Configuration Manual

MSc Research Project
Data Analytics

Souvik Maitra
Student ID: X20194617

School of Computing
National College of Ireland

Supervisor: Mr. Prashanth Nayak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Souvik Maitra
Student ID:	X20194617
Programme:	Data Analytics **Year:** 2022
Module:	Research Project
Lecturer:	Mr. Prashanth Nayak
Submission Due Date:	15/08/2022
Project Title:	Prediction Of Non-Alcoholic Fatty Liver Disease Stages Through CT-Scan And Sonography Using Neural Network
Word Count:	**922** **Page Count: 23**

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: ………………………………………………………………………………………………

Date: ………………………………………………………………………………………………

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

Forename Surname
Student ID:

1 Introduction

In this Configuration Manual all the perquisites required to reproduce the research and its outcomes on individual environment are mentioned. The software and the hardware requirement along with a snapshot of code for Data Import and Exploratory Data Analysis , Data Pre-processing, Label Encoding, Feature Selection, all the models-built Cross Validation and Evaluation are included. The structure of the report is as follows, Section 2, gives the information about environment configuration.
Section 3, provides detail about data collection. Section 4 is data exploration consists of Data Pre-processing and Exploratory Data Analysis. Label Encoding is explained in section 5. Section 6 provides the details about Feature Selection. Class Balancing and train test splits for the data for model training and testing are covered in Section 7 . Section 8 provides the details about the models built and cross validation. Section 9, explains how results are computed and visualized.

2 Environment

This section provides the details of Software and Hardware requirements to implement the research done.

2.1 Hardware Requirements

Below Figure 1, provides the hardware specifications required. Intel i5-1135G7 is the 11th Generation Intel Core CPU @ 2.40 GHz, 8 GB installed DDR4 RAM Memory at speed of 3200 Mhz, 64 Bit Windows 10 operating System, 512 GB SSD.

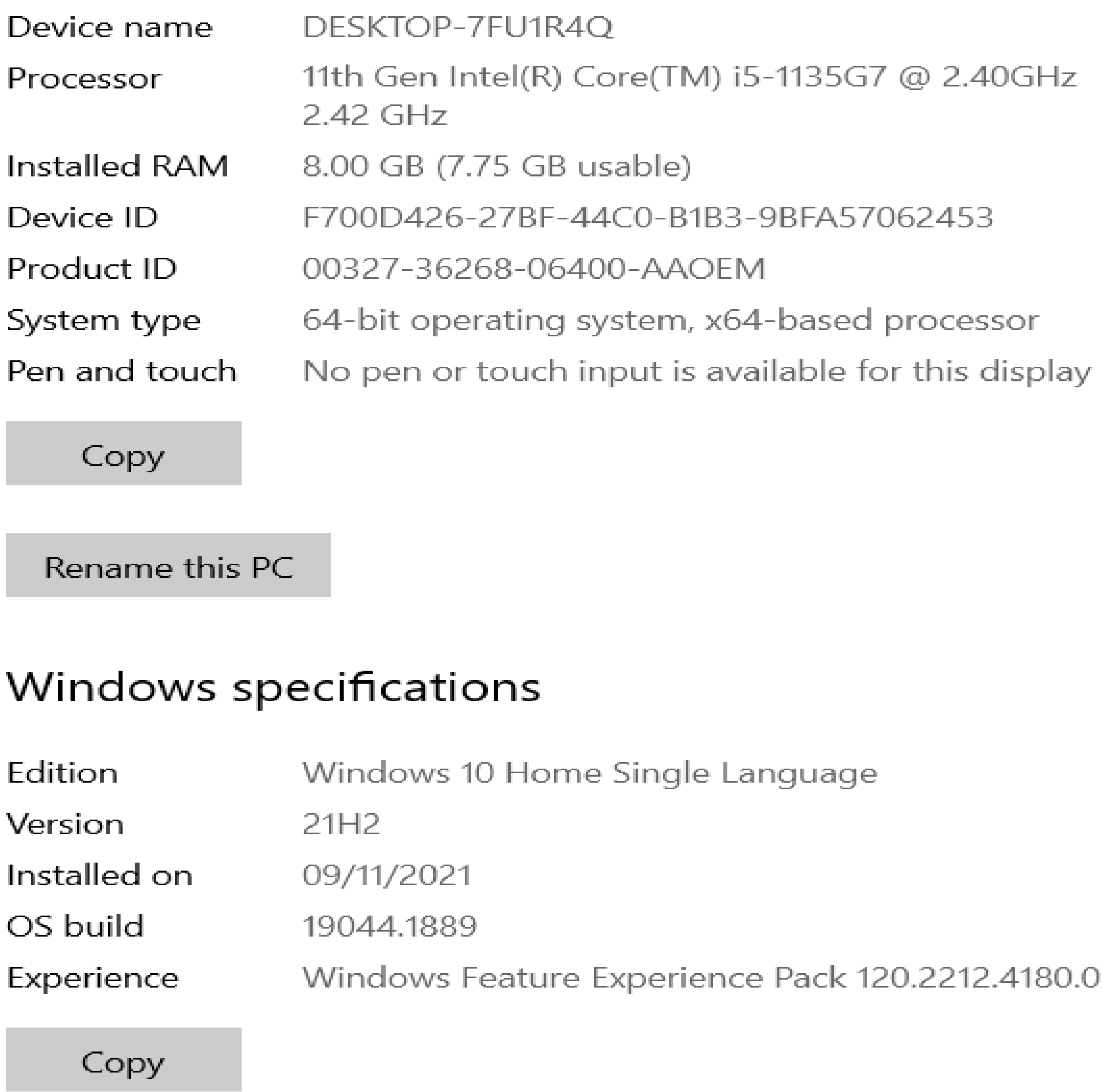

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 for Windows (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

3 Data Collection

The data is taken from https://www.kaggle.com/datasets/fedesoriano/cirrhosis-prediction-dataset.

