UNIVERSITY OF ENGINEERING AND TECHNOLOGY (NEW CAMPUS), LAHORE

INTRODUCTION TO MACHINE LEARNING

Project

Session 2021

Submitted By:

MUHAMMAD ASAD	(2021-EE-267)
MUHAMMAD RAMZAN	(2021-EE-268)
AHTISHAM NADEEM	(2021-EE-332)

Submitted To:

Dr. HARIS ANWAR

ELECTRICAL ENGINEERING DEPARTMENT

PREDICTING Electrical Faults Detection in a Transmission Line

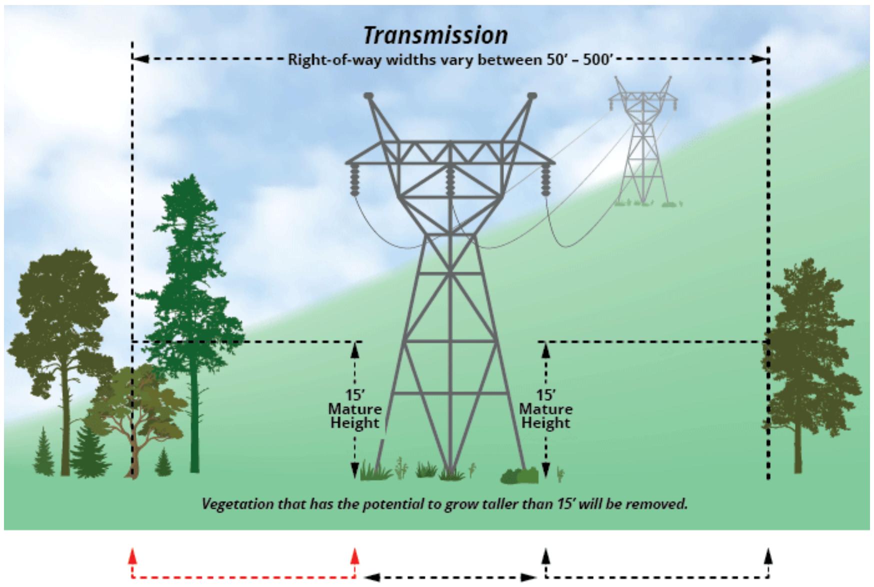
Introduction:

The objective of this project was to develop a predictive model to determine the electrical fault analysis in transmission line. Fault Prediction using decision tree algorithm.

Methodology:

1. **Data Preprocessing:** The dataset was preprocessed by handling missing values, encoding categorical variables, and scaling numeric features.
2. **Splitting the Data**: The preprocessed data was split into training and testing sets, with a ratio of 80:20.
3. **Model Selection and Training:** Traditional machine learning algorithms, including Logistic Regression, Naive Bayes, Decision Tree, and Random Forest, were trained on the training set.
4. **Evaluation Metrics:** The performance of each model was evaluated using metrics such as accuracy, precision, recall, and F1-score on both the training and testing sets.
5. **Feature Importance:** For the Decision Tree model, the important features contributing to the predictions were visualized using a bar plot.

Electrical Faults Detection and Classification


Transmission
Right-of-way widths vary between 50' – 500'
15' Mature Height
15' Mature Height
Vegetation that has the potential to grow taller than 15' will be removed.

The transmission line is the most crucial part of the power system. The requirement of power and its allegiance has grown up exponentially over the modern era, and the prominent role of a transmission line is to transmit electric power from the source area to the distribution network. The electrical power system consists of so many complex dynamic and interacting elements that are always prone to disturbance or an electrical fault.

The power system consists of 4 generators of 11 × 10^3 V, each pair located at each end of the transmission line. Transformers are present in between to simulate and study the various faults at the midpoint of the transmission line.

What are Electrical Faults?

Normally, a power system operates under balanced conditions. When the system becomes unbalanced due to the failures of insulation at any point or due to the contact of live wires, a short–circuit or fault, is said to occur in the line. Faults may occur in the power system due to the number of reasons like natural disturbances (lightning, high-speed winds, earthquakes), insulation breakdown, falling of a tree, bird shorting, etc.

Types of Faults?

Faults can be categorized into two types:

1. Open-circuit Fault
2. Short-Circuit Faults

Short-Circuit Faults:

1. Symmetrical
2. Asymmetrical Faults

Symmetrical and Asymmetrical Faults

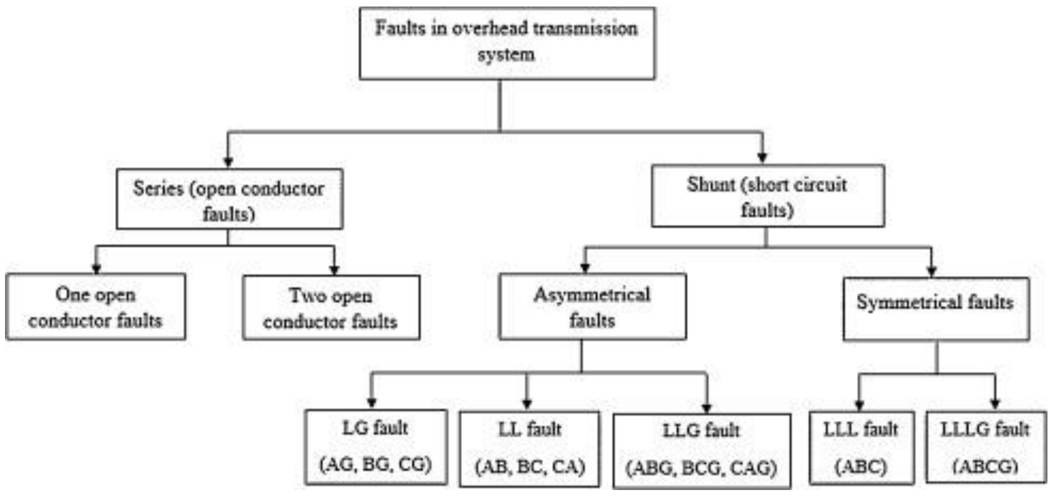
Symmetrical Faults:

- In symmetrical faults, all phases are shorted to each other or to earth (L-L-L) or (L-L-L-G).
- The nature of this type of fault is balanced.
- In this type of fault, fault currents in all phases are symmetrical i.e. their magnitudes are equal and they are equally displaced by angle 120 degree.
- It is more severe type of fault but it occurs rarely.

Asymmetrical Faults:

These faults involve only one or two phases.

