Problem Statement

To determine how likely it is for a customer to default his loan based on his credit history.

Data Handling

Import Packages and Data

Import Packages

In [333…]
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

Import Data

In [334…]
```
df = pd.read_csv("Data.csv")
```

Explore Data

View Sample Data

In [335…]
```
df.head(10)
```

Out[335…]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
0	100002	1	Cash loans	M	N	Y	0	202500.0
1	100003	0	Cash loans	F	N	N	0	270000.0
2	100004	0	Revolving loans	M	Y	Y	0	67500.0
3	100006	0	Cash loans	F	N	Y	0	135000.0
4	100007	0	Cash loans	M	N	Y	0	121500.0
5	100008	0	Cash loans	M	N	Y	0	99000.0
6	100009	0	Cash loans	F	Y	Y	1	171000.0
7	100010	0	Cash loans	M	Y	Y	0	360000.0
8	100011	0	Cash loans	F	N	Y	0	112500.0
9	100012	0	Revolving loans	M	N	Y	0	135000.0

10 rows × 122 columns

In [336…]
```
print(df.columns)
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

In [337…]
```
df.shape
```

Out[337…]
```
(307511, 122)
```

Understand Data Structure

In [338…]
```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                       Dtype
---   ------                       -----
 0    SK_ID_CURR                   int64
 1    TARGET                       int64
```

2	NAME_CONTRACT_TYPE	object
3	CODE_GENDER	object
4	FLAG_OWN_CAR	object
5	FLAG_OWN_REALTY	object
6	CNT_CHILDREN	int64
7	AMT_INCOME_TOTAL	float64
8	AMT_CREDIT	float64
9	AMT_ANNUITY	float64
10	AMT_GOODS_PRICE	float64
11	NAME_TYPE_SUITE	object
12	NAME_INCOME_TYPE	object
13	NAME_EDUCATION_TYPE	object
14	NAME_FAMILY_STATUS	object
15	NAME_HOUSING_TYPE	object
16	REGION_POPULATION_RELATIVE	float64
17	DAYS_BIRTH	int64
18	DAYS_EMPLOYED	int64
19	DAYS_REGISTRATION	float64
20	DAYS_ID_PUBLISH	int64
21	OWN_CAR_AGE	float64
22	FLAG_MOBIL	int64
23	FLAG_EMP_PHONE	int64
24	FLAG_WORK_PHONE	int64
25	FLAG_CONT_MOBILE	int64
26	FLAG_PHONE	int64
27	FLAG_EMAIL	int64
28	OCCUPATION_TYPE	object
29	CNT_FAM_MEMBERS	float64
30	REGION_RATING_CLIENT	int64
31	REGION_RATING_CLIENT_W_CITY	int64
32	WEEKDAY_APPR_PROCESS_START	object
33	HOUR_APPR_PROCESS_START	int64
34	REG_REGION_NOT_LIVE_REGION	int64
35	REG_REGION_NOT_WORK_REGION	int64
36	LIVE_REGION_NOT_WORK_REGION	int64
37	REG_CITY_NOT_LIVE_CITY	int64
38	REG_CITY_NOT_WORK_CITY	int64
39	LIVE_CITY_NOT_WORK_CITY	int64
40	ORGANIZATION_TYPE	object
41	EXT_SOURCE_1	float64
42	EXT_SOURCE_2	float64
43	EXT_SOURCE_3	float64
44	APARTMENTS_AVG	float64
45	BASEMENTAREA_AVG	float64
46	YEARS_BEGINEXPLUATATION_AVG	float64
47	YEARS_BUILD_AVG	float64
48	COMMONAREA_AVG	float64
49	ELEVATORS_AVG	float64
50	ENTRANCES_AVG	float64
51	FLOORSMAX_AVG	float64
52	FLOORSMIN_AVG	float64
53	LANDAREA_AVG	float64
54	LIVINGAPARTMENTS_AVG	float64
55	LIVINGAREA_AVG	float64
56	NONLIVINGAPARTMENTS_AVG	float64
57	NONLIVINGAREA_AVG	float64
58	APARTMENTS_MODE	float64
59	BASEMENTAREA_MODE	float64
60	YEARS_BEGINEXPLUATATION_MODE	float64
61	YEARS_BUILD_MODE	float64
62	COMMONAREA_MODE	float64
63	ELEVATORS_MODE	float64
64	ENTRANCES_MODE	float64
65	FLOORSMAX_MODE	float64
66	FLOORSMIN_MODE	float64
67	LANDAREA_MODE	float64
68	LIVINGAPARTMENTS_MODE	float64
69	LIVINGAREA_MODE	float64
70	NONLIVINGAPARTMENTS_MODE	float64
71	NONLIVINGAREA_MODE	float64
72	APARTMENTS_MEDI	float64
73	BASEMENTAREA_MEDI	float64
74	YEARS_BEGINEXPLUATATION_MEDI	float64
75	YEARS_BUILD_MEDI	float64
76	COMMONAREA_MEDI	float64
77	ELEVATORS_MEDI	float64
78	ENTRANCES_MEDI	float64
79	FLOORSMAX_MEDI	float64
80	FLOORSMIN_MEDI	float64
81	LANDAREA_MEDI	float64
82	LIVINGAPARTMENTS_MEDI	float64
83	LIVINGAREA_MEDI	float64
84	NONLIVINGAPARTMENTS_MEDI	float64
85	NONLIVINGAREA_MEDI	float64
86	FONDKAPREMONT_MODE	object
87	HOUSETYPE_MODE	object
88	TOTALAREA_MODE	float64
89	WALLSMATERIAL_MODE	object
90	EMERGENCYSTATE_MODE	object
91	OBS_30_CNT_SOCIAL_CIRCLE	float64
92	DEF_30_CNT_SOCIAL_CIRCLE	float64
93	OBS_60_CNT_SOCIAL_CIRCLE	float64