4 Data Exploration

All the Python libraries required to implement the entire project are listed in Figure 2.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
import tensorflow as tf
from sklearn.ensemble import RandomForestClassifier
from tensorflow.keras.layers import Dense, Input, Dropout, Flatten
from tensorflow.keras.models import Sequential, model_from_json
from tensorflow.keras import optimizers
import joblib
import logging
logging.getLogger('tensorflow').disabled = True
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

Figure 2: Required Python Libraries

The Figure 3 represents the code to import data.

```python
data = pd.read_csv("/content/drive/MyDrive/Thesis_Code/NAFLD.csv")
```

```python
data.head()
```

	ID	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema
0	1	400	D	D-penicillamine	21464	F	Y	Y	Y	Y
1	2	4500	C	D-penicillamine	20617	F	N	Y	Y	N
2	3	1012	D	D-penicillamine	25594	M	N	N	N	S
3	4	1925	D	D-penicillamine	19994	F	N	Y	Y	S

Figure 3: Data Import

The Figure 4 represents the code to check data information and the count of missing values for each feature column.

```
data.columns
```

```
Index(['ID', 'N_Days', 'Status', 'Drug', 'Age', 'Sex', 'Ascites',
       'Hepatomegaly', 'Spiders', 'Edema', 'Bilirubin', 'Cholesterol',
       'Albumin', 'Copper', 'Alk_Phos', 'SGOT', 'Tryglicerides', 'Platelets',
       'Prothrombin', 'Stage'],
      dtype='object')
```

Figure 4: Data Columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   ID             418 non-null    int64  
 1   N_Days         418 non-null    int64  
 2   Status         418 non-null    object 
 3   Drug           312 non-null    object 
 4   Age            418 non-null    int64  
 5   Sex            418 non-null    object 
 6   Ascites        312 non-null    object 
 7   Hepatomegaly   312 non-null    object 
 8   Spiders        312 non-null    object 
 9   Edema          418 non-null    object 
 10  Bilirubin      418 non-null    float64
 11  Cholesterol    284 non-null    float64
 12  Albumin        418 non-null    float64
 13  Copper         310 non-null    float64
 14  Alk_Phos       312 non-null    float64
 15  SGOT           312 non-null    float64
 16  Tryglicerides  282 non-null    float64
 17  Platelets      407 non-null    float64
 18  Prothrombin    416 non-null    float64
 19  Stage          412 non-null    float64
dtypes: float64(10), int64(3), object(7)
memory usage: 65.4+ KB
```

Figure 5: Data Information

```
data.describe()
```

	ID	N_Days	Age	Bilirubin	Cholesterol	Albumin	Copper
count	418.000000	418.000000	418.000000	418.000000	284.000000	418.000000	310.000000
mean	209.500000	1917.782297	18533.351675	3.220813	369.510563	3.497440	97.648387
std	120.810458	1104.672992	3815.845055	4.407506	231.944545	0.424972	85.613920
min	1.000000	41.000000	9598.000000	0.300000	120.000000	1.960000	4.000000
25%	105.250000	1092.750000	15644.500000	0.800000	249.500000	3.242500	41.250000
50%	209.500000	1730.000000	18628.000000	1.400000	309.500000	3.530000	73.000000
75%	313.750000	2613.500000	21272.500000	3.400000	400.000000	3.770000	123.000000
max	418.000000	4795.000000	28650.000000	28.000000	1775.000000	4.640000	588.000000

Figure 6: Data Statistics

```
data.isnull().sum()
```

```
ID                 0
N_Days             0
Status             0
Drug             106
Age                0
Sex                0
Ascites          106
Hepatomegaly     106
Spiders          106
Edema              0
Bilirubin          0
Cholesterol      134
Albumin            0
Copper           108
Alk_Phos         106
SGOT             106
Tryglicerides    136
Platelets         11
Prothrombin        2
Stage              6
dtype: int64
```

Figure 7: Checking for missing Values

```
data=data.drop(['ID', 'N_Days', 'Status'], axis=1)
data['Drug'] = data['Drug'].replace({'D-penicillamine':0, 'Placebo':1})
data['Cholesterol'].fillna(int(data['Cholesterol'].mean()), inplace=True)
data['Copper'].fillna(int(data['Copper'].mean()), inplace=True)
data['Alk_Phos'].fillna(int(data['Alk_Phos'].mean()), inplace=True)
data['SGOT'].fillna(int(data['SGOT'].mean()), inplace=True)
data['Tryglicerides'].fillna(int(data['Tryglicerides'].mean()), inplace=True
data['Platelets'].fillna(int(data['Platelets'].mean()), inplace=True)
data['Prothrombin'].fillna(int(data['Prothrombin'].mean()), inplace=True)
data['Hepatomegaly'] = data['Hepatomegaly'].replace({'N':0, 'Y':1})
data['Spiders'] = data['Spiders'].replace({'N':0, 'Y':1})
data['Edema'] = data['Edema'].replace({'N':0, 'Y':1, 'S':-1})
data['Sex'] = data['Sex'].replace({'M':0, 'F':1})
data['Ascites'] = data['Ascites'].replace({'N':0, 'Y':1})
```

```
data = data[data['Stage'].notna()]

# Numerical --> Median
numColumns = data.select_dtypes(include=(['int64', 'float64'])).columns

for col in numColumns:
    data[col].fillna(data[col].median(), inplace=True)

# Categorical --> Most Frequent
catColumns = data.select_dtypes(include=('object')).columns

for col in catColumns:
    data[col].fillna(data[col].mode().values[0], inplace=True)


data.Stage = data.Stage.astype(int)
```

Figure 8: Handling Missing Data

```
# Plotting the Number of patients with liver disease vs Number of patients with no liver disease
sns.countplot(data=data, x = 'Stage', label='Count')

S1, S2, S3, S4 = data['Stage'].value_counts()
print('Number of patients diagnosed with Stage 1 disease: ',S1)
print('Number of patients diagnosed with Stage 2 disease: ',S2)
print('Number of patients diagnosed with Stage 3 disease: ',S3)
print('Number of patients diagnosed with Stage 4 disease: ',S4)
```

