1 Introduction

1.1 Phase 1 Summary

Phase 1 of loan approval prediction project focused on data exploration and visualization of the dataset. This phase involved an assessment of the data to identify and address potential issues and to set a foundation for predictive modeling.

The primary objectives were to understand the dataset's context, identify data issues, explore data distributions through visualizations, and prepare the data through cleaning and preprocessing steps.

The dataset comprised data on 953 loan applicants, detailing attributes like gender, marital status, income levels, loan amounts, and credit histories. Basic statistical summaries and data types were established for further analysis.

Data Quality and Preprocessing included the following steps:

- Data Quality Report: Examination of continuous and categorical variables for missing values, outliers, and data distribution characteristics.
- Outlier Removal: Outliers were identified using interquartile range calculations and addressed through clamping.
- Imputation of Missing Values: Missing data were handled using median imputation for skewed distributions (like LoanAmount) and mode imputation for categorical variables.

Data Exploration and Visualization:

Histograms and Bar Plots were used to visualize the distribution of both continuous and categorical variables.

Histograms for continuous variables such as ApplicantIncome, CoapplicantIncome, and LoanAmount showed skewed distributions. This was important for deciding on the median imputation strategy for missing values, as median imputation is less sensitive to outliers in skewed distributions.

Bar Plots for categorical variables showed significant demographic insights:

- A higher proportion of loan applicants were male compared to female.
- Married applicants outnumbered unmarried ones.
- Most applicants did not have dependents and were graduates.
- Most applicants were not self-employed but had a credit history.
- The distribution of property types was roughly even among Rural, Semiurban, and Urban areas.

Scatter Plot Matrix (SPLOM) and Box Plots were used to analyze relationships between continuous variables and categorical groupings.

SPLOM highlighted potential linear relationships between some pairs of continuous variables. Notably, ApplicantIncome and LoanAmount appeared to have a linear relationship, suggesting that higher incomes might correlate with larger loan amounts requested.

Box Plots provided insights into the distribution of continuous variables across different levels of categorical variables:

- Males generally had a higher income range compared to females.
- Married applicants generally applied for and received larger loan amounts than their unmarried counterparts.
- Graduates had higher incomes and loan amounts compared to non-graduates.
- Self-employed applicants showed a wider range of incomes and typically higher loan amounts, reflecting higher variability in their financial status.

Correlation Analysis was used to find relationships between key financial metrics using correlation heatmaps.

- A mild negative correlation between ApplicantIncome and CoapplicantIncome, suggesting that higher applicant incomes were accompanied by lower coapplicant incomes.
- A positive correlation between CoapplicantIncome and LoanAmount, as well as between ApplicantIncome and LoanAmount, indicating that higher incomes (both applicant and coapplicant) were associated with larger loan amounts.

Phase 1 resulted in a refined dataset ready for modeling. The data cleaning and preprocessing set a strong foundation for predictive accuracy in Phase 2. The EDA done in Phase 1 helped in understanding the distribution and relationships of the features, which came in handy during the feature selection process in Phase 2.

1.2 Report Overview

Phase 2 Mainly focusses on predictive modelling and model evaluation to determine the best approach for predicting loan approvals. The key activities performed in Phase 2 are:

Summarizing Phase 1 and setting clear objectives for Phase 2

Using the cleaned and preprocessed data from Phase 1, one-hot encoding of categorical features is done, and numerical features are scaled to prepare them for modeling.

Feature Selection: Feature selection techniques such as random forest classifier and ANOVA F-tests are used to identify the most predictive features.

Model Fitting & Tuning: 4 models including KNN, Decision Trees, and Gaussian Naive Bayes are fitted to the data and tuned for optimal performance.

Model Evaluation: In-depth analysis is done using various metrics such as accuracy, precision, recall, F1 score, and AUC to evaluate model performance.

Model Comparison: Comparison of the performance metrics across all models to identify the most effective model. Cross validation and T-tests are performed to assess the statistical significance of the models.

Furthermore, potential biases, limitations of modelling techniques, and the implications of the findings are discussed.

1.3 Overview of Methodology

The methodology used in Phase 2 focuses on developing, tuning, and evaluating various predictive models to determine the most effective approach for predicting loan approval outcomes.

Data Preparation and feature selection:

- Categorical features are encoded through one-hot encoding, and numerical features are scaled using MinMax scaling to ensure all features contribute equally to the model predictions.
- Feature selection is done through Random Forest Classifier and F-tests. The most influential features are identified. This improves the model performance by reducing overfitting.

Model Selection and fitting:

- The dataset is divided into a training set (70%) and testing set (30%).
- Four models are fitted including K-Nearest Neighbors (KNN), Decision Trees, Gaussian Naive Bayes.
- Each model is fitted to the training data.

Hyperparameter Tuning:

- For models like KNN and Decision Trees, a GridSearchCV approach is used. This method systematically works through multiple combinations of parameter tunes, cross-validating as it goes to determine which tune gives the best performance based on a predefined scoring metric (ROC-AUC score).

Model Evaluation:

- Models are evaluated using several performance metrics including accuracy, precision, recall, F1 score, and the area under the ROC curve (AUC). These metrics

help in assessing the effectiveness of each model in predicting loan approvals accurately.
- ROC curves, Confusion Matrices and Precision vs Threshold charts for each model are evaluated for insights.

- Validation techniques such as stratified k-fold cross-validation are employed to ensure that the model's performance is robust across different subsets of the dataset. This helps in assessing the model’s generalizability to new data.

Model Comparison and Selection

- After tuning and evaluating individual models, a comparative analysis is conducted to rank the models based on their performance metrics.

- The model that shows the best balance between accuracy, precision, recall, and computational efficiency is selected as the final model for predicting loan approvals.

2 Objectives

The objective of Phase 2 is to develop, optimize, and evaluate predictive models that accurately forecast the outcome of loan approval decisions based on the cleansed and preprocessed dataset from Phase 1. This includes:

- Utilizing various machine learning algorithms to create models that can effectively predict loan approvals

- Conducting thorough hyperparameter tuning for each model to find the optimal settings that maximize the predictive accuracy

- Assessing each model's effectiveness using a range of performance metrics

- Identifying and rank the importance of various features in the prediction process using methods such as feature importance

- Performing a comparative analysis of all developed models

- Provide a comprehensive report of the methods used and the findings.

3 Predictive Modelling

3.1 Further Pre-processing

The cleansed and preprocessed data from Phase 1 is loaded onto the environment and the data types and basic summary information for each of the columns are displayed. Due to prior processing, no missing values are observed and all categoricals have been turned into numeric columns.

In [2]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np

loan = pd.read_csv("Phase2_Group88.csv")
print("Column data types:\n", loan.dtypes)
print("\nColumn missing values:\n", loan.isna().sum())
print("\n", loan.describe())
display(loan.head(10))
```

```
Column data types:
 Gender                int64
Married                int64
Dependents             int64
Education              int64
Self_Employed          int64
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount             int64
Loan_Amount_Term       int64
Credit_History         int64
Property_Area         object
Loan_Status            int64
dtype: object

Column missing values:
 Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

	Gender	Married	Dependents	Education	Self_Employed
count	953.000000	953.000000	953.000000	953.000000	953.000000
mean	0.816369	0.656873	0.769150	0.771249	0.140609
std	0.387386	0.475003	1.016261	0.420249	0.347800
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	1.000000	0.000000
50%	1.000000	1.000000	0.000000	1.000000	0.000000
75%	1.000000	1.000000	2.000000	1.000000	0.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
count	953.000000	953.000000	953.000000	953.000000
mean	4622.578174	1433.790178	137.497377	7.648478
std	2504.288990	1632.284630	56.162843	1.142771
min	150.000000	0.000000	9.000000	0.000000
25%	2873.000000	0.000000	100.000000	8.000000
50%	3775.000000	1250.000000	128.000000	8.000000
75%	5818.000000	2306.000000	162.000000	8.000000
max	10235.500000	5765.000000	265.000000	9.000000

	Credit_History	Loan_Status
count	953.000000	953.000000
mean	0.843652	0.675761
std	0.363376	0.468336
min	0.000000	0.000000
25%	1.000000	0.000000
50%	1.000000	1.000000
75%	1.000000	1.000000
max	1.000000	1.000000

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicar
0	1	0	0	1	0	5849.0	
1	1	1	1	1	0	4583.0	
2	1	1	0	1	1	3000.0	
3	1	1	0	0	0	2583.0	
4	1	0	0	1	0	6000.0	
5	1	1	2	1	1	5417.0	
6	1	1	0	0	0	2333.0	
7	1	1	3	1	0	3036.0	
8	1	1	2	1	0	4006.0	
9	1	1	1	1	0	10235.5	

The predictive features and the target variable are split into two variables.

In [3]:
```
Data = loan.drop(columns='Loan_Status')
target = loan['Loan_Status']
```

One-Hot-Encoding and Scaling

The Property_Area column is a descriptive feature and therefore needs to undergo one-hot-encoding since its levels are not ordinal. This is performed in the cell below:

In [4]:
```
Data = pd.get_dummies(Data, columns=['Property_Area'], dtype='int')
Data.head()
```

Out[4]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	1	0	0	1	0	5849.0	
1	1	1	1	1	0	4583.0	
2	1	1	0	1	1	3000.0	
3	1	1	0	0	0	2583.0	
4	1	0	0	1	0	6000.0	

The features are scaled using MinMax scaling as shown below.

In [5]:
```
from sklearn import preprocessing

Data_df = Data.copy()

Data_scaler = preprocessing.MinMaxScaler()
Data_scaler.fit(Data)
Data = Data_scaler.fit_transform(Data)
Data = pd.DataFrame(Data, columns=Data_df.columns)
```

```
Data.head()
```

Out[5]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	1.0	0.0	0.000000	1.0	0.0	0.565069	
1	1.0	1.0	0.333333	1.0	0.0	0.439542	
2	1.0	1.0	0.000000	1.0	1.0	0.282584	
3	1.0	1.0	0.000000	0.0	0.0	0.241237	
4	1.0	0.0	0.000000	1.0	0.0	0.580041	

3.2 Predictive Modelling

Feature Selection

Random Forest Feature Importance

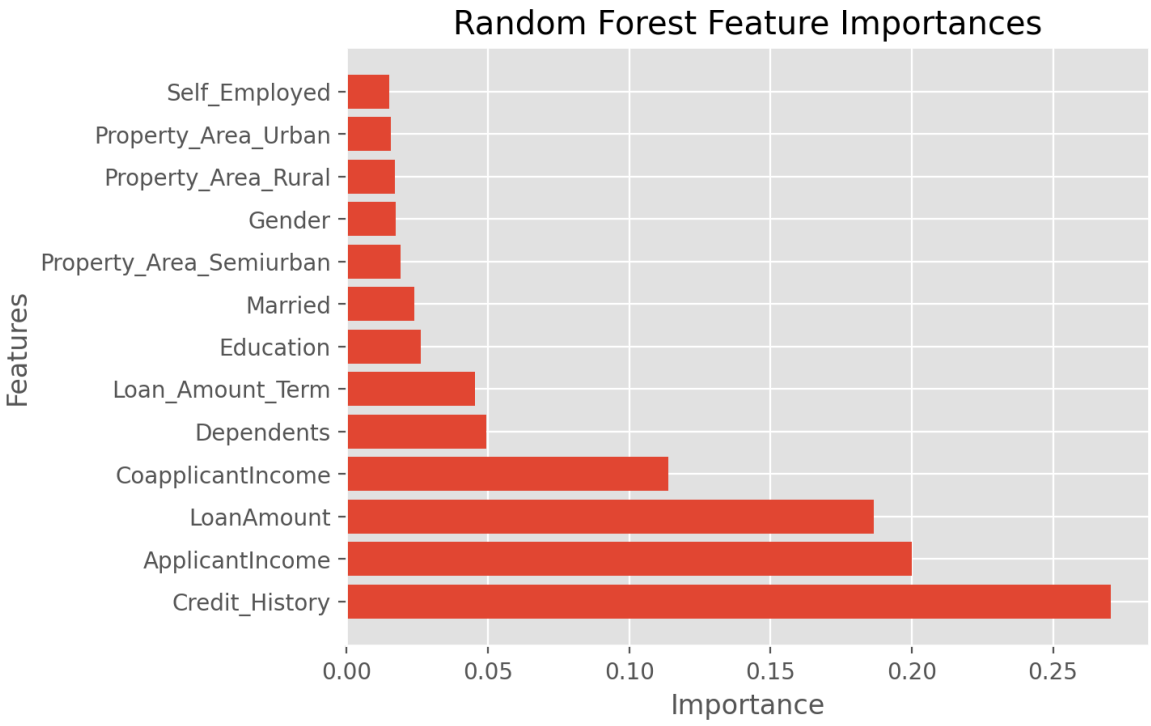
A Random Forest Classifier model, which was trained to predict loan approval outcomes, based on the features was used. In the Random Forest model, the importance of each feature is evaluated in terms of how effectively it improves in making accurate predictions at each decision node within the trees of the forest. It helps in understanding which variables are most influential in the loan approval process. More importance means that it is better at capturing the target variable.

- Credit History appears to be the most significant predictor, suggesting that the historical reliability of a borrower in managing credit is highly indicative of their likelihood of having a loan approved.
- ApplicantIncome and LoanAmount also show importance, which means that higher incomes and reasonable loan requests correlate positively with loan approval.
- Other features like CoapplicantIncome, Dependents, Loan_Amount_Term, and Education show moderate importance, therefore these variables are less critical than the borrower's credit history or income level.
- The least important features include Property Area (Urban, Semiurban, Rural) and Self_Employed, suggesting that these factors are least important in the prediction of loan approval.

In [6]:

```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

num_features = 10
model_rfi = RandomForestClassifier(n_estimators=100)
model_rfi.fit(Data, target)
fs_indices_rfi = np.argsort(model_rfi.feature_importances_)[::-1]

best_features_rfi = Data.columns[fs_indices_rfi].values
```

```
feature_importances_rfi = model_rfi.feature_importances_[fs_indices_rfi]

import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.style.use("ggplot")

def plot_imp(best_features, scores, method_name, xlab):
    plt.barh(best_features, scores)
    plt.title(method_name + ' Feature Importances')
    plt.xlabel(xlab)
    plt.ylabel("Features")
    plt.show()

plot_imp(best_features_rfi, feature_importances_rfi, 'Random Forest', 'Importanc
```



F-Classif Feature Importance

ANOVA F-test is used to evaluate the importance of categorical features in predicting loan approval outcomes. The F-test measures the degree of variance between different groups of a categorical variable in relation to the target variable.

