Configuration Manual

MSc Research Project

Data Analytics

Rishab Rao Gauravaram

Student ID: x19202504

School of Computing

National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rishab Rao Gauravaram		
Student ID:	X19202504		
Programme:	MSc Data Analytics	**Year:**	2020-2021
Module:	MSc Data Analytics – Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	16 August 2021		
Word Count:	2193	**Page Count:**	39

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rishab Rao Gauravaram ……………………………………………………………………………………………………

Date: 23/09/2021 ……………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rishab Rao Gauravaram
Student ID: x19202504

1 Introduction

This configuration manual will illustrate the environment setup and the configurations for the same to implement ”Prediction and Classification of Electrocardiogram-Signals using Machine Learning using Apache Spark”.

2 Environment Specification and Configuration

2.1 Hardware Configurations

The hardware configuration of the system used for this project is shown in the Figure

Device specifications

Legion Y540-15IRH-PG0

Device name	LAPTOP-8UKBO5GP
Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	76309F6C-1D3C-4316-B226-5633C4A2E562
Product ID	00327-35886-09734-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

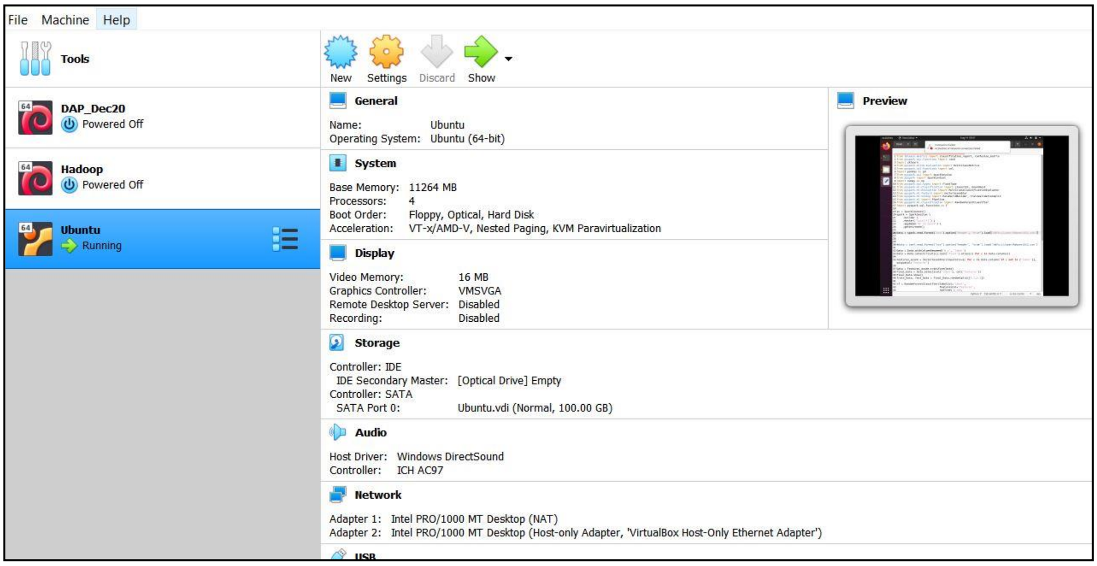
2.2 Software Specifications and Requirements

1. To download the Apache ecosystem on a Virtual Machine, VirtualBox application was installed with Ubuntu Operating System.

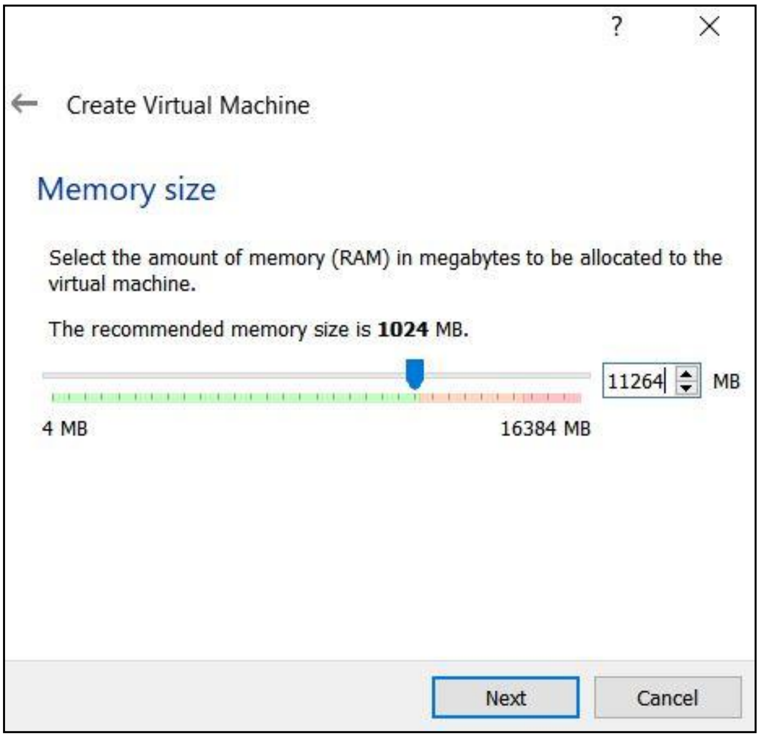
The download link for VirtualBox is as follows –
https://download.virtualbox.org/virtualbox/6.1.26/VirtualBox-6.1.26-145957-Win.exe

The download link for the Ubuntu OS execution file is as follows –
https://ubuntu.com/download/desktop/thank-you?version=20.04.2.0&architecture=amd64

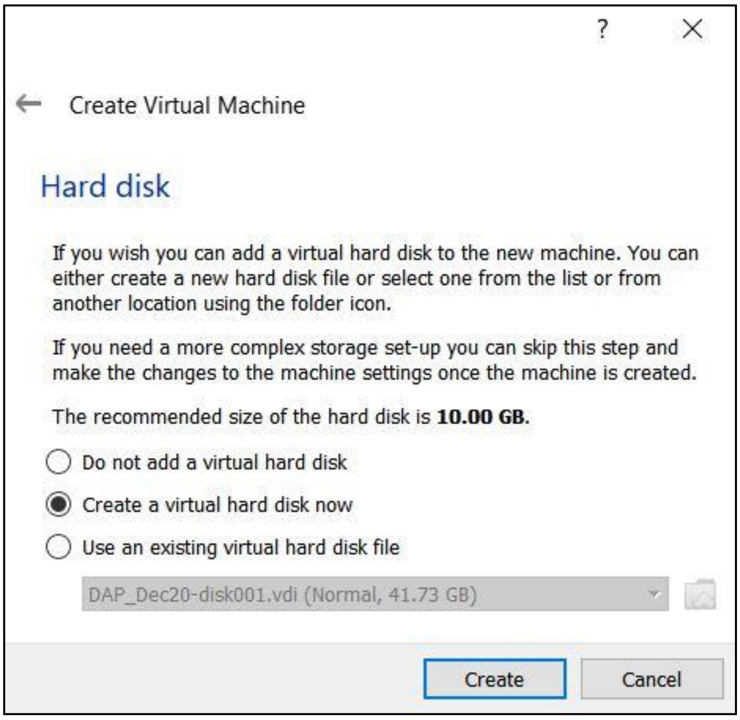
2. After downloading both the files, the following steps will provide a detailed description to setup VirtualBox and Ubuntu on the VirtualBox. After downloading the execution file for VirtualBox, run the execution file and then open it.



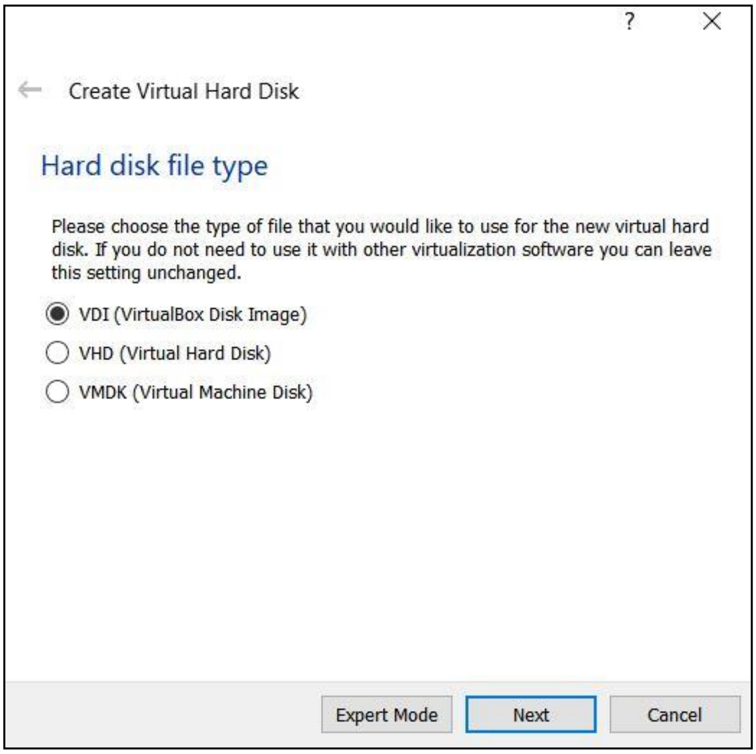
3. VirtualBox Setup –
   Click the "**New**" button. This is used to create a virtual machine as shown. Then press the "**Create**" button. Assign the Memory Size. In this case 11GB Ram was used.



The next step is to create the hard disk. Select the option as shown. Then press "**Next**".

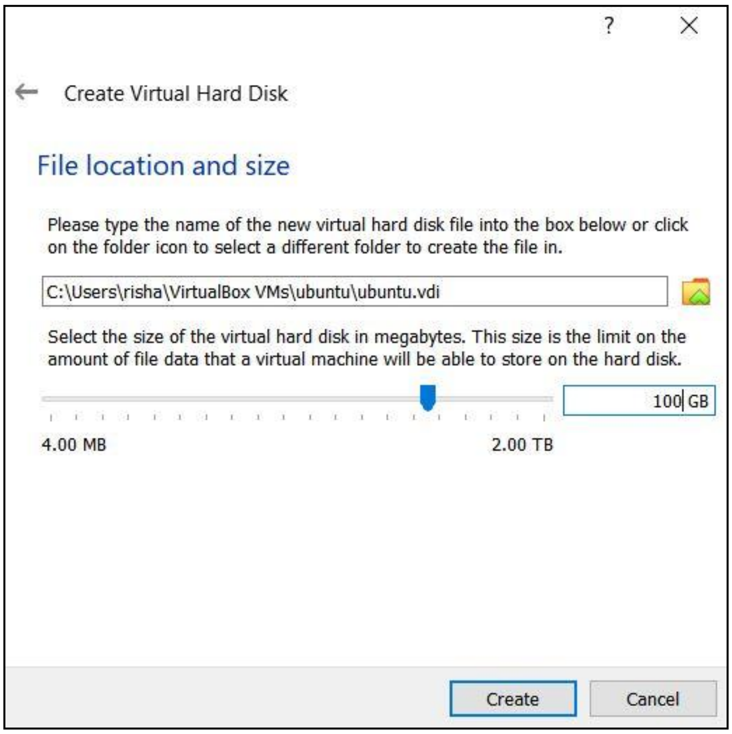


Select VirtualBox Disk Image and press “Next”



