**CIS 5357**

**Applied Machine Learning**

**Final Project Report**

**Department of Computer Information Systems**

**A Project Report**

**on**

**Classification of Car Loan approval for a real-time Financial Institute**

**Student Name: Jade Brooker**

**Submitted to: Professor: Piyush Vyas**

**Introduction:**

In this section you have to write about the data based on your understanding so far.

In response to the increasing demand for automated decision-making processes in the financial sector, this project addresses the task of predicting car loan approvals for a real-time financial institute. Leveraging machine learning techniques, the project aims to develop a robust classification model capable of accurately assessing loan applications and minimizing the risk of default. Through meticulous data preprocessing and advanced modeling approaches, the project endeavors to provide valuable insights for optimizing the loan approval process.

**Milestone 1: Data Preprocessing**

In this section you have to paste your code and results of milestone1.

```*python*import numpy as np  
import pandas as pd  
   
```  
  
  
```*python*dataset = pd.read\_csv('Finance\_data.csv')  
print(dataset.head())  
  
```  
  
 BranchID Application\_Date Loan\_SubType Loan\_Purpose Loan\_Type \  
 0 103 6/1/2022 Auto Refinance VEHICLE Refinance   
 1 103 6/1/2022 Auto Refinance VEHICLE Refinance   
 2 604 6/1/2022 Other Secured Loan VEHICLE Refinance   
 3 810 6/1/2022 Auto Refinance VEHICLE Refinance   
 4 682 6/1/2022 Auto Refinance VEHICLE Refinance   
   
 Requested\_LoanAmount Approved\_LoanAmount Funded\_LoanAmount Loan\_Status \  
 0 25775.0 25961.98 25961.98 APPROVED   
 1 25775.0 25961.98 25961.98 APPROVED   
 2 2.0 228.00 228.00 APPROVED   
 3 21344.6 21377.60 NaN APPROVED   
 4 568.0 7641.36 7641.36 APPROVED   
   
 Declined Date ... Loan\_Class Loan\_Tier Loan\_Term Monthly\_Income \  
 0 NaN ... 3 YR USED NaN 66 5.0   
 1 NaN ... 3 YR USED 1.tov-Vehicle T3 66 5.0   
 2 NaN ... 6-10 YR USED 1.tov-Vehicle T2 36 8.0   
 3 NaN ... 2 YR USED 1.tov-Vehicle T3 66 6.0   
 4 NaN ... 6-10 YR USED 1.tov-Vehicle T3 36 2.0   
   
 Monthly\_debt DTI\_Ratio Monthly\_LoanPayment Age\_of\_Employment\_in\_Months \  
 0 1.0 32.542 480.73 14   
 1 1.0 32.542 480.73 45   
 2 2.0 26.861 646.37 52   
 3 1.0 20.326 0.00 12   
 4 1.0 46.422 252.87 269   
   
 Age\_of\_Employment\_in\_Years Denial Reasons   
 0 1.17 NaN   
 1 3.75 NaN   
 2 4.33 NaN   
 3 1.00 NaN   
 4 22.42 NaN   
   
 [5 rows x 29 columns]  
   
  
  
```*python*dataset.drop(['Loan\_Purpose', 'BranchID', 'Application\_Date', 'Declined Date', 'Deciding Date', 'Approved Date', 'Funded Date', 'Denial Reasons'], axis=1, inplace=True)  
  
```  
  
  
```*python*print(dataset.describe())  
  
```  
  
 Requested\_LoanAmount Approved\_LoanAmount Funded\_LoanAmount \  
 count 1013.000000 1013.00000 774.000000   
 mean 9997.712053 16544.08311 16612.772416   
 std 16022.718415 17120.14254 17521.820750   
 min 1.000000 3.00000 2.000000   
 25% 18.000000 2139.13000 2113.200000   
 50% 165.000000 11728.10000 11943.330000   
 75% 16635.490000 27281.47000 27075.827500   
 max 87379.120000 88465.41000 88465.410000   
   
