Model_Building

```
import numpy as np    # import numpy library
import pandas as pd   # import pandas library for accessing and
analyzing the data
from sklearn.impute import KNNImputer
#KNN Iputation library for handaling missing data commented out after
processing once and stored the imputed data in new file as it takes 1
hour to process,
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
import matplotlib.pyplot as plt # import matplotlib library for plots
and visualization
import seaborn as sns
from sklearn.model_selection import train_test_split   # import train-
test split for splitting the data into train and test
from sklearn.preprocessing import MinMaxScaler   #library used for
scaling and standardizing the data
%matplotlib inline
#It is used to plot the matplotlib charts just below the code cells

df=pd.read_csv('Loan_Cleaned_10.csv')

df

        Unnamed: 0  Loan Status  Current Loan Amount  Term  Credit
Score  \
0                0            1                11520     1
741.0
1                1            1                 3441     1
734.0
2                2            1                21029     1
747.0
3                3            1                18743     1
747.0
4                4            1                11731     1
746.0
...            ...          ...                  ...   ...            .
..
230998      256444            0                11953     1
717.0
230999      256446            1                 3911     1
718.0
231000      256447            1                 5078     1
737.0
231001      256448            0                12116     1
```

```
746.0
231002      256450            1                27902     0
678.0

        Years in current job  Annual Income  Monthly Debt  \
0                       10.0        33694.0        584.03
1                        4.0        42269.0       1106.04
2                       10.0        90126.0       1321.85
3                       10.0        38072.0        751.92
4                        4.0        50025.0        355.18
...                      ...            ...           ...
230998                  10.0        39844.0        982.82
230999                   2.0        90041.0       1706.58
231000                  10.0        77186.0       1376.47
231001                   9.0        52504.0        297.96
231002                  10.0       117480.0       2111.38

        Years of Credit History  Months since last delinquent  \
0                          12.3                          41.0
1                          26.3                          24.0
2                          28.8                          35.6
3                          26.2                          40.0
4                          11.5                          42.4
...                         ...                           ...
230998                     11.7                          52.2
230999                     19.9                          47.8
231000                     19.1                          47.0
231001                     15.1                          82.0
231002                     18.0                          11.0

        Number of Open Accounts  Number of Credit Problems  \
0                          10.0                        0.0
1                          17.0                        0.0
2                           5.0                        0.0
3                           9.0                        0.0
4                          12.0                        0.0
...                         ...                        ...
230998                      9.0                        1.0
230999                     16.0                        0.0
231000                      9.0                        0.0
231001                      8.0                        0.0
231002                     10.0                        0.0

        Current Credit Balance  Maximum Open Credit  Bankruptcies  Tax
Liens  \
0                       6760.0                16056             0
0
1                       6262.0                19149             0
0
2                      20967.0                28335             0
```

```
0
3                       22529.0                 43915             0
0
4                       17391.0                 37081             0
0
...                         ...                   ...           ...
...
230998                   4176.0                  4783             1
0
230999                  39804.7                 44080             0
0
231000                   1717.0                  9758             0
0
231001                   3315.0                 20090             0
0
231002                  28317.0                 62371             0
0

        Home Mortgage  Own Home  Purpose
0                   1         0   203605
1                   1         0    14196
2                   1         0   203605
3                   0         1   203605
4                   0         0   203605
...               ...       ...      ...
230998              1         0   203605
230999              0         0   203605
231000              0         1   203605
231001              1         0   203605
231002              1         0   203605

[231003 rows x 19 columns]
```

```
X=df.drop(columns=['Loan Status','Unnamed: 0'])
```

```
type(X)
```

```
pandas.core.frame.DataFrame
```

```
y = df['Loan Status']
```

```
y.value_counts()
```

```
Loan Status
1    175812
0     55191
Name: count, dtype: int64
```

```
# Scaling
scaler = MinMaxScaler()
X_scaled=scaler.fit_transform(X)
```

```
X = pd.DataFrame(X_scaled,columns=X.columns)


X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size =
0.25, random_state=100)

X_train.head(2)
```

```
        Current Loan Amount  Term  Credit Score  Years in current job
\
199453             0.000239   0.0      0.653333                   0.3