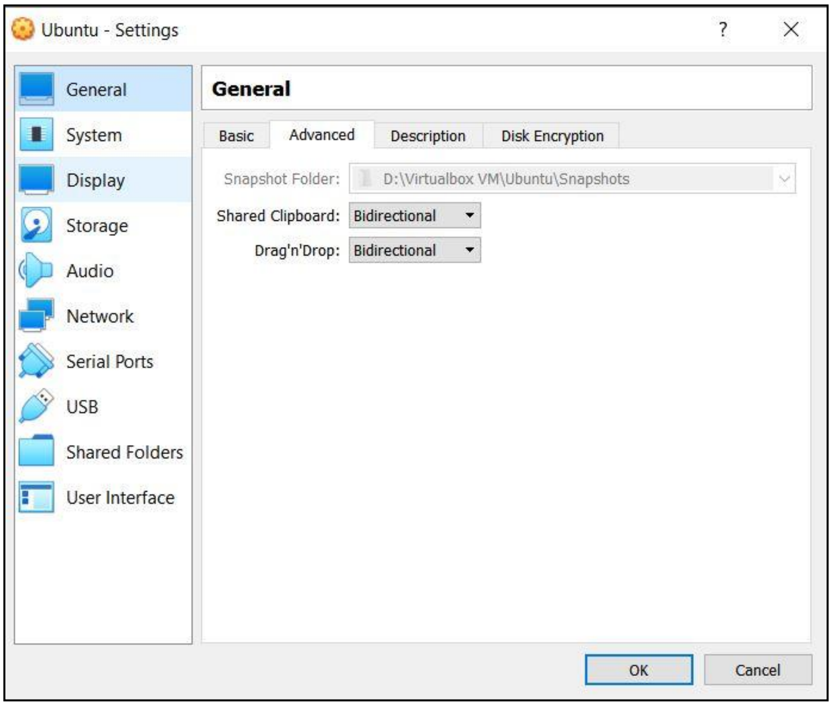
Choose the storage type. If the host device has less storage, then choose "dynamically allocated" option, else, choose "fixed size"

Select the file path for the Virtual Machine to be stored and storage size

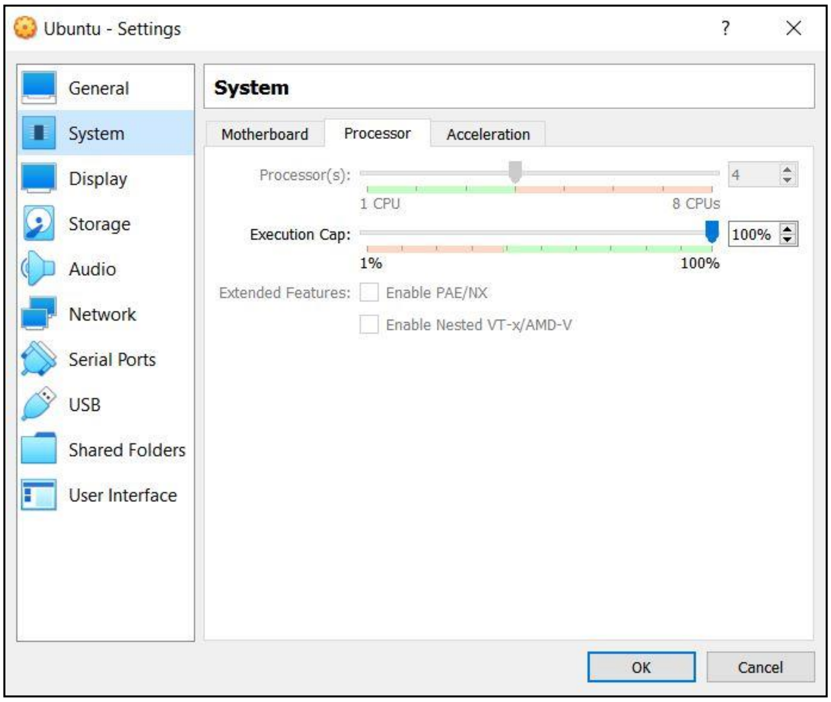


After the virtual machine is created, open “Settings” and select the options as shown in the figure

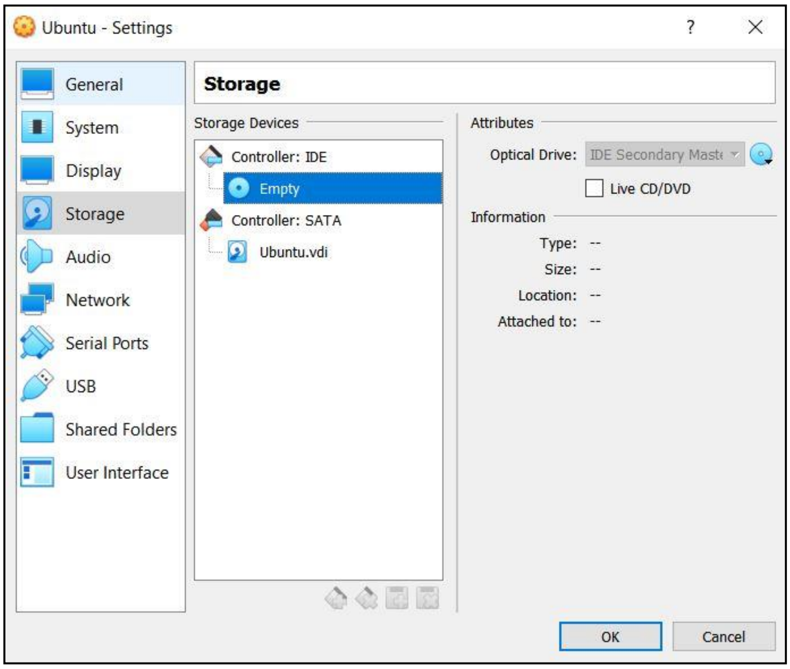
-General



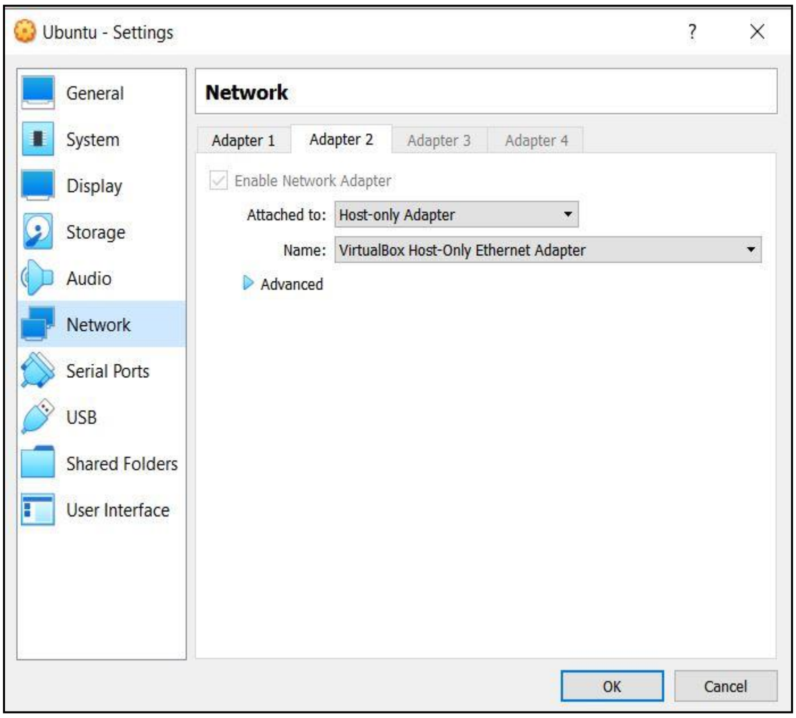
-System Storage



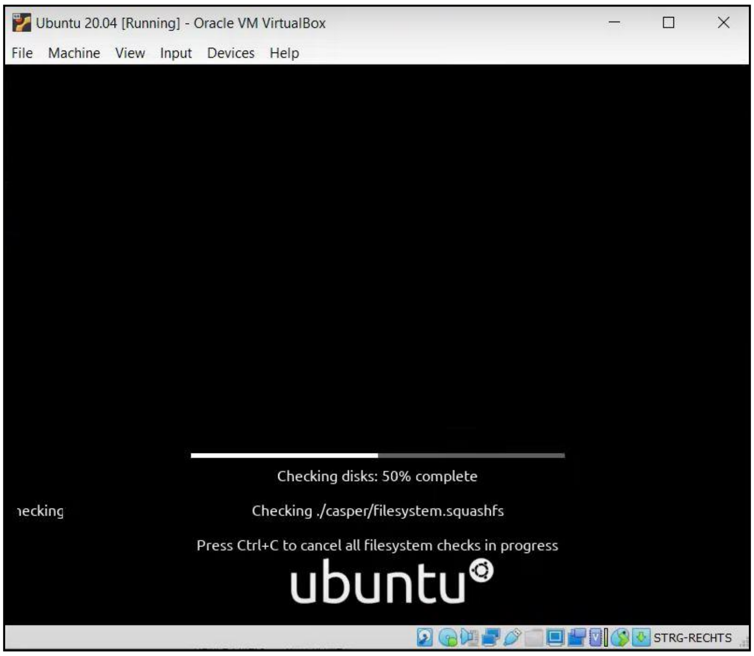
On the top right corner select the "empty disk" option and choose the Ubuntu ISO file previously downloaded.



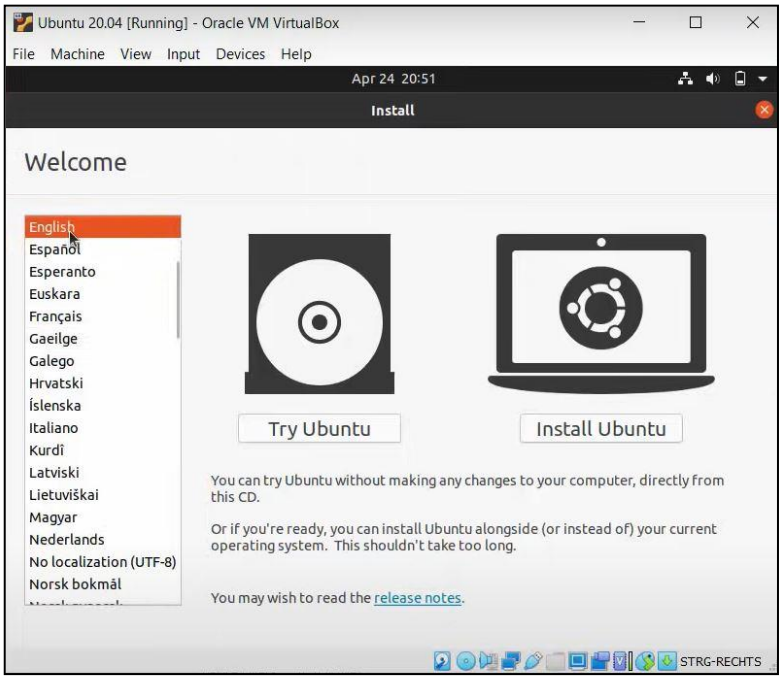
-Network



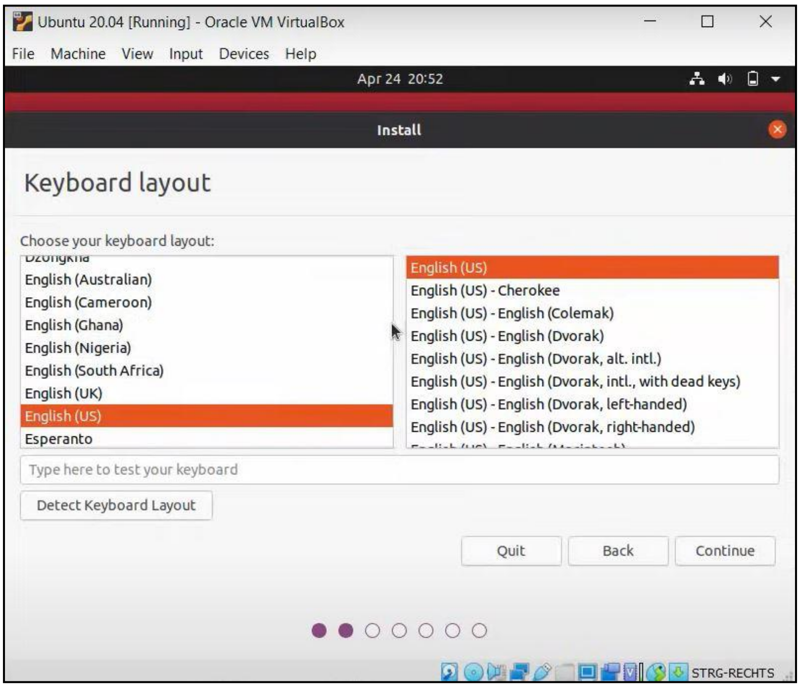
4. Ubuntu Setup –
   After VirtualBox settings configuration, close it and select the “Start” option. This will start the virtual machine.