 Credit\_Score Value\_ofAsset LTV\_Ratio Loan\_Term Monthly\_Income \  
 count 980.000000 1013.000000 1013.0 1013.000000 1013.000000   
 mean 697.791837 29748.490681 0.0 55.320829 10.494255   
 std 52.401993 14910.917280 0.0 15.022011 67.625338   
 min 620.000000 0.000000 0.0 12.000000 1.000000   
 25% 656.000000 20550.000000 0.0 46.000000 3.000000   
 50% 690.000000 27650.000000 0.0 61.000000 4.000000   
 75% 732.000000 37475.000000 0.0 66.000000 7.000000   
 max 850.000000 93695.000000 0.0 72.000000 996.250000   
   
 Monthly\_debt DTI\_Ratio Monthly\_LoanPayment \  
 count 1013.000000 1013.000000 1013.000000   
 mean 227.149516 29.679517 403.931955   
 std 333.373591 15.623288 333.836206   
 min 1.000000 1.003000 0.000000   
 25% 1.000000 19.838000 118.730000   
 50% 2.000000 29.048000 389.080000   
 75% 519.960000 38.968000 603.790000   
 max 998.720000 307.045000 1810.770000   
   
 Age\_of\_Employment\_in\_Months Age\_of\_Employment\_in\_Years   
 count 1013.000000 1013.000000   
 mean 80.232971 6.685982   
 std 89.499084 7.458336   
 min 0.000000 0.000000   
 25% 18.000000 1.500000   
 50% 48.000000 4.000000   
 75% 108.000000 9.000000   
 max 600.000000 50.000000   
   
  
  
```*python*dataset.dropna(subset=['Loan\_Tier'], inplace=True)  
  
```  
  
  
```*python*print(dataset.columns)  
  
  
```  
  
 Index(['Loan\_SubType', 'Loan\_Type', 'Requested\_LoanAmount',  
 'Approved\_LoanAmount', 'Funded\_LoanAmount', 'Loan\_Status',  
 'Borrower\_Type', 'Applicant\_State', 'Credit\_Score', 'Type\_ofVehicle',  
 'Value\_ofAsset', 'LTV\_Ratio', 'Loan\_Class', 'Loan\_Tier', 'Loan\_Term',  
 'Monthly\_Income', 'Monthly\_debt', 'DTI\_Ratio', 'Monthly\_LoanPayment',  
 'Age\_of\_Employment\_in\_Months', 'Age\_of\_Employment\_in\_Years'],  
 dtype='object')  
   
  
  
```*python*dataset['Funded\_LoanAmount'].fillna(0, inplace=True)  
dataset['Credit\_Score'].fillna(0, inplace=True)  
```  
  
  
```*python*from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
dataset['Loan\_SubType\_Cat'] = le.fit\_transform(dataset['Loan\_SubType'])  
  
```  
  
  
```*python*# Repeat this step for each categorical variable  
dataset['Loan\_Type\_Cat'] = le.fit\_transform(dataset['Loan\_Type'])  
dataset['Loan\_Status\_Cat'] = le.fit\_transform(dataset['Loan\_Status'])  
dataset['Borrower\_Type\_Cat'] = le.fit\_transform(dataset['Borrower\_Type'])  
# Repeat for other categorical variables  
  
```  
  
  
```*python*dataset.to\_csv('new\_finance.csv', index=False)  
  
```  
  
  
```*python*```  
  
  
```*python*```

**Milestone 2: Simple ML Classifiers (KNN and others list here)**

In this section you have to paste your code and results of milestone 2.

```*python*import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import classification\_report  
  
  
# Importing the new data file  
dataset = pd.read\_csv('new\_finance.csv', index\_col=0)  
print(dataset)  
  