97271              0.000042   1.0      0.346667                   0.8


        Annual Income  Monthly Debt  Years of Credit History  \
199453       1.000000      0.692041                 0.266766
97271        0.260097      0.162323                 0.143070

        Months since last delinquent  Number of Open Accounts  \
199453                      0.172727                 0.236842
97271                       0.125000                 0.197368

        Number of Credit Problems  Current Credit Balance  \
199453                        0.0                0.234762
97271                         0.0                0.212523

        Maximum Open Credit  Bankruptcies  Tax Liens  Home Mortgage
Own Home  \
199453             0.000184           0.0        0.0            1.0
0.0
97271              0.000076           0.0        0.0            0.0
1.0

         Purpose
199453  1.000000
97271   0.068548
```

```
X_train.shape
```

```
(173252, 17)
```

```
X_test.head(2)
```

```
        Current Loan Amount  Term  Credit Score  Years in current job
\
196836             0.000054   1.0      0.848000                   1.0

61487              0.000046   1.0      0.405333                   1.0


        Annual Income  Monthly Debt  Years of Credit History  \
```

```
196836       0.376166      0.119752                 0.339791
61487        0.419985      0.096373                 0.189270

        Months since last delinquent  Number of Open Accounts  \
196836                      0.264773                 0.105263
61487                       0.213636                 0.078947

        Number of Credit Problems  Current Credit Balance  \
196836                        0.0                0.317542
61487                         0.0                0.238346

        Maximum Open Credit  Bankruptcies  Tax Liens  Home Mortgage
Own Home  \
196836             0.000131           0.0        0.0            0.0
0.0
61487              0.000138           0.0        0.0            1.0
0.0

         Purpose
196836  1.000000
61487   0.014778
```

```
X_test.shape
```

```
(57751, 17)
```

```
Y_train.head(2)
```

```
199453    1
97271     1
Name: Loan Status, dtype: int64
```

```
Y_train.shape
```

```
(173252,)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report,ConfusionMatrixDisplay, \
                            precision_score, recall_score, f1_score,
roc_auc_score,roc_curve
```

```
models={
    "Logisitic Regression":LogisticRegression(),
    "Decision Tree":DecisionTreeClassifier(),
    "Random Forest":RandomForestClassifier(),
    "Gradient Boost":GradientBoostingClassifier()
}
```

```
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, Y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Training set performance
    model_train_accuracy = accuracy_score(Y_train, y_train_pred) #
Calculate Accuracy
    model_train_f1 = f1_score(Y_train, y_train_pred,
average='weighted') # Calculate F1-score
    model_train_precision = precision_score(Y_train, y_train_pred) #
Calculate Precision
    model_train_recall = recall_score(Y_train, y_train_pred) #
Calculate Recall
    model_train_rocauc_score = roc_auc_score(Y_train, y_train_pred)


    # Test set performance
    model_test_accuracy = accuracy_score(Y_test, y_test_pred) #
Calculate Accuracy
    model_test_f1 = f1_score(Y_test, y_test_pred, average='weighted')
# Calculate F1-score
    model_test_precision = precision_score(Y_test, y_test_pred) #
Calculate Precision
    model_test_recall = recall_score(Y_test, y_test_pred) # Calculate
Recall
    model_test_rocauc_score = roc_auc_score(Y_test, y_test_pred)
#Calculate Roc


    print(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Accuracy: {:.4f}".format(model_train_accuracy))
    print('- F1 score: {:.4f}'.format(model_train_f1))

    print('- Precision: {:.4f}'.format(model_train_precision))
    print('- Recall: {:.4f}'.format(model_train_recall))
    print('- Roc Auc Score: {:.4f}'.format(model_train_rocauc_score))



    print('----------------------------------')

    print('Model performance for Test set')
    print('- Accuracy: {:.4f}'.format(model_test_accuracy))
    print('- F1 score: {:.4f}'.format(model_test_f1))
```

```python
    print('- Precision: {:.4f}'.format(model_test_precision))
    print('- Recall: {:.4f}'.format(model_test_recall))
    print('- Roc Auc Score: {:.4f}'.format(model_test_rocauc_score))


    print('='*35)
    print('\n')
```

```
Logisitic Regression
Model performance for Training set
- Accuracy: 0.7665
- F1 score: 0.7099
- Precision: 0.7809
- Recall: 0.9633
- Roc Auc Score: 0.5523
----------------------------------
Model performance for Test set
- Accuracy: 0.7676
- F1 score: 0.7108
- Precision: 0.7821
- Recall: 0.9635
- Roc Auc Score: 0.5513
===================================


Decision Tree
Model performance for Training set
- Accuracy: 1.0000
- F1 score: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- Roc Auc Score: 1.0000
----------------------------------
Model performance for Test set
- Accuracy: 0.6917
- F1 score: 0.6952
- Precision: 0.8044
- Recall: 0.7869
- Roc Auc Score: 0.5867
===================================


Random Forest
Model performance for Training set
- Accuracy: 1.0000
- F1 score: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- Roc Auc Score: 1.0000
----------------------------------
```

```
Model performance for Test set
- Accuracy: 0.7827
- F1 score: 0.7414
- Precision: 0.7980
- Recall: 0.9572
- Roc Auc Score: 0.5900
===================================


Gradient Boost
Model performance for Training set
- Accuracy: 0.7694
- F1 score: 0.7098
- Precision: 0.7806
- Recall: 0.9694
- Roc Auc Score: 0.5516
----------------------------------
Model performance for Test set
- Accuracy: 0.7700
- F1 score: 0.7097
- Precision: 0.7812
- Recall: 0.9699
- Roc Auc Score: 0.5494
===================================
```