- In this type of fault, three phase lines become unbalanced.
- There are mainly three types namely line to ground (L-G), line to line (L-L) and double line to ground (LL-G) faults.
- These types of faults mostly occur on power system.


Faults in overhead transmission system
Series (open conductor faults)
Shunt (short circuit faults)
One open conductor faults
Two open conductor faults
Asymmetrical faults
Symmetrical faults
LG fault (AG, BG, CG)
LL fault (AB, BC, CA)
LLG fault (ABG, BCG, CAG)
LLL fault (ABC)
LLLG fault (ABCG)

About this dataset file:

This file contains the dataset to classify the types of faults.

Inputs - [Ia,Ib,Ic,Va,Vb,Vc]

Ia = Current in line A
Ib = Current in line B
Ic = Current in line C

Va = Voltage in line A
Vb = Voltage in line B
Vc = Voltage in line C

Output: [G C B A]

[0 0 0 0] - No Fault
[1 0 0 1] - LG fault (Between Phase A and Ground)
[0 0 1 1] - LL fault (Between Phase A and Phase B)
[1 0 1 1] - LLG Fault (Between Phases A, B and Ground)
[0 1 1 1] - LLL Fault (Between all three phases)
[1 1 1 1] - LLLG fault (Three phase symmetrical fault)

Here, we conclude that there are 6 types of faults, hence 6 output classes.

Objectives:

- Dataset exploration using various types of data visualization.
- Build various Machine Learning models that can predict the Fault type in transmission Line.

Python Implementation

Import Libraries:

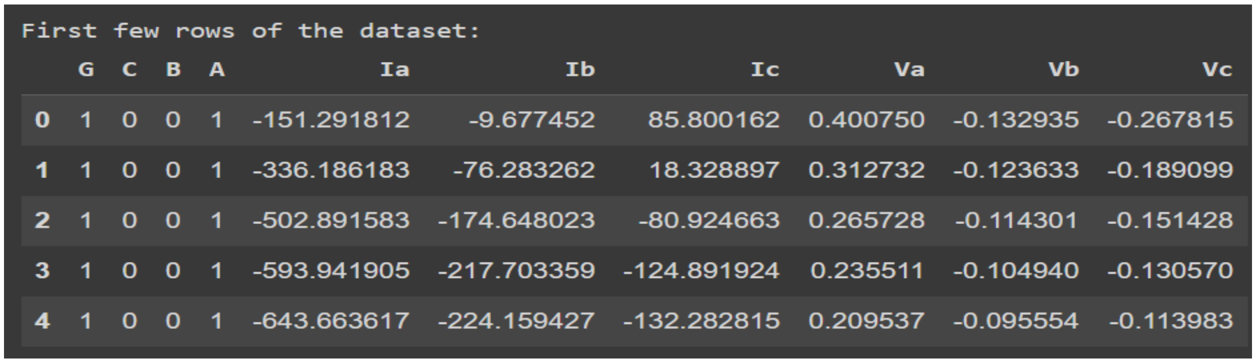
```
''' Install these Libraries
pip install pandas
pip install numpy
pip install scikit-learn
pip install matplotlib
pip install seaborn
pip install jinja2
from IPython.display import clear_output
!pip3 install -U lazypredict
'''
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
import time
from tabulate import tabulate
```

Load the dataset and perform exploratory data analysis to get an understanding of the data.

Let's begin by loading the dataset and performing exploratory data analysis (EDA) on the power system fault analysis.

Step 1: Loading the Dataset

```
# Load the dataset
df = pd.read_csv('classData.csv')
# Display the first few rows of the dataset
print("First few rows of the dataset:")
df.head()
```

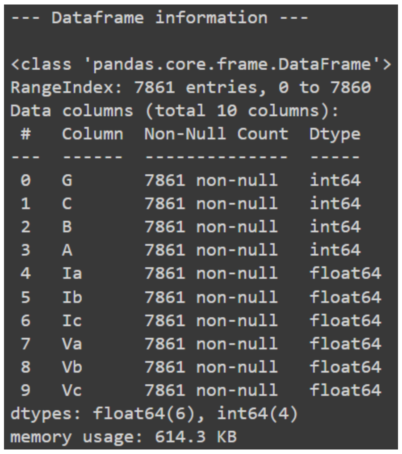


```
First few rows of the dataset:
   G  C  B  A          Ia          Ib          Ic        Va        Vb        Vc
0  1  0  0  1 -151.291812   -9.677452   85.800162  0.400750 -0.132935 -0.267815
1  1  0  0  1 -336.186183  -76.283262   18.328897  0.312732 -0.123633 -0.189099
2  1  0  0  1 -502.891583 -174.648023  -80.924663  0.265728 -0.114301 -0.151428
3  1  0  0  1 -593.941905 -217.703359 -124.891924  0.235511 -0.104940 -0.130570
4  1  0  0  1 -643.663617 -224.159427 -132.282815  0.209537 -0.095554 -0.113983
```

Step 2: Exploratory Data Analysis (EDA)

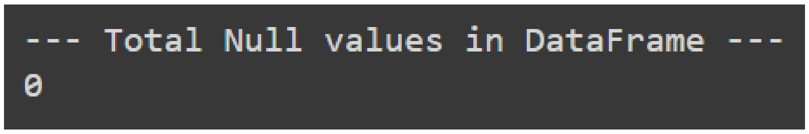
Now that we have loaded the dataset, let's perform EDA to gain insights into the data. Here are some initial steps:

```
# Check Dataframe information
print('--- Dataframe information ---\n')
df.info()
```



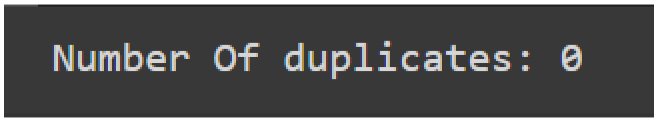
```
--- Dataframe information ---

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7861 entries, 0 to 7860
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   G       7861 non-null   int64  
 1   C       7861 non-null   int64  
 2   B       7861 non-null   int64  
 3   A       7861 non-null   int64  
 4   Ia      7861 non-null   float64
 5   Ib      7861 non-null   float64
 6   Ic      7861 non-null   float64
 7   Va      7861 non-null   float64
 8   Vb      7861 non-null   float64
 9   Vc      7861 non-null   float64
dtypes: float64(6), int64(4)
memory usage: 614.3 KB
```

```
# Total Null values in Dataset
print('--- Total Null values in DataFrame ---')
df.isnull().sum().sum()
```