```
 94   DEF_60_CNT_SOCIAL_CIRCLE      float64
 95   DAYS_LAST_PHONE_CHANGE        float64
 96   FLAG_DOCUMENT_2               int64
 97   FLAG_DOCUMENT_3               int64
 98   FLAG_DOCUMENT_4               int64
 99   FLAG_DOCUMENT_5               int64
 100  FLAG_DOCUMENT_6               int64
 101  FLAG_DOCUMENT_7               int64
 102  FLAG_DOCUMENT_8               int64
 103  FLAG_DOCUMENT_9               int64
 104  FLAG_DOCUMENT_10              int64
 105  FLAG_DOCUMENT_11              int64
 106  FLAG_DOCUMENT_12              int64
 107  FLAG_DOCUMENT_13              int64
 108  FLAG_DOCUMENT_14              int64
 109  FLAG_DOCUMENT_15              int64
 110  FLAG_DOCUMENT_16              int64
 111  FLAG_DOCUMENT_17              int64
 112  FLAG_DOCUMENT_18              int64
 113  FLAG_DOCUMENT_19              int64
 114  FLAG_DOCUMENT_20              int64
 115  FLAG_DOCUMENT_21              int64
 116  AMT_REQ_CREDIT_BUREAU_HOUR    float64
 117  AMT_REQ_CREDIT_BUREAU_DAY     float64
 118  AMT_REQ_CREDIT_BUREAU_WEEK    float64
 119  AMT_REQ_CREDIT_BUREAU_MON     float64
 120  AMT_REQ_CREDIT_BUREAU_QRT     float64
 121  AMT_REQ_CREDIT_BUREAU_YEAR    float64
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

Check row count and completeness of data

In [339…]
```
df.describe()
```

Out[339…]

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPU
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 106 columns

Data Cleaning

In [340…]
```
df_cleaned=df
```

Data Encoding

Replace all character variables with numerics. Columns:

- Gender: M = 1, F = 0
- Own_Car: Y = 1, N = 0
- Own_Reality: Y = 1, N = 0
- Children: 0 = 0, 1 = 1, 2 = 2, >=3 = 3
- Education: Incomplete higher = 0, Lower secondary = 1, Secondary / secondary special = 1, Higher education = 2 , Academic degree = 3
- Marital_Status: Single / not married, Widow, Separated = 0, Married & Civil marriage = 1