Logistic Regression

- Logistic Regression demonstrates consistent performance across training and test sets, with accuracy around 77%.
- It achieves a high recall (~96%), meaning it correctly identifies almost all positive cases (approved loans).
- The precision (~78%) indicates it manages false positives relatively well. However, the low ROC AUC (~0.55) suggests the model struggles to differentiate between approved and denied loans effectively.

Decision Tree

- Decision Trees achieve perfect performance on the training set (100%), indicating overfitting to the training data.
- Test accuracy drops to ~69%, with a marginal improvement in ROC AUC (~0.59), showing reduced generalization to unseen data.
- While recall (~78%) is reasonable, the precision (~80%) indicates that the model is prone to false positives.
- Inference: Overfits easily; not ideal unless tuned to prevent memorization.

Random Forest

- Random Forest mitigates the overfitting problem seen in Decision Trees by averaging across multiple trees.
- It achieves an accuracy of ~78% on the test set, with balanced recall (~96%) and precision (~80%).
- The ROC AUC (~0.59) still shows limited ability to distinguish between approved and denied loans.
- Inference: A robust model for handling large datasets and complex relationships but requires optimization for better performance.

Gradient Boosting

- Gradient Boosting delivers similar accuracy (~77%) and recall (~97%) to Logistic Regression but at the cost of slightly reduced interpretability.
- Its ROC AUC (~0.55) suggests challenges in separating loan approval classes effectively. While it avoids overfitting seen in Decision Trees, its performance gains over Logistic Regression are minimal. -Inference: Performs well with non-linear data but may not offer significant advantages over simpler models in this scenario.

Final-Algo to Use

- Random Forest

```
rf_params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

random_cv_model = [
                    ("RF",RandomForestClassifier(),rf_params)
]

from sklearn.model_selection import RandomizedSearchCV

model_param = {}
for name, model, params in random_cv_model:
    random = RandomizedSearchCV(estimator=model,
                                   param_distributions=params,
                                   n_iter=20,
                                   cv=5,
                                   verbose=2,
                                   n_jobs=-1)
    random.fit(X_train, Y_train)
    model_param[name] = random.best_params_

for model_name in model_param:
    print(f"---------------- Best Params for {model_name}
```

```
-------------------")
    print(model_param[model_name])
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
---------------- Best Params for RF -------------------
{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 1,
'max_depth': 30, 'bootstrap': False}
```

