

***Faculty of Science and Technology***

**Assignment Coversheet**

|  |  |
| --- | --- |
| **Student ID number &**  **Student Name** | u3244564  Doniyor Musurmanov |
| **Unit name** | Software Technology 1 |
| **Unit number** | 4438 |
| **Unit Tutor** | Gurpreet Kaur |
| **Assignment name** | ST1 Capstone Project – COLT 1 2023 |
| **Due date** | 30, April 2023 |
| **Date submitted** | 30, April 2023 |

**You must keep a photocopy or electronic copy of your assignment.**

**Student declaration**

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

**A close-up of a logo

Description automatically generated with low confidence**

**Signature of student: \_\_\_\_\_\_\_\_\_\_\_ Date:** 30, April 2023

Table of Contents

[Introduction 1](#_Toc131590041)

[Methodology 2](#_Toc131590042)

[Stage 1: Algorithm Design Stage 2](#_Toc131590043)

[Dataset Description 3](#_Toc131590044)

[Exploratory Data Analysis 3](#_Toc131590045)

[Predictive Data Analytics Stage 11](#_Toc131590046)

[Model Preparation and Development 14](#_Toc131590047)

[Stage 2: Algorithm Implementation Stage 22](#_Toc131590048)

[Stage 3: Software Deployment Stage 22](#_Toc131590049)

[Conclusions 23](#_Toc131590050)

[References 23](#_Toc131590051)

# Introduction

This report describes the details of Python Capstone Project for ST1 unit within the scope of the project requirements provided in the assignment handout. I have decided to work on the project using a Credit Card Approvals dataset available in Kaggle data repositories.

Credit card approval is a crucial process in the banking industry, where financial institutions assess the creditworthiness of their customers before issuing them credit cards. In the United States alone, millions of credit card applications are submitted every year, and it is the responsibility of the banks to ensure that they only approve those that meet their credit standards.

Traditionally, credit card approval has been based on a number of factors, such as credit history, income, and debt-to-income ratio. However, this process can be time-consuming and prone to errors, and therefore there is a need to develop a more efficient and accurate method.

In this report, we present the details of a prototype software platform for credit card approval, which utilizes a data-driven scientific approach. This platform includes several Python software tools that were developed as part of a capstone project. The platform involves exploratory data analysis, predictive analytics, and implementation as a desktop Tkinter application, as well as an online web-based Flask/Streamlit application.

The proposed platform aims to improve the accuracy and efficiency of credit card approval by utilizing machine learning algorithms to analyze customer data and predict their creditworthiness. By incorporating various customer attributes such as credit history, income, and debt-to-income ratio, the platform can provide more accurate credit assessments than traditional methods.

Overall, this report presents a promising solution to the challenges faced by the banking industry in credit card approval. With further development and testing, this platform has the potential to become an industry standard, revolutionizing the way credit card approvals are processed.

# Methodology

The methodology used for developing the software platform involves 3 stages as outlined below:

1. Design and development of decision support algorithms based on exploratory data analysis and predictive analytics, for identifying the best performing algorithm for solving a real world problem.
2. Implementation of best performing algorithm as a desktop Tkinter software tool.
3. Deployment of the tool as a web or cloud enabled platform tool.

## Stage 1: Algorithm Design Stage

Stage 1 is most important preliminary stage and depending on the complexity of the problem and dataset used, the design of algorithms for exploratory data analysis and predictive analytics algorithms will vary.

### Dataset Description

There is only one dataset used for this project and it is publicly available from Kaggle. The dataset consists of 691 observations, 15 features and 1 target/class attribute.

The dataset contains information about credit card applicants, including their gender, age, debt, marital status, bank customer status, industry, ethnicity, years employed, prior default status, employment status, credit score, driver's license status, citizenship status, zip code, income, and whether their credit card application was approved or not.

Each row in the dataset represents a different credit card application, with a total of 16 attributes and one target variable (Approved), which is a binary variable indicating whether the application was approved or not.

The Gender attribute represents the applicant's gender, with possible values of Male or Female. The Age attribute represents the applicant's age in years, while the Debt attribute represents the total debt owed by the applicant. The Married attribute represents the applicant's marital status, with possible values of Married or Single.

The BankCustomer attribute represents whether the applicant is an existing bank customer, with possible values of Yes or No. The Industry attribute represents the applicant's industry of employment, while the Ethnicity attribute represents the applicant's ethnicity.

The YearsEmployed attribute represents the number of years the applicant has been employed, while the PriorDefault attribute represents whether the applicant has previously defaulted on a loan, with possible values of Yes or No. The Employed attribute represents the applicant's current employment status, with possible values of Employed or Unemployed.

The CreditScore attribute represents the applicant's credit score, while the DriversLicense attribute represents whether the applicant has a driver's license, with possible values of Yes or No. The Citizen attribute represents the applicant's citizenship status, with possible values of Citizen or Non-Citizen.

The ZipCode attribute represents the applicant's zip code, while the Income attribute represents the applicant's income in dollars. The target variable, Approved, represents whether the credit card application was approved, with possible values of + (approved) or - (not approved).

### Exploratory Data Analysis The initial stage of the software development process involved comprehending the data, conducting basic exploratory data analysis, and creating visualizations. Google Colab was selected as the experimental environment due to its virtual hardware and resources, which eliminate the need for additional physical hardware and can be accessed directly from a web browser. The programming language used for this project was Python, and the scripts were created using an online Jupyter notebook via Google Colab, which can be accessed using a free Google account. The notebook files were saved virtually on Google Drive without the need for any additional configurations. Prior to conducting the exploratory data analysis, certain Python libraries for EDA were imported, and the dataset was obtained using the following Python script.

