CP8210 Assignment 3

Supervised and Unsupervised ML Models

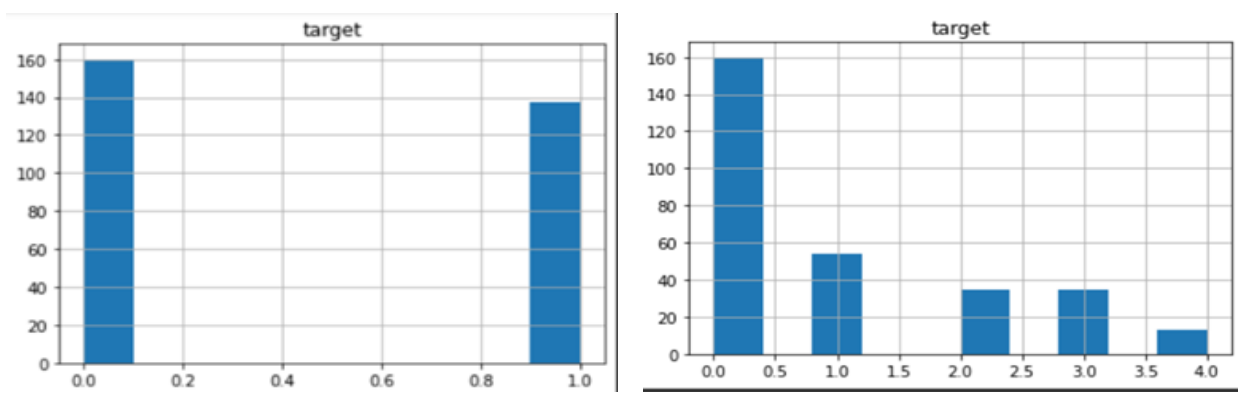
Part A: Supervised ML Model:

In this section the processed Cleveland heart disease dataset from the UCI machine learning repository is studied [1]. There are a total of 13 features and the target variable ranges from 0 to 4. This target variable defines the state of the heart disease with 0 being not-present. In the following sections below the support vector machine and a simple artificial neural network is implemented in python and the source code is shown under the appendix section.

The dataset has some missing values which is why pre-processing the dataset is necessary. The categorical features are also one hot encoded using the get_dummies function. The correlation matrix is also studied and feature selection is performed to get rid of any redundant variables. All the models are pre-processed in the same way. The target variables are converted to binary for all the cases where binary classification is performed. Target equal to 0 means that the heart disease is not present and if it is 1 then it is present.

The data is also split for testing, training, and validation. For all the SVM models, 80% of the data is used for training and 20% for testing. For all the ANN models, the data is first split into 80% for training and 20% for testing. The 80% training data is further split for validation and 25% of 80% is used for validating the ANN model. This means that 60% is used for training, 20% for testing, and another 20% for validation.

Figure A.1: Distribution of Data:



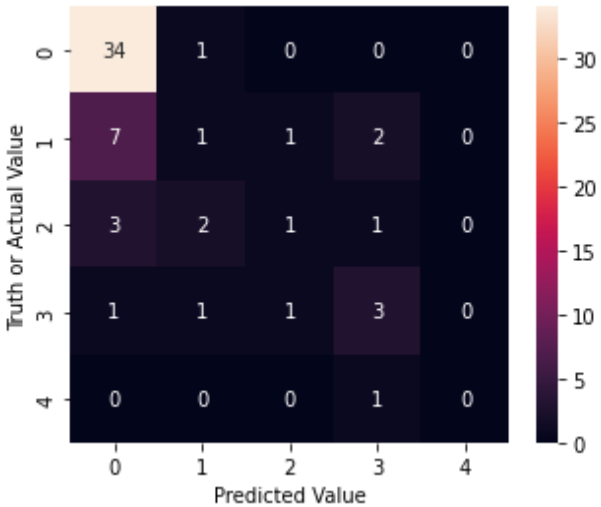
A.1 SVM for Multi-Class Classification

Results:

```
the model performance
-----------------------------
MSE is 0.7833333333333333
R-squared score is 0.65
```

```
model.score(x_test,y_test)
```

```
0.65
```



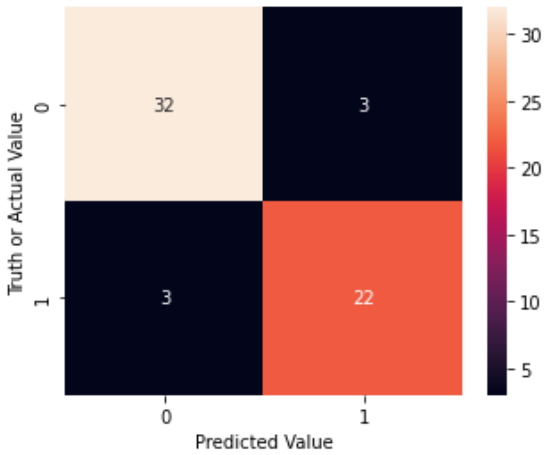
A.2. SVM for Binary Classification

Results:

```
the model performance
-----------------------------
MSE is 0.1
R-squared score is 0.9
```

```
model.score(x_test,y_test)
```

```
0.9
```



A.3. SVM for Numerical Results

```
the model performance
-----------------------------
MSE is 0.5876160023629258
R-squared score is 0.51
```

```
model.score(x_test,y_test)
```

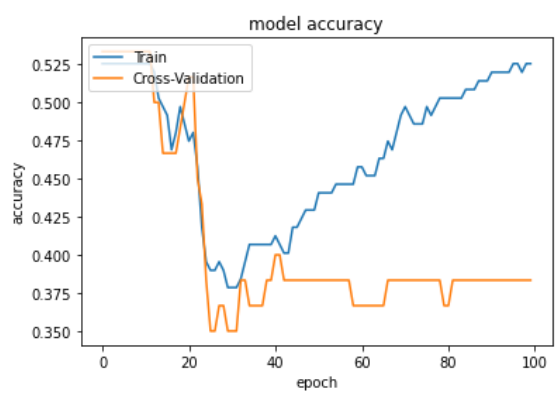
```
0.5115637015685679
```

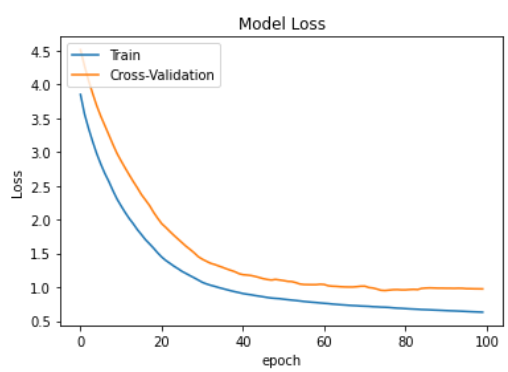
A.4. ANN for numerical results

Model:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 32)                928

 dense_21 (Dense)            (None, 32)                1056

 dense_22 (Dense)            (None, 64)                2112

 dense_23 (Dense)            (None, 1)                 65

=================================================================
Total params: 4,161
Trainable params: 4,161
Non-trainable params: 0
_________________________________________________________________
```

Results:





```
the model performance
-----------------------------
MSE is 0.6521666299876069
MAE is 0.6330441969446838
RMSE is 0.8075683438493654
```

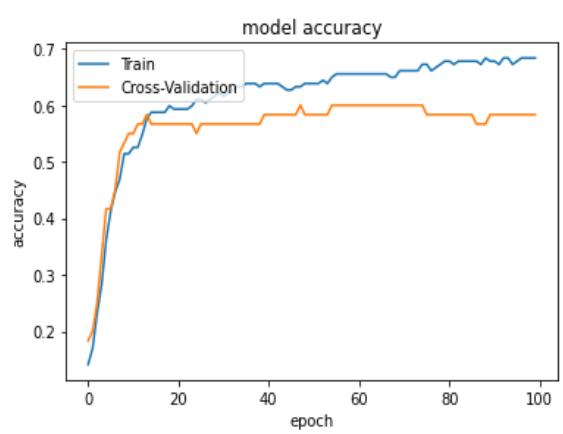
SVM Observations:

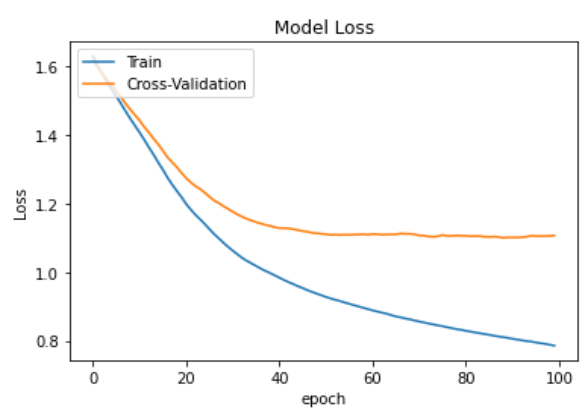
The SVM model is a great ML model for performing regression and classification. It can handle binary classification problem as well as multiclass classification problem and this is the reason why SVM is selected for this assignment. From the results shown above it is clear that the SVM model does a better job at prediction when the problem is changed to binary classification. The model achieves the best accuracy and r squared score of 90%. The confusion matrix also shows that only 3 samples were classified as false positive and 3 as false negative when the model was tested with the 20% data. The other scenarios where the SVM model is used for regression and multi-class classification did not meet a good performance score.

A.5. ANN for Multi-Class Classification

Result:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                928

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 64)                2112

 dense_3 (Dense)             (None, 5)                 325

=================================================================
Total params: 4,421
Trainable params: 4,421
Non-trainable params: 0
```





```
print('Train Split: ')
loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Train Split:
6/6 [==============================] - 0s 3ms/step - loss: 0.7843 - accuracy: 0.6836
Accuracy   :  0.68
```

```
print('Evaluation Split: ')
loss, accuracy =  model.evaluate(x_val, y_val, verbose=2)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

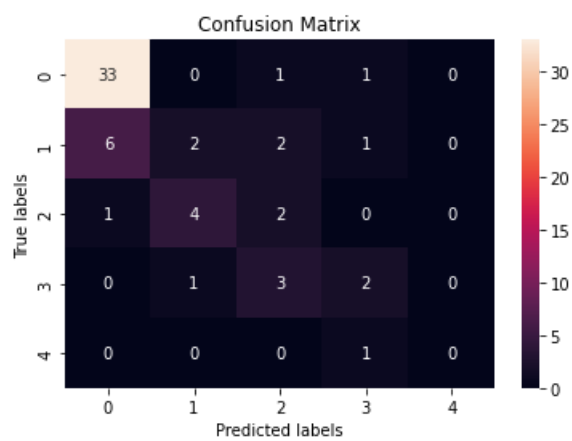
```
Evaluation Split:
2/2 - 0s - loss: 1.1070 - accuracy: 0.5833 - 18ms/epoch - 9ms/step
Accuracy   :  0.58
```

```
print('Test Split: ')
loss, accuracy =  model.evaluate(x_test, y_test, verbose=2)

print("Accuracy   : {:5.2f}".format(accuracy))
```

```
Test Split:
2/2 - 0s - loss: 0.8308 - accuracy: 0.6500 - 18ms/epoch - 9ms/step
Accuracy   :  0.65
```

```
print('the model performance')
print('------------------------------')
print('MSE is {}'.format(mse))
print('MAE is {}'.format(mae))
print('RMSE is {}'.format(rmse))
```

```
the model performance
------------------------------
MSE is 0.08479560911655426
MAE is 0.18152010440826416
RMSE is 0.2911968529224396
```

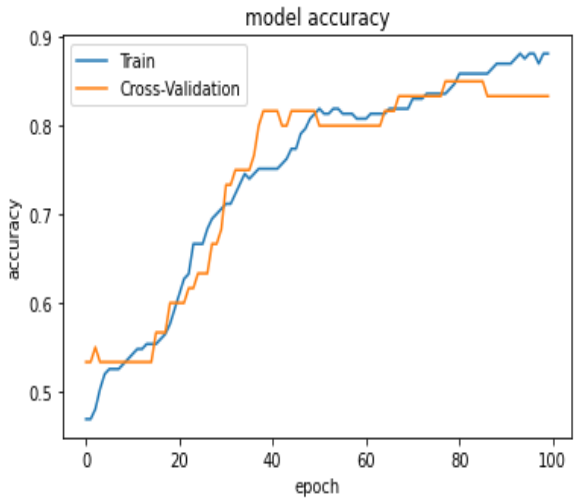


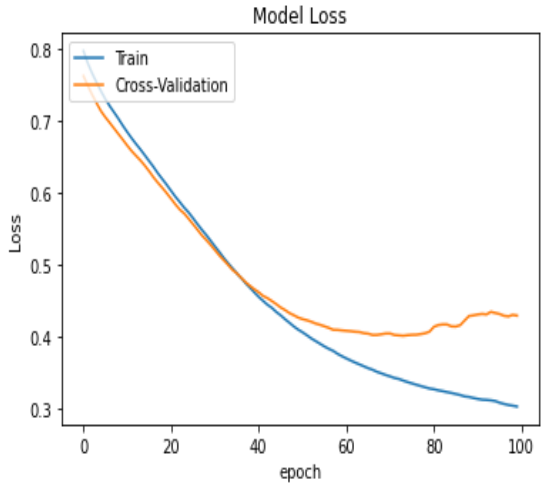
A.6. ANN for Binary Classification

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                928

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 64)                2112

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 4,161
Trainable params: 4,161
Non-trainable params: 0
_________________________________________________________________
```

