INDEX

ABSTRACT

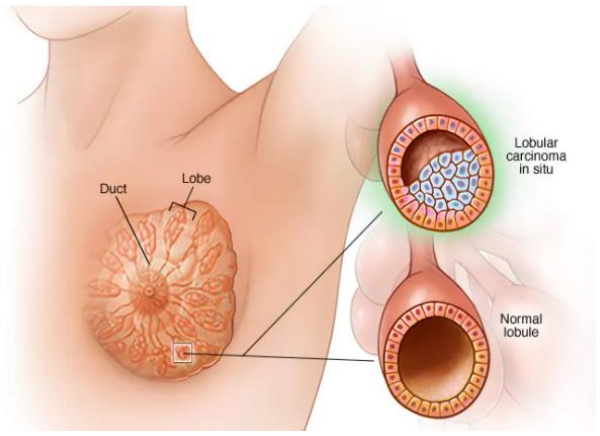
We tried processing a method to predict breast cancer on the basis of ten individual attributes, including radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry and fractal dimension. The mean, standard error, and worst of these features are computed for each image which results in 30 features.

Principal component analysis (PCA) was used to identify valuable parts of the data and further reduce the dimensions of the data and identify these features that causes breast cancer using a dataset and we use Logistic regression to create a model and predict the output.

INTRODUCTION AND EXPLANATION

Breast Cancer

Breast cancer is a type of cancer that starts in the breast. Cancer starts when cells begin to grow out of control. Breast cancer cells usually form a tumor that can often be seen on an x-ray or felt as a lump. Breast cancer can occur in both men and women, but usually appears in women.



Software Requirements

- **Matlab -** It is a programming and numeric computing platform used to analyze data, develop algorithms, and create models. We performed PCA in matlab by using the Singular value decomposition (SVD) method.

- **Google colab –** It is a Cloud service provided by Google to run complex algorithms which would require a lot of computational power. We analyzed data and performed logistic regression in google colab.

Dataset

A collection of related sets information that is composed of separate elements but can be manipulated as a unit by a computer is called as a dataset. Here, we have used a .csv file which we convert to a .xlsx file for easier reading in matlab. Our dataset consists data of 569 patients who either have caner or not. It has two attributes, id and diagnosis in columns one and two respectively.
We have ten real valued features of the nucleus of the cell computed from column three to column thirty two. They are,

- Radius: distances from centre to points on the perimeter
- Texture: standard deviation of grey-scale values
- Perimeter
- Area
- Smoothness: local variation in radius lengths
- Compactness: perimeter^2 / area - 1.0
- Concavity: severity of concave portions of the contour
- Concave points: number of concave portions of the contour
- Symmetry
- Fractal dimension: "coastline approximation" - 1

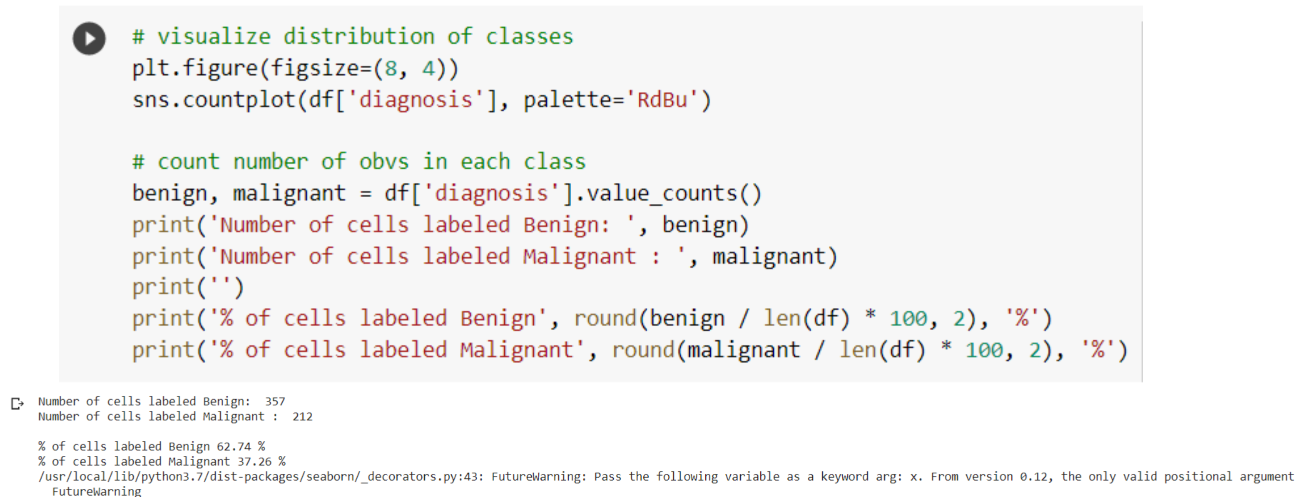
The mean, standard error, and worst of these features are computed for each image, resulting in 30 features.
Here, the mean is basically the mean of all the tests taken by a person, standard error is the standard deviation of the given values and worst is the average of the three largest values of the observation.

Diagnosis

When cells divide rapidly and abnormally it results in the formation of tumors. They are of two types:

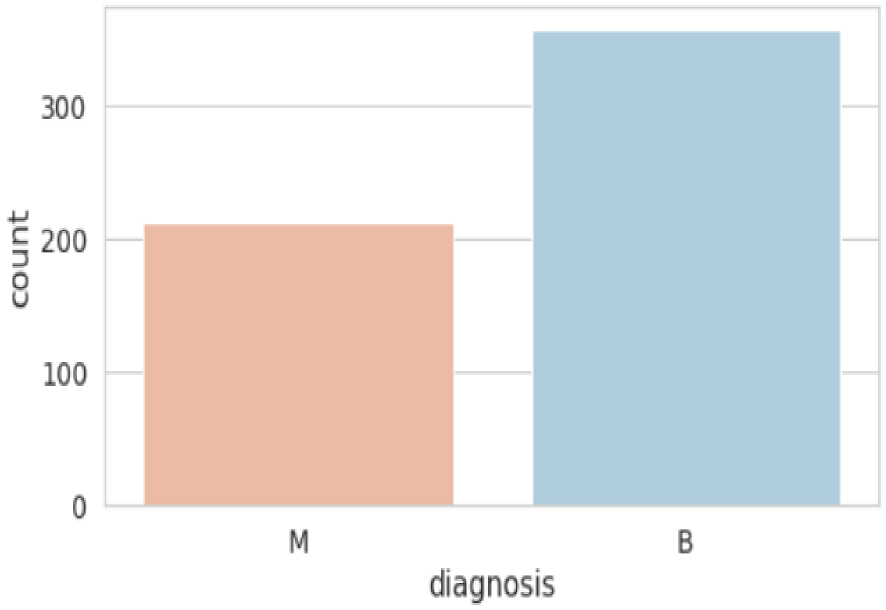
1. Malignant (cancerous)
2. Benign (non-cancerous)

From the figure, we can see that among 569 patients, 357 patients are labelled benign and 212 as malignant.



```python
# visualize distribution of classes
plt.figure(figsize=(8, 4))
sns.countplot(df['diagnosis'], palette='RdBu')

# count number of obvs in each class
benign, malignant = df['diagnosis'].value_counts()
print('Number of cells labeled Benign: ', benign)
print('Number of cells labeled Malignant : ', malignant)
print('')
print('% of cells labeled Benign', round(benign / len(df) * 100, 2), '%')
print('% of cells labeled Malignant', round(malignant / len(df) * 100, 2), '%')
```