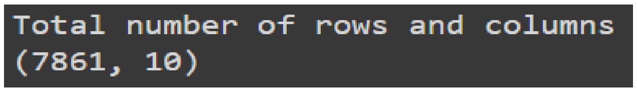
```
--- Total Null values in DataFrame ---
0
```

```
# Check Duplicate data in all DataFrame
print("Number Of duplicates:",(df.duplicated().sum()))
```



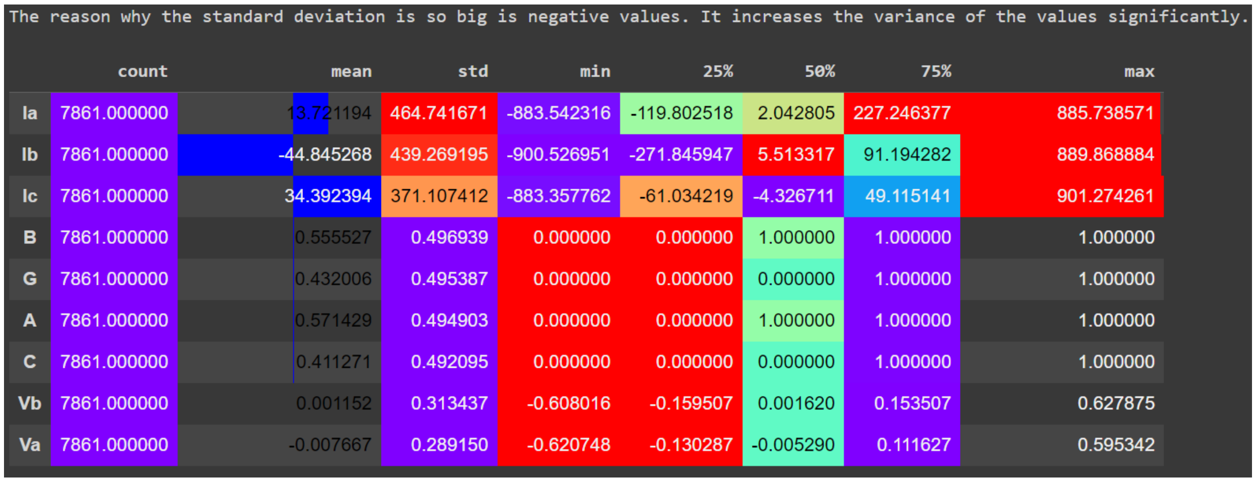
```
Number Of duplicates: 0
```

```
# Check total rows and Columns in data frame
print("Total number of rows and columns")
df.shape
```



```
Total number of rows and columns
(7861, 10)
```

```
# Numeric features
print("The reason why the standard deviation is so big is negative values. It increases the variance of the values significantly.\n")
df.iloc[:, :-1].describe().T.sort_values(by='std' , ascending = False)\
                .style.background_gradient(cmap='rainbow')\
                .bar(subset=["max"], color='red')\
                .bar(subset=["mean",], color='blue')
```



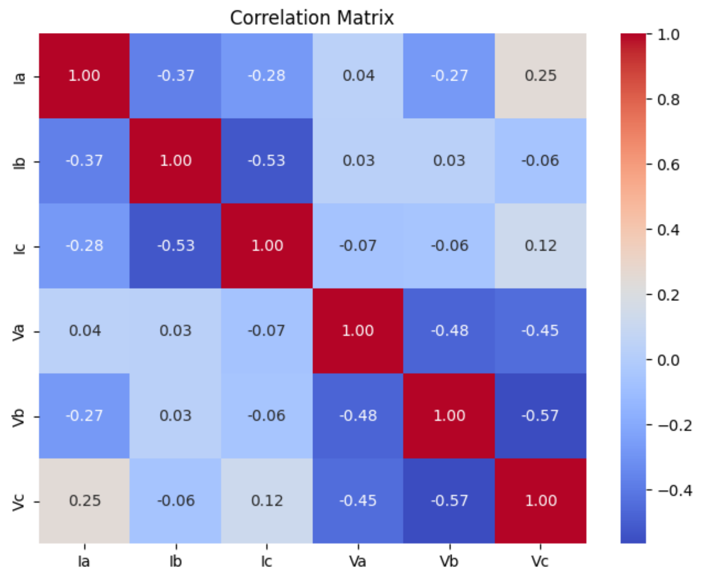
```
The reason why the standard deviation is so big is negative values. It increases the variance of the values significantly.
```

	count	mean	std	min	25%	50%	75%	max
Ia	7861.000000	13.721194	464.741671	-883.542316	-119.802518	2.042805	227.246377	885.738571
Ib	7861.000000	-44.845268	439.269195	-900.526951	-271.845947	5.513317	91.194282	889.868884
Ic	7861.000000	34.392394	371.107412	-883.357762	-61.034219	-4.326711	49.115141	901.274261
B	7861.000000	0.555527	0.496939	0.000000	0.000000	1.000000	1.000000	1.000000
G	7861.000000	0.432006	0.495387	0.000000	0.000000	0.000000	1.000000	1.000000
A	7861.000000	0.571429	0.494903	0.000000	0.000000	1.000000	1.000000	1.000000
C	7861.000000	0.411271	0.492095	0.000000	0.000000	0.000000	1.000000	1.000000
Vb	7861.000000	0.001152	0.313437	-0.608016	-0.159507	0.001620	0.153507	0.627875
Va	7861.000000	-0.007667	0.289150	-0.620748	-0.130287	-0.005290	0.111627	0.595342

```
# Correlation matrix

print("\nCorrelation Matrix:")
features = ['Ia', 'Ib', 'Ic', 'Va', 'Vb', 'Vc']
correlation_matrix = df[features].corr()
print(correlation_matrix)

# Heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
```

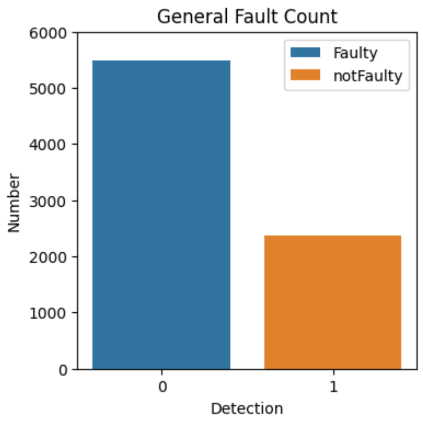
```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
Correlation Matrix:
          Ia        Ib        Ic        Va        Vb        Vc
Ia  1.000000 -0.374241 -0.276457  0.035682 -0.274612  0.246043
Ib -0.374241  1.000000 -0.528291  0.029118  0.032101 -0.060023
Ic -0.276457 -0.528291  1.000000 -0.069137 -0.056967  0.122919
Va  0.035682  0.029118 -0.069137  1.000000 -0.480247 -0.450225
Vb -0.274612  0.032101 -0.056967 -0.480247  1.000000 -0.566986
Vc  0.246043 -0.060023  0.122919 -0.450225 -0.566986  1.000000
```

