Configuration Manual

MSc Research Project
Data Analytics

Sakshi Dubey
Student ID: X19201290

School of Computing
National College of Ireland

Supervisor: DR. Bharathi Chakravarthi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Sakshi Dubey……………………………………………………………………………………

Student ID: X19201290……………………………………………………………………………………..……

Programme: Data Analytics …………………………………………… **Year:** 2021-2022

Module: MS Research Project ………………………………………………………………………………………….………

Lecturer: Dr. Bharathi Chakravarthi……………………………………………….………

Submission Due Date: 16/10/2021………………………………………………………………………………….………

Project Title: A Comparative study of Breast cancer diagnosis and classification using neural networks and machine learning models

Word Count: ……………………………………… 1685 **Page Count:** …………………………………….…….……… 14 pages

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: …… 29/01/2022

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

SAKSHI DUBEY
X19201290

1 Introduction

The configuration manual summarizes the implementation of scripts for the present research topic. The manual is documented with detailed steps and resources to ensure smooth execution of code without errors. The manual provides information about the hardware and software configuration as well that helps in running scripts. The following steps will assist in execution of our project.

2 System Configuration

2.1 Hardware Configuration

Device name: LAPTOP-SSH9LG1B
Processor: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Installed Ram: 16.0 GB (15.8 GB usable)
System type: 64-bit operating system, x64-based processor

2.2 Software Configuration

We have implemented our project using python based Jupyter notebook IDE which is available in the anaconda package. We have illustrated the steps to execute the developed scripts below.

3 Downloads and Installation

- **Python**

We have implemented our project using python. Python strongly supports machine learning and deep learning models with different tools, modules, libraries, features that assist the pre-processing stage of the model and optimizes the model's performance. Hence, it is essential to download the latest version of python and have it installed in your environment to ensure smooth execution of scripts. The latest version of python can be downloaded from the python website. One can select and download the software installer of desired version based on the operation system of the device. The confirmation of successful installation can be obtained by checking ‘python vision’ query in windows command prompt that will update you with recent installation. Figure 1 shows the download page of python.

Figure 1: Download page of python

- **Anaconda**

Our next step is installation of Anaconda Navigator package. Anaconda package is a python based integrated development environment that can be used for checking results and developing code for your scripts. Jupyter Notebook and Spyder are one of the highly used Integrated development environments in anaconda navigator package. One can download the anaconda navigator package from the official website of operating systems. After successful download and installation of anaconda navigator package, we can observe different integrated development environment that can be selected as per the requirement of development. We have used the jupyter integrated development environment for our research project.

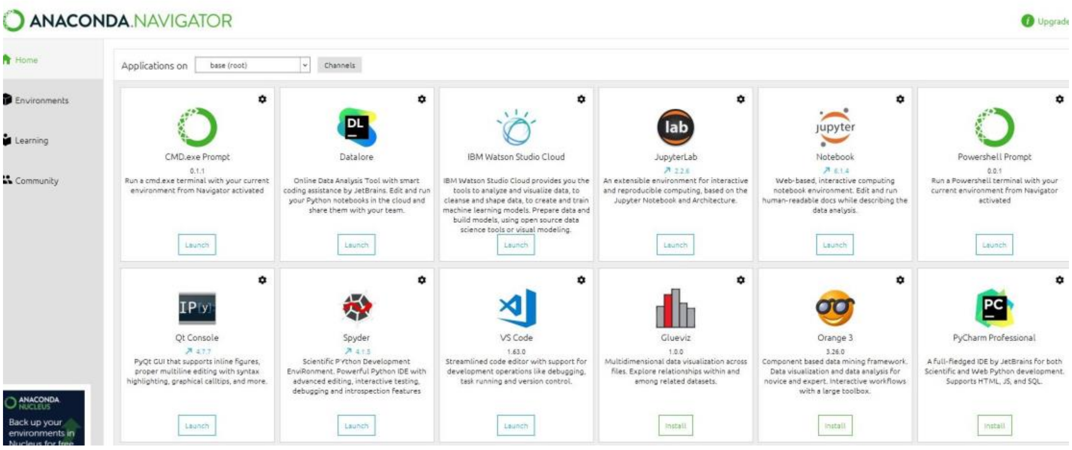


Figure 2: Anaconda Navigator

- **Data source information**

In this study, we have used a public data source from Kaggle that contains information of breast cancer data of patients of Wisconsin hospital containing cancerous and non-cancerous breast mass. We have used this data in our classification, visualizations and several deep learning and machine learning models that included random forest classifier, k nearest neighbors, decision tree classifier and support vector machine.

4 Project Development

In the project development section, we need to create a new python 3 notebook with suitable title to start our program execution. The file format is .ipynb and as we begin with implementation of our machine learning and deep learning models, we need to install additional libraries to support our program execution. Essential libraries can be installed in the command prompt/jupyter notebook by using pip command.