### from google.colab import drive

drive.mount("/content/drive")

#Import Required Packages for EDA

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

import plotly.graph\_objects as go

import plotly.express as px

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

#Read the dataset/s

df = pd.read\_csv('/content/drive/…../heart.csv')

1. The EDA starts with understanding the basic description of data as described next:

#1. Checking description(first 5 and last 5 rows)

df.head()

A picture containing application

Description automatically generated

df.tail() #last 5 rows

Graphical user interface, application

Description automatically generated

#rows and columns-data shape(attributes & samples)

df.shape

(690, 16)

# name of the attributes

df.columns

Text

Description automatically generated with medium confidence

#unique values for each attribute

df.nunique()

A picture containing table

Description automatically generated

#Complete info about data frame

df.info()

Table

Description automatically generated

#3. Visualising data  distribution in detail

fig = plt.figure(figsize =(16,16))

ax=fig.gca()

df.hist(ax=ax,bins =30)

plt.show()

Chart

Description automatically generated

Background pattern

Description automatically generated with low confidence

#detecting outliers

df.plot(kind='box', subplots=True,

        layout=(2,7),sharex=False,sharey=False, figsize=(20, 10), color='deeppink');

A picture containing chart

Description automatically generated

#identify the outliers

# define continuous variable & plot

continous\_features = ['Age','Debt','YearsEmployed', 'CreditScore','ZipCode', 'Income']

def outliers(df\_out, drop = False):

    for each\_feature in df\_out.columns:

        feature\_data = df\_out[each\_feature]

        Q1 = np.percentile(feature\_data, 25.) # 25th percentile of the data of the given feature

        Q3 = np.percentile(feature\_data, 75.) # 75th percentile of the data of the given feature

        IQR = Q3-Q1 #Interquartile Range

        outlier\_step = IQR \* 1.5 #That's we were talking about above

        outliers = feature\_data[~((feature\_data >= Q1 - outlier\_step) & (feature\_data <= Q3 + outlier\_step))].index.tolist()

        if not drop:

            print('For the feature {}, No of Outliers is {}'.format(each\_feature, len(outliers)))

        if drop:

            df.drop(outliers, inplace = True, errors = 'ignore')

            print('Outliers from {} feature removed'.format(each\_feature))

outliers(df[continous\_features])

Text

Description automatically generated

#drop the outliers

outliers(df[continous\_features], drop = True)

Text

Description automatically generated

#check if outliers got removed

df.plot(kind='box', subplots=True,

        layout=(2,7),sharex=False,sharey=False, figsize=(20, 10), color='deeppink');

A picture containing calendar

Description automatically generated

#Check data shape after outlier removal

df.shape

(473, 16)

#checking target value distribution

print(df.Approved.value\_counts())

fig, ax = plt.subplots(figsize=(5,4))

name = ["Approved", "Not\_Approved"]

ax = df.Approved.value\_counts().plot(kind='bar')

ax.set\_title("Credit Card Approval Classes", fontsize = 13, weight = 'bold')

ax.set\_xticklabels (name, rotation = 0)

# To calculate the percentage

totals = []

for i in ax.patches:

    totals.append(i.get\_height())

total = sum(totals)

for i in ax.patches:

    ax.text(i.get\_x()+.09, i.get\_height()-50, \

            str(round((i.get\_height()/total)\*100, 2))+'%', fontsize=14,

                color='white', weight = 'bold')

plt.tight\_layout()

Chart, bar chart

Description automatically generated

#check correlation between variables

sns.set(style="white")

plt.rcParams['figure.figsize'] = (15, 10)

sns.heatmap(df.corr(), annot = True, linewidths=.5, cmap="Blues")

plt.title('Corelation Between Variables', fontsize = 30)

plt.show()

A picture containing chart

Description automatically generated

!pip install <https://github.com/pandas-profiling/pandas-profiling/archive/master.zip>

#obtain full profiler report

#restart kernel

#re-run import libraries and data

import pandas as pd

import numpy as np

from pandas\_profiling import ProfileReport

profile = ProfileReport(df,title="Heart Disease EDA",

                        html={'style':{'full\_width':True}})

profile.to\_notebook\_iframe()

Graphical user interface

Description automatically generated with low confidence

### Predictive Data Analytics Stage

For predictive analytics, several processing steps are required. These include pre-processing, classifier comparison to identify the best machine learning classifier and performance evaluation with different objective metrics, such as accuracy, classification report, confusion matrix, ROC-AUC curve and prediction report was obtained using the Python scikit-learn package. Each of these steps are described next.