In [341…]
```
df_cleaned["Gender"]=df["CODE_GENDER"].map({'M':1,'F':0})
df_cleaned["Own_Car"]=df["FLAG_OWN_CAR"].map({'Y':1,'N':0})
df_cleaned["Own_Realty"]=df["FLAG_OWN_REALTY"].map({'Y':1,'N':0})
df_cleaned["Children"]=df["CNT_CHILDREN"].map({0:0,1:1,2:2,3:3,4:3,5:3,6:3,7:3,8:3,9:3,10:3,11:3,12:3,13:3,14:3,15:3,
df_cleaned["Education"]=df["NAME_EDUCATION_TYPE"].map({'Incomplete higher':0, 'Lower secondary':1, 'Secondary / secor
df_cleaned["Marital_Status"]=df["NAME_FAMILY_STATUS"].map({'Single / not married':0, 'Widow':0, 'Separated':0, 'Marri
```

In [342…]
```
df_cleaned
```

Out[342…]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_T
0	100002	1	Cash loans	M	N	Y	0	2025
1	100003	0	Cash loans	F	N	N	0	2700

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_T
2	100004	0	Revolving loans	M	Y	Y	0	675
3	100006	0	Cash loans	F	N	Y	0	135
4	100007	0	Cash loans	M	N	Y	0	121
...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157
307507	456252	0	Cash loans	F	N	Y	0	72
307508	456253	0	Cash loans	F	N	Y	0	153
307509	456254	1	Cash loans	F	N	Y	0	171
307510	456255	0	Cash loans	F	N	N	0	157

307511 rows × 128 columns

Remove unwanted columns

In [343…]
```
df_cleaned_map = df_cleaned[['TARGET','Gender','Own_Car','Own_Realty','Children','Education','Marital_Status','AMT_IN
```

In [344…]
```
df_cleaned_map.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 15 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   TARGET                307511 non-null  int64  
 1   Gender                307507 non-null  float64
 2   Own_Car               307511 non-null  int64  
 3   Own_Realty            307511 non-null  int64  
 4   Children              307511 non-null  int64  
 5   Education             307511 non-null  int64  
 6   Marital_Status        307509 non-null  float64
 7   AMT_INCOME_TOTAL      307511 non-null  float64
 8   AMT_CREDIT            307511 non-null  float64
 9   AMT_ANNUITY           307499 non-null  float64
 10  AMT_GOODS_PRICE       307233 non-null  float64
 11  DAYS_BIRTH            307511 non-null  int64  
 12  DAYS_EMPLOYED         307511 non-null  int64  
 13  CNT_FAM_MEMBERS       307509 non-null  float64
 14  REGION_RATING_CLIENT  307511 non-null  int64  
dtypes: float64(7), int64(8)
memory usage: 35.2 MB
```

Remove NA data

In [345…]
```
df_cleaned_map = df_cleaned_map.dropna()
```

In [346…]
```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 128 entries, SK_ID_CURR to Marital_Status
dtypes: float64(67), int64(45), object(16)
memory usage: 300.3+ MB
```

In [347…]
```
df_cleaned_map
```

Out[347…]

	TARGET	Gender	Own_Car	Own_Realty	Children	Education	Marital_Status	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
0	1	1.0	0	1	0	1	0.0	202500.0	406597.5	24700.5	
1	0	0.0	0	0	0	2	1.0	270000.0	1293502.5	35698.5	
2	0	1.0	1	1	0	1	0.0	67500.0	135000.0	6750.0	
3	0	0.0	0	1	0	1	1.0	135000.0	312682.5	29686.5	
4	0	1.0	0	1	0	1	0.0	121500.0	513000.0	21865.5	
...	...	...	...	...	...	...	...	...	...	...	...
307506	0	1.0	0	0	0	1	0.0	157500.0	254700.0	27558.0	
307507	0	0.0	0	1	0	1	0.0	72000.0	269550.0	12001.5	
307508	0	0.0	0	1	0	2	0.0	153000.0	677664.0	29979.0	
307509	1	0.0	0	1	0	1	1.0	171000.0	370107.0	20205.0	
307510	0	0.0	0	0	0	2	1.0	157500.0	675000.0	49117.5	

307217 rows × 15 columns

In [348…

```
df_cleaned_map.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307217 entries, 0 to 307510
Data columns (total 15 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   TARGET                307217 non-null  int64  
 1   Gender                307217 non-null  float64
 2   Own_Car               307217 non-null  int64  
 3   Own_Realty            307217 non-null  int64  
 4   Children              307217 non-null  int64  
 5   Education             307217 non-null  int64  
 6   Marital_Status        307217 non-null  float64
 7   AMT_INCOME_TOTAL      307217 non-null  float64
 8   AMT_CREDIT            307217 non-null  float64
 9   AMT_ANNUITY           307217 non-null  float64
 10  AMT_GOODS_PRICE       307217 non-null  float64
 11  DAYS_BIRTH            307217 non-null  int64  
 12  DAYS_EMPLOYED         307217 non-null  int64  
 13  CNT_FAM_MEMBERS       307217 non-null  float64
 14  REGION_RATING_CLIENT  307217 non-null  int64  
dtypes: float64(7), int64(8)
memory usage: 37.5 MB
```