For breast cancer classification, we need to install specific libraries to assist in executing our models. The libraries used in the initial stages are :-

- Scikit-learn
- Matplotlib
- Pandas
- Numpy
- Sklearn
- Plotly.express
- Seaborn

We have also installed a new tenser flow module ‘keras’ to execute the neural network models. We can install ‘keras’ package in two steps in our command prompt.

- Command prompt:1) pip install tensorFlow==1.8
  2)pip install keras

In the last stage of our coding, we can launch our script in the jupyter notebook command or running every block of code in the cell. The errors in any of the steps will be displayed in the output window. Once we begin with execution of our code, it is required to convert and fetch data from the data frame.

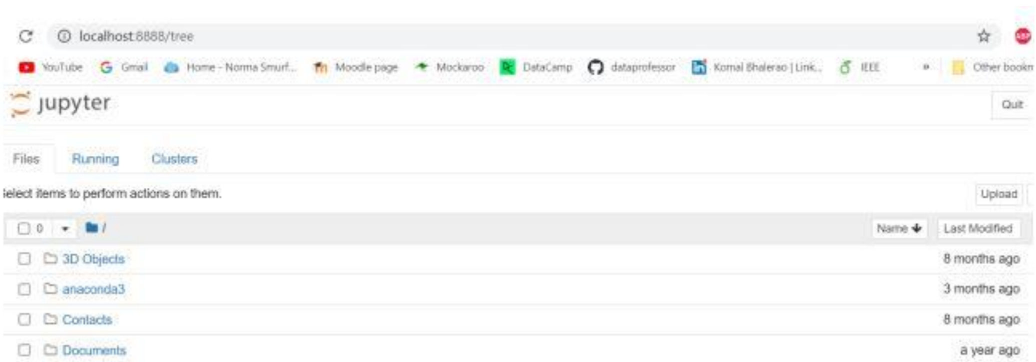


Figure 3: Jupyter notebook homepage

5 Data Preparation

5.1 Data pre-processing

Data preprocessing plays a significant role and getting rid of inconsistencies, noise, missing values and incorrect entries in the dataset and makes it easier for interpretation and evaluation. We looked for such noise and inconsistencies in our Wisconsin dataset and found columns that contained incorrect data, outliers and null values. The insignificant data entries were dropped, outliers were removed and further the data was pushed for data exploration data visualization process.

```
df.head()
```

✓ 0.6s Python

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	peri
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	

5 rows × 31 columns

Figure 4

Let us remove Id column and last column named unnamed: 32 since they are not useful for our classification

```
df.drop(['id','Unnamed: 32'],axis = 1,inplace = True)
df
```

✓ 0.6s Python

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	pe
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380	17.33	
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990	23.41	
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570	25.53	
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910	26.50	
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450	26.40	
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690	38.25	
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980	34.12	
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740	39.42	
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456	30.37	

Figure 5

There are no null values in any of the column..!!!

```
df.duplicated().sum()
```

✓ 0.6s

0

Figure 6

Figure 4 shows the initial stage of program execution
In figure 5, we have performed data cleaning on our dataset. The ID column and last column with maximum null values have been dropped
In figure 6, we can see no null values in the columns and data is ready for next stage.

5.2 Data Exploration

In our exploratory data analysis and data visualization, we have installed a new module "Autoviz" to conduct analysis and visualization of dependent and independent variables.On can install the "Autoviz" package by executing few steps in the command prompt.

- Command prompt: pip install Autoviz

```python
from autoviz.AutoViz_Class import AutoViz_Class

AV = AutoViz_Class()
dftc = AV.AutoViz(
    filename='',
    sep='' ,
    depVar='diagnosis',
    dfte=df,
    header=0,
    verbose=1,
    lowess=False,
    chart_format='png',
    max_rows_analyzed=300000,
    max_cols_analyzed=30
)
```