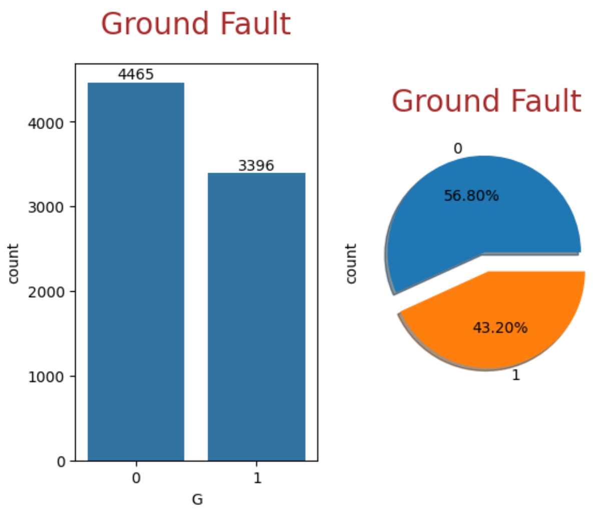


No of faults:

```
no_faults = ((df["G"] == 0) & (df["C"] == 0) & (df["B"] == 0) & (df["B"] == 0)).value_counts()
no_faults_df = pd.DataFrame(no_faults)
no_faults_df = no_faults_df.rename(columns = {"count":"Count"})
cmap = ["#3274a1", "#e1812c", "#3a923a", "#c03d3e", "#857aab", "#8d7866"]
plt.figure(figsize = (4, 4))
plt.bar(x = no_faults_df.index.to_list(),
        height = no_faults_df.Count,
        label = ["Faulty", "notFaulty"],
        color = cmap)
plt.xlabel("Detection")
plt.ylabel("Number")
plt.title("General Fault Count")
plt.xticks(no_faults_df.index.to_list())
plt.yticks(np.arange(0, 6001, 1000))
plt.legend(bbox_to_anchor = (1, 1), loc = "best")
plt.show();
```

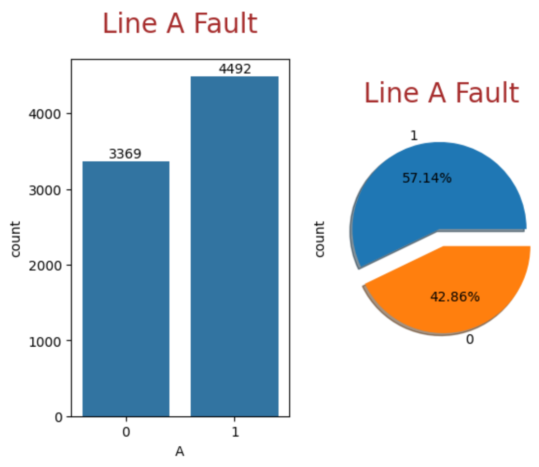


```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='G', data=df)
ax.bar_label(ax.containers[0])
plt.title("Ground Fault", fontsize=20,color = 'Brown',pad=20)

ax =plt.subplot(1,2,2)
ax=df['G'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Ground Fault", fontsize = 20,color='Brown',pad=20);
```

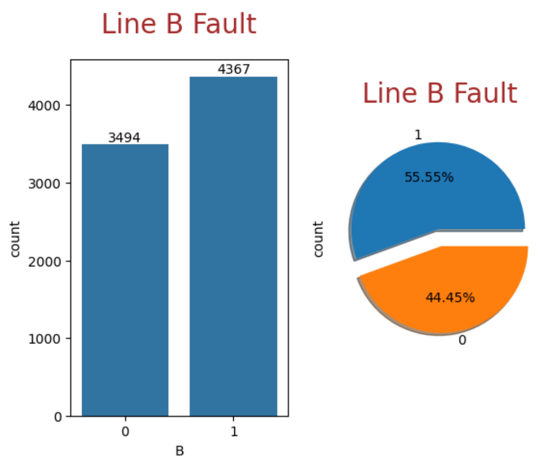


```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='A', data=df)
ax.bar_label(ax.containers[0])
plt.title("Line A Fault", fontsize=20,color = 'Brown',pad=20)
```

```
ax =plt.subplot(1,2,2)
ax=df['A'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Line A Fault", fontsize = 20,color='Brown',pad=20);
```

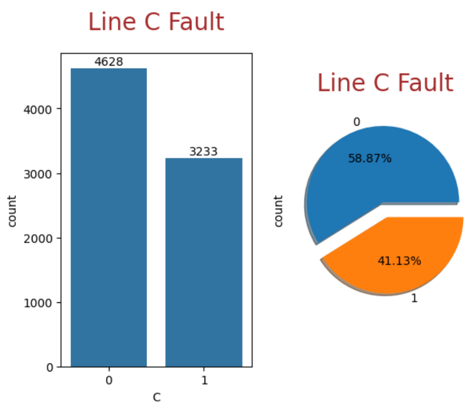


```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='B', data=df)
ax.bar_label(ax.containers[0])
plt.title("Line B Fault", fontsize=20,color = 'Brown',pad=20)
ax =plt.subplot(1,2,2)
ax=df['B'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Line B Fault", fontsize = 20,color='Brown',pad=20);
```