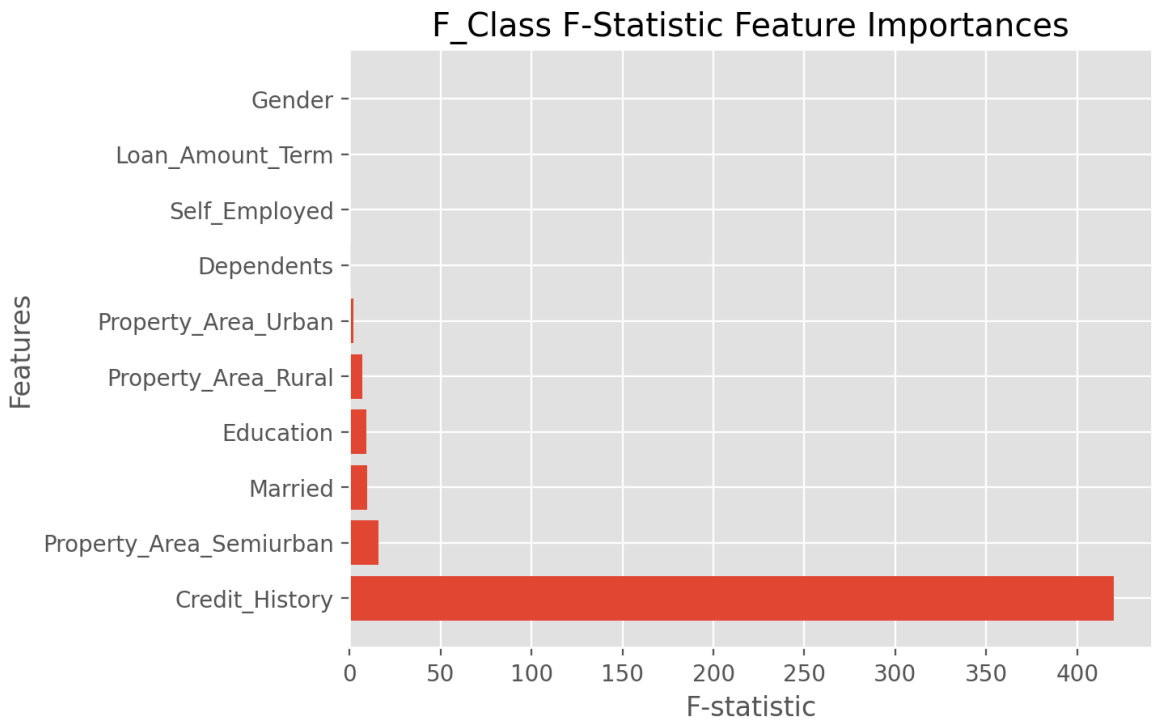
The F-Statistic chart ranks the features by their statistical significance in affecting the target variable. Higher f-statistic value indicates higher effect.

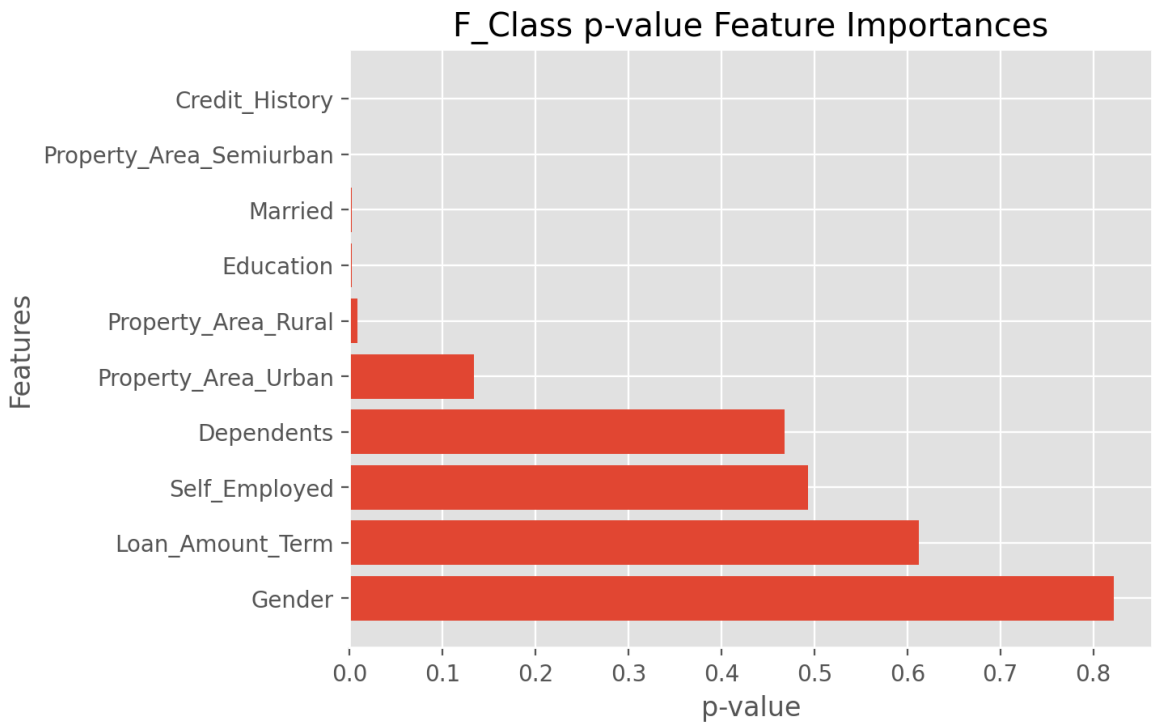
- Credit history is the most influential feature, with a significantly higher F-statistic value compared to other features, indicating strong predictive power in distinguishing between approved and unapproved loans.
- Property area, education, and marital status have marginal importance but still contribute to the model.
- Gender, employment, dependents, and loan amount term have the least influence based on their F-statistic values.

The F-Test P-Value chart inversely ranks the features by the probability that the observed differences in category means could have occurred by chance. Lower p-values indicate higher statistical significance.

- Credit history again shows the highest significance (lowest p-value
- Marital status, property area, and education are moderately significant.
- Features like Gender and Loan Amount Term show high p-values, suggesting that any observed differences in loan approval rates in these categories are due to randomness rather than an underlying effect.

In [7]:
```
from sklearn.feature_selection import f_classif

cat_feat = Data[["Gender", "Married", "Dependents", "Education", "Self_Employed"

f_statistic, p_values = f_classif(cat_feat, target)
f_statistic
p_values

fs_indices_fstat = np.argsort(f_statistic)[::-1]
best_features_fstat = cat_feat.columns[fs_indices_fstat].values
fstatistic_ordered = f_statistic[fs_indices_fstat]

fs_indices_pvalue = np.argsort(p_values)[::-1]
best_features_pvalue = cat_feat.columns[fs_indices_pvalue].values
pvalue_ordered = p_values[fs_indices_pvalue]

plot_imp(best_features_fstat, fstatistic_ordered, 'F_Class F-Statistic', 'F-stat
plot_imp(best_features_pvalue, pvalue_ordered, 'F_Class p-value', 'p-value')
```





F-Regression Feature Importance

F-regression is used to assess the impact of continuous features on the target variable. Since gender and employment status do not have much impact on the outcome, these variables have been dropped. The F-Statistic chart shows the degree to which each continuous feature contributes to explaining the variance in loan approvals.

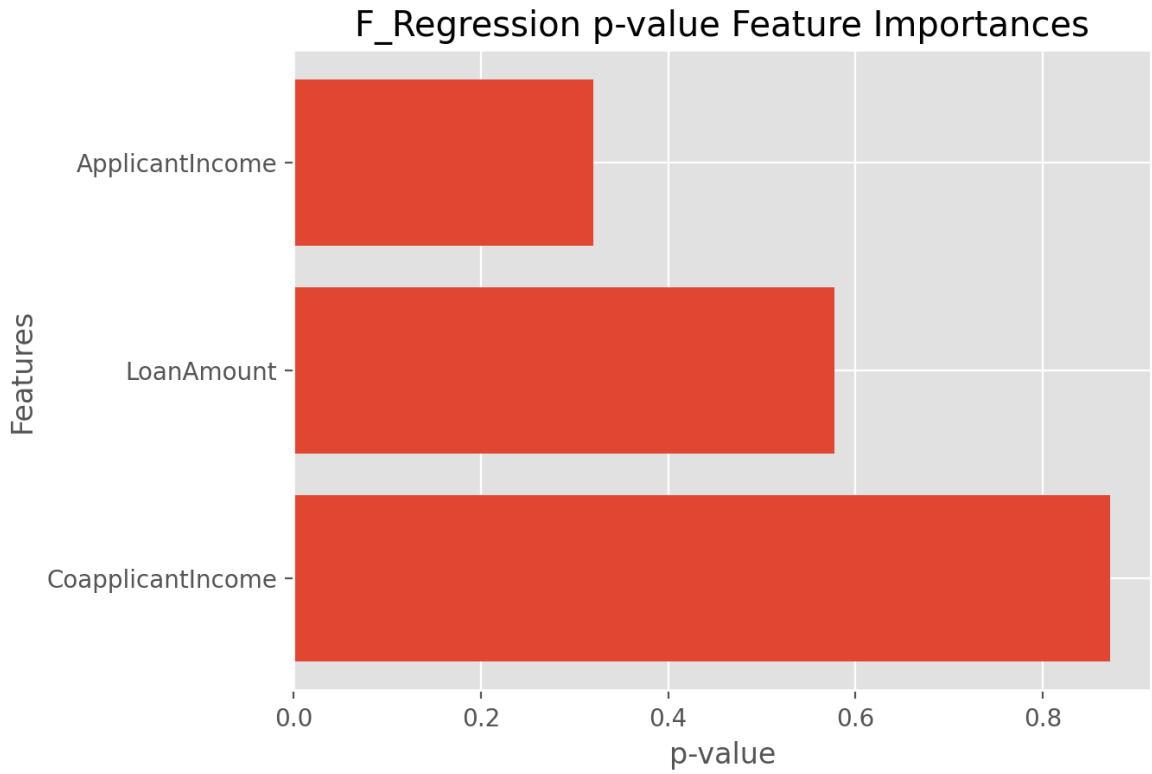
- ApplicantIncome has the highest F-statistic, indicating that it is the most significant predictor among the continuous features. Its high value suggests a strong relationship with loan approval outcomes.
- LoanAmount has moderate importance and CoapplicantIncome has the least importance.

The P-value chart ranks feature by the likelihood that their observed relationship with the target variable could be due to chance. Lower p-values indicate stronger evidence against the null hypothesis (no effect).

- CoapplicantIncome has the highest p-value, suggesting that its relationship with loan approval might not be as strong or consistent as the other variables.
- ApplicantIncome and LoanAmount show lower p-values, therefore they are significant in predicting loan approval outcomes as observed in the F-statistic values.

In [8]:
```
from sklearn.feature_selection import f_regression

cont_feat = Data[["ApplicantIncome", "CoapplicantIncome", "LoanAmount"]]

f_statistic, p_values = f_classif(cont_feat, target)
f_statistic
p_values
```

```
fs_indices_fstat = np.argsort(f_statistic)[::-1]
best_features_fstat = cont_feat.columns[fs_indices_fstat].values
fstatistic_ordered = f_statistic[fs_indices_fstat]

fs_indices_pvalue = np.argsort(p_values)[::-1]
best_features_pvalue = cont_feat.columns[fs_indices_pvalue].values
pvalue_ordered = p_values[fs_indices_pvalue]

plot_imp(best_features_fstat, fstatistic_ordered, 'F_Regression F-Statistic', 'F
plot_imp(best_features_pvalue, pvalue_ordered, 'F_Regression p-value', 'p-value'

Data = Data.drop(columns=['Gender', 'Self_Employed'])
```



3.3 Model Fitting & Tuning

Stratifiedkfold method is used, and the dataset is split into 5 separate folds such that each fold maintains the same percentage of samples of each target class as the whole set. It prevents the model from being biased toward the majority class. During the cross-validation process, each fold is used as a test set, while the remaining 4 folds serve as the training set. The data is shuffled before splitting into folds for reducing variance and making sure that no single class of data is clustered in a single fold. By setting a random state, it is ensured that the random shuffling of data is consistent for each run.

It is used with GridSearchCV for systematically tuning hyperparameters while evaluating model performance across different subsets of the data, ensuring the final model is both robust and generalizable

In [9]:
```
from sklearn.model_selection import StratifiedKFold, GridSearchCV

cv_method = StratifiedKFold(n_splits=5, shuffle=True, random_state=999)
```

Using train_test_split, the data is split into training and testing sets. 30% of the data is allocated to the testing set, and 70% of the data is allocated to the training set. stratify=target ensures that train and test sets have approximately the same percentage of samples of each class as the original dataset.

The dimensions of the training dataset are (667, 11) and the dimensions of the testing set are (286, 11)

In [10]:
```
from sklearn.model_selection import train_test_split

Data_train, Data_test, target_train, target_test = train_test_split(Data, target
                                                                    test_size =
                                                                    stratify = t

print(Data_train.shape)
print(Data_test.shape)
```

```
(667, 11)
(286, 11)
```

KNN Classifier

K-Nearest Neighbors (KNN) classifier using GridSearchCV is used to test different combinations of the hyperparameters “n_neighbors” and p (the power parameter for the Minkowski metric) to identify the combination that gives the best predictive performance.

n_neighbors values (1, 5, 10, 15, 20) have been tested to see their impact on model performance.

Manhattan distance (p=1): This is the sum of the absolute differences of their Cartesian coordinates. It is more suitable for high-dimensional data.

Euclidean distance (p=2): This is the square root of the sum of the squared differences between Cartesian coordinates. It is the most common distance metric for KNN and is suitable in lower-dimensional spaces.

The hyperparameter with n_neighbors=1 and p=2 shows the highest mean score (0.8018). The range in its scores, from 0.706 to 0.862, suggests some variability in performance across different data splits but generally high effectiveness. This means that using the Euclidean distance with only the closest neighbor gives the best performance for area under the ROC curve. This means that the model benefits from making decisions based on the most immediate examples in the training set, rather than generalizing based on a larger neighborhood.