Summary of the results after each epoch:

	loss	accuracy	val_loss	val_accuracy	epoch
95	0.307875	0.881356	0.431171	0.833333	95
96	0.306182	0.881356	0.428731	0.833333	96
97	0.304753	0.870057	0.427914	0.833333	97
98	0.303721	0.881356	0.429920	0.833333	98
99	0.302599	0.881356	0.428931	0.833333	99


model accuracy
Train
Cross-Validation
accuracy
epoch
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100


Model Loss
Train
Cross-Validation
Loss
epoch
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100

```
print('Train Split: ')
loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Train Split:
6/6 [==============================] - 0s 2ms/step - loss: 0.3016 - accuracy: 0.8757
Accuracy   :  0.88
```

```
print('Evaluation Split: ')
loss, accuracy =  model.evaluate(x_val, y_val, verbose=2)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

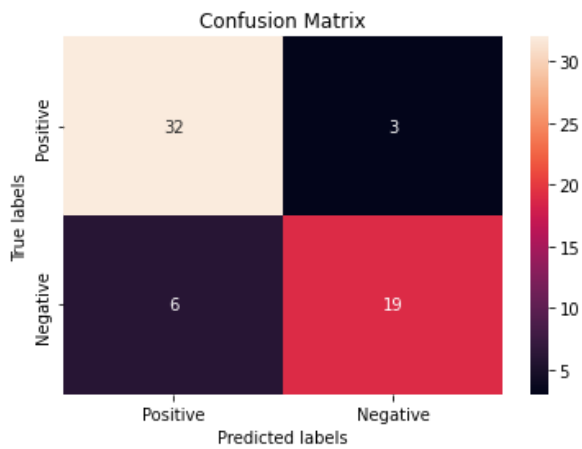
```
Evaluation Split:
2/2 - 0s - loss: 0.4289 - accuracy: 0.8333 - 22ms/epoch - 11ms/step
Accuracy   :  0.83
```

```
print('Test Split: ')
loss, accuracy =  model.evaluate(x_test, y_test, verbose=2)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

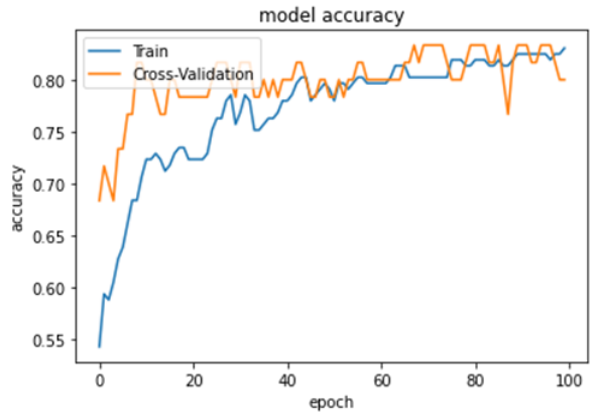
```
Test Split:
2/2 - 0s - loss: 0.3620 - accuracy: 0.8500 - 15ms/epoch - 8ms/step
Accuracy   :  0.85
```

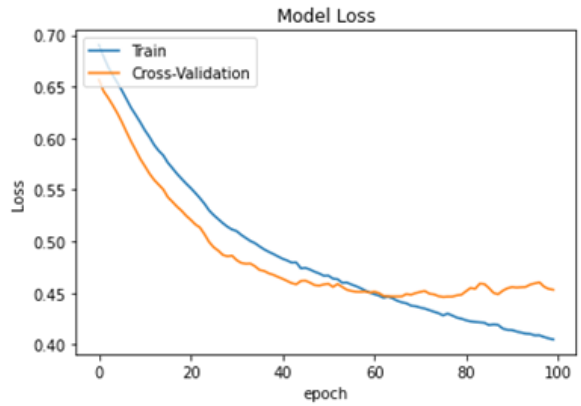
```
print('the model performance')
print('------------------------------')
print('MSE is {}'.format(mse))
print('MAE is {}'.format(mae))
print('RMSE is {}'.format(rmse))
```

```
the model performance
------------------------------
MSE is 4.933941165741477
MAE is 1.9050912565241258
RMSE is 2.2212476597042206
```



A.7. ANN Binary Classification without dummy variables and feature selection:






```
print('Train Split: ')
loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Train Split:
6/6 [==============================] - 0s 2ms/step - loss: 0.4038 - accuracy: 0.8305
Accuracy   :  0.83
```

```
] print('Evaluation Split: ')
loss, accuracy =  model.evaluate(x_val, y_val, verbose=2)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Evaluation Split:
2/2 - 0s - loss: 0.4532 - accuracy: 0.8000 - 22ms/epoch - 11ms/step
Accuracy   :  0.80
```

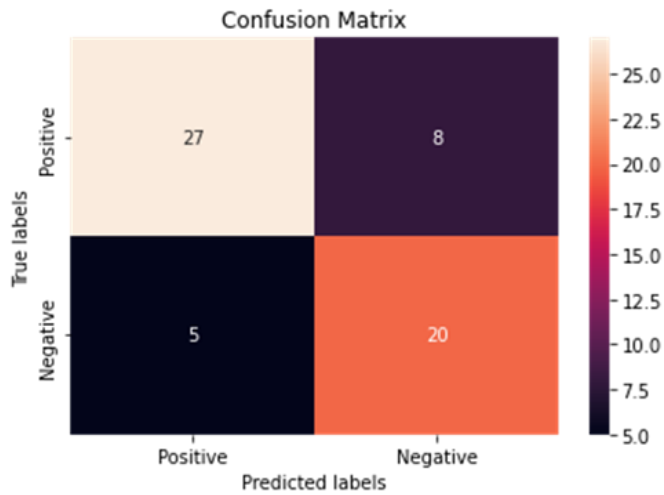
```
] print('Test Split: ')
loss, accuracy =  model.evaluate(x_test, y_test, verbose=2)

print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Test Split:
2/2 - 0s - loss: 0.4669 - accuracy: 0.7833 - 19ms/epoch - 9ms/step
Accuracy   :  0.78
```

```
y_pred=model.predict(x_test)
mse=mean_squared_error(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
```

```
print('the model performance')
print('------------------------------')
print('MSE is {}'.format(mse))
print('MAE is {}'.format(mae))
print('RMSE is {}'.format(rmse))
```

```
the model performance
------------------------------
MSE is 0.15511484027039976
MAE is 0.32056479652722675
RMSE is 0.3938462139851033
```