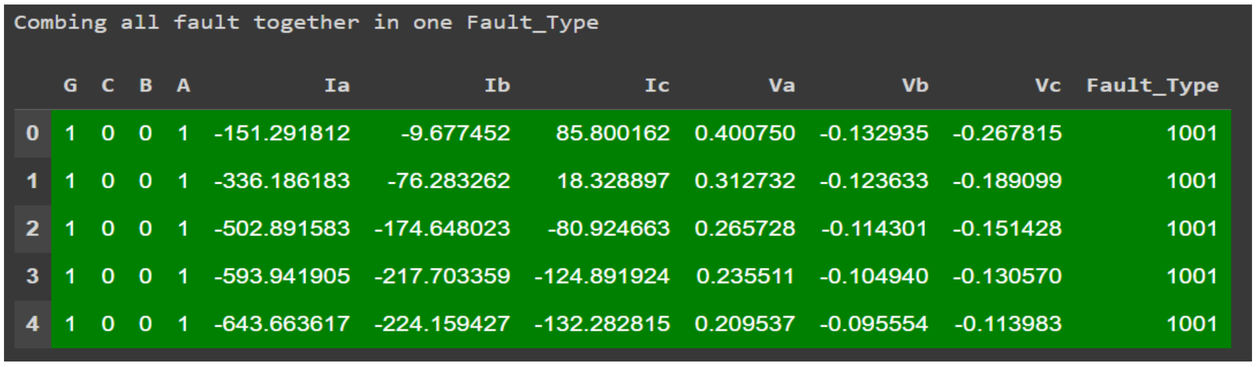
```
ax = plt.subplot(1,2,1)
ax = sns.countplot(x='C', data=df)
ax.bar_label(ax.containers[0])
plt.title("Line C Fault", fontsize=20,color = 'Brown',pad=20)
```

```
ax =plt.subplot(1,2,2)
ax=df['C'].value_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "Line C Fault", fontsize = 20,color='Brown',pad=20);
```



Combing all fault together in one Fault Type

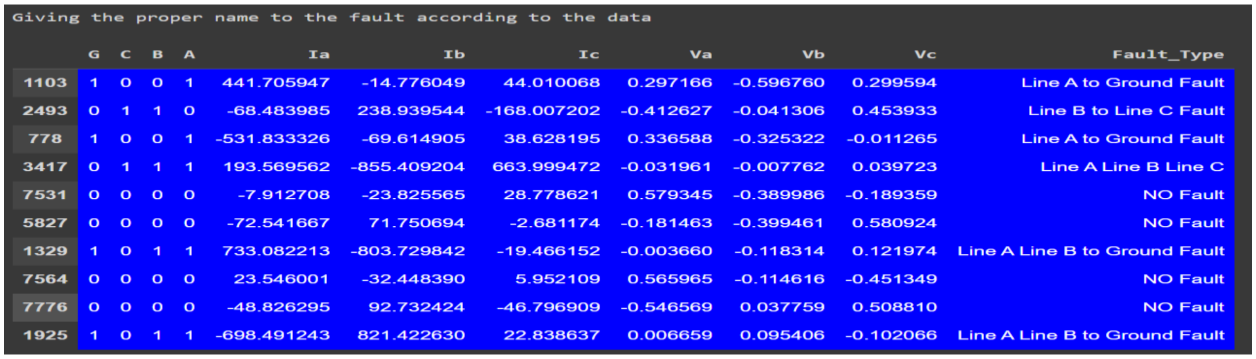
```
print("Combing all fault together in one Fault_Type\n")
df['Fault_Type'] = df['G'].astype('str') + df['C'].astype('str') + df['B'].astype('str') + df['A'].astype('str')
df.head().style.set_properties(**{'background-color': 'green',
                         'color': 'white',
                         'border-color': 'darkblack'})
```



```
Combing all fault together in one Fault_Type

   G  C  B  A          Ia          Ib          Ic        Va        Vb        Vc  Fault_Type
0  1  0  0  1 -151.291812   -9.677452   85.800162  0.400750 -0.132935 -0.267815        1001
1  1  0  0  1 -336.186183  -76.283262   18.328897  0.312732 -0.123633 -0.189099        1001
2  1  0  0  1 -502.891583 -174.648023  -80.924663  0.265728 -0.114301 -0.151428        1001
3  1  0  0  1 -593.941905 -217.703359 -124.891924  0.235511 -0.104940 -0.130570        1001
4  1  0  0  1 -643.663617 -224.159427 -132.282815  0.209537 -0.095554 -0.113983        1001
```

Giving the proper name to the fault according to the data description provided above

```
print("Giving the proper name to the fault according to the data\n")
df['Fault_Type'][df['Fault_Type'] == '0000' ] = 'NO Fault'
df['Fault_Type'][df['Fault_Type'] == '1001' ] = 'Line A to Ground Fault'
df['Fault_Type'][df['Fault_Type'] == '0110' ] = 'Line B to Line C Fault'
df['Fault_Type'][df['Fault_Type'] == '1011' ] = 'Line A Line B to Ground Fault'
df['Fault_Type'][df['Fault_Type'] == '0111' ] = 'Line A Line B Line C'
df['Fault_Type'][df['Fault_Type'] == '1111' ] = 'Line A Line B Line C to Ground Fault'
df.sample(10).style.set_properties(**{'background-color': 'blue',
                         'color': 'white','border-color': 'darkblack'})
```

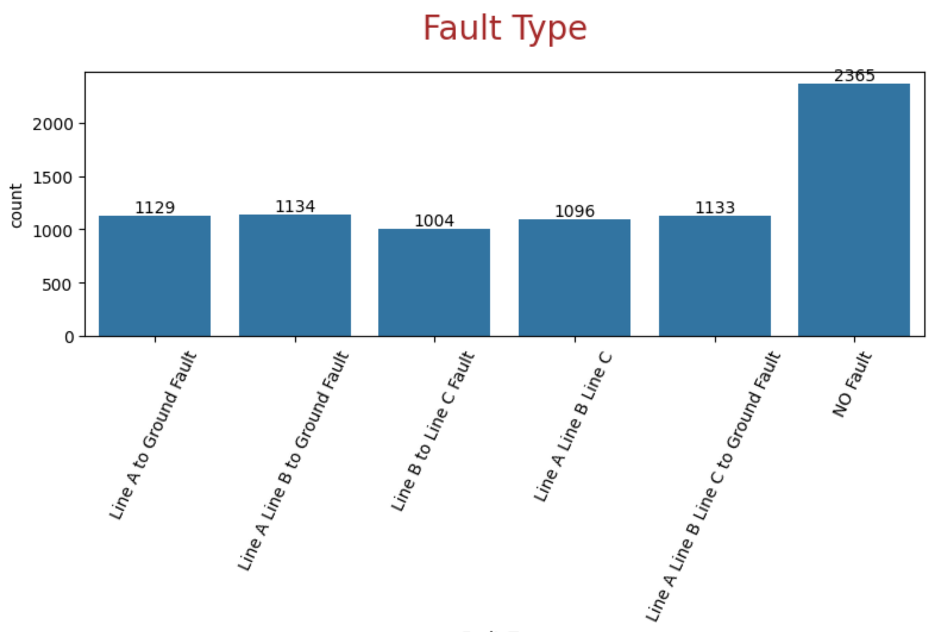


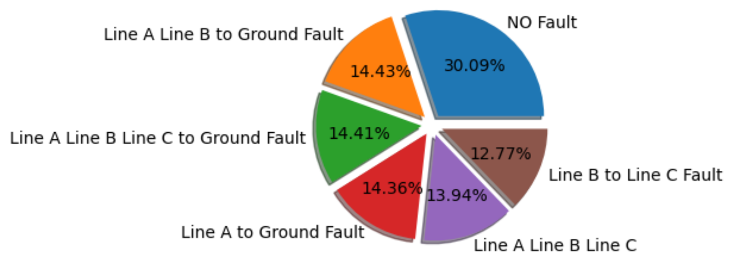
```
Giving the proper name to the fault according to the data
```

