UNIVERSITY OF CANBERRA
AUSTRALIA'S CAPITAL UNIVERSITY

Faculty of Science and Technology

Assignment Coversheet

Student ID number & Student Name	u3175010 Zeguo Li
Unit name	Software Technology 1 G
Unit number	8995
Unit Tutor	Mr. Pranav Gupta Thursday 11:30 – 13:30
Assignment name	ST1 Capstone Project – Semester 1 2023
Due date	12/05/2023
Date submitted	11/05/2023

You must keep a photocopy or electronic copy of your assignment.

Student declaration

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

Signature of student: _____ **Date:** 11-May-2023

Table of Contents

Introduction

The purpose of this report is to demonstrate real world applicability of machine learning within the scope defined in ST1G Capstone project requirements. In this report, the Cancer Data from Kaggle [1] is used for data analysis and the training of machine leaner.

Cancer is one of the most fatal and incurable diseases in the world, despite being discovered as early as in the 19th Century, they are still as deadly as ever. Most of the patients survived due to an early discovery of the symptom, which makes an early diagnosis of the disease crucial to the patient's survival. As patient with early diagnosis are able to treat the disease earlier and increase the chance of recovery. Furthermore, an accurate classification of the tumours, such as benign tumour, can prevent the patient from undergoing treatment that are unnecessary. Since the treatment itself can be detrimental to the patients as well, such as chemotherapy. Therefore, correct diagnosis of the disease led by accurate classification of malignant and benign cells will be an interesting topic to investigate and the result of the research will contribute to the field of medicine.

Machine Learning, as an artificial intelligence technique, has advantage in classification by nature. The technique will be suitable for providing a solution to this problem, by classifying tumour cells based on their critical features, such as radius, concave point, smoothness and more.

The rest of the report well demonstrate the implementation of a machine learning model and a web application. The dataset [1] will be imported using Python in Visual Studio Code IDE. Followed by exploratory data analysis and data visualisation. Issues found in Exploratory data analysis will be solved by data pre-processing. And lastly, predictive analysis and web application will be implemented using python machine learning and web development libraries and packages such as Scikit-Learn [2] and Streamlit [3].

Methodology

In this project, the following step of implementation is followed:

1. Select dataset that contains cancer cell geometric data.
2. Import necessary data analysis/manipulation modules and machine learning modules for later steps.
3. Perform exploratory data analysis, in search of data integrity, data type, number of samples, number of features and more. Determine linearity of the data.
4. Select machine learners that are suitable for classification problem. And perform preliminary predictive data analysis to choose the best candidates amongst the algorithms.
5. Hyperparameter Tuning, tune the hyper parameter of the selected best performing algorithm to improve its performance furthermore.
6. Evaluate the model's performance using classification report.
7. Save the trained, hyper parameter tuned model for later use.
8. Perform model interpretation analysis using Lime [4] package, observe model behaviour.
9. Export the model and data visualisation artefacts (figure, classification report, object) to Streamlit web application.
10. Implement a cancer cell prediction web application prototype.

Stage 1: Algorithm design

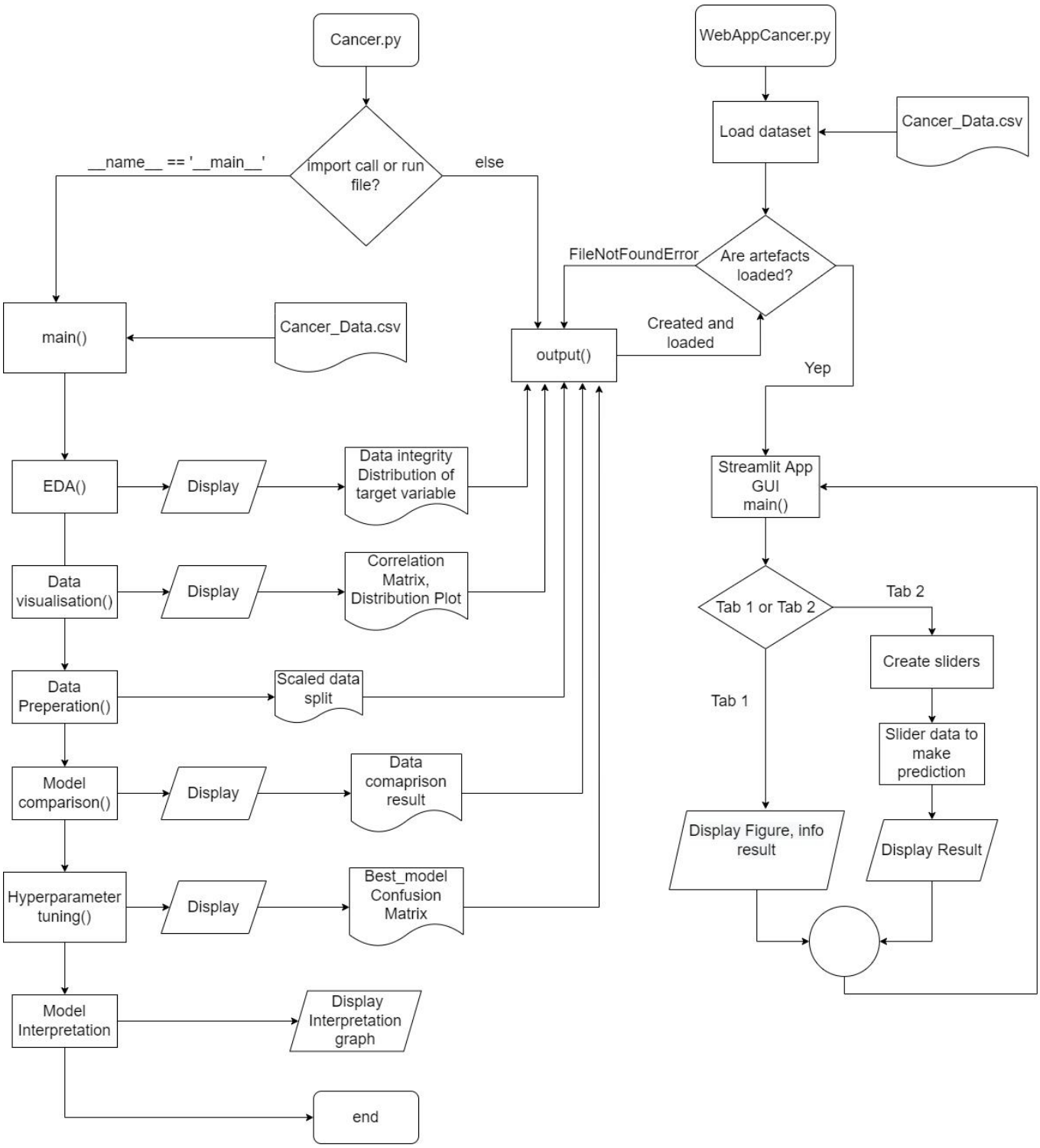


Figure 1 Flow Chart of algorithm design

Figure 1 illustrates the program's design. There will be two python modules built for this project. The first one is Cancer.py. This module will be responsible for the Exploratory Data Analysis, Data Visualisation, Data pre-processing, predictive analysis (Model training), hyperparameter tuning, and model interpretation. The artefacts such as figure, trained model, classification report generated by this module will be captured and stored in local directory by the output function. WebAppCancer.py will import artefacts generated by Cancer.py for showcase and interactive cell estimation using the trained model.

Dataset description

The dataset used in this project is Cancer Data from Kaggle repository [1] . It is available publicly and similar dataset can be found in scikit-learn [2] machine learning package, namely Breast Cancer Dataset. However, in this project, the CSV file from Kaggle is used. The dataset has 569 samples of tumour cell's geometric data generated from image of tumour cells. It has 30 features, and 1 classification attribute. The feature consists of data like the mean, standard error, worst/largest value of the radius of cell, compactness of cell, perimeter of cell and more. The target attribute is separated into two categories. With B meaning the sample cell is benign, and M meaning that the sample cell is malignant. Therefore, in order to predict the tumour cell, these data will be fed to the machine learner. But first, I need to perform exploratory data analysis to examine the data.