* Pre-processing: Since the dataset consists of a combination of continuous and categorical attributes/variables, there is a need to pre-process the data with attribute transformation, standardization and normalisation. We used scikit-learn’s OrdinalEncoder() function to perform attribute transformation.
* Normalisation of the independent values of the dataframe by was done by dropping the target from the dataframe, normalising it, and then reattaching the target to the dataframe:-

#pre-processing

from sklearn.exceptions import DataDimensionalityWarning

#encode object columns to integers

from sklearn import preprocessing

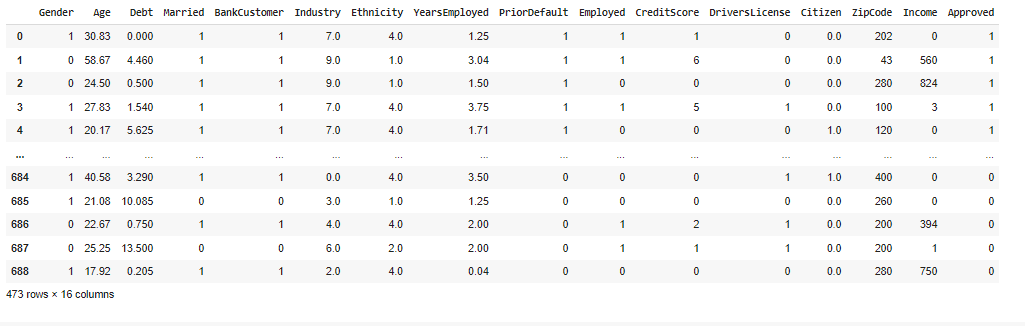
from sklearn.preprocessing import OrdinalEncoder

for col in df:

  if df[col].dtype =='object':

    df[col]=OrdinalEncoder().fit\_transform(df[col].values.reshape(-1,1))

df



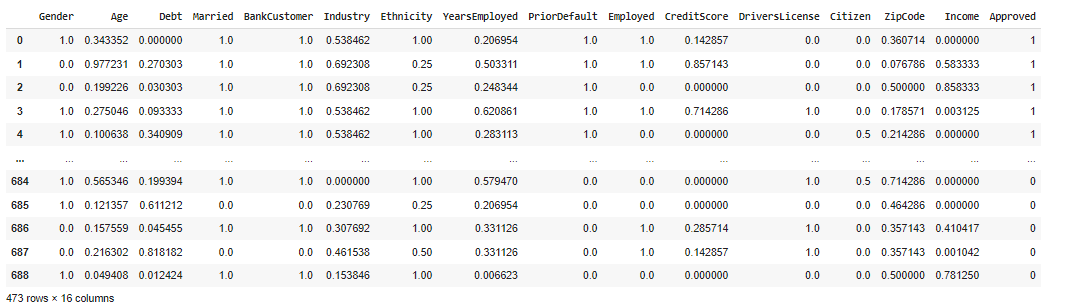
class\_label =df['hd']

df = df.drop(['hd'], axis =1)

df = (df-df.min())/(df.max()-df.min())

df['hd']=class\_label

df



#pre-processing

cc\_data = df.copy()

le = preprocessing.LabelEncoder()

gender = le.fit\_transform(list(cc\_data["Gender"])) # gender (1 = male; 0 = female)

age = le.fit\_transform(list(cc\_data["Age"])) # age in years

debt = le.fit\_transform(list(cc\_data["Debt"])) # debt amount

married = le.fit\_transform(list(cc\_data["Married"])) # marital status (1 = yes; 0 = no)

bankcustomer = le.fit\_transform(list(cc\_data["BankCustomer"])) # bank customer (1 = yes; 0 = no)

industry = le.fit\_transform(list(cc\_data["Industry"])) # type of industry

ethnicity = le.fit\_transform(list(cc\_data["Ethnicity"])) # ethnicity

yearsemployed = le.fit\_transform(list(cc\_data["YearsEmployed"])) # years employed

priordefault = le.fit\_transform(list(cc\_data["PriorDefault"])) # previous credit default (1 = yes; 0 = no)

employed = le.fit\_transform(list(cc\_data["Employed"])) # employed status (1 = yes; 0 = no)

creditscore = le.fit\_transform(list(cc\_data["CreditScore"])) # credit score

driverslicense = le.fit\_transform(list(cc\_data["DriversLicense"])) # driver's license (1 = yes; 0 = no)

citizen = le.fit\_transform(list(cc\_data["Citizen"])) # citizenship status

zipcode = le.fit\_transform(list(cc\_data["ZipCode"])) # zip code

income = le.fit\_transform(list(cc\_data["Income"])) # income

approved = le.fit\_transform(list(cc\_data["Approved"])) # credit card approval (1 = approved; 0 = not approved)

### Model Preparation and Development

Steps used for machine learning model preparation are described below:

* + Convert the dataframe to training and validation/test subsets by taking a random sample of 80% of the data and defining it as train subset. This leaves 20% of the data for validation/testing
  + Create the validation/test set by dropping all of the rows that comprise the training set from the dataframe.
  + Create y\_train by using using the last column of train (target class).
  + Create x\_train by using all of the columns in train except the last one.
  + The validation set of y\_val and x\_val or (y\_test and x\_test), can be created using the same methodology that used to create y\_train and x\_train

import sklearn.model\_selection

x = list(zip(gender,  age,  debt, married,  bankcustomer, industry, ethnicity,  yearsemployed,  priordefault, employed, creditscore,  driverslicense, citizen,  zipcode,  income))

y = list(approved)

# Test options and evaluation metric

num\_folds = 5

seed = 7

scoring = 'accuracy'

# Model Test/Train

# Splitting what we are trying to predict into 4 different arrays -

# X train is a section of the x array(attributes) and vise versa for Y(features)

# The test data will test the accuracy of the model created

x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(x, y, test\_size = 0.20, random\_state=seed)

#splitting 20% of our data into test samples. If we train the model with higher data it already has seen that information and knows

#size of train and test subsets after splitting

np.shape(x\_train), np.shape(x\_test)