```
Number of patients diagnosed with Stage 1 disease:  155
Number of patients diagnosed with Stage 2 disease:  144
Number of patients diagnosed with Stage 3 disease:  92
Number of patients diagnosed with Stage 4 disease:  21
```

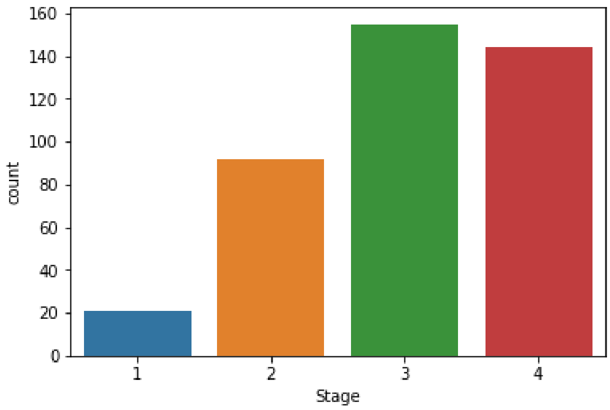

Figure 9: Plotting the Number of patients with liver disease vs Number of patients with no liver disease

```
# Plotting the Number of Male and Female patients
sns.countplot(data=data, x = 'Sex', label='Count')

M, F = data['Sex'].value_counts()
print('Number of patients that are male: ',M)
print('Number of patients that are female: ',F)
```

```
Number of patients that are male:  368
Number of patients that are female:  44
```

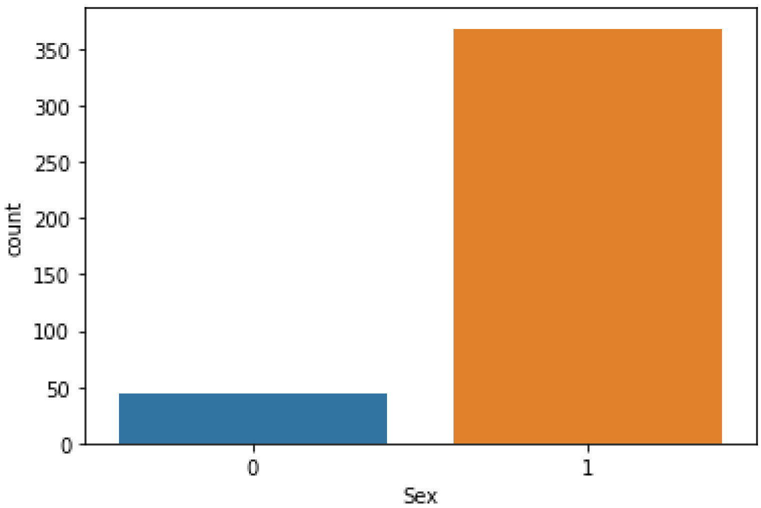

Figure 10: Plotting the Number of Male and Female patients

```
data[['Sex', 'Stage','Age']].groupby(['Stage','Sex'], a
```

	Stage	Sex	Age
6	4	0	22352.411765
7	4	1	19274.511811
4	3	0	19704.125000
5	3	1	17674.007194
2	2	0	18662.125000
3	2	1	18010.797619
0	1	0	16924.333333
1	1	1	17139.444444

Figure 11: Plotting patient Age vs Gender

```
# Plotting Age vs Gender
g = sns.FacetGrid(data, col="Stage", row="Sex", margin_titles=True)
g.map(plt.hist, "Age", color="red")
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Disease by Gender and Age')
```

```
Text(0.5, 0.98, 'Disease by Gender and Age')
```

Figure 12: Plotting patient Age vs Gender

```python
# Plotting Gender(Male/Female) along with Bilirubin and Prothrombin
g = sns.FacetGrid(data, col="Sex", row="Stage", margin_titles=True)
g.map(plt.scatter,"Bilirubin", "Prothrombin", edgecolor="w")
plt.subplots_adjust(top=0.9)
```

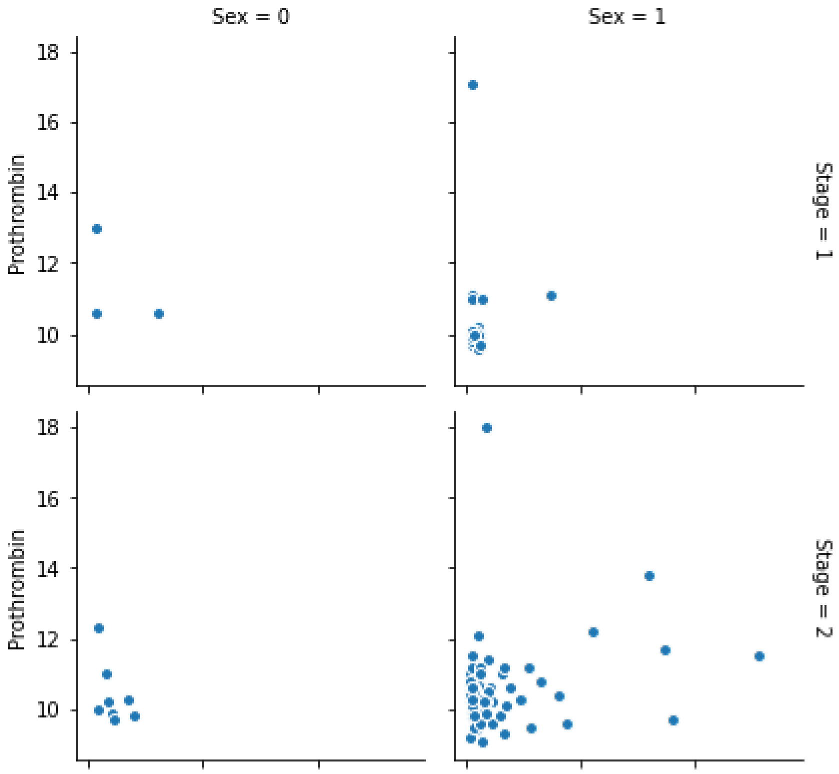

Figure 13: Plotting Gender (Male/Female) along with Bilirubin and Prothrombin

