F I T 1 0 4 3 Introduction to Data Science

A S S I G N M E N T - 2

33370311 Mohib Ali Khan

Task 1 (Introduction)

a.)

The data I am supposed to audit consists of information related to bank the dataset consists of various columns containing information about the individuals, such as ID, month, age, occupation, annual income, monthly in-hand salary, number of bank accounts, number of credit cards, interest rate, number of loans, payment behavior, and many more. Each column has its own unique characteristics, which provide valuable insights into the data. I will be using this data to train our SVM model and then later we will test our model to produce Credit_Scores.

In [247…

```
#importing neccessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#checking the data
df = pd.read_csv('FIT1043-Credit-Scores-Dataset.csv')
df.head(4)
```

Out[247]:

	ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank
0	106981	8	41	2	14619.585	1005.298750	
1	108774	1	28	12	70883.440	5663.953333	
2	111896	3	29	12	14395.830	1027.652500	
3	32731	2	25	1	11189.065	1159.422083	

b.)

In [248…
```
for column in df.columns:
    column_data_type = df[column].dtype
    unique_counter = df[column].nunique()
    if unique_counter == len(df):
        print(f"Column {column} has data type {column_data_type} and all
    else:
        print(f"Column {column} has data type {column_data_type} and Colu
```

```
Column ID has data type int64 and Column ID has 2096 unique values.
Column Month has data type int64 and Column Month has 8 unique values.
Column Age has data type int64 and Column Age has 43 unique values.
Column Occupation has data type int64 and Column Occupation has 15 unique
values.
Column Annual_Income has data type float64 and Column Annual_Income has 1
954 unique values.
Column Monthly_Inhand_Salary has data type float64 and Column Monthly_Inh
and_Salary has 1959 unique values.
Column Num_Bank_Accounts has data type int64 and Column Num_Bank_Accounts
has 12 unique values.
Column Num_Credit_Card has data type int64 and Column Num_Credit_Card has
12 unique values.
Column Interest_Rate has data type int64 and Column Interest_Rate has 34
unique values.
Column Num_of_Loan has data type int64 and Column Num_of_Loan has 10 uniq
ue values.
Column Delay_from_due_date has data type int64 and Column Delay_from_due_
date has 63 unique values.
Column Num_of_Delayed_Payment has data type int64 and Column Num_of_Delay
ed_Payment has 26 unique values.
Column Changed_Credit_Limit has data type float64 and Column Changed_Cred
it_Limit has 1314 unique values.
Column Num_Credit_Inquiries has data type int64 and Column Num_Credit_Inq
uiries has 18 unique values.
Column Credit_Mix has data type int64 and Column Credit_Mix has 3 unique
values.
Column Outstanding_Debt has data type float64 and Column Outstanding_Debt
has 1948 unique values.
Column Credit_Utilization_Ratio has data type float64 and all values in c
olumn Credit_Utilization_Ratio are unique.
Column Credit_History_Age has data type int64 and Column Credit_History_A
ge has 383 unique values.
Column Payment_of_Min_Amount has data type int64 and Column Payment_of_Mi
n_Amount has 2 unique values.
Column Total_EMI_per_month has data type float64 and Column Total_EMI_per
_month has 1736 unique values.
Column Amount_invested_monthly has data type float64 and Column Amount_in
vested_monthly has 1914 unique values.
Column Payment_Behaviour has data type int64 and Column Payment_Behaviour
has 6 unique values.
Column Monthly_Balance has data type float64 and Column Monthly_Balance h
as 2098 unique values.
Column Credit_Score has data type int64 and Column Credit_Score has 3 uni
que values.
```

In [249…
```
pd.set_option('display.max_columns',100)
df.describe()
```

Out[249]:

	ID	Month	Age	Occupation	Annual_Income	Monthly
count	2100.000000	2100.000000	2100.000000	2100.000000	2100.000000	
mean	83333.452381	4.411429	33.197143	7.927619	51244.805155	
std	41013.880408	2.279068	10.754180	4.325104	38801.235929	
min	10033.000000	1.000000	14.000000	1.000000	7011.685000	
25%	48108.000000	2.000000	24.000000	4.000000	19696.500000	
50%	81490.000000	4.000000	33.000000	8.000000	37087.920000	
75%	118895.250000	6.000000	42.000000	12.000000	73327.380000	
max	155610.000000	8.000000	56.000000	15.000000	179987.280000	

Information about each column

"ID"

It may seem at first that the ID column must be contain all the unique values but upon audit we have found out that ID has 2096 unique values and the highest ID is 155610. The data type used for ID column is integer.

"Month"

The Month column has the entry to every Month number in a year but if we see the table above it clearly provides us with an extra detail that none of the data is recorded from months after the 8th month since the max instance of Month column is "8". The data type for the Month column is integer and it obviously is not unique.

"Age"

The Age column consists of the age of person which has the maximum entry of 56 Year Old and a minimum entry of 14 Year Old. The mean age is 33 Years Old. The data type for the Age column is integer and it is not unique.

"Occupation"

The Occupation column consists of the integer between 1-15 which indicate the job of a person, the mean integer is 7.9 hence the mean job is Developer. The data type of this column is integer and it is not unique.

"Annual_Income"

The Annual_Income column consist of the annual income of a person, it has data type of float and is not unique. The maximum income recorded is 179987.28 and the minimum income recorded is 1.000000.

"Monthly_Inhand_Salary"

The Monthly_Inhand_Salary column consist of the monthly salary of a person and has a data type of float and is not unique. The maximum monthly salry recorded is 15101.940000 and the minimum monthly salary recorded is 319.556250.

"Num_Bank_Accounts"

The Num_Bank_Accounts column consists of 12 Unique entries and it has a data type of integer. The maximum number of accounts held by a person indicated by max is 11.

"Num_Credit_Card"

The Num_Credit_Card columns consists of 12 unique values it has an integer data type and the maximum number of credit cards owned by a person is 11 and the minimum is 0.

"Interest_Rate"

The Interest_Rate column consists of float data type entry with maximum rate as 34 and minmum as 1.

"Num_of_Loan"

The Num_of_Loan column consists of number of loans taken by a person which has an integer data type and has a maximum entry of 9 and mimimum of 0. Which means that the minimum times one has taken loan is 0 and the maximum time loan was taken is 9.

"Delay_from_due_date"

The Delay_from_due_date column consists of number of days the payment was delayed in integer type with maximum amount a payment was delayed was 62 days and minimum 0 days. The mean for the number of days payment was delayed is 21 days.

"Num_of_Delayed_Payment"

The Num_of_Delayed_Payment column consists of the number of times payment was delayed with the maximum count of 25 and minimum of 0, the column has an integer data type.

"Changed_Credit_Limit"

The Changed_Credit_Limit column consist of float data type.

"Num_Credit_Inquiries"

The Num_Credit_Inquiries consists of integer type representing number of time credit report was accessed with the maximum count of 17 and minimum count of 0.

"Credit_Mix"

The Credit_Mix column represents type of credit accounts, with mean of 2 indicating Standard type of account.

"Outstanding_Debt"

The Outstanding_Debt columns represents the amount of outstanding with the maximum recorded data of 4997, the column has float data type.

"Credit_Utilization_Ratio"

The Credit_Utilization_Ratio strangely consist of all the unique values with having float data type.

"Credit_History_Age"

The Credit_History_Age column consists of integer values representing amount of time ones credit account was opened.

"Payment_of_Min_Amount"

The Payment_of_Min_Amount column consists of integer data type with the mean of 0.522381z.

"Total_EMI_per_month"

The Total_EMI_per_month had a maximum value of 1779.103254 with float data type.