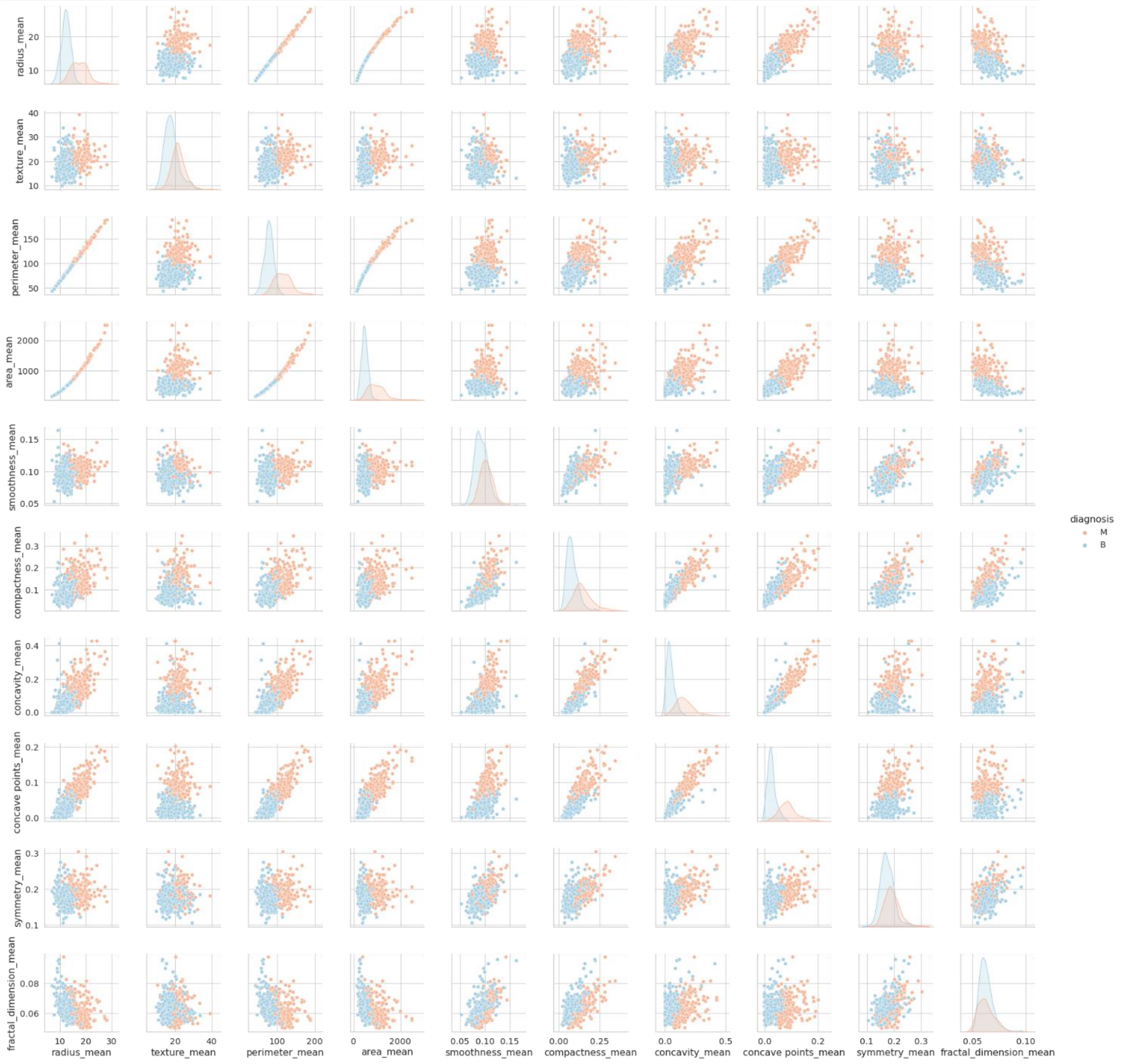
```
Number of cells labeled Benign:  357
Number of cells labeled Malignant :  212

% of cells labeled Benign 62.74 %
% of cells labeled Malignant 37.26 %
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument
  FutureWarning
```



For easier understanding of data, we have represented malignant as 1 and benign as 0 in matlab.

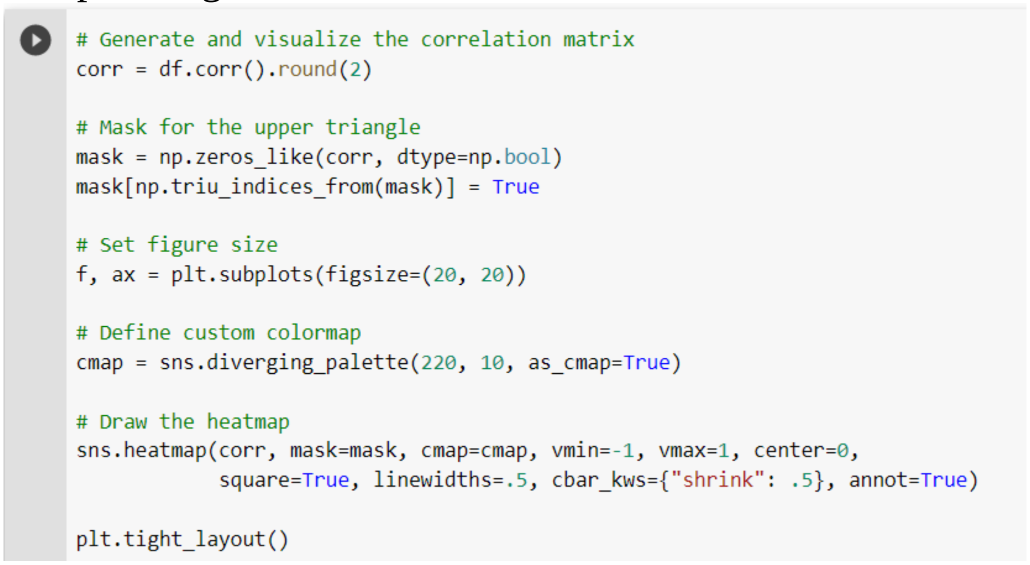
Now we'll observe patterns from 10 mean columns, here, as we can see, there are some interesting patterns visible. For instance, the almost perfectly linear patterns between the radius, perimeter and area attributes are hinting at the presence of multicollinearity between these variables. Another set of variables that can possibly imply multicollinearity are the concavity, concave_points and compactness.

```
[ ]  # generate a scatter plot matrix with the "mean" columns
     cols = ['diagnosis',
             'radius_mean',
             'texture_mean',
             'perimeter_mean',
             'area_mean',
             'smoothness_mean',
             'compactness_mean',
             'concavity_mean',
             'concave points_mean',
             'symmetry_mean',
             'fractal_dimension_mean']

     sns.pairplot(data=df[cols], hue='diagnosis', palette='RdBu')
```

