DATA MINING AND MACHINE LEARNING

FINAL PROJECT

TEAM MEMBERS

- YASHWANTH HOLUR NARAYANASWAMY
- RAJENDRA LEENA SAI
- SAICHAND MEKALA
- POLA SUDHARSHAN
- JYOTHSNA YALAMADDI

INTRODUCTION

- WE HAVE TAKEN A CARDIOVASCULAR DISEASE PREDICTION DATASET. IN THIS DATASET THERE ARE MANY FEATURES RELATED TO PATIENTS' HEALTH AND LIFESTYLE, INCLUDING AGE, SEX, GENERAL HEALTH, CHECK-UP FREQUENCY, EXERCISE HABITS, SMOKING HISTORY, AND THE PRESENCE OF VARIOUS DISEASES. EACH ENTRY REPRESENTS A UNIQUE PATIENT, AND THE FEATURES CAPTURE VARIOUS FACTORS ASSOCIATED WITH DISEASE PROGNOSIS.

OBJECTIVE

- OUR MAIN OBJECTIVE IS TO DEVELOP A PREDICTIVE MODEL THAT CAN EFFECTIVELY FORECAST THE PROGNOSIS OF VARIOUS DISEASES BASED ON THE PROVIDED FEATURES. BY LEVERAGING THE POWER OF MACHINE LEARNING, WE AIM TO ENHANCE THE MODEL'S ACCURACY AND PREDICTIVE PERFORMANCE.

COUNT OF ROWS AND COLUMNS IN DATA

```
In [2]: rawdata = pd.read_csv('CVD_cleaned.csv')
        #Number of rows and columns present in the dataset
        rawdata.shape

Out[2]: (308854, 19)
```

Data Exploration

```
In [3]: rawdata.head()
```

Out[3]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diabetes	Arthritis	Sex	Age_Category	Height_(cm)	Weight_(kg)	BMI	Smoking_History	Alcohol_Consu
0	Poor	Within the past 2 years	No	No	No	No	No	No	Yes	Female	70-74	150.0	32.66	14.54	Yes	
1	Very Good	Within the past year	No	Yes	No	No	No	Yes	No	Female	70-74	165.0	77.11	28.29	No	
2	Very Good	Within the past year	Yes	No	No	No	No	Yes	No	Female	60-64	163.0	88.45	33.47	No	
3	Poor	Within the past year	Yes	Yes	No	No	No	Yes	No	Male	75-79	180.0	93.44	28.73	No	
4	Good	Within the past year	No	No	No	No	No	No	No	Male	80+	191.0	88.45	24.37	Yes	

- OUR DATA IS TAKEN FROM KAGGLE
- THE DATA SET HAS DATA ABOUT CARDIO VASCULAR DISEASE
- IT CONTAINS 308854 ROWS AND 19 COLUMNS
- 11 COLUMNS ARE CATEGORICAL COLUMNS
- 9 ARE NUMERICAL COLUMNS