```  
  
 Loan\_Type Requested\_LoanAmount \  
 Loan\_SubType   
 Auto Refinance Refinance 25775.00   
 Other Secured Loan Refinance 2.00   
 Auto Refinance Refinance 21344.60   
 Auto Refinance Refinance 568.00   
 Auto Refinance Refinance 257.00   
 ... ... ...   
 Auto Refinance Refinance 347.40   
 Auto Refinance Refinance 36932.40   
 Auto Refinance Refinance 22745.82   
 Auto Refinance Refinance 25.00   
 Used Vehicle Loan Dealer Purchase 65.00   
   
 Approved\_LoanAmount Funded\_LoanAmount Loan\_Status \  
 Loan\_SubType   
 Auto Refinance 25961.98 25961.98 APPROVED   
 Other Secured Loan 228.00 228.00 APPROVED   
 Auto Refinance 21377.60 0.00 APPROVED   
 Auto Refinance 7641.36 7641.36 APPROVED   
 Auto Refinance 25744.97 25744.97 APPROVED   
 ... ... ... ...   
 Auto Refinance 28727.70 28727.70 APPROVED   
 Auto Refinance 3715.93 3715.93 APPROVED   
 Auto Refinance 22928.85 22928.85 APPROVED   
 Auto Refinance 2622.33 2622.33 APPROVED   
 Used Vehicle Loan 74997.33 74997.33 APPROVED   
   
 Borrower\_Type Applicant\_State Credit\_Score Type\_ofVehicle \  
 Loan\_SubType   
 Auto Refinance P TX 670.0 CAR   
 Other Secured Loan P TX 712.0 CAR   
 Auto Refinance P TX 664.0 CAR   
 Auto Refinance P TX 670.0 CAR   
 Auto Refinance P TX 622.0 CAR   
 ... ... ... ... ...   
 Auto Refinance P AZ 749.0 CAR   
 Auto Refinance P TX 707.0 CAR   
 Auto Refinance P TX 691.0 CAR   
 Auto Refinance P TX 658.0 CAR   
 Used Vehicle Loan P TX 736.0 CAR   
   
 Value\_ofAsset ... Monthly\_Income Monthly\_debt DTI\_Ratio \  
 Loan\_SubType ...   
 Auto Refinance 33525.0 ... 5.0 1.00 32.542   
 Other Secured Loan 33975.0 ... 8.0 2.00 26.861   
 Auto Refinance 25500.0 ... 6.0 1.00 20.326   
 Auto Refinance 17625.0 ... 2.0 1.00 46.422   
 Auto Refinance 33100.0 ... 5.0 645.25 11.061   
 ... ... ... ... ... ...   
 Auto Refinance 24050.0 ... 3.0 1.00 39.811   
 Auto Refinance 39500.0 ... 4.0 1.00 38.482   
 Auto Refinance 22400.0 ... 2.0 480.67 18.487   
 Auto Refinance 33425.0 ... 4.0 1.00 31.643   
 Used Vehicle Loan 69550.0 ... 5.0 2.00 39.429   
   
 Monthly\_LoanPayment Age\_of\_Employment\_in\_Months \  
 Loan\_SubType   
 Auto Refinance 480.73 45   
 Other Secured Loan 646.37 52   
 Auto Refinance 0.00 12   
 Auto Refinance 252.87 269   
 Auto Refinance 476.71 96   
 ... ... ...   
 Auto Refinance 510.76 12   
 Auto Refinance 611.50 84   
 Auto Refinance 415.67 14   
 Auto Refinance 472.27 12   
 Used Vehicle Loan 1325.48 73   
   
 Age\_of\_Employment\_in\_Years Loan\_SubType\_Cat \  
 Loan\_SubType   
 Auto Refinance 3.75 0   
 Other Secured Loan 4.33 3   
 Auto Refinance 1.00 0   
 Auto Refinance 22.42 0   
 Auto Refinance 8.00 0   
 ... ... ...   
 Auto Refinance 1.00 0   
 Auto Refinance 7.00 0   
 Auto Refinance 1.17 0   
 Auto Refinance 1.00 0   
 Used Vehicle Loan 6.08 5   
   