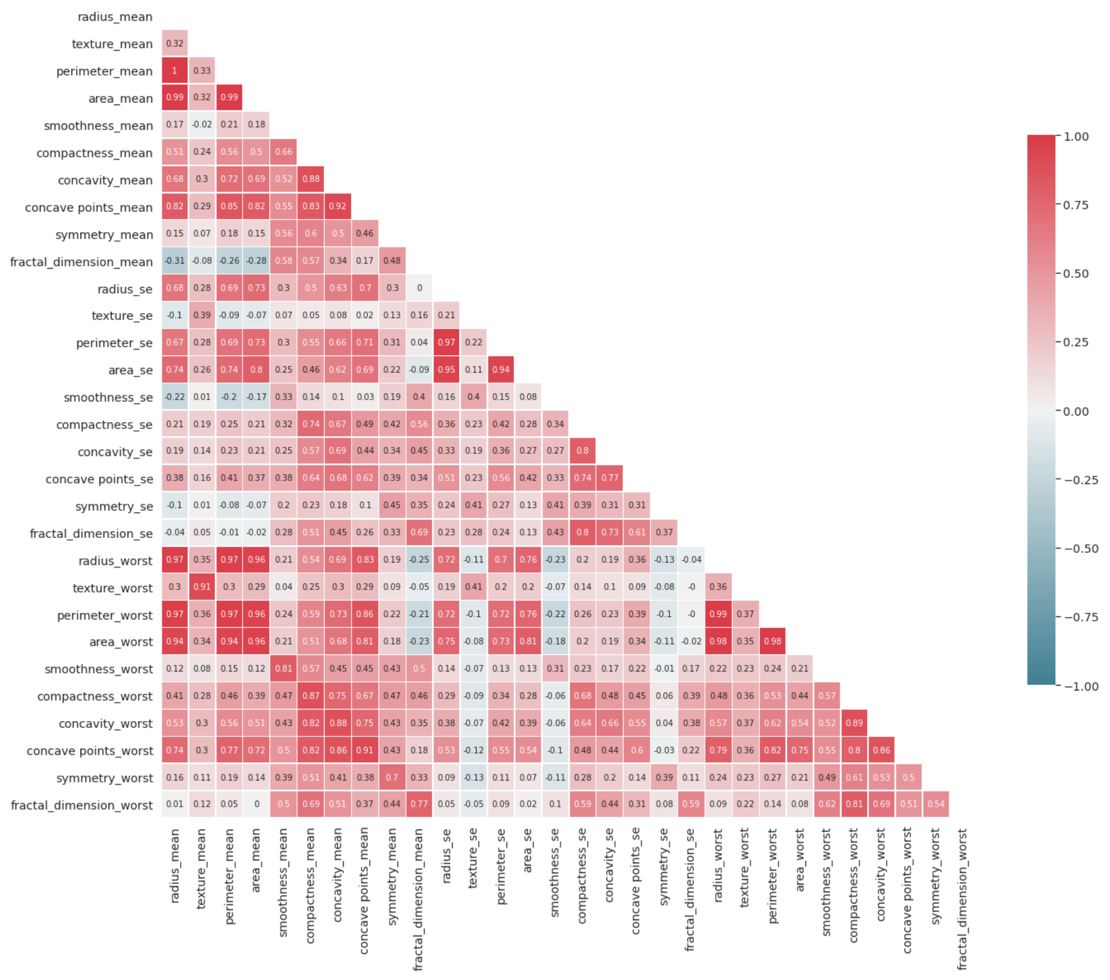


Correlation matrix

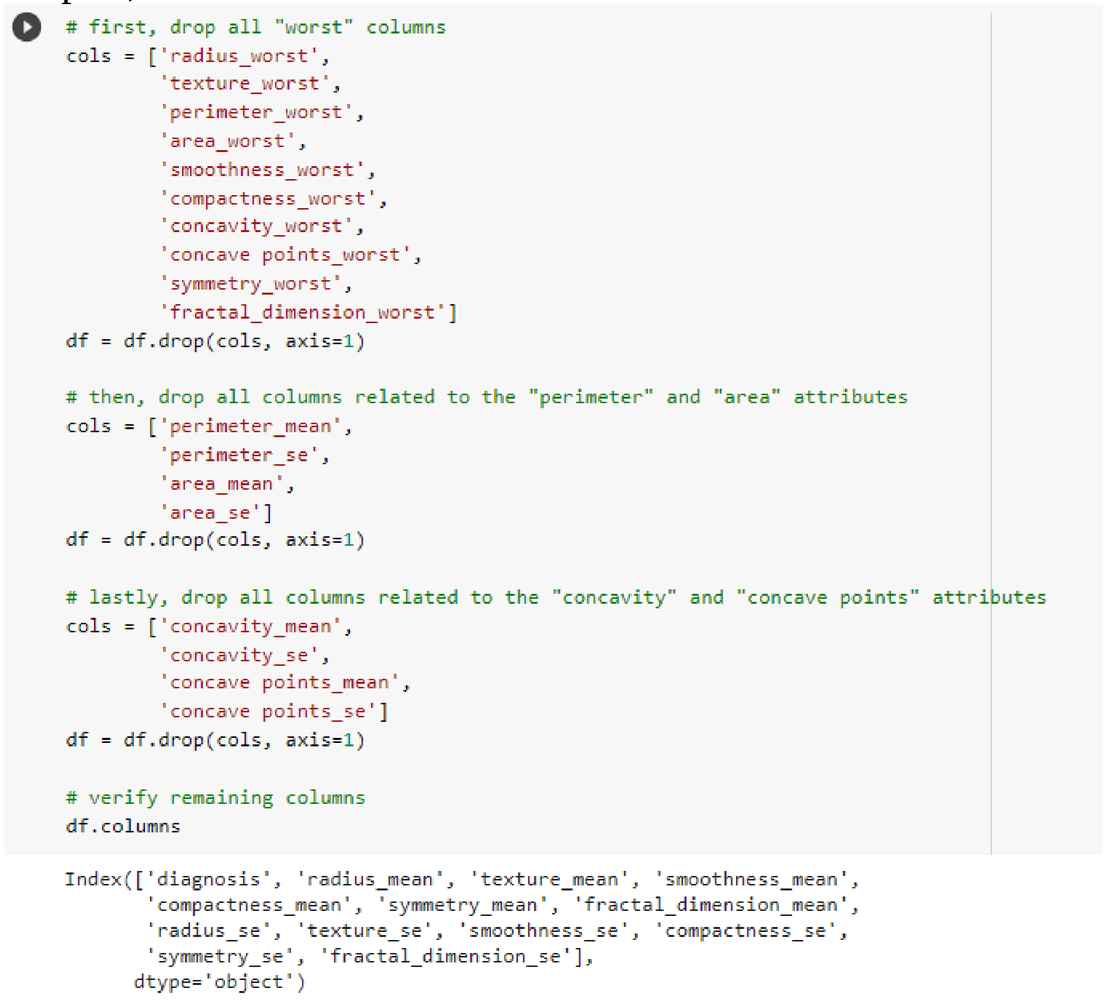
A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. We mask the upper triangular matrix because the same values will appear on the other side as same variables are been compared again.



```
# Generate and visualize the correlation matrix
corr = df.corr().round(2)

# Mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set figure size
f, ax = plt.subplots(figsize=(20, 20))

# Define custom colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.tight_layout()
```

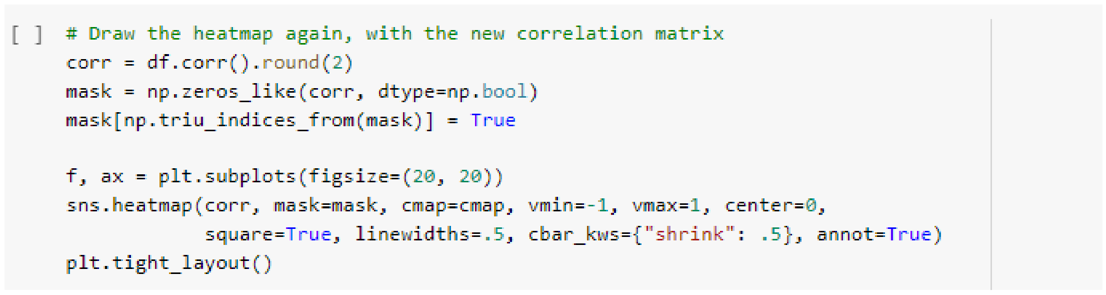


Here, it is noticed that the worst columns have more or less similar values to those in the mean columns. This is because worst columns are basically the subset of mean columns so to make this even more simpler, we remove the worst columns.

