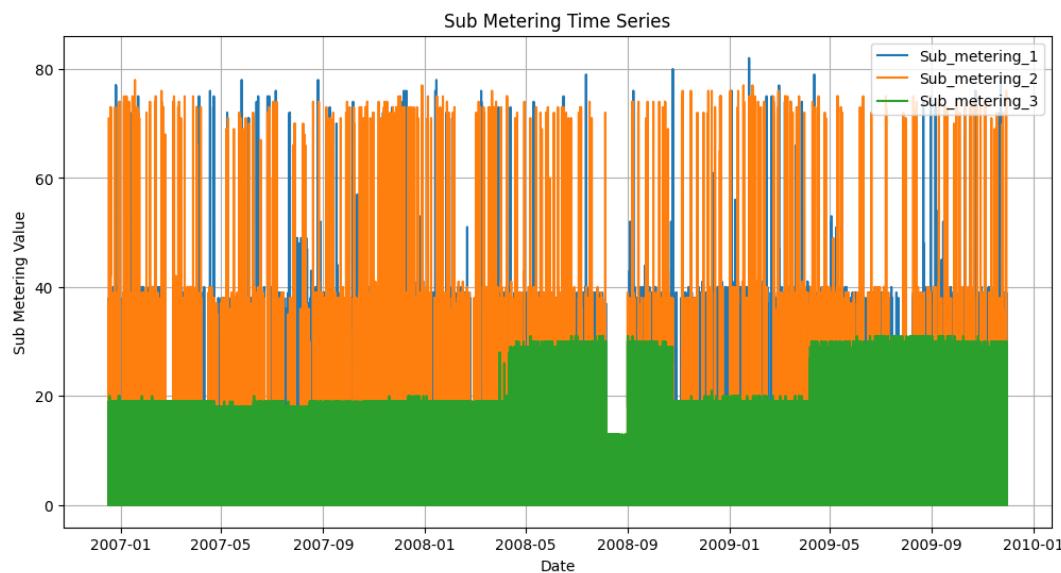
plt.xlabel('Date')

plt.ylabel('Sub Metering Value')

plt.legend(loc='upper right')

plt.grid(True)

plt.show()
```



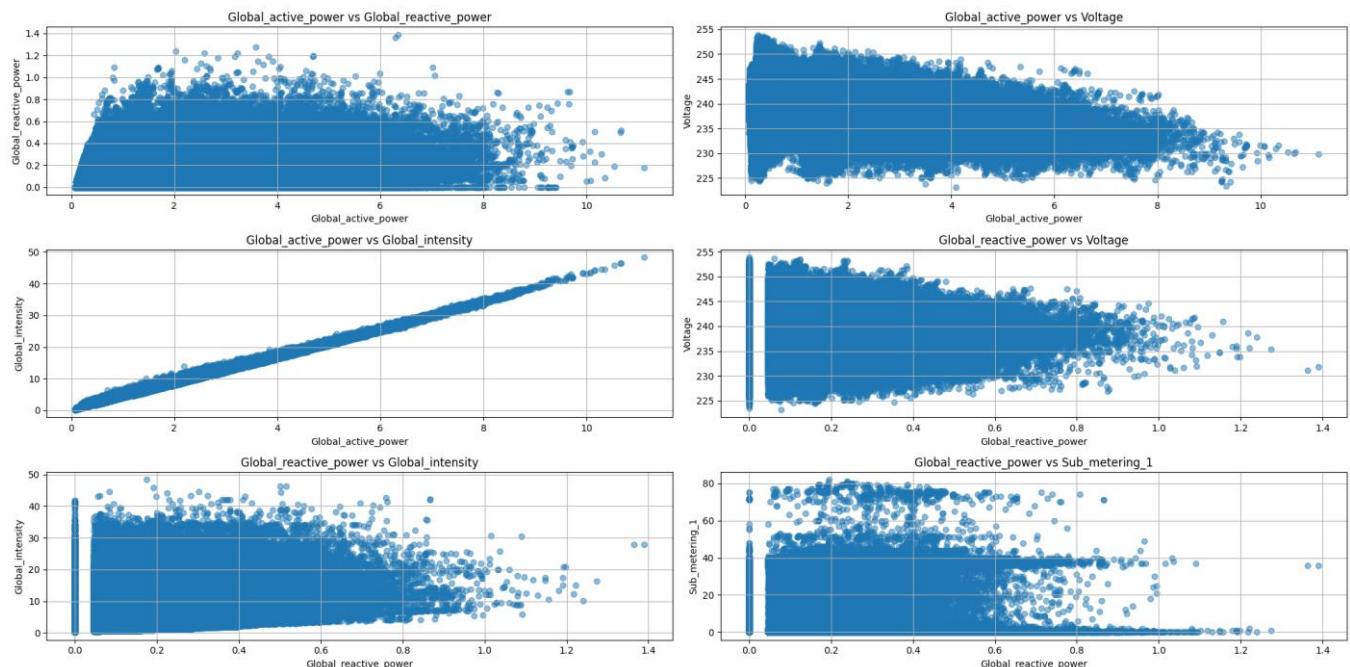
### Task – 6: Create multiple other plots.

#### Scatter Plot

```
pairs = [
    ('Global_active_power', 'Global_reactive_power'),
    ('Global_active_power', 'Voltage'),
    ('Global_active_power', 'Global_intensity'),
    ('Global_reactive_power', 'Voltage'),
    ('Global_reactive_power', 'Global_intensity'),
    ('Global_reactive_power', 'Sub_metering_1'),
]
```

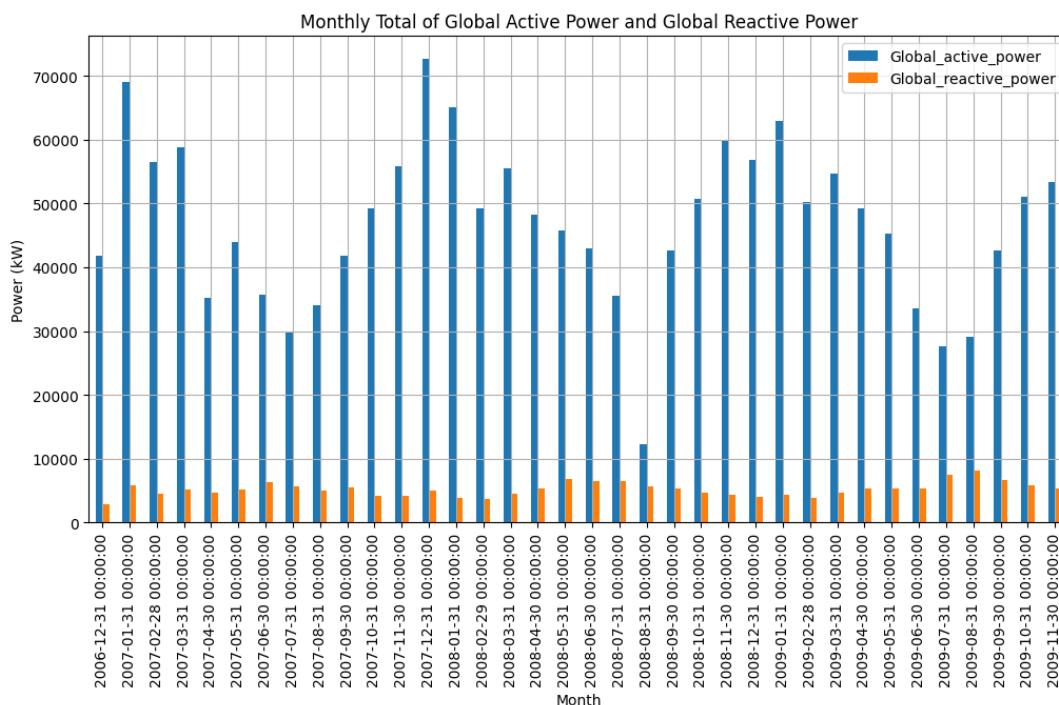
```
nrows, ncols = 3, 2
fig, axes = plt.subplots(nrows, ncols, figsize=(20, 10))
axes = axes.flatten()
for ax, (x_col, y_col) in zip(axes, pairs):
```

```
    ax.scatter(newdf[x_col], newdf[y_col], marker='o', alpha=0.5)
    ax.set_title(f'{x_col} vs {y_col}')
    ax.set_xlabel(x_col)
    ax.set_ylabel(y_col)
    ax.grid(True)
plt.tight_layout()
plt.show()
```



## Bar Chart

```
monthly_data = newdf.resample('ME').sum()
monthly_data[['Global_active_power', 'Global_reactive_power']].plot(kind='bar', figsize=(12, 6))
plt.title('Monthly Total of Global Active Power and Global Reactive Power')
plt.xlabel('Month')
plt.ylabel('Power (kW)')
plt.grid(True)
plt.show()
```



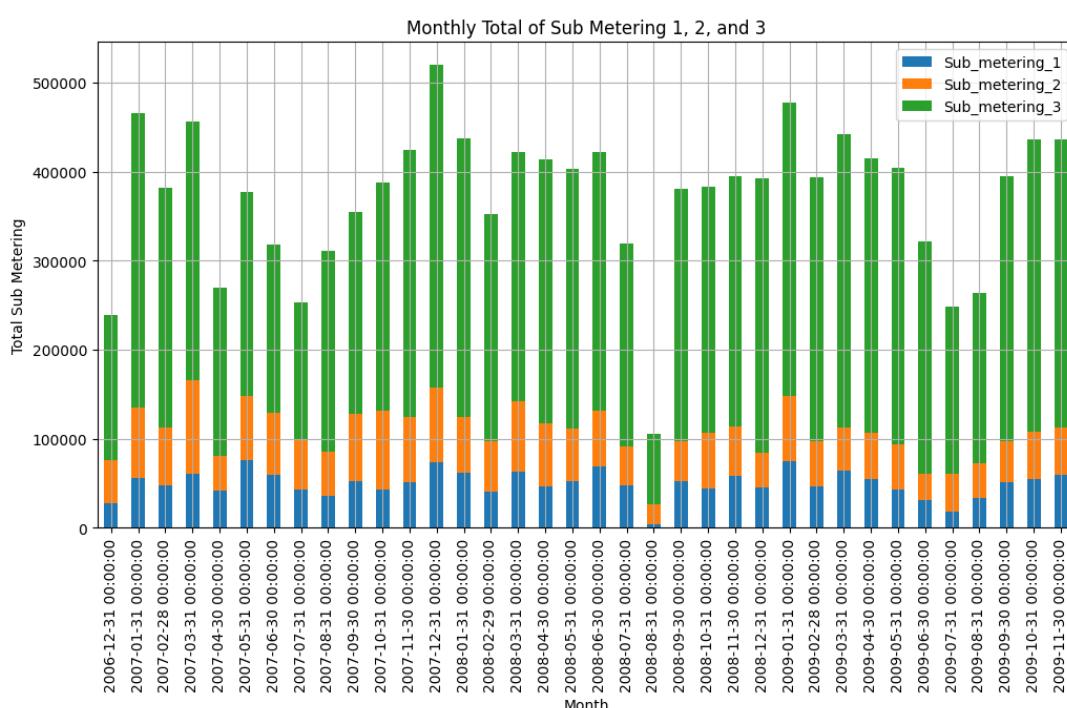
```

monthly_sub_metering = newdf[['Sub_metering_1',
                             'Sub_metering_3']].resample('ME').sum()

monthly_sub_metering.plot(kind='bar', stacked=True, figsize=(12, 6))

plt.title('Monthly Total of Sub Metering 1, 2, and 3')
plt.xlabel('Month')
plt.ylabel('Total Sub Metering')
plt.grid(True)
plt.show()

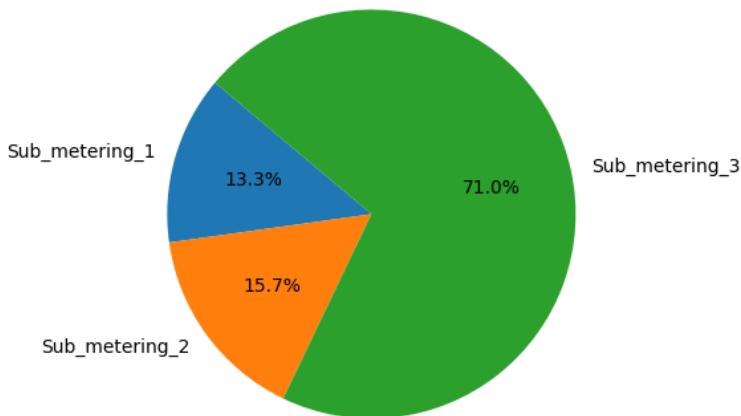
```



## Pie Chart

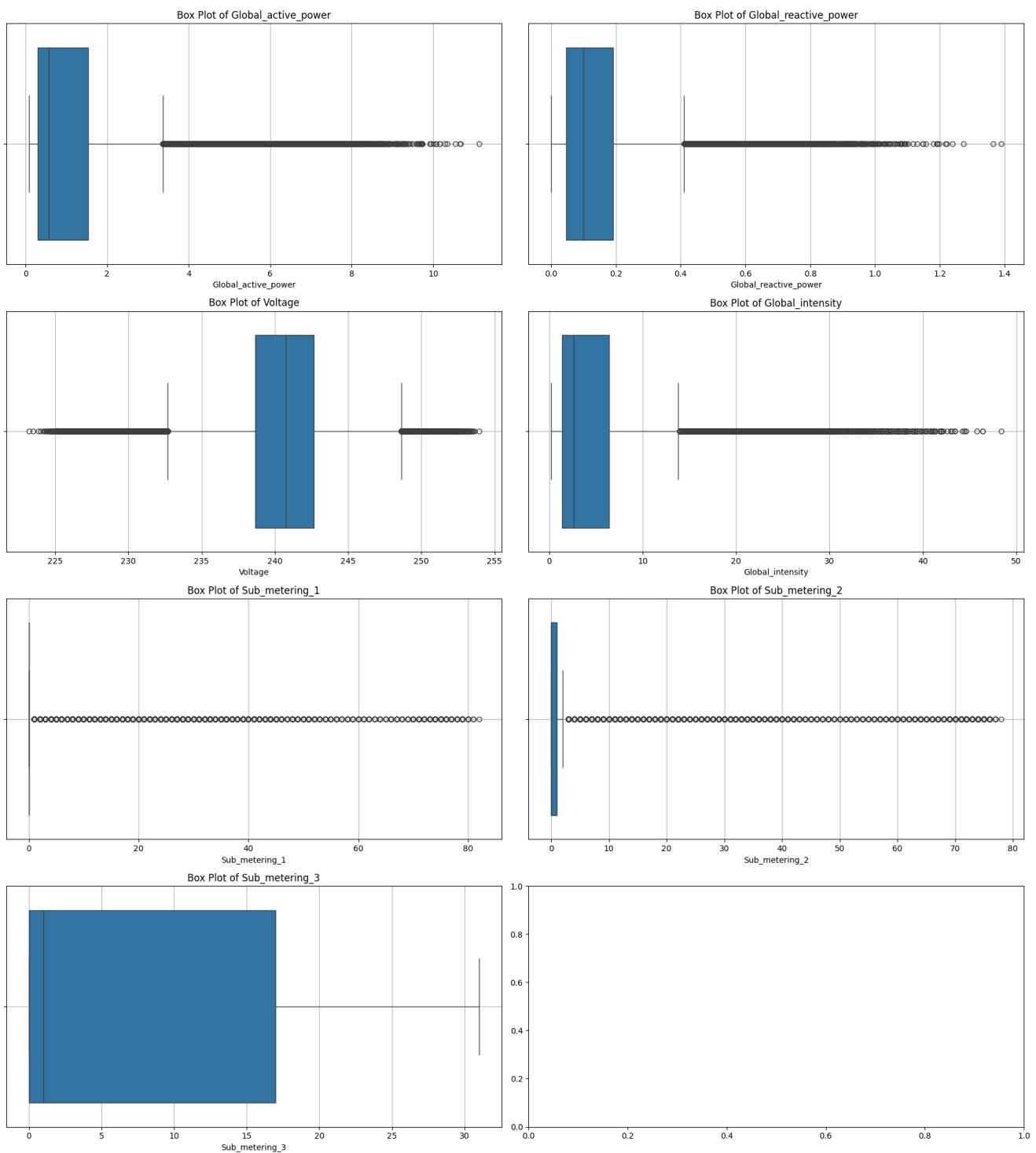
```
total_sub_metering = monthly_sub_metering.sum()
plt.figure(figsize=(5, 5))
plt.pie(total_sub_metering, labels=total_sub_metering.index, autopct='%.1f%%', startangle=140)
plt.title('Total Contribution of Sub Metering Over Entire Period')
plt.show()
```

Total Contribution of Sub Metering Over Entire Period



## Box Plot