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

params_KNN = [{'n_neighbors': [1, 5, 10, 15, 20],
                   'p': [1, 2]}]

knn = KNeighborsClassifier()

gs_knn = GridSearchCV(estimator=knn,
                          param_grid=params_KNN,
                          cv=cv_method,
                          refit=True,
                          n_jobs=-2,
                          scoring='roc_auc',
                          verbose=1)

gs_knn.fit(Data_train, target_train)

def get_search_results(gs):

    def model_result(scores, params):
        scores = {'mean_score': np.mean(scores),
             'std_score': np.std(scores),
             'min_score': np.min(scores),
             'max_score': np.max(scores)}
        return pd.Series({**params,**scores})

    models = []
    scores = []

    for i in range(gs.n_splits_):
        key = f"split{i}_test_score"
        r = gs.cv_results_[key]
        scores.append(r.reshape(-1,1))

    all_scores = np.hstack(scores)
    for p, s in zip(gs.cv_results_['params'], all_scores):
        models.append((model_result(s, p)))

    pipe_results = pd.concat(models, axis=1).T.sort_values(['mean_score'], ascen

    columns_first = ['mean_score', 'std_score', 'max_score', 'min_score']
    columns = columns_first + [c for c in pipe_results.columns if c not in colum

    return pipe_results[columns]
```

```
results_KNN = get_search_results(gs_knn)
results_KNN
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[11]:

	mean_score	std_score	max_score	min_score	n_neighbors	p
1	0.801982	0.053941	0.862532	0.706848	1.0	2.0
3	0.795704	0.065873	0.862791	0.670284	5.0	2.0
7	0.793036	0.035885	0.835101	0.729199	15.0	2.0
0	0.792990	0.059256	0.847222	0.685142	1.0	1.0
4	0.790970	0.035021	0.827399	0.724289	10.0	1.0
2	0.787028	0.052624	0.844961	0.689535	5.0	1.0
6	0.783784	0.035869	0.818939	0.716796	15.0	1.0
5	0.779482	0.045210	0.820707	0.692377	10.0	2.0
9	0.778897	0.044587	0.824031	0.699742	20.0	2.0
8	0.766219	0.040632	0.811757	0.697545	20.0	1.0

Performance Metrics Summary

Accuracy (77.27%): Proportion of all predictions that were correct, which is the sum of true positives and true negatives divided by the total observations.

Error (22.73%): It is the complement of accuracy and represents the proportion of predictions that the model got wrong.

Precision (80.48%): Proportion of positive identifications that were actually correct, which is true positives divided by the sum of true positives and false positives. It measures the classifier's exactness.

Recall (87.56%): Proportion of actual positives that were identified correctly. It measures the classifier’s completeness. It is true positives divided by the sum of true positives and false negatives.

F1 Score (83.87%): it is the weighted average of precision and recall. It is calculated as 2((precision * Recall)/(precision + recall))

AUC Score (71.74%): Measure of the ability of the classifier to distinguish between classes. This means that the model is moderately good.

ROC Curve

The ROC curve plots the true positive rate (recall) against the false positive rate at various threshold settings. The closer the curve comes to the upper left corner of the plot, the better the model's performance. The dotted line represents a random guess, and a perfect model would reach the point (0,1), meaning no false positives and a 100% true positive rate. There is a a steep rise at the beginning, which quickly moves the TPR to a

high value with a relatively low increase in the FPR. This indicates that the model is able to achieve high sensitivity (recall) without incurring a significant cost in terms of increased false positives, at least at lower threshold levels. The curve approaches the top left corner but doesn't reach close to the top or far left, suggesting a moderate ability to distinguish between the classes.

Confusion Matrix

The confusion matrix provides a summary of the prediction results comparing the actual and predicted classifications. The number of correct and incorrect predictions are summarized with count values and broken down by each class. In this matrix:

52 true negatives: The model correctly predicted the negative class.

169 true positives: The model correctly predicted the positive class.

41 false positives: The model incorrectly predicted the positive class.

24 false negatives: The model incorrectly predicted the negative class.

Precision vs Threshold Chart

It shows how precision varies with different threshold levels for classifying a positive instance. shows a clear positive trend, indicating that as the threshold for classifying an observation as positive increases, the precision of the model also increases.

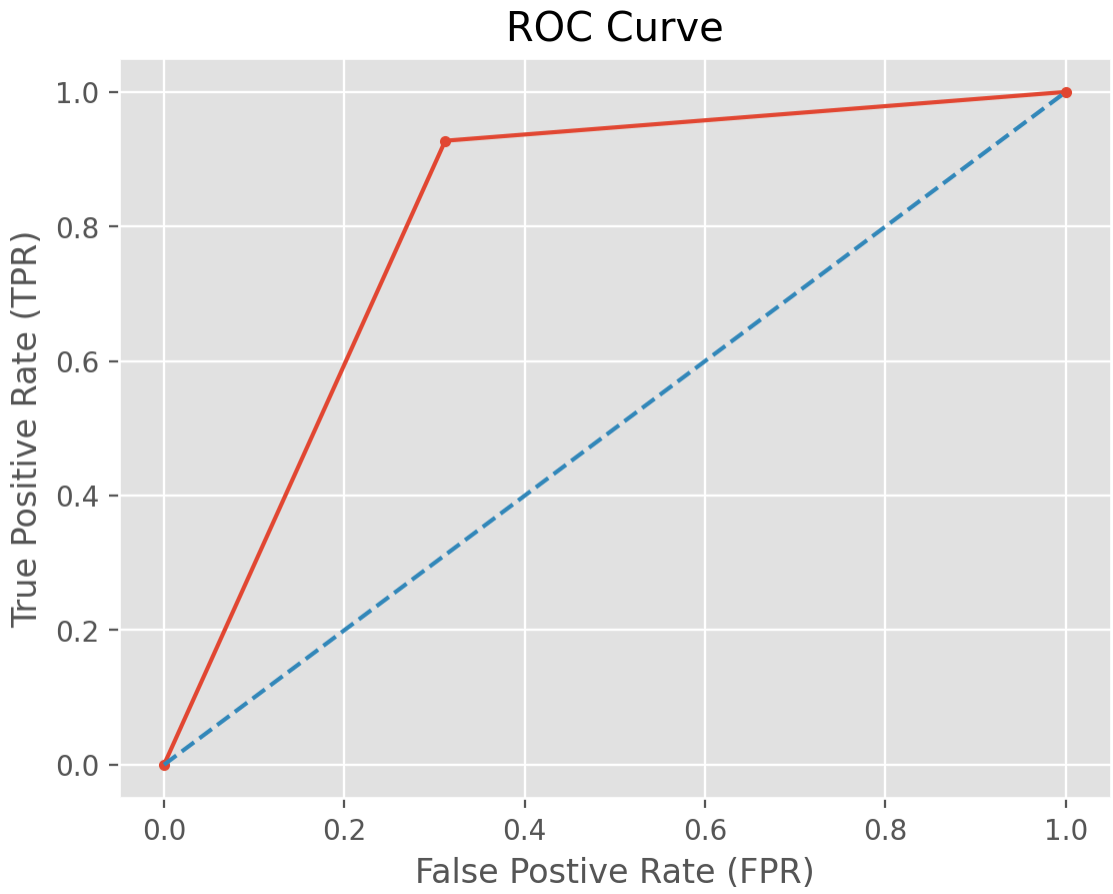
The KNN model shows a reasonable performance in classifying the test data, with good performance in identifying true positives (high recall) and a decent overall accuracy.

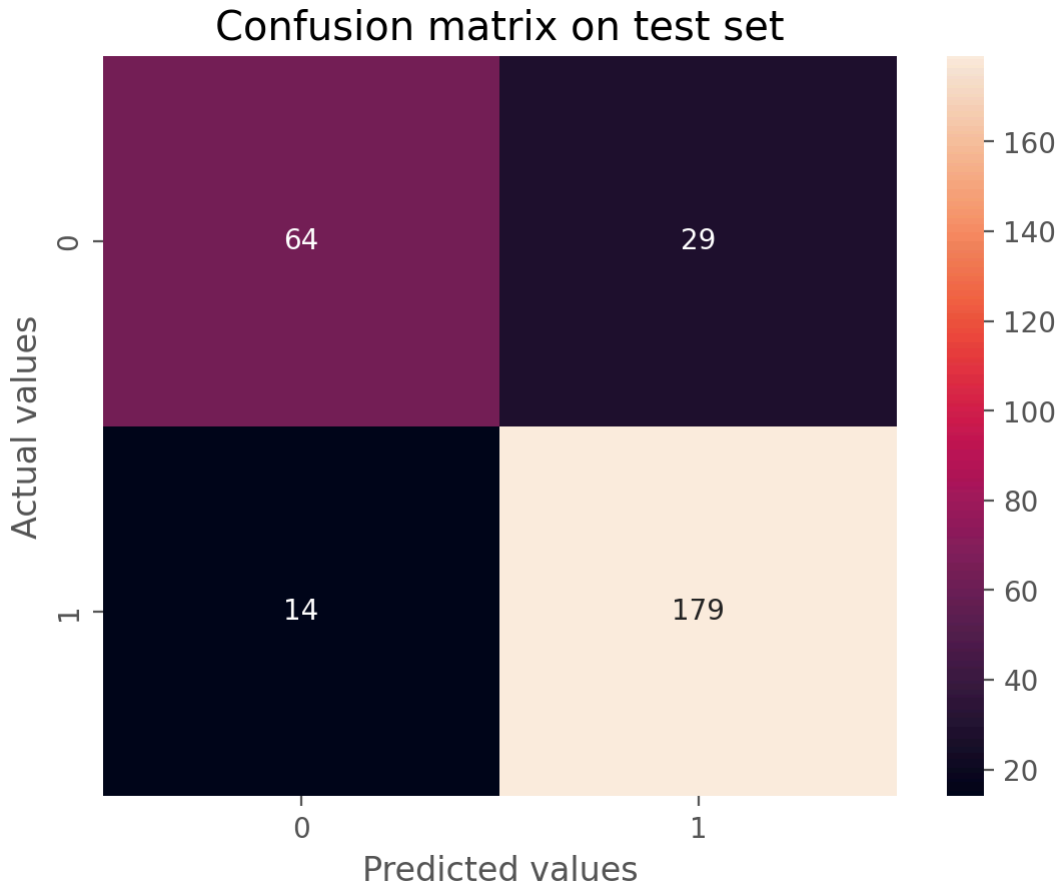
In [12]:
```
from sklearn import metrics
import seaborn as sns

knn_pred = gs_knn.predict(Data_test)

def get_metrics(gs, target_test, pred, test_data):

    accuracy = metrics.accuracy_score(target_test, pred)
    error = 1 - metrics.accuracy_score(target_test, pred)
    precision = metrics.precision_score(target_test, pred)
    recall = metrics.recall_score(target_test, pred)
    f1 = metrics.f1_score(target_test, pred)
    auc = metrics.roc_auc_score(target_test, pred)

    data = {
    'Value': [accuracy, error, precision, recall, f1, auc]
    }

    df = pd.DataFrame(data, index=['Accuracy', 'Error', 'Precision', 'Recall', '
    display(df)

    print(metrics.classification_report(target_test, pred))

    proba = gs.predict_proba(test_data)

    fpr, tpr, _ = metrics.roc_curve(target_test, proba[:, 1])
```

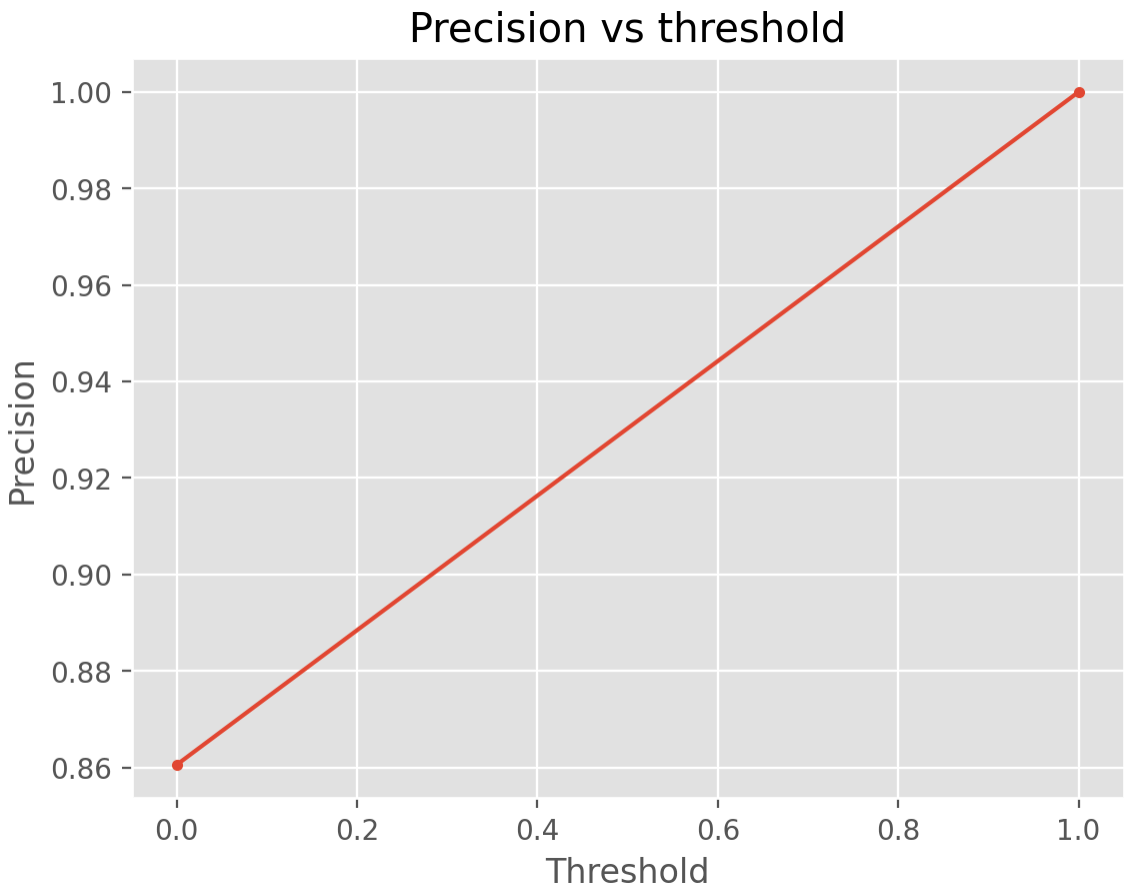
```
    df = pd.DataFrame({'fpr': fpr, 'tpr': tpr})

    ax = df.plot.line(x='fpr', y='tpr', title='ROC Curve', legend=False, marker
    plt.plot([0, 1], [0, 1], '--')
    ax.set_xlabel("False Postive Rate (FPR)")
    ax.set_ylabel("True Positive Rate (TPR)")
    plt.show();

    cf_matrix = metrics.confusion_matrix(target_test, pred)
    ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
    ax.set(xlabel='Predicted values', ylabel='Actual values')
    ax.set_title("Confusion matrix on test set")
    plt.show()

    precision, recall, thresholds = metrics.precision_recall_curve(target_test,
    accuracy = [metrics.accuracy_score(target_test, proba[:,1]>=t) for t in thre
    results = pd.DataFrame({"Precision": precision[1:],
                        "Recall": recall[1:],
                        "Threshold": thresholds,
                        "Accuracy": accuracy})

    ax = results.plot.line(x='Threshold', y='Precision', title='Precision vs thr
    ax.set_xlabel("Threshold")
    ax.set_ylabel("Precision")
    plt.show();

get_metrics(gs_knn, target_test, knn_pred, Data_test)
```

	Value
Accuracy	0.849650
Error	0.150350
Precision	0.860577
Recall	0.927461
F1 Score	0.892768
AUC Score	0.807817

```
              precision    recall  f1-score   support

           0       0.82      0.69      0.75        93
           1       0.86      0.93      0.89       193

    accuracy                           0.85       286
   macro avg       0.84      0.81      0.82       286
weighted avg       0.85      0.85      0.85       286
```


ROC Curve
True Positive Rate (TPR)
False Postive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion matrix on test set
Actual values
Predicted values
64
29
14
179
0
1
160
140
120
100
80
60
40
20



Decision Tree Classifier 1

The tuning of decision tree classifier is done using GridSearchCV, which performs exhaustive search over Decision Tree Classifier. The specified parameters were:

max_depth values of 3, 4, and 5.

min_samples_split values of 2, 3, 4, and 5.