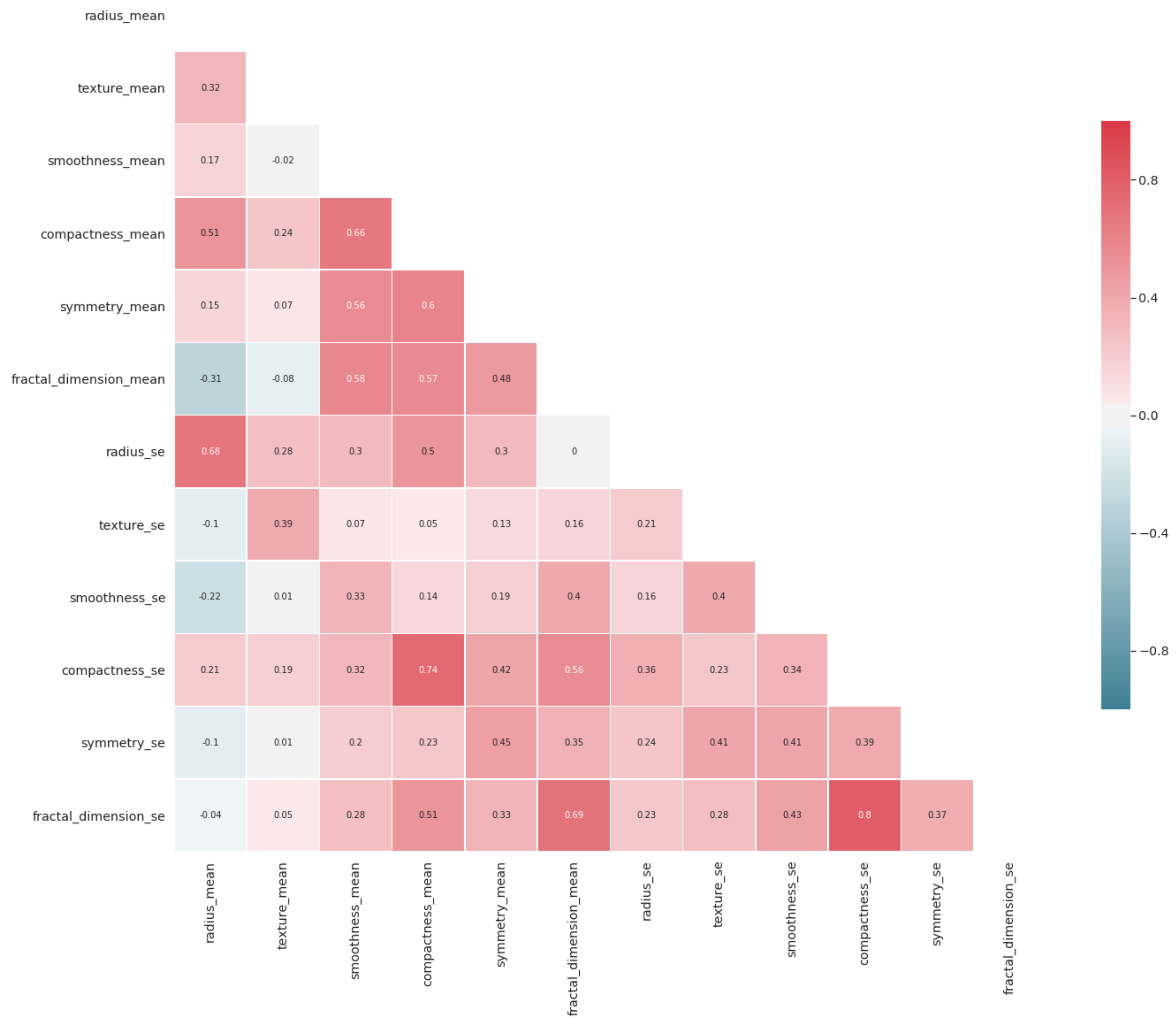


```
# first, drop all "worst" columns
cols = ['radius_worst',
        'texture_worst',
        'perimeter_worst',
        'area_worst',
        'smoothness_worst',
        'compactness_worst',
        'concavity_worst',
        'concave points_worst',
        'symmetry_worst',
        'fractal_dimension_worst']
df = df.drop(cols, axis=1)

# then, drop all columns related to the "perimeter" and "area" attributes
cols = ['perimeter_mean',
        'perimeter_se',
        'area_mean',
        'area_se']
df = df.drop(cols, axis=1)

# lastly, drop all columns related to the "concavity" and "concave points" attributes
cols = ['concavity_mean',
        'concavity_se',
        'concave points_mean',
        'concave points_se']
df = df.drop(cols, axis=1)

# verify remaining columns
df.columns
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'smoothness_mean',
       'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'symmetry_se', 'fractal_dimension_se'],
      dtype='object')
```



```
[ ]  # Draw the heatmap again, with the new correlation matrix
     corr = df.corr().round(2)
     mask = np.zeros_like(corr, dtype=np.bool)
     mask[np.triu_indices_from(mask)] = True

     f, ax = plt.subplots(figsize=(20, 20))
     sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
                 square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
     plt.tight_layout()
```



Principal component analysis (PCA)

It is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

PCA can be performed in two ways:

1. By finding mean, covariance and maximizing the variance

   Maximizing the variance,

   Z = C1X+ C2Y

   Linear combination of X and Y

   So, to find the possible value of C1 and C2 where the variance will be the highest, we solve it using the optimization technique

   Let us consider C1 as C1 and C2 and U2

Z = U1X1+ U2X2+………+ UDXD

Now let us consider x and u as a vector that forms this

$$\vec{X} = \begin{bmatrix} X1 \\ X2 \\ \vdots \\ XD \end{bmatrix} \qquad U = \begin{bmatrix} U1 \\ U2 \\ \vdots \\ UD \end{bmatrix}$$

And now we can express z as an inner product

Z = U.X = UTX = XTU

Let us assume that we have a sample set and we know that these samples are vectors given in some space

$\vec{X} = \{ \vec{X1}, \vec{X2}, \ldots\ldots\ldots \vec{XN}\}$

Mean is the average of all the vectors given in some space

Sample mean

X = 1/N∑ Xi

Covariance forms a matrix of size DxD and is generally positive semi definite matrix i.e., all the eigenvectors are positive or zero.

Covariance of x

C = 1/N∑ Xi . Xi'

Variance is z = utx, where u is the linear transformation which transforms D dimensional space to a single dimensional space.

$\mathrm{Var}(z) = 1/N\sum(Z-\vec{Z})\text{^}2$

On simplifying, we get

Var(z) = U'CU

To maximize var(z), we can assume that U tends to infinity

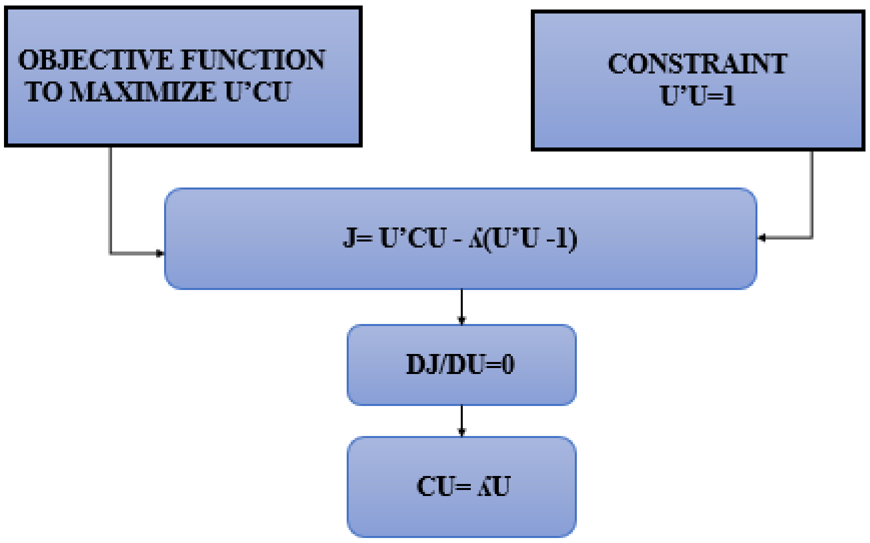
To avoid this, we must have limitations on the size of the vectors

||U|| = 1

Or

U'U = 1

Here, U is the linear transformation which transforms D dimensional space to a s single dimensional space. The lagrange multiplier is used to augment the constraint into the equation



Finally, to compute the principal components,
T = mean*eigenvectors(c).