Exploratory data analysis

In this analysis, a virtual environment is built using Microsoft Visual Studio Code IDE. Python Version 3.11.3 is used. And the following libraries and packages are imported into the virtual environment.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import pandas as pd
import numpy as np
import joblib
from lime import lime_tabular
```

Figure 2 modules imported in Cancer.py.

As shown in Figure 2, number of modules is imported. Scikit-learn are imported for predictive analysis, and data pre-processing. Matplotlib and Seaborn are imported for data visualisation. Pandas and NumPy are imported for Data Exploration. Lime is imported for model interpretation. And lastly, Joblib is imported for models and artefact saving.

```
# Model save name

saveFile = 'Best_SVM_Model.sav'

## Load the Cancer Dataset

fileName = 'Cancer_Data.csv'
cancer_dataset = pd.read_csv(fileName)
```

Figure 3 Import dataset.

Figure 3 shows the dataset import.

The following are the question that I would like to answer. Some of the snip shots codes are compiled from the jupyter notebook, nevertheless the program is **not** made in jupyter notebook format (.ipynb). So, rest assured.

(1) What does the dataset look like?

```
# Data head, Example of what the data looks like
print(cancer_dataset.head(5))

         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
...
3                 0.17300          NaN
4                 0.07678          NaN

[5 rows x 33 columns]
```

Figure 4 First Five row of the dataset.

(2) How many rows and column does the dataset have?

```
# shape
print(cancer_dataset.shape)
```

```
(569, 33)
```

Figure 5 Shape of Dataset

(3) What is the data type of these data, and what are the features?

```
cancer_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Figure 6 Data type of dataset.

So, we can tell from Figure 6 that there are three types of data. They are int, object, and float.

(4) How many benign cases and malignant samples are in the dataset?

```
# Distribution of benign and malignant case
print(cancer_dataset.groupby('diagnosis').size())
```

```
diagnosis
B    357
M    212
dtype: int64
```

Figure 7 Number of Benign and Malignant cases

From Figure 7, we can see that the dataset is slightly imbalanced with more Benign cases. Nevertheless, the imbalanced will have minimum impact on the model's performance.

(5) Is there any missing value?

```
checkForNull = dataset.isnull().sum()
print(checkForNull)
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

Figure 8 Check Missing Value

From Figure 8, we can tell that none of the features have value missing. However, there are a feature, namely Unnamed: 32, have 569 missing values.

Based on the exploratory data analysis, there are a few things that draws attention.

1. From Figure 4, we can tell that the data type of the target attribute is object, which will make the later analysis difficult.
2. From Figure 4, we can tell that ID feature has no correlation with the samples target attribute, the tumour cell's state is not determined by its assigned ID. Hence we will need to remove it in the next stage of analysis.
3. From Figure 4, 6, and 8, a feature with 569 missing values, denoted as NaN is shown in the analysis. The result indicates that this is an empty column in the CSV file, which will need to be remove as well. Because some machine leaner such as the K nearest Neighbour model cannot take empty data as input.

Data Visualisation

To visualise the data, we first need to remove unwanted feature and column as described above. To do that, we will use these two algorithms:

```
# Drop unnecessary data entry for visualisaiton and re-examine the dataset
dataset.drop(['id', 'Unnamed: 32'], axis = 1, inplace = True)
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
(569, 31)
```

Figure 9 Remove before and after comparison.

In Figure 9, we can see that the two columns are removed as intended, and other features remain intact.

With that out of the way, the data visualisation can proceed.

Also, the diagnosis variable needs to be changed from B, M to 0, 1 in order to plot figures.

```
# Replace string B&M to binary 0, 1 so that correlation can be analysed
dataset['diagnosis'] = dataset['diagnosis'].replace({'B': 0, 'M':1})
```

Figure 10 Substitute object variable to binary

Heat map – Correlation Matrix

```
# Heat map
heat_map, ax = plt.subplots(figsize=(20, 20)) #Set size of the heat map
sns.heatmap(dataset[dataset.columns].corr(), annot=True, ax = ax)
plt.title("Correlation Matrix")
plt.show()
```

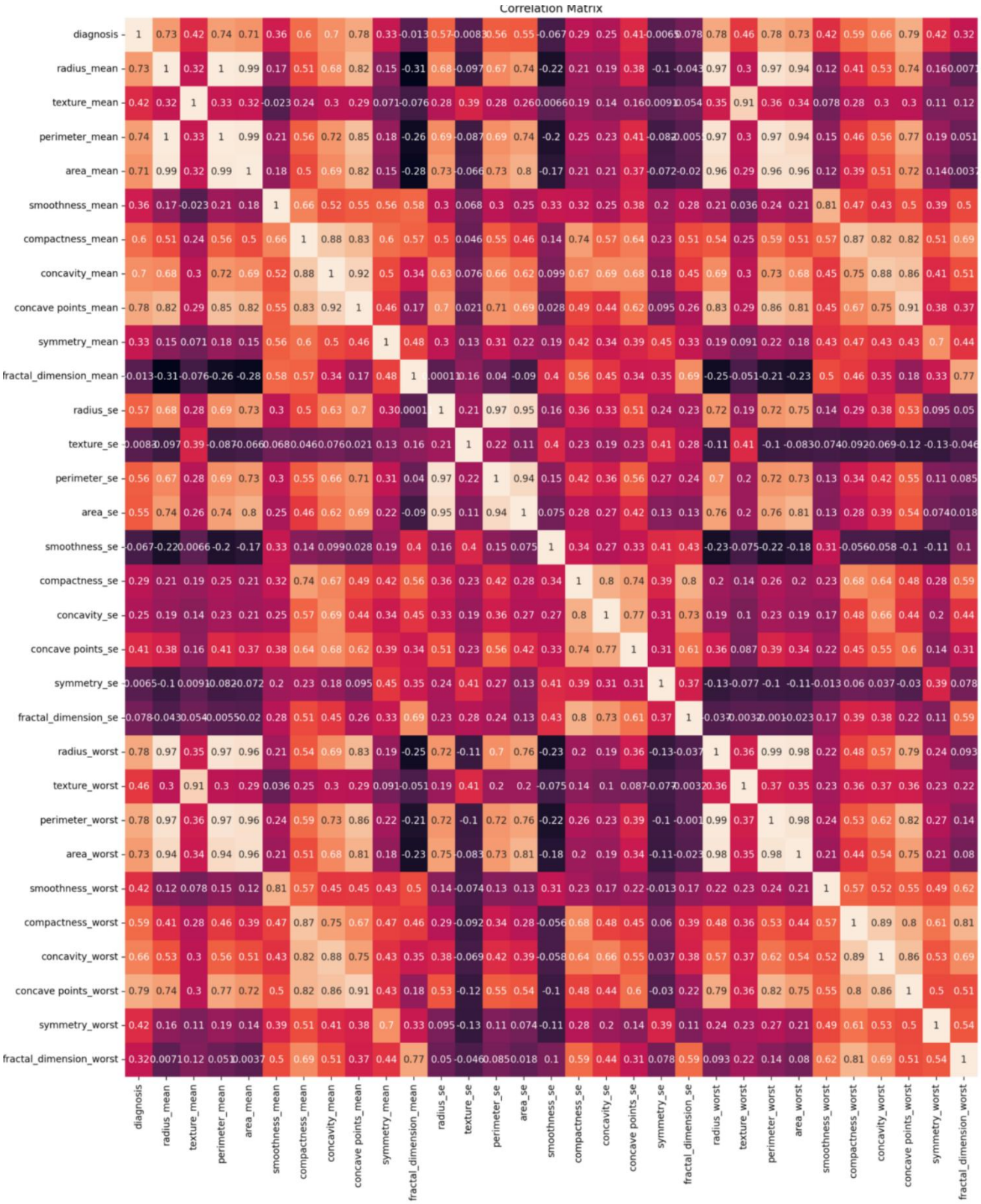


Figure 11 Heatmap of Correlation Matrix

Distribution of features and target