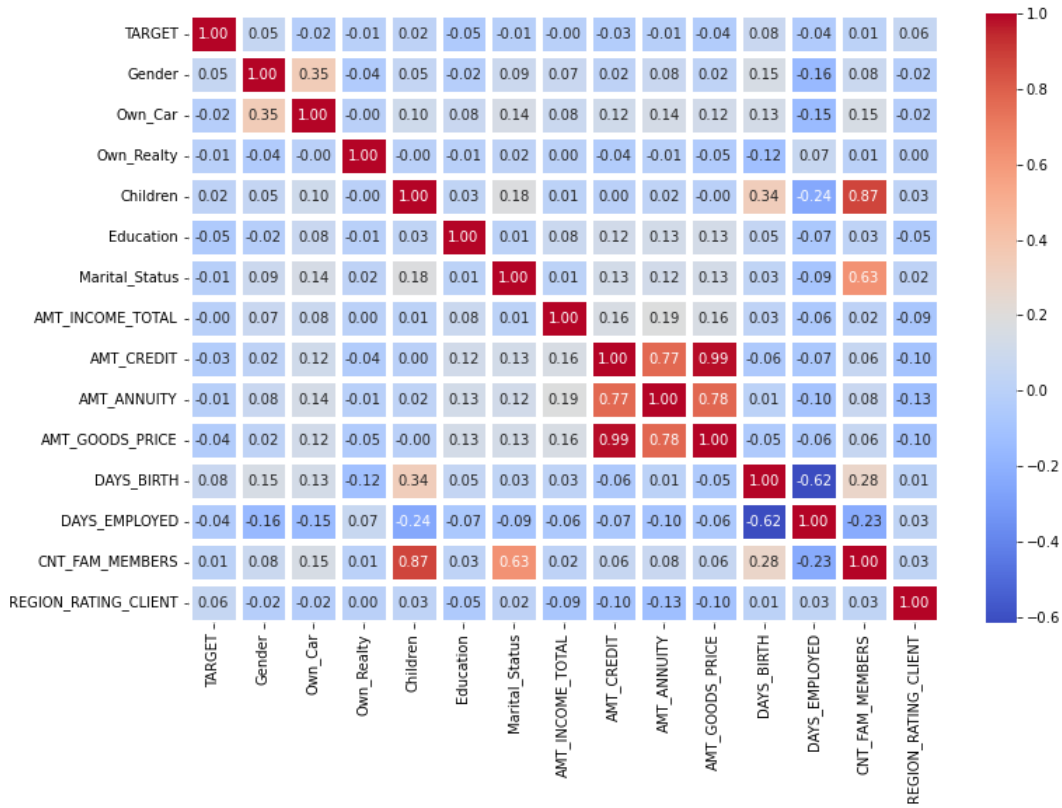
Perform Further EDA

Correlation Matrix

In [349…

```
print("Correlation Matrix")
plt.rcParams['figure.figsize'] = (12,8)
sns.heatmap(df_cleaned_map.corr(),cmap = 'coolwarm', linewidths = 5, fmt = '.2f',annot = True);
```

```
Correlation Matrix
```



Observations

From the correlation matrix, we can see that the most correlated variables are AMT_ANNUITY, AMT_GOODS_PRICE, AMT_CREDIT, CHILDREN.