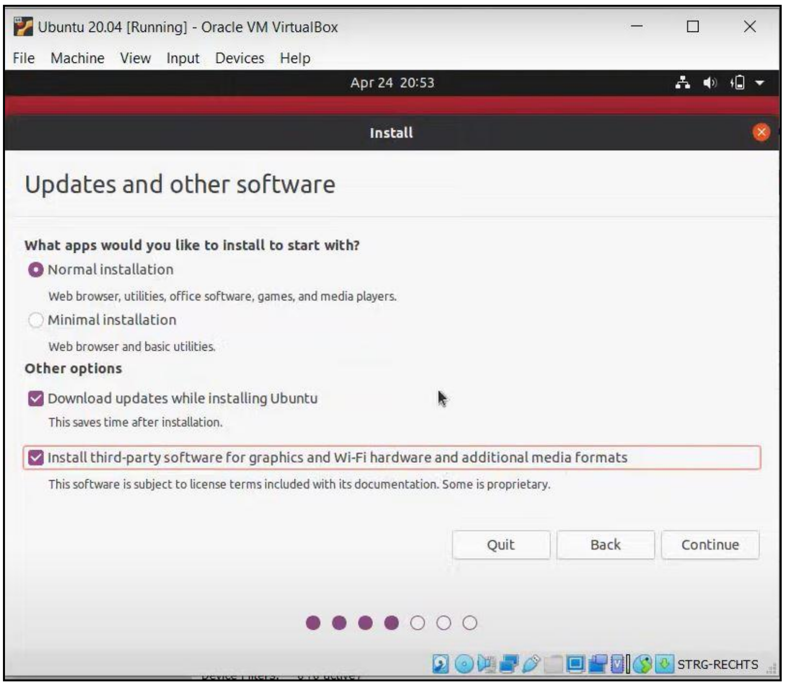
Select the choice of language and click “Install Ubuntu”



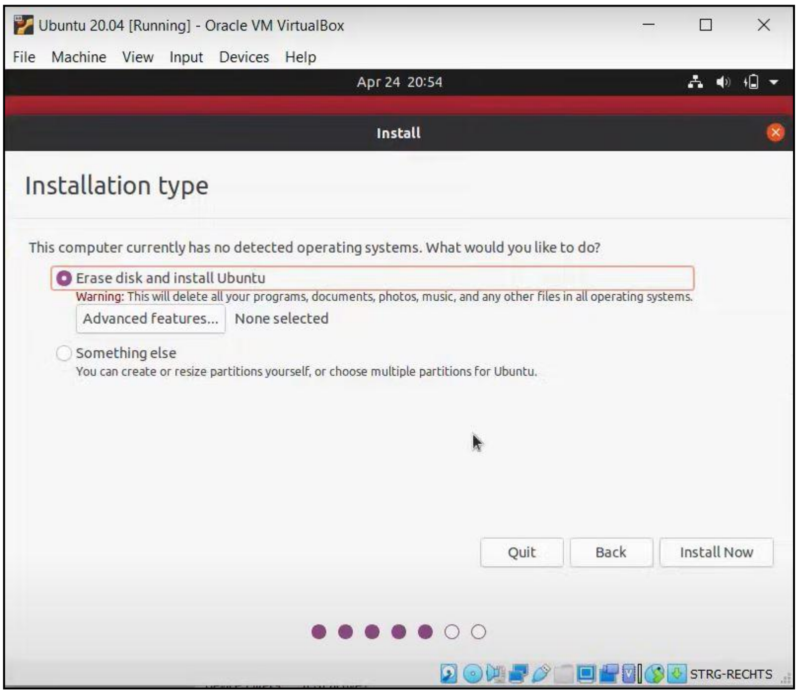
Select the keyboard layout style



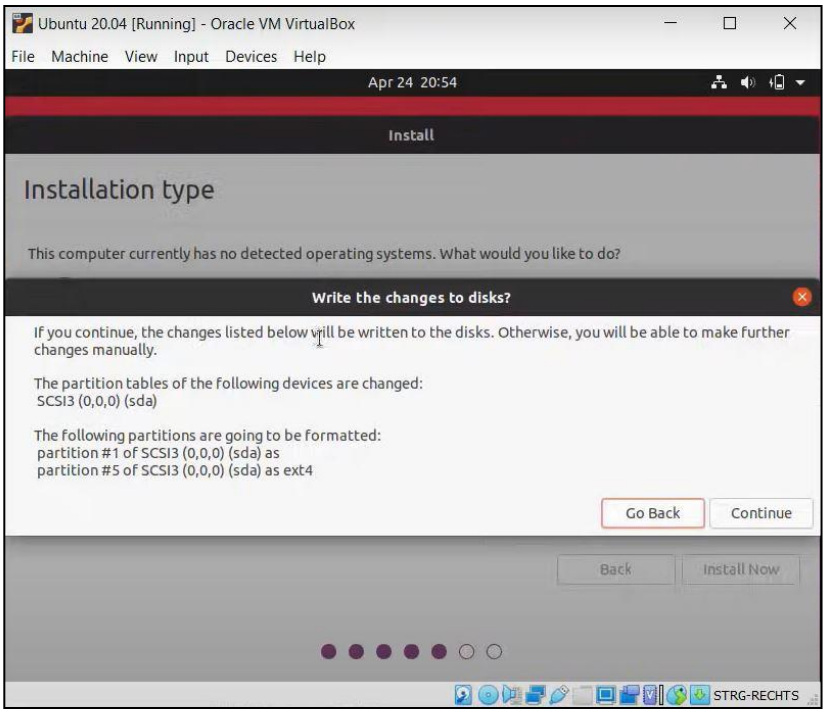
Select all the checkboxes except Minimal installation.



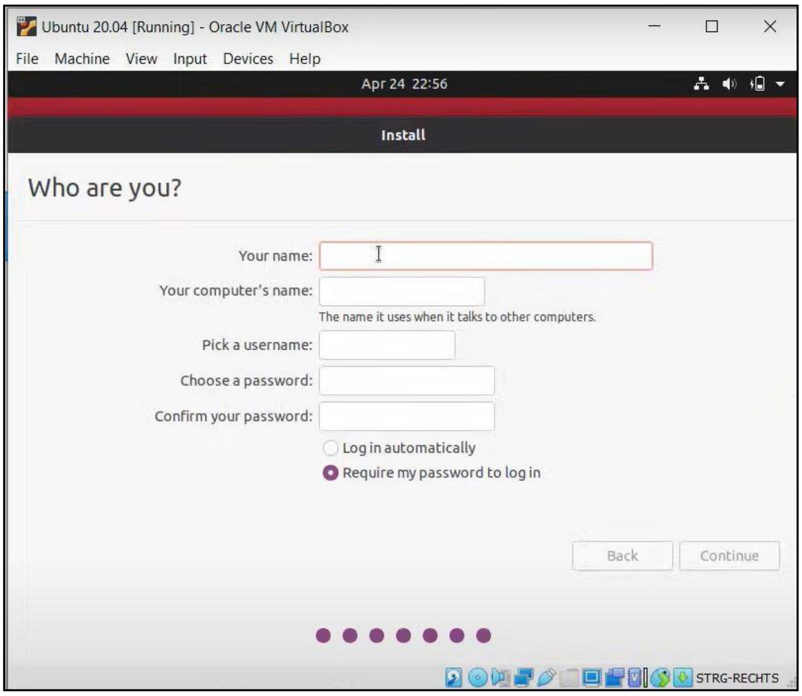
Choose the installation type.



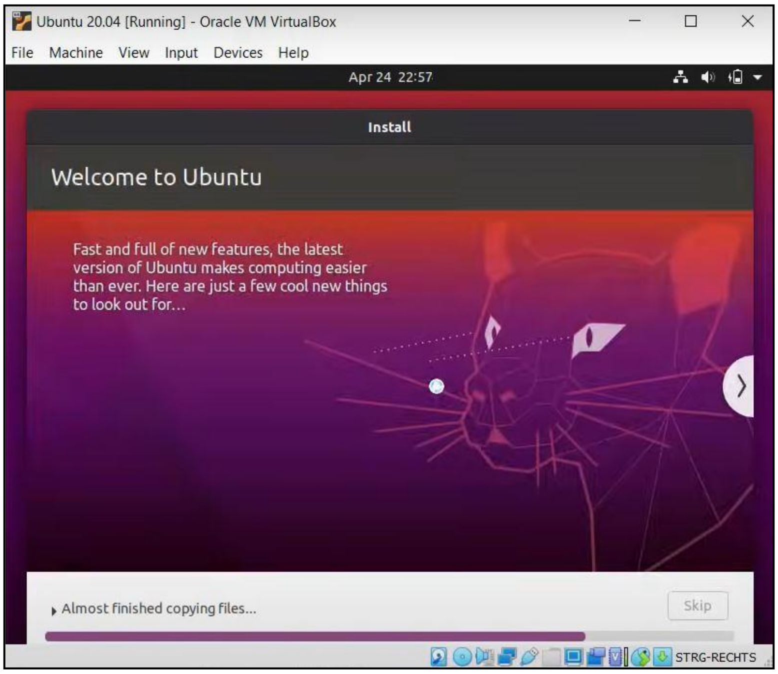
The following screen will request for disk formatting This will **not** make any changes to the host system. The clean-up and formatting are **only** for the virtual machine.