```
columns_to_plot = ['Global_active_power', 'Global_reactive_power', 'Voltage',
                   'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(18, 20))
axes = axes.flatten()
for i, column in enumerate(columns_to_plot):
    sns.boxplot(x=newdf[column], ax=axes[i])
    axes[i].set_title(f'Box Plot of {column}')
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```



## Count Plot

```

hourly_data = newdf[['Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']].resample('h').sum()
hourly_data['Hour'] = hourly_data.index.hour
hourly_data_melted = hourly_data.melt(id_vars='Hour', var_name='Sub_metering', value_name='Total')

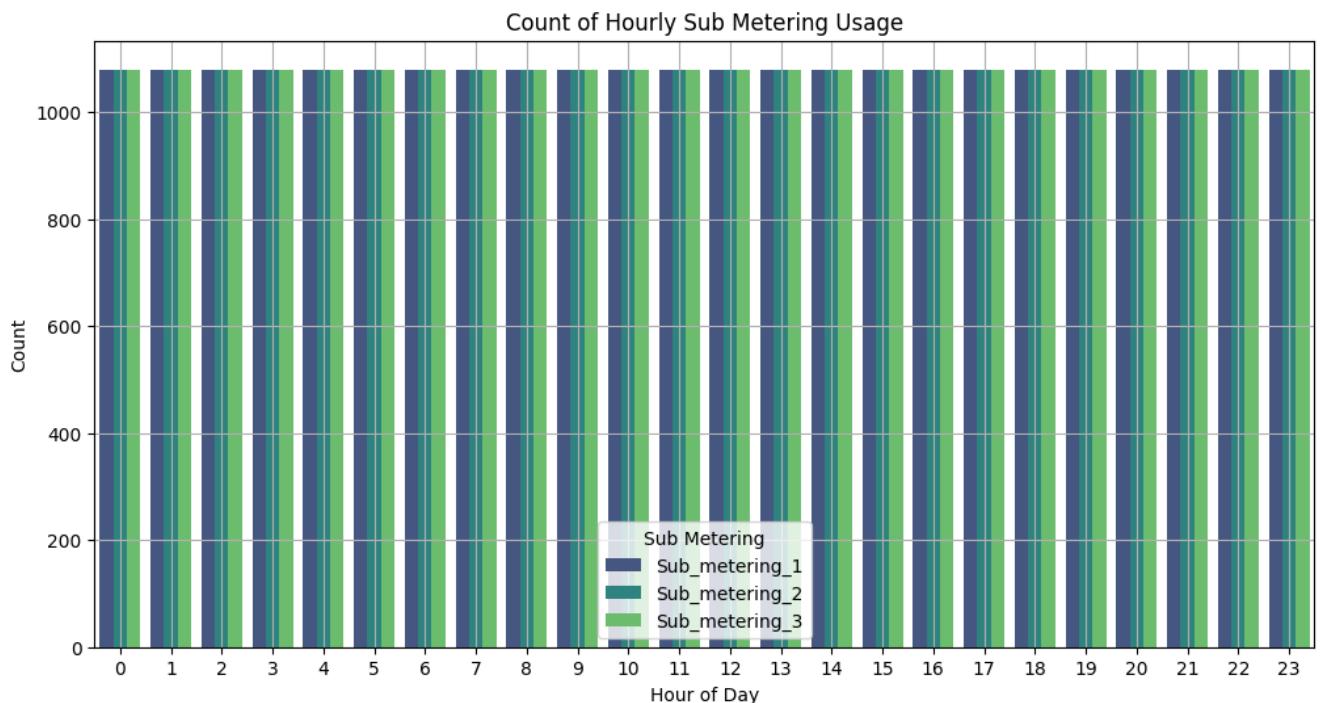
plt.figure(figsize=(12, 6))
sns.countplot(x='Hour', data=hourly_data_melted, hue='Sub_metering', palette='viridis')

```

```

plt.title('Count of Hourly Sub Metering Usage')
plt.xlabel('Hour of Day')
plt.ylabel('Count')
plt.legend(title='Sub Metering')
plt.grid(True)
plt.show()

```



## Distplot

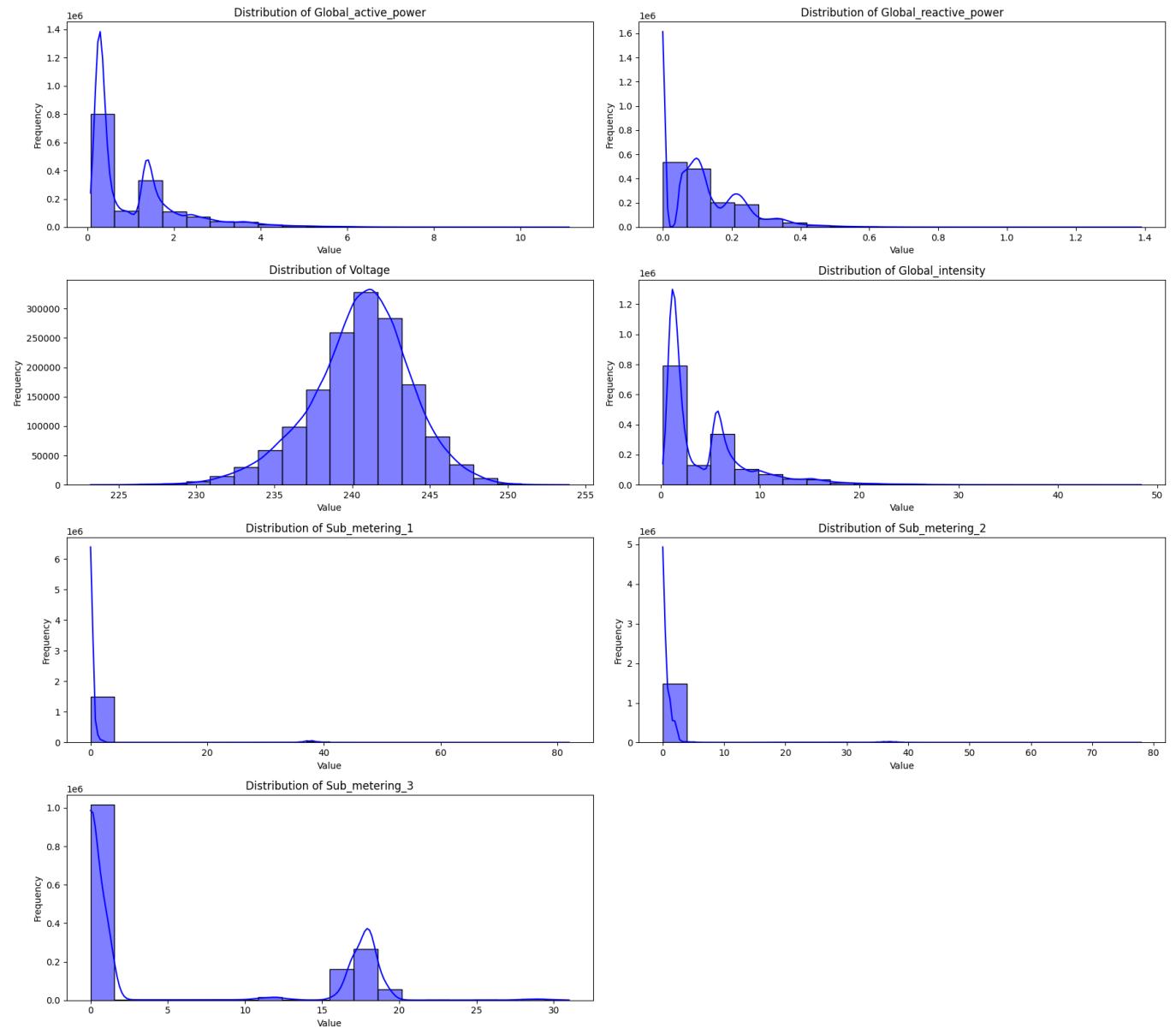
```

columns_to_plot = ['Global_active_power', 'Global_reactive_power', 'Voltage',
                   'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']

plt.figure(figsize=(18, 16))

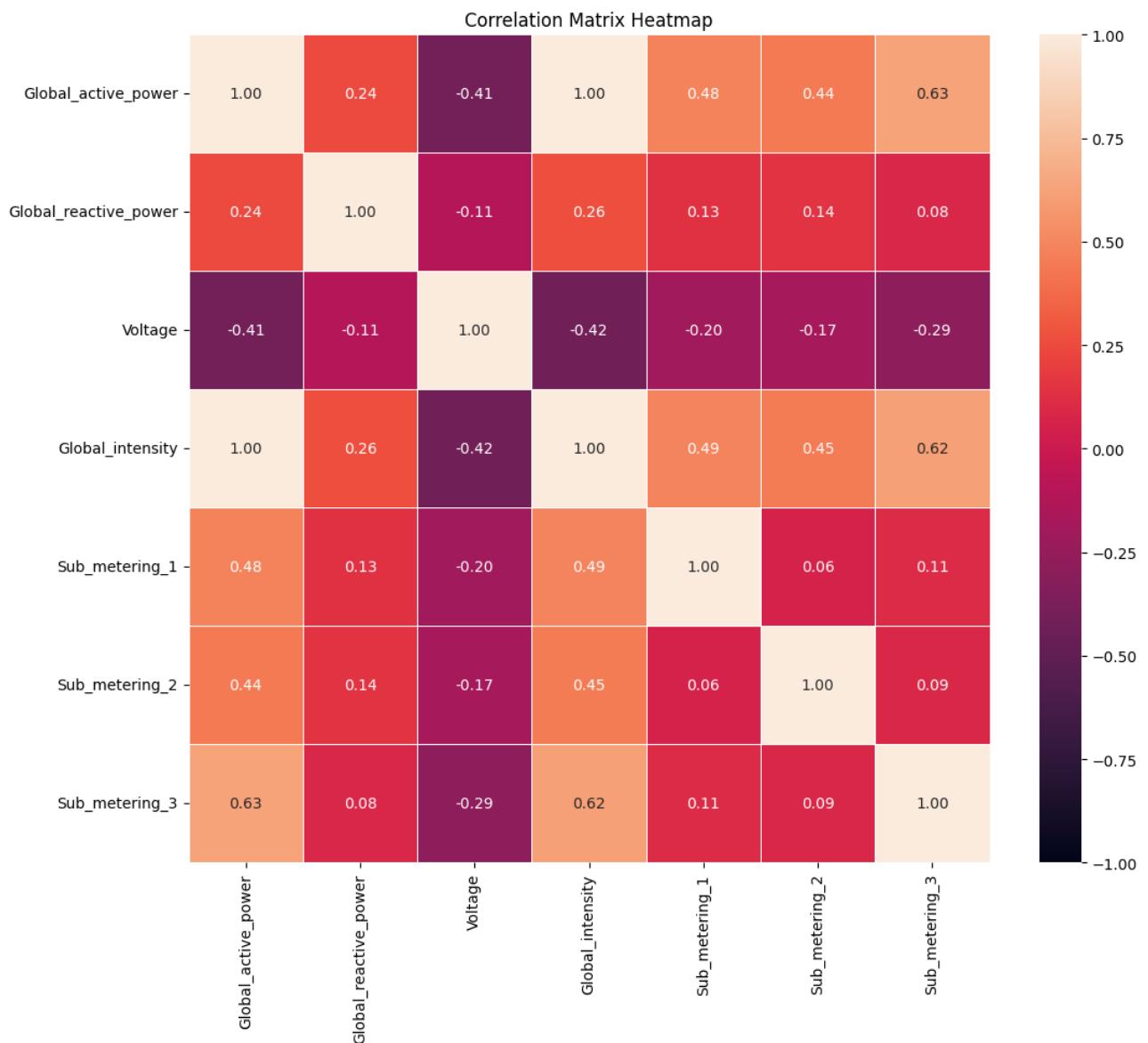
for i, column in enumerate(columns_to_plot):
    plt.subplot(4, 2, i + 1)
    sns.histplot(newdf[column], kde=True, color='blue', bins=20)
    plt.title(f'Distribution of {column}')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.tight_layout()
plt.show()

```



## Heatmap

```
correlation_matrix = newdf.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='%.2f', linewidths=0.5, vmin=-1, vmax=1)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



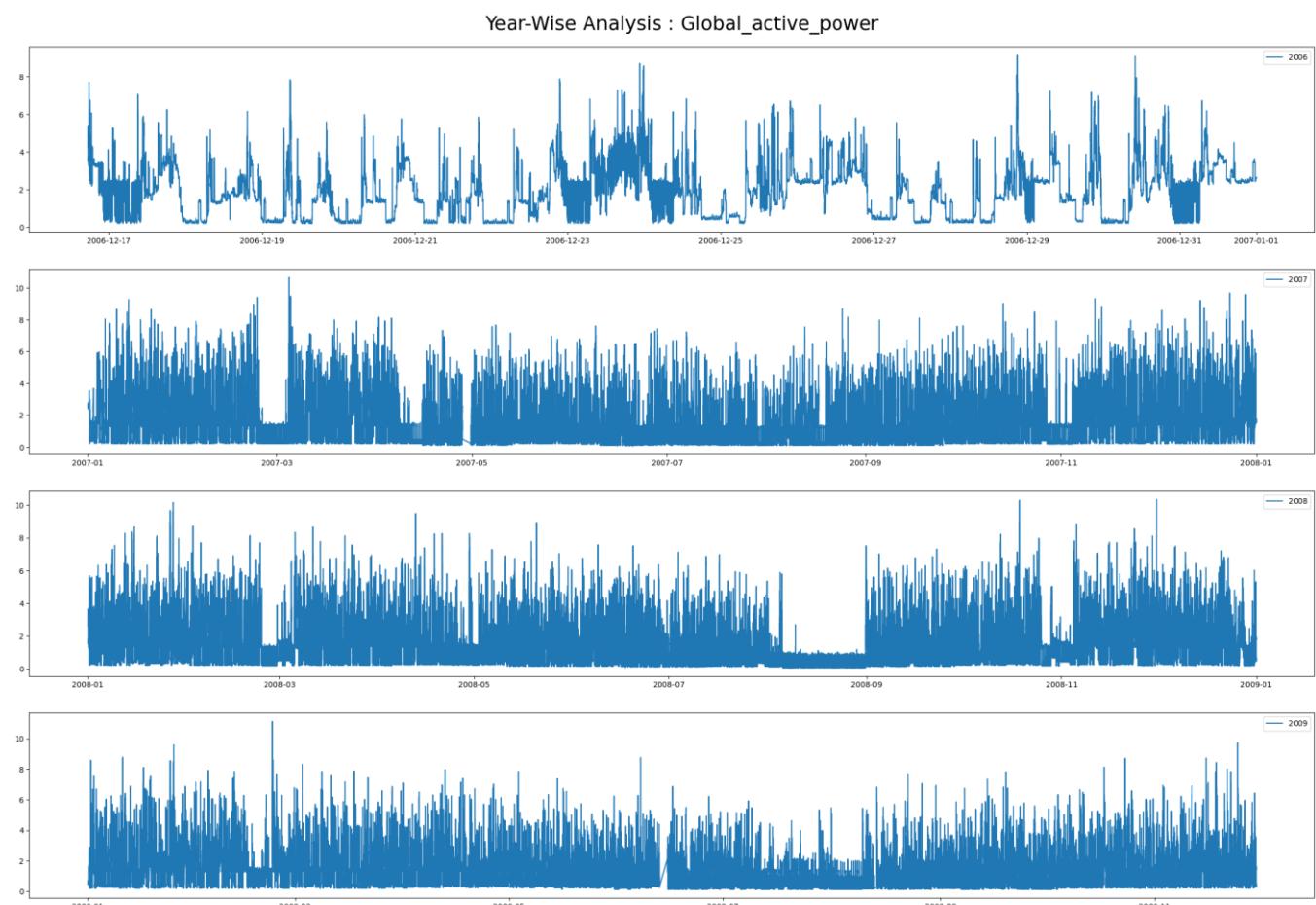
## Visualise Each Parameter Early