((227, 13), (57, 13))

# Predictive analytics model development by comparing different Scikit-learn classification algorithms

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import ExtraTreesClassifier

models = []

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC()))

models.append(('GBM', GradientBoostingClassifier()))

models.append(('RF', RandomForestClassifier()))

# evaluate each model in turn

results = []

names = []

print("Performance on Training set")

for name, model in models:

  kfold = KFold(n\_splits=num\_folds,shuffle=True,random\_state=seed)

  cv\_results = cross\_val\_score(model, x\_train, y\_train, cv=kfold, scoring='accuracy')

  results.append(cv\_results)

  names.append(name)

  msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

  msg += '\n'

  print(msg)

|  |
| --- |
| Performance on Training set |
| NB: 0.856888 (0.038956) |
|  |
| SVM: 0.722752 (0.042235) |
|  |
| GBM: 0.847879 (0.039126) |
|  |
| RF: 0.865962 (0.025178) |

# Compare Algorithms' Performance

fig = plt.figure()

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

Chart, box and whisker chart

Description automatically generated

#Model Evaluation by testing with independent/external test data set.

# Make predictions on validation/test dataset

#Model Evaluation by testing with independent/external test data set.

# Make predictions on validation/test dataset

models.append(('DT', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC()))

models.append(('GBM', GradientBoostingClassifier()))

models.append(('RF', RandomForestClassifier()))

dt = DecisionTreeClassifier()

nb = GaussianNB()

gb = GradientBoostingClassifier()

rf = RandomForestClassifier()

best\_model = rf

best\_model.fit(x\_train, y\_train)

y\_pred = best\_model.predict(x\_test)

print("Best Model Accuracy Score on Test Set:", accuracy\_score(y\_test, y\_pred))

Best Model Accuracy Score on Test Set: 0.7894736842105263

#Model Performance Evaluation Metric 1 - Classification Report

print(classification\_report(y\_test, y\_pred))

Table

Description automatically generated

#Model Performance Evaluation Metric 2

#Confusion matrix

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

plt.show()

Chart, square

Description automatically generated

#Model Evaluation Metric 3- ROC-AUC curve

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

best\_model = rf

best\_model.fit(x\_train, y\_train)

rf\_roc\_auc = roc\_auc\_score(y\_test,best\_model.predict(x\_test))

fpr,tpr,thresholds = roc\_curve(y\_test, best\_model.predict\_proba(x\_test)[:,1])

plt.figure()

plt.plot(fpr,tpr,label = 'Random Forest(area = %0.2f)'% rf\_roc\_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('Receiver Operating Characteristic')

plt.legend(loc='lower right')

plt.savefig('LOC\_ROC')

plt.show()

Chart, line chart

Description automatically generated

#Model Evaluation Metric 4-prediction report

for x in range(len(y\_pred)):

  print("Predicted: ", y\_pred[x], "Actual: ", y\_test[x], "Data: ", x\_test[x],)

Predicted: 0 Actual: 0 Data: (0, 308, 5, 1, 1, 6, 2, 0, 0, 0, 0, 1, 1, 18, 25)

Predicted: 0 Actual: 1 Data: (0, 22, 12, 0, 1, 8, 2, 114, 0, 0, 0, 1, 1, 100, 0)

Predicted: 1 Actual: 1 Data: (0, 19, 163, 1, 1, 13, 1, 25, 1, 1, 11, 0, 0, 0, 199)

Predicted: 0 Actual: 1 Data: (1, 283, 115, 1, 1, 5, 4, 1, 0, 0, 0, 0, 0, 0, 195)

Predicted: 1 Actual: 1 Data: (1, 51, 99, 0, 0, 7, 4, 51, 1, 1, 1, 1, 0, 23, 93)

Predicted: 1 Actual: 1 Data: (0, 203, 30, 1, 1, 2, 4, 19, 1, 1, 4, 0, 0, 0, 174)

Predicted: 0 Actual: 0 Data: (1, 125, 32, 0, 0, 10, 1, 64, 0, 0, 0, 0, 0, 60, 116)

Predicted: 1 Actual: 1 Data: (1, 320, 176, 1, 1, 0, 4, 30, 1, 1, 4, 1, 0, 67, 0)

Predicted: 0 Actual: 0 Data: (1, 282, 90, 1, 1, 13, 0, 2, 0, 0, 0, 1, 0, 154, 0)

Predicted: 0 Actual: 0 Data: (1, 102, 77, 1, 1, 4, 1, 52, 0, 1, 1, 1, 0, 133, 21)

Predicted: 1 Actual: 1 Data: (1, 61, 163, 1, 1, 13, 4, 7, 1, 1, 6, 0, 0, 38, 0)

Predicted: 0 Actual: 0 Data: (1, 212, 54, 1, 1, 1, 4, 6, 0, 0, 0, 1, 0, 76, 0)

Predicted: 1 Actual: 1 Data: (1, 212, 77, 1, 1, 9, 1, 112, 1, 0, 0, 1, 0, 32, 126)

Predicted: 0 Actual: 0 Data: (1, 53, 29, 0, 0, 4, 1, 6, 0, 0, 0, 0, 0, 96, 77)

Predicted: 1 Actual: 1 Data: (1, 191, 91, 1, 1, 1, 0, 70, 1, 1, 2, 0, 0, 94, 125)