```
# Distribution of each attribute
distplot = plt.figure(figsize = (28,21))
for feature in range(len(dataset.columns)):
    plt.subplot(6, 6, feature + 1)
    sns.histplot(data = dataset, x = dataset.columns[feature], hue = 'diagnosis' )
```

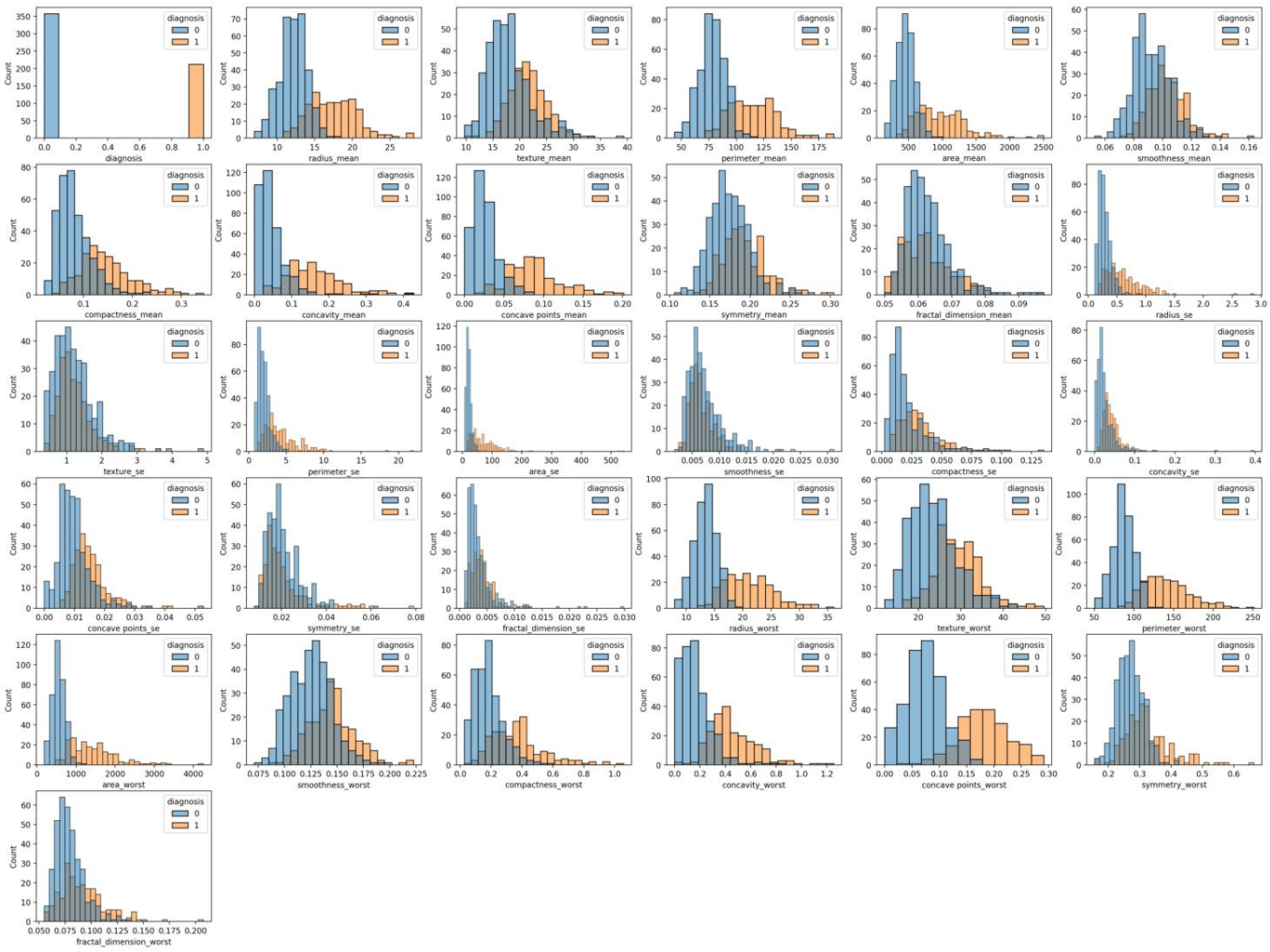


Figure 12 Distribution plot

The distribution plot illustrates the distribution of features and target. From the plot, we can tell by the overlapping of the graph, that we are dealing with non-linearly solvable data. Hence, in terms of algorithm choice, I posit that that machine learners that is suitable for non-linear issue will perform better.

Predictive Analysis

To use the data to train the machine learning model, several procedures need to be undertaken.

Feature Scaling

As shown in Figure 4, we can tell that the magnitude between data is drastically different. Some varies in decimal, and some varies in ten's and hundred's due to the difference in unit . Some machine leaners are inherently sensitive to the magnitude and scale of the data. For example. Distance-based machine leaner such as Support Vector Machine (SVM) and K Nearest Neighbour (KNN). They use the distance between each data points to estimate their probability [5]. Therefore, scale the data into lower or similar magnitudes will help the algorithm to perform better.

Example:

```
## Preparing the data for analysis
X = cancer_dataset.drop('diagnosis', axis = 1) #Everything except the diagnosis column
y = cancer_dataset['diagnosis'] # The diagnosis column
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
# Test run of analysis using Support vector machine
model = SVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred, average = 'weighted'))
print("Recall:",metrics.recall_score(y_test, y_pred, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, y_pred, average = 'weighted'))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9473684210526315
Precision: 0.9514695830485304
Recall: 0.9473684210526315
F1-score: 0.9464615931721194
[[71  0]
 [ 6 37]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        71
           1       1.00      0.86      0.92        43

    accuracy                           0.95       114
   macro avg       0.96      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Figure 13 Unscaled

We can see that the accuracy is 94.7% using unscaled data.

```
# Feature scaling
# SVM with scaled data, proves that scaled data help the algorithm significantly 95% to 97%
scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)

model_with_scaled_data = SVC()
model_with_scaled_data.fit(scaled_X_train, y_train)
new_y_pred = model_with_scaled_data.predict(scaled_X_test)

print("Accuracy:",metrics.accuracy_score(y_test, new_y_pred))
print("Precision:",metrics.precision_score(y_test, new_y_pred, average = 'weighted'))
print("Recall:",metrics.recall_score(y_test, new_y_pred, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, new_y_pred, average = 'weighted'))
print(confusion_matrix(y_test, new_y_pred))
print(classification_report(y_test, new_y_pred))
```

```
Accuracy: 0.9824561403508771
Precision: 0.9829367940398942
Recall: 0.9824561403508771
F1-score: 0.9823691172375383
[[71  0]
 [ 2 41]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

Figure 14 Scaled.

The Accuracy of this SVM algorithm improve from 94.7% to 98.2%.

Therefore, after splitting the feature columns and the target column:

```
# Seperate feature and target set and Feature scaling
scaler = StandardScaler()
X = dataset.drop('diagnosis', axis = 1) #Everything except the diagnosis column
X_scaled = scaler.fit_transform(X)
y = dataset['diagnosis'] # The diagnosis column
```

Figure 15 Feature scaling using standard scaler.

```
[[ 1.09706398 -2.07333501  1.26993369 ...  2.29607613  2.75062224
   1.93701461]
 [ 1.82982061 -0.35363241  1.68595471 ...  1.0870843  -0.24388967
   0.28118999]
 [ 1.57988811  0.45618695  1.56650313 ...  1.95500035  1.152255
   0.20139121]
 ...
 [ 0.70228425  2.0455738   0.67267578 ...  0.41406869 -1.10454895
  -0.31840916]
 [ 1.83834103  2.33645719  1.98252415 ...  2.28998549  1.91908301
   2.21963528]
 [-1.80840125  1.22179204 -1.81438851 ... -1.74506282 -0.04813821
  -0.75120669]]
```

Figure 16 Feature Columns after scaling.

We will use StandardScaler() from Scikit-Learn to standardise our feature columns X. The value after the feature is scaled represents the standard deviation the value is away from the mean value of the feature. Calculated by this formula:

$$X^{'} = \frac{X - \mu}{\sigma}$$

Train Test Split

After scaling our data, a split of the data into training set and validation (Test) set is performed using trian_test_split().

