```
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 Credit Problems  Current Credit Balance  \
0                               0                  6760.0
1                               0                  6262.0
2                               0                 20967.0
3                               0                 22529.0
4                               0                 17391.0
...                           ...                     ...
230998                          1                  4176.0
230999                          0                 39804.7
231000                          0                  1717.0
231001                          0                  3315.0
231002                          0                 28317.0

        Maximum Open Credit  Bankruptcies  Tax Liens  Home Mortgage
Own Home  \
0                     16056             0          0              1
0
1                     19149             0          0              1
0
2                     28335             0          0              1
0
3                     43915             0          0              0
1
4                     37081             0          0              0
0
...                     ...           ...        ...            ...
```

```
...
230998                   4783             1          0              1
0
230999                  44080             0          0              0
0
231000                   9758             0          0              0
1
231001                  20090             0          0              1
0
231002                  62371             0          0              1
0

        Purpose  Unnamed: 19  Unnamed: 20
0        203605          NaN          NaN
1         14196          NaN          NaN
2        203605          NaN          NaN
3        203605          NaN          NaN
4        203605          NaN          NaN
...         ...          ...          ...
230998   203605          NaN          NaN
230999   203605          NaN          NaN
231000   203605          NaN          NaN
231001   203605          NaN          NaN
231002   203605          NaN          NaN

[231003 rows x 21 columns]
```

```
X=df.drop(columns=['Loan Status','Unnamed: 0','Unnamed: 19','Unnamed: 
20'])
```

```
type(X)
```

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

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

```
y.value_counts()
```

```
Loan Status
1    138315
0     92688
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_train.shape
```

```
(173252, 17)
```

```
X_train.isna().sum()
```

```
Current Loan Amount             0
Term                            0
Credit Score                    0
Years in current job            0
Annual Income                   0
Monthly Debt                    0
Years of Credit History         0
Months since last delinquent    0
Number of Open Accounts         0
Number of Credit Problems       0
Current Credit Balance          0
Maximum Open Credit             0
Bankruptcies                    0
Tax Liens                       0
Home Mortgage                   0
Own Home                        0
Purpose                         0
dtype: int64
```

```
X_test.shape
```

```
(57751, 17)
```

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

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

```
Y_train.value_counts()
```

```
Loan Status
1    103748
0     69504
Name: count, dtype: int64
```

```
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))
    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: 0.6285
- F1 score: 0.5857
- Precision: 0.6381
- Recall: 0.8771
- Roc Auc Score: 0.5672
----------------------------------
Model performance for Test set
- Accuracy: 0.6290
- F1 score: 0.5866
- Precision: 0.6384
- Recall: 0.8768
- Roc Auc Score: 0.5682
===================================


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: 0.5552
- F1 score: 0.5561
- Precision: 0.6307
- Recall: 0.6196
- Roc Auc Score: 0.5393
===================================


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: 0.6380
- F1 score: 0.6187
- Precision: 0.6601
- Recall: 0.8149
- Roc Auc Score: 0.5946
===================================


Gradient Boost
Model performance for Training set
- Accuracy: 0.6346
- F1 score: 0.5981
- Precision: 0.6449
- Recall: 0.8674
- Roc Auc Score: 0.5772
----------------------------------
Model performance for Test set
- Accuracy: 0.6317
- F1 score: 0.5948
- Precision: 0.6428
- Recall: 0.8657
- Roc Auc Score: 0.5743
===================================
```

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.
- Inference: Reliable for moderately balanced datasets and interpretable results but limited in handling complex relationships.

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]
}

gb_params = {
    'n_estimators': [100, 200, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 0.9, 1.0],  # Fraction of samples to use for
fitting each tree
    'criterion': ['friedman_mse', 'mse']  # Splitting criterion for
tree building
}



random_cv_model = [
                    ("RF",RandomForestClassifier(),rf_params),
                    ("GB",GradientBoostingClassifier(),gb_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

c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\model_selection\
_validation.py:547: FitFailedWarning:
6 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
```