Figure 7: Data visualization using Autoviz

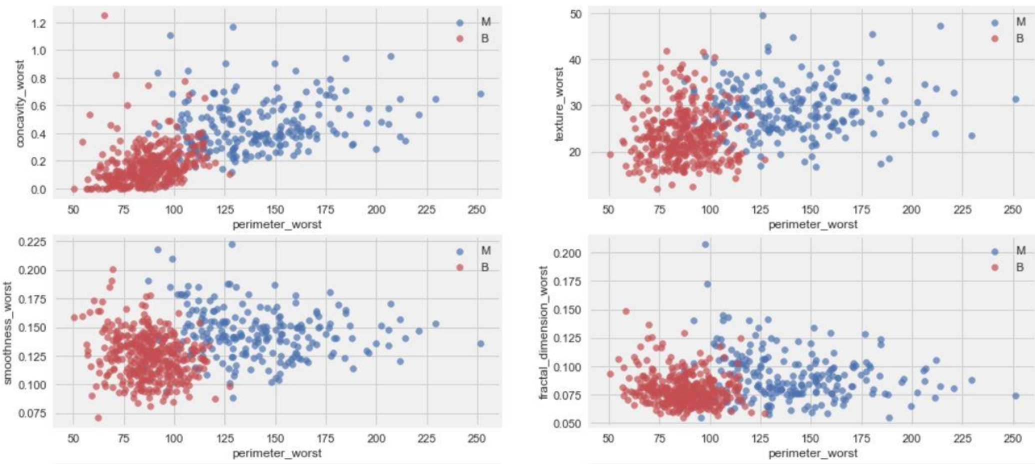


Figure 8: Pairwise scatterplot of variables.

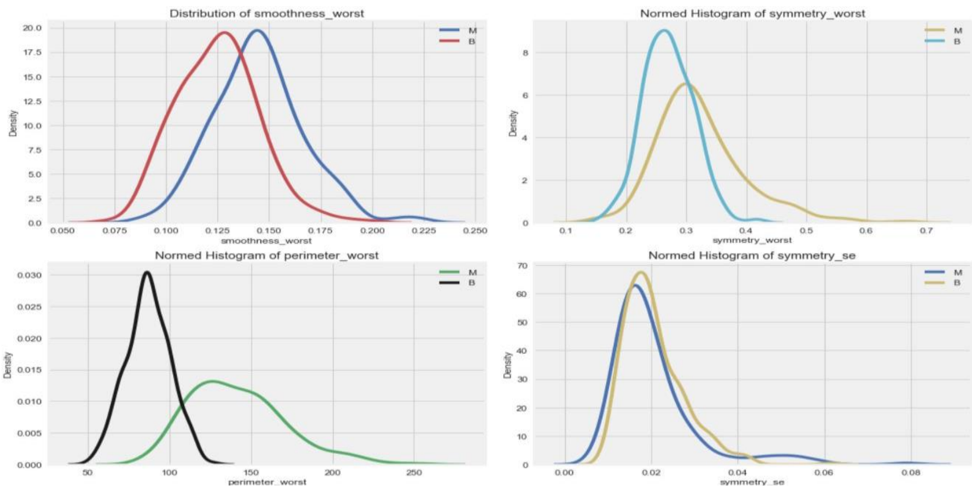


Figure 9: Pairwise distribution of normed histogram of variables

In Figure 8 we can observe pairwise scatter plot of continuous variables with respect to target variable diagnosis that has two classes ‘malignant’ and ‘benign’, while in Figure 9. We can see pairwise distribution of normed histogram of continuous variables.

```
''' Plot a Shifted Correlation Matrix '''
# Diagonal correlation is always unity & less relevant, shifted variant shows
# only relevant cases
def corrMat(df,id=False):

    corr_mat = df.corr().round(2)
    f, ax = plt.subplots(figsize=(12,7))
    mask = np.triu(np.ones_like(corr_mat, dtype=bool))
    mask = mask[1:,:-1]
    corr = corr_mat.iloc[1:,:-1].copy()
    sns.heatmap(corr,mask=mask,vmin=-0.3,vmax=0.3,center=0,
                cmap='RdPu_r',square=False,lw=2,annot=True,cbar=False)
#     bottom, top = ax.get_ylim()
#     ax.set_ylim(bottom + 0.5, top - 0.5)
    ax.set_title('Shifted Linear Correlation Matrix')

corrMat(df.drop(['diagnosis'],axis = 1))
```

Figure 10: Shifted correlation matrix. In this stage, we have determined variables that can be successful contributors for model building.

```
fig = px.histogram(data_frame = df,
                   x = "radius_mean",
                   color="diagnosis", title="<b>diagnosis vs radius mean</b>",
                   pattern_shape_sequence=['x'],
                   template='plotly_dark')

fig1 = px.histogram(data_frame = df,
                   x = "texture_mean",
                   color="diagnosis", title="<b>diagnosis vs texture mean</b>",
                   pattern_shape_sequence=['x'],
                   template='plotly_dark')

fig.update_layout(bargap=0.2)
fig1.update_layout(bargap = 0.2)

fig.show()
fig1.show()
```

Figure 11: Analysis of radius mean and texture mean. In this stage we have conducted analysis of target variable “diagnosis” with two highly correlated variables radius mean and texture mean. The visualizations of this comparative analysis is present in below figures.