```
# Train test split
scaled_X_train, scaled_X_test, y_train, y_test = train_test_split(
                                        X_scaled, y, test_size = 0.2, random_state = 42)
```

Figure 17 Train Test Split

With the training set vs. Testing set ration to be 80:20. And a fixed random state so that the performance result will not vary in order to have a more stable comparison in the next stage.

Model comparison

In order to choose the best performing model for this task, an algorithm comparison is performed using K-fold Cross Validation Technique.

Four candidates are chosen based on there competence in solving nonlinear classification problems.

- Support Vector Machine
- Classification and Regression Tree (CART), also known as Decision Tree
- K Nearest Neighbour
- Naïve Bayes

```python
def modelComparison(scaled_X_train, y_train):
    # Model collection - Spot Check

    models = []
    models.append(('KNN', KNeighborsClassifier()))
    models.append(('CART', DecisionTreeClassifier()))
    models.append(('NB', GaussianNB()))
    models.append(('SVM', SVC(gamma='auto')))

    results = []
    names = []
    message = ""
    for name, model in models:
        kfold = KFold(n_splits=10, random_state=42, shuffle=True)
        cv_results = cross_val_score(model, scaled_X_train, y_train, cv=kfold, scoring='accuracy')
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        message += msg + '\n'
        print(msg)

    # Compare Algorithms

    compare_algorithm = plt.figure()
    compare_algorithm.suptitle('Algorithm Comparison')
    ax = compare_algorithm.add_subplot(111)
    plt.boxplot(results)
    ax.set_xticklabels(names)
    plt.show()
    return models, compare_algorithm, message
```

Figure 18 Model Comparison Code - K fold Cross Validation

In this algorithm, the train-test-dataset is split 10 times (n_splits = 10) covering all parts of the train-test-dataset. Using this technique can help us to avoid bias results caused by over-fitting the model, which means the model is performing excellently in the training process but perform poorly when it meets unseen data. The result of the validation is shown below.

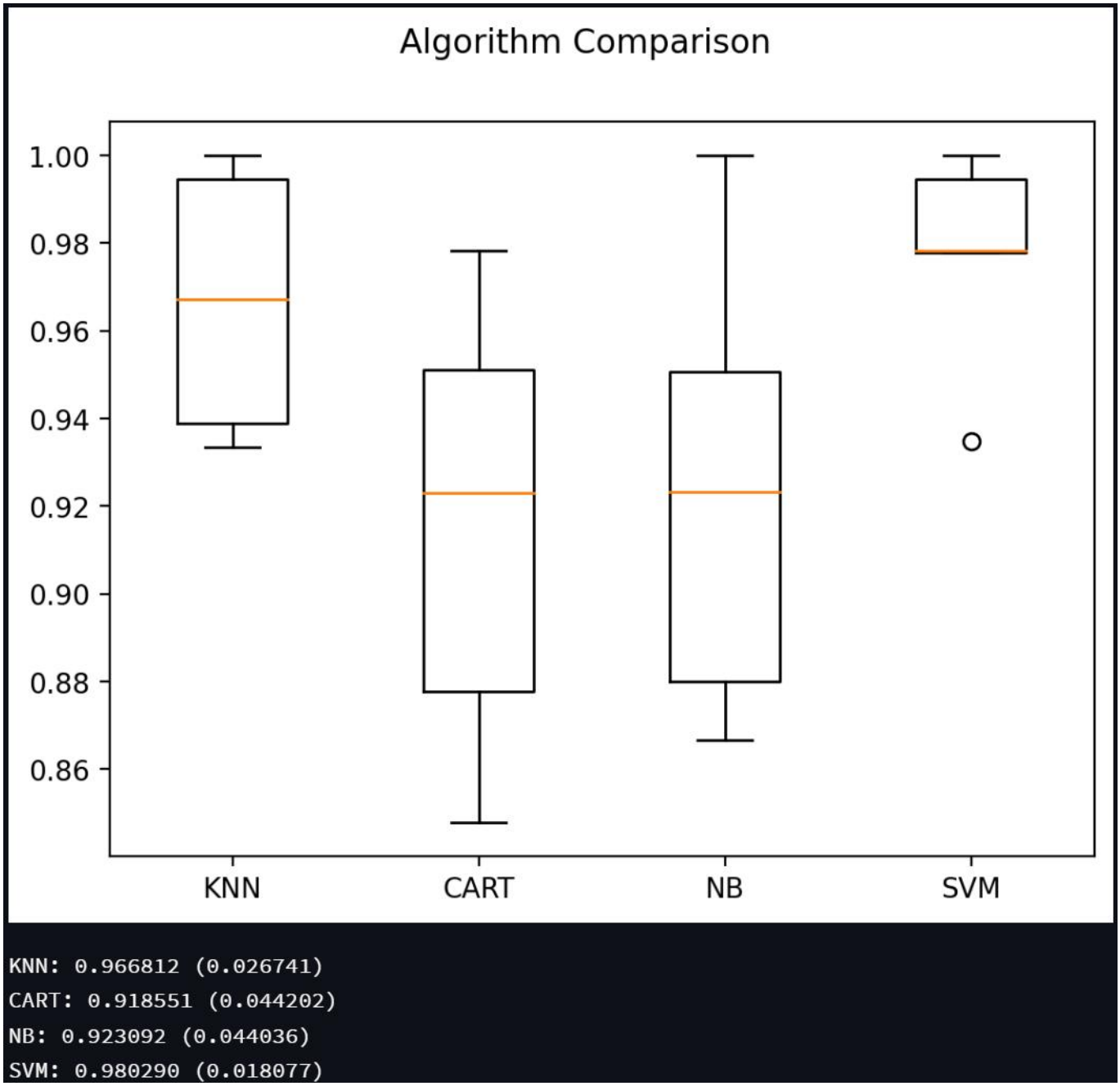


Figure 19 Comparison result

From the result, we can see that all the candidates perform well, there accuracy score all reaches over 90%. The phenomenon can be explained by the fact that the dataset is relatively small. With only 569 sample, the result may not be indicative. Nevertheless, we can see that SVM outperform the rest of the candidates with the highest accuracy score of 98%. From the box plot, we can tell that it performed stably with a few outliers lying at approximately 93%, which is still a very good performance. And the standard deviation supports the fact that it is performing stably as well.

Hence, for the choice of machine learner, support vector machine is the model that will be deployed in the Web Application for cell estimation.

Stage 2: Algorithm Implementation

Hyperparameter tuning.

With the best model for solving this problem identified, model training will be in process. But first, we can tune the Hyperparameter of the model to increase it performance on the model using GridSearchCV().

```
## Hyper parameter tuning for the best result

# hyper parameter of current better model
better_model = models[3][1]
params = better_model.get_params()
print(params)
```

Figure 20 Current Param Code

{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'auto', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}

In these parameters, four of them are worth mentioning.

Support vector machine works as a separator, for example, when the datapoint (support vector) is plotted in a 3D space, support vector machine tries to find a hyperplane that can separate data in to N categories. Thus:

1. C – Fault tolerance, how many "mistakes" is it allowed to make, with higher C meaning more and lower meaning less. Sometimes, the lowest is not always the best.
2. Gamma – How concave can the hyperplane be, with more concaveness/curvature enabled (smaller gamma), the hyperplane will be more "uneven" and fit the support vectors more closely, and vice versa. Again, smaller value is not always the best.
3. Kernel – for support vector machine, there are a few kernels that the SVM can use for calculation, mostly RBF and Linear. Since we are dealing with a nonlinear classification problem. We will use the RBF kernel.
4. Probability – SVM is not a probability-based machine leaner (unlike Naïve Bayes). However, probability can be enabled to mimic a probabilistic estimation. For the Web Application's function, the probability will be set as TRUE.