2. Using singular value decomposition

   $X = U.\,\Sigma.V^T$

   Where,

   U – orthogonal matrix with unit eigenvectors of AA' in columns

   V – orthogonal matrix with unit eigenvectors of A'A in columns

   Σ – leading diagonal as square root of eigenvalues of A'A and AA'

   $T = U.\,\Sigma$

   Singular values in **Σ** gives us the indication of the amount of variance of the dataset that principal components capture. Here we take 2 principal components as we have to results for the diagnosis, either they have cancer or they don't.

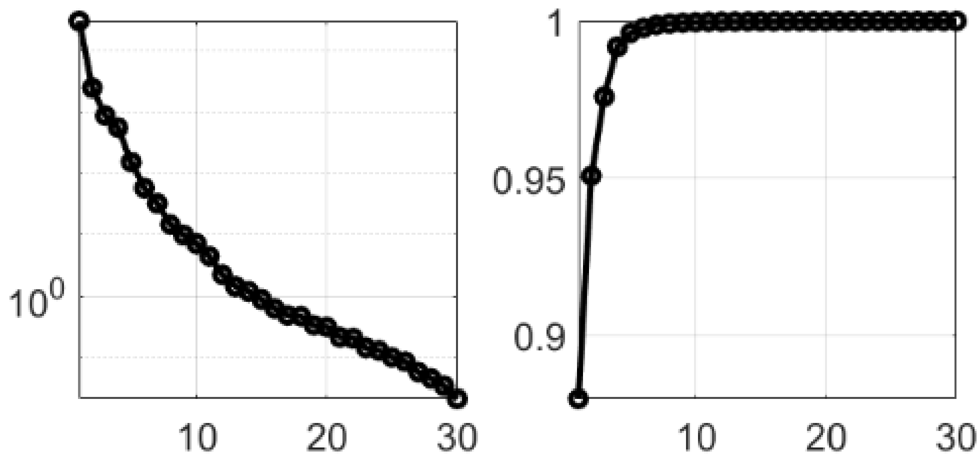
```
for i=1:size(Data,1)
     x = V(:,1)' * Data(i,:)';
     y = V(:,2)' * Data(i,:)';

     PC1(i) = x;
     PC2(i) = y;

     if (diagnosis(i) == 1)
         plot(x,y,'rx','LineWidth',3);
     else
         plot(x,y,'bo','LineWidth',3);
     end
end
xlabel('PC1'), ylabel('PC2')
```

Similarly we do the same for the new dataset, but we run it only once as we are comparing the data of this patient to the previous dataset values

```
[U1,Z1,V1] = svd(NewData);
x = V1(:,1)' * NewData(1,:)';
y = V1(:,2)' * NewData(1,:)';
plot(x, y, 'g*', 'MarkerSize', 10)
```

Plotting graphs

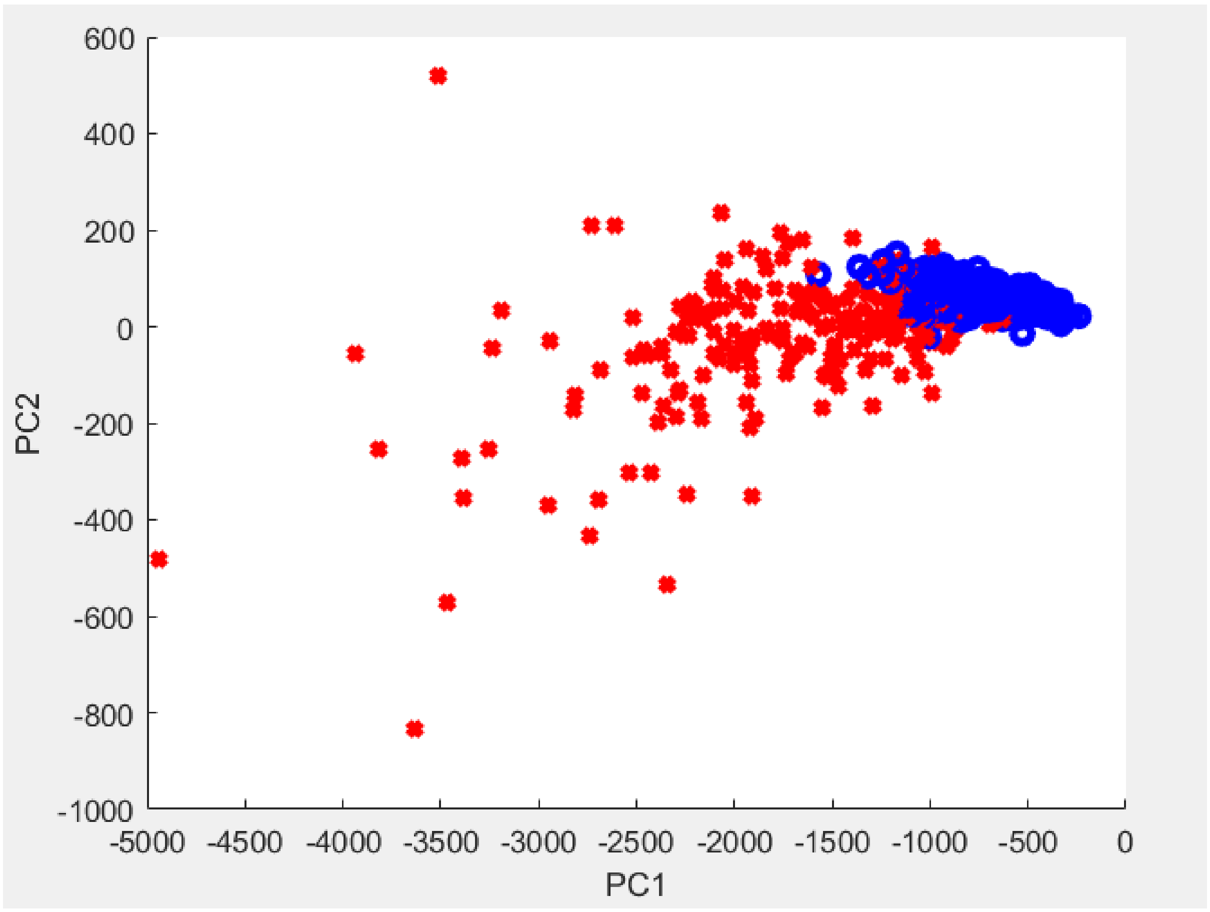
Percentage of plotting singular values and cumulative variance graph:



As seen in the 1st graph, the percentage of plotting the singular values on a log scale, x axis - (number of columns (rank)), y axis - magnitude