Configurations with a max_depth of 5 have the highest mean scores across all splits, with the best results achieved with a max_depth of 5 and a min_samples_split of 2. The mean scores, standard deviations, max scores, and min scores for these configurations are very similar, indicating that the min_samples_split has little impact on the outcome when max_depth is set to 5. Therefore, increasing the tree depth will allow the model to capture more complex patterns.

In [13]:
```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier(criterion='gini', random_state=111)

params_DT = {'max_depth': [3, 4, 5],
                  'min_samples_split': [2, 3, 4, 5]}

gs_DT = GridSearchCV(estimator=DT,
                           param_grid=params_DT,
                           cv=cv_method,
                           refit=True,
                           n_jobs=-2,
```

```
                           scoring='roc_auc',
                           verbose=1)

gs_DT.fit(Data_train, target_train);

results_DT = get_search_results(gs_DT)
results_DT
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[13]:

	mean_score	std_score	max_score	min_score	max_depth	min_samples_split
8	0.802274	0.037570	0.857828	0.745995	5.0	2.0
9	0.802274	0.037570	0.857828	0.745995	5.0	3.0
10	0.800698	0.037271	0.857828	0.745995	5.0	4.0
11	0.800698	0.037271	0.857828	0.745995	5.0	5.0
4	0.774004	0.052230	0.833207	0.678553	4.0	2.0
5	0.774004	0.052230	0.833207	0.678553	4.0	3.0
6	0.774004	0.052230	0.833207	0.678553	4.0	4.0
7	0.774004	0.052230	0.833207	0.678553	4.0	5.0
0	0.754870	0.062966	0.809091	0.633075	3.0	2.0
1	0.754870	0.062966	0.809091	0.633075	3.0	3.0
2	0.754870	0.062966	0.809091	0.633075	3.0	4.0
3	0.754870	0.062966	0.809091	0.633075	3.0	5.0

Performance Metrics Summary

- Accuracy (68.53%): Indicates a moderate level of overall correctness of the model.
- Precision (81.21%): High precision reflects that when the model predicts an instance as positive, it is correct most of the time.
- Recall (69.43%): Shows that the model is able to identify approximately 69% of all actual positives.
- F1 Score (74.86%): indicates a balance between these metrics.
- Error Rate (31.47%): Complements the accuracy, showing the proportion of all predictions that were incorrect, which is moderate.

ROC Curve

The curve generally stays above the diagonal line (indicating better than random performance), showing a good balance between the True Positive Rate (TPR) and False Positive Rate (FPR).

The curve approaches closer to the top-left corner but doesn't reach it, indicating a moderate ability to discriminate between the classes, which is also reflected in the AUC score of 0.680.

Confusion Matrix

- True Positives (134)
- True Negatives (62)
- False Positives (31)
- False Negatives (59)

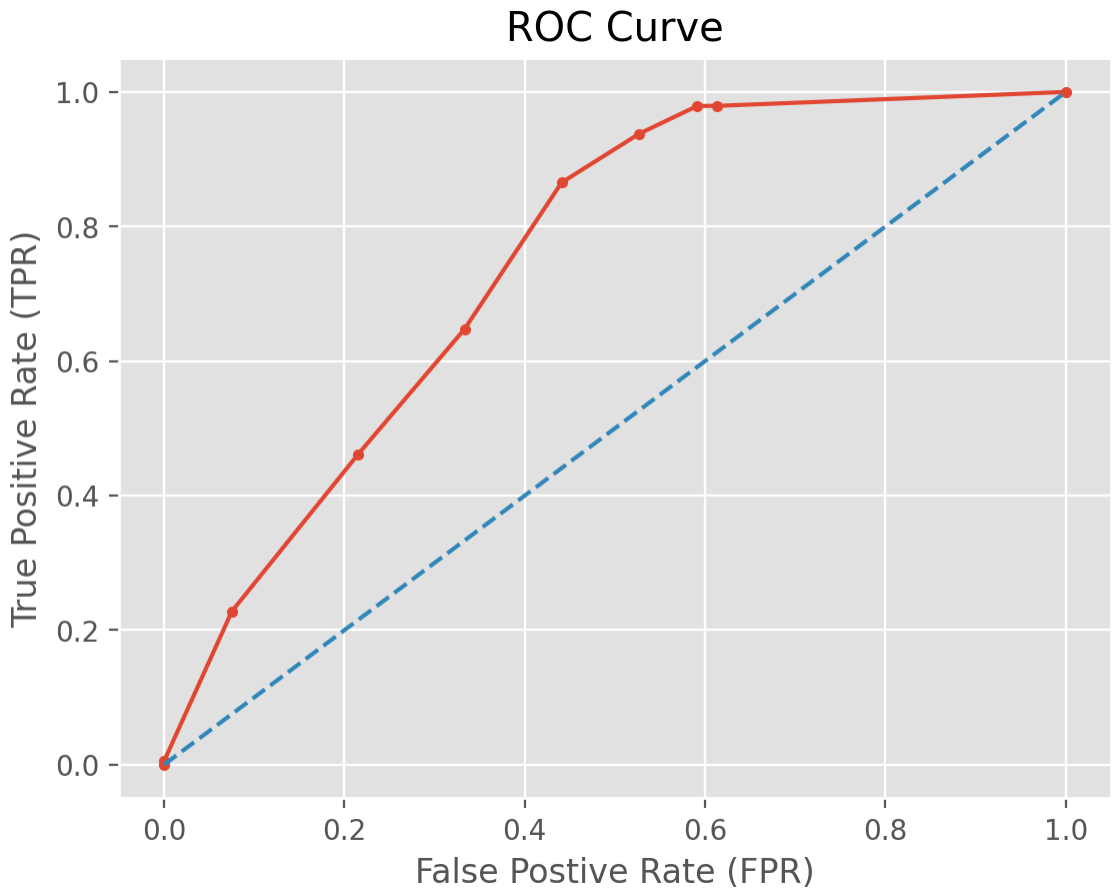
Precision vs Threshold Chart

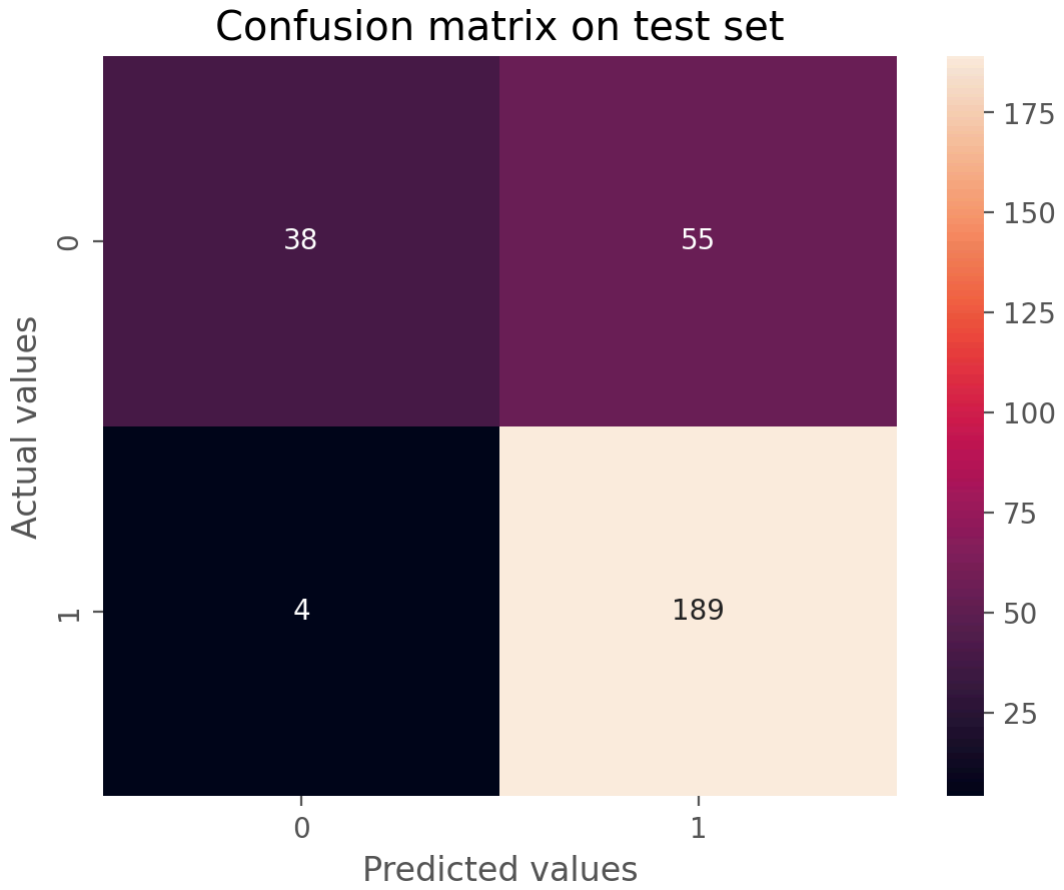
A significant spike in precision at a threshold near 1 indicates that the model is highly confident in a very small number of cases.

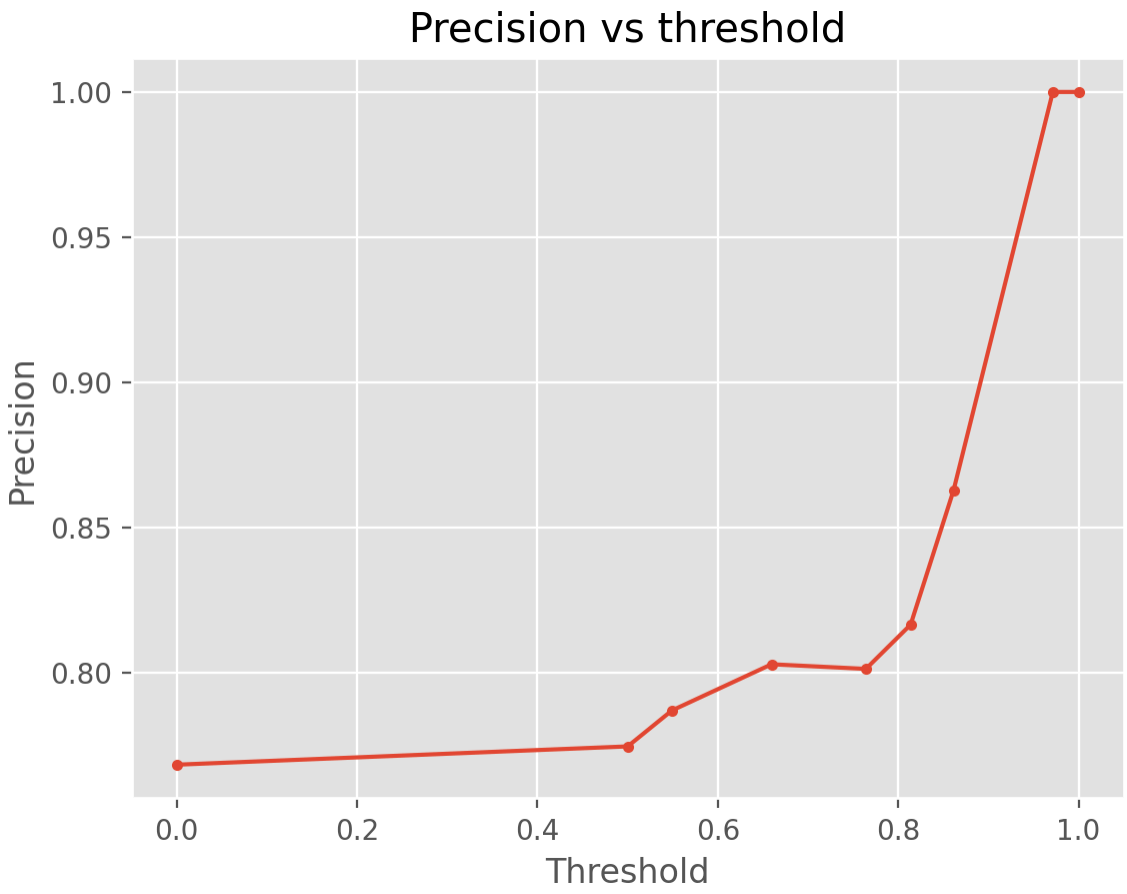
In [14]:
```
dt_pred = gs_DT.predict(Data_test)
get_metrics(gs_DT, target_test, dt_pred, Data_test)
```

	Value
Accuracy	0.793706
Error	0.206294
Precision	0.774590
Recall	0.979275
F1 Score	0.864989
AUC Score	0.693938

```
              precision    recall  f1-score   support

           0       0.90      0.41      0.56        93
           1       0.77      0.98      0.86       193

    accuracy                           0.79       286
   macro avg       0.84      0.69      0.71       286
weighted avg       0.82      0.79      0.77       286
```


ROC Curve
True Positive Rate (TPR)
False Postive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion matrix on test set
Actual values
Predicted values
38
55
4
189
0
1
175
150
125
100
75
50
25



Decision Tree Classifier 2

Since the best performing decision tree classifier above used a max_depth of 5 (which was the maximum max_depth value modelled), there is a possibility that the performance can be improved by further increasing max_depth. Therefore, a new set of decision tree classifiers are modelled with max_deoth set to 5, 8, 10, 13, and 15.

max_depth=5 produced the best mean score in the previous one, and only [3, 4, 5] were tested for max_depth. Thus, the max_depth increased to test if a better result can be obtained with [5, 8, 10, 13, 15] for max_depth.

The best results were observed with a max_depth of 10 and min_samples_split of 3, giving a mean ROC AUC score of 0.836589. This indicated a balanced complexity that captured sufficient data characteristics without fitting noise.

In [15]:
```
DT = DecisionTreeClassifier(criterion='gini', random_state=111)

params_DT = {'max_depth': [5, 8, 10, 13, 15],
                  'min_samples_split': [2, 3, 4, 5]}

gs_DT = GridSearchCV(estimator=DT,
                           param_grid=params_DT,
                           cv=cv_method,
                           refit=True,
                           n_jobs=-2,
                           scoring='roc_auc',
                           verbose=1)
```

```
gs_DT.fit(Data_train, target_train);

results_DT = get_search_results(gs_DT)
results_DT
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[15]:

	mean_score	std_score	max_score	min_score	max_depth	min_samples_split
9	0.833533	0.057917	0.877146	0.719380	10.0	3.0
4	0.832157	0.050198	0.898106	0.755039	8.0	2.0
10	0.831403	0.052663	0.888510	0.732558	10.0	4.0
11	0.830907	0.048956	0.895581	0.745866	10.0	5.0
5	0.829247	0.053065	0.894949	0.745736	8.0	3.0
8	0.828966	0.057600	0.887500	0.719380	10.0	2.0
7	0.828581	0.047546	0.891035	0.759044	8.0	5.0
6	0.825170	0.052467	0.891035	0.743540	8.0	4.0
15	0.819755	0.053035	0.893182	0.727907	13.0	5.0
19	0.814469	0.057917	0.895707	0.716021	15.0	5.0
14	0.812949	0.046616	0.878914	0.733075	13.0	4.0
13	0.811990	0.050055	0.882449	0.725581	13.0	3.0
12	0.810145	0.061572	0.898990	0.707494	13.0	2.0
18	0.809714	0.050252	0.884343	0.727003	15.0	4.0
1	0.802274	0.037570	0.857828	0.745995	5.0	3.0
0	0.802274	0.037570	0.857828	0.745995	5.0	2.0
16	0.801576	0.056075	0.881566	0.709302	15.0	2.0
3	0.800698	0.037271	0.857828	0.745995	5.0	5.0
2	0.800698	0.037271	0.857828	0.745995	5.0	4.0
17	0.797367	0.058398	0.880934	0.703230	15.0	3.0

Performance Metrics Summary

- Accuracy (68.53%)
- Precision (81.21%)
- Recall (69.43%)
- F1 Score (74.86%)

ROC Curve

The ROC curve shows that the Decision Tree model has a moderate ability to distinguish between the positive and negative classes. The AUC score of 0.680 indicates moderate performance as the curve does not reach close to the top left corner.