```
# Find optimal hyper parameter for this model
params_grid = {'C':[10, 5, 1, 0.5, 0.1, 0.01],
               'gamma':[0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1],
                'kernel': ['rbf'],
                'probability': [True]}
grs = GridSearchCV(better_model, params_grid, refit = True, verbose = 1, cv = 20)
grs.fit(scaled_X_train, y_train)

# Display hyper parameter changes
print ("\n Best Hyper Parameters: ", grs.best_params_,"\n", "Best estimator: ", grs.best_estimator_,"\n")
```

```
Fitting 20 folds for each of 42 candidates, totalling 840 fits

 Best Hyper Parameters:  {'C': 10, 'gamma': 0.01, 'kernel': 'rbf', 'probability': True}
 Best estimator:  SVC(C=10, gamma=0.01, probability=True)
```

Figure 21 GridSearchCV code

20 folds of calculation using the above configurations are conducted, a total of 840 attempts. The function then returns the best configuration, which is:

{C =10, gamma = 0.01}

Trained model and result

```
# Train model with hyper parameter tuned
model_best = grs.best_estimator_
y_prediction = model_best.predict(scaled_X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_prediction))
print("Precision:",metrics.precision_score(y_test, y_prediction, average = 'weighted'))
print("Recall:",metrics.recall_score(y_test, y_prediction, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, y_prediction, average = 'weighted'))
print(confusion_matrix(y_test, y_prediction))
print(classification_report(y_test, y_prediction))
```

```
Accuracy: 0.9824561403508771
Precision: 0.9829367940398942
Recall: 0.9824561403508771
F1-score: 0.9823691172375383
[[71  0]
 [ 2 41]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

Figure 22 GridSerachCV Result

The result showed a small improvement.

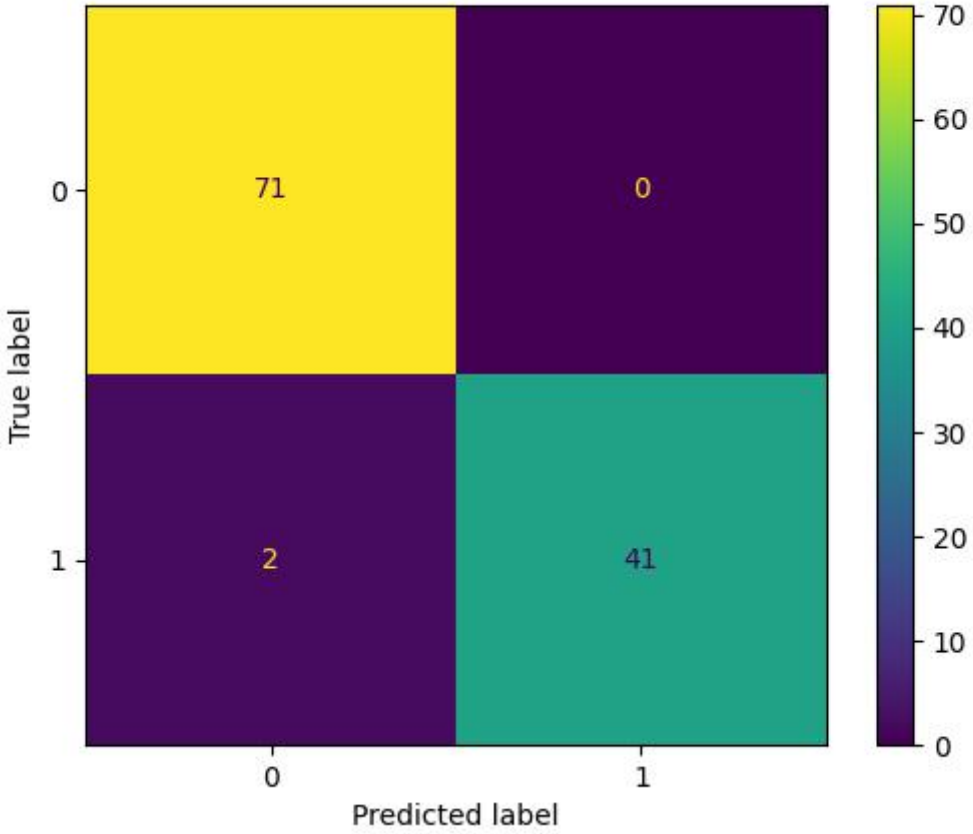


Figure 23 Confusion Matrix of the model

Save Model

And finally, save the trained model for Web Application.

```
# Save the model for later use in Web App
joblib.dump(model_best, saveFile)
```

Figure 24 Save model code.

Best_SVM_Model.sav	11-May-2023 21:25	SAV File	17 KB

Figure 25 Saved model example.

Model Interpretation

A train model does not give out any information about what the reason that the model is making the prediction. Thus, the Lime package [4] is used for model interpretation.

```
def modelInterpretation(scaled_X_train, scaled_X_test, model_best, X):

    exp = lime_tabular.LimeTabularExplainer(scaled_X_train,
                                            class_names= ['benign', 'maglinant'],
                                            feature_names= X,
                                            mode = 'classification' )

    explanation = exp.explain_instance(scaled_X_test[2], model_best.predict_proba,
                                       num_features = len(X),
                                       top_labels = 2)

    explanation.as_pyplot_figure()
    plt.show()
```

Figure 26 Model interpretation using Lime.

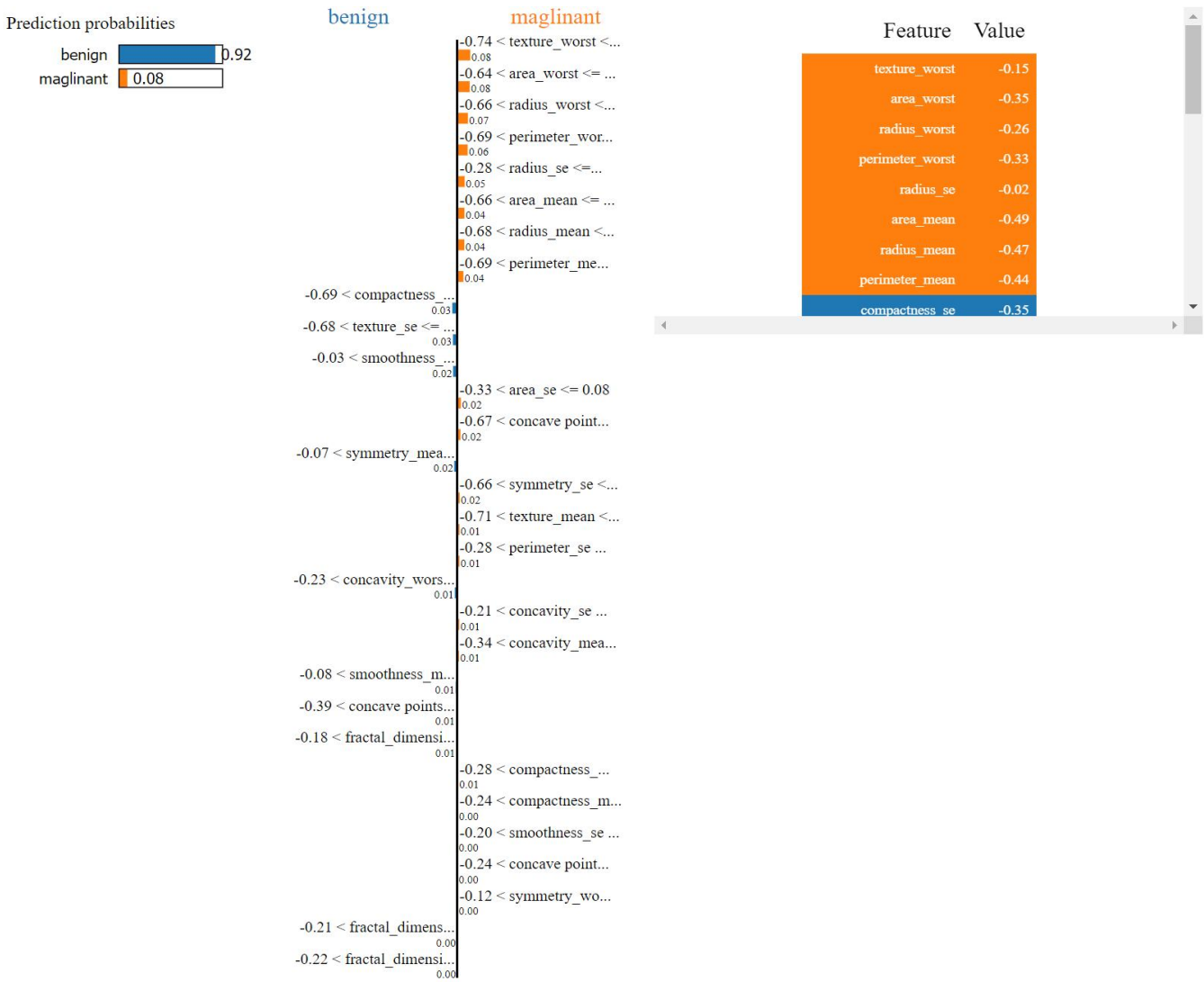


Figure 27 Interpretation Figure

From the above figures, we can see that the scaled_X_test[2] is chosen to make a prediction. The above interpretation result indicates that the model is 92% confident that the result is a benign case. And the vertical bar chart illustrates the factors that influence the model to make the prediction.