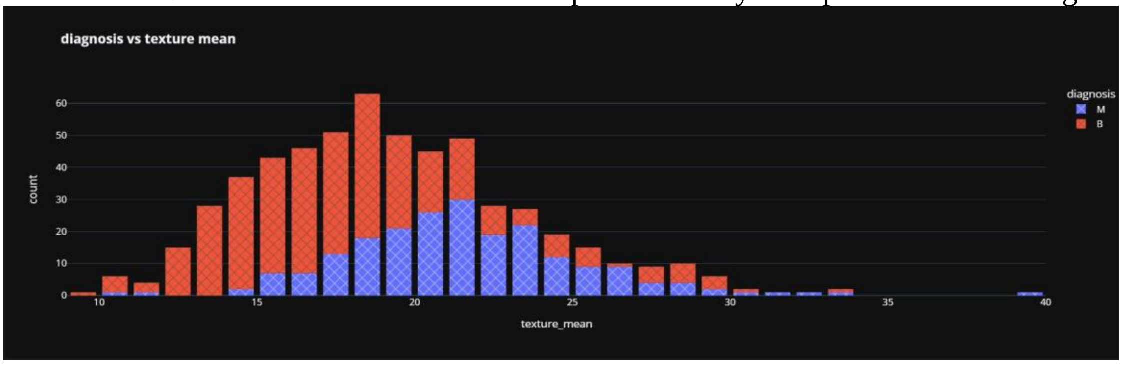


Figure 12

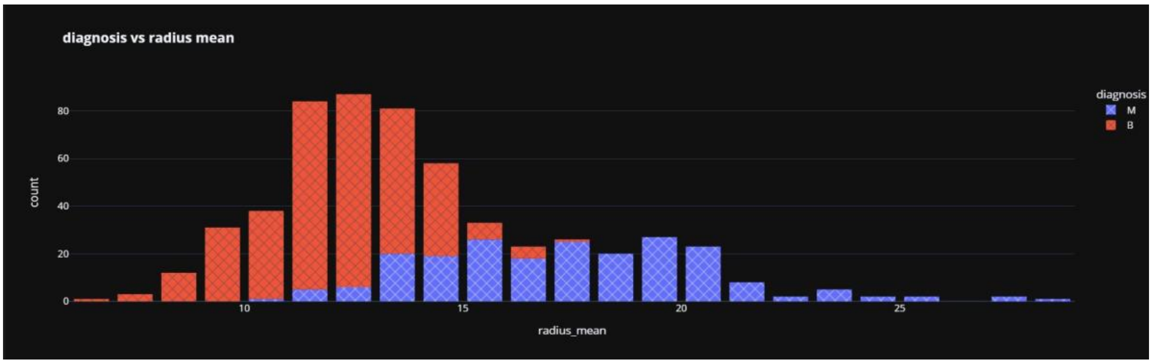


Figure 13

Group by analysis of both the tumors

```
df1 = df.groupby('diagnosis').agg({'radius_mean' : 'mean', 'texture_mean' : 'mean', 'perimeter_mean' : 'mean',
        'area_mean' : 'mean', 'smoothness_mean' : 'mean', 'compactness_mean' : 'mean', 'concavity_mean' :'mean',
        'concave points_mean' : 'mean', 'symmetry_mean' : 'mean', 'fractal_dimension_mean' : 'mean',
        'radius_se' : 'mean', 'texture_se' : 'mean', 'perimeter_se' : 'mean', 'area_se' : 'mean', 'smoothness_se' : 'mean',
        'compactness_se' : 'mean', 'concavity_se' : 'mean', 'concave points_se' : 'mean', 'symmetry_se' : 'mean',
        'fractal_dimension_se' : 'mean', 'radius_worst' : 'mean', 'texture_worst' : 'mean',
        'perimeter_worst' : 'mean', 'area_worst' : 'mean', 'smoothness_worst' : 'mean',
        'compactness_worst' : 'mean', 'concavity_worst' : 'mean', 'concave points_worst' : 'mean',
        'symmetry_worst' : 'mean', 'fractal_dimension_worst': 'mean'})
df1
```

Figure 14 : In this stage, we have conducted Groupy analysis of both tumours to understand the relationship between dependent and independent variables and also the highly co related variables. The visualization of group by analysis of both tumours is given below figure 15.