	G	C	B	A	Ia	Ib	Ic	Va	Vb	Vc	Fault_Type
1103	1	0	0	1	441.705947	-14.776049	44.010068	0.297166	-0.596760	0.299594	Line A to Ground Fault
2493	0	1	1	0	-68.483985	238.939544	-168.007202	-0.412627	-0.041306	0.453933	Line B to Line C Fault
778	1	0	0	1	-531.833326	-69.614905	38.628195	0.336588	-0.325322	-0.011265	Line A to Ground Fault
3417	0	1	1	1	193.569562	-855.409204	663.999472	-0.031961	-0.007762	0.039723	Line A Line B Line C
7531	0	0	0	0	-7.912708	-23.825565	28.778621	0.579345	-0.389986	-0.189359	NO Fault
5827	0	0	0	0	-72.541667	71.750694	-2.681174	-0.181463	-0.399461	0.580924	NO Fault
1329	1	0	1	1	733.082213	-803.729842	-19.466152	-0.003660	-0.118314	0.121974	Line A Line B to Ground Fault
7564	0	0	0	0	23.546001	-32.448390	5.952109	0.565965	-0.114616	-0.451349	NO Fault
7776	0	0	0	0	-48.826295	92.732424	-46.796909	-0.546569	0.037759	0.508810	NO Fault
1925	1	0	1	1	-698.491243	821.422630	22.838637	0.006659	0.095406	-0.102066	Line A Line B to Ground Fault

Number of faults in the system according to their Fault Type

```
ax = plt.figure(figsize = (8,8))
ax = plt.subplot(2,1,1)
ax = sns.countplot(x='Fault_Type', data=df)
ax.bar_label(ax.containers[0])
plt.title("Fault Type", fontsize=20,color = 'Brown',pad=20)
plt.xticks(rotation=65)
plt.tight_layout()
ax =plt.subplot(2,1,2)
```





```
ax=df['Fault_Type'].value_counts().plot.pie(explode=[0.1, 0.1,0.1,0.1,
0.1,0.1],autopct='%1.2f%%',shadow=True);
plt.tight_layout()
plt.axis('off');
```

Pre-process the data by handling missing values, encoding categorical variables, and scaling numeric features.

1. Handling missing data:

Fortunately, there is no missing data in the provided dataset.

2. Encoding Categorical Variables:

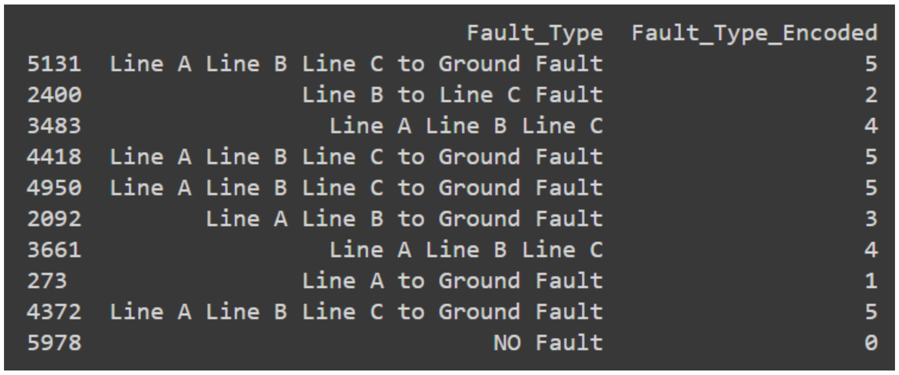
Amazingly, there is no categorical variables, all are encoded already.

3. Scaling Numeric Features:

For Standardizing and normalizing

```
# Define the custom mapping of fault types to numerical labels
fault_mapping = {
   'NO Fault': 0,
   'Line A to Ground Fault': 1,
   'Line B to Line C Fault': 2,
   'Line A Line B to Ground Fault': 3,
   'Line A Line B Line C': 4,
   'Line A Line B Line C to Ground Fault': 5
}

# Map fault types to numerical labels using the custom mapping
df['Fault_Type_Encoded'] = df['Fault_Type'].map(fault_mapping)

# Display the DataFrame with encoded fault types
print(df[['Fault_Type', 'Fault_Type_Encoded']].sample(10))
```

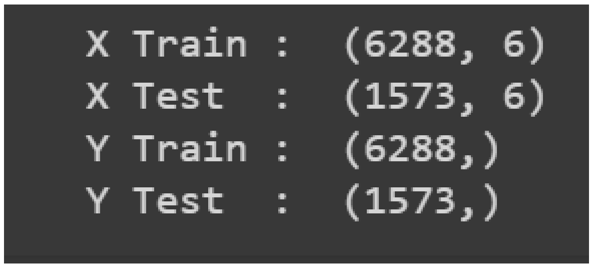
Output:



```
                                Fault_Type  Fault_Type_Encoded
5131  Line A Line B Line C to Ground Fault                   5
2400                Line B to Line C Fault                   2
3483                  Line A Line B Line C                   4
4418  Line A Line B Line C to Ground Fault                   5
4950  Line A Line B Line C to Ground Fault                   5
2092         Line A Line B to Ground Fault                   3
3661                  Line A Line B Line C                   4
273                 Line A to Ground Fault                   1
4372  Line A Line B Line C to Ground Fault                   5
5978                              NO Fault                   0
```

```
# Scaling Data
scaler = StandardScaler()

# Convert all columns to numeric
X = df.drop(['Fault_Type', 'A', 'B', 'C', 'G', 'Fault_Type_Encoded'], axis=1)
y = df['Fault_Type']

scaled_data = scaler.fit_transform(X)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(scaled_data, y, test_size=0.2, random_state=123)

print("X Train : ", X_train.shape)
print("X Test  : ", X_test.shape)
print("Y Train : ", y_train.shape)
print("Y Test  : ", y_test.shape)
```