Predicted: 0 Actual: 0 Data: (1, 132, 1, 0, 0, 10, 4, 77, 0, 0, 0, 1, 0, 144, 0)

Predicted: 1 Actual: 0 Data: (0, 40, 25, 1, 1, 4, 4, 21, 1, 1, 5, 1, 0, 44, 5)

Predicted: 0 Actual: 0 Data: (1, 140, 39, 1, 1, 7, 4, 42, 0, 0, 0, 0, 0, 67, 0)

Predicted: 1 Actual: 1 Data: (1, 86, 29, 1, 1, 9, 4, 82, 1, 1, 2, 1, 0, 154, 179)

Predicted: 0 Actual: 0 Data: (1, 66, 15, 1, 1, 5, 4, 3, 0, 0, 0, 1, 0, 96, 37)

Predicted: 0 Actual: 0 Data: (0, 321, 91, 0, 0, 6, 2, 84, 0, 0, 0, 1, 0, 0, 4)

Predicted: 1 Actual: 1 Data: (0, 160, 84, 1, 1, 8, 1, 16, 1, 1, 9, 1, 0, 63, 0)

Predicted: 1 Actual: 1 Data: (1, 266, 16, 1, 1, 1, 1, 84, 1, 0, 0, 1, 0, 105, 0)

Predicted: 0 Actual: 0 Data: (1, 66, 138, 1, 1, 2, 4, 2, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (0, 87, 65, 0, 0, 1, 0, 80, 0, 0, 0, 0, 0, 67, 110)

Predicted: 1 Actual: 1 Data: (1, 256, 100, 0, 0, 7, 4, 68, 1, 0, 0, 1, 0, 15, 172)

Predicted: 1 Actual: 1 Data: (1, 208, 57, 0, 0, 7, 4, 2, 1, 1, 1, 0, 0, 14, 156)

Predicted: 1 Actual: 1 Data: (0, 31, 143, 1, 1, 7, 4, 39, 1, 1, 6, 1, 0, 10, 124)

Predicted: 0 Actual: 0 Data: (1, 171, 191, 0, 0, 6, 2, 0, 0, 1, 2, 0, 0, 52, 1)

Predicted: 1 Actual: 1 Data: (1, 149, 16, 1, 1, 4, 4, 42, 1, 1, 11, 0, 0, 9, 117)

Predicted: 0 Actual: 0 Data: (1, 41, 164, 1, 1, 4, 4, 46, 0, 0, 0, 1, 0, 43, 0)

Predicted: 0 Actual: 0 Data: (1, 31, 0, 1, 1, 9, 4, 17, 0, 0, 0, 0, 0, 52, 1)

Predicted: 1 Actual: 1 Data: (1, 72, 163, 1, 1, 9, 4, 57, 1, 1, 7, 1, 0, 31, 139)

Predicted: 1 Actual: 1 Data: (1, 113, 91, 1, 1, 8, 4, 78, 1, 1, 1, 1, 0, 37, 0)

Predicted: 1 Actual: 1 Data: (1, 113, 51, 0, 0, 4, 4, 25, 1, 1, 5, 1, 0, 52, 217)

Predicted: 1 Actual: 0 Data: (1, 39, 150, 0, 0, 5, 1, 22, 1, 0, 0, 1, 0, 44, 0)

Predicted: 0 Actual: 0 Data: (0, 74, 39, 1, 1, 9, 4, 6, 0, 0, 0, 1, 0, 37, 139)

Predicted: 0 Actual: 1 Data: (1, 108, 9, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (1, 92, 69, 1, 1, 4, 4, 51, 1, 1, 6, 0, 0, 0, 124)

Predicted: 0 Actual: 0 Data: (0, 324, 54, 1, 1, 6, 2, 98, 0, 1, 1, 0, 0, 0, 10)

Predicted: 1 Actual: 1 Data: (1, 171, 5, 0, 0, 4, 1, 70, 1, 1, 1, 1, 0, 137, 224)

Predicted: 0 Actual: 0 Data: (0, 82, 27, 1, 1, 9, 4, 10, 0, 1, 1, 1, 0, 67, 11)

Predicted: 0 Actual: 0 Data: (1, 83, 174, 1, 1, 4, 4, 48, 0, 0, 0, 0, 1, 23, 0)

Predicted: 0 Actual: 0 Data: (1, 105, 29, 1, 1, 5, 4, 9, 0, 1, 2, 1, 0, 59, 3)

Predicted: 1 Actual: 1 Data: (1, 76, 65, 1, 1, 4, 4, 27, 1, 1, 11, 1, 0, 17, 185)

Predicted: 0 Actual: 0 Data: (1, 78, 181, 1, 1, 4, 4, 3, 0, 1, 2, 0, 0, 0, 216)

Predicted: 0 Actual: 0 Data: (0, 11, 16, 1, 1, 1, 4, 4, 0, 1, 6, 1, 0, 84, 29)

Predicted: 0 Actual: 0 Data: (1, 245, 39, 0, 0, 7, 4, 6, 0, 0, 0, 0, 0, 0, 74)

Predicted: 1 Actual: 0 Data: (0, 133, 32, 1, 1, 9, 4, 25, 1, 1, 2, 1, 0, 55, 114)

Predicted: 1 Actual: 1 Data: (0, 153, 118, 1, 1, 4, 0, 75, 1, 1, 7, 1, 0, 0, 200)

Predicted: 0 Actual: 0 Data: (0, 6, 11, 1, 1, 6, 2, 0, 0, 1, 1, 0, 0, 52, 58)