Export artefacts for WebApp

After the model is trained, it is necessary to program a function that export all the relevant artefacts (figure, model, object), so that the web application can utilise it without having to rerun Cancer.py every instance when user is using the application.

```
## Protect the code from runniing in case of import
if __name__ == '__main__':
    main()
```

```
## Main function

def main():
    drop_dataset = EDA(cancer_dataset)[2]
    dataset = dataVisualisation(drop_dataset)[2]
    X, scaled_X_train, scaled_X_test, y_train, y_test, scaler = dataPreperation(dataset)
    models, fig, message = modelComparison(scaled_X_train, y_train)
    model_best, tuned_result, report = hypTuning(models, scaled_X_train, scaled_X_test, y_train, y_test)
    modelInterpretation(scaled_X_train, scaled_X_test, model_best, X)
```

Figure 28 Main function

```
def output(brand_new_dataset):
    checkForNull, distBM, drop_dataset = EDA(brand_new_dataset)
    heat_map, distplot, dataset = dataVisualisation(drop_dataset)
    X, scaled_X_train, scaled_X_test, y_train, y_test, scaler = dataPreperation(dataset)
    models, fig, message = modelComparison(scaled_X_train, y_train)
    model_best, tuned_result, report = hypTuning(models, scaled_X_train, scaled_X_test, y_train, y_test)
    modelInterpretation(scaled_X_train, scaled_X_test, model_best, X)
    joblib.dump(checkForNull, 'checkForNull')
    joblib.dump(distBM, 'distBM')
    joblib.dump(heat_map, 'heat_map')
    joblib.dump(distplot, 'distplot')
    joblib.dump(X, 'X')
    joblib.dump(scaler,'scaler')
    joblib.dump(fig, 'compare_algorithm')
    joblib.dump(message, 'message')
    joblib.dump(tuned_result, 'tuned_result')
    joblib.dump(report, 'report')
    joblib.dump(dataset, 'cancer_dataset_processed')
```

Figure 29 Output function

As shown above, the output function works similarly to the main function, with difference being that the output function will use the Joblib package to export and save the artifacts as binary save files into the same directory Cancer.py is in.

Best_SVM_Model.sav	11-May-2023 21:25	SAV File	17 KB
Confusion_Matrix.png	11-May-2023 21:25	PNG File	14 KB
cancer.py	11-May-2023 21:11	Python Source File	11 KB
cancer_dataset_processed	11-May-2023 16:25	File	140 KB
compare_algorithm	11-May-2023 16:25	File	116 KB
distplot	11-May-2023 16:25	File	6,459 KB
message	11-May-2023 16:25	File	1 KB
report	11-May-2023 16:25	File	1 KB
scaler	11-May-2023 16:25	File	3 KB
tuned_result	11-May-2023 16:25	File	17 KB
X	11-May-2023 16:25	File	135 KB
heat_map	11-May-2023 16:25	File	1,829 KB
distBM	11-May-2023 16:25	File	2 KB
checkForNull	11-May-2023 16:25	File	3 KB
webAppSVMCancer.py	11-May-2023 16:23	Python Source File	5 KB

Figure 30 Artefact saved.

Stage 3: Software Deployment

Import of Web App

```
## Load dataset
dataset = pd.read_csv('Cancer_Data.csv')
raw_dataset = dataset
```

```
import streamlit as sl
from cancer import raw_dataset, output, saveFile
from PIL import Image
import numpy as np
import pandas as pd
import joblib
```

Figure 31 Web App Import

For the Web application, Streamlit is used to create a web interface. Besides the aforementioned module import used in Cancer.py, PIL which stands for python Pillow built in imaging library is also used to display Confusion_Matrix.fig.

Load artefacts

```
# While loop to load objects/model created by Cancer.py
# Will not stop until it is sure that everything is loaded
while True:
    try:
        loaded_model = joblib.load(saveFile)
        checkForNull = joblib.load('checkForNull')
        distBM = joblib.load('distBM')
        heat_map = joblib.load('heat_map')
        distplot = joblib.load('distplot')
        X = joblib.load('X')
        scaler = joblib.load('scaler')
        compare_algorithm = joblib.load('compare_algorithm')
        message = joblib.load('message')
        tuned_result = joblib.load('tuned_result')
        report = joblib.load('report')
        cancer_dataset = joblib.load('cancer_dataset_processed')
        break
    except FileNotFoundError:
        output(dataset) # First time running this code will be redirected here,
```

Figure 32 While loop load artefact.

For first time users, there will not be anything to load. So, a FileNotFoundError will raise, redirecting the program to the output function imported from Cancer.py, thus generating all the artefact needed.

Web App Layout

```
def main():
    # Title and sub title
    sl.title("Cancer Cell Prediction")
    sl.markdown("Predict if cell is cancerous based on its geometric features")

    # Two tabs for data visualisation and prediction program
    dsTab, ceTab = sl.tabs(["Data Showcase","Cell estimator"])

    # Data Showcase Tab
    with dsTab:
        dataShowcase()

    # Cell Estimator Tab
    with ceTab:
        slider_input = create_sliders()
        make_prediction(slider_input)
```

The web application is contained in the main function. It has two tabs, Data Showcase and Cell estimator. Which does what their name suggested.

Data showcase tab

```
@sl.cache_data # Streamlit caching function, stores already loaded function into cache so no re-load is need
def dataShowcase():
    sl.header("Cancer dataset")
    sl.write(raw_dataset)

    sl.subheader("Check for missing value")
    sl.write(checkForNull)

    sl.subheader("Number of benign and malignant cases in this dataset")
    sl.write(distBM)

    sl.header("Cancer dataset after cleaning")
    sl.text("Dropped ID and Blank column, convert diagnosis into binary data")
    sl.write(cancer_dataset)

    sl.header("Heat Map - Correlation Matrix")
    sl.pyplot(heat_map)

    sl.header("Distribution of dataset features and target")
    sl.pyplot(distplot)

    sl.header("Algorithm vs. Dataset")
    sl.text("Four algorithm was chosen for comparison.\nK Nearest Neighbour, Decision Tree, Naive Bayes, and
    sl.pyplot(compare_algorithm)
    sl.text(message)

    sl.header("Hyper parameter tuning result")
    sl.text(tuned_result)
    sl.subheader("Classification Report")
    sl.text(report)
    sl.subheader("Confusion Matrix of the model")
    image = Image.open('Confusion_Matrix.png')
    sl.image(image)

if __name__ == '__main__':
    main()
```

All the results and figured generated from Cancer.py is placed in this tab. It is worth mentioning that the Streamlit function @sl.cache_data is used. Due to the nature of Streamlit, the webpage will refresh itself each time the user made an input. In this case, when the slider moves, the whole application will refresh itself. The cache function solved this problem by saving the unchanged data and skip the execution of the function that has already been executed. The speed of estimation vastly improved.