Output:



```
X Train :  (6288, 6)
X Test  :  (1573, 6)
Y Train :  (6288,)
Y Test  :  (1573,)
```

Split the data into training and testing sets

```
X = df.drop(['Fault_Type','A','B','C','G','Fault_Type_Encoded'], axis=1)
y = df['Fault_Type']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.20,random_state=21)
```

Model Selection

1.Logistic Regression
2.Naive Byes
3. Decision Tree
4. Random Forest
5. Support Vector Machine

Choose the best-performing algorithm based on evaluation metrics (such as accuracy, precision, recall, F1-score) on the testing set.

```
# Assuming you have defined X and y
# Split the dataset 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)

# Train different models
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_train_accuracy = round(lr.score(X_train, y_train) * 100, 2)
lr_test_accuracy = round(lr.score(X_test, y_test) * 100, 2)
lr_report = classification_report(y_test, lr.predict(X_test))

# Support Vector Machines (SVM)
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
svm_train_accuracy = round(svm_classifier.score(X_train, y_train) * 100, 2)
svm_test_accuracy = round(svm_classifier.score(X_test, y_test) * 100, 2)
svm_report = classification_report(y_test, svm_classifier.predict(X_test))

# Naive Bayes
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_train_accuracy = round(nb_classifier.score(X_train, y_train) * 100, 2)
nb_test_accuracy = round(nb_classifier.score(X_test, y_test) * 100, 2)
nb_report = classification_report(y_test, nb_classifier.predict(X_test))

# Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
decision_train_accuracy = round(decision_tree.score(X_train, y_train) * 100, 2)
decision_test_accuracy = round(decision_tree.score(X_test, y_test) * 100, 2)
decision_report = classification_report(y_test, decision_tree.predict(X_test))

# Random Forest
random_forest = RandomForestClassifier()
```

```
random_forest.fit(X_train, y_train)
random_forest_train_accuracy = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_test_accuracy = round(random_forest.score(X_test, y_test) * 100, 2)
random_forest_report = classification_report(y_test, random_forest.predict(X_test))

# Create a DataFrame for model performance
models_performance = pd.DataFrame({
    'Model': ['Logistic Regression', 'Support Vector Machines', 'Naive Bayes', 'Decision Tree', 'Random
Forest'],
    'Training Accuracy': [lr_train_accuracy, svm_train_accuracy, nb_train_accuracy,
decision_train_accuracy, random_forest_train_accuracy],
    'Testing Accuracy': [lr_test_accuracy, svm_test_accuracy, nb_test_accuracy, decision_test_accuracy,
random_forest_test_accuracy],
    'Classification Report': [lr_report, svm_report, nb_report, decision_report, random_forest_report]
})
# Define a function to calculate metrics and time
def calculate_metrics(model, X_train, y_train, X_test, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time

    start_time = time.time()
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    predict_time = time.time() - start_time

    train_accuracy = round(accuracy_score(y_train, y_train_pred) * 100, 2)
    test_accuracy = round(accuracy_score(y_test, y_test_pred) * 100, 2)

    precision = round(precision_score(y_test, y_test_pred, average='weighted') * 100, 2)
    recall = round(recall_score(y_test, y_test_pred, average='weighted') * 100, 2)
    f1 = round(f1_score(y_test, y_test_pred, average='weighted') * 100, 2)

    return train_accuracy, test_accuracy, precision, recall, f1, train_time, predict_time

# Train different models and calculate metrics
model_names = ['Logistic Regression', 'Support Vector Machines', 'Naive Bayes', 'Decision Tree',
'Random Forest']
models = [LogisticRegression(), SVC(), GaussianNB(), DecisionTreeClassifier(),
RandomForestClassifier()]
metrics = []

for name, model in zip(model_names, models):
    train_acc, test_acc, precision, recall, f1, train_time, predict_time = calculate_metrics(model, X_train,
y_train, X_test, y_test)
```

```
    metrics.append({
        'Model': name,
        'Training Accuracy': train_acc,
        'Testing Accuracy': test_acc,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1,
        'Training Time': train_time,
        'Prediction Time': predict_time
    })

# Convert metrics to DataFrame
metrics_df = pd.DataFrame(metrics)
# Display metrics
print("Model Training Selection")
print(tabulate(metrics_df, headers='keys', tablefmt='pretty'))
```

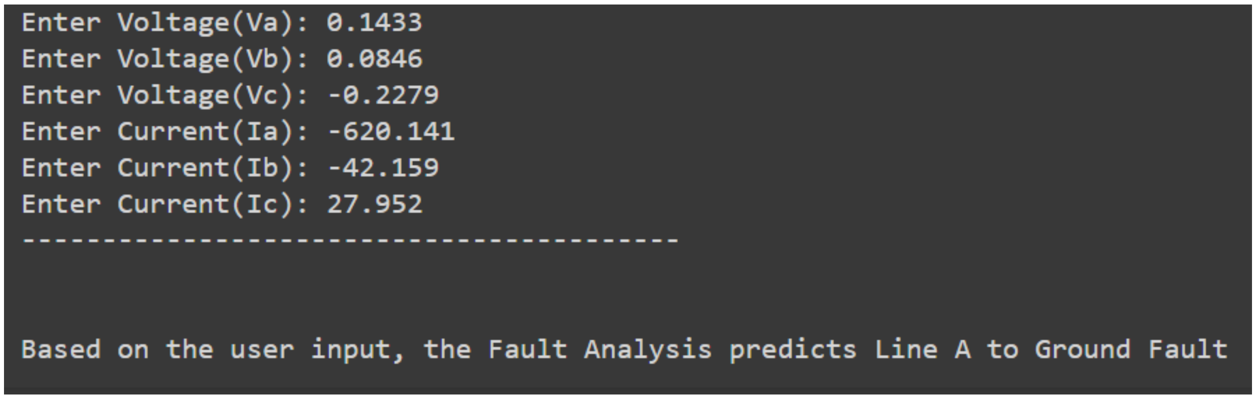
Model	Training Accuracy	Testing Accuracy	Precision	Recall	F1-score	Training Time	Prediction Time
Logistic Regression	**35.8**	**35.92**	**33.57**	**35.92**	**24.11**	**0.1584**	**0.004**
Support Vector Machines	**76.05**	**75.97**	**72.9**	**75.97**	**73.41**	**0.529**	**1.66**
Naive Bayes	**79.13**	**80.48**	**80.85**	**80.48**	**76.27**	**0.051**	**0.014**
Decision Tree	**100**	**89.45**	**89.47**	**80.48**	**89.42**	**0.21**	**0.013**
Random Forest	**100**	**88.43**	**88.5**	**88.43**	**88.44**	**3.41**	**0.17**