```
def visualize_yearly(data, feat_name):
    fig, axis = plt.subplots(4, 1, figsize=(30, 20))
    for i, d in enumerate(zip(axis, list(data[feat_name].groupby(data.index.year)))):
        d[0].plot(pd.DataFrame(d[1][1]), label=d[1][0])
        d[0].legend(loc='upper right')
    fig.text(0.40, 0.9, 'Year-Wise Analysis : %s ' % feat_name, va='center', fontdict={'fontsize': 25})
    plt.show()
```

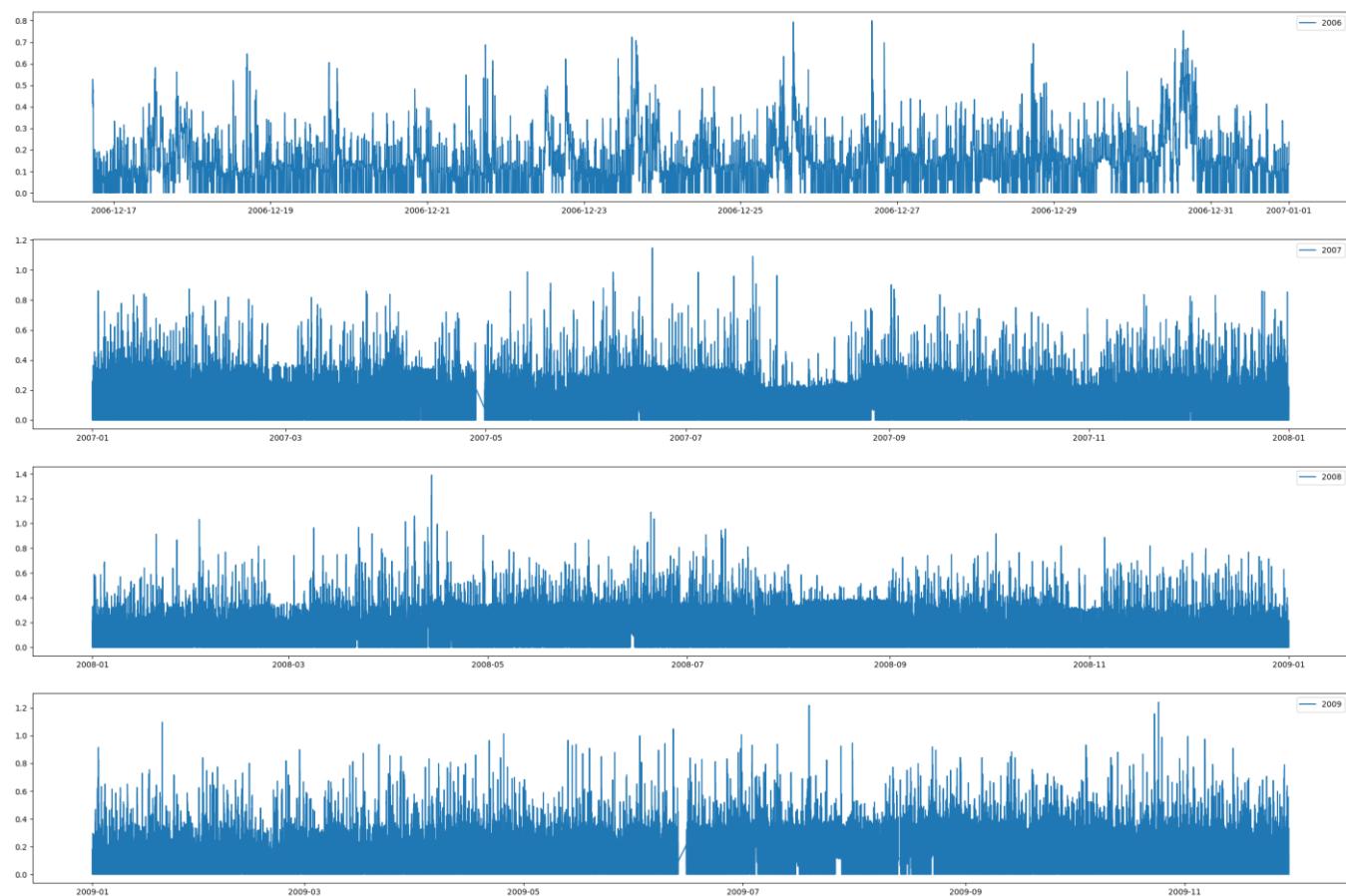
```
visualize_yearly(data=newdf, feat_name='Global_active_power')
```

```
visualize_yearly(data=newdf, feat_name='Global_reactive_power')
```

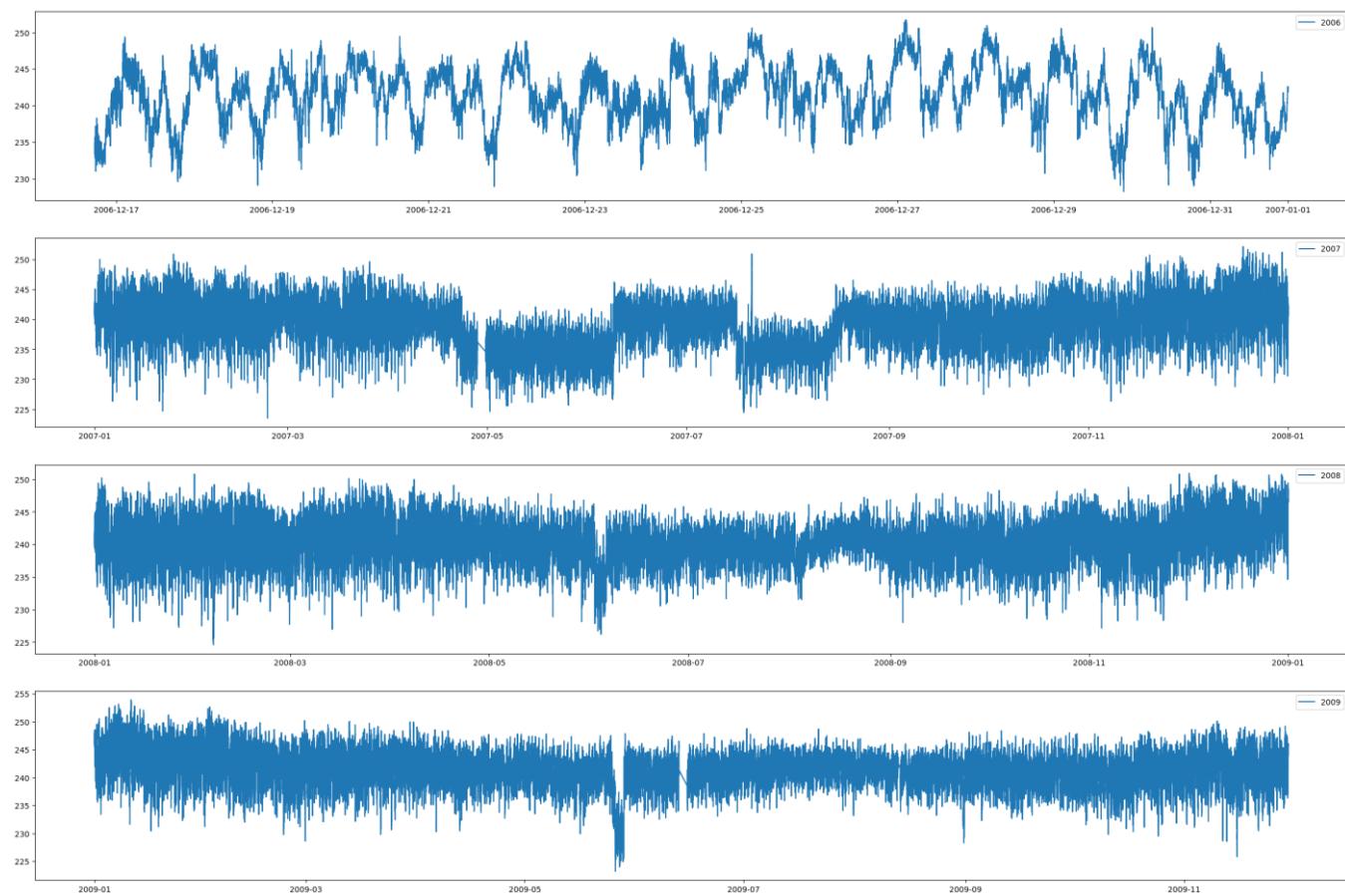
```
visualize_yearly(data=newdf,feat_name='Voltage')
visualize_yearly(data=newdf,feat_name='Global_intensity')
visualize_yearly(data=newdf,feat_name='Sub_metering_1')
visualize_yearly(data=newdf,feat_name='Sub_metering_2')
visualize_yearly(data=newdf,feat_name='Sub_metering_3')
```



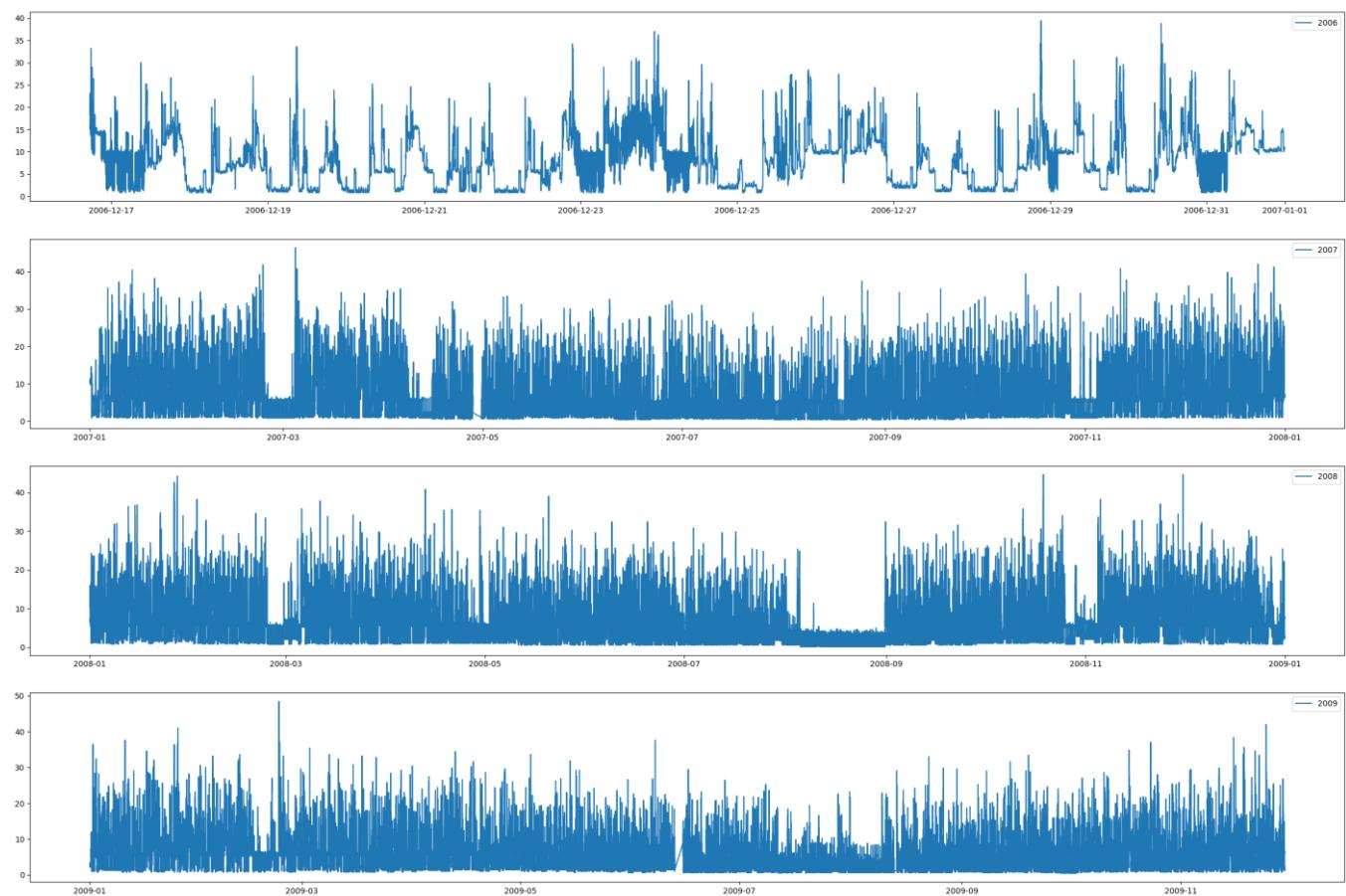
## Year-Wise Analysis : Global\_reactive\_power



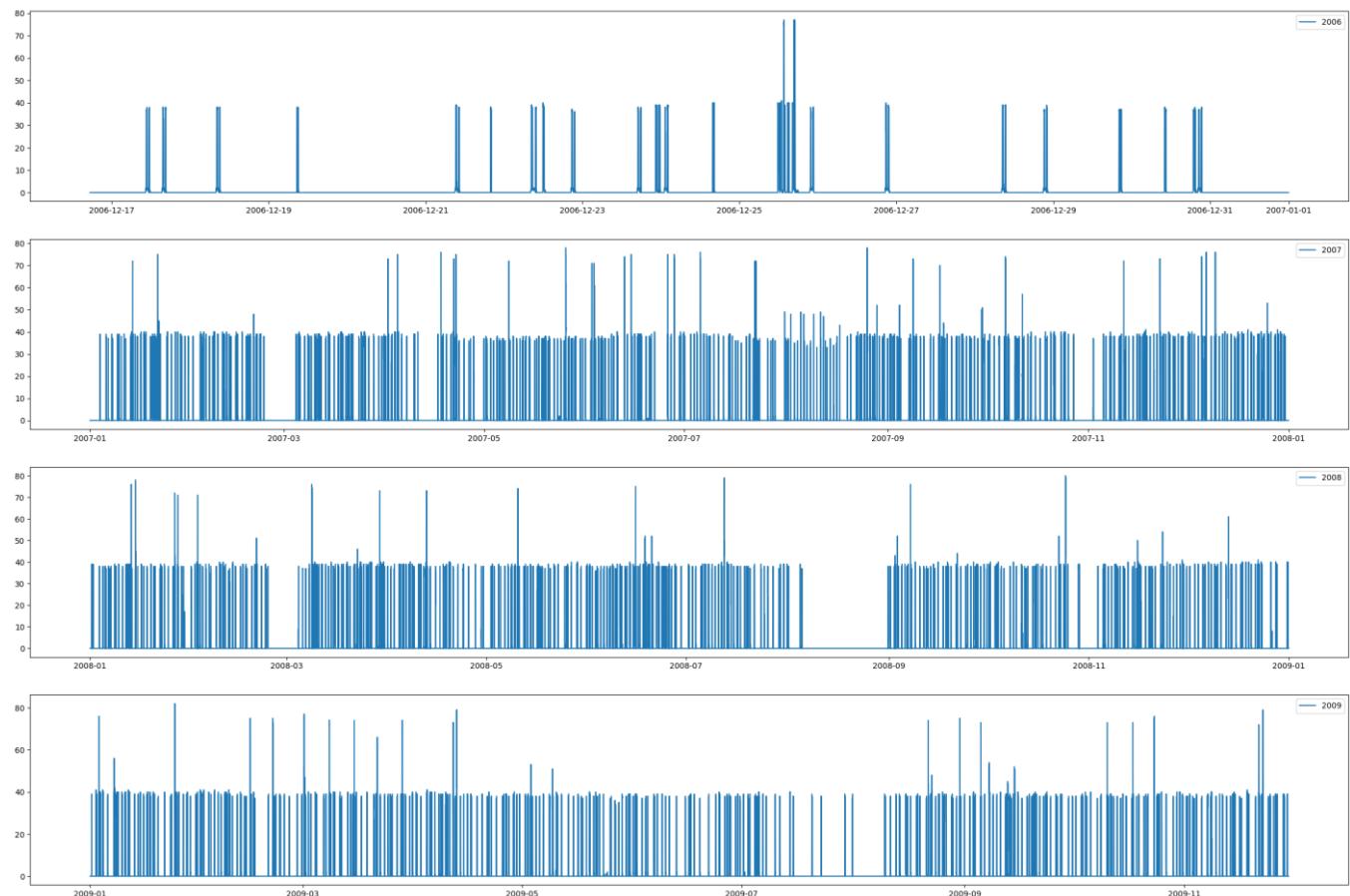
## Year-Wise Analysis : Voltage



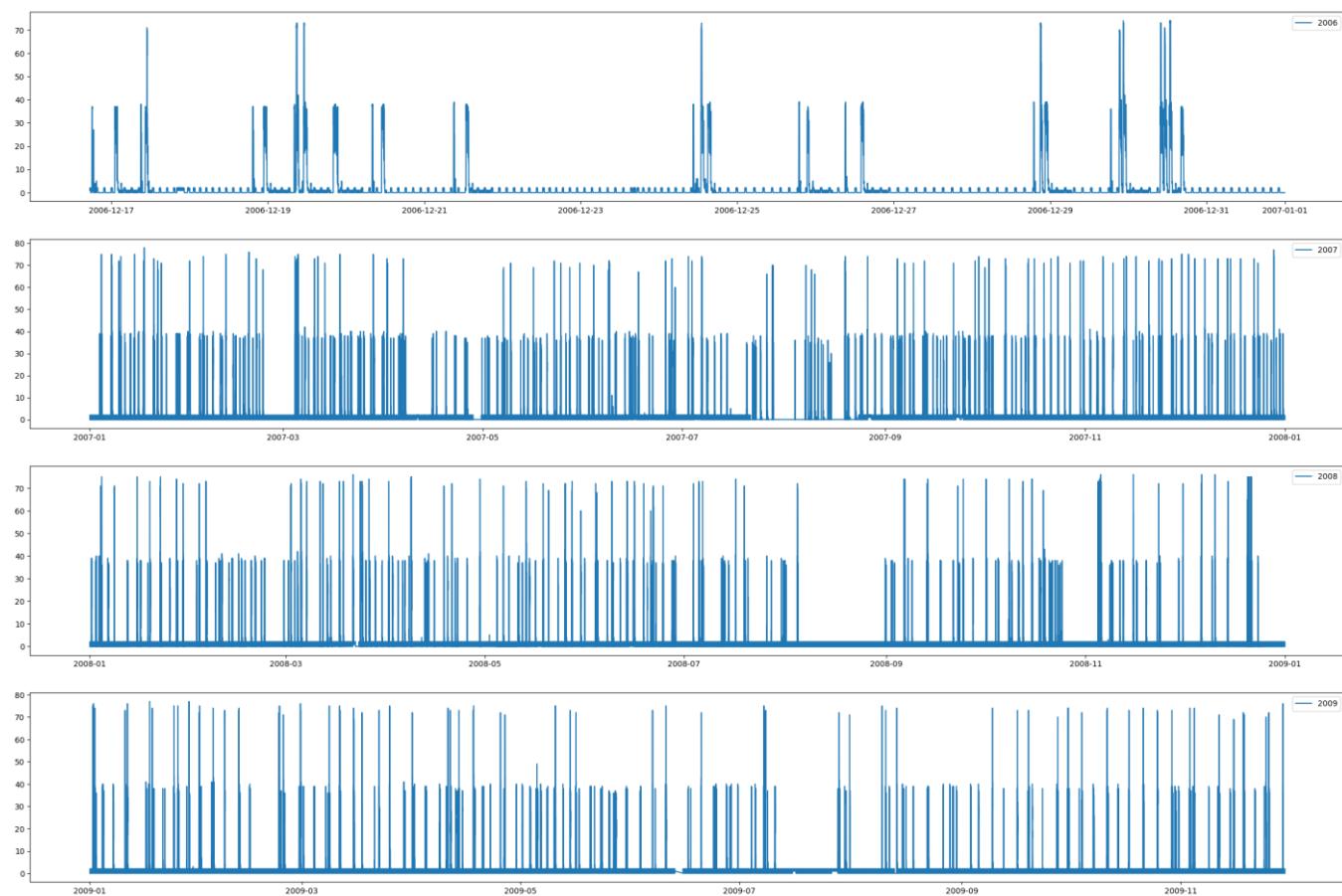
## Year-Wise Analysis : Global\_intensity



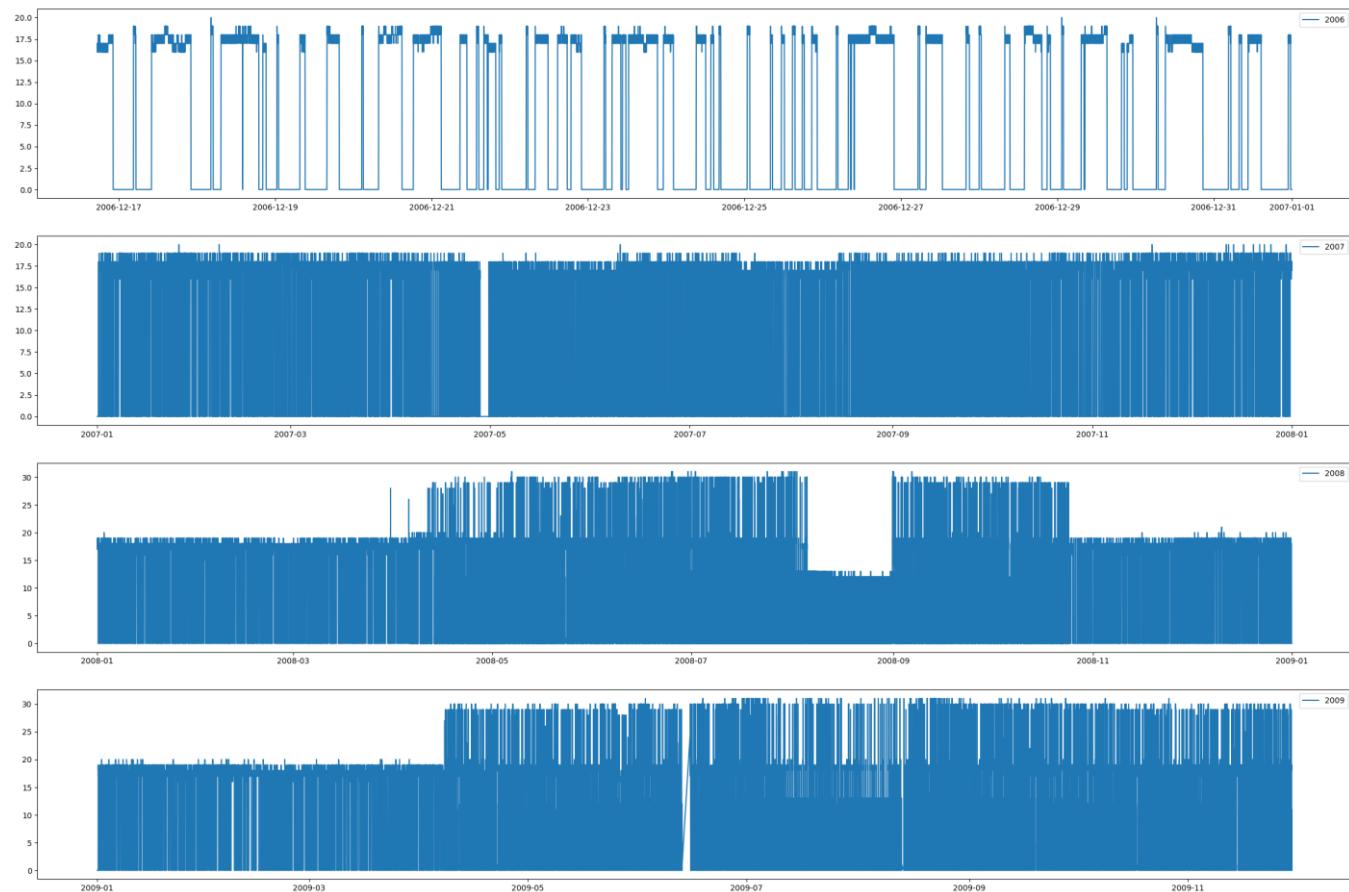
## Year-Wise Analysis : Sub\_metering\_1



## Year-Wise Analysis : Sub\_metering\_2



## Year-Wise Analysis : Sub\_metering\_3



## Use of each type of plot

- **Histogram:** Ideal for understanding the distribution of power consumption levels.
- **Time Series Plot:** Best for analysing trends and patterns over the 4-year period.
- **Plot for Sub Metering:** Key for comparing the contributions of different sub-meters over time.
- **Scatterplot:** Useful for examining relationships between two continuous variables.
- **Bar Chart:** Great for comparing quantities across different time periods.
- **Pie Chart:** Shows proportions of sub-metering contributions to total consumption.
- **Count Plot:** Displays the frequency of different power consumption categories.
- **Boxplot:** Summarizes distributions and highlights outliers in consumption data.
- **Heatmap:** Effective for identifying correlations and patterns across variables and time.
- **Distplot:** Combines a histogram and KDE to show a detailed distribution.

## LAB ASSIGNMENT 3

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Implement simple and multi-linear regression to predict profits for a food truck. Compare the performance of the model on linear and multi-linear regression.		

### Objective

The objective of this lab assignment is to implement simple and multi-linear regression models to predict profits for a food truck business. By comparing the performance of these two regression models, you will gain insights into when and how to use simple and multi-linear regression techniques.

### Steps

#### 1. Data Preparation:

- Construct a numpy array containing both features (Population, Years in Business) and the target variable (Profit).
- Separate the data into feature set XXX and target variable yyy.
- Address any missing values in yyy by substituting NaN with the average of yyy.

#### 2. Simple Linear Regression:

- Develop a function called simple\_linear\_regression to compute the coefficients ( $\beta_0 \backslash \beta_1 \backslash \beta_2$ ), make predictions, and calculate the mean squared error (MSE) for a single feature (Years in Business).
- Extract the Years in Business as X\_simple and apply the simple linear regression model.

#### 3. Multiple Linear Regression:

- Create a function named multi\_linear\_regression to determine coefficients, predictions, and MSE using all features.
- Prepare the feature set X\_multi by adding a column of ones for the intercept term and then perform the multiple linear regression analysis.

#### 4. Plotting:

- Generate two subplots:
  - The first subplot will display the results of the simple linear regression (Years in Business vs. Profit).
  - The second subplot will illustrate the results of the multiple linear regression (Population vs. Profit).
- Show both actual and predicted values in these plots.

#### 5. Print Results:

- Print out the coefficients, intercepts, and MSE for both the simple and multiple linear regression models.

## Code