Cell estimator Tab - ESTIMATION FUNCTION

This is the core or the project. It consists of slider created using the feature name of the dataset using for loop. And the allowed minimum and maximum input is set to the minimum and maximum of each feature. The default input is the mean of the features. The result will be shown on this tab providing the estimated result (Benign, Malignant) and the Confidence of the model prediction.

```python
# Prediction function

def make_prediction(sliderList):
    sliderList = np.array(list(sliderList)).reshape(1, -1) # Convert list to ndarray and lower dimmensio
    sliderList = scaler.transform(sliderList) # Use scaler to scale original input, since the model is t
    predict = loaded_model.predict(sliderList) # Make prediction based on input

    if predict[0] == 0:
        sl.markdown("Result: Benign")
    else:
        sl.markdown("Result: Malignant")

    prediction_prob = loaded_model.predict_proba(sliderList) # Show confidence of model prediction
    proba = prediction_prob[0][predict[0]] # The prediction
    sl.metric(label = "Confidence", value = "{:.2f}%".format(proba*100),
              delta = "{:.2f}%".format((proba - 0.5) * 100)) # Show precentage change of confidence co

# Create slider and record slider input

def create_sliders():
    sliderList = []
    for features in X.columns:  # Generate sliders using loop
        slider = sl.slider(label = features,
                           max_value = float(cancer_dataset[features].max()),
                           min_value = float(cancer_dataset[features].min()),
                           value = float(cancer_dataset[features].mean()))
        sliderList.append(slider)
    return sliderList
```

Web App Sample showcase

Cancer Cell Prediction

Predict if cell is cancerous based on its geometric features

Data Showcase Cell estimator

Cancer dataset

	is	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mea
60		10.17	14.88	64.55	311.9	0.1134	0.080
61		8.598	20.98	54.66	221.8	0.1243	0.089
62		14.25	22.15	96.42	645.7	0.1049	0.200
63		9.173	13.86	59.2	260.9	0.0772	0.087
64		12.68	23.84	82.69	499	0.1122	0.126
65		14.78	23.94	97.4	668.3	0.1172	0.147
66		9.465	21.01	60.11	269.4	0.1044	0.077
67		11.31	19.04	71.8	394.1	0.0814	0.04
68		9.029	17.33	58.79	250.5	0.1066	0.141
69		12.78	16.49	81.37	502.5	0.0983	0.052

Check for missing value

	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_w	0
Unnamed: 32	569

Number of benign and malignant cases in this dataset

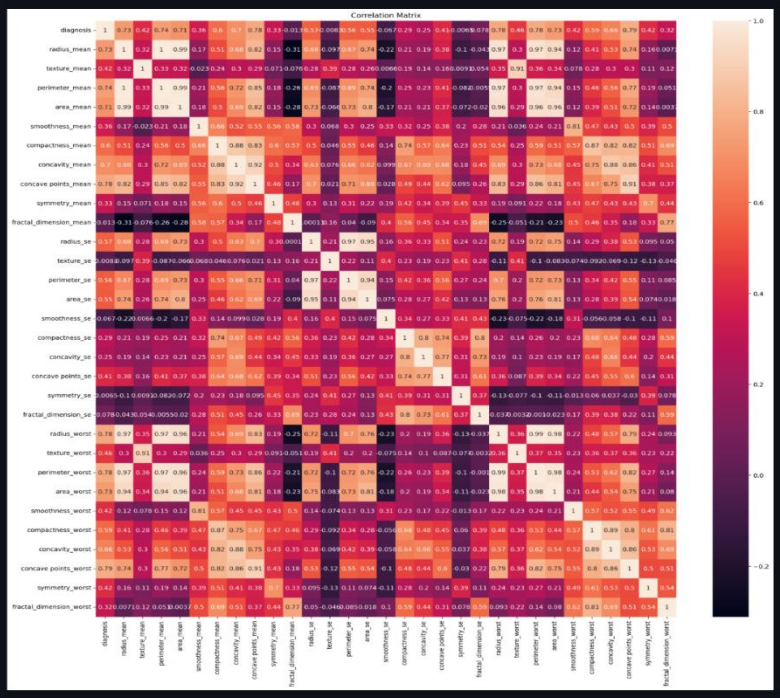
diagnosis	0
B	357
M	212

Cancer dataset after cleaning

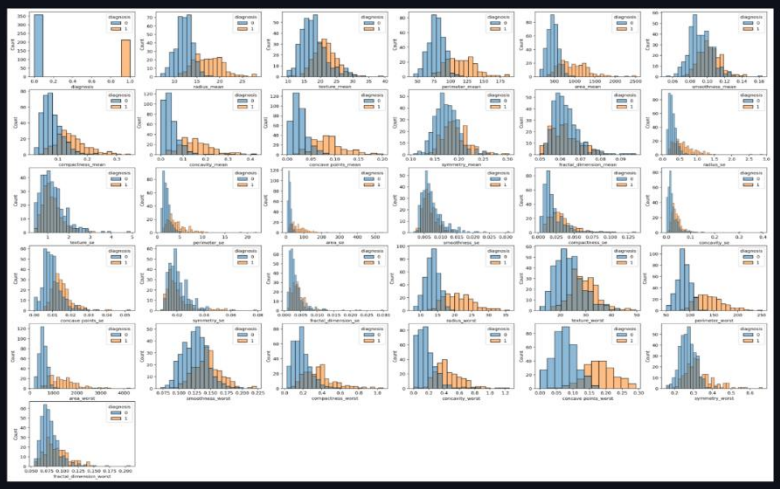
Dropped ID and Blank column, convert diagnosis into binary data

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactn
37	0	13.03	18.42	82.61	523.8	0.0898	
38	1	14.99	25.2	95.54	698.8	0.0939	
39	1	13.48	20.82	88.4	559.2	0.1016	
40	1	13.44	21.58	86.18	563	0.0816	
41	1	10.95	21.35	71.9	371.1	0.1227	
42	1	19.07	24.81	128.3	1,104	0.0908	
43	1	13.28	20.28	87.32	545.2	0.1041	
44	1	13.17	21.81	85.42	531.5	0.0971	
45	1	18.65	17.6	123.7	1,076	0.1099	
46	0	8.196	16.84	51.71	201.9	0.086	
47	1	13.17	18.66	85.98	534.6	0.1158	