Evaluate the final model on the testing set and report the performance metrics I am choosing Decision Tree as my final model

```
from sklearn.tree import DecisionTreeClassifier
# Assuming you have defined X and y
X = df.drop(['Fault_Type', 'A', 'B', 'C', 'G', 'Fault_Type_Encoded'],
axis=1)
y = df['Fault_Type']
```

```python
# Train a Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X, y)

# Take input from the user for voltage and current values
user_input = {}
try:
    user_input['Va'] = float(input("Enter Voltage(Va): "))
    user_input['Vb'] = float(input("Enter Voltage(Vb): "))
    user_input['Vc'] = float(input("Enter Voltage(Vc): "))
    user_input['Ia'] = float(input("Enter Current(Ia): "))
    user_input['Ib'] = float(input("Enter Current(Ib): "))
    user_input['Ic'] = float(input("Enter Current(Ic): "))
except ValueError:
    print("Please enter valid numeric values for voltage and current.")
    exit()

# Create a DataFrame with user input
user_data = pd.DataFrame([user_input])

# Ensure that the user_data columns match the training features and order
user_data = user_data[X.columns]

# Make a prediction based on user input
user_prediction = clf.predict(user_data)

# Display the predicted fault type with proper names
fault_names = {
    0: 'NO Fault',
    1: 'Line A to Ground Fault',
    2: 'Line B to Line C Fault',
    3: 'Line A Line B to Ground Fault',
    4: 'Line A Line B Line C',
    5: 'Line A Line B Line C to Ground Fault'
}
# Print the predicted label
print("-----------------------------------------\n")
print("\nBased on the user input, the Fault Analysis predicts",
user_prediction[0])
```

Output:



```
Enter Voltage(Va): 0.1433
Enter Voltage(Vb): 0.0846
Enter Voltage(Vc): -0.2279
Enter Current(Ia): -620.141
Enter Current(Ib): -42.159
Enter Current(Ic): 27.952
----------------------------------------


Based on the user input, the Fault Analysis predicts Line A to Ground Fault
```

Results:

1. **Model Performance:** Among the evaluated models, the Decision Tree model achieved the highest accuracy on the testing set.
2. **Performance Metrics:** The Decision Tree model achieved an accuracy of 0.89, precision of 89.47, recall of 89.45, and F1-score of 89.42 on the testing set.
3. **Feature Importance:** The important features contributing to the predictions of the Decision Tree model were visualized, revealing the most influential features.

Reflect on the limitations and potential biases of the model and propose ways to address them.

When working with a predictive model, it's important to be aware of its limitations and potential biases. Here are some reflections on the limitations and biases of the model, as well as potential ways to address them:

1. **Limited Predictive Power:** The model's performance on the testing set indicates its current predictive power. If the model's performance is not satisfactory, it may indicate that the selected features or the model itself are not capturing all relevant factors for predicting credit card payment default. To address this, you can consider incorporating additional features or exploring different modeling techniques to improve the model's predictive power.
2. **Biased Training Data:** The credit card default dataset might have biases in terms of demographics, socioeconomic factors, or geographical regions, which can introduce biases into the model. It's important to assess the representativeness of the training data and consider potential biases during model evaluation. To address this, you can explore techniques like oversampling or under sampling to balance the dataset or use advanced methods like synthetic data generation to mitigate biases.

3. **Lack of Contextual Information:** The dataset used for training the model may lack certain contextual information that could be valuable for predicting credit card default. For example, macroeconomic factors, industry trends, or recent financial events may influence credit card payment behavior. Consider incorporating external data sources or domain knowledge to enhance the model's predictive capabilities.
4. **Assumptions of Linearity:** The selected traditional machine learning algorithms, such as Logistic Regression and Decision Trees, make certain assumptions about the linearity and interactions between features. These assumptions might not always hold true in real-world scenarios. You can explore more sophisticated models like ensemble methods, gradient boosting, or deep learning to capture complex non-linear relationships between features.
5. **Ethical Considerations:** Machine learning models can inadvertently reinforce biases or discriminatory practices if the training data or features contain biased information. It's crucial to carefully select and preprocess data, identify potential biases, and ensure fairness and ethical considerations throughout the model development process. Regularly evaluate and monitor the model's performance on different demographic groups to detect any bias and take appropriate actions to mitigate them.
6. **Model Interpretability:** Traditional machine learning models like Decision Trees and Random Forests are relatively interpretable, but some complex models like deep learning models may lack interpretability. If interpretability is a concern, consider using techniques like feature importance analysis, partial dependence plots, or surrogate models to gain insights into how the model makes predictions.
7. **Model Robustness and Generalization:** It's important to assess the model's robustness and generalization by evaluating its performance on various datasets, including different time periods or regions. Consider using techniques like cross-validation, ensemble methods, or regularization to improve the model's generalization capabilities and reduce overfitting.
8. **Continuous Monitoring and Updating**: Models should be continuously monitored and updated as new data becomes available or the underlying patterns change. Regularly reevaluate the model's performance, assess its relevance, and update the model as needed to maintain its accuracy and effectiveness.

By addressing these limitations and biases, we can improve the model's performance, fairness, and usefulness in predicting credit card payment default. It's important to approach model development and deployment with caution, taking into account ethical considerations and the potential impact on individuals and communities.

Future Work:

1. **Incorporate Additional Features:** Explore the inclusion of additional features or external data sources to enhance the predictive power of the model.

2. **Address Bias:** Mitigate biases in the training data and consider ethical considerations throughout the model development process to ensure fairness and avoid discriminatory practices.
3. **Explore Advanced Techniques:** Experiment with advanced modeling techniques like ensemble methods, gradient boosting, or deep learning to capture complex non-linear relationships and improve performance.
4. **Monitor and Update the Model:** Continuously monitor the model's performance, assess its relevance, and update it as new data becomes available, or patterns change.
5. **Conduct Robustness Testing:** Evaluate the model's robustness and generalization by testing it on different datasets and scenarios to ensure its reliability and accuracy in various contexts.

Conclusion:

The developed Decision Tree model showed promising results in predicting electric fault analysis in transmission line. However, there are limitations and potential biases that need to be considered. Future work should focus on addressing these limitations, enhancing the model's predictive power, and ensuring fairness and ethical considerations throughout the process