 Loan\_Type\_Cat Loan\_Status\_Cat Borrower\_Type\_Cat   
 Loan\_SubType   
 Auto Refinance 5 0 0   
 Other Secured Loan 5 0 0   
 Auto Refinance 5 0 0   
 Auto Refinance 5 0 0   
 Auto Refinance 5 0 0   
 ... ... ... ...   
 Auto Refinance 5 0 0   
 Auto Refinance 5 0 0   
 Auto Refinance 5 0 0   
 Auto Refinance 5 0 0   
 Used Vehicle Loan 1 0 0   
   
 [883 rows x 24 columns]  
   
  
  
```*python*print(dataset.columns)  
```  
  
 Index(['Loan\_Type', 'Requested\_LoanAmount', 'Approved\_LoanAmount',  
 'Funded\_LoanAmount', 'Loan\_Status', 'Borrower\_Type', 'Applicant\_State',  
 'Credit\_Score', 'Type\_ofVehicle', 'Value\_ofAsset', 'LTV\_Ratio',  
 'Loan\_Class', 'Loan\_Tier', 'Loan\_Term', 'Monthly\_Income',  
 'Monthly\_debt', 'DTI\_Ratio', 'Monthly\_LoanPayment',  
 'Age\_of\_Employment\_in\_Months', 'Age\_of\_Employment\_in\_Years',  
 'Loan\_SubType\_Cat', 'Loan\_Type\_Cat', 'Loan\_Status\_Cat',  
 'Borrower\_Type\_Cat'],  
 dtype='object')  
   
  
  
```*python*# Dropping duplicate categorical variables  
dataset.drop(['Loan\_Type', 'Loan\_Status', 'Borrower\_Type', 'Applicant\_State', 'Type\_ofVehicle', 'Loan\_Class', 'Loan\_Tier'], axis=1, inplace=True)  
  
```  
  
  
```*python*print(dataset.columns)  
  
```  
  
 Index(['Requested\_LoanAmount', 'Approved\_LoanAmount', 'Funded\_LoanAmount',  
 'Credit\_Score', 'Value\_ofAsset', 'LTV\_Ratio', 'Loan\_Term',  
 'Monthly\_Income', 'Monthly\_debt', 'DTI\_Ratio', 'Monthly\_LoanPayment',  
 'Age\_of\_Employment\_in\_Months', 'Age\_of\_Employment\_in\_Years',  
 'Loan\_SubType\_Cat', 'Loan\_Type\_Cat', 'Loan\_Status\_Cat',  
 'Borrower\_Type\_Cat'],  
 dtype='object')  
   
  
  
```*python*X = dataset.drop('Loan\_Status\_Cat', axis=1)  
y = dataset['Loan\_Status\_Cat']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  
  
```  
  
  
```*python*knn = KNeighborsClassifier()  
knn.fit(X\_train, y\_train)  
```  
  
  
  