Heat Map - Correlation Matrix

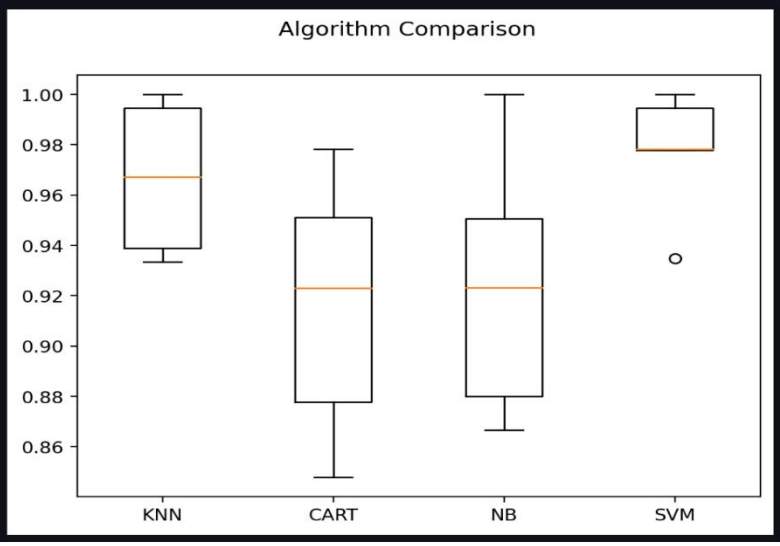


Distribution of dataset features and target

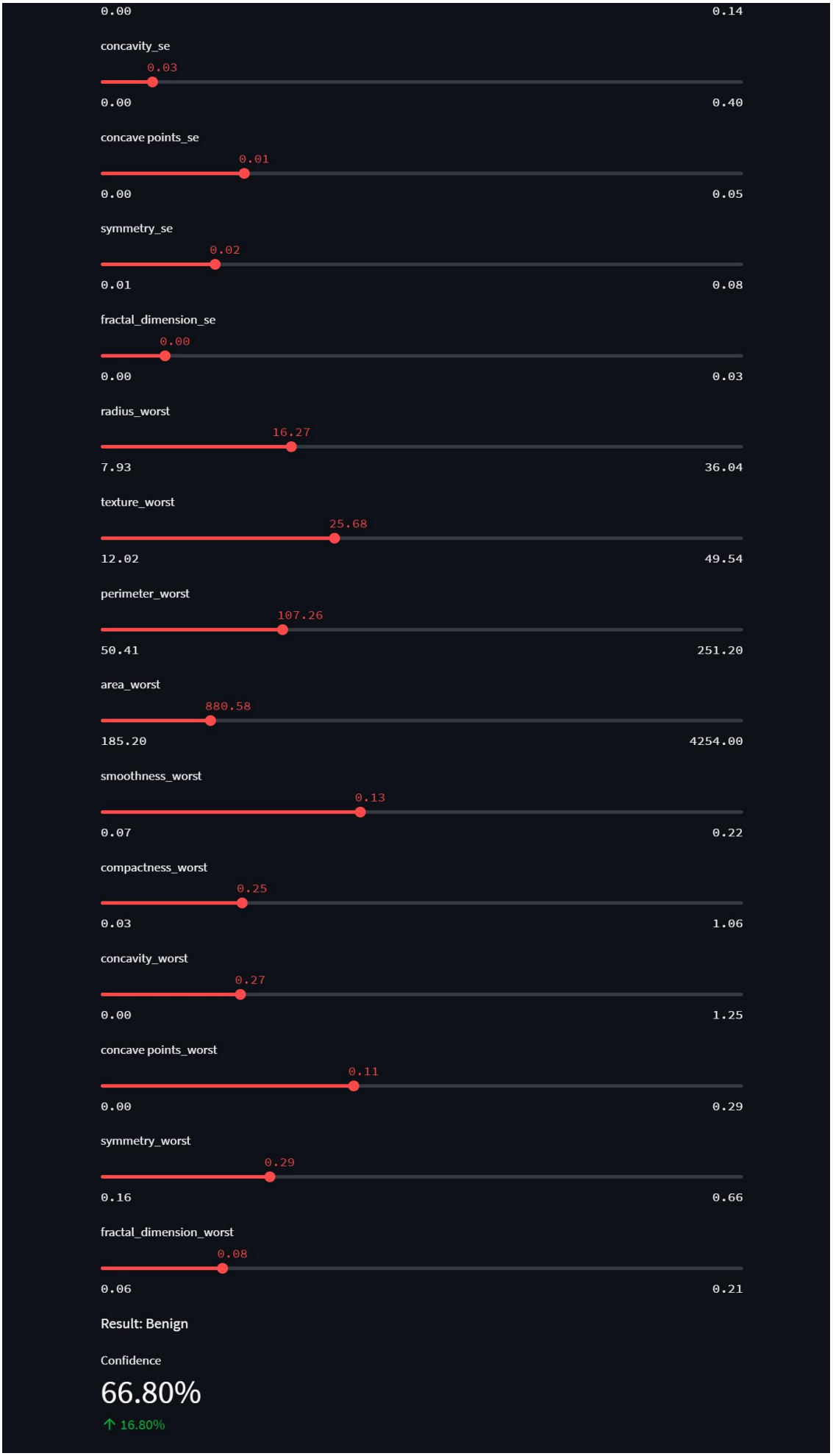


Algorithm vs. Dataset

```
Four algorithm was chosen for comparison.
K Nearest Neighbour, Decision Tree, Naive Bayes, and Support Vector Machine.
```




Cancer Cell Prediction
Predict if cell is cancerous based on its geometric features
Data Showcase
Cell estimator
radius_mean
14.13
6.98
28.11
texture_mean
19.29
9.71
39.28
perimeter_mean
91.97
43.79
188.50
area_mean
654.89
143.50
2501.00
smoothness_mean
0.10
0.05
0.16
compactness_mean
0.10
0.02
0.35
concavity_mean
0.09
0.00
0.43
concave points_mean
0.05
0.00
0.20
symmetry_mean
0.18
0.11
0.30
fractal_dimension_mean
0.06
0.05
0.10
radius_se
0.41
0.11
2.87
texture_se
1.22
0.36
4.88
perimeter_se
2.87
0.76
21.98
area_se
40.34
6.80
542.20
smoothness_se
0.01
0.00
0.03
compactness_se
0.03
0.00
0.14
concavity_se
0.03
0.00
0.40


0.00
0.14
concavity_se
0.03
0.00
0.40
concave points_se
0.01
0.00
0.05
symmetry_se
0.02
0.01
0.08
fractal_dimension_se
0.00
0.00
0.03
radius_worst
16.27
7.93
36.04
texture_worst
25.68
12.02
49.54
perimeter_worst
107.26
50.41
251.20
area_worst
880.58
185.20
4254.00
smoothness_worst
0.13
0.07
0.22
compactness_worst
0.25
0.03
1.06
concavity_worst
0.27
0.00
1.25
concave points_worst
0.11
0.00
0.29
symmetry_worst
0.29
0.16
0.66
fractal_dimension_worst
0.08
0.06
0.21
Result: Benign
Confidence
66.80%
↑ 16.80%


25.68
12.02
49.54
perimeter_worst
107.26
50.41
251.20
area_worst
880.58
185.20
4254.00
smoothness_worst
0.13
0.07
0.22
compactness_worst
0.25
0.03
1.06
concavity_worst
0.27
0.00
1.25
concave points_worst
0.22
0.00
0.29
symmetry_worst
0.59
0.16
0.66
fractal_dimension_worst
0.19
0.06
0.21
Result: Malignant
Confidence
98.81%
↑ 48.81%

Evidence of Testing

Screen shots of testing has been provided in the previous section of the report.

Test plan**:**

In terms of Test plans. The implementation of machine learner training was done in Jupyter Lab using Jupyter notebook. Each step of the implementation was run individually to ensure the data structure and intended input and output is as expected. It is further tested when the program is modularised into functions. There are resemblances between the test plan adopted in this project and unit testing.

As for the software deployment stage. Initially, the webpage content and function are added block by block. Upon completion, the development enters a lite version of user acceptance testing. Tested by other individual (my partner) besides the developer (me). A suggestion/complaint was received from the tester, which is the program ran too slow. It is because the output function was not created at that time of development. The lack of output function resulted in the web application execute the Cancer.py program every time the user moves the slider (input a value). Nevertheless, the inefficiency of the code is fixed.

Reflection

In this project, I experienced most of the part of the software development life cycle, including planning/designing, analysis on data and possible implementation, implementation of both the model training module and web application, and the testing of the program and its integration. Throughout the development, I faced many hurdles. Since the beginning of the project, setting up virtual environments had spent a considerable amount of time. I have struggled for approximately an hour on this matter. And the situation is resolved in minutes by researching online. It demonstrates that in the field of software development, seeking help online as needed is of utmost importance. Furthermore, understanding the data structure and data type of each parameter will help a developer to quickly debug in the coding process. In the webapp, the slider input is stored as a list. Not only that converting it to a ndarray is important, but also the dimensionality needs to be reduced for support vector machine to make prediction. Which raises another point, that is, documentation is a programmer's best friend. Approximately 60% of the bug I had encountered is solved by studying the documentation.

References

[1] E. Taha, “Cancer Data,” 2023. [Online]. Available: https://www.kaggle.com/datasets/erdemtaha/cancer-data. [Accessed 25 April 2023].

[2] Scikit-Learn, “Scikit-Learn Machine Learning in Python,” March 2023. [Online]. Available: https://scikit-learn.org/stable/. [Accessed 25 April 2023].

[3] Streamlit, “A faster way to build and share data apps,” 2023. [Online]. Available: https://streamlit.io/. [Accessed 1 May 2023].

[4] P. Banerjee, “Explain your model predictions with LIME,” 2020. [Online]. Available: https://www.kaggle.com/code/prashant111/explain-your-model-predictions-with-lime. [Accessed 1 May 2023].

[5] Analytics Vidhya, “Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023),” 11 April 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/#What_Is_Normalization?. [Accessed 3 May 2023].