ANN observation:

A simple Artificial Neural Network model was implemented for the three different scenarios reported above. The model architecture for all cases is the same with 2 hidden layers which is enough to approximate the simple dataset used for this research. The number of neurons in the input layer is 32 as well as in the first hidden layer. The second hidden layer has 64 neurons and the two hidden layers employ the RELU activation function. The learning rate is equal to 0.0001 and the Adam optimizer is used. The loss function employed for the binary classification problem is the binary cross entropy, for regression it is mean squared error, and for multiclass classification problem it is categorical cross entropy. The output layer of the neural network model is also different for all cases. The softmax activation function is used for multiclass problem, and linear is used for the regression problem. The batch size is equal to 16 and the models are trained for 100 epochs. MSE, MAE, and RMSE are observed during training, testing, and validation. To understand the importance of pre-processing the data another ANN model is implemented to solve binary classification problem on dataset with minimal pre-processing and is not encoded with dummy variables and feature selection is not performed. We can observe that the models performance drops to 78% compared to the accuracy achieved when the dataset is preprocessed which is 85% for this case. The best model performance is achieved when pre-processed dataset is used and the problem is binary classification. From the accuracy and loss plots it can be observed that there is some overfitting because the training accuracy achieved by the model is

88% but the test and validation accuracies are comparatively less. This has to do with uneven distribution of data. It is also important to observe the error between the true labels and the predicted labels which is why the MSE, MAE, and RMSE are observed all for the cases. We can see that although the best validation accuracy of 85% was achieved with the binary classification case, but it should also be noted that higher error is also noticed in this case which makes it a less desirable model for this problem. This could be fixed if the neural network model is deeper with more neurons, but it is also important to be cautious when experimenting with the architecture so that overfitting is avoided. Much favorable results are achieved when the model is finetuned and the neurons in the first and second hidden layer are increased to 120. The training accuracy is 94% and test accuracy is 88%. The results for this experiment are shown below.

A.8: Fine-tuned ANN Model

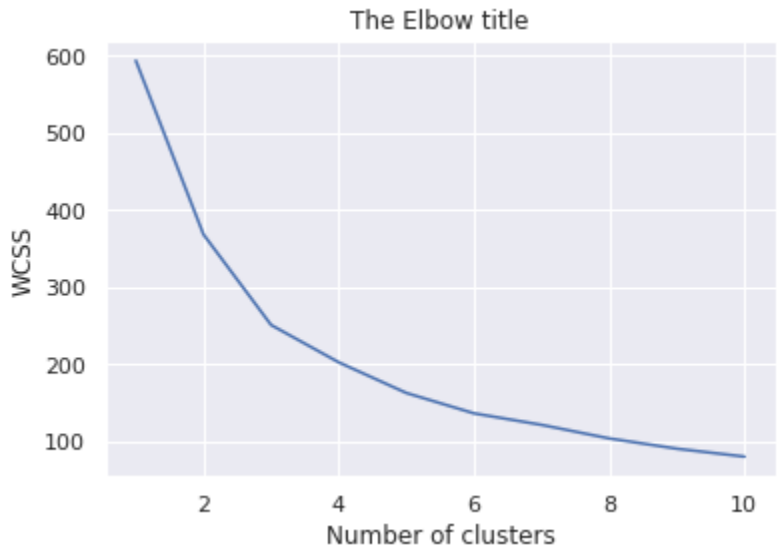
```
144] print('Train Split: ')
     loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

     print("Accuracy   : {:5.2f} ".format(accuracy))

     Train Split:
     6/6 [==============================] - 0s 1ms/step - loss: 0.1977 - accuracy: 0.9379
     Accuracy   :  0.94

145] print('Evaluation Split: ')
     loss, accuracy =  model.evaluate(x_val, y_val, verbose=2)

     print("Accuracy   : {:5.2f} ".format(accuracy))

     Evaluation Split:
     2/2 - 0s - loss: 0.5050 - accuracy: 0.7500 - 18ms/epoch - 9ms/step
     Accuracy   :  0.75

146] print('Test Split: ')
     loss, accuracy =  model.evaluate(x_test, y_test, verbose=2)

     print("Accuracy   : {:5.2f} ".format(accuracy))

     Test Split:
     2/2 - 0s - loss: 0.3410 - accuracy: 0.8833 - 18ms/epoch - 9ms/step
     Accuracy   :  0.88
```

Observations:

The best performance overall is achieved with the SVM model with a test accuracy of 90% and with an MSE of 0.1. The second-best model performance is achieved with the fine-tuned ANN model which is 88% test accuracy. It will require more fine-tuning to outperform the SVM model because neural networks are better function approximators, but it is tricky to find the best model parameters as the choice is arbitrary and is based on trial and error. The model performance can also be increased with a balanced dataset. Having a balanced dataset and having more samples to work with, will improve the training and test performance and will avoid any overfitting.

Part B: Unsupervised ML Model

The results for the elbow method as well as the SSE are shown below for 'age' and 'chol' features. The K-means clustering algorithm is capable of showing the relationship of different features in a dataset and we are able to extract meaningful information.



	Clusters	WSS
0	1	594.000000
1	2	368.707404
2	3	251.306565
3	4	202.905795
4	5	163.148180
5	6	136.737969
6	7	121.801252
7	8	104.213476
8	9	90.915605
9	10	80.648683

Appendix:

Python code for Q2.A:

```
[23] import pandas as pd
     import numpy as np
     from sklearn.metrics import mean_squared_error, r2_score
     from sklearn.model_selection import train_test_split, cross_val_score

     import matplotlib
     from matplotlib import pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import StandardScaler

     #svm
     from sklearn import svm
     from sklearn.svm import SVC
```

```
[24] from google.colab import drive
     drive.mount('/content/drive/')

     Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

```
#load dataset
path='/content/drive/MyDrive/ds_heart/heart_disease.csv'
df=pd.read_csv(path)

df.head()
```

	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1

✓ 0s completed at 6:43 PM

```
[26] #drop samples with missing values
     df.isnull().sum()

     age        0
     sex        0
     cp         0
     testbps    0
     chol       0
     fbs        0
     restecg    0
     thalach    0
     exang      0
     oldpeak    0
     slope      0
     ca         4
     thal       2
     target     0
     dtype: int64
```