MISSING VALUES

Missing Values

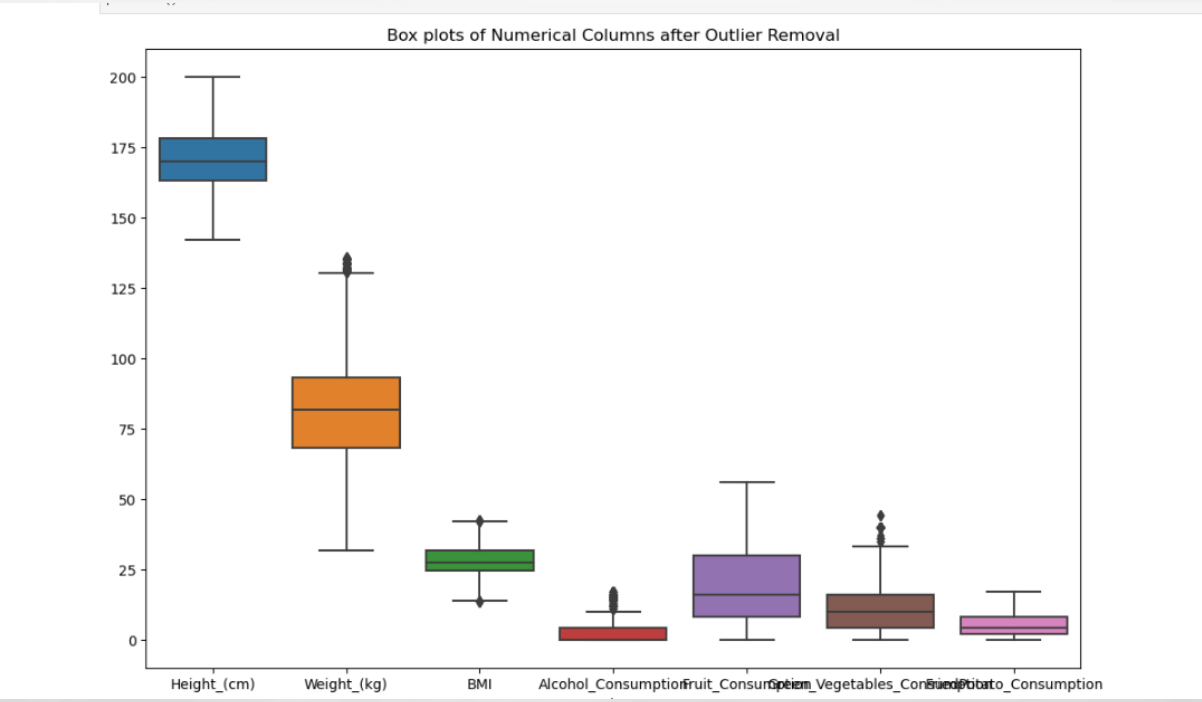
```
# Check for missing values in the entire DataFrame
missing_values = rawdata.isnull().sum()

# Check if there are any missing values in the DataFrame
if missing_values.sum() > 0:
    print("Missing values are present.")
    print("Columns with missing values:")
    print(missing_values[missing_values > 0])
else:
    print("No missing values are present.")
```