```
# Plotting Bilirubin and Prothrombin
sns.jointplot("Bilirubin", "Prothrombin", data=data, kind="reg")
```

```
seaborn.axisgrid.JointGrid at 0x7ff1ea54c910>
```

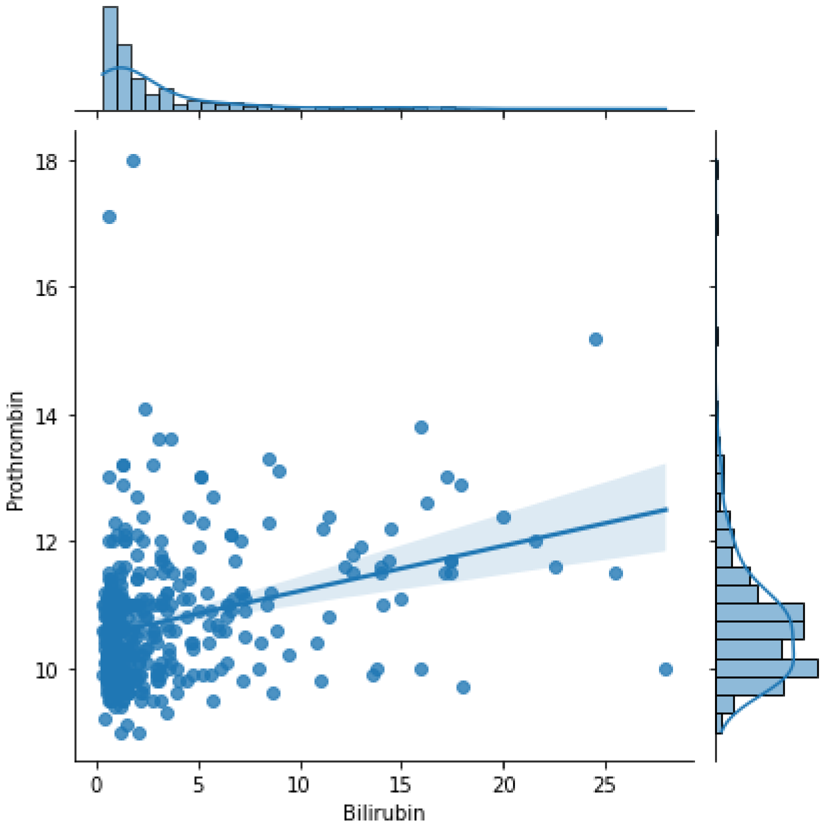

Figure 14: Plotting Bilirubin and Prothrombin

```python
# Plotting Alkaline_Phosphotase vs Albumin
sns.jointplot("Alk_Phos", "Albumin", data=data, kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x7ff1ea40f810>
```

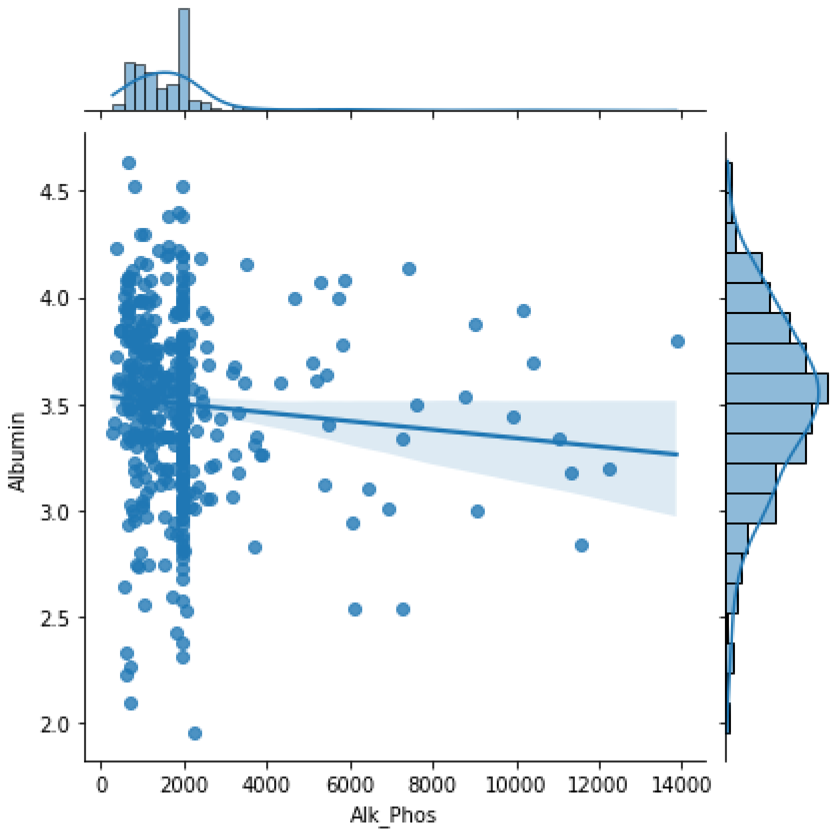

Figure 15: Plotting Alkaline_Phosphotase vs Albumin

```
# Plotting Sex(Male/Female) along with Cholesterol and Copper
g = sns.FacetGrid(data, col="Sex", row="Stage", margin_titles=True)
g.map(plt.scatter,"Cholesterol", "Copper",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