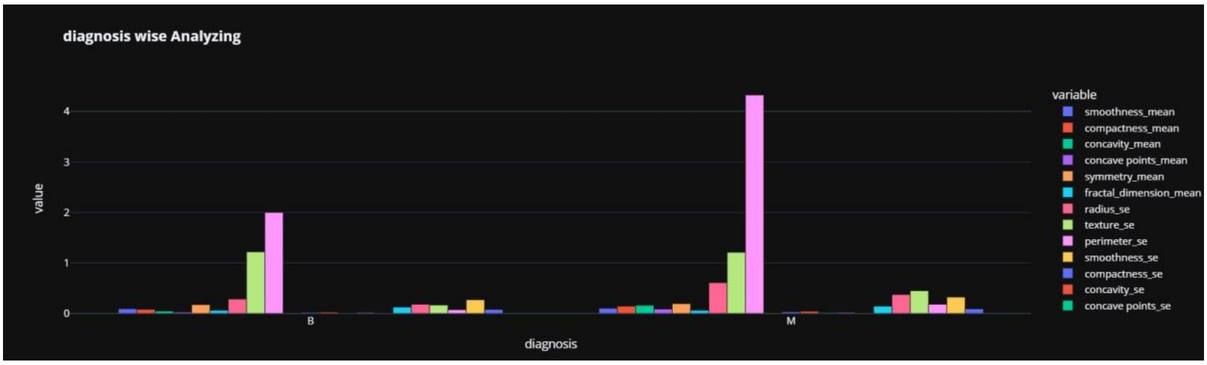
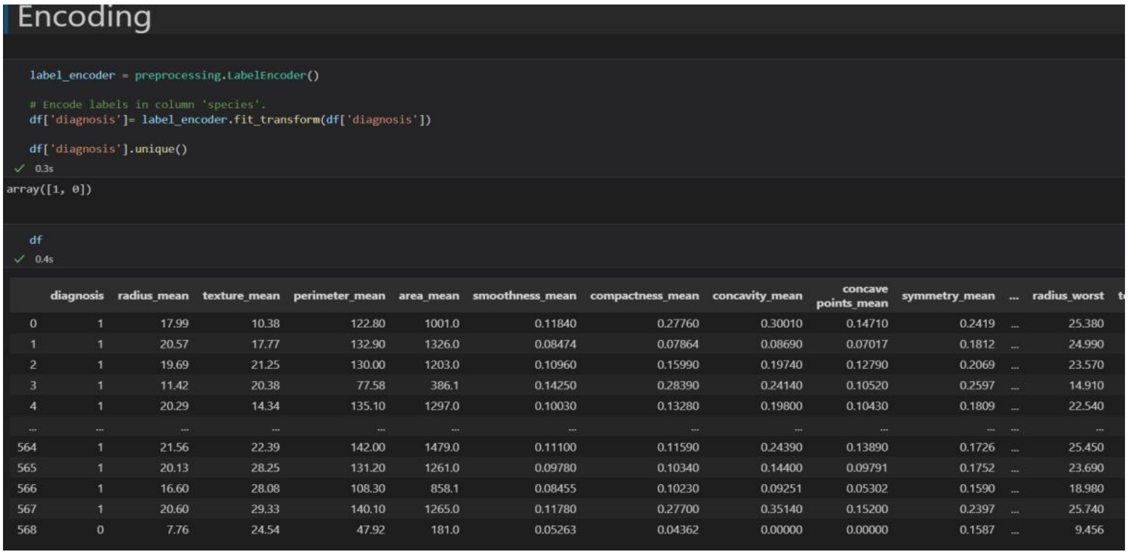


Figure 15 : Group by analysis of both tumours

5.3 Encoding

In figures 16 and 17 we can observe feature scaling and encoding. We have implemented feature scaling on our target variable ‘diagnosis ‘using standard scalar() function. We have encoded our target variable to numeric data type using a Label encoder. We have transformed our target variable “diagnosis” from categorial to integer type i.e., from B and M to a factor binary number 0 and 1 for benign and malignant tumor. Also, for the artificial neural network model, the factor was transformed to numeric data type.



Encoding

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['diagnosis']= label_encoder.fit_transform(df['diagnosis'])

df['diagnosis'].unique()
```

0.3s

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

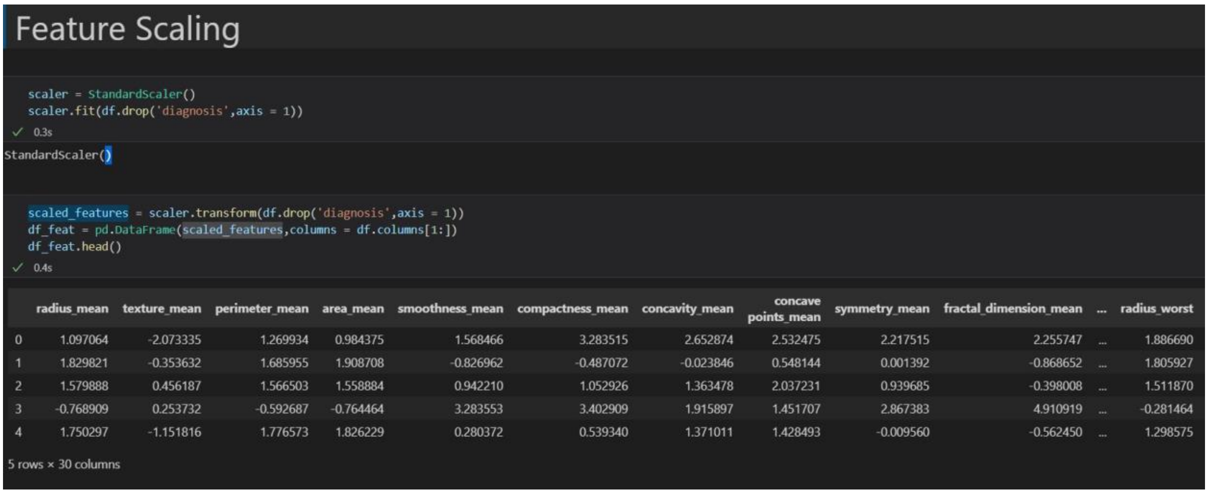
```
df
```

0.4s

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540
...	...	...	...	...	...	...	...	...	...	...	...	...
564	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450
565	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690
566	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980
567	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740
568	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456

Figure 16: Encoding

5.4 Feature scaling



Feature Scaling