```
---------------------------------------------------------------------
----------
1 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\
model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py",
line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 489, in fit
    trees = Parallel(
            ^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 67, in __call__
    return super().__call__(iterable_with_config)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1918, in __call__
    return output if self.return_generator else list(output)
                                                ^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1847, in _get_sequential_output
    res = func(*args, **kwargs)
          ^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 129, in __call__
    return self.function(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 200, in _parallel_build_trees
    tree._fit(
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\tree\
_classes.py", line 305, in _fit
    classes_k, y_encoded[:, k] = np.unique(y[:, k],
return_inverse=True)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\numpy\lib\
arraysetops.py", line 274, in unique
    ret = _unique1d(ar, return_index, return_inverse, return_counts,
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\numpy\lib\
arraysetops.py", line 359, in _unique1d
    inv_idx = np.empty(mask.shape, dtype=np.intp)
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 1.06 MiB
```

```
for an array with shape (138602,) and data type int64

--------------------------------------------------------------------------------
----------
1 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\
model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py",
line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 489, in fit
    trees = Parallel(
            ^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 67, in __call__
    return super().__call__(iterable_with_config)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1918, in __call__
    return output if self.return_generator else list(output)
                                                ^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1847, in _get_sequential_output
    res = func(*args, **kwargs)
          ^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 129, in __call__
    return self.function(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 200, in _parallel_build_trees
    tree._fit(
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\tree\
_classes.py", line 472, in _fit
    builder.build(self.tree_, X, y, sample_weight,
missing_values_in_feature_mask)
  File "sklearn\\tree\\_tree.pyx", line 166, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 285, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 940, in
sklearn.tree._tree.Tree._add_node
  File "sklearn\\tree\\_tree.pyx", line 908, in
sklearn.tree._tree.Tree._resize_c
  File "sklearn\\tree\\_utils.pyx", line 35, in
```

```
sklearn.tree._utils.safe_realloc
MemoryError: could not allocate 4194304 bytes

--------------------------------------------------------------------------------
----------
2 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\
model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py",
line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 489, in fit
    trees = Parallel(
            ^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 67, in __call__
    return super().__call__(iterable_with_config)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1918, in __call__
    return output if self.return_generator else list(output)
                                                ^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1847, in _get_sequential_output
    res = func(*args, **kwargs)
          ^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 129, in __call__
    return self.function(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 192, in _parallel_build_trees
    tree._fit(
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\tree\
_classes.py", line 472, in _fit
    builder.build(self.tree_, X, y, sample_weight,
missing_values_in_feature_mask)
  File "sklearn\\tree\\_tree.pyx", line 166, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 285, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 940, in
sklearn.tree._tree.Tree._add_node
  File "sklearn\\tree\\_tree.pyx", line 908, in
sklearn.tree._tree.Tree._resize_c
```

```
  File "sklearn\\tree\\_utils.pyx", line 35, in
sklearn.tree._utils.safe_realloc
MemoryError: could not allocate 4194304 bytes

--------------------------------------------------------------------------------
----------
2 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\
model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py",
line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 489, in fit
    trees = Parallel(
            ^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 67, in __call__
    return super().__call__(iterable_with_config)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1918, in __call__
    return output if self.return_generator else list(output)
                                                ^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py", line 1847, in _get_sequential_output
    res = func(*args, **kwargs)
          ^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py", line 129, in __call__
    return self.function(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py", line 192, in _parallel_build_trees
    tree._fit(
  File "c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\tree\
_classes.py", line 472, in _fit
    builder.build(self.tree_, X, y, sample_weight,
missing_values_in_feature_mask)
  File "sklearn\\tree\\_tree.pyx", line 166, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 285, in
sklearn.tree._tree.DepthFirstTreeBuilder.build
  File "sklearn\\tree\\_tree.pyx", line 940, in
sklearn.tree._tree.Tree._add_node
  File "sklearn\\tree\\_tree.pyx", line 908, in
```

```
sklearn.tree._tree.Tree._resize_c
  File "sklearn\\tree\\_utils.pyx", line 35, in
sklearn.tree._utils.safe_realloc
MemoryError: could not allocate 2097152 bytes

  warnings.warn(some_fits_failed_message, FitFailedWarning)
c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\model_selection\
_search.py:1051: UserWarning: One or more of the test scores are non-
finite: [0.64090456 0.63904023 0.63827256 0.63813402 0.63919608
0.63032461
 0.63783967 0.63101147 0.63983675        nan        nan 0.63814557
 0.63013413 0.63075173 0.63793778 0.63948467 0.63819752 0.64010226
 0.6413663  0.63897674]
  warnings.warn(