Precision vs Threshold

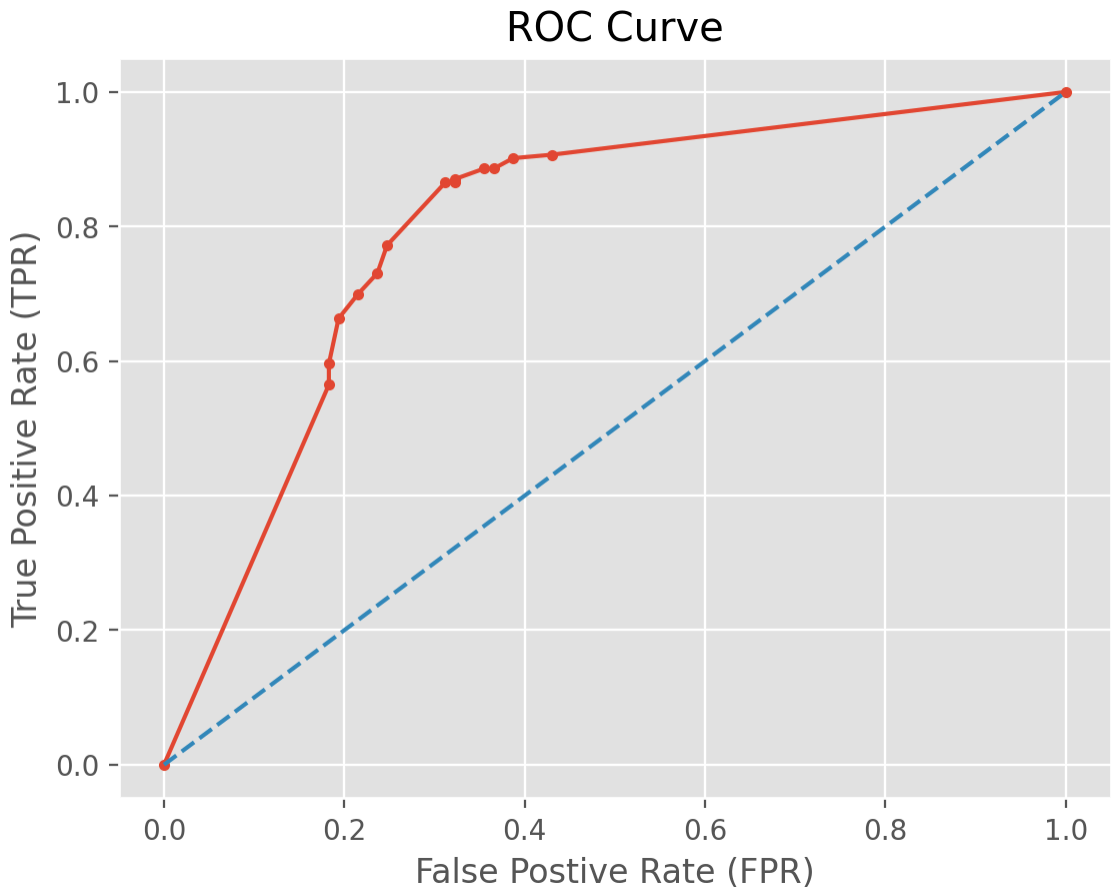
The Precision vs. Threshold graph shows an increase in precision as the decision threshold increases, peaking sharply near the threshold of 1.0. This suggests that the Decision Tree becomes highly precise but at the expense of recall as the threshold approaches 1.

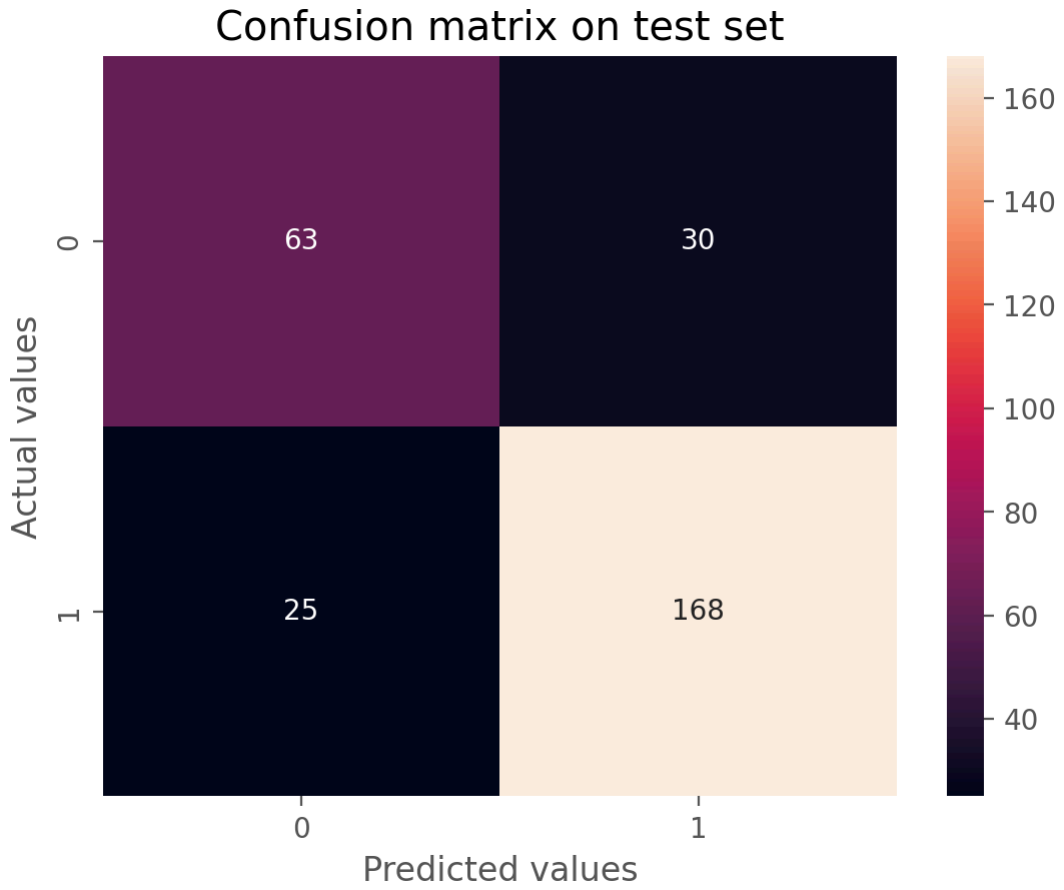
Comparing the KNN model with the Decision Tree, it is observed that KNN outperforms decision tree model in terms of accuracy, recall, F1 score and AUC score. Precision is similar with KNN being slightly lower. Therefore, KNN model has better performance compared to the decision tree.

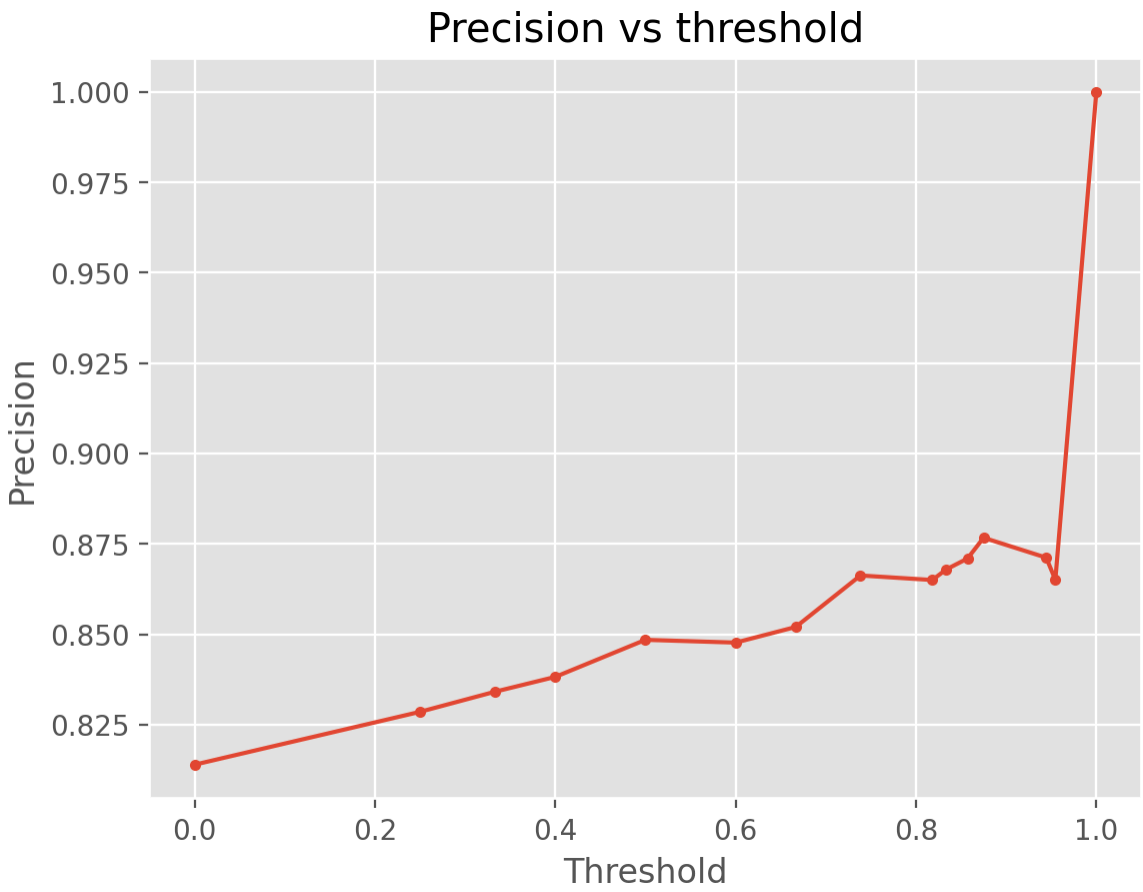
In [16]:
```
dt_pred = gs_DT.predict(Data_test)
get_metrics(gs_DT, target_test, dt_pred, Data_test)
```

	Value
Accuracy	0.807692
Error	0.192308
Precision	0.848485
Recall	0.870466
F1 Score	0.859335
AUC Score	0.773943

```
              precision    recall  f1-score   support

           0       0.72      0.68      0.70        93
           1       0.85      0.87      0.86       193

    accuracy                           0.81       286
   macro avg       0.78      0.77      0.78       286
weighted avg       0.81      0.81      0.81       286
```


ROC Curve
True Positive Rate (TPR)
False Postive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion matrix on test set
Actual values
Predicted values
63
30
25
168
0
1
160
140
120
100
80
60
40



Gaussian Naive Bayes Classifier

Power transformation is used to transform continuous features resemble a normal distribution, which is an assumption of the Gaussian NB algorithm. The Gaussian Naive Bayes model is then fitted using the transformed data. Histograms for continuous features before and after transformation clearly show the better normal distribution in the transformed data. Using np.logspace(1,-3, num=200) generates 200 values for var_smoothing from 10 to 0.001.

Parameter with var_smoothing = 0.089074 gives the best result with a mean_score of 0.809675 with a standard deviation of 0.023405. This helped the Gaussian Naive Bayes model achieve the best balance between bias and variance.