Predicted: 1 Actual: 1 Data: (1, 43, 133, 1, 1, 2, 4, 45, 1, 1, 14, 0, 0, 17, 67)

Predicted: 1 Actual: 0 Data: (1, 57, 168, 1, 1, 1, 4, 13, 1, 0, 0, 1, 0, 31, 42)

Predicted: 1 Actual: 1 Data: (1, 47, 63, 1, 1, 4, 4, 46, 1, 1, 11, 1, 0, 67, 199)

Predicted: 0 Actual: 0 Data: (1, 172, 65, 1, 1, 4, 4, 30, 0, 0, 0, 1, 0, 96, 0)

Predicted: 1 Actual: 1 Data: (1, 338, 205, 1, 1, 13, 1, 128, 1, 1, 9, 1, 0, 0, 149)

Predicted: 1 Actual: 0 Data: (1, 269, 45, 1, 1, 4, 4, 57, 1, 0, 0, 1, 0, 44, 0)

Predicted: 0 Actual: 0 Data: (1, 36, 125, 1, 1, 0, 4, 1, 0, 1, 1, 0, 0, 31, 1)

Predicted: 0 Actual: 0 Data: (1, 234, 48, 1, 1, 4, 4, 36, 0, 0, 0, 0, 0, 0, 95)

Predicted: 1 Actual: 0 Data: (1, 138, 81, 1, 1, 7, 4, 72, 1, 1, 3, 1, 0, 106, 0)

Predicted: 0 Actual: 0 Data: (1, 159, 45, 0, 0, 7, 4, 1, 0, 0, 0, 0, 1, 52, 0)

Predicted: 0 Actual: 0 Data: (0, 60, 172, 1, 1, 4, 4, 6, 0, 0, 0, 1, 0, 62, 0)

Predicted: 0 Actual: 0 Data: (0, 215, 65, 1, 1, 1, 1, 5, 0, 0, 0, 0, 0, 90, 85)

Predicted: 1 Actual: 1 Data: (1, 71, 47, 0, 0, 7, 4, 66, 1, 1, 6, 0, 0, 23, 0)

Predicted: 1 Actual: 1 Data: (0, 101, 0, 1, 1, 4, 4, 0, 0, 0, 0, 0, 2, 0, 0)

Predicted: 0 Actual: 0 Data: (1, 1, 122, 1, 1, 3, 4, 25, 0, 0, 0, 0, 0, 159, 0)

Predicted: 1 Actual: 1 Data: (1, 27, 54, 1, 1, 1, 4, 36, 1, 1, 2, 0, 0, 37, 93)

Predicted: 1 Actual: 1 Data: (1, 46, 126, 1, 1, 5, 4, 36, 1, 1, 1, 0, 0, 52, 82)

Predicted: 0 Actual: 0 Data: (1, 48, 28, 0, 0, 4, 4, 46, 0, 0, 0, 1, 1, 84, 0)

Predicted: 1 Actual: 1 Data: (1, 238, 118, 1, 1, 2, 0, 72, 1, 1, 1, 0, 0, 0, 114)

Predicted: 0 Actual: 0 Data: (0, 199, 84, 1, 1, 6, 2, 0, 0, 1, 6, 0, 0, 0, 76)

Predicted: 0 Actual: 0 Data: (1, 172, 126, 1, 1, 3, 0, 38, 1, 0, 0, 1, 1, 134, 0)

Predicted: 0 Actual: 0 Data: (1, 108, 28, 1, 1, 8, 4, 28, 0, 0, 0, 0, 0, 31, 0)

Predicted: 0 Actual: 0 Data: (1, 74, 6, 1, 1, 0, 4, 2, 0, 0, 0, 0, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (1, 5, 75, 1, 1, 7, 4, 2, 0, 1, 1, 0, 0, 0, 6)

Predicted: 0 Actual: 1 Data: (1, 218, 0, 1, 1, 4, 4, 0, 0, 0, 0, 0, 2, 0, 0)

Predicted: 0 Actual: 0 Data: (1, 42, 9, 1, 1, 9, 4, 3, 0, 0, 0, 0, 0, 67, 0)

Predicted: 0 Actual: 0 Data: (1, 53, 0, 1, 1, 4, 4, 13, 0, 0, 0, 1, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (0, 75, 53, 1, 1, 11, 3, 0, 0, 1, 1, 0, 0, 67, 36)

Predicted: 1 Actual: 1 Data: (1, 64, 163, 1, 1, 8, 4, 17, 1, 0, 0, 0, 0, 31, 0)

Predicted: 0 Actual: 0 Data: (0, 306, 122, 1, 1, 2, 1, 64, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 1 Data: (1, 54, 45, 1, 1, 7, 4, 36, 0, 0, 0, 0, 0, 48, 8)

Predicted: 0 Actual: 0 Data: (1, 111, 68, 0, 0, 4, 4, 2, 0, 0, 0, 0, 1, 23, 0)

Predicted: 0 Actual: 0 Data: (1, 223, 23, 1, 1, 4, 4, 9, 0, 1, 2, 0, 0, 78, 114)

Predicted: 1 Actual: 1 Data: (1, 220, 153, 1, 1, 13, 4, 57, 1, 1, 6, 0, 0, 154, 75)

Predicted: 1 Actual: 1 Data: (1, 79, 45, 1, 1, 4, 1, 34, 1, 0, 0, 0, 0, 135, 76)