```

import numpy as np
import matplotlib.pyplot as plt

data = np.array(
    [
        [10000, 5, 10000],
        [15000, 6, 12000],
        [20000, 6, 13000],
        [9000, 5, 12000],
        [12000, 4, np.nan],
    ]
)
X = data[:, :-1]
y = data[:, -1]

mean_y = np.nanmean(y)
y[np.isnan(y)] = mean_y

def simple_linear_regression(X_simple, y):
    X_simple_mean = np.mean(X_simple)
    y_mean = np.mean(y)
    beta1 = np.sum((X_simple - X_simple_mean) * (y - y_mean)) / np.sum(
        (X_simple - X_simple_mean) ** 2
    )
    beta0 = y_mean - beta1 * X_simple_mean
    y_pred = beta0 + beta1 * X_simple
    mse = np.mean((y - y_pred) ** 2)
    return beta0, beta1, y_pred, mse

def multi_linear_regression(X_multi, y):
    XTX = np.dot(X_multi.T, X_multi)
    XTX_inv = np.linalg.inv(XTX)
    XTy = np.dot(X_multi.T, y)
    beta = np.dot(XTX_inv, XTy)
    y_pred = np.dot(X_multi, beta)
    mse = np.mean((y - y_pred) ** 2)
    return beta, y_pred, mse

X_simple = X[:, 1]
beta0, beta1, y_pred_simple, mse_simple = simple_linear_regression(X_simple, y)
X_multi = np.hstack([np.ones((X.shape[0], 1)), X])
beta_multi, y_pred_multi, mse_multi = multi_linear_regression(X_multi, y)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(X_simple, y, color="blue", label="Actual Profit")
plt.plot(

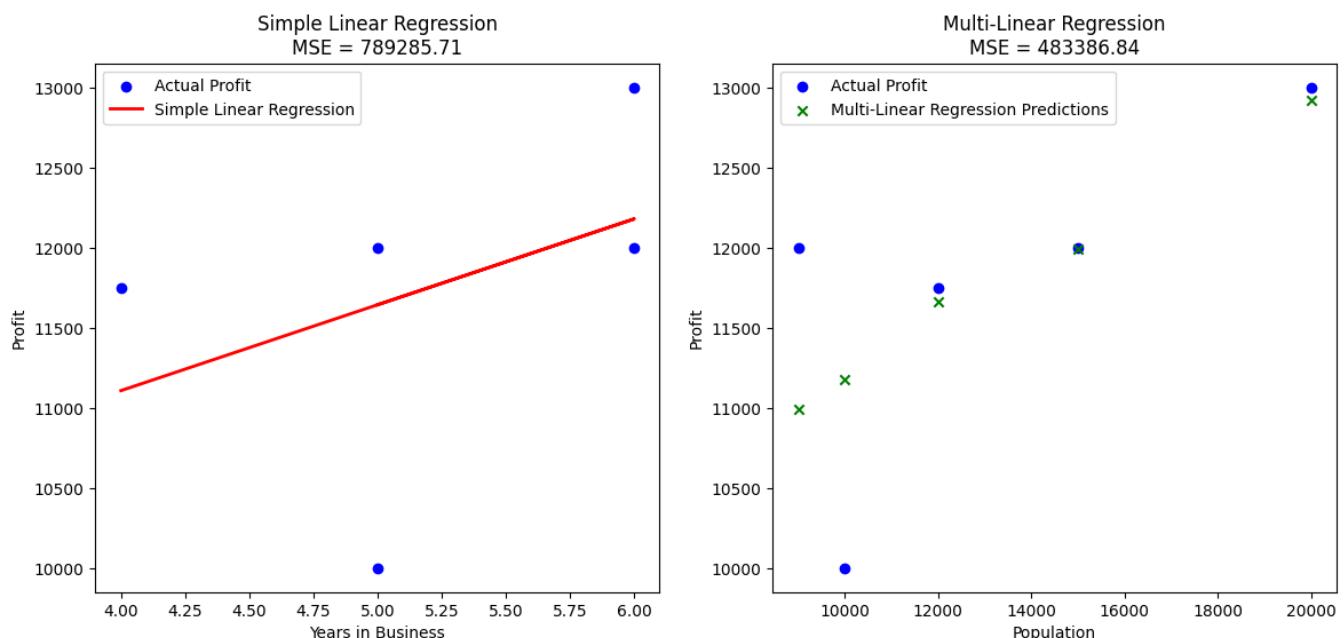
```

```

    X_simple, y_pred_simple, color="red", linewidth=2, label="Simple Linear Regression"
)
plt.xlabel("Years in Business")
plt.ylabel("Profit")
plt.title(f"Simple Linear Regression\nMSE = {mse_simple:.2f}")
plt.legend()

plt.subplot(1, 2, 2)
X_population = X[:, 0]
plt.scatter(X_population, y, color="blue", label="Actual Profit")
plt.scatter(
    X_population,
    y_pred_multi,
    color="green",
    marker="x",
    label="Multi-Linear Regression Predictions",
)
plt.xlabel("Population")
plt.ylabel("Profit")
plt.title(f"Multi-Linear Regression\nMSE = {mse_multi:.2f}")
plt.legend()
plt.show()

```



```

X = np.array([[10000], [15000], [20000], [9000]])
y = np.array([10000, 12000, 13000, 12000])

```

```

Population_mean = X.mean()
Population_std = X.std()
Population_normalized = (X - Population_mean) / Population_std

```