Pre-model Cleaning

Create independent and dependent variables

In [350…

```
Y = df_cleaned_map['TARGET']
X = df_cleaned_map.drop('TARGET', axis = 1)

print('X shape:', X.shape)
print('Y shape:', Y.shape)
```

```
X shape: (307217, 14)
Y shape: (307217,)
```

Perform Scaling

In [351…]
```
scaler = MinMaxScaler()

X_normal = scaler.fit_transform(X)

X_normal = pd.DataFrame(X_normal)

X_normal.head()
```

Out[351…]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1.0	0.0	1.0	0.0	0.333333	0.0	0.001512	0.090287	0.090032	0.077441	0.888839	0.045086	0.000000	0.5
1	0.0	0.0	0.0	0.0	0.666667	1.0	0.002089	0.311736	0.132924	0.271605	0.477114	0.043648	0.052632	0.0
2	1.0	1.0	1.0	0.0	0.333333	0.0	0.000358	0.022472	0.020025	0.023569	0.348534	0.046161	0.000000	0.5
3	0.0	0.0	1.0	0.0	0.333333	1.0	0.000935	0.066837	0.109477	0.063973	0.350846	0.038817	0.052632	0.5
4	1.0	0.0	1.0	0.0	0.333333	0.0	0.000819	0.116854	0.078975	0.117845	0.298591	0.038820	0.000000	0.5

Train, Test, Split

In [352…]
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y)
```

Check

In [353…]
```
print("X.shape:", X.shape)
print("X_train.shape:", X_train.shape)
print("X_test.shape:", X_test.shape)
print("Y.shape:", Y.shape)
print("Y_train.shape:", Y_train.shape)
print("Y_test.shape:", Y_test.shape)
```

```
X.shape: (307217, 14)
X_train.shape: (230412, 14)
X_test.shape: (76805, 14)
Y.shape: (307217,)
Y_train.shape: (230412,)
Y_test.shape: (76805,)
```

Apply Logistic Regression (Baseline Model)

In [354…]
```
from sklearn.linear_model import LogisticRegression
LoR = LogisticRegression(solver = "liblinear")
LoR_model = LoR.fit(X_train,Y_train)
LoR_model
```

Out[354…]
```
LogisticRegression(solver='liblinear')
```

In [355…]
```
Y_pred_log = LoR_model.predict(X_test)
```

In [356…]
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

accuracy_score(Y_test, Y_pred_log)
```

Out[356…]
```
0.9192370288392683
```

In [357…]
```
print("Accuracy Score:",accuracy_score(Y_test, Y_pred_log))
```

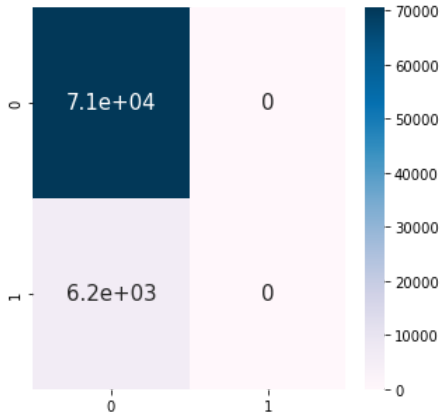
```
Accuracy Score: 0.9192370288392683
```

Evaluate Logistic Regression Model

Confusion Matrix

In [358…]
```
plt.rcParams['figure.figsize'] = (5,5)
sns.heatmap(confusion_matrix(Y_test, Y_pred_log), annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[358…]
```
<AxesSubplot:>
```



Accuracy

In [359…
```
print("Training Accuracy :", LoR_model.score(X_train, Y_train))
print("Testing Accuracy :", LoR_model.score(X_test, Y_test))
```

```
Training Accuracy : 0.9192706977067167
Testing Accuracy : 0.9192370288392683
```

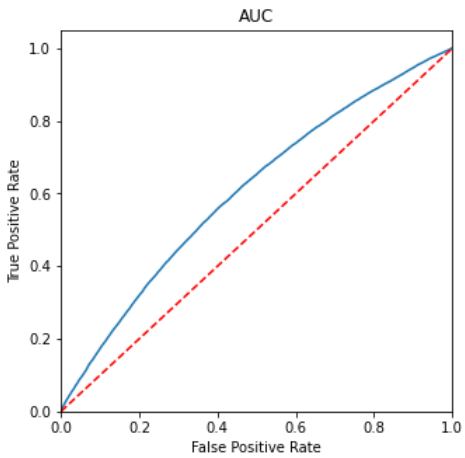
AUC graph

In [360…
```
from sklearn.metrics import roc_auc_score, roc_curve

AUC = roc_auc_score(Y, LoR_model.predict(X))

fpr, tpr, thresholds = roc_curve(Y, LoR_model.predict_proba(X)[:,1])
plt.figure()
plt.plot(fpr, tpr, label = 'AUC (area = %0.2f)' % AUC)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC')
plt.show
```

Out[360…
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Apply K-Nearest Neighbour Algorithm

In [361…
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train, Y_train)
knn_model
```