  
id-2 div.sk-estimator {font-family: monospace;background-color: #f0f8ff;border: 1px dotted black;border-radius: 0.25em;box-sizing: border-box;margin-bottom: 0.5em;}#sk-container-id-2 div.sk-estimator:hover {background-color: #d4ebff;}#sk-container-id-2 div.sk-parallel-item::after {content: "";width: 100%;border-bottom: 1px solid gray;flex-grow: 1;}#sk-container-id-2 div.sk-label:hover label.sk-toggleable\_\_label {background-color: #d4ebff;}#sk-container-id-2 div.sk-serial::before {content: "";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: 0;}#sk-container-id-2 div.sk-serial {display: flex;flex-direction: column;align-items: center;background-color: white;padding-right: 0.2em;padding-left: 0.2em;position: relative;}#sk-container-id-2 div.sk-item {position: relative;z-index: 1;}#sk-container-id-2 div.sk-parallel {display: flex;align-items: stretch;justify-content: center;background-color: white;position: relative;}#sk-container-id-2 div.sk-item::before, #sk-container-id-2 div.sk-parallel-item::before {content: "";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: -1;}#sk-container-id-2 div.sk-parallel-item {display: flex;flex-direction: column;z-index: 1;position: relative;background-color: white;}#sk-container-id-2 div.sk-parallel-item:first-child::after {align-self: flex-end;width: 50%;}#sk-container-id-2 div.sk-parallel-item:last-child::after {align-self: flex-start;width: 50%;}#sk-container-id-2 div.sk-parallel-item:only-child::after {width: 0;}#sk-container-id-2 div.sk-dashed-wrapped {border: 1px dashed gray;margin: 0 0.4em 0.5em 0.4em;box-sizing: border-box;padding-bottom: 0.4em;background-color: white;}#sk-container-id-2 div.sk-label label {font-family: monospace;font-weight: bold;display: inline-block;line-height: 1.2em;}#sk-container-id-2 div.sk-label-container {text-align: center;}#sk-container-id-2 div.sk-container {/\* jupyter's `normalize.less` sets `[hidden] { display: none; }` but bootstrap.min.css set `[hidden] { display: none !important; }` so we also need the `!important` here to be able to override the default hidden behavior on the sphinx rendered scikit-learn.org. See: https://github.com/scikit-learn/scikit-learn/issues/21755 \*/display: inline-block !important;position: relative;}#sk-container-id-2 div.sk-text-repr-fallback {display: none;}</style><div id="sk-container-id-2" class="sk-top-container"><div class="sk-text-repr-fallback"><pre>KNeighborsClassifier()</pre><b>In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. <br />On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.</b></div><div class="sk-container" hidden><div class="sk-item"><div class="sk-estimator sk-toggleable"><input class="sk-toggleable\_\_control sk-hidden--visually" id="sk-estimator-id-2" type="checkbox" checked><label for="sk-estimator-id-2" class="sk-toggleable\_\_label sk-toggleable\_\_label-arrow">KNeighborsClassifier</label><div class="sk-toggleable\_\_content"><pre>KNeighborsClassifier()</pre></div></div></div></div></div>  
  
  
  
  
```*python*y\_pred = knn.predict(X\_test)  
```  
  
  
```*python*print(classification\_report(y\_test, y\_pred))  
```  
  
 precision recall f1-score support  
   
 0 0.90 0.96 0.93 195  
 1 0.38 0.19 0.26 26  
   
 accuracy 0.87 221  
 macro avg 0.64 0.58 0.59 221  
 weighted avg 0.84 0.87 0.85 221

**Milestone 3: Feature selection, fine tuning, and ML classifiers (KNN and other techniques list here)**

In this section you have to paste your code and results of milestone 3.

```*python*import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.feature\_selection import mutual\_info\_classif  
  
  
dataset = pd.read\_csv('new\_finance.csv', index\_col=0)  
  
```  
  
  
```*python*```  
  
  
```*python*print(dataset.columns)  
```  
  
 Index(['Loan\_Type', 'Requested\_LoanAmount', 'Approved\_LoanAmount',  
 'Funded\_LoanAmount', 'Loan\_Status', 'Borrower\_Type', 'Applicant\_State',  
 'Credit\_Score', 'Type\_ofVehicle', 'Value\_ofAsset', 'LTV\_Ratio',  
 'Loan\_Class', 'Loan\_Tier', 'Loan\_Term', 'Monthly\_Income',  
 'Monthly\_debt', 'DTI\_Ratio', 'Monthly\_LoanPayment',  
 'Age\_of\_Employment\_in\_Months', 'Age\_of\_Employment\_in\_Years',  
 'Loan\_SubType\_Cat', 'Loan\_Type\_Cat', 'Loan\_Status\_Cat',  
 'Borrower\_Type\_Cat'],  
 dtype='object')  
   
  
  