```

X = np.hstack((np.ones((X.shape[0], 1)), Population_normalized))
y = y

```

```
w = np.zeros(X.shape[1])

learning_rate = 0.01
n_iterations = 1000

def predict(X, w):
    return X.dot(w)

def compute_cost(X, y, w):
    m = len(y)
    predictions = predict(X, w)
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    return cost

def gradient_descent(X, y, w, learning_rate, n_iterations):
    m = len(y)
    cost_history = []

    for i in range(n_iterations):
        predictions = predict(X, w)
        errors = predictions - y
        gradients = (1 / m) * X.T.dot(errors)
        w -= learning_rate * gradients

        cost = compute_cost(X, y, w)
        cost_history.append(cost)

    return w, cost_history

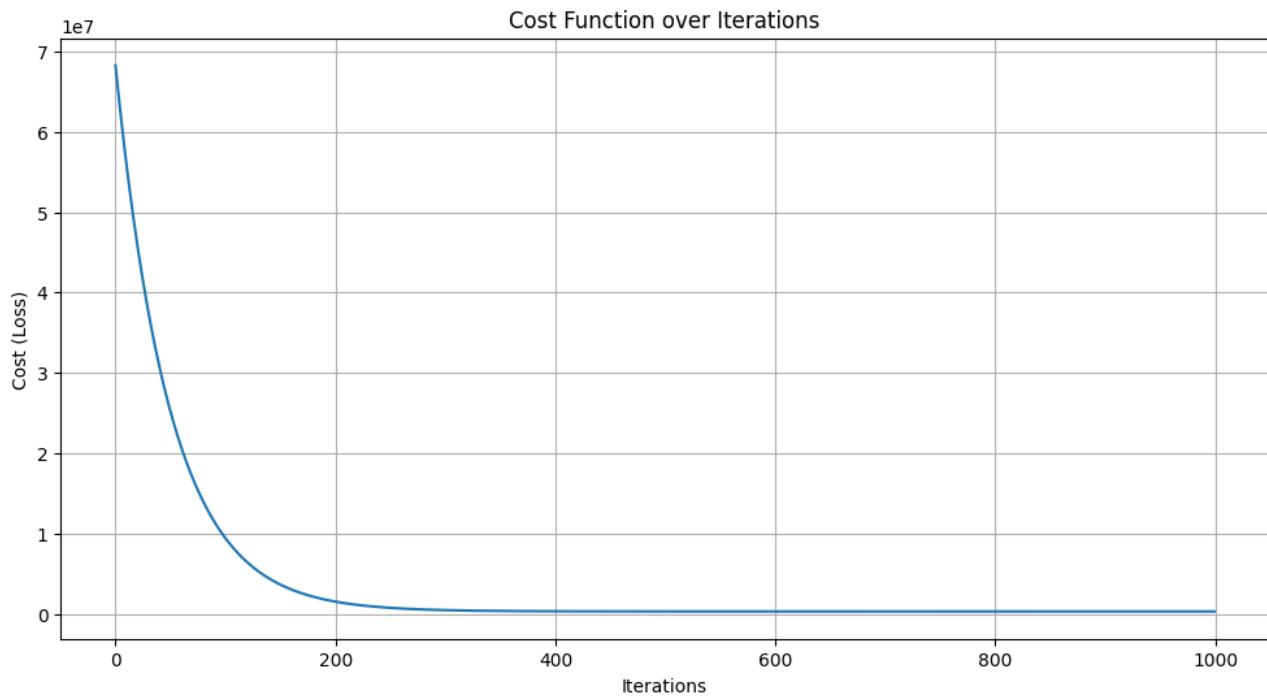
w, cost_history = gradient_descent(X, y, w, learning_rate, n_iterations)

print("Final weights: {}")  
plt.figure(figsize=(12, 6))

plt.plot(range(len(cost_history)), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost (Loss)")
plt.title("Cost Function over Iterations")
plt.grid(True)
plt.show()
```

## Output

Final weights: [11749.49273784 769.20068232]



## LAB ASSIGNMENT 4

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs		

### Objective

To fit data points by assigning different weights to each point based on its proximity to the query point.

**Dataset:** Synthetic sinusoidal patterned data (Generated using *numpy* library)

### Code

```
import numpy as np
import matplotlib.pyplot as plt
```

#### *# Generate Dataset and splitting into features and target vars*

We generate 80 random data points X uniformly distributed between 0 and 5. The target variable y is generated by taking the sine of X and adding some random noise to simulate real-world data.

```
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])
```

#### *# Normal Linear Regression*

```
def normal_equation_linear_regression(X, y):
    m = X.shape[0]

    X_augmented = np.hstack([np.ones((m, 1)), X])

    XTX = np.dot(X_augmented.T, X_augmented)
    XTX_inv = np.linalg.pinv(XTX)
    theta = np.dot(XTX_inv, np.dot(X_augmented.T, y))
    return theta

theta = normal_equation_linear_regression(X, y)
```

```

def predict(X, theta):
    m = X.shape[0]
    X_augmented = np.hstack([np.ones((m, 1)), X])
    y_pred = np.dot(X_augmented, theta)
    return y_pred

y_pred = predict(X, theta)

plt.figure(figsize=(12, 8))
plt.scatter(X, y, label="Training Data", color="blue")
plt.plot(X, y_pred, color="red", linewidth=2, label="Linear Regression Fit")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Normal Linear Regression")
plt.legend()
plt.show()

print("Intercept: {theta[0]}")
print("Coefficient: {theta[1]}")

```

### **# Implementation of Locally Weighted Regression algorithm**

$$w^{(i)} = \exp\left(\frac{-(x^{(i)} - x)^2}{2\tau^2}\right) \quad J(\theta) = \frac{1}{2} \sum_{i=1}^m w^{(i)} (\theta^T x^{(i)} - y^{(i)})^2$$

```

def lwlr(X, y, x_query, tau):
    m = X.shape[0]
    W = np.exp(-np.sum((X - x_query) ** 2, axis=1) / (2 * tau**2))
    W = np.diag(W)

    X_augmented = np.hstack([np.ones((m, 1)), X])
    x_query_augmented = np.array([1, x_query.item()]).reshape(1, 2)

```

```

XTWX = np.dot(np.dot(X_augmented.T, W), X_augmented)
XTWy = np.dot(np.dot(X_augmented.T, W), y)
theta = np.linalg.solve(XTWX, XTWy)
y_query = np.dot(x_query_augmented, theta)
return y_query

```

```

def predict_lwlr(X, y, X_query, tau):
    y_pred = np.array([lwlr(X, y, x_query, tau) for x_query in X_query])
    return y_pred

```

### **# Using Multiple Query Points**

```

X_query = np.linspace(0, 5, 100).reshape(-1, 1)
taus = [0.1, 0.3, 0.8, 2.0]

```

```

predictions = {}
for tau in taus:
    predictions[tau] = predict_lwlr(X, y, X_query, tau)

```

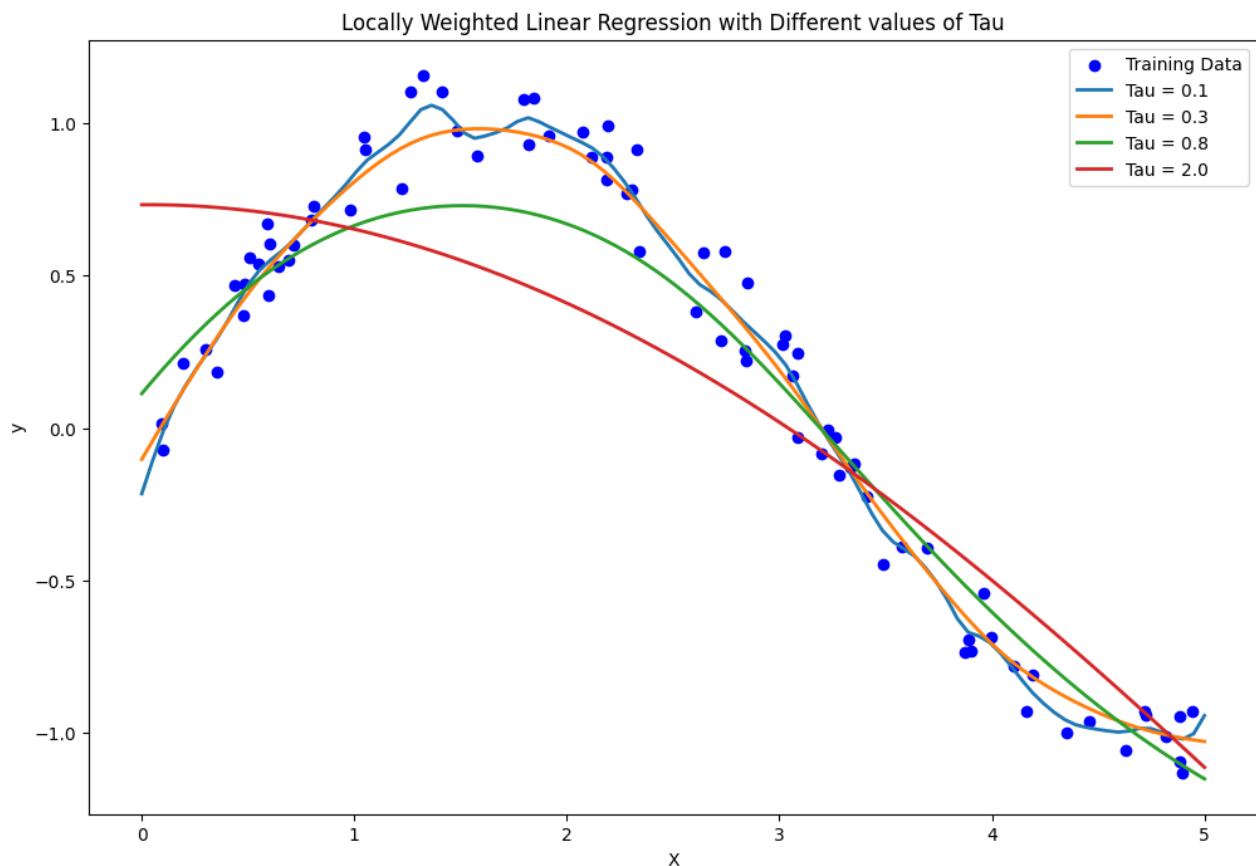
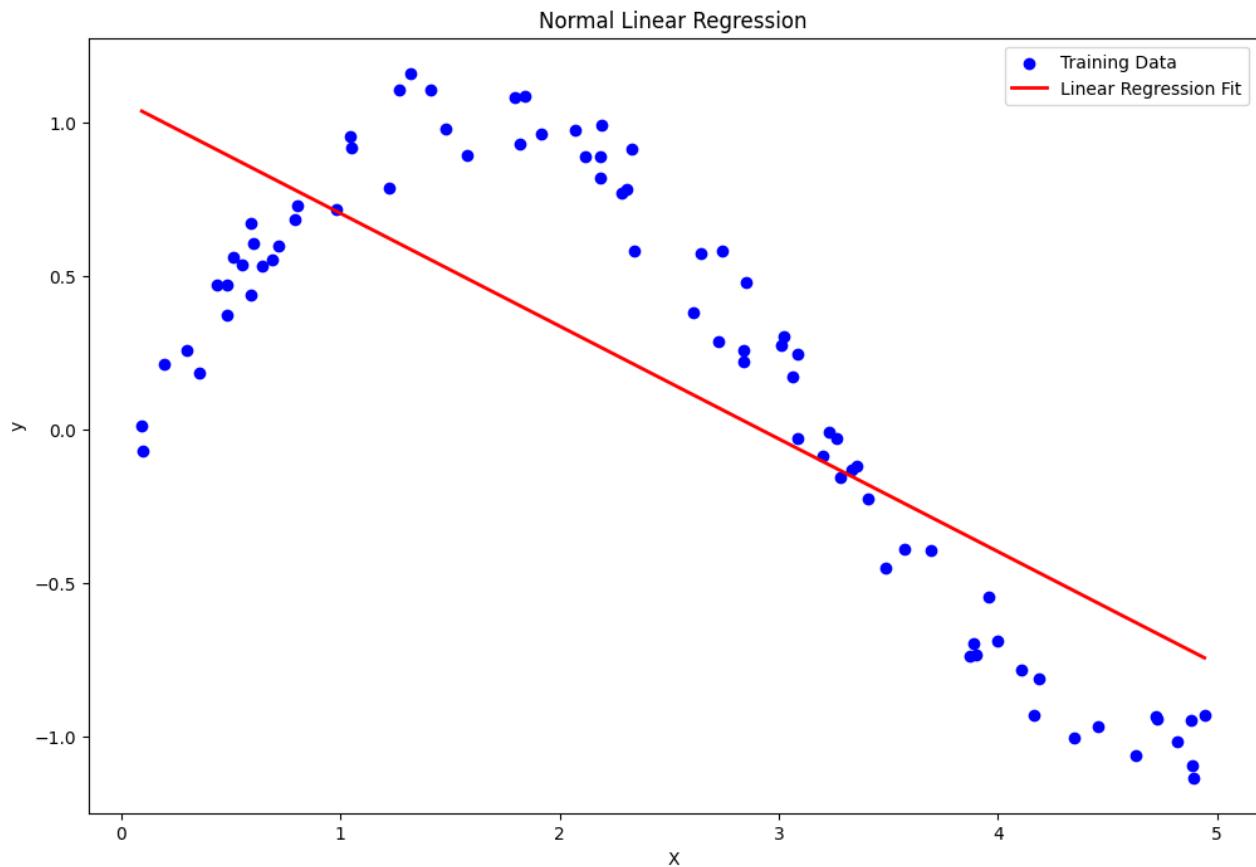
### **# Plotting the Graph**

```

plt.figure(figsize=(12, 8))
plt.scatter(X, y, label="Training Data", color="blue")
for tau in taus:
    plt.plot(X_query, predictions[tau], linewidth=2, label=f'Tau = {tau}')
plt.xlabel("X")
plt.ylabel("y")
plt.title("Locally Weighted Linear Regression with Different values of Tau")
plt.legend()
plt.show()

```

## Output



## LAB ASSIGNMENT 5

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	For a given set of training data examples stored in a .CSV file, implement and demonstrate various feature selection algorithms and compare the performance of the algorithms.		

### Objective

The objective of this lab assignment is to implement and demonstrate various feature selection algorithms on a given training dataset stored in a .CSV file. The goal is to evaluate the effectiveness of these algorithms in terms of improving model accuracy and reducing dimensionality.

### Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
columns = ["Pregnancies", "Glucose", "BloodPressure", "Skin Thickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
dataset = pd.read_csv('./master_pima-indians-diabetes.csv')
dataset.head()
array = dataset.values
X = array[:, 0:8]
y = array[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

### Univariate Analysis

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)
np.set_printoptions(precision=3)
```

```
features = fit.transform(X)
print("Scores: ")
print(fit.scores_)
print("Features: ")
print(features[0:5, :])
```

```
Scores:
 [ 110.727 1406.59     17.505    51.008 2219.398   127.671      5.356  178.011]
```

```
Features:
 [[ 85.      0.     26.6   31. ]
 [183.      0.     23.3   32. ]
 [ 89.     94.     28.1   21. ]
 [137.    168.     43.1   33. ]
 [116.      0.     25.6   30. ]]
```

## Recursive Feature Elimination

```
from sklearn.feature_selection import RFE
array = dataset.values
X = array[:, 0:8]
y = array[:, 8]
model = LogisticRegression()
rfe = RFE(model, n_features_to_select=4)
fit = rfe.fit(X,y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
reduced_dataset = dataset.iloc[:, :8].loc[:, fit.support_]
reduced_dataset.head()
```

```
Num Features: 4
Selected Features: [ True  True False False  False  True  True False]
Feature Ranking: [1 1 3 5 4 1 1 2]
```

<b>6</b>	<b>148</b>	<b>33.6</b>	<b>0.627</b>
0	1	85	26.6
1	8	183	23.3
2	1	89	28.1
3	0	137	43.1
4	5	116	25.6
			0.201

## Lasso - L1 Regularization

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
lasso = Lasso(alpha=0.05)
lasso.fit(X_train, y_train)
lasso_coefficients = lasso.coef_
selected_features = [feature for feature, coef in zip(columns, lasso_coefficients) if coef != 0]
print("Selected Features:", selected_features)

Selected Features: ['Pregnancies', 'GLucose', 'BloodPressure', 'Skin Thickness', 'BMI', 'Age']
```

## Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_importances = rf_model.feature_importances_
sorted_indices = rf_importances.argsort()[:-1]
selected_features_rf = np.array(columns)[sorted_indices]
print("Feature Importances (Random Forest):")
for feature, importance in zip(selected_features_rf, rf_importances[sorted_indices]):
    print(f"{feature}: {importance:.4f}")
```

```
Feature Importances (Random Forest):
GLucose: 0.2590
BMI: 0.1503
Age: 0.1357
DiabetesPedigreeFunction: 0.1280
Pregnancies: 0.0976
BloodPressure: 0.0871
Insulin: 0.0768
Skin Thickness: 0.0655
```

## XGBoost

```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')

xgb_model.fit(X_train, y_train)

xgb_importances = xgb_model.feature_importances_

sorted_indices = xgb_importances.argsort()[:-1]

selected_features_xgb = np.array(columns)[sorted_indices]

print("Feature Importances (XGBoost):")

for feature, importance in zip(selected_features_xgb, xgb_importances[sorted_indices]):

    print(f'{feature}: {importance:.4f}')
```

```
Feature Importances (XGBoost):
GLucose: 0.2428
Age: 0.1437
BMI: 0.1319
Pregnancies: 0.1155
Skin Thickness: 0.1063
Insulin: 0.1061
DiabetesPedigreeFunction: 0.0844
BloodPressure: 0.0693
```