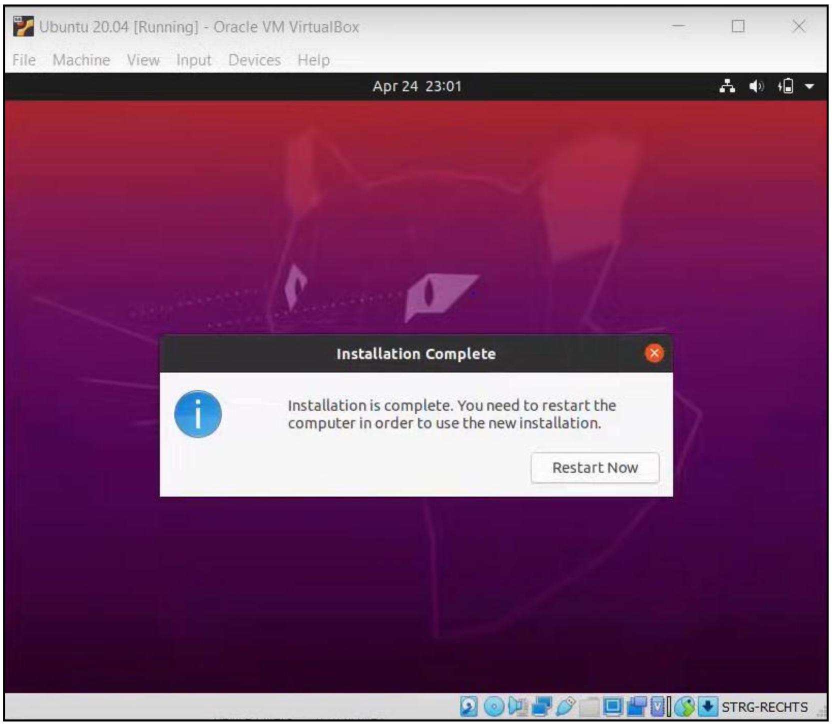
Enter the username and password. **DO NOT FORGET THE LOGIN DETAILS. SAVE IT SOMEWHERE IF NECESSARY**



The installation will commence. It will take some time for the OS to completely installed.



Post installation the virtual machine will request for a reboot. Press “Restart Now”



After the system restarts, enter the login details and open the terminal by pressing “Ctrl + Alt + t” and enter the following command as shown in the figure. This will install all essential Ubuntu packages.

```
rishab@rishab:~$ sudo apt install build-essential dkms linux-headers-$(uname -r)
```

2.3 Hadoop Installation

1. Open the Terminal (Ctrl + Alt + t) and update Ubuntu OS using the commands –
   sudo apt-get update
   sudo apt-get upgrade

2. Install Java-8 jdk using the following command –
   sudo apt install openjdk-8-jdk

3. Check where Java is installed (Java Path). This is required to configure Hadoop.

```
rishab@rishab:~$ sudo update-alternatives --config java
[sudo] password for rishab:
There is only one alternative in link group java (providing /usr/bin/java):
 /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
Nothing to configure.
```

4. Open the **/etc/profile** file using the command –
   sudo nano /etc/profile

   Add the following lines of code to set the path for Java. And then save the file using the command –
   source /etc/profile

```
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

export JAVA_HOME=/usr
```

5. Open the sysctl.conf file using the command –
   sudo nano /etc/profile/sysctl.conf
   Add the following lines of code.

```
###################################################################
# Magic system request Key
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

6. To make these changes to take immediate effect, reboot the system.

7. Configure the SSH for Hadoop. Create a user group for Hadoop using the following commands –

```
rishab@rishab:~$ sudo addgroup hadoopgroup
```

```
rishab@rishab:~$ sudo adduser -ingroup hadoopgroup hduser
```

8. Install SSH

```
rishab@rishab:~$ sudo apt-get install ssh
```

   Enable SSH

```
rishab@rishab:~$ sudo systemctl enable ssh
```

   Start SSH

```
rishab@rishab:~$ sudo systemctl start ssh
```

9. Switch user by typing the following command

```
rishab@rishab:~$ su - hduser
Password:
hduser@rishab:~$
```

10. Generating public/private keys folder

```
hduser@rishab:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
```

11. Copy the keys to the authorized keys folder

```
hduser@rishab:~$ cat /home/hduser/.ssh/id_rsa.pub >> /home/hduser/.ssh/authorized_keys
```

12. Change the permissions for the keys

```
hduser@rishab:~$ cd .ssh/
hduser@rishab:~/.ssh$ chmod 666 ./authorized_keys
```

13. Download Hadoop 3.3.0 using the command

```
hduser@rishab:~$ wget https://archive.apache.org/dist/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

14. Extract the contents of the Hadoop zip file using the command -
    tar -xvf hadoop 3.3.0.tar.gz
15. Move the extracted file into **cd/usr/local.** Change ownership using the following command -
    sudo chown -R hduser:hadoopgroup /usr/local/Hadoop-3.3.0

16. Create a symbolic link with hadoop using the command

```
hduser@rishab:~$ sudo ln -sf /usr/local/hadoop-3.3.0/ /usr/local/hadoop/
```

17. Open the **bashrc file** using the command sudo nano ./.bashrc and configure it using the following lines of code. Then save the file using **source**

```
#Hadoop Config
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop

export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_PREFIX/lib/native"
export JAVA_HOME="/usr"
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin
```

18. Open the Hadoop environment to set the java path

```
 GNU nano 4.8     /usr/local/hadoop/etc/hadoop/hadoop-env.sh
# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME="/usr"
```

19. Navigate to the hadoop directory using **cd /usr/local/hadoop/etc/hadoop** and navigate the .xml files and add the following –

sudo nano core-site.xml

```
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

sudo nano mapred-site.xml

```
<configuration>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

sudo nano yarn-site.xml

```
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
```

sudo nano hdfs-site.xml

```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.name.dir</name>
  <value>file:/usr/local/hadoop/hadoopdata/hdfs/namenode</value>
</property>
</configuration>
```

20. Navigate to the home directory using the command **cd**

21. Format the namenode

```
hduser@rishab:~$ sudo hdfs namenode -format
```

22. Start Hadoop services by first navigating to the Hadoop directory
cd /usr/local/hadoop

```
hduser@rishab:/usr/local/hadoop$ start-dfs.sh
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Starting namenodes on [localhost]
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Starting datanodes
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Starting secondary namenodes [rishab]
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
hduser@rishab:/usr/local/hadoop$ start-yarn.sh
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Starting resourcemanager
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Starting nodemanagers
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
hduser@rishab:/usr/local/hadoop$ jps
11136 DataNode
11345 SecondaryNameNode
11683 NodeManager
12024 Jps
11533 ResourceManager
10959 NameNode
```

2.4 Apache Spark Installation

1. Download and extract Apache Spark 3.1.2 in **cd /usr/local** using the link https://www.apache.org/dyn/closer.lua/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz

2. Create a symbolic link of spark using the command
   sudo ln –s spark-3.1.2-bin-hadoop3.2 spark

3. Change the ownership of the Spark directory to Hadoop group and ownership to hduser as seen in section 2.3 (**sudo chown -R hduser:hadoopgroup spark/**)

4. Configure the .bashrc file using the following lines of code and save it using **source**

```
#Spark Config
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_PYTHON=python3
```

5. Navigate to **cd/usr/local/spark** and start the services using **sbin/start-master.sh**

2.5 Anaconda Installation

1. Download Anaconda using the following link:
   https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh

2. Once the download is complete, extract it using the command:

```
hduser@rishab:~$ bash ~/Downloads/Anaconda3-2021.05-Linux-x86_64.sh
```

   Once the installation commences, read the terms and conditions, and answer **"yes"** to everything. Keep in mind to answer "yes" when the installer requests permission to prepend anaconda3 path to the .bashrc file.

3. Then source the .bashrc file for the changes to take immediate effect
   The terminal should look like the following:

```
(base) hduser@rishab:~$
```

4. To implement Deep Neural Networks, create a separate environment on Anaconda and install TensorFlow and Keras.

5. First step is to install conda forge. This is used to download and install TensorFlow. Use the following commands on the terminal after installing Anaconda:
   conda config –add channels conda-forge
   conda config –set channel_priority strict

6. Before installing Keras and TensorFlow a new environment must be created. Open anaconda navigator

```
(base) hduser@rishab:~$ anaconda-navigator
```

   Go to environments and create a new environment named "TensorFlow".

7. After the new environment is created, select the TensorFlow environment and install TensorFlow Package

keras	Deep learning library for theano and tensorflow	2.2.5
opt_einsum	Optimizing einsum functions in numpy, tensorflow, dask, and more with contraction order optimization.	3.3.0
tensorboard	Tensorflow's visualization toolkit	2.4.0
tensorboard-plugin-wit		1.6.0
tensorflow	Tensorflow is a machine learning library.	2.4.1
tensorflow-base	Tensorflow is a machine learning library, base package contains only tensorflow.	2.4.1
tensorflow-estimator	Tensorflow estimator is a high-level tensorflow api that greatly simplifies machine learning programming.	2.5.0

3 Implementation of the research project

3.1 Download the dataset

a) Download the wfdb package using the following line on the terminal:
pip install wfdb

b) Open a new python file using the terminal as shown

```
(base) hduser@rishab:~/Final$ nano Data_Download.py
```

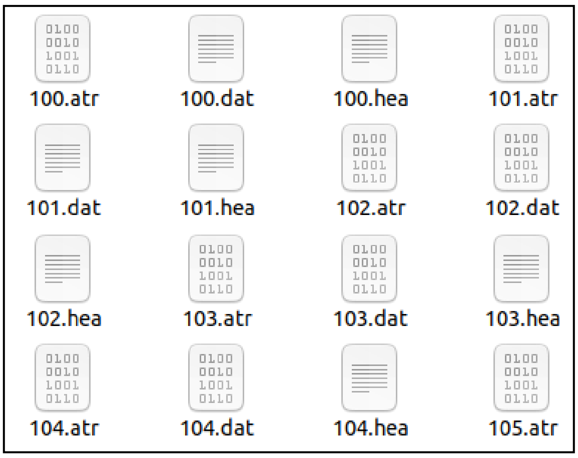
c) Type the following lines of code to download the data

```
hduser@rishab: ~/Final
  GNU nano 4.8                    Data_Download.py

import wfdb
# It will take some time to download the dataset
# The dataset will be downloaded inside the directory 'data'
def download():
    wfdb.dl_database('mitdb', dl_dir = 'data')

if __name__ == "__main__":
    download()
```

d) Save the file and exit the text editor. Run the python script using the following line:

```
(base) hduser@rishab:~/Final$ python Data_Download.py
```

e) The data will be downloaded under the folder name “Data”. The contents of the folder will look like the following:



3.2 Data Preprocessing, Transformation and Exploratory Data Analysis

a) Open a new python file using a text editor (nano, vim, vi, gedit, etc) in the terminal for Data Processing, Data Transformation, Signal extraction and EDA.