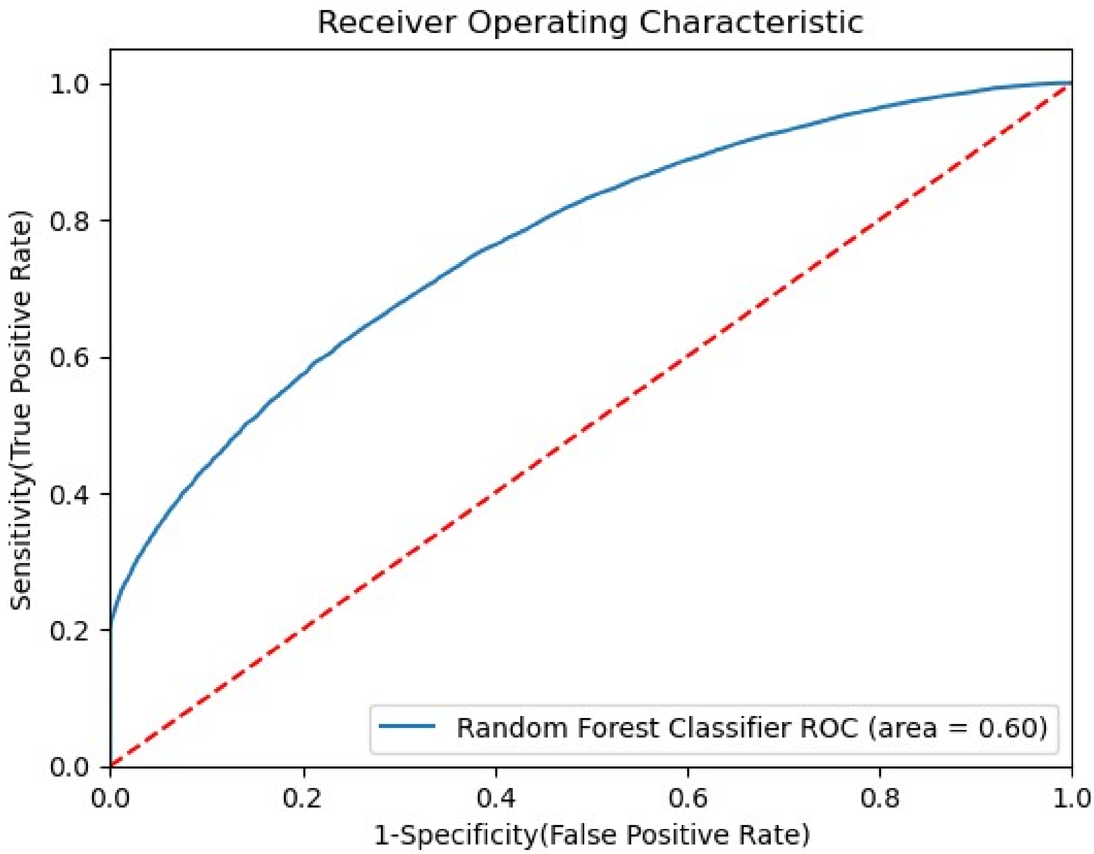
```
models={

    "Random
Forest":RandomForestClassifier(n_estimators=100,min_samples_split=2,mi
n_samples_leaf= 1, max_depth= 30, bootstrap= False),

}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, Y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Training set performance
    model_train_accuracy = accuracy_score(Y_train, y_train_pred) #
Calculate Accuracy
    model_train_f1 = f1_score(Y_train, y_train_pred,
average='weighted') # Calculate F1-score
    model_train_precision = precision_score(Y_train, y_train_pred) #
Calculate Precision
    model_train_recall = recall_score(Y_train, y_train_pred) #
Calculate Recall
    model_train_rocauc_score = roc_auc_score(Y_train, y_train_pred)


    # Test set performance
    model_test_accuracy = accuracy_score(Y_test, y_test_pred) #
Calculate Accuracy
    model_test_f1 = f1_score(Y_test, y_test_pred, average='weighted')
# Calculate F1-score
    model_test_precision = precision_score(Y_test, y_test_pred) #
Calculate Precision
    model_test_recall = recall_score(Y_test, y_test_pred) # Calculate
Recall
    model_test_rocauc_score = roc_auc_score(Y_test, y_test_pred)
#Calculate Roc


    print(list(models.keys())[i])
```

```python
    print('Model performance for Training set')
    print("- Accuracy: {:.4f}".format(model_train_accuracy))
    print('- F1 score: {:.4f}'.format(model_train_f1))

    print('- Precision: {:.4f}'.format(model_train_precision))
    print('- Recall: {:.4f}'.format(model_train_recall))
    print('- Roc Auc Score: {:.4f}'.format(model_train_rocauc_score))



    print('----------------------------------')

    print('Model performance for Test set')
    print('- Accuracy: {:.4f}'.format(model_test_accuracy))
    print('- F1 score: {:.4f}'.format(model_test_f1))
    print('- Precision: {:.4f}'.format(model_test_precision))
    print('- Recall: {:.4f}'.format(model_test_recall))
    print('- Roc Auc Score: {:.4f}'.format(model_test_rocauc_score))


    print('='*35)
    print('\n')
```

```
Random Forest
Model performance for Training set
- Accuracy: 1.0000
- F1 score: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- Roc Auc Score: 1.0000
----------------------------------
Model performance for Test set
- Accuracy: 0.7858
- F1 score: 0.7468
- Precision: 0.8011
- Recall: 0.9565
- Roc Auc Score: 0.5973
===================================
```

```python
## Plot ROC AUC Curve
from sklearn.metrics import roc_auc_score,roc_curve
plt.figure()

# Add the models to the list that you want to view on the ROC plot
auc_models = [
{
    'label': 'Random Forest Classifier',
    'model':
```

```
RandomForestClassifier(n_estimators=1000,min_samples_split=2,

max_features=7,max_depth=None),
    'auc':  0.5973
},

]
# create loop through all model
for algo in auc_models:
    model = algo['model'] # select the model
    model.fit(X_train, Y_train) # train the model
# Compute False postive rate, and True positive rate
    fpr, tpr, thresholds = roc_curve(Y_test,
model.predict_proba(X_test)[:,1])
# Calculate Area under the curve to display on the plot
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (algo['label'],
algo['auc']))

# Custom settings for the plot
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.savefig("auc.png")
plt.show()
```



The achieved ROC AUC score of 0.6 indicates that the model performs moderately better.