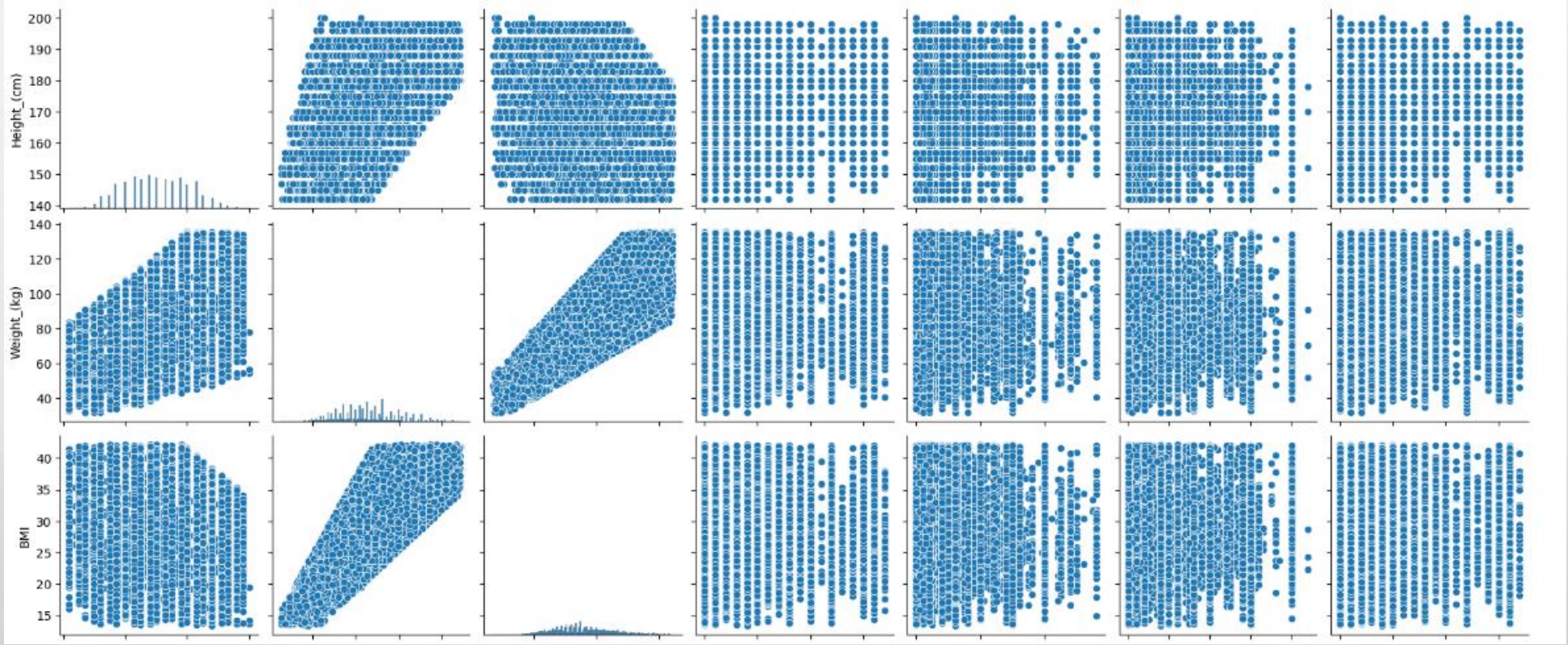
```
No missing values are present.
```

- OUR DATA IS CLEANED AND THERE ARE NO MISSING VALUES IN THE DATA

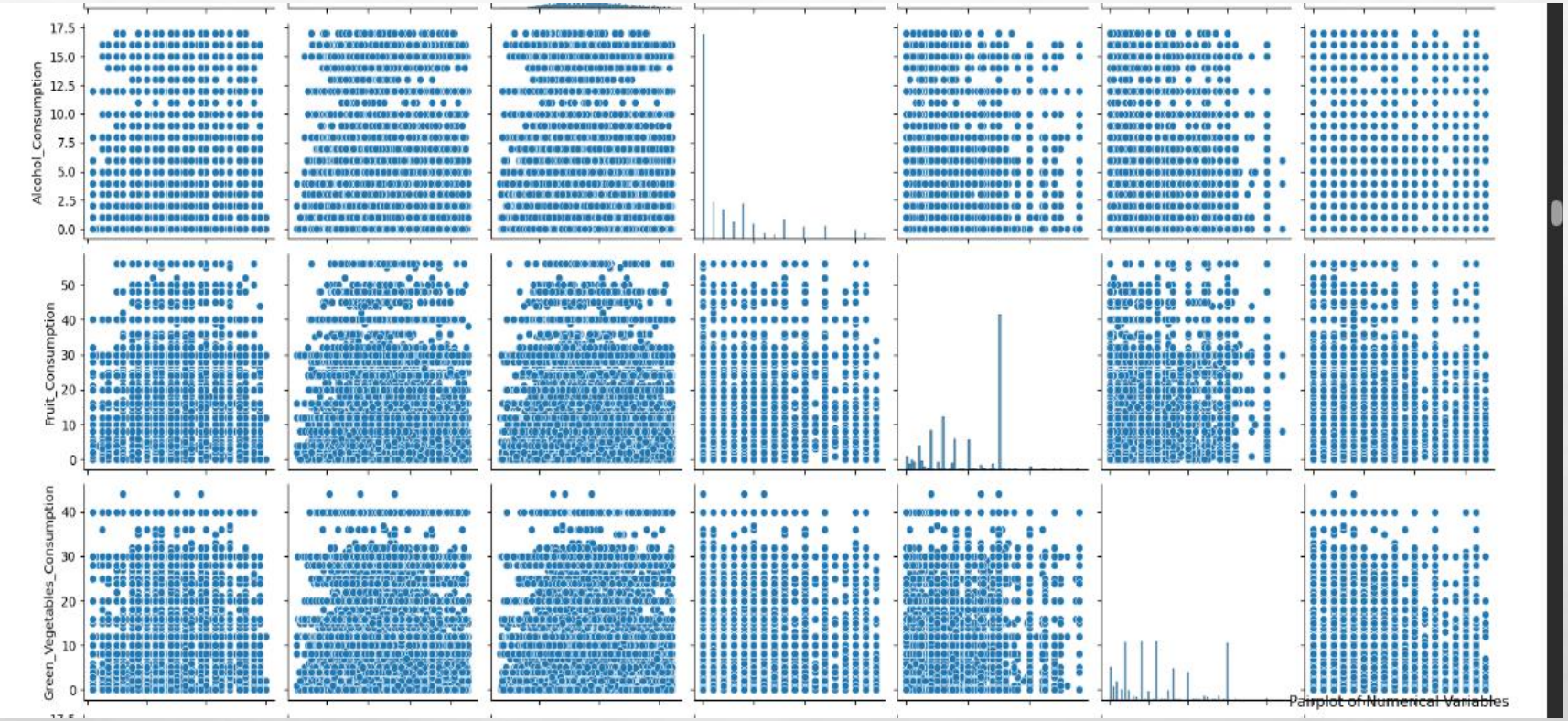
HANDLING OUTLIERS



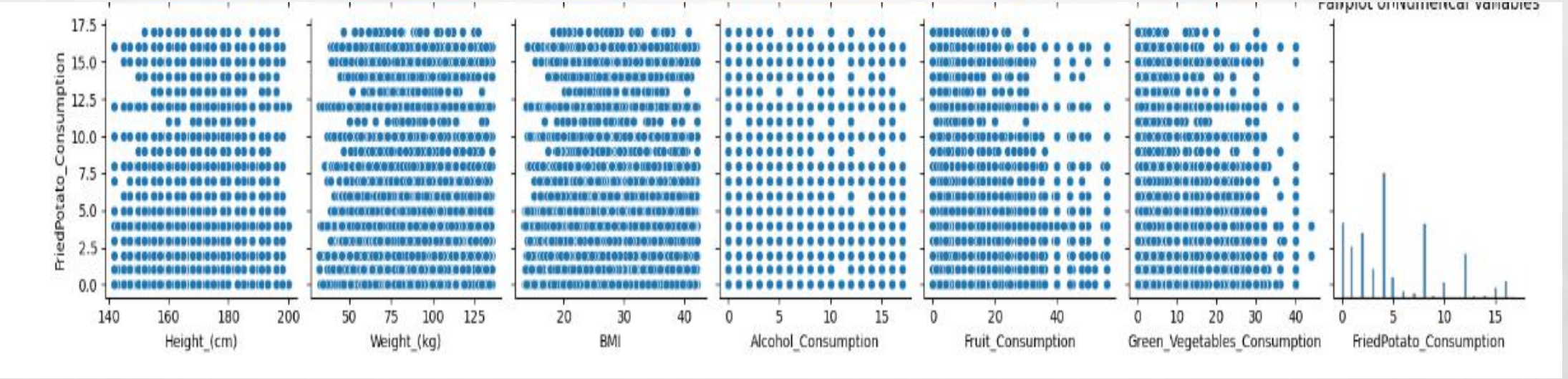
CORRELATION BETWEEN NUMERICAL VALUES



CORRELATION BETWEEN NUMERICAL VALUES



CORRELATION BETWEEN NUMERICAL VALUES



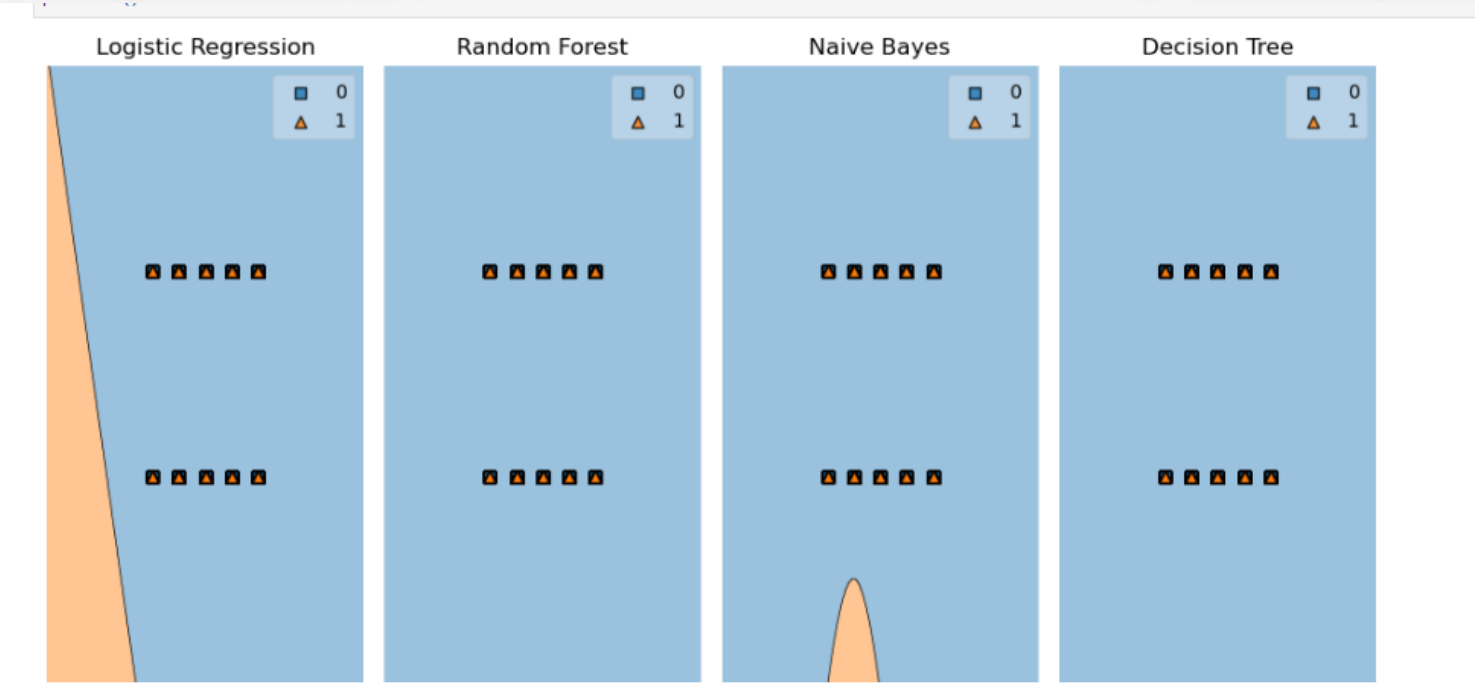
MODEL SELECTION