## Comparing Random Forest with and without feature selection

```
import xgboost as xgb

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

from sklearn.ensemble import RandomForestClassifier


dataset = pd.read_csv('./master_pima-indians-diabetes.csv')

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 1: Train XGBoost on the original dataset

xgb_model_original = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
```

```
xgb_model_original.fit(X_train, y_train)
y_pred_original = xgb_model_original.predict(X_test)
accuracy_original = accuracy_score(y_test, y_pred_original)
f1_original = f1_score(y_test, y_pred_original)
precision_original = precision_score(y_test, y_pred_original)
recall_original = recall_score(y_test, y_pred_original)
```

*# Step 2: Feature selection using Random Forest*

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_importances = rf_model.feature_importances_
sorted_indices = rf_importances.argsort()[:-1]
selected_features_rf = np.array(X.columns)[sorted_indices]
top_n = 5
X_train_selected = X_train[selected_features_rf[:top_n]]
X_test_selected = X_test[selected_features_rf[:top_n]]
```

*# Step 3: Train XGBoost on the reduced dataset (with feature selection)*

```
xgb_model_selected = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb_model_selected.fit(X_train_selected, y_train)
y_pred_selected = xgb_model_selected.predict(X_test_selected)
accuracy_selected = accuracy_score(y_test, y_pred_selected)
f1_selected = f1_score(y_test, y_pred_selected)
precision_selected = precision_score(y_test, y_pred_selected)
recall_selected = recall_score(y_test, y_pred_selected)
print("Performance on Original Dataset:")
print(f"Accuracy: {accuracy_original:.4f}, F1-Score: {f1_original:.4f}, Precision: {precision_original:.4f}, Recall: {recall_original:.4f}")
print("\nPerformance on Reduced Dataset (With Feature Selection):")
print(f"Accuracy: {accuracy_selected:.4f}, F1-Score: {f1_selected:.4f}, Precision: {precision_selected:.4f}, Recall: {recall_selected:.4f}")
```

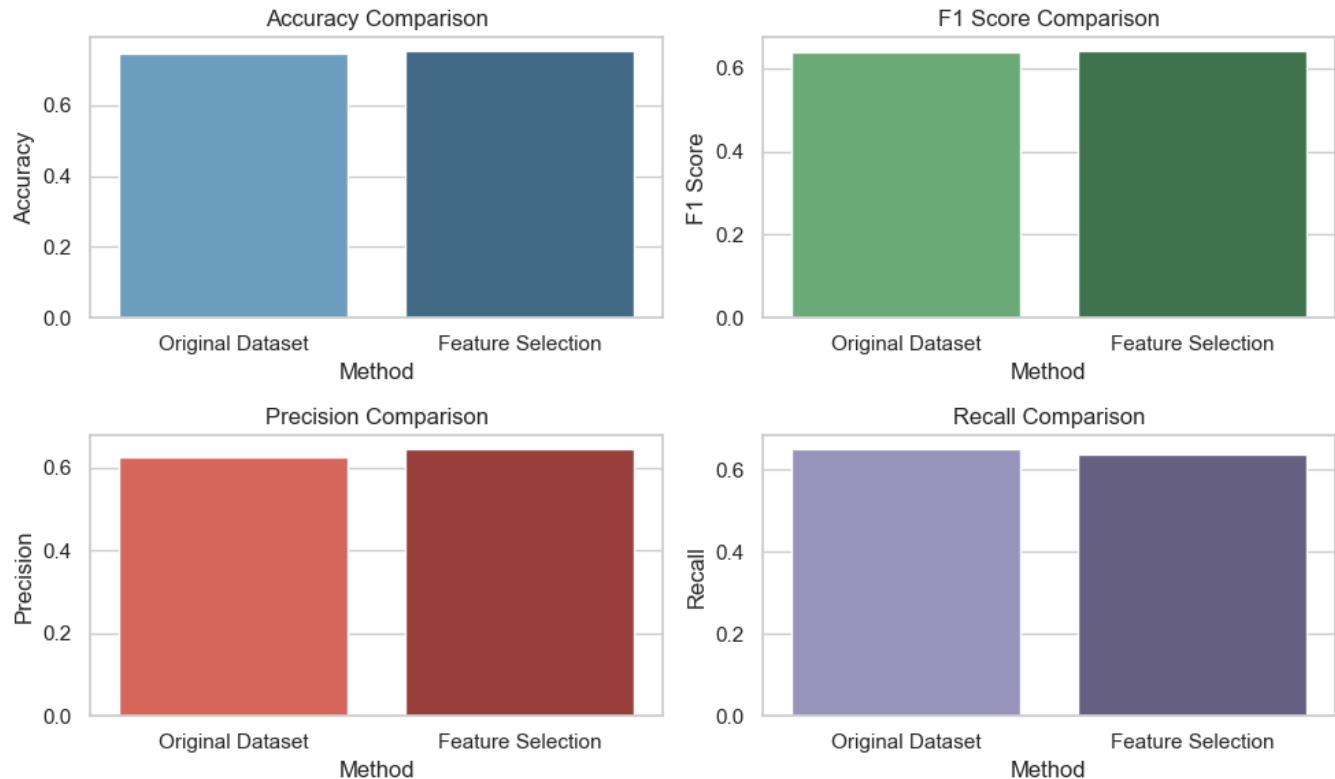
## Output - Visualisation

**Performance on Original Dataset:**

Accuracy: 0.7446, F1-Score: 0.6380, Precision: 0.6265, Recall: 0.6500

**Performance on Reduced Dataset (With Feature Selection):**

Accuracy: 0.7532, F1-Score: 0.6415, Precision: 0.6456, Recall: 0.6375



## LAB ASSIGNMENT 6

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Apply Different Machine Learning Approaches for the Classification Task. Compare the Performance of Different ML Approaches in Terms of Accuracy, Precision, and Recall.		

### Objective

The objective of this lab assignment is to apply various Machine Learning (ML) approaches to a classification task and compare their performance in terms of accuracy, precision, and recall. You will gain hands-on experience in implementing and evaluating different ML algorithms, understanding their strengths and weaknesses, and interpreting their results.

### Code

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("data.csv")
data.head()
data_cleaned = data.drop(columns=["id", "Unnamed: 32"])
label_encoder = LabelEncoder()
data_cleaned["diagnosis"] = label_encoder.fit_transform(data_cleaned["diagnosis"])

X = data_cleaned.drop(columns=["diagnosis"])

```

```
y = data_cleaned["diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
models = {
    "Logistic Regression": LogisticRegression(max_iter=10000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Support Vector Machine": SVC(),
}
metrics = {"Model": [], "Accuracy": [], "Precision": [], "Recall": []}

for model_name, model in models.items():
    if model_name in [
        "Logistic Regression",
        "K-Nearest Neighbors",
        "Support Vector Machine",
    ]:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    metrics["Model"].append(model_name)
```

```

metrics["Accuracy"].append(accuracy_score(y_test, y_pred))
metrics["Precision"].append(precision_score(y_test, y_pred))
metrics["Recall"].append(recall_score(y_test, y_pred))
metrics_df = pd.DataFrame(metrics)
print(metrics_df)

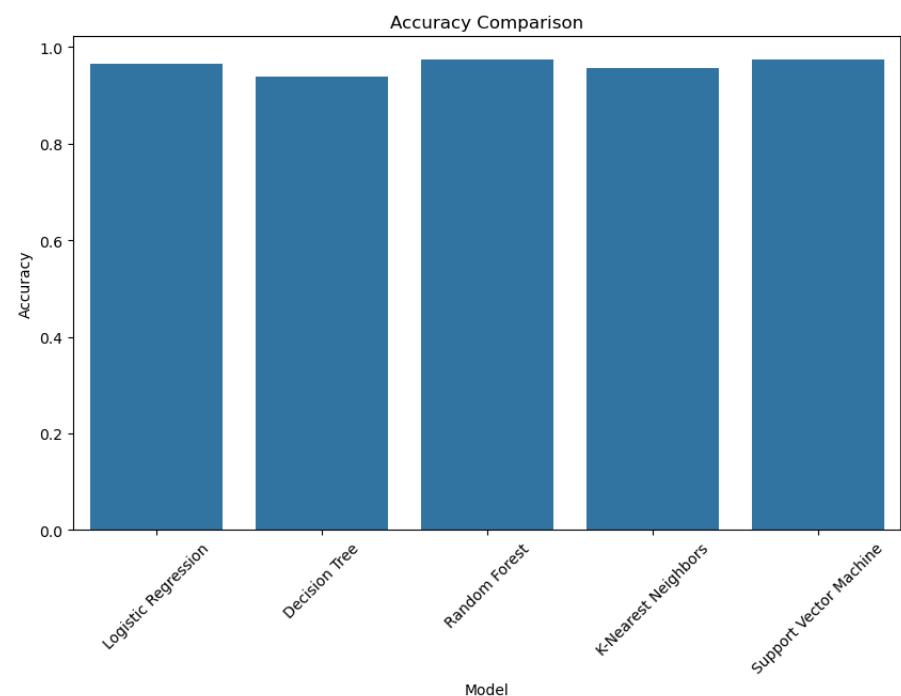
```

```

plt.figure(figsize=(10, 6))
sns.barplot(x="Model", y="Accuracy", data=metrics_df)
plt.title("Accuracy Comparison")
plt.xticks(rotation=45)
plt.show()

```

	Model	Accuracy	Precision	Recall
0	Logistic Regression	0.964912	0.975000	0.928571
1	Decision Tree	0.938596	0.906977	0.928571
2	Random Forest	0.973684	1.000000	0.928571
3	K-Nearest Neighbors	0.956140	0.974359	0.904762
4	Support Vector Machine	0.973684	1.000000	0.928571



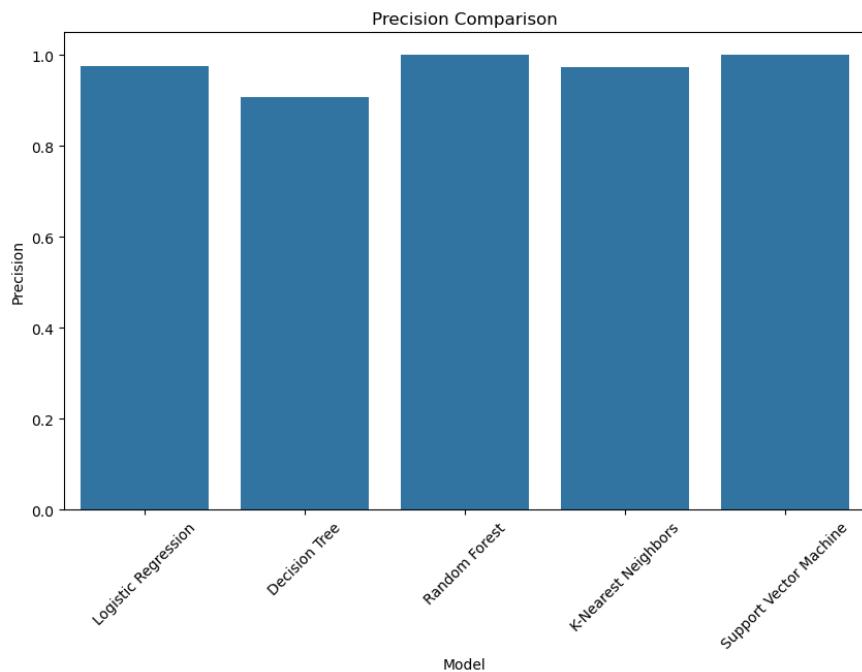
```

plt.figure(figsize=(10, 6))
sns.barplot(x="Model", y="Precision", data=metrics_df)
plt.title("Precision Comparison")

```

```
plt.xticks(rotation=45)
```

```
plt.show()
```



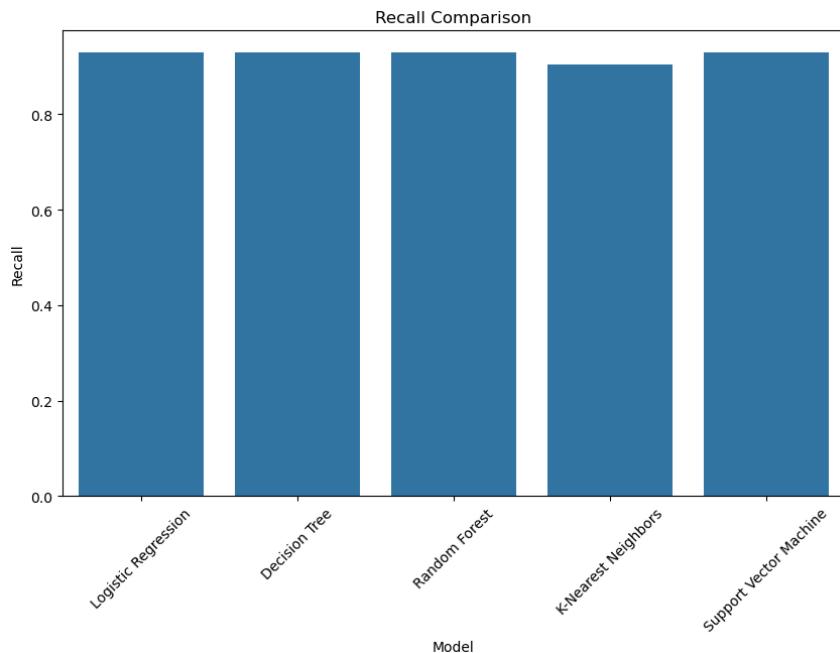
```
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x="Model", y="Recall", data=metrics_df)
```

```
plt.title("Recall Comparison")
```

```
plt.xticks(rotation=45)
```

```
plt.show()
```



```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [50, 100, 150],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
}

grid_search = GridSearchCV(
    RandomForestClassifier(), param_grid, cv=5, scoring="accuracy"
)
grid_search.fit(X_train, y_train)

print("Best parameters for Random Forest:", grid_search.best_params_)
best_rf = grid_search.best_estimator_
y_pred_best_rf = best_rf.predict(X_test)

accuracy_best_rf = accuracy_score(y_test, y_pred_best_rf)
precision_best_rf = precision_score(y_test, y_pred_best_rf)
recall_best_rf = recall_score(y_test, y_pred_best_rf)

print(
    f"Best Random Forest - Accuracy: {accuracy_best_rf}, Precision: {precision_best_rf}, Recall: {recall_best_rf}"
)
```

## Output

```
Best parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
Best Random Forest - Accuracy: 0.9649122807017544, Precision: 1.0, Recall: 0.9047619047619048
```

## LAB ASSIGNMENT 7

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Train a Machine Learning Classifier on an Imbalanced Dataset. Then Balance the Dataset Using Oversampling Techniques. Compare the Model Performance Before and After Oversampling.		

### Objective

In this lab assignment, you will work with an imbalanced dataset and train a machine learning classifier on it. Afterward, you will apply oversampling techniques to balance the dataset and compare the model's performance before and after oversampling. The goal is to observe how oversampling affects the classifier's performance when dealing with imbalanced data.

**Dataset:** <https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv>

### Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
column_names = [
    "Pregnancies",
    "Glucose",
    "BloodPressure",
    "SkinThickness",
    "Insulin",
    "BMI",
    "DiabetesPedigreeFunction",
    "Age",
    "Outcome",
]

```

```
data = pd.read_csv(url, names=column_names)
print("\nClass Distribution:\n", data["Outcome"].value_counts())
```

```
Class Distribution:
  Outcome
  0    500
  1    268
Name: count, dtype: int64
```

```
X = data.drop("Outcome", axis=1)
y = data["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("\nBefore Oversampling:")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Before Oversampling:
[[85 15]
 [22 32]]
      precision    recall   f1-score   support
          0       0.79      0.85      0.82      100
          1       0.68      0.59      0.63       54

      accuracy          0.76      154
     macro avg       0.74      0.72      0.73      154
  weighted avg       0.75      0.76      0.76      154
```

```

smote = SMOTE(random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

clf_smote = RandomForestClassifier(random_state=42)

clf_smote.fit(X_train_smote, y_train_smote)

y_pred_smote = clf_smote.predict(X_test)

print("\nAfter Oversampling (SMOTE):")

print(confusion_matrix(y_test, y_pred_smote))

print(classification_report(y_test, y_pred_smote))

```

After Oversampling (SMOTE):				
[[74 26]				
[15 39]]				
	precision	recall	f1-score	support
0	0.83	0.74	0.78	100
1	0.60	0.72	0.66	54
 accuracy				0.73
macro avg				0.72
weighted avg				0.74
				154

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```

def evaluate_model(y_true, y_pred):

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    return {
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
    }

```

```
y_pred = clf.predict(X_test)
results_before = evaluate_model(y_test, y_pred)

y_pred_smote = clf_smote.predict(X_test)
results_after = evaluate_model(y_test, y_pred_smote)

print("\nPerformance Metrics Comparison:")
print(f"{'Metric':<15} {'Before Oversampling':<25} {'After Oversampling':<25}")
for metric in results_before.keys():
    print(
        f"{metric.capitalize():<15} {results_before[metric]:<25.4f} {results_after[metric]:<25.4f}"
    )
```

## Output

Performance Metrics Comparison:		
Metric	Before Oversampling	After Oversampling
Accuracy	0.7597	0.7338
Precision	0.6809	0.6000
Recall	0.5926	0.7222
F1_score	0.6337	0.6555

## LAB ASSIGNMENT 8

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Apply Different Feature Selection Approaches for the Classification/Regression Task. Compare the Performance of Different Feature Selection Approaches.		