nano preprocess.py

b) Import the following packages

```
import numpy as np
import sys, os
import wfdb
import matplotlib.pyplot as plt
import pywt
import pickle as pk
from collections import Counter
import time
import pandas as pd
start_time = time.time()
```

c) The names of the data files, five types of classes, different types of beats and different types of beats assigned to each class are stored in different lists and dictionaries.

```
#The data names, beats and sub beats are annotated as prescribed by the authors of the database
#The link for the annotations - https://archive.physionet.org/physiobank/annotations.shtml

data_names = ['100', '101', '102', '103', '104', '105', '106', '107',
              '108', '109', '111', '112', '113', '114', '115', '116',
              '117', '118', '119', '121', '122', '123', '124', '200',
              '201', '202', '203', '205', '207', '208', '209', '210',
              '212', '213', '214', '215', '217', '219', '220', '221',
              '222', '223', '228', '230', '231', '232', '233', '234']


types = ['N', 'S', 'V', 'F', 'Q']
beats = ['N', 'L', 'R', 'e', 'j', 'A', 'a', 'J', 'S', 'V', 'E', 'F', '/', 'f', 'Q']
sub_beats = {'N':'N', 'L':'N', 'R':'N', 'e':'N', 'j':'N',
            'A':'S', 'a':'S', 'J':'S', 'S':'S',
            'V':'V', 'E':'V',
            'F':'F',
            '/':'Q', 'f':'Q', 'Q':'Q'}

#Empty lists to store features and labels
X = []
Y = []
num = 100
```

d) Data Preprocessing using Denoising[1] (DWT for decomposition and IDWT for reconstruction of the wavelets), normalising the wavelets and extracting the waveform for five different classes of heartbeats.

```python
#Denoising and Normalising to clean the data and extract
#N, S, V, F, Q classes of signals
def denoise(sig, sigma, wn='bior1.3'):
    #sig -> the input signal, sigma -> value for wavelet transform, wn -> wavelet transform family (biorthogonal in this case)
    threshold = sigma * np.sqrt(2*np.log2(len(sig)))
    c = pywt.wavedec(sig, wn) #signal decomposition (DWT)
    thresh = lambda x: pywt.threshold(x,threshold,'soft') #Soften the signals
    nc = list(map(thresh, c))
    return pywt.waverec(nc, wn) #signal reconstruction (IDWT)

#Parsing the previously downloaded data.
#There are multiple file for each record.
#Select the files with the .atr extension.

for d in data_names:
    #Parsing and extracting the data as prescribed in the official websie
    r=wfdb.rdrecord('./data/'+d)
    ann=wfdb.rdann('./data/'+d, 'atr', return_label_elements=['label_store', 'symbol'])
    #Signals in file 114 are stored differently
    if d!='114':
        sig = np.array(r.p_signal[:,0])
    else:
        sig = np.array(r.p_signal[:,1])
    sig = denoise(sig, 0.005, 'bior1.3') #Calling the denoise function
    sig = (sig-min(sig)) / (max(sig)-min(sig)) #Normalising the signals between the lower and upper bound.


    # Extracting the features (signals) and labels (class for each signal)
    sig_len = len(sig)
    sym = ann.symbol
    pos = ann.sample
    beat_len = len(sym)
    for i in range(beat_len):
         #Since the file names start from 100
        if sym[i] in beats and pos[i]-num>=0 and pos[i]+num+1<=sig_len:
            a = sig[pos[i]-num:pos[i]+num+1]
            if len(a) != 2*num+1:
                print("Length error")
                continue
            X.append(a)
            Y.append(types.index(sub_beats[sym[i]]))

X = np.array(X) # Features data, containing the signals
Y = np.array(Y) # Label data, containing the classes of the corresponding signals
```

e) Exploratory Data Analysis on the Data

```python
#EDA and Visualising the extracted signals
#Find the number of records in each class of the heartbeats
values, counts = np.unique(Y, return_counts=True)
print(values, counts)
plt.figure(figsize=(20,10))
my_circle=plt.Circle( (0,0), 0.7, color = 'white')
plt.pie(counts, labels=['N', 'S', 'V', 'F', 'Q'], colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0'],autopct='%2.2f%%', textprops={'fontsize': 18})
plt.title('Different classes of heartbeats', fontsize = 25)
p=plt.gcf()
#p.gca().add_artist(my_circle)

plt.show()
```

```python
#Plotting the graphs for each class of heartbeat
values, indexes = np.unique(Y, return_index=True)
for i in indexes:
    plt.plot(X[i])
    plt.xlabel("beat sample")
    plt.ylabel("time")
    if Y[i]==0:
        plt.title("Class N")
    elif Y[i]==1:
        plt.title("Class S")
    elif Y[i]==2:
        plt.title("Class V")
    elif Y[i]==3:
        plt.title("Class F")
    else:
        plt.title("Class Q")
    plt.show()
```

[1] https://stackoverflow.com/questions/32184232/filter-ecg-signal-with-wavelet

f) Storing the transformed and cleaned data into a .csv that will be stored on the HDFS.

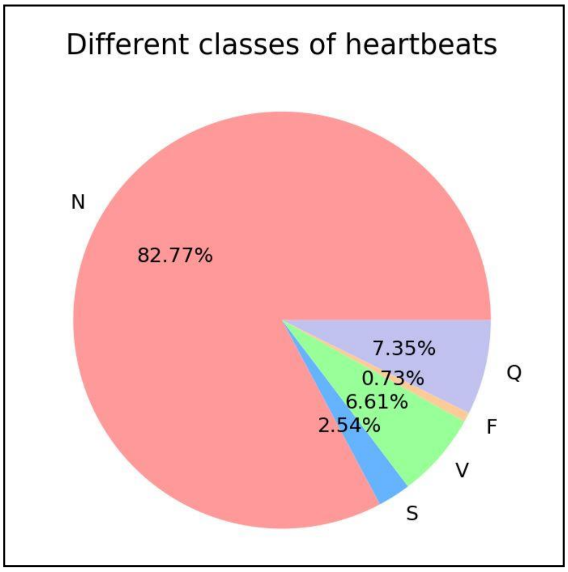
```
#Data to run ML algorithms on Spark
Features = pd.DataFrame(X)
#print(Features)

Labels = pd.DataFrame(Y)
#print(Labels)

ECG = pd.merge(Features, Labels, right_index = True, left_index = True)
#print(ECG)
ECD_Data = ECG.to_csv(r'ECG.csv', index=False)

```

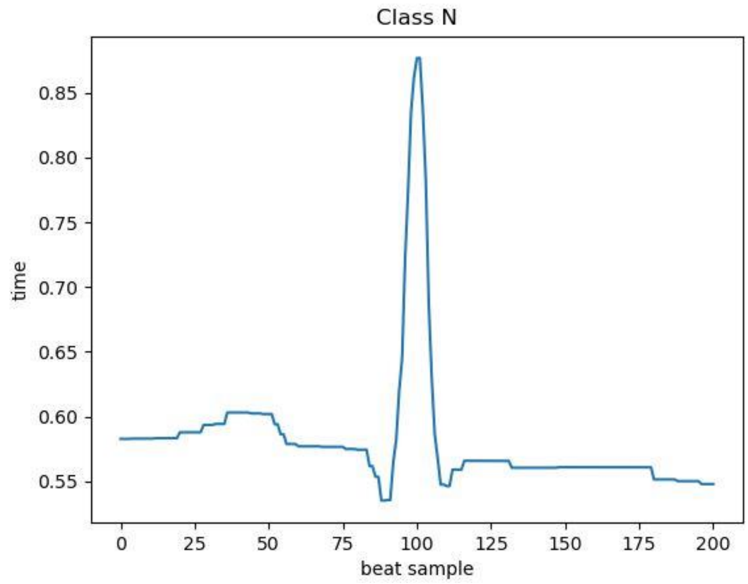
g) Storing the transformed and cleaned data in a .pk file that can be used to implement models locally

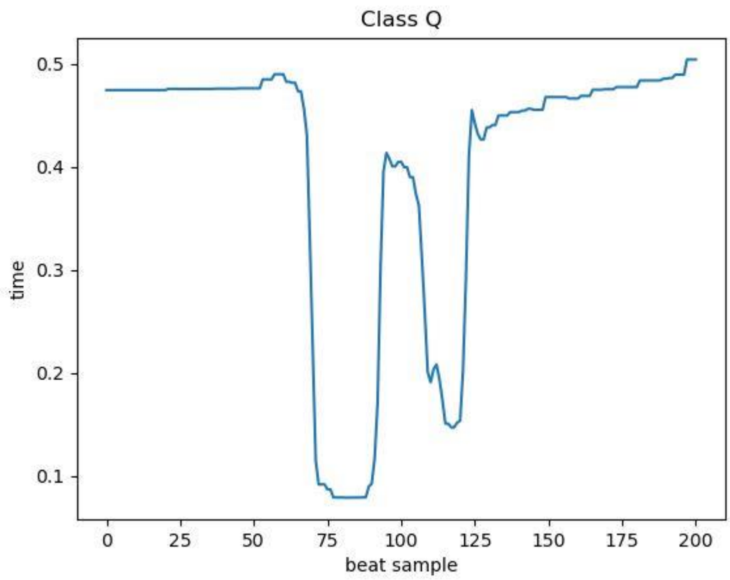
```
#Data to run ML algorithms locally
fn = "ECG_Data"+".pk"
with open(fn, "wb") as fw:
    pk.dump(X, fw, protocol=pk.HIGHEST_PROTOCOL)
    pk.dump(Y, fw, protocol=pk.HIGHEST_PROTOCOL)
```

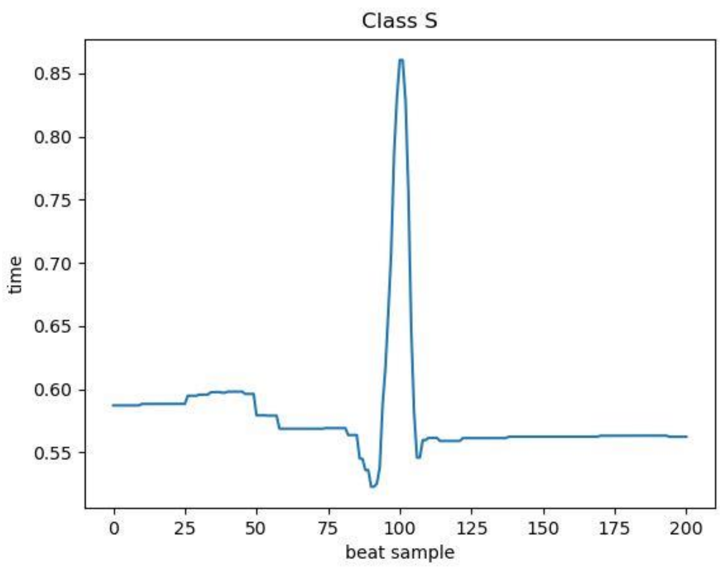
h) Save and run the python file using **python preprocess.py**

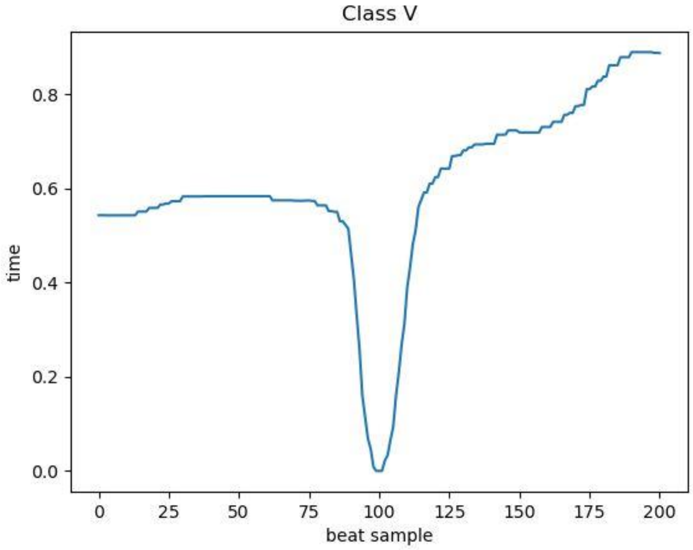
OUTPUTS -

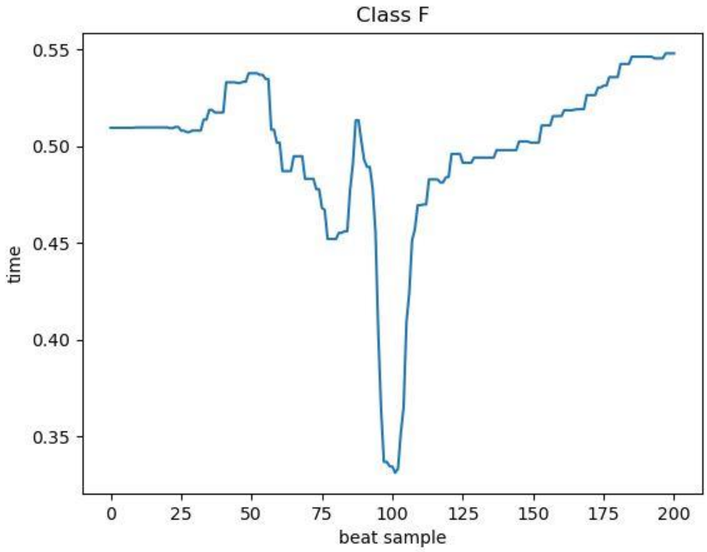



Class N
0.85
0.80
0.75
0.70
0.65
0.60
0.55
time
0
25
50
75
100
125
150
175
200
beat sample