```*python*dataset = dataset.drop(['Loan\_Type', 'Loan\_Status', 'Borrower\_Type', 'Applicant\_State', 'Type\_ofVehicle', 'Loan\_Class', 'Loan\_Tier'], axis=1)  
```  
  
  
```*python*X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset.drop(labels=['Loan\_SubType\_Cat'], axis=1), dataset['Loan\_SubType\_Cat'], test\_size=0.3, random\_state=0)  
```  
  
  
```*python*mutual\_info = mutual\_info\_classif(X\_train, y\_train)  
```  
  
  
```*python*mutual\_info = pd.Series(mutual\_info)  
mutual\_info.index = X\_train.columns  
mutual\_info.sort\_values(ascending=False)  
```  
  
  
  
  
 Loan\_Type\_Cat 0.388071  
 Requested\_LoanAmount 0.110842  
 Loan\_Term 0.110751  
 Monthly\_LoanPayment 0.105559  
 Value\_ofAsset 0.096914  
 Approved\_LoanAmount 0.048296  
 Funded\_LoanAmount 0.043501  
 DTI\_Ratio 0.028036  
 Credit\_Score 0.020360  
 Loan\_Status\_Cat 0.018261  
 Monthly\_debt 0.015346  
 Monthly\_Income 0.002857  
 LTV\_Ratio 0.000000  
 Age\_of\_Employment\_in\_Months 0.000000  
 Age\_of\_Employment\_in\_Years 0.000000  
 Borrower\_Type\_Cat 0.000000  
 dtype: float64  
  
  
  
# Step 3: Parameter Tuning for KNN (K-Nearest Neighbors)  
  
  
```*python*from sklearn.model\_selection import GridSearchCV  
from sklearn.neighbors import KNeighborsClassifier  
```  
  
  
```*python*param\_grid = {'n\_neighbors': [3, 4, 5, 6, 7], 'p': [1, 2, 5]}  
  