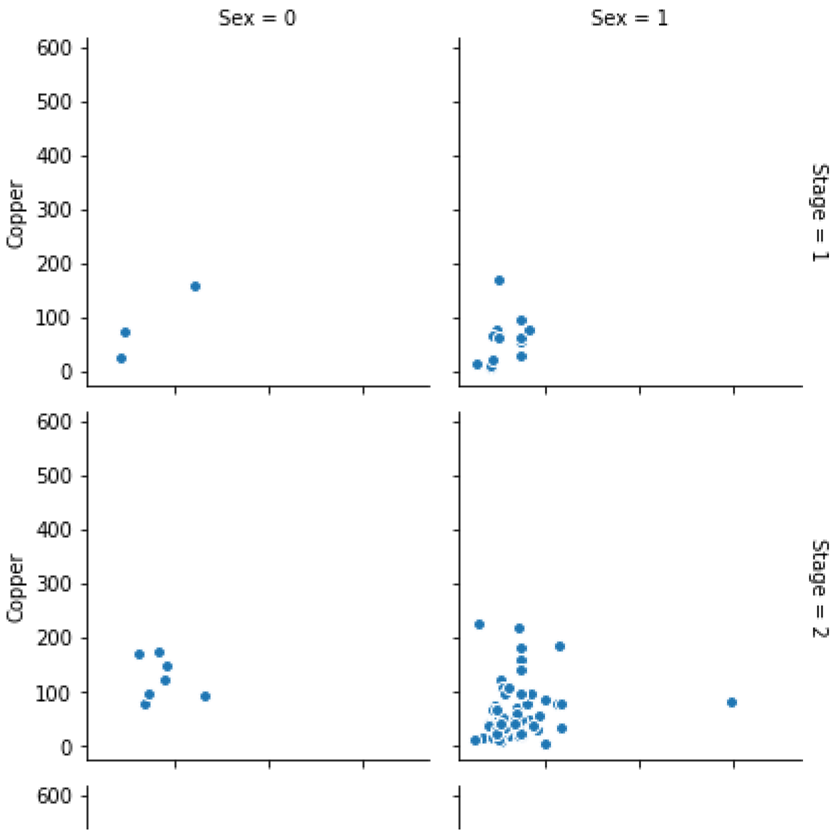

Figure 16: Plotting Sex (Male/Female) along with Cholesterol and Copper

```
]: # Plotting Gender(Male/Female) along with SGOT and Platelets and Triglycerides
g = sns.FacetGrid(data, col="Sex", row="Stage", margin_titles=True)
g.map(plt.scatter,"SGOT", 'Platelets','Tryglicerides',  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

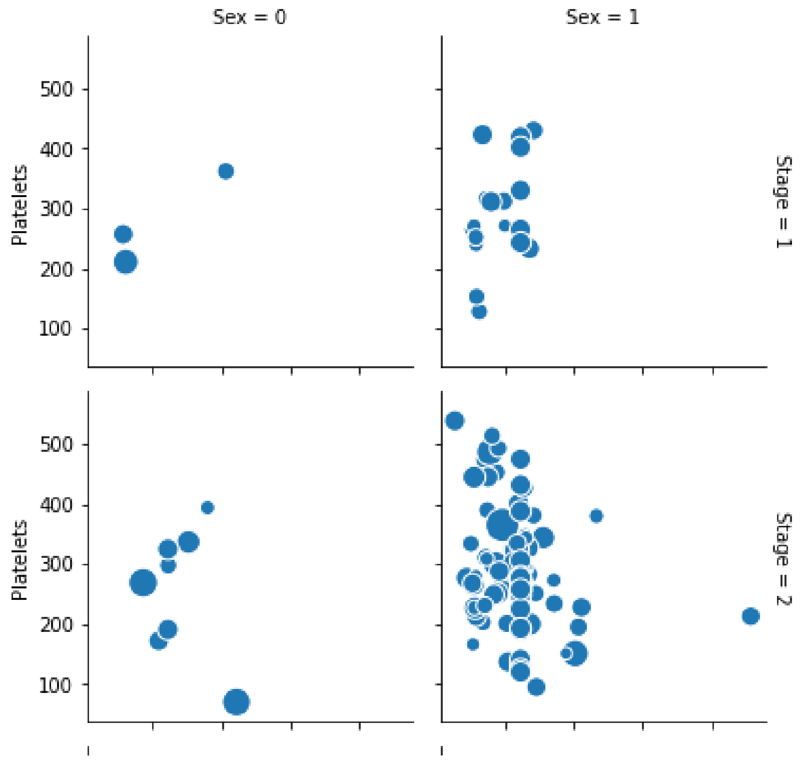

Figure 17: Plotting Gender (Male/Female) along with SGOT and Platelets and Triglycerides

5 Label Encoding

The Figure 18, illustrate the code to encode all the columns of object type.

```
objs = data.select_dtypes(include=['object']).columns

for cat in objs:
    le = LabelEncoder()
    le.fit(data[cat])
    data[cat] = le.transform(data[cat])
data.head()
```

	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Chole
0	0.0	21464	1	1.0	1.0	1.0	1	14.5	
1	0.0	20617	1	0.0	1.0	1.0	0	1.1	
2	0.0	25594	0	0.0	0.0	0.0	-1	1.4	
3	0.0	19994	1	0.0	1.0	1.0	-1	1.8	
4	1.0	13918	1	0.0	1.0	1.0	0	3.4	

Figure 18: Label Encoding

6 Feature Selection

Recursive Feature Estimator is used to select the features using Decision Tree classifier and running a pipeline to find the best features. The Repeated Stratified K-Fold cross-validation is used to check to validate the features selected. Figure 19 below, shows the implementation of this process.

```
liver_corr = data.corr().abs()
liver_corr
```

	Drug	Age	Sex	Ascites	Hepatomegaly	S
Drug	1.000000	0.152435	0.023498	0.022043	0.105890	0.[illegible]
Age	0.152435	1.000000	0.167506	0.186908	0.105552	0.[illegible]
Sex	0.023498	0.167506	1.000000	0.014659	0.020082	0.[illegible]
Ascites	0.022043	0.186908	0.014659	1.000000	0.082779	0.[illegible]
Hepatomegaly	0.105890	0.105552	0.020082	0.082779	1.000000	0.[illegible]
Spiders	0.137920	0.073749	0.106730	0.194552	0.124224	1.[illegible]
Edema	0.076846	0.043956	0.011353	0.303015	0.062644	0.[illegible]
Bilirubin	0.073527	0.006288	0.028275	0.334812	0.237078	0.[illegible]
Cholesterol	0.016932	0.131378	0.009995	0.053662	0.117976	0.[illegible]
Albumin	0.043349	0.182836	0.028522	0.320399	0.268351	0.[illegible]
Copper	0.001683	0.053460	0.216280	0.220451	0.208767	0.[illegible]

Figure 19: Feature selection