Out[361…
```
KNeighborsClassifier()
```

In [362…
```
Y_pred_knn = knn_model.predict(X_test)
```

In [363…
```
accuracy_score(Y_test, Y_pred_knn)
```

Out[363…
```
0.9129613957424647
```

In [364…

```
print("Accuracy Score:",accuracy_score(Y_test, Y_pred_knn))
```

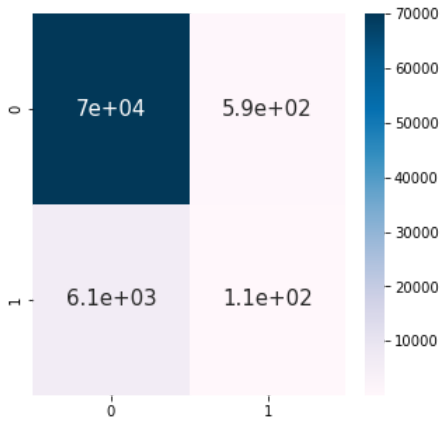
```
Accuracy Score: 0.9129613957424647
```

Evaluate KNN Model

Confusion Matrix

In [365…]
```
plt.rcParams['figure.figsize'] = (5,5)
sns.heatmap(confusion_matrix(Y_test, Y_pred_knn), annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[365…]
```
<AxesSubplot:>
```



Accuracy

In [366…]
```
print("Accuracy Score:",accuracy_score(Y_test, Y_pred_knn))
```

```
Accuracy Score: 0.9129613957424647
```

Refine KNN Model

Optimise hyperparameter value

In [367…]
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

In [368…]
```
knn_params = {"n_neighbors": np.arange(1,20)}
```

In [369…]
```
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, knn_params, cv=5)
knn_cv.fit(X_train, Y_train)
```

Out[369…]
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19])})
```

In [370…]
```
print("Best score: " + str(knn_cv.best_score_))
print("Best params: " + str(knn_cv.best_params_))
```

```
Best score: 0.9192229571473559
Best params: {'n_neighbors': 18}
```

Insert the best hyperparameter via Gridsearch and CV

In [371…]
```
knn = KNeighborsClassifier(5)
knn_tuned = knn.fit(X_test, Y_test)
```

In [372…]
```
knn_tuned.score(X_train, Y_train)
```

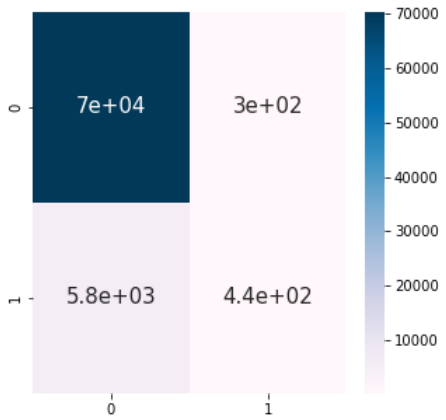
Out[372…]
```
0.9128344009860597
```

In [373…]
```
y_pred_tuned = knn_tuned.predict(X_test)
```

Confusion Matrix

In [374…]
```
plt.rcParams['figure.figsize'] = (5,5)
sns.heatmap(confusion_matrix(Y_test, y_pred_tuned), annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[374… <AxesSubplot:>



Summmary and findings

In this assignment, we have only used Logistic Regression and K-Nearest Neighbour Algorithm on the dataset. From the training of the models, the testing accuracy of the Logistic Regression is 0.9198, while that of KNN Model is 0.9136 before fine-tuning, and 0.9132 after fine-tuning.

It can be seen that the Logistic Regression Model is more accurate than the KNN Model.

Limitation of dataset

It should be noted that there were numerous data with empty or NA fields. These invalid data may skew the output results, which may lead to a less inaccurate finding.

Further improvements

We can perform fine-tuning on the logistic regression model by performing cross validation in order to increase the accuracy of the logistic regression model.

Separately, we can also make use of ensemble models by combining the predictions of several models to improve the accuracy of the results. These includes using Random Forest or XG Boost.

For data with empty or NA fields, they should be investigated by the data collection officer if there are uncaptured information or incomplete information provided by the interviewees or processed by the interviewers.