"Amount_invested_monthly"

The Amount_invested_monthly column represents the amount of money invested, the column has maximum value of 297.064670 and a mean amount of 56.302384. The column stores all the value in float data type.

"Payment_Behaviour"

The Payment_Behaviour column represents all the values as integer type.

"Monthly_Balance"

The Monthly_Balance column represents available credit as float data type with maximum recorded instance of 1181.113695 and mean of 399.349997.

"Credit_Score"

The Credit_Score column represents the credit scores for each person, it has a mean of 1.865714 which means that the mean lies towards Standard credit score.

Task 2 Supervised Learning

a.)

Supervised machine learning involves training an algorithm to make predictions based on labeled data, which includes input features and output labels. The algorithm learns to make predictions by identifying patterns in the data.

The training dataset is used to teach the algorithm to identify patterns and make accurate predictions on new data. It is the data that is fed into the algorithm during the training process, where the algorithm learns to identify patterns and make predictions based on the input features and output labels.The test dataset is a separate portion of labeled data used to evaluate how well the algorithm generalizes to new, unseen data. By evaluating the model's performance on the test dataset, we can get an estimate of how well the model is likely to perform on new, real-world data.

b.)

In [250…
```
X = df.iloc[:, :-1]  # features DataFrame excludes Credit_Score
y = df.iloc[:, -1]   # label DataFrame only includes row and columns of C
```

c.)

In [251…
```
#importing neccessary library to split data into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size = 0.28, random_state = 33
)
```

Task 3 Classification

a.)

retrieved from https://www.analyticssteps.com/blogs/binary and-multiclass-classification-machine-learning

Binary classification is a task that involves categorizing data into two distinct classes. Essentially, it is a form of prediction that assigns a given thing to one of two groups. Let's say we have a dataset containing information about customers of an online shopping platform. The platform wants to predict which customers are likely to make a purchase and which are not, so they can target their marketing efforts more effectively. The dataset includes features such as age, gender, browsing history, time spent on the platform, and whether or not the customer made a purchase in the past. The goal of binary classification in this case would be to build a model that can predict whether a new customer is likely to make a purchase or not. The two classes would be "likely to make a purchase" and "not likely to make a purchase".

Multi-class classification is a process of classifying a dataset into multiple distinct classes, without any predetermined limit on the number of classes. Let's say we have a dataset containing information about customers who visit a shopping mall. The goal is to classify each customer into one of four categories: "Window Shoppers", "Casual Shoppers", "Regular Shoppers", and "High-Value Shoppers". We can use various features such as age, gender, income, and spending habits to classify the customers into one of these four categories. For example, customers who are young and have a low income and low spending habits could be classified as "Window Shoppers", while customers who are middle-aged, have a moderate income and moderate spending habits could be classified as "Casual Shoppers". Customers who are older, have a higher income, and higher spending habits could be classified as "Regular Shoppers", while customers who are wealthy and spend a lot at the mall could be classified as "High-Value Shoppers". Multi-class and binary classification are two different types of classification problems in machine learning. As mentioned before binary classification involves predicting a response variable that has only two classes, while multi-class classification involves predicting a response variable that has more than two classes.

In addition to the difference in the number of classes being classified, there are also differences in the types of algorithms that are commonly used for binary and multi-class classification. Binary classification often uses algorithms such as logistic regression, k-nearest neighbors, decision trees, support vector machines, and naive Bayes. These algorithms are well suited to the task of determining whether a particular data point belongs to one of two categories. On the other hand, multi-class classification typically involves more complex algorithms, such as neural networks, decision forests, and gradient boosting, which can handle multiple categories simultaneously. These algorithms are designed to handle the additional complexity of multi-class classification, such as handling imbalanced classes, dealing with noisy data, and identifying patterns and relationships between multiple categories. Additionally, some algorithms can be adapted for both binary and multi-class classification, such as random forests, which can be used for both types of classification problems with minimal modifications

b1.)