```
upper_tri = liver_corr.where(np.triu(np.ones(liver_corr.shape),k=1).astype(np.bool))
upper_tri
```

Figure 20: Find the feature with high correlation

```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.8)]
data = data.drop(to_drop, axis=1)
data.head()
```

Figure 21: Dropping Features with high correlation

7 Class Balancing

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

```
3    155
4    144
2     92
1     21
Name: Stage, dtype: int64
```

```
smote = SMOTE(k_neighbors = 3)
data, data['Stage'] = smote.fit_resample(data, data['Stage'])
```

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

```
4    155
3    155
2    155
1    155
Name: Stage, dtype: int64
```

Figure 22: SMOTE Class Balancing

```
: train, test = train_test_split(data, test_size=0.2, random_state=42)
```

```
: #pd.DataFrame(train).to_csv('/content/drive/MyDrive/Thesis_Code/train.csv')
train = pd.read_csv('/content/drive/MyDrive/Thesis_Code/train.csv',index_col=0)
train
```

Figure 23: Splitting training and testing set (in 80:20 ratio)

```
#pd.DataFrame(test).to_csv('/content/drive/MyDrive/Thesis_Code/test.csv')
test = pd.read_csv('/content/drive/MyDrive/Thesis_Code/test.csv',index_col=0)
test
```

	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albu
618	0.152718	22874	0	0.0	0.847282	0.0	0	0.822174	208.991405	3.55
1	0.000000	20617	1	0.0	1.000000	1.0	0	1.100000	302.000000	4.14
424	0.265687	14518	1	0.0	0.000000	0.0	0	1.067157	223.054823	3.60
304	1.000000	15730	1	0.0	1.000000	1.0	0	2.900000	426.000000	3.61

```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.8)]
data = data.drop(to_drop, axis=1)
data.head()
```

Figure 24: Splitting training and testing set (80:20)

```
X_train = train.drop('Stage', axis=1)
X_test= test.drop('Stage', axis=1)
y_train=train['Stage']
y_test =test['Stage']
X_train = np.asarray(X_train).astype(np.int64)
X_test = np.asarray(X_test).astype(np.int64)
y_train = np.asarray(y_train).astype(np.int64)
y_test = np.asarray(y_test).astype(np.int64)
print (X_train.shape)
print (y_train.shape)
print (X_test.shape)
print (y_test.shape)
```

Figure 25: Feature and target set in training and testing data

```
scaler = MinMaxScaler(feature_range=(0, 1))
X_train= scaler.fit_transform(X_train)
X_test= scaler.fit_transform(X_test)
```

```
X_train.shape, X_test.shape
```

```
((496, 16), (120, 16))
```

Figure 26: Data Scaling

```
X_train.shape, X_test.shape
```

```
((496, 16), (120, 16))
```

```
xtrain=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
xtest =X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

Figure 27: Reshaping for neural network models

8 Deep Learning Models

8.1 RNN

```
rnn = Sequential()

rnn.add(Dense(32, activation='relu', input_shape=(16,)))
rnn.add(Dense(5, activation='sigmoid'))

rnn.summary()
rnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                544

 dense_1 (Dense)             (None, 5)                 165

=================================================================
Total params: 709
Trainable params: 709
Non-trainable params: 0
_________________________________________________________________
```

Figure 28: Implementation of RNN

```
rnn.fit(xtrain, y_train, validation_data=(xtest, y_test), epochs=200)
```

```
333
Epoch 3/200
16/16 [==============================] - 0s 5ms/step - loss: 1.4576 - accuracy: 0.2661 - val_loss: 1.4204 - val_accuracy: 0.3
083
Epoch 4/200
16/16 [==============================] - 0s 4ms/step - loss: 1.4077 - accuracy: 0.3992 - val_loss: 1.3672 - val_accuracy: 0.4
500
Epoch 5/200
16/16 [==============================] - 0s 5ms/step - loss: 1.3649 - accuracy: 0.4778 - val_loss: 1.3219 - val_accuracy: 0.4
917
Epoch 6/200
16/16 [==============================] - 0s 6ms/step - loss: 1.3276 - accuracy: 0.4899 - val_loss: 1.2871 - val_accuracy: 0.4
```

Figure 29: Implementation of RNN Model training

```
json = open('/content/drive/MyDrive/Thesis_Code/rnnmodel.json')

rnnjson = json.read()
json.close()

rnn = model_from_json(rnnjson)
## load weights into new model
rnn.load_weights("/content/drive/MyDrive/Thesis_Code/rnn.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
rnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
Loaded model from disk
```

```
#xtest= np.reshape(X_test, (X_test[0], X_test[1]))
rnn_predicted= np.argmax(rnn.predict(xtest), axis=1)
```

Figure 30: Loading saved model and making predictions

```
loss, rnn_score  = rnn.evaluate(xtrain, y_train)
loss, rnn_score_test = rnn.evaluate(xtest, y_test)
rnn_score_test
```

```
16/16 [==============================] - 0s 2ms/step - loss: 0.8675 - accuracy: 0.6250
4/4 [==============================] - 0s 3ms/step - loss: 0.8737 - accuracy: 0.6417

0.6416666507720947
```

```
tn, fp, fn, tp = confusion_matrix(y_test, rnn_predicted)
rnnspecificity = (tn / (tn+fp)).max()
rnnsensitivity = (tp / (tp + fn)).max()
rnnbalancedAccuracy = np.round(((rnnsensitivity + rnnspecificity) / 2)*100 ,2)

print('specificity: \n', rnnspecificity)
print('sensitivity: \n', rnnsensitivity)
print('Balanced Accuracy: \n', rnnbalancedAccuracy)

print(confusion_matrix(y_test,rnn_predicted))
sns.heatmap(confusion_matrix(y_test,rnn_predicted),annot=True,fmt="d")

print(classification_report(y_test,rnn_predicted))
```