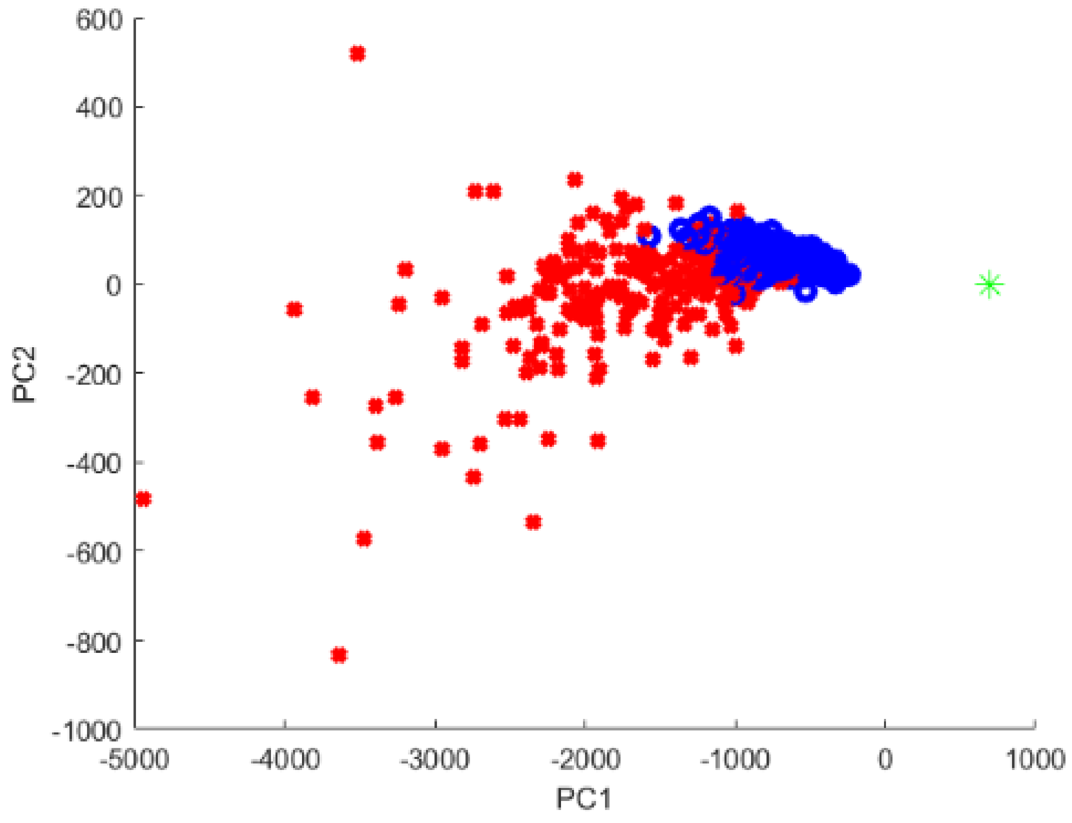
of log of the singular value. We can see that the first few values are quiet high, which indicate that those first few values have the most energy, so they are more likely to have the maximum variance and then it is seen that the graph tapers off.
In the 2nd graph, we plot the cumulative variance, it basically gives us the percentage of variances accounted for by the first n components. Since we have taken two principal values, we can see that nearly 95% of the data has been extracted.

Graph of data after PCA:



We can see the formation of two clusters. The red cluster or color indicates the patients who have cancer and the blue cluster or color indicate the patients who don't have cancer.

Graph using new data:



We predict if a person has cancer or not with the data provides. Certain values are given and when the model runs, we observe that the data of the new patient lies on the side where the patients do not suffer from cancer. This indicates that the patient has a higher chance of not having cancer.

Logistic Regression dataset

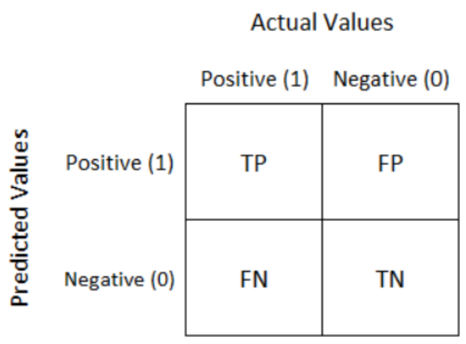
The dataset is split into two parts: Training and Testing datasets. Training dataset is an initial set of data used to help a program understand how to apply technologies like neural networks to learn and produce sophisticated results while the test set is a set of observations used to evaluate the performance of the model using some performance metric.
We have set the test size to 0.3; i.e., 70% of the data will be assigned to the training set, and the remaining 30% will be used as a test set. In order to obtain consistent results, we will set the random state parameter to a value of 40.

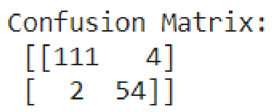
Model

The model can take some unlabeled data and effectively assign each observation a probability ranging from 0 to 1. This is the key feature of a logistic regression model. In our model, a probability of 1 corresponds to the "Benign" class, whereas a probability of 0 corresponds to the "Malignant" class. Therefore, we can apply a threshold value of 0.5 to our predictions, assigning all values closer to 0 a label of "M" and assigning all values closer to 1 a label of "B".

Confusion Matrix

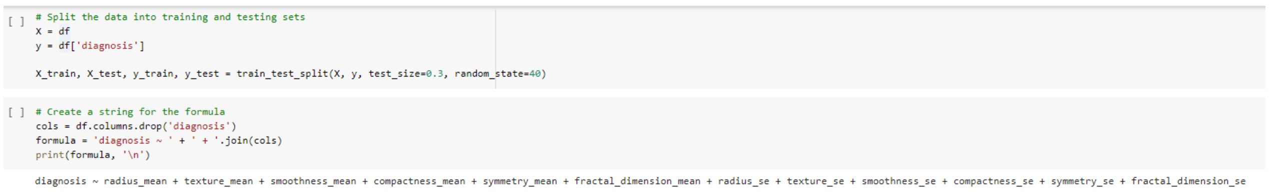
It is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.





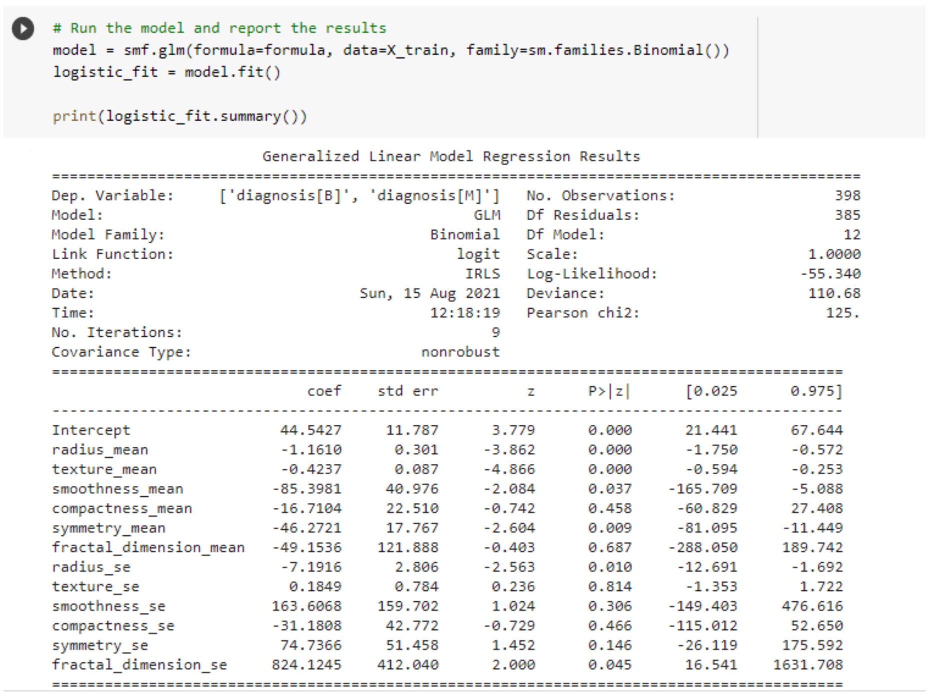
```
Confusion Matrix:
 [[111   4]
 [  2  54]]
```

True Positive:
Interpretation: You predicted positive and it’s true.
You predicted that a woman is pregnant and she actually is.
True Negative:
Interpretation: You predicted negative and it’s true.
You predicted that a man is not pregnant and he actually is not.
False Positive: (Type 1 Error)
Interpretation: You predicted positive and it’s false.
You predicted that a man is pregnant but he actually is not.
False Negative: (Type 2 Error)
Interpretation: You predicted negative and it’s false.
You predicted that a woman is not pregnant but she actually is.



```
[ ] # Split the data into training and testing sets
    X = df
    y = df['diagnosis']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)
```

```
[ ] # Create a string for the formula
    cols = df.columns.drop('diagnosis')
    formula = 'diagnosis ~ ' + ' + '.join(cols)
    print(formula, '\n')

    diagnosis ~ radius_mean + texture_mean + smoothness_mean + compactness_mean + symmetry_mean + fractal_dimension_mean + radius_se + texture_se + smoothness_se + compactness_se + symmetry_se + fractal_dimension_se
```