```
[27] df.shape

     (303, 14)
```

```
[28] data=df.dropna()
```

```
[29] data.head()
```

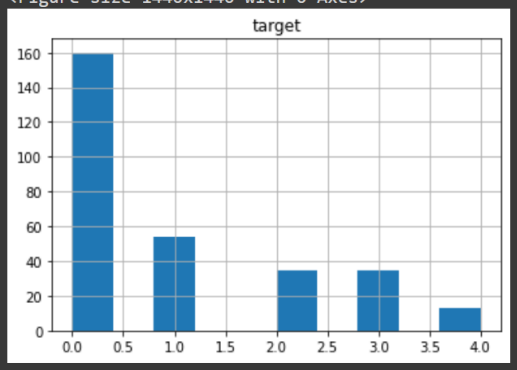
	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2

```
[30] data.shape
```

```
(297, 14)
```

```
[31] plt.figure(figsize=(20,20))
     data.hist('target')
     plt.show('target')
```

```
<Figure size 1440x1440 with 0 Axes>
```




```
data['target'].value_counts()
```

```
0    160
1     54
2     35
3     35
4     13
Name: target, dtype: int64
```

```
[33] data.describe()
```

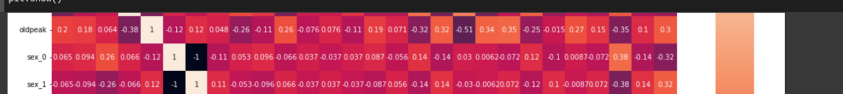
	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.00
mean	54.542088	0.676768	3.158249	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.055556	1.602694	0.676768	4.730640	0.94
std	9.049736	0.468500	0.964859	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.166123	0.618187	0.938965	1.938629	1.23
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000	3.000000	0.00
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.000000	1.000000	0.000000	3.000000	0.00
50%	56.000000	1.000000	3.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000	3.000000	0.00
75%	61.000000	1.000000	4.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000	7.000000	2.00
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000	7.000000	4.00

```
[34] from sklearn import preprocessing

     x=data.drop('target', axis=1)
     y=data['target']
```

```
#get dummy variables
x=pd.get_dummies(x, columns=['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
#scale
standardScaler = StandardScaler()
columns_to_scale = ['age', 'testbps', 'chol', 'thalach', 'oldpeak']
x[columns_to_scale] = standardScaler.fit_transform(x[columns_to_scale])
```

```
[69] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=5)
     x_train.shape,x_test.shape

     ((237, 28), (60, 28))

[70] y_train

     267    2
     97     3
     184    1
     170    3
     154    3
           ..
     8      2
     73     1
     119    2
     191    3
     209    1
     Name: target, Length: 237, dtype: int64
```

```
##plot the correlation matrix##
plt.figure(figsize=(20,20))
cor = x_train.corr()
sns.heatmap(cor, vmin=-1,vmax=1, annot=True)
plt.show()
```




```
[72] #feature selection
     def correlation(dataset, threshold):
         col_corr = set()  # Set of all the names of correlated columns
         corr_matrix = dataset.corr()
         for i in range(len(corr_matrix.columns)):
             for j in range(i):
                 if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                     colname = corr_matrix.columns[i]  # getting the name of column
                     col_corr.add(colname)
         return col_corr

[73] corr_features = correlation(x_train, 0.8)
     len(set(corr_features))

     6
```

```
corr_features
```

```
{'exang_1', 'fbs_1', 'restecg_2', 'sex_1', 'slope_2', 'thal_7.0'}
```

```
x_train.drop(corr_features,axis=1)
x_test.drop(corr_features,axis=1)
```

	age	testbps	chol	thalach	oldpeak	sex_0	cp_1	cp_2	cp_3	cp_4	...	restecg_1	exang_0	slope_1	slope_3	ca_0.0	ca_1.0	ca_2.0	ca_3.0
56	-0.502750	0.468418	-0.276443	0.585108	-0.391318	0	0	0	1	0	...	0	1	0	0	0	1	0	0
118	0.936181	-0.095506	1.592176	-0.768432	0.639470	0	0	0	0	1	...	0	0	1	0	0	0	0	1
133	-0.392063	0.468418	0.262952	1.589347	-0.906712	0	0	0	0	1	...	0	0	1	0	1	0	0	0
24	0.604120	-0.095506	-0.796575	-0.768432	1.154864	0	0	0	0	1		0	0	0	0	0	0	1	0

SVM for multiclass classification:

```
[76] model=svm.SVC()
     model.fit(x_train,y_train)

     SVC()

[77] y_pred=model.predict(x_test)

[78] mse=mean_squared_error(y_test,y_pred)

[79] r=round(model.score(x_test,y_test),2)

[80] print('the model performance')
     print('-----------------------------')
     print('MSE is {}'.format(mse))
     print('R-squared score is {}'.format(r))

     the model performance
     -----------------------------
     MSE is 0.7833333333333333
     R-squared score is 0.65

[81] model.score(x_test,y_test)

     0.65

[82] predicted_output = model.predict(x_test)
     predicted_output
```

0s completed at 7:01 PM

SVM for binary classification:

```
model=svm.SVC(random_state=5, probability=True)
model.fit(x_train,y_train)
```

```
SVC(probability=True, random_state=5)
```

SVM for regression:

```
model=svm.SVR()
model.fit(x_train,y_train)
```

```
SVR()
```

```
[82] predicted_output = model.predict(x_test)
     predicted_output
```

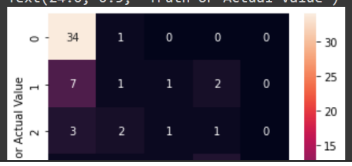
```
array([0, 2, 0, 3, 1, 0, 0, 0, 3, 1, 0, 3, 0, 3, 0, 0, 0, 0, 2, 0, 0, 1,
       0, 0, 2, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 3, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
```

```
[83] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, predicted_output)
     cm
```

```
array([[34,  1,  0,  0,  0],
       [ 7,  1,  1,  2,  0],
       [ 3,  2,  1,  1,  0],
       [ 1,  1,  1,  3,  0],
       [ 0,  0,  0,  1,  0]])
```

```
[84] import seaborn as sn
     from matplotlib import pyplot as plt
     plt.figure(figsize = (5,4))
     sn.heatmap(cm, annot=True)
     plt.xlabel('Predicted Value')
     plt.ylabel('Truth or Actual Value')
```