Predicted: 1 Actual: 1 Data: (1, 132, 86, 1, 1, 13, 4, 84, 1, 1, 3, 1, 0, 98, 189)

Predicted: 1 Actual: 0 Data: (0, 69, 135, 1, 1, 9, 4, 42, 1, 1, 10, 0, 0, 23, 148)

Predicted: 1 Actual: 1 Data: (0, 295, 180, 1, 1, 2, 4, 52, 1, 1, 3, 1, 0, 51, 0)

Predicted: 0 Actual: 0 Data: (1, 90, 186, 0, 0, 6, 2, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (1, 117, 25, 1, 1, 4, 1, 77, 1, 1, 3, 1, 0, 104, 65)

Predicted: 0 Actual: 0 Data: (1, 28, 156, 1, 1, 6, 2, 10, 0, 0, 0, 0, 0, 23, 0)

Predicted: 0 Actual: 0 Data: (1, 22, 121, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 44, 0)

Predicted: 1 Actual: 1 Data: (1, 188, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 96, 179)

Predicted: 1 Actual: 1 Data: (1, 51, 72, 0, 0, 10, 4, 27, 1, 1, 8, 1, 0, 52, 1)

Predicted: 1 Actual: 1 Data: (1, 197, 65, 1, 1, 7, 4, 0, 1, 0, 0, 1, 0, 83, 76)

Predicted: 0 Actual: 1 Data: (0, 47, 173, 1, 1, 4, 1, 95, 1, 0, 0, 0, 0, 109, 0)

Predicted: 0 Actual: 0 Data: (0, 23, 150, 0, 0, 9, 1, 4, 0, 0, 0, 0, 0, 109, 0)

Predicted: 1 Actual: 1 Data: (0, 136, 84, 1, 1, 4, 4, 27, 1, 1, 1, 1, 0, 119, 0)

Predicted: 1 Actual: 0 Data: (1, 148, 33, 0, 0, 1, 0, 13, 1, 1, 10, 1, 0, 42, 26)

Predicted: 0 Actual: 0 Data: (1, 123, 25, 1, 1, 9, 1, 4, 0, 0, 0, 1, 0, 75, 87)

Predicted: 1 Actual: 1 Data: (0, 281, 131, 1, 1, 4, 4, 105, 1, 1, 6, 1, 0, 0, 163)

Predicted: 0 Actual: 0 Data: (1, 227, 51, 1, 1, 5, 4, 13, 0, 0, 0, 1, 0, 100, 2)

Predicted: 1 Actual: 1 Data: (0, 135, 33, 1, 1, 4, 4, 57, 1, 1, 5, 1, 0, 100, 171)

Predicted: 0 Actual: 0 Data: (1, 191, 41, 1, 1, 1, 0, 3, 0, 0, 0, 1, 0, 139, 209)

Predicted: 1 Actual: 1 Data: (0, 17, 138, 1, 1, 2, 4, 33, 1, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (1, 81, 69, 1, 1, 6, 2, 80, 0, 0, 0, 0, 0, 52, 24)

Predicted: 1 Actual: 0 Data: (0, 69, 163, 0, 0, 9, 4, 64, 1, 0, 0, 1, 0, 92, 0)

Predicted: 0 Actual: 0 Data: (1, 235, 101, 1, 1, 6, 2, 0, 0, 1, 2, 0, 0, 1, 1)

Predicted: 0 Actual: 0 Data: (1, 36, 45, 0, 0, 8, 4, 46, 1, 0, 0, 1, 0, 31, 20)

Predicted: 0 Actual: 0 Data: (0, 50, 16, 0, 0, 3, 3, 25, 0, 0, 0, 0, 0, 90, 0)

Predicted: 0 Actual: 0 Data: (0, 97, 179, 1, 1, 10, 4, 25, 0, 0, 0, 1, 0, 62, 152)

Predicted: 0 Actual: 0 Data: (1, 258, 39, 1, 1, 0, 4, 122, 0, 1, 1, 1, 0, 113, 53)

Predicted: 1 Actual: 0 Data: (0, 314, 124, 1, 1, 5, 1, 99, 1, 1, 3, 1, 0, 31, 35)

Predicted: 0 Actual: 0 Data: (1, 184, 39, 1, 1, 7, 4, 28, 0, 0, 0, 0, 0, 37, 0)

Predicted: 0 Actual: 0 Data: (0, 71, 26, 1, 1, 1, 4, 2, 0, 0, 0, 0, 0, 46, 0)

Predicted: 1 Actual: 1 Data: (1, 340, 163, 1, 1, 3, 3, 130, 1, 1, 7, 1, 0, 4, 0)

Predicted: 0 Actual: 0 Data: (1, 12, 9, 1, 1, 9, 4, 8, 0, 1, 4, 0, 0, 52, 8)

Predicted: 0 Actual: 0 Data: (1, 267, 122, 0, 0, 4, 4, 39, 0, 0, 0, 0, 0, 52, 2)

Predicted: 0 Actual: 0 Data: (0, 120, 49, 1, 1, 6, 2, 0, 0, 0, 0, 0, 0, 109, 1)

Predicted: 0 Actual: 0 Data: (1, 178, 65, 0, 0, 7, 4, 100, 0, 0, 0, 1, 0, 96, 0)

Predicted: 0 Actual: 0 Data: (1, 84, 19, 0, 0, 8, 1, 3, 0, 0, 0, 0, 0, 84, 1)