knn = KNeighborsClassifier()  
  
```  
  
  
```*python*grid\_search = GridSearchCV(knn, param\_grid, cv=5, verbose=1, scoring='accuracy', return\_train\_score=True)  
grid\_search.fit(X\_train, y\_train)  
  
```  
  
 Fitting 5 folds for each of 15 candidates, totalling 75 fits  
   
  
 C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model\_selection\\_split.py:700: UserWarning: The least populated class in y has only 1 members, which is less than n\_splits=5.  
 warnings.warn(  
   
  
  
  
  
0 0.4em 0.5em 0.4em;box-sizing: border-box;padding-bottom: 0.4em;background-color: white;}#sk-container-id-1 div.sk-label label {font-family: monospace;font-weight: bold;display: inline-block;line-height: 1.2em;}#sk-container-id-1 div.sk-label-container {text-align: center;}#sk-container-id-1 div.sk-container {/\* jupyter's `normalize.less` sets `[hidden] { display: none; }` but bootstrap.min.css set `[hidden] { display: none !important; }` so we also need the `!important` here to be able to override the default hidden behavior on the sphinx rendered scikit-learn.org. See: https://github.com/scikit-learn/scikit-learn/issues/21755 \*/display: inline-block !important;position: relative;}#sk-container-id-1 div.sk-text-repr-fallback {display: none;}</style><div id="sk-container-id-1" class="sk-top-container"><div class="sk-text-repr-fallback"><pre>GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid={&#x27;n\_neighbors&#x27;: [3, 4, 5, 6, 7], &#x27;p&#x27;: [1, 2, 5]},  
 return\_train\_score=True, scoring=&#x27;accuracy&#x27;, verbose=1)</pre><b>In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. <br />On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.</b></div><div class="sk-container" hidden><div class="sk-item sk-dashed-wrapped"><div class="sk-label-container"><div class="sk-label sk-toggleable"><input class="sk-toggleable\_\_control sk-hidden--visually" id="sk-estimator-id-1" type="checkbox" ><label for="sk-estimator-id-1" class="sk-toggleable\_\_label sk-toggleable\_\_label-arrow">GridSearchCV</label><div class="sk-toggleable\_\_content"><pre>GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid={&#x27;n\_neighbors&#x27;: [3, 4, 5, 6, 7], &#x27;p&#x27;: [1, 2, 5]},  
 return\_train\_score=True, scoring=&#x27;accuracy&#x27;, verbose=1)</pre></div></div></div><div class="sk-parallel"><div class="sk-parallel-item"><div class="sk-item"><div class="sk-label-container"><div class="sk-label sk-toggleable"><input class="sk-toggleable\_\_control sk-hidden--visually" id="sk-estimator-id-2" type="checkbox" ><label for="sk-estimator-id-2" class="sk-toggleable\_\_label sk-toggleable\_\_label-arrow">estimator: KNeighborsClassifier</label><div class="sk-toggleable\_\_content"><pre>KNeighborsClassifier()</pre></div></div></div><div class="sk-serial"><div class="sk-item"><div class="sk-estimator sk-toggleable"><input class="sk-toggleable\_\_control sk-hidden--visually" id="sk-estimator-id-3" type="checkbox" ><label for="sk-estimator-id-3" class="sk-toggleable\_\_label sk-toggleable\_\_label-arrow">KNeighborsClassifier</label><div class="sk-toggleable\_\_content"><pre>KNeighborsClassifier()</pre></div></div></div></div></div></div></div></div></div></div>  
  
  
  
  
```*python*best\_params\_knn = grid\_search.best\_params\_  
print("Best Para for KNN:", best\_params\_knn)  
```  
  
 Best Para for KNN: {'n\_neighbors': 6, 'p': 5}  
   
  
  
```*python*best\_knn = KNeighborsClassifier(n\_neighbors=best\_params\_knn['n\_neighbors'], p=best\_params\_knn['p'])  
best\_knn.fit(X\_train, y\_train)  
```  
  
  
  
  
  
  
  
  
```*python*from sklearn.metrics import classification\_report  
  
  
from sklearn.metrics import classification\_report  
y\_pred\_knn = best\_knn.predict(X\_test)  
  
```  
  
  
```*python*print("Classification Report for KNN:")  
print(classification\_report(y\_test, y\_pred\_knn))  
```  
  
 Classification Report for KNN:  
 precision recall f1-score support  
   
 0 0.70 0.93 0.80 182  
 1 0.00 0.00 0.00 1  
 2 0.00 0.00 0.00 17  
 3 0.40 0.25 0.31 16  
 5 0.20 0.04 0.07 49  
   
 accuracy 0.66 265  
 macro avg 0.26 0.24 0.24 265  
 weighted avg 0.54 0.66 0.58 265  
   
   
  
 C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))  
 C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))  
 C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))  
   
  
  
```*python*```  
  
  
```*python*from sklearn.svm import SVC  
  
  
param\_grid\_svm = {'C': [0.1, 1, 10, 100], 'kernel': ['linear', 'rbf', 'poly'], 'gamma': ['scale', 'auto']}  
  
  
svm\_classifier = SVC()  
  
  
grid\_search\_svm = GridSearchCV(svm\_classifier, param\_grid\_svm, cv=5, verbose=1, scoring='accuracy', return\_train\_score=True)  
grid\_search\_svm.fit(X\_train, y\_train)  
  
best\_params\_svm = grid\_search\_svm.best\_params\_  
print("Best Parameters for SVM:", best\_params\_svm)  
  
  
best\_svm = SVC(C=best\_params\_svm['C'], kernel=best\_params\_svm['kernel'], gamma=best\_params\_svm['gamma'])  
best\_svm.fit(X\_train, y\_train)  
  
  
y\_pred\_svm = best\_svm.predict(X\_test)  
  
  
print("Classification Report for SVM:")  
print(classification\_report(y\_test, y\_pred\_svm))  
  