```
Text(24.0, 0.5, 'Truth or Actual Value')
```



ANN code:

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from sklearn.model_selection import train_test_split

    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Activation, Dense, BatchNormalization, Dropout
    from tensorflow.keras import optimizers
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    import seaborn as sns


    from tensorflow.keras.metrics import categorical_crossentropy

    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.model_selection import train_test_split, cross_val_score

    import matplotlib
    from matplotlib import pyplot as plt
    import seaborn as sns
    from sklearn.preprocessing import StandardScaler

    from sklearn.metrics import mean_absolute_error
    from sklearn.metrics import precision_score
    from sklearn.metrics import recall_score
    from sklearn.metrics import f1_score
```

```
from google.colab import drive

drive.mount('/content/drive/')
```

```
] #load dataset
  path='/content/drive/MyDrive/ds_heart/heart_disease.csv'
  df=pd.read_csv(path)

  df.head()
```

	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

```
#drop samples with missing values
df.isnull().sum()
```

```
age        0
sex        0
cp         0
testbps    0
chol       0
fbs        0
restecg    0
thalach    0
exang      0
oldpeak    0
slope      0
ca         4
thal       2
target     0
```

```
[ ] df.shape
```

```
(303, 14)
```

```
[ ] data=df.dropna()
```

```
[ ] data.head()
```

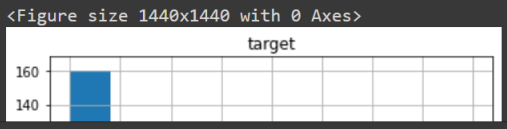
	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

```
[ ] data.shape
```

```
(297, 14)
```

```
plt.figure(figsize=(20,20))
data.hist('target')
plt.show('target')
```

```
<Figure size 1440x1440 with 0 Axes>
```



```
[ ] data['target'].value_counts()
```

```
0    160
1     54
2     35
3     35
4     13
Name: target, dtype: int64
```

```
[ ] data.describe()
```

	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpe
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.0000
mean	54.542088	0.676768	3.158249	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.0555
std	9.049736	0.468500	0.964859	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.1661
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.0000
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.0000
50%	56.000000	1.000000	3.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.8000
75%	61.000000	1.000000	4.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.6000
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.2000

```
from sklearn import preprocessing

x=data.drop('target', axis=1)
y=data['target']
```

```
[ ] y=data['target']
```

```
[ ] #get dummy variables
    x=pd.get_dummies(x, columns=['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
    #scale
    standardScaler = StandardScaler()
    columns_to_scale = ['age', 'testbps', 'chol', 'thalach', 'oldpeak']
    x[columns_to_scale] = standardScaler.fit_transform(x[columns_to_scale])
```

```
[ ] x.head()
```

	age	testbps	chol	thalach	oldpeak	sex_0	sex_1	cp_1	cp_2	cp_3	...	slope_1	slope_2	slope_3	ca_
0	0.936181	0.750380	-0.276443	0.017494	1.068965	0	1	1	0	0	...	0	0	1	
1	1.378929	1.596266	0.744555	-1.816334	0.381773	0	1	0	0	0	...	0	1	0	
2	1.378929	-0.659431	-0.353500	-0.899420	1.326662	0	1	0	0	0	...	0	1	0	
3	-1.941680	-0.095506	0.051047	1.633010	2.099753	0	1	0	0	1	...	0	0	1	
4	-1.498933	-0.095506	-0.835103	0.978071	0.295874	1	0	0	1	0	...	1	0	0	

5 rows × 28 columns

```
[ ]  x_train, x_test, y_train_, y_test_ = train_test_split(x, y, test_size=0.2, random_state=5)

     x_train, x_val, y_train_, y_val_ = train_test_split(x_train, y_train_, test_size=0.25, random_state=5)
```

```
x_train.shape,x_test.shape,x_val.shape,y_train_.shape,y_test_.shape,y_val_.shape
```


```
[ ]  x_train, x_test, y_train_, y_test_ = train_test_split(x, y, test_size=0.2, random_state=5)

     x_train, x_val, y_train_, y_val_ = train_test_split(x_train, y_train_, test_size=0.25, random_state=5)
```


```
[ ] x_train.shape,x_test.shape,x_val.shape,y_train_.shape,y_test_.shape,y_val_.shape
```

```
((177, 28), (60, 28), (60, 28), (177,), (60,), (60,))
```

```
##plot the correlation matrix##
plt.figure(figsize=(20,20))
cor = x_train.corr()
sns.heatmap(cor, vmin=-1,vmax=1, annot=True)
plt.show()
```

```
[ ] #feature selection
    def correlation(dataset, threshold):
        col_corr = set()  # Set of all the names of correlated columns
        corr_matrix = dataset.corr()
        for i in range(len(corr_matrix.columns)):
            for j in range(i):
                if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                    colname = corr_matrix.columns[i]  # getting the name of column
                    col_corr.add(colname)
        return col_corr
```

```
[ ] corr_features = correlation(x_train, 0.8)
    len(set(corr_features))
```

```
6
```

```
[ ] corr_features
```

```
{'exang_1', 'fbs_1', 'restecg_2', 'sex_1', 'slope_2', 'thal_7.0'}
```

```
[ ] x_train.drop(corr_features,axis=1)
    x_test.drop(corr_features,axis=1)
    x_val.drop(corr_features,axis=1)
```

	age	testbps	chol	thalach	oldpeak	sex_0	cp_1	cp_2	cp_3	cp_4	...	restecg_1	exang_0	slope_1
95	-0.281376	-0.208291	0.147367	0.497783	-0.906712	0	0	0	0	1	...	0	0	1
54	0.604120	-0.095506	0.108839	-0.244481	0.295874	0	0	0	0	1	...	0	0	1
102	0.272059	-0.208291	1.072045	0.410458	-0.906712	1	0	0	0	1	...	0	1	1
131	-0.392063	-2.125634	-0.392028	0.192145	-0.906712	0	0	0	1	0	...	0	0	1

```python
# Encode the labeles
y_train = pd.get_dummies(y_train_, prefix='train')
y_val = pd.get_dummies(y_val_, prefix='val')
y_test = pd.get_dummies(y_test_, prefix='test')

y_train
```

	train_0	train_1	train_2	train_3	train_4
12	0	0	1	0	0
140	1	0	0	0	0
175	0	1	0	0	0
301	0	1	0	0	0
41	1	0	0	0	0
...	...	...	...	...	...
217	1	0	0	0	0
234	1	0	0	0	0
196	1	0	0	0	0
259	0	1	0	0	0
137	0	0	0	1	0

177 rows × 5 columns

```python
# We decalred a function for creating a model.
def ANN_model():
    model = Sequential()
```