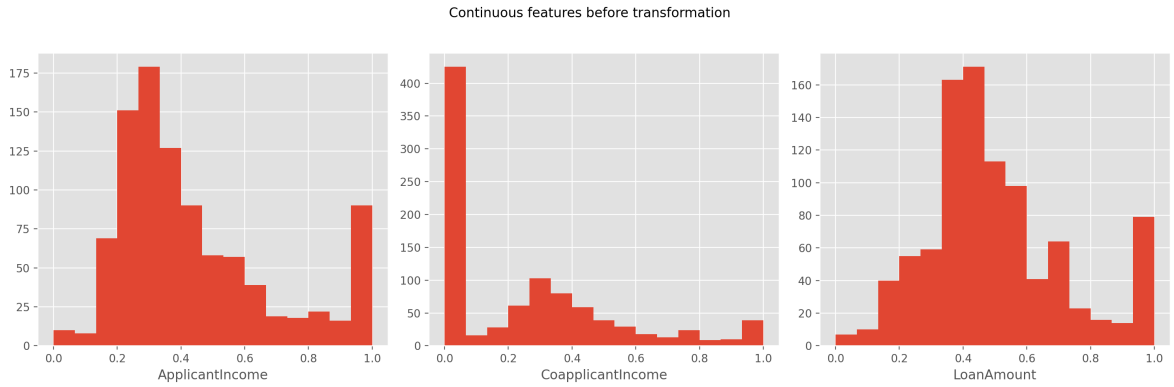
In [17]:
```
from sklearn.preprocessing import PowerTransformer
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import RandomizedSearchCV

fig, axes = plt.subplots(nrows=1, ncols=3, sharey=False, figsize=(15, 5))
plt.subplots_adjust(top=3, bottom=2)


for i, col in enumerate(cont_feat):
    cont_feat[col].hist(ax=axes[i], bins = 15).set_xlabel(col)
    fig.suptitle("\n".join(["Continuous features before transformation"]))
    fig.tight_layout(rect=[0, 0, 1, 0.96])

cont_transformed = PowerTransformer(standardize=False).fit_transform(Data_train[
```

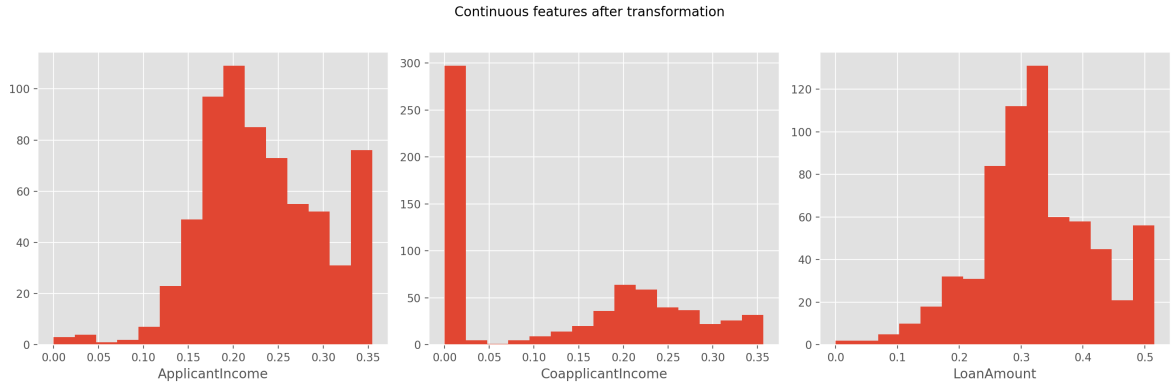
```
df_cont_transformed = pd.DataFrame(cont_transformed, columns=cont_feat.columns)

fig, axes = plt.subplots(nrows=1, ncols=3, sharey=False, figsize=(15, 5))
plt.subplots_adjust(top=5, bottom=4)
for i, col in enumerate(df_cont_transformed):
    df_cont_transformed[col].hist(ax=axes[i], bins = 15).set_xlabel(col)
    fig.suptitle("\n".join(["Continuous features after transformation"]))
    fig.tight_layout(rect=[0, 0, 1, 0.96])

Data_train_transformed = Data_train
df_cont_transformed.index = Data_train_transformed.index
Data_train_transformed[cont_feat.columns] = df_cont_transformed

params_NB = {'var_smoothing': np.logspace(1,-3, num=200)}

NB = GaussianNB()

n_iter_search = 20
gs_NB = RandomizedSearchCV(estimator=NB,
                           param_distributions=params_NB,
                           cv=cv_method,
                           refit=True,
                           n_jobs=-2,
                           scoring='roc_auc',
                           n_iter=n_iter_search,
                           verbose=1)

gs_NB.fit(Data_train_transformed.values, target_train.values)

results_NB = get_search_results(gs_NB)
results_NB
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[17]:

	mean_score	std_score	max_score	min_score	var_smoothing
16	0.809873	0.025717	0.837626	0.762791	0.074020
11	0.808988	0.015907	0.835859	0.786047	3.292971
3	0.808937	0.015968	0.835859	0.785788	3.612343
10	0.808833	0.015932	0.835859	0.786047	3.144035
14	0.808677	0.015630	0.836111	0.788372	7.232634
9	0.806078	0.022582	0.835859	0.767183	0.283310
19	0.805841	0.019163	0.835859	0.775452	1.304902
1	0.805837	0.020473	0.835354	0.771318	0.517092
15	0.805833	0.021102	0.835101	0.769767	0.450056
7	0.805788	0.019897	0.835354	0.772868	0.714943
4	0.805736	0.020064	0.835354	0.772351	0.682607
0	0.805568	0.022069	0.836111	0.767959	0.310787
8	0.804935	0.029007	0.836869	0.750646	0.035297
17	0.804312	0.030606	0.837374	0.746770	0.032176
5	0.801470	0.033305	0.837626	0.739018	0.011623
12	0.801114	0.032992	0.835354	0.739018	0.001320
13	0.801061	0.033139	0.835606	0.738760	0.001517
6	0.800858	0.033076	0.835101	0.738501	0.002899
18	0.800650	0.033507	0.835354	0.737468	0.004009
2	0.800444	0.033982	0.835606	0.736176	0.006986





Performance Metrics Summary

Since the model was trained on transformed data, Gaussian transformation is performed on the test data before making predictions.

- Accuracy: 77.97% indicates a good overall performance.
- Precision: 76.42%
- Recall: 97.41% shows that the model is very effective at identifying the positive class, but the trade-off is a lower precision.
- F1 Score: 85.65% is a balanced measure of precision and recall, showing the model's robustness despite the precision-recall trade-off.
- AUC Score: 67.52% suggests the model's ability to distinguish between the classes could be improved

ROC Curve

The ROC curve shows a steady increase and a high area under the curve (AUC), indicating that the model distinguishes well between the two classes for a wide range of thresholds.

Confusion Matrix

- True Negatives: 35
- False Positives: 58
- False Negatives: 5
- True Positives: 188

Precision vs Threshold

The precision is relatively stable across a wide range of thresholds but then drops dramatically as the threshold approaches 1.

Comparing the Gaussian Naive Bayes model to KNN, it is observed that accuracy scores are very similar, while KNN shows higher precision. Naïve Bayes has higher recall and slightly higher F1 Score. KNN has a higher AUC score. NB has a higher tendency for false positives but very few false negatives, while KNN has more balanced error distribution between false positives and false negatives.

In [19]:
```
cont_transformed_test = PowerTransformer(standardize=False).fit_transform(Data_t
df_cont_transformed_test = pd.DataFrame(cont_transformed_test, columns=cont_feat
Data_test_transformed = Data_test
```

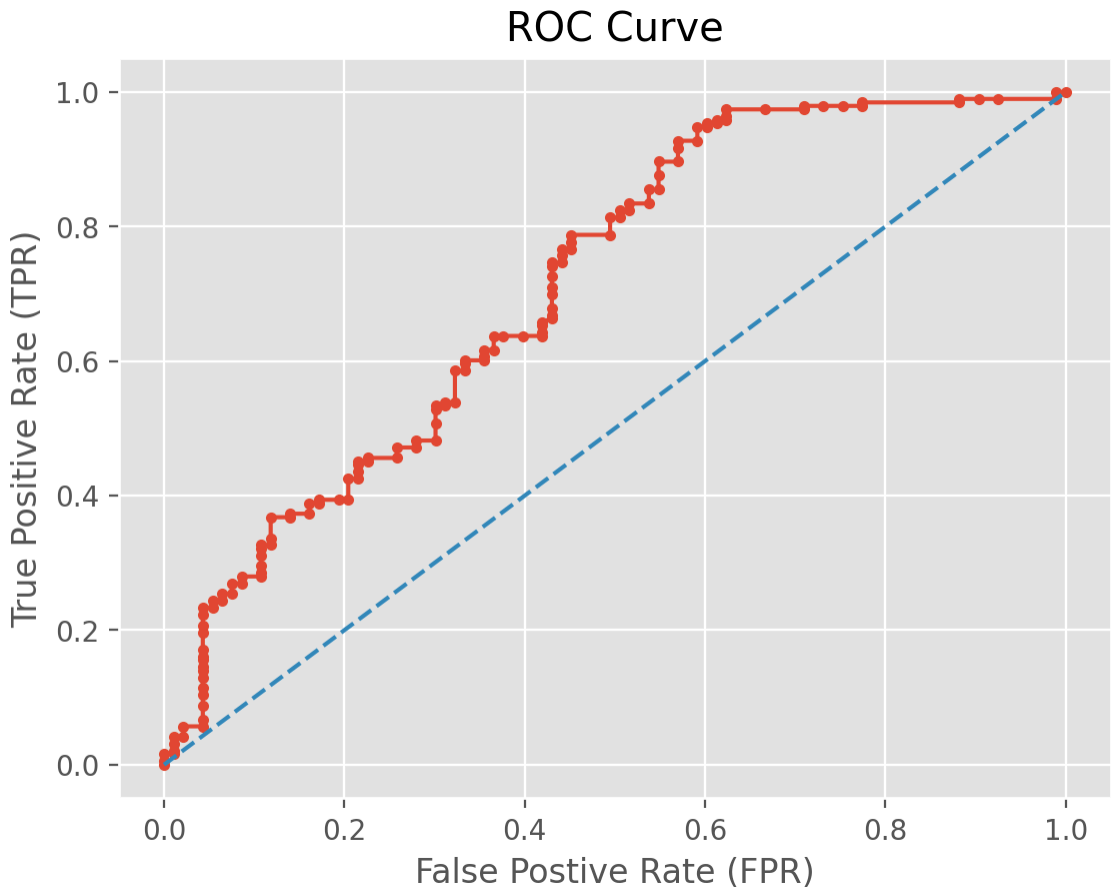
```
df_cont_transformed_test.index = Data_test_transformed.index
Data_test_transformed[cont_feat.columns] = df_cont_transformed_test

NB_pred = gs_NB.predict(Data_test_transformed)
get_metrics(gs_NB, target_test, NB_pred, Data_test_transformed)
```

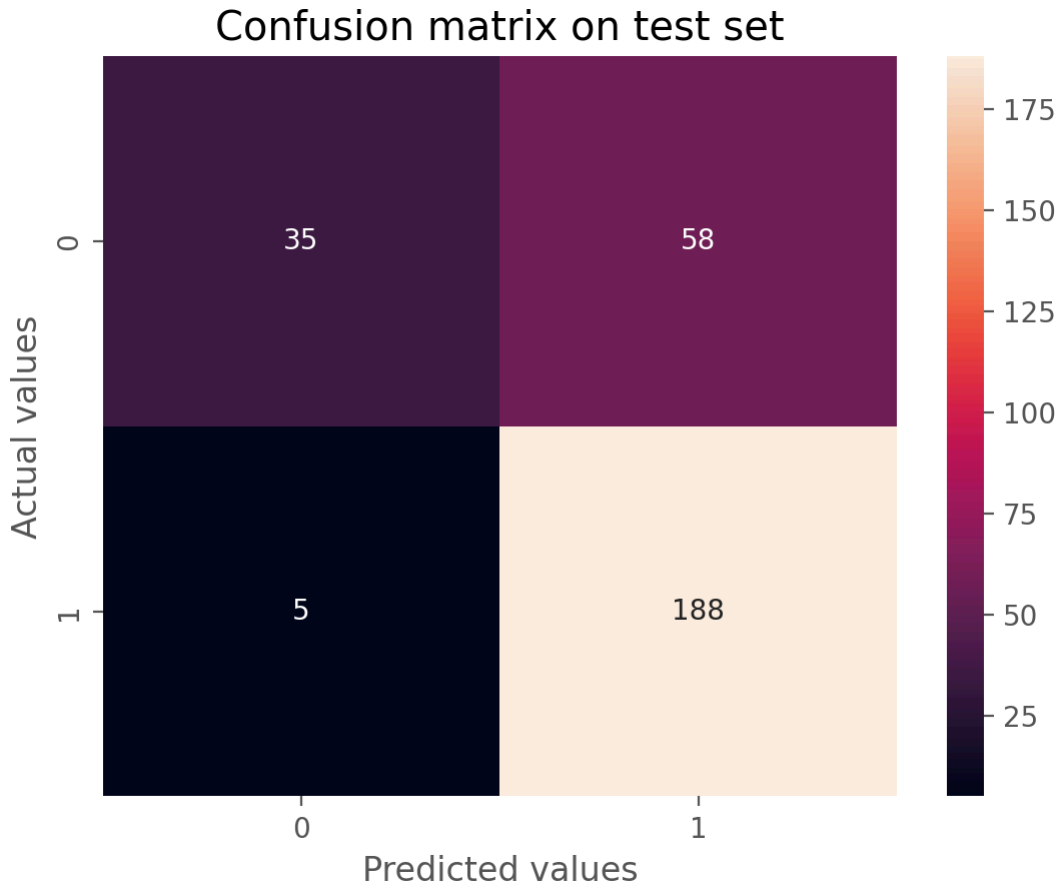
```
C:\Users\Kavinda Goonesekere\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but GaussianNB was fitted without feature names
  warnings.warn(
```

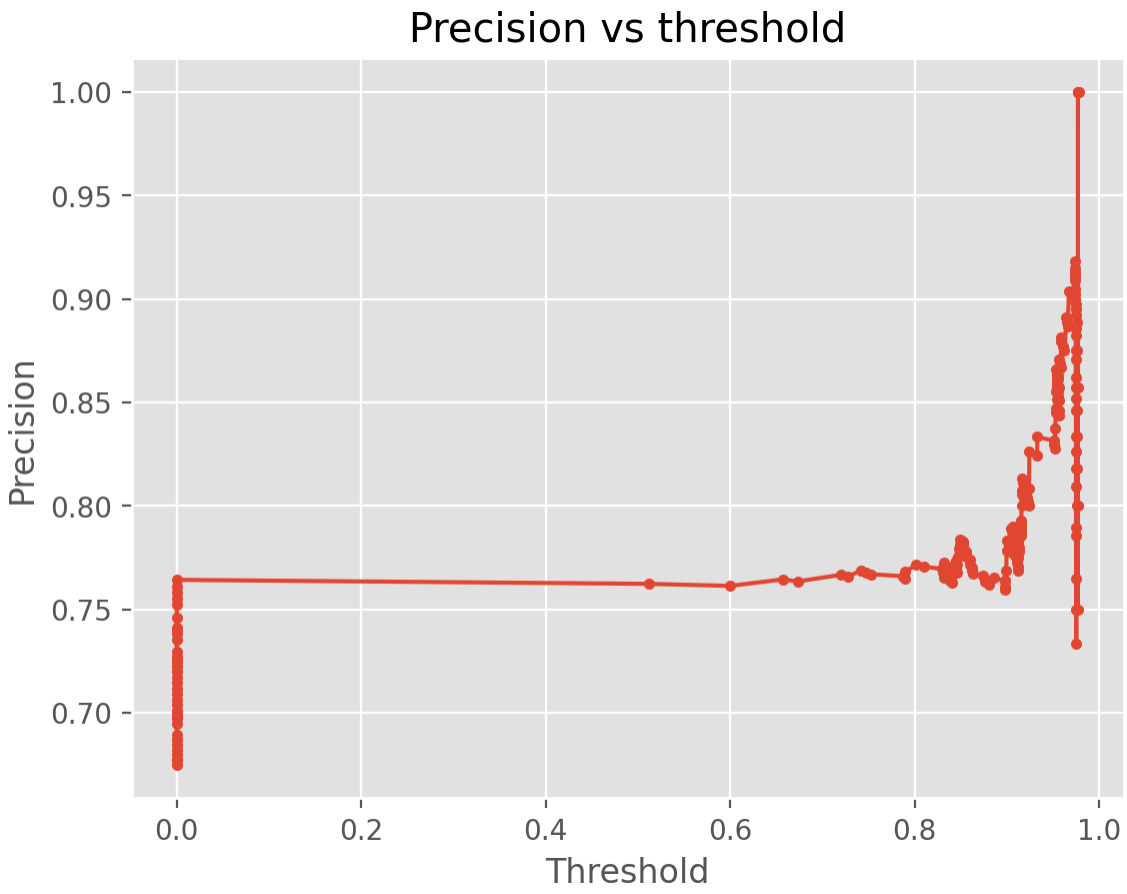
	Value
Accuracy	0.779720
Error	0.220280
Precision	0.764228
Recall	0.974093
F1 Score	0.856492
AUC Score	0.675219