As per the discussion during lab and my research from chatgpt I have found out that Data normalization and scaling are essential preprocessing steps for Support Vector Machine (SVM) and Support Vector Regression (SVR) models. Normalization refers to transforming the data so that it has a mean of 0 and a standard deviation of 1, while scaling involves adjusting the data to a specific range or scale.

SVM and SVR rely on distance measures between data points. If the features are not normalized or scaled, the calculation of distances can be biased towards features with larger scales, leading to suboptimal results. Moreover, the optimization problem solved by SVM and SVR is sensitive to the scale of features. The influence of features with larger scales can be overwhelming, resulting in suboptimal solutions. Normalization or scaling of data can address these issues and improve the convergence rate of the optimization process in SVM and SVR models, leading to faster training times. Furthermore, it can enhance the accuracy and stability of the models. Therefore, data normalization and scaling are crucial preparatory steps for SVM and SVR models, facilitating the creation of more accurate and reliable models.

b2.)

In [252…
```
# Before scaling we need to Check for missing values in each column
missing_values = df.isnull().any()

if missing_values.any():
    print("Columns with missing values:\n")
    print(missing_values[missing_values == True])
else:
    print("No missing values found.")
```

```
No missing values found.
```

In [253…
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

c1.)

retrieved from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

Support Vector Machine (SVM) is a popular machine learning algorithm used for classification and regression problems. SVM is capable of solving both linear and non-linear problems and is effective in many real-world scenarios. The fundamental principle of SVM is to create a hyperplane or a decision boundary that can separate data into different classes. SVM aims to find the optimal hyperplane that maximizes the margin between the different classes, which makes it a powerful algorithm for dealing with high-dimensional datasets. One of the advantages of SVM is that it can handle both linearly separable and non-linearly separable datasets through the use of kernel functions

c2.)

retrieved from https://techvidvan.com/tutorials/svm-kernel functions/#:~:text=A%20kernel%20is%20a%

In SVM, a kernel is a function used to simplify complex calculations and allow the algorithm to solve problems that may not have a linear solution. Kernels can take data to higher dimensions and perform smooth calculations, allowing the algorithm to form a hyperplane in the higher dimension without increasing complexity. This helps the algorithm to find the best possible separation between the classes in the dataset. Overall, kernels are a powerful tool in SVM that helps to solve many practical problems.

c3.)

In [254…
```
from sklearn import svm

#SVM classifier object with linear kernel
svm_model = svm.SVC(kernel='linear')

# fiting the model on the training data
svm_model.fit(X_train, y_train)
```

Out[254]:
```
SVC(kernel='linear')
```

d1.)

In [255…
```
# make predictions on the test data
y_pred = svm_model.predict(X_test)
```