```
5-fold cross validation:

Accuracy: 0.92 (+/- 0.00) [Logistic Regression]
Accuracy: 0.91 (+/- 0.00) [Random Forest]
Accuracy: 0.57 (+/- 0.00) [Naive Bayes]
Accuracy: 0.85 (+/- 0.00) [Decision Tree]
```

- WE USED 4 DIFFERENT CLASSIFIERS WHICH ARE LOGISTIC REGRESSION, RANDOM FOREST, NAIVE BAYES, DECISION TREE
- WERE THE ACCURACIES OF LOGISTIC REGRESSION WAS 0.92
- RANDOM FOREST WAS 0.91
- NAIVE BAYES WAS 0.57
- DECISION TREE' S ACCURACY WAS 0.85

DECISION REGIONS GRAPH FOR MODELS



VOTING CLASSIFIER

- VOTING CLASSIFIER IS AN ENSEMBLE METHOD IN MACHINE LEARNING CLASSIFIERS
- WHERE VOTING CLASSIFIER USES A VOTING RULE I.E. HARD OR SOFT TO PREDICT THE CLASS LABELS
- IT BALANCES OUT THE INDIVIDUAL WEAKNESSES OF MODEL WHEN THE ACCURACY IS SAME.
- HERE WE HAVE COMBINED ALL THE INDIVIDUAL CLASSIFIERS USED TO BUILD A ENSEMBLE.