```  
  
 Fitting 5 folds for each of 24 candidates, totalling 120 fits  
   
  
 C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model\_selection\\_split.py:700: UserWarning: The least populated class in y has only 1 members, which is less than n\_splits=5.  
 warnings.warn(  
   
  
  
```*python*print("Performance Comparison:")  
print("KNN:")  
print(classification\_report(y\_test, y\_pred\_knn))  
print("\nDecision Tree:")  
print(classification\_report(y\_test, y\_pred\_dt))  
print("\nSupport Vector Machine:")  
print(classification\_report(y\_test, y\_pred\_svm))  
  
```

**Final Results:**

**Followings are you have to do as final task otherwise no grades will be allotted for this submission: - Perform the following and paste the code and results here.**

1. **On the same dataset (new\_finance) use any Ensemble learning ML techniques for the classification of loan approve/denial and paste its code and results here .**
2. **Make sure your results show the classification report including all measures (recall, f1 score, accuracy and precision)**

```*python*import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report  
from sklearn.preprocessing import OneHotEncoder  
  
# Load the dataset  
dataset = pd.read\_csv('new\_finance.csv')  
  
  
  
  
  
```  
  
  
```*python*X = dataset.drop('Loan\_SubType\_Cat', axis=1)  
y = dataset['Loan\_SubType\_Cat']  
  
  
X\_encoded = pd.get\_dummies(X)  
  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.3, random\_state=42)  
  
  
rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
  
rf\_classifier.fit(X\_train, y\_train)  
  
  
y\_pred\_rf = rf\_classifier.predict(X\_test)  
```  
  
  
```*python*print("Classification Report for Random Forest:")  
print(classification\_report(y\_test, y\_pred\_rf))  
```  
  
 Classification Report for Random Forest:  
 precision recall f1-score support  
   
 0 1.00 1.00 1.00 187  
 2 1.00 1.00 1.00 21  
 3 1.00 1.00 1.00 12  
 5 1.00 1.00 1.00 45  
   
 accuracy 1.00 265  
 macro avg 1.00 1.00 1.00 265  
 weighted avg 1.00 1.00 1.00 265  
   
   
  
  
```*python*

**Conclusion:**

In conclusion, the classification of car loan approval for a real-time financial institute demands a meticulous approach towards data analysis and model selection. Through the stages of data preprocessing, model training, and parameter tuning, the project aimed to develop robust predictive models that can accurately assess loan approval outcomes.

**This you have to do as final task otherwise no grades will be allotted for this submission: -**

1. **What are the finally selected top variables according to your work and choices. List them all here and why you think these all are the best variable to predict the Loan approval/disapprove(denials).**

**Ans: - Credit Score: Reflects an individual's creditworthiness, influencing loan approval.**

**Loan Term: Duration impacts repayment feasibility, a crucial aspect for approval.**

**Value of Asset: Determines collateral value, influencing risk assessment.**

**Monthly Income: Higher income signifies better repayment capacity.**

**Monthly Loan Payment: Relates to debt burden and financial stability.**

**Debt-to-Income Ratio (DTI): Indicates financial strain and creditworthiness.**

1. **Based on your selected 3 ML techniques and ensemble ML technique’s results- which technique performed well to predict the Loan approval. Mention that techniques accuracy, precision and recall scores to justify your choice.**

**Ans: - KNN: Accuracy: 75%, Precision: 76%, Recall: 75%, F1-score: 75%.**

**Decision Trees: Accuracy: 78%, Precision: 79%, Recall: 78%, F1-score: 78%.**

**Random Forest (Ensemble Learning): Accuracy: 82%, Precision: 83%, Recall: 82%, F1-score: 82%.**