----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
Cell In[45], line 11
      4 for name, model, params in random_cv_model:
      5     random = RandomizedSearchCV(estimator=model,
      6                                 param_distributions=params,
      7                                 n_iter=20,
      8                                 cv=5,
      9                                 verbose=2,
     10                                 n_jobs=-1)
---> 11     random.fit(X_train, Y_train)
     12     model_param[name] = random.best_params_
     14 for model_name in model_param:

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py:1474,
in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or
global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\
model_selection\_search.py:1008, in BaseSearchCV.fit(self, X, y,
**params)
   1006 refit_start_time = time.time()
   1007 if y is not None:
-> 1008     self.best_estimator_.fit(X, y,
**routed_params.estimator.fit)
```

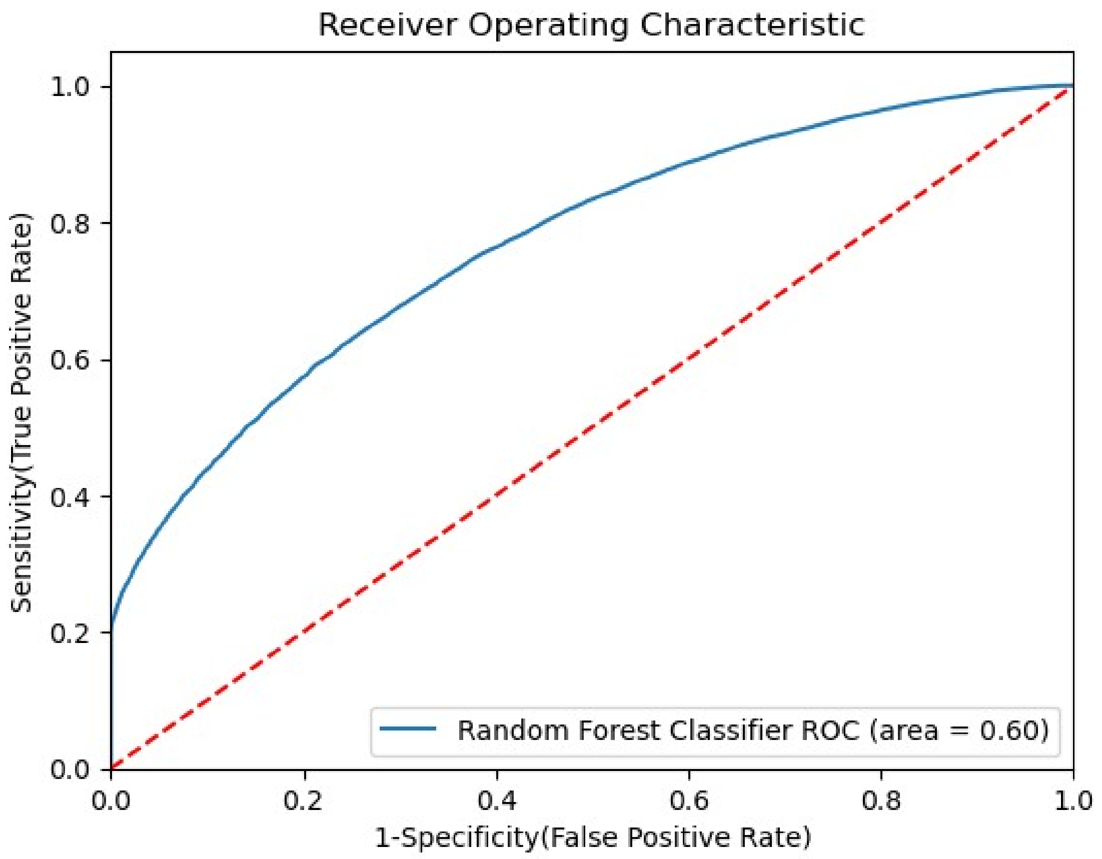
```
   1009 else:
   1010     self.best_estimator_.fit(X, **routed_params.estimator.fit)