```
              precision    recall  f1-score   support

           0       0.88      0.38      0.53        93
           1       0.76      0.97      0.86       193

    accuracy                           0.78       286
   macro avg       0.82      0.68      0.69       286
weighted avg       0.80      0.78      0.75       286
```

```
C:\Users\Kavinda Goonesekere\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but GaussianNB was fitted without feature names
  warnings.warn(
```


ROC Curve
True Positive Rate (TPR)
False Postive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion matrix on test set
Actual values
Predicted values
35
58
5
188
0
1
175
150
125
100
75
50
25



Random Forest Classifier

N estimators tested were 100, 250, and 500 and max depths tested were 5, 7, 10, and 12.

From the grid search, the best performing hyperparameter combination was Max Depth = 12 and Number of Estimators = 100

This combination achieved the highest mean ROC-AUC score of 0.896647, indicating strong model performance with the ability to discriminate between the classes effectively. The standard deviation was relatively low at 0.041710, suggesting consistent performance across different folds in cross-validation. The maximum and minimum ROC-AUC scores were 0.931008 and 0.815375, confirming its robustness. This means that deeper trees and a moderate number of estimators provided the best results. increasing the number of estimators after 100 did not improve performance significantly.

In [504…

```
RF = RandomForestClassifier(random_state=999)

params_RF = {'n_estimators': [100, 250, 500],
             'max_depth': [5, 7, 10, 12]}

gs_RF = GridSearchCV(estimator=RF,
                     param_grid=params_RF,
                     cv=cv_method,
                     verbose=1,
                     n_jobs=-2,
                     scoring='roc_auc')

gs_RF.fit(Data_train, target_train)
```

```
results_RF = get_search_results(gs_RF)
results_RF
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[504…

	mean_score	std_score	max_score	min_score	max_depth	n_estimators
9	0.896647	0.041710	0.931008	0.815375	12.0	100.0
10	0.894607	0.040464	0.928424	0.816021	12.0	250.0
11	0.894064	0.044575	0.924548	0.805943	12.0	500.0
8	0.890624	0.045623	0.920930	0.800000	10.0	500.0
7	0.890093	0.044079	0.921964	0.802584	10.0	250.0
6	0.888664	0.047504	0.923773	0.794832	10.0	100.0
3	0.877576	0.045971	0.908010	0.786305	7.0	100.0
4	0.875576	0.045723	0.903283	0.784496	7.0	250.0
5	0.874520	0.047276	0.909091	0.780879	7.0	500.0
2	0.846056	0.049593	0.882576	0.747804	5.0	500.0
1	0.844759	0.049926	0.884848	0.746512	5.0	250.0
0	0.840558	0.053569	0.883207	0.735401	5.0	100.0

Performance Metrics Summary

- Accuracy (80.77%): A high percentage of predictions are correct.
- Precision (81.08%) and Recall (93.26%): High precision coupled with high recall indicates that the model is returning accurate results and returning a majority of all positive results.
- F1 Score (86.75%): Harmonic mean of precision and recall, which is quite high, suggesting a balanced classifier.
- AUC Score (74.05%): Reflects the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one, with a good score of over 70%.

ROC Curve

This curve demonstrates excellent model performance, almost reaching the top left corner, indicating a high true positive rate with a low false positive rate. The area under the ROC curve (AUC) appears to be quite high, which is excellent for predictive accuracy.

Precision vs Threshold

Initially, precision starts at a lower value and gradually increases, indicating that as the threshold for predicting a positive class is raised, the model becomes more conservative but more precise. There is a sudden spike near the threshold of 1.

Confusion Matrix

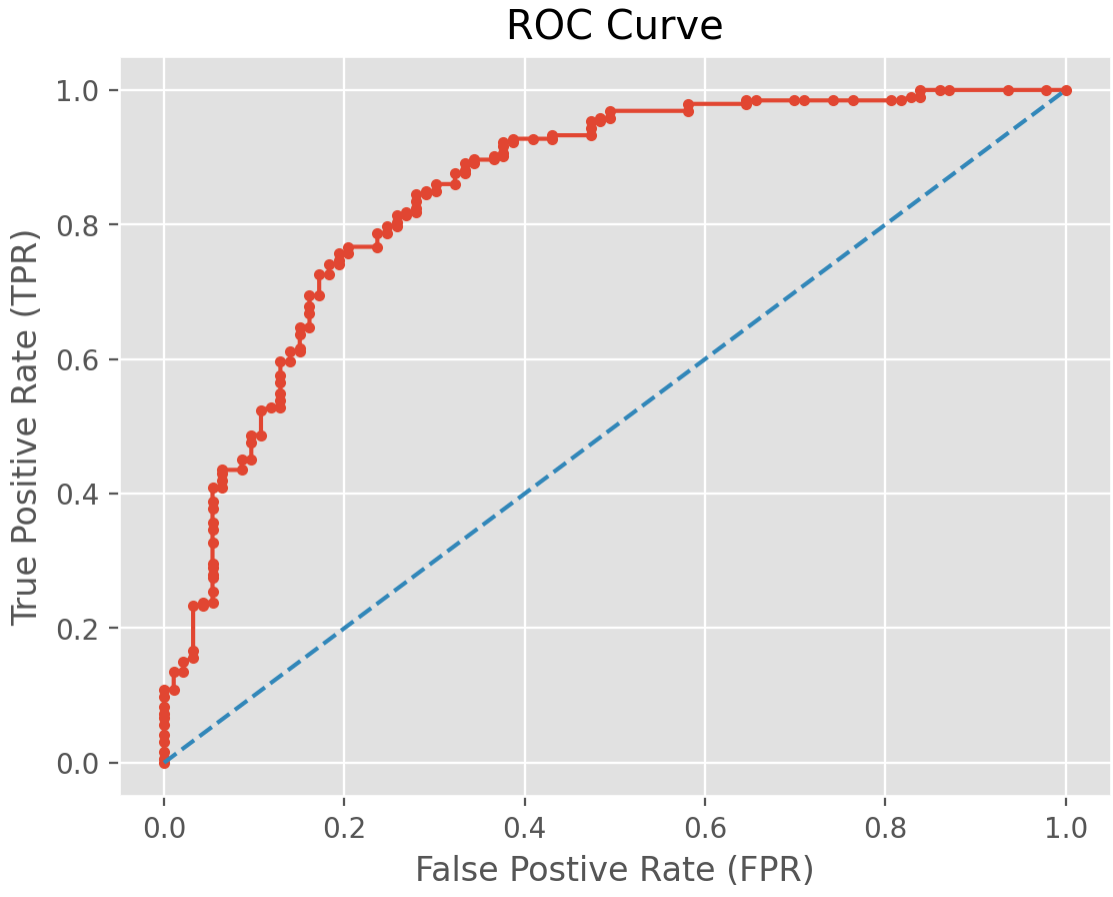
The model predicted more true positives (180) and true negatives (51) correctly, while it misclassified 42 false positives and 13 false negatives, which is a smaller number, confirming the high values seen in recall and precision metrics.

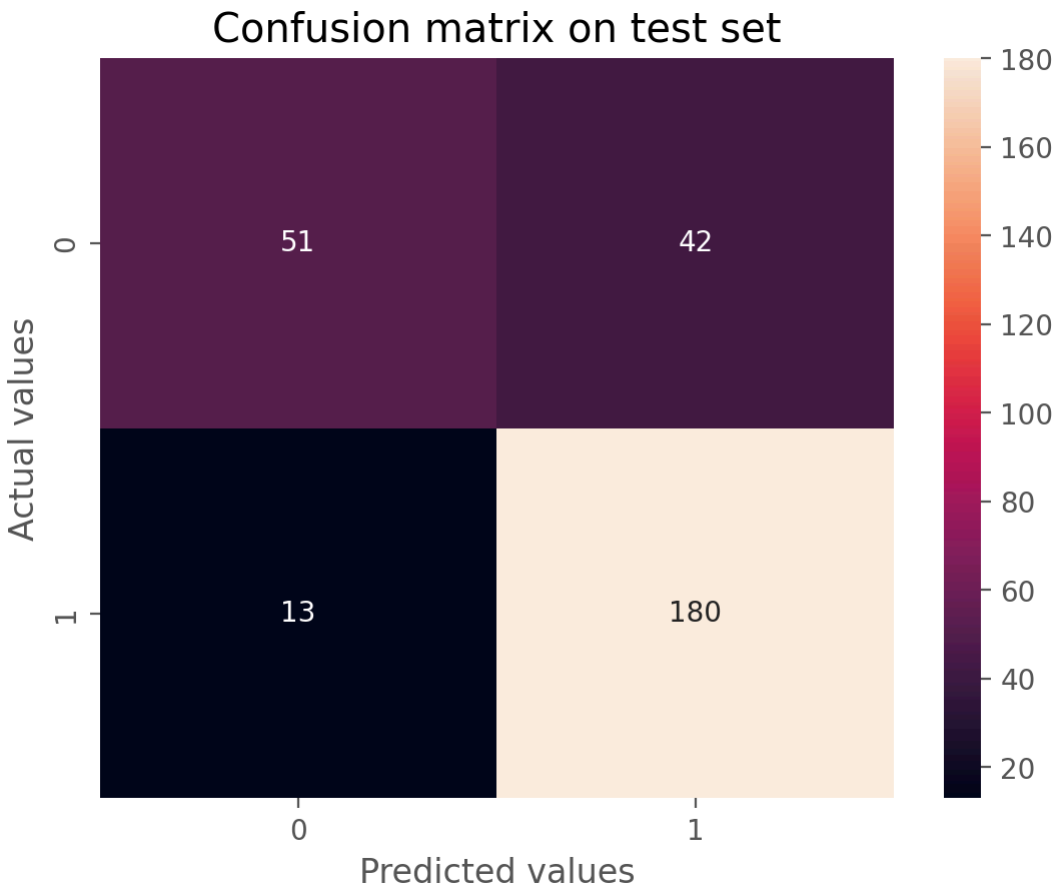
In [524…
```
RF_pred = gs_RF.predict(Data_test)
get_metrics(gs_RF, target_test, RF_pred, Data_test)
```

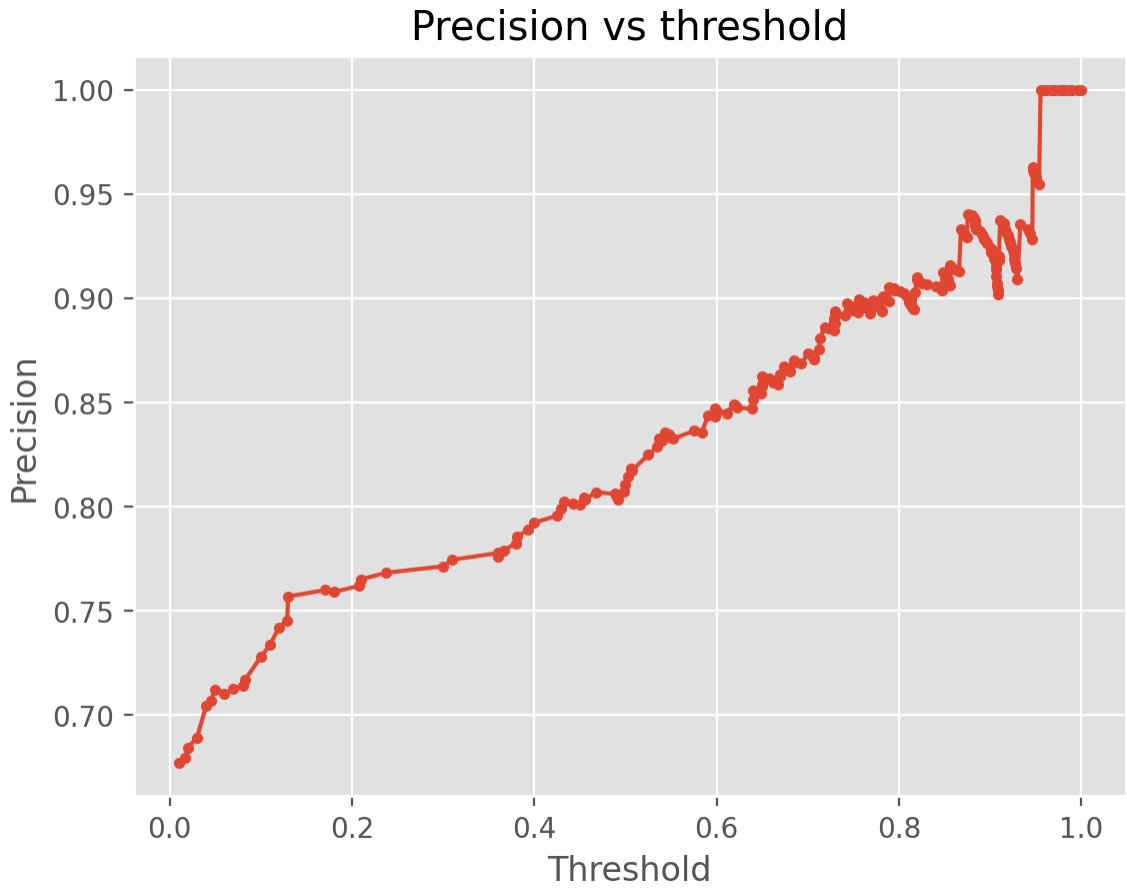
```
[[ 51  42]
 [ 13 180]]
```

	Value
Accuracy	0.807692
Error	0.192308
Precision	0.810811
Recall	0.932642
F1 Score	0.867470
AUC Score	0.740515

	precision	recall	f1-score	support
0	0.80	0.55	0.65	93
1	0.81	0.93	0.87	193
accuracy			0.81	286
macro avg	0.80	0.74	0.76	286
weighted avg	0.81	0.81	0.80	286







3.4 Neural Network Model Fitting & Tuning

Neural Networks (NN) are inspired by biological neural networks of the brain. These computational models are designed to recognize patterns and solve complex problems by mimicking the way neurons signal each other. Neural networks are especially powerful in handling data with high dimensionality and complexity, such as images, sound, and text, making them integral to advancing fields like computer vision, natural language processing, and audio recognition (Sewak Mohit, Karim Rezaul & Pujari Pradeep 2018).