ANN for binary classification:

```python
def ANN_bmodel():
    model = Sequential()

    model.add(Dense(32, input_shape = (x_train.shape[1],)))

    model.add(Dense(32,Activation('relu')))


    model.add(Dense(64, Activation('relu')))


    model.add(Dense(1))


    learning_rate = 0.0001
    optimizer = optimizers.Adam(learning_rate)
    model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                  optimizer=optimizer,
                  metrics=['accuracy'])
    return model

model = ANN_bmodel()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                928

 dense_1 (Dense)             (None, 32)                1056
```

ANN for multiclass classification:

```python
def ANN_model():
    model = Sequential()
    model.add(Dense(32, input_shape = (x_train.shape[1],)))

    model.add(Dense(32,Activation('relu')))


    model.add(Dense(64, Activation('relu')))


    model.add(Dense(5, activation='softmax'))

    learning_rate = 0.0001
    optimizer = optimizers.Adam(learning_rate)
    model.compile(loss='categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
    return model


EPOCHS = 100
batch_size = 16

model = ANN_model()
model.summary()


history = model.fit(x_train,y_train,batch_size = batch_size,epochs=EPOCHS,verbose=1,shuffle=True,validation_data = (x_val, y_val))
```

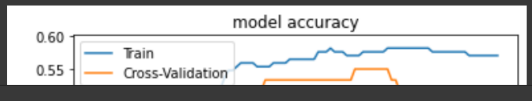
ANN for regression:

```
def ANN_reg():
    model = Sequential()
    model.add(Dense(32, input_shape = (x_train.shape[1],)))
    model.add(Dense(32,Activation('relu')))
    model.add(Dense(64, Activation('relu')))
    model.add(Dense(1, activation='linear'))


    learning_rate = 0.0001
    optimizer = optimizers.Adam(learning_rate)
    model.compile(loss='mse',optimizer=optimizer,metrics=['mae', 'accuracy'])

    return model

model = ANN_reg()

model.summary()


EPOCHS = 100
batch_size = 16


history = model.fit(x_train, y_train,batch_size = batch_size,epochs=EPOCHS, verbose=0,shuffle=True, validation_data = (x_val, y_val))
```

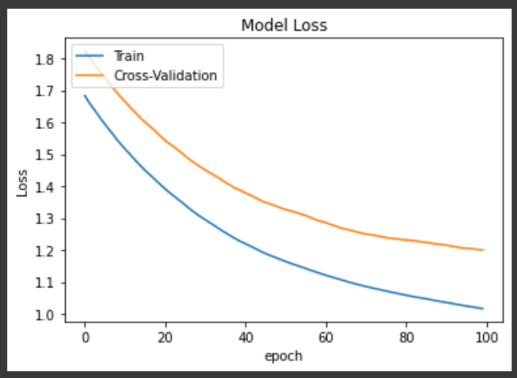
```
[ ] #predict
    example_batch = x_test[:10]
    example_result = model.predict(example_batch)
    print('predicted values: ')
    example_result
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <cl
predicted values:
array([[0.48958203, 0.25132146, 0.12084929, 0.08215811, 0.05608913],
       [0.06026403, 0.22300462, 0.31766328, 0.3490938 , 0.0499742 ],
       [0.8871569 , 0.03679028, 0.01113734, 0.01461681, 0.05029877],
       [0.14853366, 0.23289543, 0.30900323, 0.28606272, 0.02350497],
       [0.18795924, 0.36007705, 0.18142295, 0.19458258, 0.07595821],
       [0.11501727, 0.13406198, 0.56356186, 0.15689372, 0.03046518],
       [0.7962167 , 0.08512557, 0.02748029, 0.01654   , 0.07463735],
       [0.6552876 , 0.18607238, 0.06423805, 0.0293553 , 0.06504665],
       [0.34922978, 0.21123096, 0.2419615 , 0.16860239, 0.02897531],
       [0.11902854, 0.3379742 , 0.29952478, 0.23281026, 0.01066221]],
      dtype=float32)
```

```
[ ] from matplotlib import pyplot as plt
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['Train', 'Cross-Validation'], loc='upper left')
    plt.show()
```



```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('epoch')
    plt.legend(['Train', 'Cross-Validation'], loc='upper left')
    plt.show()
```




```
[ ] print('Train Split: ')
    loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

    print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Train Split:
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
177/177 [==============================] - 0s 53us/sample - loss: 1.0162 - accuracy: 0.5706
Accuracy   :  0.57
```

```
[ ] print('Train Split: ')
    loss, accuracy = model.evaluate(x_train, y_train, verbose=1)

    print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Train Split:
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
177/177 [==============================] - 0s 53us/sample - loss: 1.0162 - accuracy: 0.5706
Accuracy   :  0.57
```

```
[ ] print('Evaluation Split: ')
    loss, accuracy =  model.evaluate(x_val, y_val, verbose=2)

    print("Accuracy   : {:5.2f} ".format(accuracy))
```

```
Evaluation Split:
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
60/60 - 0s - loss: 1.2004 - accuracy: 0.5000
Accuracy   :  0.50
```

```
[ ] print('Test Split: ')
    loss, accuracy =  model.evaluate(x_test, y_test, verbose=2)

    print("Accuracy   : {:5.2f}".format(accuracy))
```

```
Test Split:
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
60/60 - 0s - loss: 1.0182 - accuracy: 0.6000
Accuracy   :  0.60
```

```
Accuracy   :  0.60
```

```
[ ] y_pred=model.predict(x_test)
    mse=mean_squared_error(y_test,y_pred)
    mae=mean_absolute_error(y_test,y_pred)
    rmse=np.sqrt(mean_squared_error(y_test,y_pred))
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
```

```
[ ] print('the model performance')
    print('------------------------------')
    print('MSE is {}'.format(mse))
    print('MAE is {}'.format(mae))
    print('RMSE is {}'.format(rmse))
```

```
the model performance
------------------------------
MSE is 0.10254476964473724
MAE is 0.2164897620677948
RMSE is 0.32022613286972046
```

```
[ ] predicted_output = model.predict(x_test)
    predicted_output
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <clas
array([[0.48958203, 0.25132146, 0.12084929, 0.08215811, 0.05608913],
```