OUTPUT OF VOTING CLASSIFIER

```
Accuracy: 0.92 (+/- 0.00) [Logistic Regression]
Accuracy: 0.91 (+/- 0.00) [Random Forest]
Accuracy: 0.85 (+/- 0.00) [Voting_Classifier_Hard]
Accuracy: 0.91 (+/- 0.00) [Voting_Classifier_Soft]
```

HYPER PARAMETER TUNING USING LINEAR REGRESSION

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate, StratifiedKFold
import numpy as np

# Define hyperparameter values to tune
penalty_values = ['l1', 'l2']
C_values = [0.001, 0.01, 0.1, 1, 10, 100]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for penalty in penalty_values:
    for C in C_values:
        clf = LogisticRegression(penalty=penalty, C=C, solver='liblinear')
        scores = cross_validate(clf, X_train, y_train,
                                scoring=['accuracy', 'precision', 'recall', 'roc_auc'],
                                cv=kk, n_jobs=None, return_estimator=True)

        print('Penalty:', penalty, 'C:', C)
        print('Accuracy:', np.mean(scores['test_accuracy']),
              'Precision:', np.mean(scores['test_precision']),
              'Recall:', np.mean(scores['test_recall']),
              'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
Penalty: l1 C: 0.001
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.7465363703288904
Penalty: l1 C: 0.01
Accuracy: 0.9156524618976822 Precision: 0.5286461736197093 Recall: 0.037878700899840696 ROC-AUC: 0.8235450798761599
Penalty: l1 C: 0.1
Accuracy: 0.9155591664720202 Precision: 0.5141492279639289 Recall: 0.057877193989580786 ROC-AUC: 0.8298239961180107
Penalty: l1 C: 1
Accuracy: 0.9153080054929461 Precision: 0.500419564304617 Recall: 0.061012571937025493 ROC-AUC: 0.8299550875795442
Penalty: l1 C: 10
Accuracy: 0.9153797662140569 Precision: 0.5035970714275612 Recall: 0.06177528380143228 ROC-AUC: 0.8299549194622508
Penalty: l1 C: 100
Accuracy: 0.9153654143787973 Precision: 0.5030230003099226 Recall: 0.061605864033639976 ROC-AUC: 0.8299436504291725
Penalty: l2 C: 0.001
Accuracy: 0.9155017529517272 Precision: 0.5682094557190465 Recall: 0.007287848562694643 ROC-AUC: 0.8040694757044546
Penalty: l2 C: 0.01
Accuracy: 0.9155806955122541 Precision: 0.5173819694747368 Recall: 0.0421156302472768 ROC-AUC: 0.8226511331967613
Penalty: l2 C: 0.1
Accuracy: 0.9156022245524881 Precision: 0.5169972191763289 Recall: 0.05567387591670374 ROC-AUC: 0.8297430077164032
Penalty: l2 C: 1
Accuracy: 0.9153295335033041 Precision: 0.5013825709955108 Recall: 0.06084308041160176 ROC-AUC: 0.8299673934268146
Penalty: l2 C: 10
Accuracy: 0.9153438863684393 Precision: 0.501884712032429 Recall: 0.06177528380143227 ROC-AUC: 0.8299504134249247
Penalty: l2 C: 100
Accuracy: 0.9153582387186369 Precision: 0.5025097120324291 Recall: 0.06194477532685601 ROC-AUC: 0.8299480941404911
```

HYPER PARAMETER TUNING USING RANDOM FOREST

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
import numpy as np

# Define hyperparameter values to tune
n_estimators_values = [100, 200, 300]  # Example values for number of trees
max_depth_values = [None, 5, 10, 15, 20]
min_samples_leaf_values = [1, 5, 10, 15, 20]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for n_estimators in n_estimators_values:
    for max_depth in max_depth_values:
        for min_samples_leaf in min_samples_leaf_values:
            clf = RandomForestClassifier(n_estimators=n_estimators,
                                         max_depth=max_depth,
                                         min_samples_leaf=min_samples_leaf,
                                         random_state=42)
            scores = cross_validate(clf, X_train, y_train,
                                    scoring=['accuracy', 'precision', 'recall', 'roc_auc'],
                                    cv=kk, n_jobs=None, return_estimator=True)

            print('n_estimators:', n_estimators, 'Max Depth:', max_depth, 'Min Samples Leaf:', min_samples_leaf)
            print('Accuracy:', np.mean(scores['test_accuracy']),
                  'Precision:', np.mean(scores['test_precision']),
                  'Recall:', np.mean(scores['test_recall']),
                  'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
n_estimators: 100 Max Depth: None Min Samples Leaf: 1
Accuracy: 0.9146693377527366 Precision: 0.4573869036828272 Recall: 0.0362686031659467 ROC-AUC: 0.8030296004121675
n_estimators: 100 Max Depth: None Min Samples Leaf: 5
Accuracy: 0.9153941170194402 Precision: 0.5204425039128384 Recall: 0.014660227615206878 ROC-AUC: 0.8199992044667382
n_estimators: 100 Max Depth: None Min Samples Leaf: 10
Accuracy: 0.9154586984758248 Precision: 0.5693256692455777 Recall: 0.008304725957605591 ROC-AUC: 0.8234563791543309
n_estimators: 100 Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9153582351140711 Precision: 0.5950840825840826 Recall: 0.005169276252529456 ROC-AUC: 0.8241567954861366
n_estimators: 100 Max Depth: None Min Samples Leaf: 20
Accuracy: 0.9152936490232451 Precision: 0.5378205128205129 Recall: 0.002881284174571966 ROC-AUC: 0.8249901502205572
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.8028879862389491
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.8028064643352071
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.8029293927539871
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.8028671788975711
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.8029095526825151
```