Class Q
0.5
0.4
0.3
0.2
0.1
time
0
25
50
75
100
125
150
175
200
beat sample


Class S
0.85
0.80
0.75
0.70
0.65
0.60
0.55
time
0
25
50
75
100
125
150
175
200
beat sample





3.3 Experiment 1 – Local implementation of the classification and prediction models

3.3.1 Implementation of Deep Neural Network

a) On the terminal change the environment using **conda activate TensorFlow.** In this research project the name of the environment is TensorFlow.

b) Open a python file on the terminal to using a text editor to implement DNN

```
(base) hduser@rishab:~$ conda activate TensorFlow
(TensorFlow) hduser@rishab:~$ nano DNN.py
```

c) Import the required libraries

```
from matplotlib import pyplot
import numpy as np
import pickle as pk
import os, sys
from collections import Counter
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
import time
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend as K

from tensorflow.keras import callbacks
from tensorflow.keras.optimizers import Adam


start_time = time.time()
```

d) Load the dataset

```
#Load the pickled data
file = "ECG_Data"+".pk"
with open(file, "rb") as fn:
    X = pk.load(fn)
    Y = pk.load(fn)
```

e) Split the data into train data and test data in the ratio of 70:30

```
#Randomly split the Data into Train and Test in the ratio 70:30
r_seed = 1229
data_len = len(X)
np.random.seed(r_seed)
idx = list(range(data_len))
np.random.shuffle(idx)

train_len = int(data_len*0.7)
test_len = int(data_len*0.3)

X_train = X[idx][:train_len]
X_test = X[idx][train_len:]
Y_train = Y[idx][:train_len]
Y_test = Y[idx][train_len:]

X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

print(X_train.shape)
print(X_test.shape)
print(Counter(Y_train))
print(Counter(Y_test))
```

f) Create the DNN model

```
#Input shape and number of classes for classification
f_size = X_train.shape[1]
class_num = 5

#Learning rate and batch size
lr = 0.005
batch_size=128

#Convert the labels inot categorical values
Y_train = tf.keras.utils.to_categorical(Y_train, num_classes=class_num)

#Create the DNN model, using 4 layers and compile it
def make_model():
    model = Sequential()
    model.add(Conv1D(18, 7, activation='relu', input_shape=(f_size,1)))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(18, 7, activation='relu'))
    model.add(MaxPooling1D(2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(class_num, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=lr))
    return model

model = make_model()
```

g) Train and test the model and then evaluate it using accuracy, specificity, sensitivity, and execution time.

```
bin_label = lambda x: min(1,x)

#for e in range(1, 300+1):
for e in range(1, 50+1):
    #Train the model for 50 epochs
    model.fit(X_train, Y_train, batch_size=batch_size, epochs=1, verbose=0)

    #Fit the model on the test dataset
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1)

    #Evaluation
    acc = np.sum(y_pred==Y_test)/len(Y_test)

    #Map the predicted labels and the test labels for evaluation
    y_true = list(map(bin_label, Y_test))
    y_pred = list(map(bin_label, y_pred))

    #Extracting true positive, true negative, false positive and false negative
    #from the confusion matrix
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    SE = tp/(tp+fn)
    SP = tn/(fp+tn)

    print("Epoch: %d | SE: %.4f | ACC %.4f | AUC %.4f | SP: %.4f"%(e, SE, acc, auc, SP))

#Final Result
print("Final Result: \n")
print("SE: %.4f | ACC: %.4f | AUC: %.4f | SP: %.4f " %(SE, acc, auc, SP))

print("--- %s seconds ---" % (time.time() - start_time))
```

h) Close the python file and save it. Run the program using **python DNN.py**

OUTPUT –

```
Epoch: 1 | SE: 0.8145 | ACC 0.9590 | SP: 0.9935
Epoch: 2 | SE: 0.8917 | ACC 0.9694 | SP: 0.9906
Epoch: 3 | SE: 0.8783 | ACC 0.9737 | SP: 0.9970
Epoch: 4 | SE: 0.9068 | ACC 0.9752 | SP: 0.9925
Epoch: 5 | SE: 0.9171 | ACC 0.9809 | SP: 0.9965
Epoch: 6 | SE: 0.9119 | ACC 0.9800 | SP: 0.9969
Epoch: 7 | SE: 0.9404 | ACC 0.9842 | SP: 0.9960
Epoch: 8 | SE: 0.9298 | ACC 0.9818 | SP: 0.9958
Epoch: 9 | SE: 0.9308 | ACC 0.9840 | SP: 0.9974
Epoch: 10 | SE: 0.9362 | ACC 0.9843 | SP: 0.9969
Epoch: 11 | SE: 0.9350 | ACC 0.9833 | SP: 0.9959
Epoch: 12 | SE: 0.9395 | ACC 0.9838 | SP: 0.9965
Epoch: 13 | SE: 0.9444 | ACC 0.9851 | SP: 0.9964
Epoch: 14 | SE: 0.9473 | ACC 0.9794 | SP: 0.9889
Epoch: 15 | SE: 0.9503 | ACC 0.9854 | SP: 0.9956
Epoch: 16 | SE: 0.9503 | ACC 0.9861 | SP: 0.9964
Epoch: 17 | SE: 0.9383 | ACC 0.9851 | SP: 0.9973
Epoch: 18 | SE: 0.9332 | ACC 0.9844 | SP: 0.9973
Epoch: 19 | SE: 0.9505 | ACC 0.9849 | SP: 0.9946
Epoch: 20 | SE: 0.9181 | ACC 0.9825 | SP: 0.9987
Epoch: 21 | SE: 0.9426 | ACC 0.9854 | SP: 0.9967
Epoch: 22 | SE: 0.9392 | ACC 0.9848 | SP: 0.9968
Epoch: 23 | SE: 0.9496 | ACC 0.9861 | SP: 0.9963
Epoch: 24 | SE: 0.9470 | ACC 0.9847 | SP: 0.9956
Epoch: 25 | SE: 0.9494 | ACC 0.9865 | SP: 0.9970
Epoch: 26 | SE: 0.9546 | ACC 0.9862 | SP: 0.9955
Epoch: 27 | SE: 0.9487 | ACC 0.9846 | SP: 0.9960
Epoch: 28 | SE: 0.9564 | ACC 0.9857 | SP: 0.9946
Epoch: 29 | SE: 0.9565 | ACC 0.9873 | SP: 0.9966
Epoch: 30 | SE: 0.9604 | ACC 0.9875 | SP: 0.9955
Epoch: 31 | SE: 0.9609 | ACC 0.9867 | SP: 0.9945
Epoch: 32 | SE: 0.9445 | ACC 0.9866 | SP: 0.9979
Epoch: 33 | SE: 0.9600 | ACC 0.9866 | SP: 0.9955
Epoch: 34 | SE: 0.9574 | ACC 0.9870 | SP: 0.9959
Epoch: 35 | SE: 0.9604 | ACC 0.9873 | SP: 0.9953
Epoch: 36 | SE: 0.9463 | ACC 0.9848 | SP: 0.9959
Epoch: 37 | SE: 0.9489 | ACC 0.9873 | SP: 0.9975
Epoch: 38 | SE: 0.9482 | ACC 0.9856 | SP: 0.9960
Epoch: 39 | SE: 0.9616 | ACC 0.9878 | SP: 0.9958
Epoch: 40 | SE: 0.9541 | ACC 0.9873 | SP: 0.9968
Epoch: 41 | SE: 0.9564 | ACC 0.9874 | SP: 0.9964
Epoch: 42 | SE: 0.9489 | ACC 0.9869 | SP: 0.9971
Epoch: 43 | SE: 0.9571 | ACC 0.9868 | SP: 0.9955
Epoch: 44 | SE: 0.9656 | ACC 0.9872 | SP: 0.9945
Epoch: 45 | SE: 0.9565 | ACC 0.9850 | SP: 0.9939
Epoch: 46 | SE: 0.9628 | ACC 0.9868 | SP: 0.9946
Epoch: 47 | SE: 0.9626 | ACC 0.9879 | SP: 0.9954
Epoch: 48 | SE: 0.9611 | ACC 0.9862 | SP: 0.9941
Epoch: 49 | SE: 0.9484 | ACC 0.9873 | SP: 0.9976
Epoch: 50 | SE: 0.9701 | ACC 0.9869 | SP: 0.9929
Final Result:

SE: 0.9701 | ACC: 0.9869 | SP: 0.9929
--- 1057.7538135051727 seconds ---
```

3.3.2 Implementation of Random Forest model

a) On the terminal return to base environment and then open a python file to implement RF.

```
(TensorFlow) hduser@rishab:~$ conda activate
(base) hduser@rishab:~$ nano RF.py
```

b) Import the required packages

```
import pandas as pd
import numpy as np
import pickle as pk
import os, sys
import seaborn as sns
import time
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.metrics import *
from collections import Counter
from sklearn.ensemble import RandomForestClassifier
import time

start_time = time.time()
```

c) Split the data into train and test in the ration of 80:20

```
#Load the pickled data
file = "ECG_Data"+".pk"
with open(file, "rb") as fn:
    X = pk.load(fn)
    Y = pk.load(fn)

#Randomly split the Data into Train and Test in the ratio 80:20
r_seed = 1229
data_len = len(X)
np.random.seed(r_seed)
idx = list(range(data_len))
np.random.shuffle(idx)

train_len = int(data_len*0.8)
test_len = int(data_len*0.2)

X_train = X[idx][:train_len]
X_test = X[idx][train_len:]
Y_train = Y[idx][:train_len]
Y_test = Y[idx][train_len:]

print(X_train.shape)
print(X_test.shape)
print(Counter(Y_train))
print(Counter(Y_test))
```

d) Train and test the model and, then evaluate it using accuracy, specificity, sensitivity, and execution time.

```
#Random Forest Classification Model. Fit the model on the training set
rf = RandomForestClassifier(n_estimators=100)
rf_model = rf.fit(X_train, Y_train)

#Test the trained model on the training dataset
y_pred = rf_model.predict(X_test)

#Evaluation
acc = accuracy_score(Y_test, y_pred)
conf_mat = confusion_matrix(Y_test, y_pred)
print(type(conf_mat))
print(conf_mat)

#Extracting true positive, true negative, false positive and false negative
#from the confusion matrix
fp = conf_mat.sum(axis=0) - np.diag(conf_mat)
fn = conf_mat.sum(axis=1) - np.diag(conf_mat)
tp = np.diag(conf_mat)
tn = conf_mat.sum() - (fp + fn + tp)

#Converting the values into float values
fp = fp.astype(float)
fn = fn.astype(float)
tp = tp.astype(float)
tn = tn.astype(float)

#Computing sensitivity and specificity from tn, tp, fn, fp
SE = tp/(tp+fn)
SP = tn/(fp+tn)
SE = np.average(SE)
SP = np.average(SP)

print(" SE: %.4f | ACC: %.4f | SP: %4f" %(SE, acc, SP))

print("--- %s seconds ---" % (time.time() - start_time))
```

e) Save and close the python file. Run the program using **python RF.py**

OUTPUT –

```
 SE: 0.8362 | ACC: 0.9809 | SP: 0.981380
--- 359.75102639198303 seconds ---
```