Predicted: 0 Actual: 0 Data: (1, 173, 72, 1, 1, 10, 4, 4, 0, 0, 0, 1, 0, 37, 0)

Predicted: 1 Actual: 1 Data: (1, 182, 51, 1, 1, 13, 1, 80, 1, 1, 4, 1, 0, 87, 143)

Predicted: 1 Actual: 1 Data: (0, 48, 53, 1, 1, 9, 4, 48, 1, 1, 5, 0, 0, 75, 193)

Predicted: 0 Actual: 0 Data: (1, 184, 58, 1, 1, 4, 4, 36, 0, 0, 0, 0, 2, 37, 0)

Predicted: 0 Actual: 0 Data: (1, 210, 91, 1, 1, 9, 4, 72, 0, 0, 0, 0, 0, 142, 35)

Predicted: 1 Actual: 0 Data: (0, 99, 36, 1, 1, 9, 4, 31, 1, 1, 2, 0, 0, 67, 0)

Predicted: 1 Actual: 1 Data: (1, 190, 72, 1, 1, 8, 1, 102, 1, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (1, 34, 86, 0, 0, 1, 4, 25, 0, 0, 0, 1, 0, 115, 149)

Predicted: 1 Actual: 1 Data: (1, 234, 91, 1, 1, 4, 0, 98, 1, 1, 16, 0, 0, 36, 159)

Predicted: 1 Actual: 1 Data: (0, 244, 55, 0, 0, 9, 1, 3, 1, 1, 20, 1, 0, 143, 161)

Predicted: 0 Actual: 1 Data: (1, 213, 4, 0, 0, 4, 4, 36, 0, 0, 0, 1, 0, 82, 54)

Predicted: 0 Actual: 0 Data: (0, 284, 41, 1, 1, 1, 3, 8, 0, 0, 0, 0, 0, 0, 56)

Predicted: 0 Actual: 0 Data: (0, 2, 12, 1, 1, 4, 4, 25, 0, 0, 0, 0, 0, 37, 18)

Predicted: 1 Actual: 1 Data: (0, 279, 131, 1, 1, 3, 0, 98, 1, 1, 6, 0, 0, 121, 238)

Predicted: 1 Actual: 1 Data: (1, 259, 10, 0, 0, 8, 1, 42, 1, 1, 8, 0, 0, 31, 104)

### Predicted: 0 Actual: 0 Data: (0, 69, 13, 1, 1, 1, 4, 8, 0, 0, 0, 1, 1, 46, 0)

## Stage 2: Algorithm Implementation Stage

After identifying the best-performing algorithm and machine learning model for predicting heart disease in stage 1, the next step is to implement the algorithm as a software tool for desktop usage. This will be accomplished using the Python Tkinter package. Additionally, the tool will be deployed as a web or cloud-enabled platform tool.

The Pycharm project for the implementation is available at this google drive link:

<https://colab.research.google.com/drive/1cyZLmrAa-VYu1MTKzKUh9akuZg8kF8WL#scrollTo=5RFyi3HbAPd->

## Stage 3: Software Deployment Stage

In stage 2, deploying the software as a desktop tool restricts its usefulness and prevents its widespread use by all members of the care team involved in managing the patient's chronic disease. Therefore, it is necessary to deploy the software as a web-based or cloud-based tool. For this purpose, the heart disease prediction tool was deployed as a web-based platform using the Flask API, a popular micro web framework that is used for creating APIs in Python. Flask is a powerful yet simple web framework in Python that can be scaled up to handle complex applications. The Flask project deployment for the heart disease prediction tool can be accessed through the following Google Drive link:

<https://colab.research.google.com/drive/1cyZLmrAa-VYu1MTKzKUh9akuZg8kF8WL#scrollTo=5RFyi3HbAPd->

# Conclusions

This report describes the progress made in the ST1 capstone project focused on creating a software platform that uses data analysis to predict credit card approval. The platform utilizes historical data and insights to predict the probability of a credit card application being approved. The project has identified the best performing algorithm for credit card approval prediction, which can be implemented as a desktop software tool using Python's Tkinter package. However, the deployment of the software as a web or cloud-based platform would enable wider usage among all stakeholders involved in credit card application processing. The project team proposes using Flask API, a popular micro web framework for creating APIs in Python, for web deployment.



## References

1. S. Cortinhas, "Credit Card Approval Clean Data," Kaggle, 2019. [Online]. Available: https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approval-clean-data/code. [Accessed: Apr. 30, 2023].
2. "How to Increase Your Credit Approval Odds," CardRates.com, 2019. [Online]. Available: https://www.cardrates.com/wp-content/uploads/2019/09/Increase-Credit-Approval-Odds-Feat.jpg. [Accessed: Apr. 30, 2023].
3. B. V. Reddy, "Simple Linear Regression with Python Code," DataCamp, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/simple-linear-regression-python. [Accessed: Apr. 30, 2023].
4. A. Vazquez, "Multiple Linear Regression with Python Code," Towards Data Science, 2021. [Online]. Available: https://towardsdatascience.com/multiple-linear-regression-with-python-code-explained-6c0dcedf9dde. [Accessed: Apr. 30, 2023].
5. G. Varoquaux, "Logistic Regression with Python," Scikit-learn, 2021. [Online]. Available: https://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_iris\_logistic.html. [Accessed: Apr. 30, 2023].