HYPER PARAMETER TUNING USING DECISION TREE

```
max_depth_values = [None, 5, 10, 15, 20]
min_samples_leaf_values = [1, 5, 10, 15, 20]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for max_depth in max_depth_values:
    for min_samples_leaf in min_samples_leaf_values:
        clf = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_samples_leaf)
        scores = cross_validate(clf, X_train, y_train,
                                scoring=['accuracy', 'precision', 'recall', 'roc_auc'],
                                cv=kk, n_jobs=None, return_estimator=True)

        print('Max Depth:', max_depth, 'Min Samples Leaf:', min_samples_leaf)
        print('Accuracy:', np.mean(scores['test_accuracy']),
              'Precision:', np.mean(scores['test_precision']),
              'Recall:', np.mean(scores['test_recall']),
              'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
Max Depth: None Min Samples Leaf: 1
Accuracy: 0.8533856870734511 Precision: 0.19172393174784677 Recall: 0.22726940685141864 ROC-AUC: 0.5693244410450394
Max Depth: None Min Samples Leaf: 5
Accuracy: 0.8847594549814477 Precision: 0.24630016100731403 Recall: 0.17464759827207624 ROC-AUC: 0.633330180615004
Max Depth: None Min Samples Leaf: 10
Accuracy: 0.8993053682897922 Precision: 0.2801758470635311 Recall: 0.12049878729602892 ROC-AUC: 0.6811488024637051
Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9051897268233644 Precision: 0.3180882940377058 Recall: 0.10439630304682904 ROC-AUC: 0.7118116547595573
Max Depth: None Min Samples Leaf: 20
Accuracy: 0.9084692084621194 Precision: 0.3466850639982084 Recall: 0.09151817620803973 ROC-AUC: 0.7298390685196249
Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155785818 ROC-AUC: 0.7714225769212407
Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155785818 ROC-AUC: 0.7714225769212407
Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155785818 ROC-AUC: 0.7714289552773523
Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155785818 ROC-AUC: 0.7714289552773523
Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155785818 ROC-AUC: 0.7714289552773523
```

MODEL EVALUATION

```
## TRAINING AND EVALUATING ON TEST DATA


clf_vsoft=voting_clf_soft.fit(X_train, y_train)
y_pred=clf_vsoft.predict(X_test)

confM = cm(y_test, y_pred)
print(confM)

acc=accuracy_score(y_test, y_pred)
f1= f1_score(y_test, y_pred)
auc = roc_auc_score(y_test,y_pred)
print('Accuracy = ',acc, ', F1 = ', f1, ',AUC= ', auc)
```

```
[[42469    48]
 [ 3865    69]]
Accuracy =  0.9157606940647134 , F1 =  0.03406566279930881 ,AUC=  0.508205219960522
```


Thank you!