3.3.3 Implementation of Support Vector Machine model

a) Open a python file on the terminal to implement SVM.

```
(base) hduser@rishab:~$ nano SVM.py
```

b) Import the libraries

```
import numpy as np
import pandas as pd
import pickle as pk
import os, sys
import seaborn as sns
import time
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.model_selection import  GridSearchCV
from sklearn.svm import LinearSVC
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform
from sklearn.svm import SVC
from sklearn.metrics import *
from collections import Counter

start_time = time.time()

```

c) Split the data into train and test in the ration of 80:20

```
#Load the pickled data
file = "ECG_Data"+".pk"
with open(file, "rb") as fn:
    X = pk.load(fn)
    Y = pk.load(fn)

#Randomly split the Data into Train and Test in the ratio 80:20
r_seed = 1229
data_len = len(X)
np.random.seed(r_seed)
idx = list(range(data_len))
np.random.shuffle(idx)

train_len = int(data_len*0.8)
test_len = int(data_len*0.2)

X_train = X[idx][:train_len]
X_test = X[idx][train_len:]
Y_train = Y[idx][:train_len]
Y_test = Y[idx][train_len:]

print(X_train.shape)
print(X_test.shape)
print(Counter(Y_train))
print(Counter(Y_test))
```

d) Train and test the model and, then evaluate it using accuracy, specificity, sensitivity, and execution time

```
#Support Vector Machine Classification Model. Fit the model on the training dataset
svm_clf = SVC(gamma="scale")
svm_clf.fit(X_train, Y_train)

#Hyperparameter tuning og C and gamma variable to prevent overfitting
param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10)}

#3-fold cross-validation, iterated 10 times
rnd_search_cv = RandomizedSearchCV(svm_clf, param_distributions, n_iter=10, verbose=2, cv=3)
rnd_search_cv.fit(X_train, Y_train)

#Fitting the best estimator on the Training dataset
rnd_search_cv.best_estimator_.fit(X_train, Y_train)

#Fit the trained model on the test data
y_pred = rnd_search_cv.best_estimator_.predict(X_test)

#Evaluation
acc = accuracy_score(Y_test, y_pred)
conf_mat = confusion_matrix(Y_test, y_pred)

#Extracting true positive, true negative, false positive and false negative
fp = conf_mat.sum(axis=0) - np.diag(conf_mat)
fn = conf_mat.sum(axis=1) - np.diag(conf_mat)
tp = np.diag(conf_mat)
tn = conf_mat.sum() - (fp + fn + tp)

#Converting the values into float values
fp = fp.astype(float)
fn = fn.astype(float)
tp = tp.astype(float)
tn = tn.astype(float)

#Computing sensitivity and specificity from tn, tp, fn, fp
SE = tp/(tp+fn)
SP = tn/(fp+tn)
SE = np.average(SE)
SP = np.average(SP)

print(" SE: %.4f | ACC: %.4f | SP: %4f" %(SE, acc, SP))

print("--- %s seconds ---" % (time.time() - start_time))
```

e) Save the file and close it. Execute the file using **python SVM.py**

OUTPUT –

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] END ....C=1.9493313853883334, gamma=0.00355224454904102; total time= 5.1min
[CV] END ....C=1.9493313853883334, gamma=0.00355224454904102; total time= 4.9min
[CV] END ....C=1.9493313853883334, gamma=0.00355224454904102; total time= 5.1min
[CV] END .....C=3.970768395498694, gamma=0.0718097108264451З; total time= 2.4min
[CV] END .....C=3.970768395498694, gamma=0.07180971082644513; total time= 2.4min
[CV] END .....C=3.970768395498694, gamma=0.07180971082644513; total time= 2.3min
[CV] END ...C=1.9793791377756678, gamma=0.006850493921060555; total time= 3.7min
[CV] END ...C=1.9793791377756678, gamma=0.006850493921060555; total time= 3.6min
[CV] END ...C=1.9793791377756678, gamma=0.006850493921060555; total time= 3.8min
[CV] END .....C=4.702307178313259, gamma=0.04682148275809945; total time= 2.2min
[CV] END .....C=4.702307178313259, gamma=0.04682148275809945; total time= 2.0min
[CV] END .....C=4.702307178313259, gamma=0.04682148275809945; total time= 2.2min
[CV] END ...C=9.306955424014042, gamma=0.0026964216806020817; total time= 3.5min
[CV] END ...C=9.306955424014042, gamma=0.0026964216806020817; total time= 3.5min
[CV] END ...C=9.306955424014042, gamma=0.0026964216806020817; total time= 3.6min
[CV] END ....C=8.805966025659725, gamma=0.024961736585479025; total time= 2.2min
[CV] END ....C=8.805966025659725, gamma=0.024961736585479025; total time= 2.1min
[CV] END ....C=8.805966025659725, gamma=0.024961736585479025; total time= 2.2min
[CV] END ....C=8.761939120022886, gamma=0.028187311100850966; total time= 2.2min
[CV] END ....C=8.761939120022886, gamma=0.028187311100850966; total time= 2.2min
[CV] END ....C=8.761939120022886, gamma=0.028187311100850966; total time= 2.2min
[CV] END .....C=4.274866401320841, gamma=0.07036430390020663; total time= 1.9min
[CV] END .....C=4.274866401320841, gamma=0.07036430390020663; total time= 1.9min
[CV] END .....C=4.274866401320841, gamma=0.07036430390020663; total time= 1.9min
[CV] END ..C=10.258605798671827, gamma=0.0013549245402061753; total time= 4.0min
[CV] END ..C=10.258605798671827, gamma=0.0013549245402061753; total time= 3.9min
[CV] END ..C=10.258605798671827, gamma=0.0013549245402061753; total time= 4.0min
[CV] END ...C=9.720765392469097, gamma=0.0010623844616929794; total time= 4.2min
[CV] END ...C=9.720765392469097, gamma=0.0010623844616929794; total time= 4.1min
[CV] END ...C=9.720765392469097, gamma=0.0010623844616929794; total time= 4.2min
 SE: 0.7441 | ACC: 0.9637 | SP: 0.965496
--- 6156.775834560394 seconds ---
```

3.4 Experiment 2 – Implementation of the classification and prediction models on Apache Spark

The first step in this experiment is to load the data into Hadoop Distributed File System (HDFS).

```
(base) hduser@rishab:~/Final$ hdfs dfs -copyFromLocal ECG.csv /user/hduser
```

To check whether the dataset has been successfully loaded or not type the following command:

```
(base) hduser@rishab:~/Final$ hdfs dfs -ls
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Found 6 items
drwxr-xr-x   - hduser supergroup          0 2021-07-26 09:45 .sparkStaging
-rw-r--r--   1 hduser supergroup  143741699 2021-08-02 16:01 ECG.csv
drwxr-xr-x   - hduser supergroup          0 2021-07-27 18:35 ECG_Data
-rw-r--r--   1 hduser supergroup  143741699 2021-08-01 23:23 ECG_Data.csv
drwxr-xr-x   - hduser supergroup          0 2021-07-26 09:18 Final_Data
-rw-r--r--   1 hduser supergroup  143741699 2021-08-02 08:59 Test_Data.csv
```

3.4.1 Implementation of Deep Neural Network model

a) Import libraries

```
from __future__ import print_function
from sklearn.metrics import confusion_matrix
import pyspark.sql.functions as F
from matplotlib import pyplot
from pyspark.mllib.evaluation import MulticlassMetrics
from collections import Counter
from pyspark.sql.functions import col
import pandas as pd
from pyspark.sql import SparkSession
from pyspark import SparkContext
from sklearn.metrics import *
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend as K
from tensorflow.keras import callbacks
from tensorflow.keras.optimizers import Adam
from pyspark.sql.types import FloatType
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from tensorflow.keras import optimizers, regularizers
from hyperas import optim
from hyperopt import Trials, STATUS_OK, tpe
from hyperas.distributions import choice, uniform
```

b) Create Spark Context and start Spark Session

```
#Create a Spark Context
sc = SparkContext()

#Create a function to read the data and output train dataset and test dataset
def data():
    spark = SparkSession \
        .builder \
        .appName("CNN in Spark") \
        .getOrCreate()
```

c) Import the dataset from hdfs.

```

    Data = spark.read.format("csv").option("header", "true").load('hdfs:///user/hduser/ECG.csv')

```

d) Change the data type for the columns from object to float. Then rename the 0_y column to 'label'. Select the features and labels and convert them into NumPy arrays.

```
    Data = Data.select(*(col(c).cast("float").alias(c) for c in Data.columns))
    Data = Data.withColumnRenamed("0_y", "label")
    X = np.array(Data.select(*(col(c) for c in Data.columns if c not in {'label'})).collect())
    Y = np.array(Data.select('label').collect())
    Y = Y.flatten()
```

e) Split the features and labels into training set and test set in the ratio 80:20.

```
    class_num = 5
    r_seed = 1229
    data_len = len(X)
    np.random.seed(r_seed)
    idx = list(range(data_len))
    np.random.shuffle(idx)

    train_len = int(data_len*0.8)
    test_len = data_len-train_len

    X_train = X[idx][:train_len]
    X_test = X[idx][train_len:]
    Y_train = Y[idx][:train_len]
    Y_test = Y[idx][train_len:]


    Y_train = tf.keras.utils.to_categorical(Y_train, num_classes=class_num)
    Y_test = tf.keras.utils.to_categorical(Y_test, num_classes=class_num)

    return X_train, Y_train, X_test, Y_test
```

f) Function to create a Deep neural Network using 4 layers.

```
#Function to create a Deep Neural Network model using 4 layers
def model(X_train, Y_train, X_test, Y_test):
    model = Sequential()
    model.add(Dense(512, input_shape=(201,)))
    model.add(Activation('relu'))
    model.add(Dropout({{uniform(0, 1)}}))
    model.add(Dense({{choice([256, 512, 1024])}}))
    model.add(Activation('relu'))
    model.add(Dropout({{uniform(0, 1)}}))
    model.add(Dense(5))
    model.add(Activation('softmax'))
    #Compile the model
    model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
    #Train the model
    result = model.fit(X_train, Y_train,
              batch_size={{choice([64, 128])}},
              epochs=5,
              verbose=2,
              validation_split=0.2)

    #Find the best accuracy for each epoch
    validation_acc = np.amax(result.history['val_accuracy'])
    print('Best validation acc of epoch:', validation_acc)

    #Return the best accuracy and status of the model
    return {'loss': -validation_acc, 'status': STATUS_OK, 'model': model}
```

g) Use HyperOpt[2] to implement keras parallely. This function also helps select the best model (optimal model) during training that can be tested and validated.