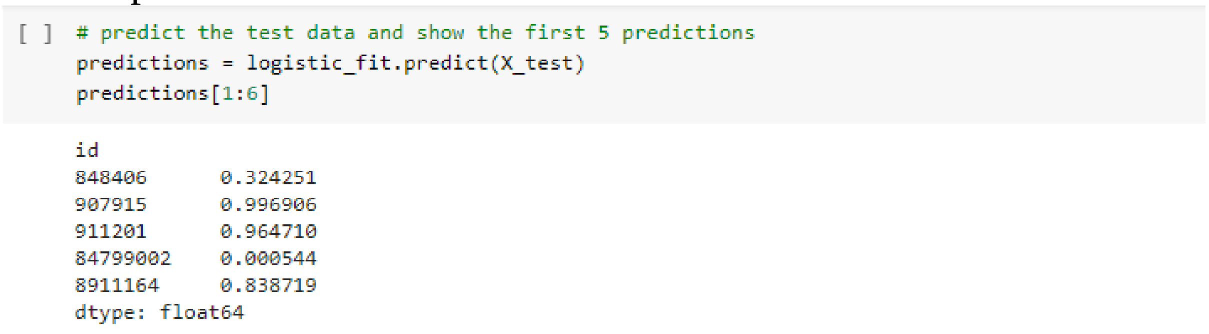


```
# Run the model and report the results
model = smf.glm(formula=formula, data=X_train, family=sm.families.Binomial())
logistic_fit = model.fit()

print(logistic_fit.summary())
```

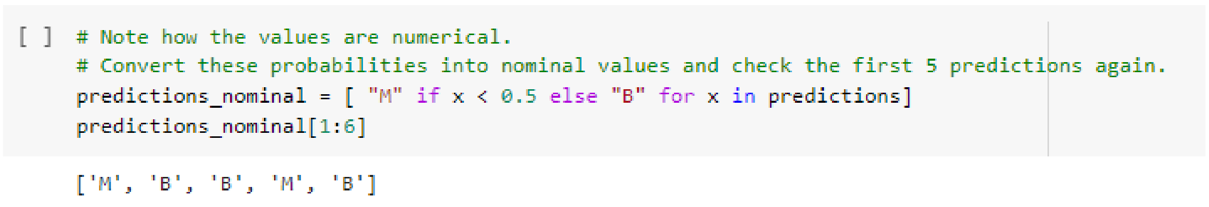
```
                 Generalized Linear Model Regression Results
==========================================================================================
Dep. Variable:     ['diagnosis[B]', 'diagnosis[M]']   No. Observations:                  398
Model:                                         GLM   Df Residuals:                      385
Model Family:                             Binomial   Df Model:                           12
Link Function:                               logit   Scale:                          1.0000
Method:                                       IRLS   Log-Likelihood:                -55.340
Date:                             Sun, 15 Aug 2021   Deviance:                       110.68
Time:                                     12:18:19   Pearson chi2:                     125.
No. Iterations:                                  9
Covariance Type:                         nonrobust
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                 44.5427     11.787      3.779      0.000      21.441      67.644
radius_mean               -1.1610      0.301     -3.862      0.000      -1.750      -0.572
texture_mean              -0.4237      0.087     -4.866      0.000      -0.594      -0.253
smoothness_mean          -85.3981     40.976     -2.084      0.037    -165.709      -5.088
compactness_mean         -16.7104     22.510     -0.742      0.458     -60.829      27.408
symmetry_mean            -46.2721     17.767     -2.604      0.009     -81.095     -11.449
fractal_dimension_mean   -49.1536    121.888     -0.403      0.687    -288.050     189.742
radius_se                 -7.1916      2.806     -2.563      0.010     -12.691      -1.692
texture_se                 0.1849      0.784      0.236      0.814      -1.353       1.722
smoothness_se            163.6068    159.702      1.024      0.306    -149.403     476.616
compactness_se           -31.1808     42.772     -0.729      0.466    -115.012      52.650
symmetry_se               74.7366     51.458      1.452      0.146     -26.119     175.592
fractal_dimension_se     824.1245    412.040      2.000      0.045      16.541    1631.708
==========================================================================================
```

Prediction

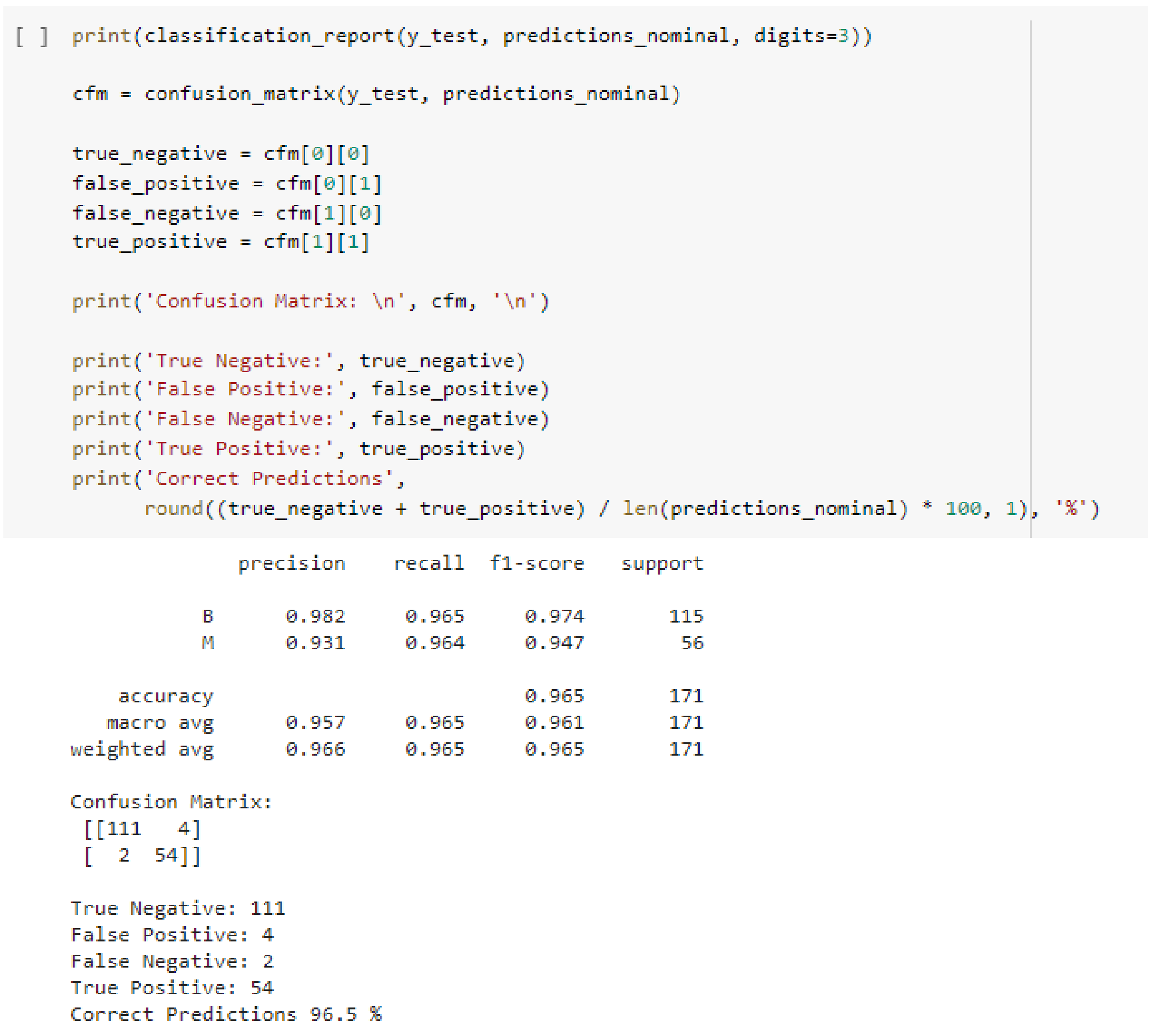
After we train the model, testing is done and based on the values that we get, we put it in the matrix (confusing matrix). The output is the correct prediction.



```
[ ] # predict the test data and show the first 5 predictions
    predictions = logistic_fit.predict(X_test)
    predictions[1:6]

    id
    848406      0.324251
    907915      0.996906
    911201      0.964710
    84799002    0.000544
    8911164     0.838719
    dtype: float64
```



```
[ ] # Note how the values are numerical.
    # Convert these probabilities into nominal values and check the first 5 predictions again.
    predictions_nominal = [ "M" if x < 0.5 else "B" for x in predictions]
    predictions_nominal[1:6]

    ['M', 'B', 'B', 'M', 'B']
```



```
[ ]  print(classification_report(y_test, predictions_nominal, digits=3))

     cfm = confusion_matrix(y_test, predictions_nominal)

     true_negative = cfm[0][0]
     false_positive = cfm[0][1]
     false_negative = cfm[1][0]
     true_positive = cfm[1][1]

     print('Confusion Matrix: \n', cfm, '\n')

     print('True Negative:', true_negative)
     print('False Positive:', false_positive)
     print('False Negative:', false_negative)
     print('True Positive:', true_positive)
     print('Correct Predictions',
           round((true_negative + true_positive) / len(predictions_nominal) * 100, 1), '%')
```

```
              precision    recall  f1-score   support

           B      0.982     0.965     0.974       115
           M      0.931     0.964     0.947        56

    accuracy                          0.965       171
   macro avg      0.957     0.965     0.961       171
weighted avg      0.966     0.965     0.965       171

Confusion Matrix:
 [[111   4]
 [  2  54]]

True Negative: 111
False Positive: 4
False Negative: 2
True Positive: 54
Correct Predictions 96.5 %
```