The basic structure of a neural network includes an input layer that receives data, one or more hidden layers that compute the operations, and an output layer that delivers the final decision or prediction. Each neuron in these layers applies a weighted sum of its inputs and passes the result through a non-linear function, or activation function, which determines whether and to what extent the signal should propagate forward. Learning occurs in neural networks through the adjustment of these weights, typically using a method called backpropagation combined with an optimization algorithm like stochastic gradient descent. This process minimizes the error between the network's prediction and the actual data outcomes during training.

The neural network (NN) approach can be adopted due to its capability to learn complex patterns from data, offering potentially superior predictive accuracy over simpler models.

Neural Network Topology

The architecture for the neural network includes an input layer, 3 hidden layers, and an output layer. The input layer's size is the number of features in our dataset, ensuring that each feature can be independently fed into the model (Uther et al. 2011).

Input Layer: The number of neurons equals the number of preprocessed features (11), which aligns with the dimensionality of our data.

Hidden Layers: Three hidden layers are used to provide sufficient complexity and depth for feature interactions:

- First Hidden Layer: 128 neurons, using ReLU (Rectified Linear Unit) activation function to introduce non-linearity, enhancing the model's ability to learn complex patterns.
- Second Hidden Layer: 64 neurons, also with ReLU activation.
- Third Hidden Layer: 32 neurons, with ReLU activation to maintain non-linear learning.
- Output Layer: A single neuron with a sigmoid activation function, suitable for binary classification task of loan approval in yes or no.

Model Parameter Values

The model can be compiled with the following configurations:

- Optimizer: Adam optimizer is chosen for its efficiency in handling sparse gradients and adaptive learning rate capabilities.
- Loss Function: Binary cross-entropy, which is appropriate for binary classification problems.

- Metrics: Accuracy, the primary metric for assessing the model's performance.

Hyperparameter Tuning

Hyperparameter tuning is critical in neural networks to optimize performance. The following five hyperparameters are selected for fine-tuning. Their impact can be analyzed through various configurations:

- Learning Rate of the Optimizer: A key parameter in the Adam optimizer that controls the size of the steps taken to reach the minimum loss function
- Number of Neurons in Hidden Layers: Adjusting the number of neurons can affect the model's capacity to learn detailed data representations but might lead to overfitting if excessively high
- Batch Size: Determines the number of samples that will be propagated through the network before the optimizer updates model weights.
- Activation Functions: Besides ReLU, sigmoid and tanh in hidden layers can be used to determine their effects on model performance.
- Number of Training Epochs: This is the number of times the learning algorithm will work through the entire training dataset. This can be used to check for overfitting and underfitting.

3.5 Model Comparison

To compare the performances of the KNN, Decision Tree, Naïve Bayes and Random Forest models, we'll look at the summary of all metrics for the 4 models:

K-Nearest Neighbors (KNN)

Accuracy: 77.27% Precision: 80.47% Recall: 87.56% F1 Score: 83.87% AUC Score: 71.74%

Decision Tree

Accuracy: 68.53% Precision: 81.21% Recall: 69.43% F1 Score: 74.86% AUC Score: 68.05%

Naive Bayes

Accuracy: 77.97% Precision: 76.42% Recall: 97.41% F1 Score: 85.65% AUC Score: 67.52%

Random Forest

Accuracy: 80.77% Precision: 81.08% Recall: 93.26% F1 Score: 86.75% AUC Score: 74.05%

Comparison Summary

Accuracy

Random Forest > Naive Bayes > KNN > Decision Tree

Precision

Decision Tree > Random Forest > KNN> Naive Bayes

Recall

Naive Bayes > Random Forest > KNN > Decision Tree

F1 Score

Random Forest > Naive Bayes > KNN > Decision Tree

AUC Score

Random Forest > KNN > Decision Tree > Naive Bayes

Random Forest emerges as the best model when considering all metrics. It offers the highest accuracy and an excellent balance between precision and recall (F1 score), making it particularly effective in this scenario where both false positives and false negatives carry significant cost.

Naive Bayes, while it has a slightly lower precision and the lowest AUC, stands out with the highest recall among all models. This suggests that it is extremely effective at identifying positive cases and classifying an application as eligible. It also has a good F1 score.

Model Cross-Validation

Now, cross validation is performed for each model, and then paired t-tests will be performed to compare the performance of each model against the Random Forest model.

Looking at the cross-validation result:

KNN: The mean ROC-AUC score from cross-validation was 0.631.

Decision Tree: This model achieved a slightly higher mean ROC-AUC score of 0.646.

Naive Bayes: Demonstrated a mean ROC-AUC of 0.673, performing better than KNN and Decision Tree.

Random Forest: Showed the highest mean ROC-AUC score at 0.769, confirming its superiority over the other models in terms of handling the dataset.

Statistical Significance Testing

Random Forest vs. KNN: The t-test between Random Forest and KNN model results yielded a statistic of 11.079 with a p-value of approximately 0.00038. This very low p-value indicates a statistically significant difference in performance, with Random Forest being significantly better than KNN.

Random Forest vs. Decision Tree: Similarly, the t-test comparing Random Forest to the Decision Tree model showed a statistic of 9.006 and p-value of 0.00084. Again, this result confirms a statistically significant difference in favor of the Random Forest model.

Random Forest vs. Naive Bayes: The t-test result is 2.830 with a p-value of 0.047, which is close to the typical alpha level of 0.05. This suggests that while the difference is statistically significant, it is not as significant as with the KNN or Decision Tree models.

In [532…

```
from scipy import stats
from sklearn.model_selection import cross_val_score


cv_results_KNN = cross_val_score(estimator=gs_knn.best_estimator_,
                                 X=Data_test,
                                 y=target_test,
                                 cv=cv_method,
                                 n_jobs=-2,
                                 scoring='roc_auc')
print("KNN cross validation result mean: ", cv_results_KNN.mean())

cv_results_DT = cross_val_score(estimator=gs_DT.best_estimator_,
                                 X=Data_test,
                                 y=target_test,
                                 cv=cv_method,
                                 n_jobs=-2,
                                 scoring='roc_auc')
print("Decision Tree cross validation result mean: ", cv_results_DT.mean())

cv_results_NB = cross_val_score(estimator=gs_NB.best_estimator_,
                                 X=Data_test_transformed,
                                 y=target_test,
                                 cv=cv_method,
                                 n_jobs=-2,
                                 scoring='roc_auc')
print("Naive Bayes cross validation result mean: ", cv_results_NB.mean())

cv_results_RF = cross_val_score(estimator=gs_RF.best_estimator_,
                                 X=Data_test,
                                 y=target_test,
                                 cv=cv_method,
                                 n_jobs=-2,
                                 scoring='roc_auc')
print("Random Forest cross validation result mean: ", cv_results_RF.mean())

print("\nRandom Forest-KNN cross-validation t-test:\n")
print(stats.ttest_rel(cv_results_RF, cv_results_KNN))
print("Random Forest-Decision Tree cross-validation t-test:\n")
print(stats.ttest_rel(cv_results_RF, cv_results_DT))
print("\nRandom Forest-Naive Bayes cross-validation t-test:\n")
print(stats.ttest_rel(cv_results_RF, cv_results_NB))
```

```
KNN cross validation result mean:  0.6310616284300494
Decision Tree cross validation result mean:  0.6460563802669066
Naive Bayes cross validation result mean:  0.6725730204954582
Random Forest cross validation result mean:  0.7694351713742296

Random Forest-KNN cross-validation t-test:

TtestResult(statistic=11.079035119143889, pvalue=0.0003774993944693136, df=4)
Random Forest-Decision Tree cross-validation t-test:

TtestResult(statistic=9.005604779564871, pvalue=0.000841816109576454, df=4)

Random Forest-Naive Bayes cross-validation t-test:

TtestResult(statistic=2.829811821052687, pvalue=0.04735408867863665, df=4)
```

4 Critique & Limitations

4.1 Strengths

Feature importance aligned with real world practices when it comes to loan approvals, such as credit history being a significant predictor as compared to gender.

All models showed good metrics in terms of prediction metrics. None of the models performed extremely poorly in predicting the loan approval.

The application of k-fold cross-validation across all models ensured that the performance metrics were reliable and not overly fitted to a particular subset of the data, thereby enhancing the credibility of the model evaluations.

4.2 Limitations

While all models showed good performance metrics, none of the models performed outstandingly with accuracy above 90%.

While Random Forest provided the best performance, such ensemble models can be complex and less interpretable compared to simpler models like Decision Trees.

New feature engineering was not performed to create potential features that might have improved predicted power for the target.

Although the models were evaluated thoroughly, the treatment of class imbalances in the target was not explicitly addressed. Imbalanced data can bias the models towards the majority class, affecting the reliability of the predictive performance, especially for minority classes.

5 Summary & Conclusions

5.1 Project Summary

This project consisting of Phase 1 and 2 was aimed at developing a robust predictive model for loan approval decisions using a structured approach across two distinct phases: data preparation and predictive modeling. The project used various techniques and methodologies to ensure the data was well-prepared and the models developed were highly effective in predicting outcomes.

Phase 1: Data Preparation

Objectives

- To prepare and cleanse the dataset for predictive modeling.
- To perform exploratory data analysis (EDA) to understand the data's characteristics and relationships.

Methodology

Data Cleaning and Preprocessing step included the following:

- Outlier Detection and Removal: Identified and handled outliers to prevent them from skewing the results.
- Missing Value Imputation: Addressed missing data through median imputation strategies to ensure data integrity.

Exploratory Data Analysis step consisted of:

- Utilized various visualization tools to explore distributions, correlations, and potential relationships within the data.
- Generated insights on important features that could influence the loan approval decision, setting the stage for more focused feature selection in Phase 2.

Phase 2: Predictive Modeling

Objectives

- To develop and evaluate multiple machine learning models to predict loan approvals accurately.
- To optimize model parameters and select the best model based on performance metrics.

Methodology

Feature Selection and Engineering:

- Performed one-hot encoding on Property_Area
- Features were transformed using MinMax scaler
- Applied advanced techniques like Random Forest classifier and ANOVA F-tests to determine and prioritize the most impactful features for the predictive models. F-regression test was used for continuos variables.

Model Development and Optimization:

- Tested 4 models including K-Nearest Neighbors (KNN), Decision Trees, Gaussian Naive Bayes, and Random Forest.
- Employed GridSearchCV to fine-tune the models, ensuring optimal performance.

Model Evaluation:

- Evaluated models using performance metrics such as accuracy, precision, recall, F1 score, and Area Under the ROC Curve (AUC).
- Conducted k-fold cross-validation to validate the model's performance consistently across different subsets of data.

Model Comparison and Selection:

- Compared all models based on their performance metrics. Random Forest emerged as the superior model due to its high ROC-AUC score and significant statistical difference in performance compared to other models, as evidenced by paired t-tests.

Statistical Testing:

- Performed paired t-tests to compare the Random Forest model against other models, confirming its statistical superiority in performance.

5.2 Summary of Findings

The initial phase of the project highlighted several data quality issues, including outliers and missing values, which were effectively addressed through cleaning and imputation methods. This step was crucial as it directly influenced the performance and reliability of the subsequent predictive models.

Credit History was the most significant predictor, suggesting that the historical reliability of a borrower in managing credit is highly indicative of their likelihood of having a loan approved.

Applicant Income and Loan Amount also showed importance, which means that higher incomes and reasonable loan requests correlate positively with loan approval. Applicant income was the most significant predictor among continuous variables. Co-applicant Income was the least important.

Other features like Dependents, Loan Amount term, Gender, Employment and Education showed low importance.

Among the models tested, the Random Forest model exhibited superior performance across several metrics, including accuracy, precision, recall, F1 score, and particularly the Area Under the ROC Curve (AUC).

Paired t-tests between the Random Forest model and other models (KNN, Decision Trees, and Gaussian Naive Bayes) confirmed the statistical superiority of the Random Forest model. This was evident from very low p-values indicating that the performance improvements were statistically significant and not due to random variations in the data.

Naive Bayes performed second best despite the slightly lower precision and the lowest AUC, it had the highest recall.

5.3 Conclusions

The report successfully developed several predictive models, with the Random Forest model achieving the highest performance. Through rigorous cross-validation, the selected Random Forest model proved to be reliable and consistent across different subsets of the data.

The report found that credit history, applicant income, and loan amount were the most crucial predictors

The findings from this project provide a deep understanding of the factors influencing loan approval decisions and demonstrate the efficacy of machine learning in enhancing predictive accuracy. The Random Forest model, with its robust performance and statistical superiority, offers a promising tool for financial institutions to optimize their loan approval processes.

6 References

1. Brownlee, J 2019, How to Choose a Feature Selection Method For Machine Learning, Machine Learning Mastery.
2. — 2020, How to Perform Feature Selection With Numerical Input Data, Machine Learning Mastery.
3. Cunningham, P & Delany, SJ 2021, 'k-Nearest Neighbour Classifiers - A Tutorial', ACM Computing Surveys, vol. 54, no. 6, pp. 1–25.
4. Donges, N 2021, Random Forest: a Complete Guide for Machine Learning, Built in.
5. Feature Selection Techniques in Machine Learning n.d., www.stratascratch.com.
6. Gaussian Naive Bayes Explained With Scikit-Learn | Built In n.d., builtin.com, viewed 2 June 2024, https://builtin.com/artificial-intelligence/gaussian-naive-bayes#.
7. Khalid, S, Khalil, T & Nasreen, S 2014, 'A survey of feature selection and feature extraction techniques in machine learning', 2014 Science and Information Conference.
8. Loan Approval Prediction Dataset n.d., www.kaggle.com.
9. ml_tutorials/case_study_predicting_income_status.ipynb at master · akmand/ml_tutorials n.d., GitHub.
10. practice_exercises/Prac_SK2_Solutions.ipynb at main · akmand/practice_exercises n.d., GitHub.

11. Prusty, S, Patnaik, S & Dash, SK 2022, 'SKCV: Stratified K-fold cross-validation on ML classifiers for predicting cervical cancer', Frontiers in Nanotechnology, vol. 4.

12. Raschka, S 2018, 'Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning', arXiv (Cornell University), Cornell University.

13. SciKit Learn 2019, sklearn.model_selection.GridSearchCV — scikit-learn 0.22 Documentation, Scikit-learn.org.

14. scikit-learn 2009, 1.10. Decision Trees — scikit-learn 0.22 documentation, Scikit-learn.org.

15. Sewak Mohit, Karim Rezaul & Pujari Pradeep 2018, Practical convulational neural networks : Implement advanced deep learning models using Python, Packt Publishing, Birmingham, Uk.

16. sklearn.model_selection.StratifiedKFold — scikit-learn 0.21.3 documentation 2019, Scikit-learn.org.