### Objective

The objective of this lab assignment is to explore various feature selection techniques for classification and regression tasks.

**Dataset:** Use the UCI Iris dataset for the classification task and the Boston Housing dataset for the regression task.

### Code

```

import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency
class SelectKBest:
    def __init__(self, score_func, k='all'):
        self.score_func = score_func
        self.k = k
        self.scores_ = None
        self.selected_features_ = None

    def fit(self, X, y):
        y = y.ravel()
        self.scores_ = np.array([self.score_func(X.iloc[:, i], y) for i in range(X.shape[1])])

        if self.k == 'all':
            self.k = X.shape[1]
            self.selected_indices_ = np.argsort(self.scores_)[-self.k:]
            self.selected_features_ = X.columns[self.selected_indices_]

        return self

```

```
def transform(self, X):
    return X.iloc[:, self.selected_indices_]

def fit_transform(self, X, y):
    return self.fit(X, y).transform(X)

def chi2_score(x, y):
    contingency_table = pd.crosstab(x, y)
    chi2_stat, _, _, _ = chi2_contingency(contingency_table)
    return chi2_stat

from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.utils import check_X_y, check_array
class RFE:
    def __init__(self, estimator, n_features_to_select=1):
        self.estimator = estimator
        self.n_features_to_select = n_features_to_select
        self.support_ = None
        self.ranking_ = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        n_features = X.shape[1]
        self.ranking_ = np.zeros(n_features)

        while n_features > self.n_features_to_select:
            self.estimator.fit(X, y)
            importances = self.estimator.feature_importances_ if hasattr(self.estimator, 'feature_importances_')
            else np.abs(self.estimator.coef_)
            worst_feature = np.argmin(importances)
            self.ranking_[worst_feature] += 1
```

```
X = np.delete(X, worst_feature, axis=1)

n_features -= 1

self.support_ = np.where(self.ranking_ == 0)[0]
return self

def transform(self, X):
    return X.iloc[:, self.support_]

def fit_transform(self, X, y):
    return self.fit(X, y).transform(X)

from sklearn.linear_model import Lasso
from sklearn.utils import check_X_y
class L1FeatureSelector:

    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.model = Lasso(alpha=self.alpha)
        self.selected_features_ = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.model.fit(X, y)
        self.selected_features_ = np.where(self.model.coef_ != 0)[0]
        return self

    def transform(self, X):
        return X.iloc[:, self.selected_features_]

    def fit_transform(self, X, y):
        return self.fit(X, y).transform(X)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import check_X_y
class TreeBasedFeatureSelector:
    def __init__(self, n_estimators=100):
        self.n_estimators = n_estimators
        self.model = RandomForestClassifier(n_estimators=self.n_estimators)
        self.selected_features_ = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.model.fit(X, y)
        importances = self.model.feature_importances_
        self.selected_features_ = np.where(importances > 0)[0]
        return self

    def transform(self, X):
        return X.iloc[:, self.selected_features_]

    def fit_transform(self, X, y):
        return self.fit(X, y).transform(X)
```

```
# Sample Data
import pandas as pd
from sklearn.datasets import load_iris
```

```
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

```
# Univariate Feature Selection
selector_kbest = SelectKBest(score_func=chi2_score, k=2)
X_kbest = selector_kbest.fit_transform(X, y)
```

```
print("Selected Features (KBest):", selector_kbest.selected_features_)

# Recursive Feature Elimination
rfe_selector = RFE(estimator=RandomForestClassifier(), n_features_to_select=2)
X_rfe = rfe_selector.fit_transform(X, y)
print("Selected Features (RFE):", X.columns[rfe_selector.support_])

# L1-Based Feature Selection
l1_selector = L1FeatureSelector(alpha=0.1)
X_l1 = l1_selector.fit_transform(X, y)
print("Selected Features (L1):", X.columns[l1_selector.selected_features_])

# Tree-Based Feature Selection
tree_selector = TreeBasedFeatureSelector()
X_tree = tree_selector.fit_transform(X, y)
print("Selected Features (Tree-Based):", X.columns[tree_selector.selected_features_])
```

## Output

```
Selected Features (KBest): Index(['petal width (cm)', 'petal length (cm)'], dtype='object')
Selected Features (RFE): Index(['petal length (cm)', 'petal width (cm)'], dtype='object')
Selected Features (L1): Index(['petal length (cm)'], dtype='object')
Selected Features (Tree-Based): Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'],
                                     dtype='object')
```

## LAB ASSIGNMENT 9

<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Write a Program to Demonstrate the Working of the Decision Tree-Based CART Algorithm. Build the Decision Tree and Classify a New Sample Using a Suitable Dataset. Compare the Performance with That of ID3, C4.5, and CART in Terms of Accuracy, Recall, Precision, and Sensitivity.		

### Objective

The objective of this lab assignment is to implement the Decision Tree-based Classification and Regression Trees (CART) algorithm and compare its performance with other decision tree algorithms, namely ID3 and C4.5, in terms of accuracy, recall, precision, and sensitivity. The assignment includes building decision trees, classifying new samples, and evaluating the models using a suitable dataset.

### Code

```
import numpy as np
import pandas as pd
```

```
class TreeNode:
```

```
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
```

```
class CART:
```

```
    def fit(self, X, y):
        self.root = self._grow_tree(X, y)
```

```
    def _grow_tree(self, X, y):
```

```
        if len(set(y)) == 1:
            return TreeNode(value=y[0])
```

```
        best_gain = -float('inf')
```

```

best_criteria = None
best_sets = None

n_samples, n_features = X.shape
for feature_index in range(n_features):
    thresholds = np.unique(X[:, feature_index])
    for threshold in thresholds:
        left_indices = X[:, feature_index] <= threshold
        right_indices = X[:, feature_index] > threshold

        if len(y[left_indices]) == 0 or len(y[right_indices]) == 0:
            continue

        gain = self._gini_gain(y, left_indices, right_indices)
        if gain > best_gain:
            best_gain = gain
            best_criteria = (feature_index, threshold)
            best_sets = (left_indices, right_indices)

    if best_gain == -float('inf'):
        return TreeNode(value=np.bincount(y).argmax())

left_node = self._grow_tree(X[best_sets[0]], y[best_sets[0]])
right_node = self._grow_tree(X[best_sets[1]], y[best_sets[1]])

return TreeNode(feature_index=best_criteria[0], threshold=best_criteria[1], left=left_node,
               right=right_node)

def _gini_gain(self, y, left_indices, right_indices):
    parent_impurity = self._gini_impurity(y)
    left_impurity = self._gini_impurity(y[left_indices])
    right_impurity = self._gini_impurity(y[right_indices])
    n = len(y)

```

```

n_left = len(y[left_indices])
n_right = len(y[right_indices])
weighted_child_impurity = (n_left / n) * left_impurity + (n_right / n) * right_impurity
return parent_impurity - weighted_child_impurity

```

```

def _gini_impurity(self, y):
    class_counts = np.bincount(y)
    return 1.0 - np.sum((class_counts / len(y)) ** 2)

```

```

def predict(self, X):
    return [self._predict(inputs) for inputs in X]

```

```

def _predict(self, inputs):
    node = self.root
    while node.value is None:
        if inputs[node.feature_index] <= node.threshold:
            node = node.left
        else:
            node = node.right
    return node.value

```

```
from sklearn.tree import DecisionTreeClassifier
```

```

def build_id3(X_train, y_train):
    model = DecisionTreeClassifier(criterion='entropy') # ID3 uses entropy
    model.fit(X_train, y_train)
    return model

```

```

def build_c45(X_train, y_train):
    model = DecisionTreeClassifier(criterion='gini') # C4.5 uses Gini impurity
    model.fit(X_train, y_train)
    return model

```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_matrix

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test) if hasattr(model, 'predict') else model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)

    # Confusion matrix
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0 # Avoid division by zero

    return accuracy, recall, precision, sensitivity
```

```
from sklearn.model_selection import train_test_split
```

```
def load_titanic_dataset():
    url = "https://raw.githubusercontent.com/datasets/master/titanic.csv"
    data = pd.read_csv(url)

    # Preprocess the data: drop rows with missing target and fill missing values
    data = data.drop(columns=["Name", "Ticket", "Cabin"]) # Drop non-feature columns
    data['Embarked'] = data['Embarked'].fillna('S') # Fill missing Embarked
    data['Age'] = data['Age'].fillna(data['Age'].median()) # Fill missing Age with median
    data['Fare'] = data['Fare'].fillna(data['Fare'].median()) # Fill missing Fare with median

    # Convert categorical variables to dummy variables
    data = pd.get_dummies(data, columns=["Sex", "Embarked"], drop_first=True)

    X = data.drop("Survived", axis=1) # Features
    y = data["Survived"] # Target labels
```

```
return train_test_split(X, y, test_size=0.2, random_state=42)

import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

def compare_models():
    X_train, X_test, y_train, y_test = load_titanic_dataset()

    # Implement CART from scratch
    cart_model = CART()
    cart_model.fit(X_train.to_numpy(), y_train.to_numpy())

    # Using scikit-learn for ID3 and C4.5
    id3_model = build_id3(X_train, y_train)
    c45_model = build_c45(X_train, y_train)

    # Evaluate models
    cart_metrics = evaluate_model(cart_model, X_test.to_numpy(), y_test.to_numpy())
    id3_metrics = evaluate_model(id3_model, X_test, y_test)
    c45_metrics = evaluate_model(c45_model, X_test, y_test)

    # Print comparison
    print("CART Metrics (Accuracy, Recall, Precision, Sensitivity):", cart_metrics)
    print("ID3 Metrics (Accuracy, Recall, Precision, Sensitivity):", id3_metrics)
    print("C4.5 Metrics (Accuracy, Recall, Precision, Sensitivity):", c45_metrics)

    # Visualize the CART tree
    plt.figure(figsize=(12, 8))
    plot_tree(id3_model, filled=True, feature_names=X_train.columns, class_names=['Not Survived', 'Survived'])
    plt.title("ID3 Decision Tree")
    plt.show()
```

```

plt.figure(figsize=(12, 8))

plot_tree(c45_model, filled=True, feature_names=X_train.columns, class_names=['Not Survived', 'Survived'])

plt.title("C4.5 Decision Tree")

plt.show()

# Run the comparison

compare_models()

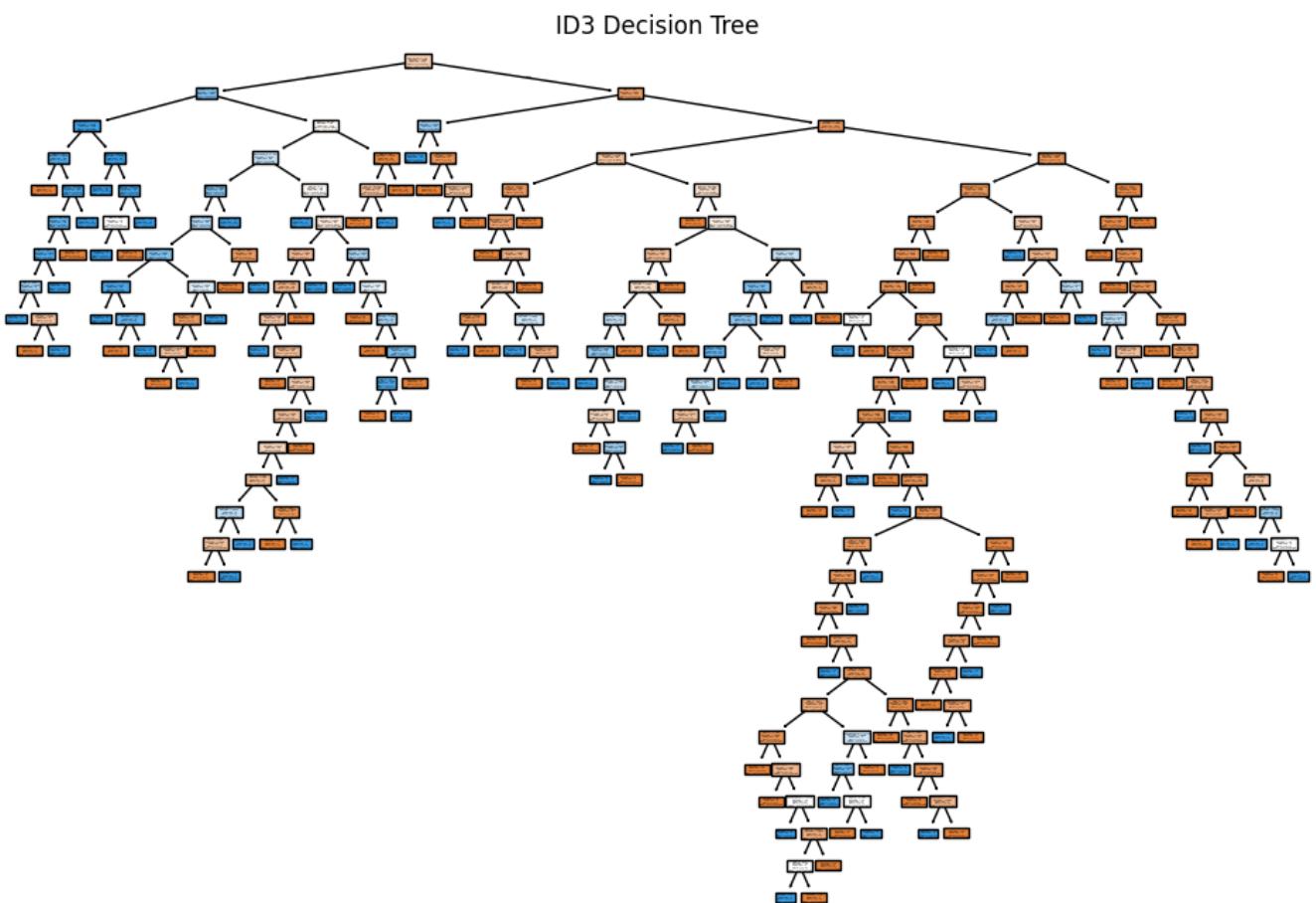
```