```
scaler = StandardScaler()
scaler.fit(df.drop('diagnosis',axis = 1))
```

0.3s

```
StandardScaler()
```

```
scaled_features = scaler.transform(df.drop('diagnosis',axis = 1))
df_feat = pd.DataFrame(scaled_features,columns = df.columns[1:])
df_feat.head()
```

0.4s

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	...	1.886690
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652	...	1.805927
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008	...	1.511870
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	...	-0.281464
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	...	1.298575

5 rows × 30 columns

Figure 17: Feature scaling

6 Model building

The stage of model building is the most crucial part as it involves selecting significant algorithms as per the data and that meet the objectives of our research. We have implemented nine machine learning models and one neural network model on our dataset. The classification algorithms include K-NN, SVC, LR, random forest, decision tree classifier, hyper parameter tuning, ada boost classifier, gradient boosting classifier, XGB boost classifier and artificial neural network model was implemented in this research. We have used python programming language to carry out our analysis. The implementation and performance results of all the classification models have been provided in below Figures.

6.1 Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import keras
import plotly.express as px
import plotly.graph_objects as go
import plotly.offline as pyo
import plotly.figure_factory as ff
from plotly import tools
from plotly.subplots import make_subplots
from plotly.offline import iplot
from sklearn import preprocessing
from category_encoders import *
from sklearn.preprocessing import LabelEncoder
%matplotlib inline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import datasets, linear_model, metrics
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

Figure 18: Importing libraries

6.2 K Nearest Neighbours

1.Prediction of Breast Cancer using KNN

```
knn = KNeighborsClassifier(n_neighbors = 9)
knn.fit(X_train,y_train)
 0.4s
KNeighborsClassifier(n_neighbors=9)

pred = knn.predict(X_test)
pred
 0.6s
array([0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0])

print(confusion_matrix(y_test,pred))
 0.3s
[[103   2]
 [  6  60]]
```

Figure 19: K nearest Neighbours model

```
error_rate= []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
 0.3s

plt.figure(figsize = (10,6))
plt.plot(range(1,40),error_rate,color = 'blue',linestyle = '--',marker = 'o',markerfacecolor='red',markersize = 10)
plt.title('Error Rate vs K')
plt.xlabel('K')
plt.ylabel('Error Rate')
 0.2s
Text(0, 0.5, 'Error Rate')
```

Figure 20: Error rate: K nearest neighbours

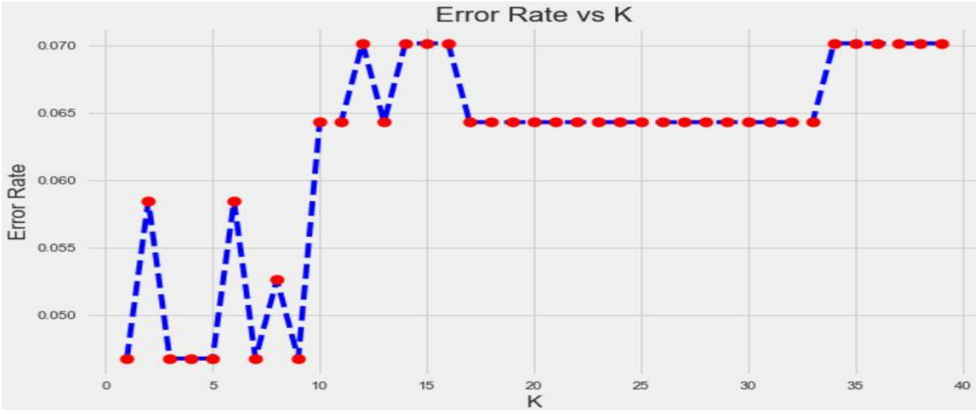


Figure 21: This block shows graphical visualization of error rate in KNN model

6.3 Random Forest classifier