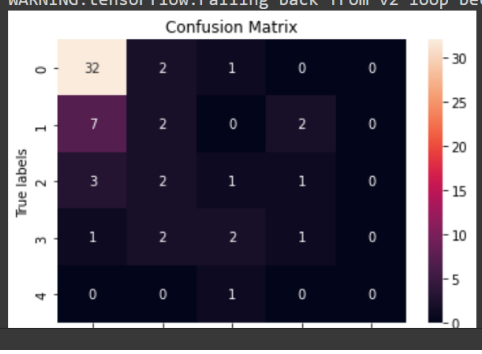
```
the model performance
-----------------------------
MSE is 0.10254476964473724
MAE is 0.2164897620677948
RMSE is 0.32022613286972046
```

```
predicted_output = model.predict(x_test)
predicted_output
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
array([[0.48958203, 0.25132146, 0.12084929, 0.08215811, 0.05608913],
       [0.06026403, 0.22300462, 0.31766328, 0.3490938 , 0.0499742 ],
       [0.8871569 , 0.03679028, 0.01113734, 0.01461681, 0.05029877],
       [0.14853366, 0.23289543, 0.30900323, 0.28606272, 0.02350497],
       [0.18795924, 0.36007705, 0.18142295, 0.19458258, 0.07595821],
       [0.11501727, 0.13406198, 0.56356186, 0.15689372, 0.03046518],
       [0.7962167 , 0.08512557, 0.02748029, 0.01654   , 0.07463735],
       [0.6552876 , 0.18607238, 0.06423805, 0.0293553 , 0.06504665],
       [0.34922978, 0.21123096, 0.2419615 , 0.16860239, 0.02897531],
       [0.11902854, 0.3379742 , 0.29952478, 0.23281026, 0.01066221],
       [0.39798236, 0.35961777, 0.09188342, 0.12127562, 0.02924077],
       [0.22740778, 0.26554662, 0.23325975, 0.24020973, 0.03357608],
       [0.8214126 , 0.10369373, 0.01027092, 0.01209611, 0.05252658],
       [0.08675177, 0.2156468 , 0.31502253, 0.3399631 , 0.04261575],
       [0.7820324 , 0.11390725, 0.03172598, 0.01710672, 0.05522775],
       [0.61469686, 0.27226898, 0.03583001, 0.0399441 , 0.03725995],
       [0.32828292, 0.30882064, 0.16765866, 0.1496702 , 0.04556761],
       [0.6197617 , 0.16954552, 0.05257477, 0.06271136, 0.09540661],
       [0.12376241, 0.2244579 , 0.40682894, 0.22850122, 0.01644946],
       [0.7587569 , 0.10715017, 0.06085149, 0.03651847, 0.03672303],
       [0.8279797 , 0.09322799, 0.01614361, 0.01485455, 0.04779415],
       [0.27122486, 0.34638524, 0.12549092, 0.20640507, 0.05049396],
       [0.554299  , 0.13763183, 0.05485472, 0.09864855, 0.15456596],
       [0.77867657, 0.09885881, 0.02666055, 0.02069625, 0.07510778],
       [0.04670763, 0.40943846, 0.29922354, 0.23420218, 0.01042812],
       [0.89311105, 0.03113876, 0.01528767, 0.0106276 , 0.04983488],
```

```
[ ]
from sklearn.metrics import confusion_matrix


ax= plt.subplot()
predict_results = model.predict(x_test)


predict_results= predict_results.argmax(axis = 1)

cm = confusion_matrix(y_test_, predict_results)

sns.heatmap(cm, annot=True, ax = ax);


ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
```



Part B:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

```
[37] from google.colab import drive
     drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

```
[38] #load dataset
     path='/content/drive/MyDrive/ds_heart/heart_disease.csv'
     dff=pd.read_csv(path)

     dff.head()
```

	age	sex	cp	testbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

```
#drop samples with missing values
dff.isnull().sum()
```

```
age        0
sex        0
cp         0
testbps    0
chol       0
fbs        0
restecg    0
thalach    0
exang      0
oldpeak    0
slope      0
ca         4
thal       2
target     0
dtype: int64
```

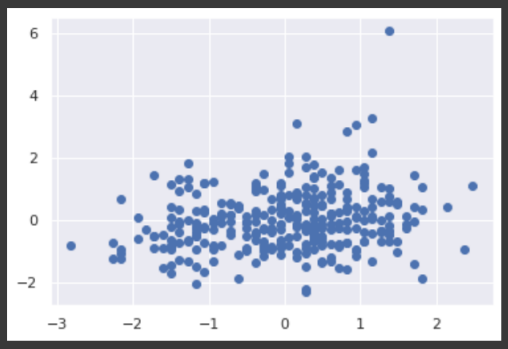
```
[40] data=dff.dropna()
```

```
[49] from sklearn.preprocessing import StandardScaler
     standardScaler = StandardScaler()
     columns_to_scale = ['age', 'testbps', 'chol', 'thalach', 'oldpeak']
     data[columns_to_scale] = standardScaler.fit_transform(data[columns_to_scale])
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3678: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
  self[col] = igetitem(value, i)
```

OneDrive

```
plt.scatter(data['age'],data['chol'])

plt.show()
```




```
x=data.loc[:,['age','chol']]
```

```
kmeans = KMeans(1)
kmeans.fit(x)
```

```
KMeans(n_clusters=1)
```

```
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
53] x=data.loc[:,['age','chol']]
```

```
56] kmeans = KMeans(1)
    kmeans.fit(x)
```

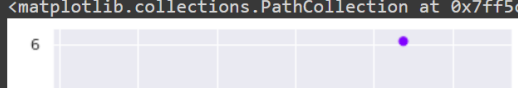
```
KMeans(n_clusters=1)
```

```
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

```
58] data_with_clusters = data.copy()
    data_with_clusters['Clusters'] = identified_clusters
    plt.scatter(data_with_clusters['age'],data_with_clusters['chol'],c=data_with_clusters['Clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7ff5c2eca6d0>
```



```python
mycenters=pd.DataFrame({'Clusters':number_clusters, 'WSS':wcss })
mycenters
```

	Clusters	WSS
0	1	594.000000
1	2	368.707404
2	3	251.306565
3	4	202.905795
4	5	163.148180
5	6	136.737969
6	7	121.801252
7	8	104.213476
8	9	90.915605
9	10	80.648683

```python
[87] import matplotlib.pyplot as plt
     from numpy import unique, where
     model = KMeans(n_clusters=4, verbose=1)

     # fit the model
     model.fit(x)

     # assign a cluster to each example
     yhat = model.predict(x)

     # retrieve unique clusters
```

```python
# define the model
model = KMeans(n_clusters=3, verbose=0)

# fit the model
model.fit(x)

# assign a cluster to each example
yhat = model.predict(x)

# retrieve unique clusters
clusters = unique(yhat)


# create scatter plot for samples from each cluster
for cluster in clusters:
  # get row indexes for samples with this cluster
  row_ix = where(yhat == cluster)
  # create scatter of these samples
  plt.scatter(data_with_clusters['age'],data_with_clusters['chol'],cmap='rainbow')
# show the plot
plt.show()
```

References:

[1] https://archive.ics.uci.edu/ml/datasets/heart+disease