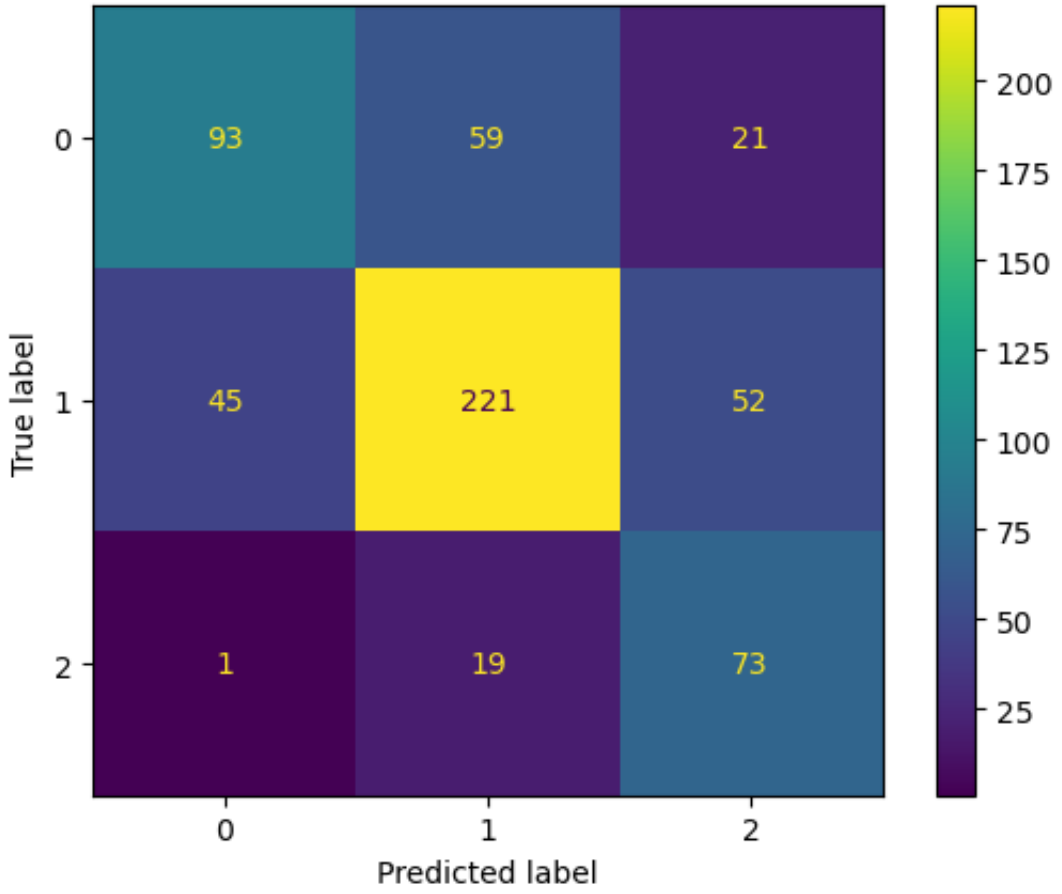
d2.)

In [256…
```
#for our 3x3 matrix
from sklearn.metrics import confusion_matrix
#for a nice display of the matrix
from sklearn.metrics import ConfusionMatrixDisplay


cm = confusion_matrix(y_test, y_pred)

out = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels = [0,1,2]
out.plot()
```

Out[256]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x249d
9bf1dc0>
```



d3.)

retrieved from https://www.kaggle.com/code/prashant111/si explanation-of-quadratic-weighted-kappa/notebook

QWK (Quadratic Weighted Kappa) is a statistical measure used to assess the agreement between two raters who classify items into mutually exclusive categories. It is often used in the evaluation of inter-rater reliability in fields such as medicine, psychology, and education. QWK ranges from -1 to 1, with values closer to 1 indicating better agreement between the raters. A QWK score of 0 indicates agreement no better than chance, while a negative score indicates less agreement than would be expected by chance.

d4.)

In [257…
```
#importing library for qwk
from sklearn.metrics import cohen_kappa_score
qwk = cohen_kappa_score(y_test,y_pred,weights = 'quadratic')
print(qwk)
```

```
0.5243415999801798
```

Task 4 Kaggle Submission

a.)

In [258…
```
#read the csv files into dataframe
kdf = pd.read_csv("FIT1043-Credit-Scores-Submission.csv")
k_dataset = pd.read_csv("33370311-MohibAliKhan-v11.csv")
y  = kdf.iloc[:, :]
print(y.shape)
```

```
(900, 23)
```

b.)

In [263…
```
#scores prediction after passing in features in the model we produced
scaled_y = sc.transform(y)
predicted_y = svm_model.predict(scaled_y)
```

c.)

In [265…
```
#copying our prediction to Credit_Scores column
k_dataset["Credit_Score"] =predicted_y
k_dataset.to_csv("33370311-MohibAliKhan-v11.csv",index=False)
check = pd.read_csv("33370311-MohibAliKhan-v11.csv")
check.shape
```

Out[265]: (900, 2)