```
forest= RandomForestClassifier(n_estimators =40, random_state = 0)
forest.fit(X_train,y_train)
y_pred = forest.predict(X_test)
forest.score(X_test,y_test)
✓ 0.1s
```

Figure 22

6.3 Logistic Regression

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
✓ 0.5s
LogisticRegression()

pred = logmodel.predict(X_test)
✓ 0.3s
```

Figure 23

6.4 Support vector machine

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
✓ 0.3s

confusion_matrix(y_test, y_pred)
✓ 0.3s
array([[104,   1],
       [  3,  63]], dtype=int64)
```

Figure 24

6.5 Decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)
✓ 0.4s

confusion_matrix(y_test, y_pred)
✓ 0.3s
array([[98,  7],
       [ 8, 58]], dtype=int64)
```

Figure 25

6.6 Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV
grid_params = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'min_samples_split' : range(2, 10, 1),
    'min_samples_leaf' : range(2, 10, 1)
}

grid_search = GridSearchCV(dtc, grid_params, cv = 5, n_jobs = -1, verbose = 1)
grid_search.fit(X_train, y_train)
✓ 6.5s
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, 10],
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(2, 10)},
             verbose=1)

dtc = grid_search.best_estimator_
y_pred = dtc.predict(X_test)
```

Figure 26

6.7 ADA boost classifier

```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(base_estimator = dtc)

parameters = {
    'n_estimators' : [50, 70, 90, 120, 180, 200],
    'learning_rate' : [0.001, 0.01, 0.1, 1, 10],
    'algorithm' : ['SAMME', 'SAMME.R']
}

grid_search = GridSearchCV(ada, parameters, n_jobs = -1, cv = 10, verbose = 1)
grid_search.fit(X_train, y_train)
```

Figure 27

6.8 Gradient boosting classifier

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()

parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

gbc = GridSearchCV(gb, parameters, cv = 5, n_jobs = -1, verbose = 1)
gbc.fit(X_train, y_train)
```

Figure 28

6.9 XG Boost classifier

```
from xgboost import XGBClassifier
xgb = XGBClassifier(booster = 'gblinear', learning_rate = 1, n_estimators = 10)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
✓ 0.9s
```

Figure 29

6.10 Deep learning model

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Importing data
df = pd.read_csv('data.csv')
del df['Unnamed: 32']

X = df.iloc[:, 2:].values
y = df.iloc[:, 1].values


# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
labelencoder_X_1 = LabelEncoder()
y = labelencoder_X_1.fit_transform(y)


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = Sequential()
```

Figure 30

```
# Adding the input layer and the first hidden layer
classifier.add(Dense(units=16, kernel_initializer='uniform', activation='relu', input_dim=30))
# Adding dropout to prevent overfitting
classifier.add(Dropout(rate=0.1))

# Adding the second hidden layer
classifier.add(Dense(units=16, kernel_initializer='uniform', activation='relu'))
# Adding dropout to prevent overfitting
classifier.add(Dropout(rate=0.1))

# Adding the output layer
classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))

# Compiling the ANN
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

#Optimizer is chosen as adam for gradient descent and Binary_crossentropy is the loss function used.

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size=75, epochs=75)
# Long scroll ahead but worth
# The batch size and number of epochs have been set using trial and error. Still looking for more efficient ways. Open to suggestions.
```

Figure 31

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       108
           1       0.97      0.95      0.96        63

    accuracy                           0.97       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171

[[106   2]
 [  3  60]]
Training Score:  96.0
The accuracy of the Deep Learning Model is:  97.07602339181285 %
```

Figure 32: This block generates the classification report of deep learning model that includes training score, accuracy and F1 score.

7 Evaluating Results

In this stage we have evaluated and analysed the performance of our classification models into two parts namely 1) Evaluating Accuracy of models ,2) Evaluating Sensitivity of models

```
# plotting bar chart
sns.set_style("whitegrid")
sns.set(rc={'figure.figsize':(15,10)})

ax = sns.barplot(x=['logmodel', 'dtc', 'forest', 'knn', 'svc','gbc','grid_search','xgb','ada','dl'], y=[logmodel_results, dtree_results, forest_results, knn_results, svc_results, gb
ax.set(xlabel='Models', ylabel='Accuracy', title='Classification Model Comparison')

a = int(logmodel_results)
b = int(dtree_results)
c = int(forest_results)
d = int(knn_results)
e = int(svc_results)
f = int(gbc_results)
g = int(grid_results)
h = int(xgb_results)
i = int(ada_results)
j = int(dl_results)
ax.text(0, logmodel_results, str(a)+'%')
ax.text(1, dtree_results, str(b)+'%')
ax.text(2, forest_results, str(c)+'%')
ax.text(3, knn_results, str(d)+'%')
ax.text(4, svc_results, str(e)+'%')
ax.text(5, gbc_results, str(f)+'%')
ax.text(6, grid_results, str(g)+'%')
ax.text(7, xgb_results, str(h)+'%')
ax.text(8, ada_results, str(i)+'%')
ax.text(8, dl_results, str(j)+'%')


plt.show()
```

Figure 33: Evaluating results

Figure 33 shows the block of code generates the evaluation metrics for comparison of classification models based on their accuracy percentage.