Figure 31: Evaluating Model Performance

8.2 DRNN

```
drnn = Sequential()

drnn.add(Dense(128, activation='relu', input_shape=(16,1)))
drnn.add(Flatten())
drnn.add(Dense(32, activation='relu'))
drnn.add(Dense(5, activation='sigmoid'))

drnn.summary()
drnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 16, 128)           256

 flatten (Flatten)           (None, 2048)              0

 dense_3 (Dense)             (None, 32)                65568

 dense_4 (Dense)             (None, 5)                 165

=================================================================
Total params: 65,989
Trainable params: 65,989
Non-trainable params: 0
```

Figure 32: Implementation of DRNN

```
drnn.fit(xtrain, y_train, validation_data=(xtest, y_test), epochs=500)
```

```
667
Epoch 3/500
16/16 [==============================] - 0s 6ms/step - loss: 1.1490 - accuracy: 0.5161 - val_loss: 1.0528 - val_accuracy: 0.5
333
Epoch 4/500
16/16 [==============================] - 0s 7ms/step - loss: 1.0773 - accuracy: 0.5504 - val_loss: 0.9987 - val_accuracy: 0.5
667
Epoch 5/500
16/16 [==============================] - 0s 6ms/step - loss: 1.0543 - accuracy: 0.5343 - val_loss: 1.0086 - val_accuracy: 0.5
417
Epoch 6/500
```

Figure 33: Implementation of DRNN Model training

```
json = open('/content/drive/MyDrive/Thesis_Code/drnnmodel.json')

drnnjson = json.read()
json.close()

rnn = model_from_json(drnnjson)# Load weights into new model
drnn.load_weights("/content/drive/MyDrive/Thesis_Code/drnn.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
drnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
Loaded model from disk
```

Figure 34: Loading saved model and

```
drnn_predicted= np.argmax(drnn.predict(X_test), axis=1)
loss, drnn_score = drnn.evaluate(X_train, y_train)
loss, drnn_score_test = drnn.evaluate(X_test, y_test)
print('DRNN Train Score: \n', drnn_score)
print('DRNN Test Score: \n', drnn_score_test)
```

```
16/16 [==============================] - 0s 3ms/step - loss: 0.4576 - accuracy: 0.808
4/4 [==============================] - 0s 5ms/step - loss: 0.6746 - accuracy: 0.7333
DRNN Train Score:
 0.8084677457809448
DRNN Test Score:
 0.7333333492279053
```

```
tn, fp, fn, tp = confusion_matrix(y_test, drnn_predicted)
drnnspecificity = (tn / (tn+fp)).max()
drnnsensitivity = (tp / (tp + fn)).max()
drnnbalancedAccuracy = np.round(((drnnsensitivity + drnnspecificity) / 2)*100 ,2)
drnn_score_test = accuracy_score(y_test,drnn_predicted)
print('specificity: \n', drnnspecificity)
print('sensitivity: \n', drnnsensitivity)
print('Balanced Accuracy: \n', drnnbalancedAccuracy)
sns.heatmap(confusion_matrix(y_test,drnn_predicted),annot=True,fmt="d")
print(classification_report(y_test,drnn_predicted))
```

Figure 31: Evaluating Model Performance

8.3 Random Forest Trees

```
with open('/content/drive/MyDrive/Thesis_Code/random_forest.pkl', 'rb') as f:
    random_forest= joblib.load(f)
random_forest = RandomForestClassifier(min_samples_leaf=5)
```

```
random_forest = RandomForestClassifier(min_samples_leaf=5)
```

```
random_forest.fit(X_train, y_train)
# Predict Output
rf_predicted = random_forest.predict(X_test)
random_forest_score = round(random_forest.score(X_train, y_train) * 100, 2)
rf_score_test = round(random_forest.score(X_test, y_test) * 100, 2)
print('Random Forest Train Score: \n', random_forest_score)
print('Random Forest Test Score: \n', rf_score_test)
tn, fp, fn, tp = confusion_matrix(y_test, rf_predicted)
rfspecificity = (tn / (tn+fp)).max()
rfsensitivity = (tp / (tp + fn)).max()
rfbalancedAccuracy = np.round(((rfsensitivity + rfspecificity) / 2)*100 ,2)
print('specificity: \n', rfspecificity)
print('sensitivity: \n', rfsensitivity)
print('Balanced Accuracy: \n', rfbalancedAccuracy)
print(classification_report(y_test,rf_predicted))
sns.heatmap(confusion_matrix(y_test, rf_predicted),annot=True,fmt="d")
```

Figure 32: Implementation of Random Forest Trees

8.4 SVM

```
with open('/content/drive/MyDrive/Thesis_Code/svc.pkl', 'rb') as f:
    svc= joblib.load(f)
```

```
svc = SVC(kernel='linear',C=10, gamma =9)
```

```
svc.fit(X_train, y_train)
# Predict Output
svc_predicted = svc.predict(X_test)
svc_score = round(svc.score(X_train, y_train) * 100, 2)
svc_score_test = round(svc.score(X_test, y_test) * 100, 2)
print('SVC Train Score: \n', svc_score)
print('SVC Test Score: \n', svc_score_test)
tn, fp, fn, tp = confusion_matrix(y_test, svc_predicted)
svcspecificity = (tn / (tn+fp)).max()
svcsensitivity = (tp / (tp + fn)).max()
svmbalancedAccuracy = np.round(((svcsensitivity + svcspecificity) / 2)*100 ,2
print('specificity: \n', svcspecificity)
print('sensitivity: \n', svcsensitivity)
print('Balanced Accuracy: \n', svmbalancedAccuracy)
sns.heatmap(confusion_matrix(y_test, svc_predicted),annot=True,fmt="d")
print(classification_report(y_test,svc_predicted))
```

Figure 33: Implementation of SVM

9 Model result

This section explains the performance of the models.