## Output

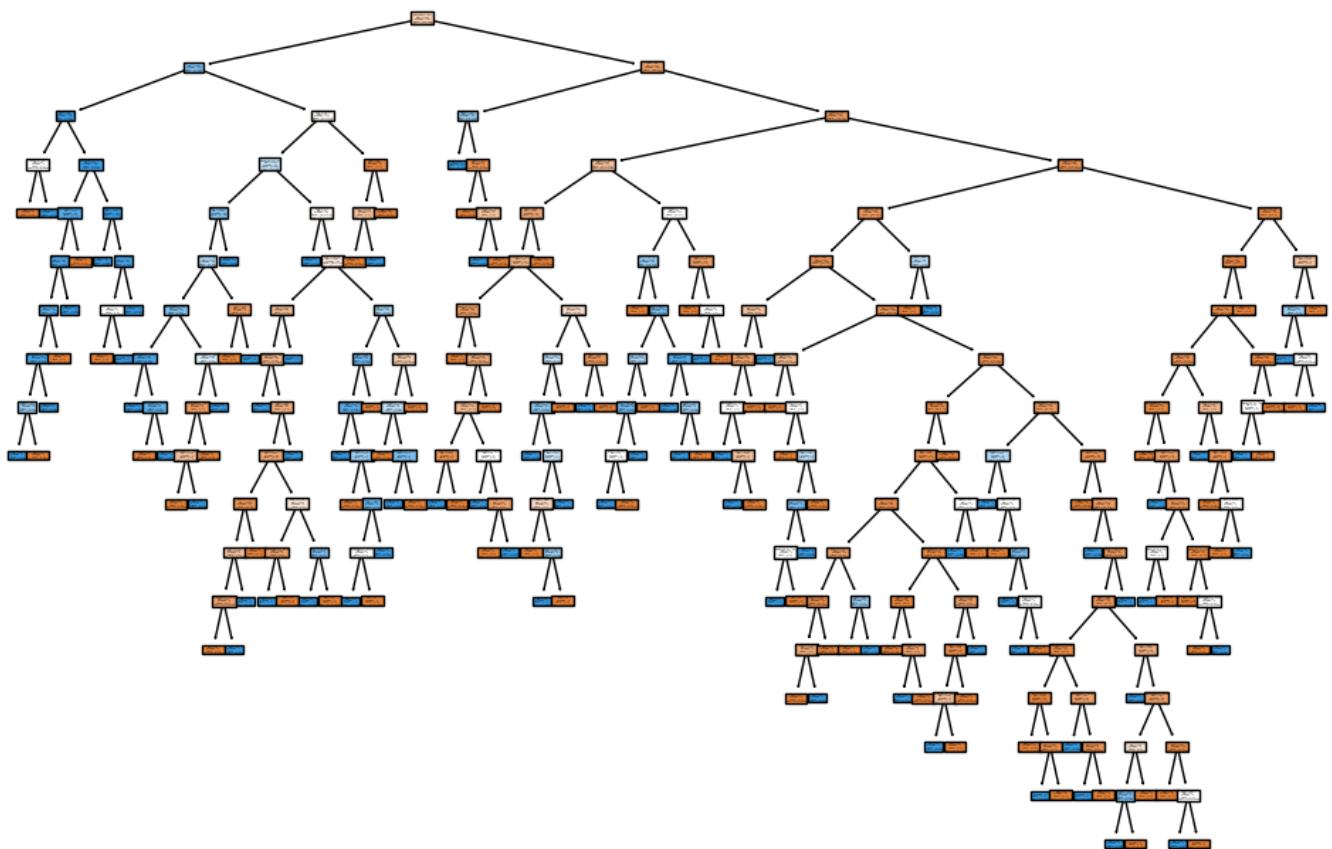
```

CART Metrics (Accuracy, Recall, Precision, Sensitivity): (0.6927374301675978, 0.6891891891891891, 0.6144578313253012, 0.6891891891891891)
ID3 Metrics (Accuracy, Recall, Precision, Sensitivity): (0.7262569832402235, 0.6891891891891891, 0.6623376623376623, 0.6891891891891891)
C4.5 Metrics (Accuracy, Recall, Precision, Sensitivity): (0.7262569832402235, 0.6756756756756757, 0.6666666666666666, 0.6756756756756757)

```



## C4.5 Decision Tree



## LAB ASSIGNMENT 10

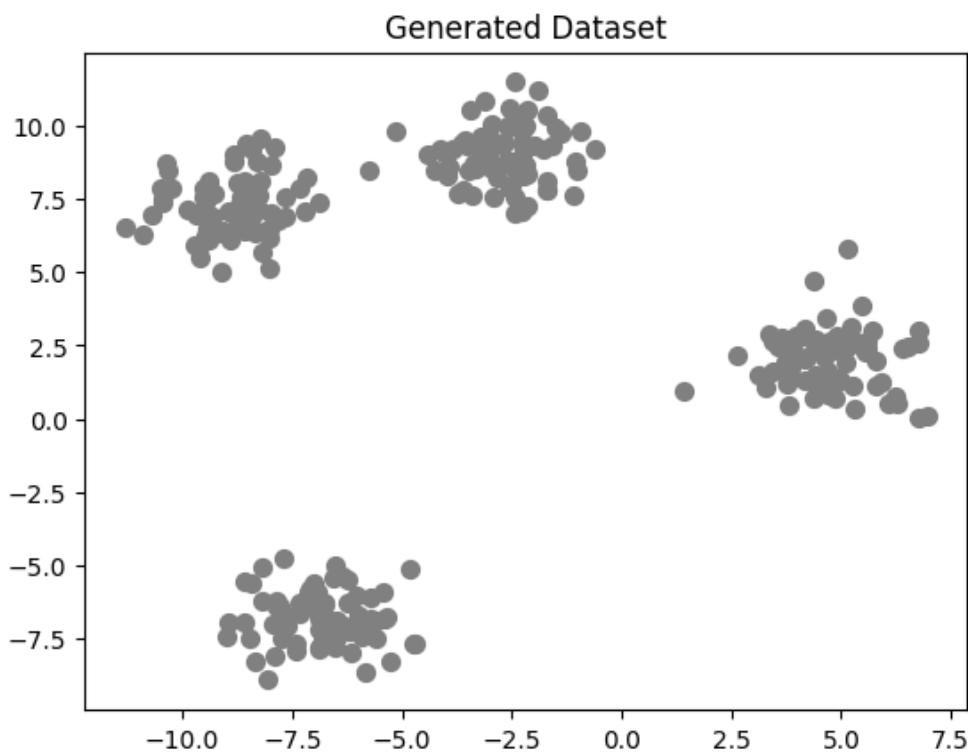
<b>Name:</b>	Harsh Shah	<b>Roll No.:</b>	21BCP359
<b>Division:</b>	6	<b>Batch:</b>	G11
<b>Aim:</b>	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.		

### Objective

The objective of this lab assignment is to implement dimensionality reduction using Principal Component Analysis (PCA) and gain hands-on experience in reducing the dimensionality of a dataset while preserving its essential information.

### Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)
plt.scatter(X[:, 0], X[:, 1], s=50, c='gray')
plt.title('Generated Dataset')
plt.show()
```



*class KMeans:*

*def \_\_init\_\_(self, k, max\_iters=100):*

*self.k = k*

*self.max\_iters = max\_iters*

*self.centroids = None*

*def initialize\_centroids(self, X):*

*np.random.seed(42)*

*random\_indices = np.random.permutation(X.shape[0])*

*centroids = X[random\_indices[:self.k]]*

*return centroids*

*def assign\_clusters(self, X, centroids):*

*distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)*

*return np.argmin(distances, axis=1)*

*def update\_centroids(self, X, labels):*

*new\_centroids = np.array([X[labels == i].mean(axis=0) for i in range(self.k)])*

*return new\_centroids*

*def fit(self, X):*

*self.centroids = self.initialize\_centroids(X)*

*for i in range(self.max\_iters):*

*labels = self.assign\_clusters(X, self.centroids)*

*new\_centroids = self.update\_centroids(X, labels)*

*if np.all(self.centroids == new\_centroids):*

*break*

*self.centroids = new\_centroids*

*return self.centroids, labels*

```

def inertia(self, X, labels):
    return np.sum((X - self.centroids[labels]) ** 2)

kmeans = KMeans(k=4)

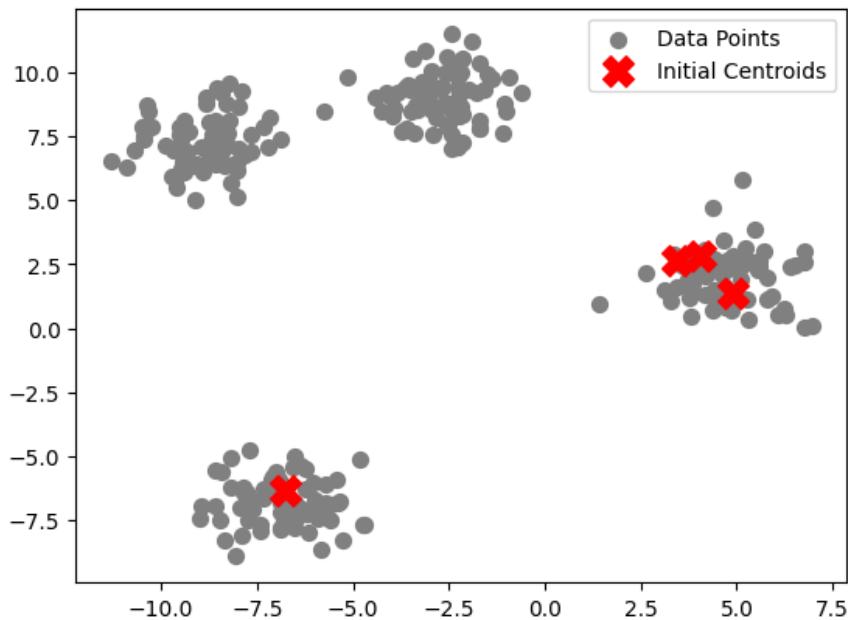
initial_centroids = kmeans.initialize_centroids(X)

centroids, labels = kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], s=50, c='gray', label='Data Points')
plt.scatter(initial_centroids[:, 0], initial_centroids[:, 1], s=200, c='red', marker='X', label='Initial Centroids')
plt.title('Initial Random Centroids')
plt.legend()
plt.show()

```

Initial Random Centroids

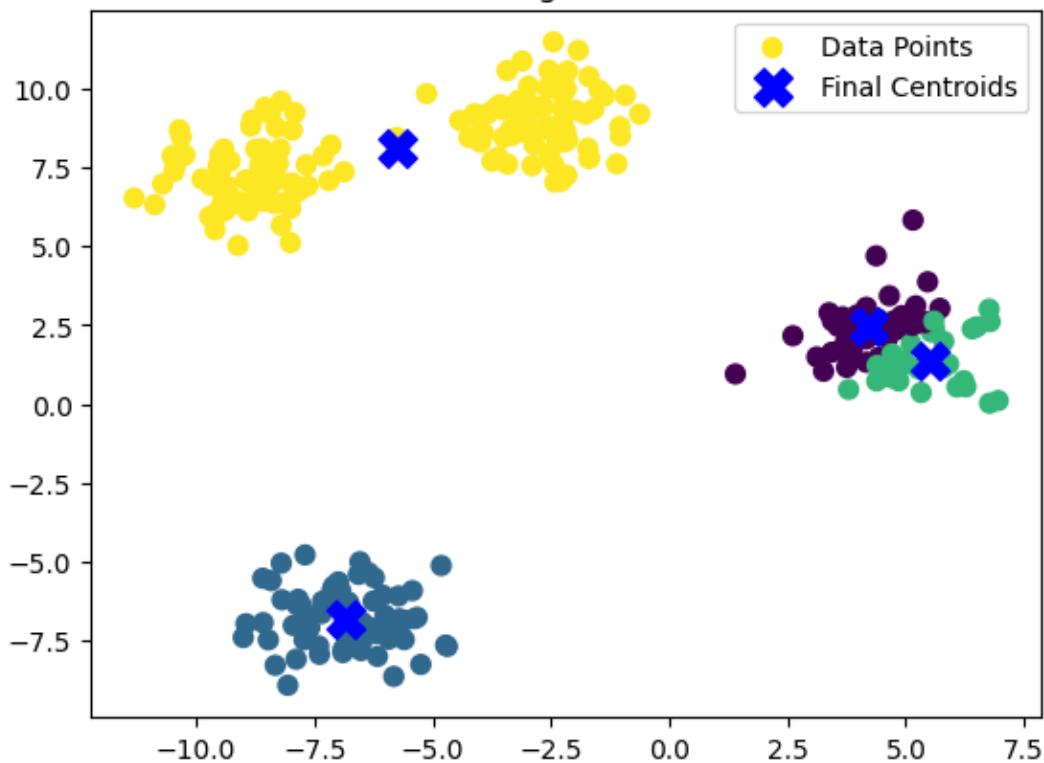


```

plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis', label='Data Points')
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='blue', marker='X', label='Final Centroids')
plt.title('K-Means Clustering with Final Centroids')
plt.legend()
plt.show()

```

### K-Means Clustering with Final Centroids



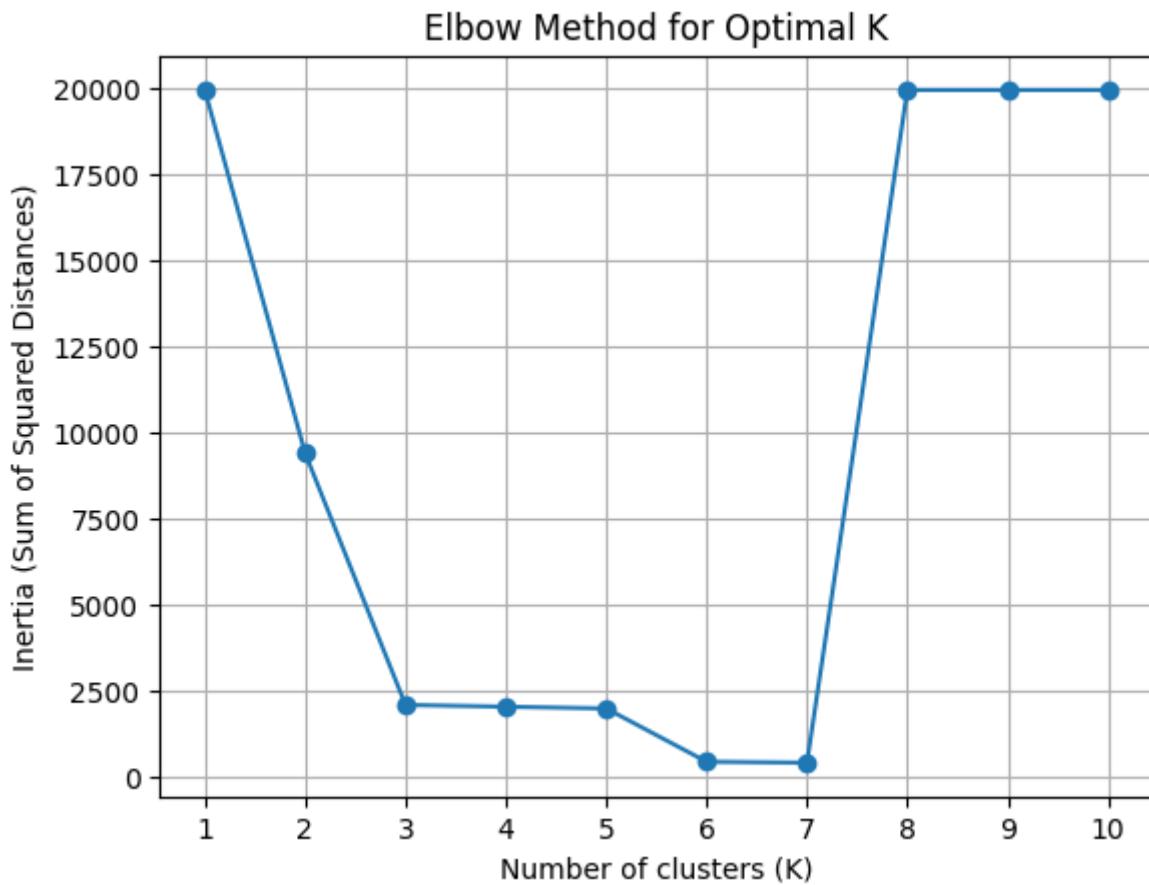
```

inertia_values = []
K_values = range(1, 11)

for k in K_values:
    kmeans = KMeans(k=k)
    centroids, labels = kmeans.fit(X)
    inertia = kmeans.inertia(X, labels)
    inertia_values.append(inertia)

plt.plot(K_values, inertia_values, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(K_values)
plt.grid()
plt.show()

```



for k, inertia in zip(K\_values, inertia\_values):

```

print(f'K = {k}, Inertia = {inertia:.2f}')

optimal_k = K_values[np.argmin(inertia_values)]
print(f"The optimal K is: {optimal_k}")

kmeans_optimal = KMeans(k=optimal_k)

centroids_optimal, labels_optimal = kmeans_optimal.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=labels_optimal, s=50, cmap='viridis', label='Data Points')
plt.scatter(centroids_optimal[:, 0], centroids_optimal[:, 1], s=200, c='blue', marker='X', label='Final Centroids')

plt.title(f'K-Means Clustering with Optimal K = {optimal_k}')
plt.legend()
plt.show()

```

## Output

```
K = 1, Inertia = 19953.77
K = 2, Inertia = 9416.21
K = 3, Inertia = 2110.41
K = 4, Inertia = 2058.54
K = 5, Inertia = 2007.60
K = 6, Inertia = 461.71
K = 7, Inertia = 434.73
K = 8, Inertia = 19953.77
K = 9, Inertia = 19953.77
K = 10, Inertia = 19953.77
```

The optimal K is: 7