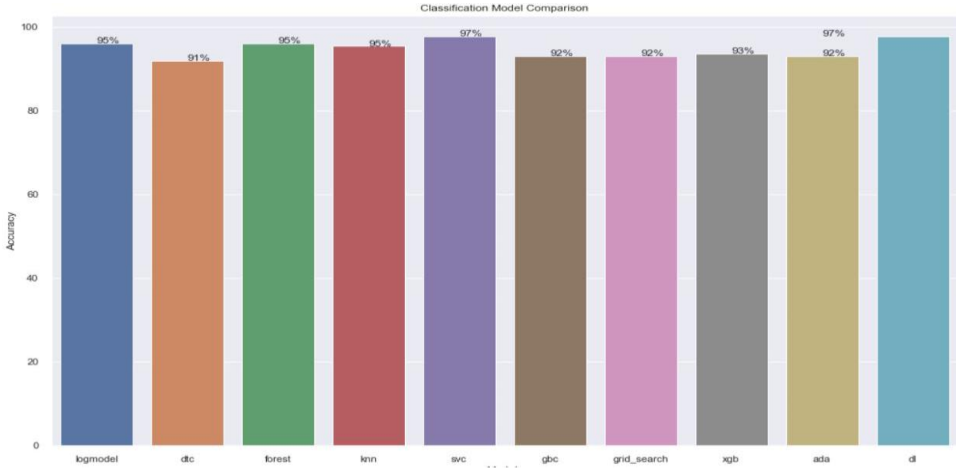


Figure 34: Evaluation of classification of models based on accuracy percentage

```
# plotting bar chart
sns.set_style("whitegrid")
sns.set(rc={'figure.figsize':(15,10)})

ax = sns.barplot(x=['logmodel', 'dtc', 'forest', 'knn', 'svc','gbc','grid_search','xgb','ada','dl'], y=[logmodel_senstivity, dtc_senstivity, rfc_senstivity, knn_senstivity, svc_sens
ax.set(xlabel='Models', ylabel='Senstivity', title='Classification Model Comparison')

a = int(logmodel_senstivity)
b = int(dtc_senstivity)
c = int(rfc_senstivity)
d = int(knn_senstivity)
e = int(svc_senstivity)
f = int(gbc_senstivity)
g = int(hp_senstivity)
h = int(xgb_senstivity)
i = int(ada_results)
j = int(dl_senstivity)
ax.text(0, logmodel_senstivity, str(a)+'%')
ax.text(1, dtc_senstivity, str(b)+'%')
ax.text(2, rfc_senstivity, str(c)+'%')
ax.text(3, knn_senstivity, str(d)+'%')
ax.text(4, svc_senstivity, str(e)+'%')
ax.text(5, gbc_senstivity, str(f)+'%')
ax.text(6, hp_senstivity, str(g)+'%')
ax.text(7, xgb_senstivity, str(h)+'%')
ax.text(8, ada_senstivity, str(i)+'%')
ax.text(8, dl_senstivity, str(j)+'%')


plt.show()
```

Figure 35: This block of code generates the evaluation metrics of classification models by taking into account their sensitivity percentage

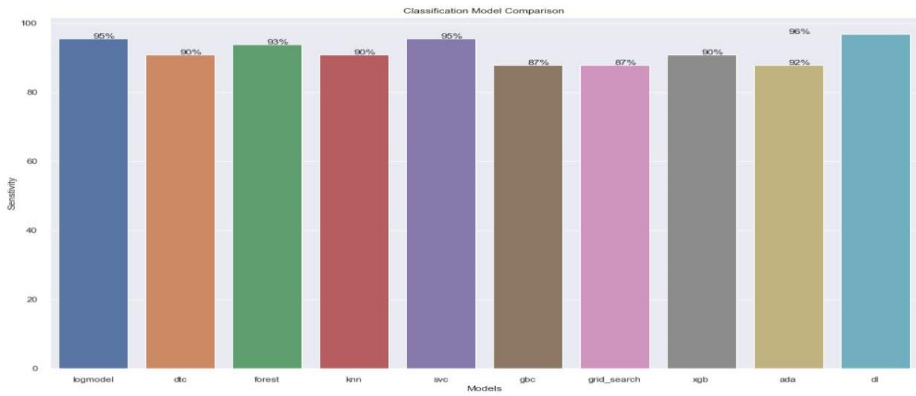


Figure 36 : Evaluation metrics