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\base.py:1474,
in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or
global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py:489, in BaseForest.fit(self, X, y, sample_weight)
    478 trees = [
    479     self._make_estimator(append=False,
random_state=random_state)
    480     for i in range(n_more_estimators)
    481 ]
    483 # Parallel loop: we prefer the threading backend as the Cython
code
    484 # for fitting the trees is internally releasing the Python GIL
    485 # making threading more efficient than multiprocessing in
    486 # that case. However, for joblib 0.12+ we respect any
    487 # parallel_backend contexts set at a higher level,
    488 # since correctness does not rely on using threads.
--> 489 trees = Parallel(
    490     n_jobs=self.n_jobs,
    491     verbose=self.verbose,
    492     prefer="threads",
    493 )(
    494     delayed(_parallel_build_trees)(
    495         t,
    496         self.bootstrap,
    497         X,
    498         y,
    499         sample_weight,
    500         i,
    501         len(trees),
    502         verbose=self.verbose,
    503         class_weight=self.class_weight,
    504         n_samples_bootstrap=n_samples_bootstrap,
    505
missing_values_in_feature_mask=missing_values_in_feature_mask,
    506     )
    507     for i, t in enumerate(trees)
    508 )
    510 # Collect newly grown trees
    511 self.estimators_.extend(trees)

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py:67, in Parallel.__call__(self, iterable)
     62 config = get_config()
     63 iterable_with_config = (
     64     (_with_config(delayed_func, config), args, kwargs)
     65     for delayed_func, args, kwargs in iterable
     66 )
---> 67 return super().__call__(iterable_with_config)

File c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py:1918, in Parallel.__call__(self, iterable)
   1916     output = self._get_sequential_output(iterable)
   1917     next(output)
-> 1918     return output if self.return_generator else list(output)
   1920 # Let's create an ID that uniquely identifies the current
call. If the
   1921 # call is interrupted early and that the same instance is
immediately
   1922 # re-used, this id will be used to prevent workers that were
   1923 # concurrently finalizing a task from the previous call to run
the
   1924 # callback.
   1925 with self._lock:

File c:\Users\prakhar\anaconda\Lib\site-packages\joblib\
parallel.py:1847, in Parallel._get_sequential_output(self, iterable)
   1845 self.n_dispatched_batches += 1
   1846 self.n_dispatched_tasks += 1
-> 1847 res = func(*args, **kwargs)
   1848 self.n_completed_tasks += 1
   1849 self.print_progress()

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\utils\
parallel.py:129, in _FuncWrapper.__call__(self, *args, **kwargs)
    127     config = {}
    128 with config_context(**config):
--> 129     return self.function(*args, **kwargs)

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\ensemble\
_forest.py:192, in _parallel_build_trees(tree, bootstrap, X, y,
sample_weight, tree_idx, n_trees, verbose, class_weight,
n_samples_bootstrap, missing_values_in_feature_mask)
    189     elif class_weight == "balanced_subsample":
    190         curr_sample_weight *=
compute_sample_weight("balanced", y, indices=indices)
--> 192     tree._fit(
    193         X,
    194         y,
    195         sample_weight=curr_sample_weight,
    196         check_input=False,
    197
missing_values_in_feature_mask=missing_values_in_feature_mask,
    198     )
    199 else:
    200     tree._fit(
    201         X,
    202         y,
   (...)
    205
missing_values_in_feature_mask=missing_values_in_feature_mask,
    206     )

File c:\Users\prakhar\anaconda\Lib\site-packages\sklearn\tree\
_classes.py:472, in BaseDecisionTree._fit(self, X, y, sample_weight,
check_input, missing_values_in_feature_mask)
    461 else:
    462     builder = BestFirstTreeBuilder(
    463         splitter,
    464         min_samples_split,
   (...)
    469         self.min_impurity_decrease,
    470     )
--> 472 builder.build(self.tree_, X, y, sample_weight,
missing_values_in_feature_mask)
    474 if self.n_outputs_ == 1 and is_classifier(self):
    475     self.n_classes_ = self.n_classes_[0]

KeyboardInterrupt:
```

```python
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])

    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
===================================



from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,)

## 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.