9.1 Model Scores

```
result = pd.DataFrame({
    'Model': [ 'Recurrent Neural Network', 'Deep Recurrent Neural Network', 'Support Vector Machine','Random Forest'],
    'Train Score': [ np.round(rnn_score*100,2), np.round(drnn_score*100,2), svc_score, random_forest_score],
    'Accuracy Score': [ np.round(rnn_score_test*100,2), np.round(drnn_score_test*100,2), svc_score_test, rf_score_test]
    'Sensitivity': [ rnnsensitivity, drnnsensitivity, svcsensitivity, rfsensitivity],
    'Specificity': [ rnnspecificity, drnnspecificity, svcspecificity, rfspecificity],
    'Balanced Accuracy Score': [ rnnbalancedAccuracy, drnnbalancedAccuracy, svmbalancedAccuracy, rfbalancedAccuracy]})
result.sort_values(by='Balanced Accuracy Score', ascending=False)
```

	Model	Train Score	Accuracy Score	Sensitivity	Specificity	Balanced Accuracy Score
1	Deep Recurrent Neural Network	80.85	73.33	0.935484	1.000000	96.77
3	Random Forest	85.28	68.33	0.885714	1.000000	94.29
0	Recurrent Neural Network	62.50	64.17	0.823529	0.862069	84.28
2	Support Vector Machine	57.06	57.50	0.781250	0.694444	73.78

Figure 34: Model Performance

9.2 Model Accuarcy

```
: plt.figure(figsize=(10,10))
  plt.plot(result['Model'], result['Train Score'], label="Training Accuracy")
  plt.plot(result['Model'], result['Accuracy Score'], label="Testing Accuracy")
  plt.legend()
```

```
: <matplotlib.legend.Legend at 0x7fb10756e690>
```

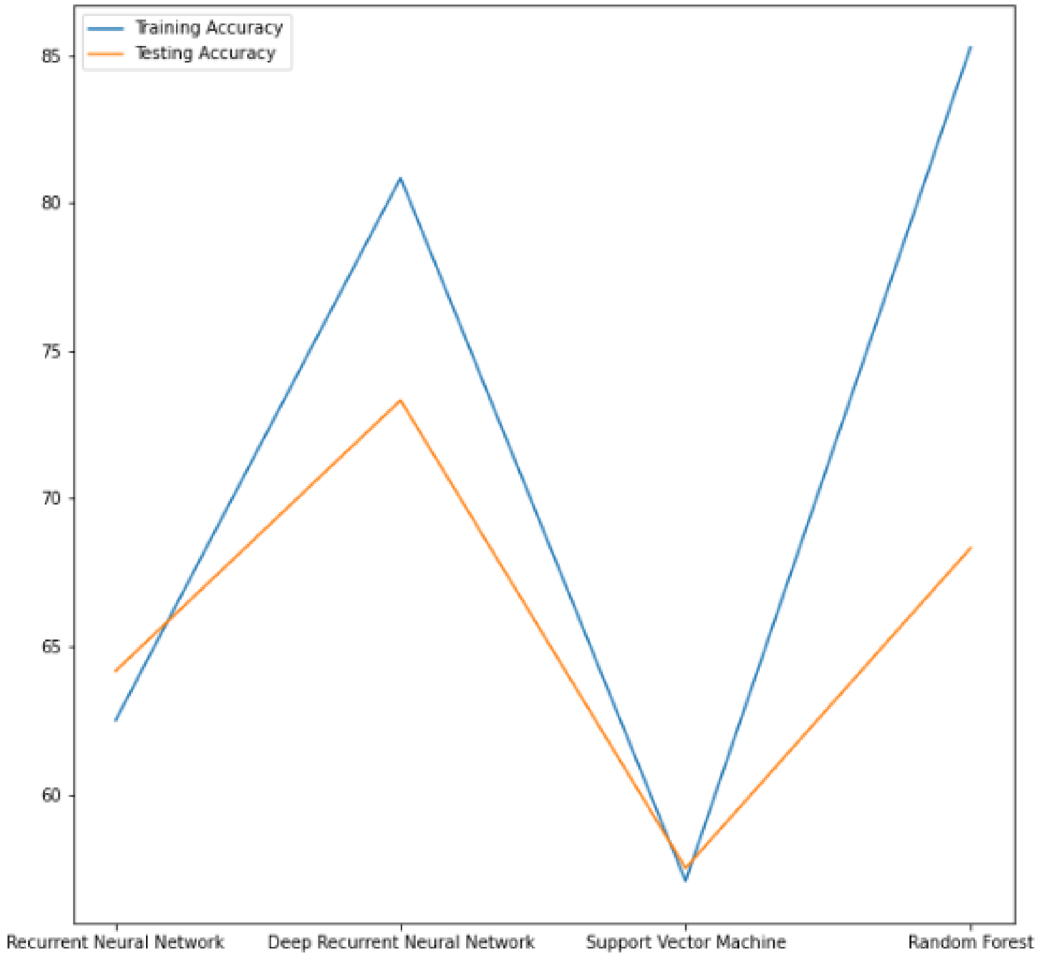

Figure 35: Model Accuracy

9.3 Model Predictions

```
stagePrediction = pd.DataFrame({'Actual': y_test,
    'Recurrent Neural Network': rnn_predicted,
    'Deep Recurrent Neural Network': drnn_predicted,
    'Support Vector Machine': svc_predicted,
    'Random Forest': rf_predicted})
stagePrediction
```

	Actual	Recurrent Neural Network	Deep Recurrent Neural Network	Support Vector Machine	Random Forest
0	4	1	3	1	1
1	3	4	3	3	4
2	1	1	1	1	1
3	3	4	3	3	3
4	2	2	2	2	2
...	...	...	...	...	...
115	2	2	2	2	2
116	4	2	4	1	2
117	2	3	4	3	2
118	2	2	2	2	2
119	4	4	4	4	4

120 rows × 5 columns

Figure 36: Model Predictions

References

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://scikit-learn.org/stable/modules/svm.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html