```python
#Using HyperOpt function, the DNN can be executed parallely
best_run, best_model = optim.minimize(model=model,
                                      data=data,
                                      algo=tpe.suggest,
                                      max_evals=5,
                                      trials=Trials())
X_train, Y_train, X_test, Y_test = data()
print("Evalutation of best performing model:")
print(best_model.evaluate(X_test, Y_test))
```

h) Use the best configuration model for evaluation.

```python
#Select the best configuration of the model and test it on the test data
y_pred = best_model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_true = list(Y_test)
y_true = np.argmax(y_true, axis=1)

#print the confusion matrix
conf = confusion_matrix(y_true, y_pred).ravel()

#Reshape the single array confusion matrix into nxn matrix
conf = np.reshape(conf,(5,5))

print(type(conf))
#Extract tn, tp, fn, fp from the confusion matrix
fp = conf.sum(axis=0) - np.diag(conf)
fn = conf.sum(axis=1) - np.diag(conf)
tp = np.diag(conf)
tn = conf.sum() - (fp + fn + tp)

fp = fp.astype(float)
fn = fn.astype(float)
tp = tp.astype(float)
tn = tn.astype(float)

#Calculate the sensitivity and specificity
SE = tp/(tp+fn)
SP = tn/(fp+tn)
SE = np.average(SE)
SP = np.average(SP)

print(" SE: %.4f | SP: %4f" %(SE, SP))
```

i) Save the file and clost it. Run the python file using **spark-submit.**

```
(TensorFlow) hduser@rishab:~/Final/Spark$ spark-submit SparkCNN.py
```

[2] https://pythonrepo.com/repo/maxpumperla-hyperas-python-deep-learning

OUTPUT –

```
Epoch 1/5
548/548 - 19s - loss: 0.4416 - accuracy: 0.8875 - val_loss: 0.3745 - val_accuracy: 0.9167

Epoch 2/5
548/548 - 16s - loss: 0.3051 - accuracy: 0.9256 - val_loss: 0.2539 - val_accuracy: 0.9381

Epoch 3/5
548/548 - 12s - loss: 0.2791 - accuracy: 0.9308 - val_loss: 0.2307 - val_accuracy: 0.9384

Epoch 4/5
548/548 - 17s - loss: 0.2603 - accuracy: 0.9344 - val_loss: 0.2070 - val_accuracy: 0.9419

Epoch 5/5
548/548 - 14s - loss: 0.2551 - accuracy: 0.9351 - val_loss: 0.2027 - val_accuracy: 0.9436

Best validation acc of epoch:
0.9436450600624084
100%|██████████| 5/5 [05:58<00:00, 71.79s/trial, best loss: -0.9436450600624084]
```

```
Evalutation of best performing model:
685/685 [==============================] - 5s 7ms/step - loss: 0.2196 - accuracy: 0.9394
[0.2196389138698578, 0.9393870234489441]
<class 'numpy.ndarray'>
 SE: 0.5314 | SP: 0.938900
```

3.4.2 Implementation of Random Forest model

a) Import the libraries

```python
from sklearn.metrics import classification_report, confusion_matrix
from pyspark.sql.functions import rand
import sklearn
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.functions import col
import pandas as pd
from pyspark.sql import SparkSession
from pyspark import SparkContext
import numpy as np
from pyspark.sql.types import FloatType
from pyspark.ml.classification import LinearSVC, OneVsRest
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
import pyspark.sql.functions as F
```

b) Create Spark Context and start Spark Session

```python
#Create a Spark Context
sc = SparkContext()

#Enable a spark session
spark = SparkSession \
    .builder \
    .appName("RF in Spark") \
    .getOrCreate()
```

c) Read the data from hdfs. Change the data type from object to float. Select all the feature columns and convert them into a vector. Select features and labels columns.

```
#Read thhe data from HDFS
Data = spark.read.format("csv").option("header", "true").load('hdfs:///user/hduser/ECG.csv')

#Rename the last coulm to labels
Data = Data.withColumnRenamed("0_y", "label")

#Convert all the column into floating type
Data = Data.select(*(col(c).cast("float").alias(c) for c in Data.columns))

#Select all the feature columns (201 columns) and
#store it a vector named features
Features_assem = VectorAssembler(inputCols=[c for c in Data.columns if c not in {'label'}],
outputCol="features")

#Apply the transformation to the data
Data = Features_assem.transform(Data)
```

d) Split the data in inito train and test ration in the ration 70:30. Create the random forest model with 200 trees with a tree depth of 5. Train the model and evaluate the performance of the model on the test data.

```
#Split the Data into train and test data in the ratio 70:30
Train_Data, Test_Data = Final_Data.randomSplit([0.7,0.3])

#Create a random forest model
rf = RandomForestClassifier(labelCol="label",
                            featuresCol="features",
                            numTrees = 200,
                            maxDepth = 5,
                            maxBins = 30)

#Fit the model on the training data
rf_model = rf.fit(Train_Data)

#Test the trained model on the test dataset
predictions = rf_model.transform(Test_Data)

#Select the features and labels from the predicted data for evaluation
y_true = predictions.select(['label']).collect()
y_pred = predictions.select(['prediction']).collect()

#Print the classification report
print(classification_report(y_true, y_pred))

#Create a confusion matrix
conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)

#Extract tn, tp, fn, fp from confusion matrix
fp = conf_mat.sum(axis=0) - np.diag(conf_mat)
fn = conf_mat.sum(axis=1) - np.diag(conf_mat)
tp = np.diag(conf_mat)
tn = conf_mat.sum() - (fp + fn + tp)

fp = fp.astype(float)
fn = fn.astype(float)
tp = tp.astype(float)
tn = tn.astype(float)

#Calculate the specificity and sensitivity
SE = tp/(tp+fn)
SP = tn/(fp+tn)
SE = np.average(SE)
SP = np.average(SP)

print(" SE: %.4f | SP: %4f" %(SE, SP))
```

e) Save the file and close it. Rub the program using the following command

```
(base) hduser@rishab:~/Final/Spark$ spark-submit SparkRF.py
```

OUTPUT –

```
              precision    recall  f1-score   support

         0.0       0.90      1.00      0.95     27222
         1.0       0.00      0.00      0.00       875
         2.0       0.96      0.55      0.70      2104
         3.0       0.00      0.00      0.00       259
         4.0       1.00      0.59      0.74      2358

    accuracy                           0.91     32818
   macro avg       0.57      0.43      0.48     32818
weighted avg       0.88      0.91      0.88     32818

[[27201     0    20     0     1]
 [  874     0     1     0     0]
 [  942     0  1161     0     1]
 [  231     0    28     0     0]
 [  958     0     5     0  1395]]
 SE: 0.4285 | SP: 0.892237
```

3.4.3 Implementation of Support Vector Machine model

a) Import the required libraries.

```python
from sklearn.metrics import classification_report, confusion_matrix
import sklearn
from pyspark.mllib.evaluation import MulticlassMetrics
from collections import Counter
from pyspark.sql.functions import col
import pandas as pd
from pyspark.sql import SparkSession
from pyspark import SparkContext
import numpy as np
from pyspark.sql.types import FloatType
from pyspark.ml.classification import LinearSVC, OneVsRest
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
```

b) Create Spark Context and start Spark Session

```python
#Create a Spark Context
sc = SparkContext()

#Enable a spark session
spark = SparkSession \
    .builder \
    .appName("SVM in Spark") \
    .getOrCreate()
```

c) Read the data from hdfs. Change the data type from object to float. Select all the feature columns and convert them into a vector. Select features and labels columns.

```
#Read thhe data from HDFS
Data = spark.read.format("csv").option("header", "true").load('hdfs:///user/hduser/ECG.csv')

#Convert all the column into floating type
Data = Data.select(*(col(c).cast("float").alias(c) for c in Data.columns))

#Rename the last coulm to labels
Data = Data.withColumnRenamed("0_y", "label")

#Select all the feature columns (201 columns) and
#store it a vector named features
Features_assem = VectorAssembler(inputCols=[c for c in Data.columns if c not in {'label'}],
outputCol="features")

#Apply the transformation to the data
Data = Features_assem.transform(Data)

#Select the features and labels columns for classification
Final_Data = Data.select(col("label"), col("features"))
```

d) Split the data into train and test. Create the support vector machine using Linear Support Vector Classifier. Since there are five classes, one vs all classifier is attached to the model to handle multi-classification. The model is trained for 25 iterations and evaluated.

```
#Split the Data into train and test data in the ratio 70:30
Train_Data, Test_Data = Final_Data.randomSplit([0.8,0.2])

#Create a linear model with 25 iterations
#Fit one vs all classifier-to enable multiclass classification
lsvc = LinearSVC(maxIter=25, regParam=0.001)
ovr = OneVsRest(classifier=lsvc)

#Fit the model on the training data
model = ovr.fit(Train_Data)

#Test the model on test data
predictions = model.transform(Test_Data)

#Select the features and labels from the predicted data for evaluation
y_true = predictions.select(['label']).collect()
y_pred = predictions.select(['prediction']).collect()

#Print the classification report
print(classification_report(y_true, y_pred))

#Create a confusion matrix
conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)

#Extract tn, tp, fn, fp from confusion matrix
fp = conf_mat.sum(axis=0) - np.diag(conf_mat)
fn = conf_mat.sum(axis=1) - np.diag(conf_mat)
tp = np.diag(conf_mat)
tn = conf_mat.sum() - (fp + fn + tp)

fp = fp.astype(float)
fn = fn.astype(float)
tp = tp.astype(float)
tn = tn.astype(float)

#Calculate the accuracy specificity and sensitivity
acc = (tp+tn)/(tp+tn+fp+fn)
acc = np.average(acc)
SE = tp/(tp+fn)
SP = tn/(fp+tn)
SE = np.average(SE)
SP = np.average(SP)

print(" SE: %.4f | SP: %4f | acc: %.4f" %(SE, SP, acc))
```

OUTPUT –

```
              precision    recall  f1-score   support

         0.0       0.83      1.00      0.91     18131
         1.0       0.00      0.00      0.00       577
         2.0       1.00      0.00      0.00      1467
         3.0       0.00      0.00      0.00       159
         4.0       0.00      0.00      0.00      1592

    accuracy                           0.83     21926
   macro avg       0.37      0.20      0.18     21926
weighted avg       0.75      0.83      0.75     21926

[[18131     0     0     0     0]
 [  577     0     0     0     0]
 [ 1466     0     1     0     0]
 [  159     0     0     0     0]
 [ 1592     0     0     0     0]]
<class 'numpy.ndarray'>
 SE: 0.2001 | SP: 0.800053 | acc: 0.9308
```

4 Conclusion

All the above steps were implemented to create Electrocardiogram classification and prediction using the MIT-BIH dataset. All the tools used were mentioned throughout the implementation stages. Two experiments – Local implementation and Apache Spark were implemented to answer the research questions and research objectives.

It is important to use the specified versions of Hadoop and Apache Spark, as errors or glitches might arise if any other version is used.

REFERENCES

[1] https://pywavelets.readthedocs.io/en/latest/\
[2] https://github.com/MIT-LCP/wfdb-python
[3] https://keras.io/guides/writing_a_training_loop_from_scratch/