MATLAB CODE

```
clc;
clf;
clear all;
```

Importing the dataset from the excel sheet

```
[B,text] = xlsread('data.xlsx');
```

Initialising the variable, 'Data' which basically has the extracted values of the dataset from the 3rd column

```
Data = B(:,3:end);
```

Initialising the variable, 'diagnosis' which basically has the extracted values of the dataset of the 2nd column which basically gives us the information wheather a person has cancer or not

```
diagnosis = B(:,2);
```

We perform Principle component analysis (PCA) using Singular Value Decomposition (SVD)

```
[U,S,V] = svd(Data,'econ');
```

econ - economy size decoposition

If data matrix is mxn

If m is less -> it returns mxm matrix

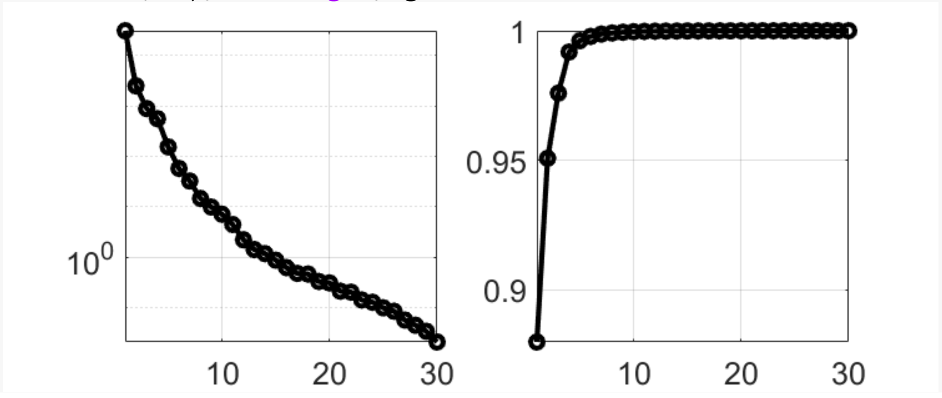
If n is less -> it returns nxn matrix

If m = n -> it returns mxn matrix

```
figure(1)
subplot(1,2,1)
semilogy(diag(S),'k-o','LineWidth',2.5)
```

Plotting the singular values on a log scale -> x axis - (number of columns (rank)), y axis - magnitude of log of the singular value

```
set(gca,'FontSize',15),axis tight, grid on
subplot(1,2,2) % to find the amount of data captured
plot(cumsum(diag(S))./sum(diag(S)),'k-o','LineWidth',2.5)
set(gca,'FontSize',15),axis tight, grid on
```



```
set(gcf,'Position',[1400 100 600 250])

figure(2)
hold on
```

Taking 2 eigen genetic sequences (cancer or no cancer)

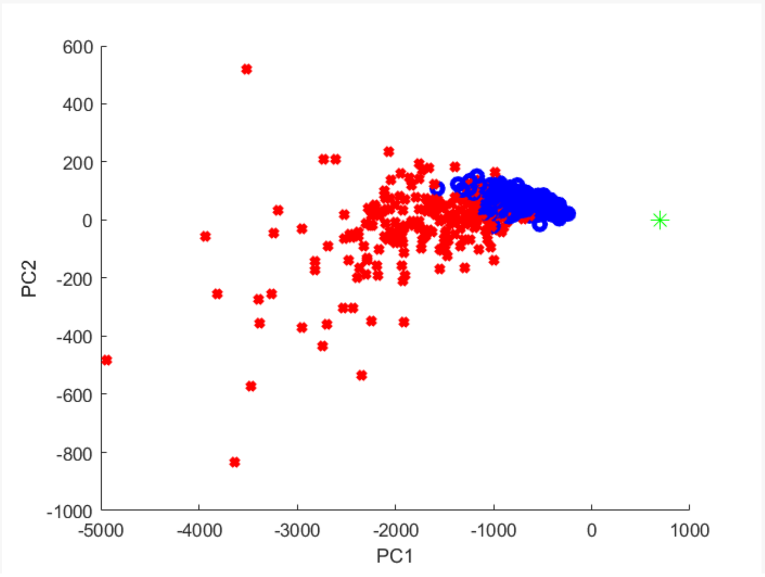
```
for i=1:size(Data,1)
     x = V(:,1)' * Data(i,:)';
     y = V(:,2)' * Data(i,:)';

     PC1(i) = x;
     PC2(i) = y;


     if (diagnosis(i) == 1)
         plot(x,y,'rx','LineWidth',3);
     else
         plot(x,y,'bo','LineWidth',3);
     end
end
xlabel('PC1'), ylabel('PC2')
```

A new dataset having the values recorded for one new patient to predict whether the person has cancer or not

```
NewData=[11.42, 20.38, 77.58, 386.1, 0.1425, 0.2839, 0.2414, 0.1052, 0.2597, 
0.09744, 0.4956, 1.156, 3.445, 27.23, 0.00911, 0.07458, 0.05661, 0.01867, 
0.05963, 0.009208, 14.91, 26.5, 98.87, 567.7, 0.2098, 0.8663, 0.6869, 0.2575, 
0.6638, 0.173];
```

We perform Principle component analysis (PCA) using Singular Value Decomposition (SVD) for the new data

```
[U1,Z1,V1] = svd(NewData);
x = V1(:,1)' * NewData(1,:)';
y = V1(:,2)' * NewData(1,:)';
plot(x, y, 'g*', 'MarkerSize', 10)
```



CONCLUSION

The method proposed in this study, can be used to effectively predict breast cancer incidents in patients.
On analyzing the data, we were able to verify the presence of multicollinearity between some of our variables. It was observed that the radius_mean column has a correlation of 1 and 0.99 with perimeter_mean and area_mean columns , respectively.
Using PCA,we reduced the large data and obtained two clusters each indicating patients either having cancer or not respectively thus making analysis easier. The cumulative proportion of the top two major components was 95% Now when we plotted the data of the new patient, we were able to predict the possibility whether the patient is suffering from cancer or not.
The use of logistic regression helped us to create a model that would predict the presence of cancerous cells in the patient by using training and testing datasets and have achieved an accuracy of 96.5% that is, our model have accurately labeled 96.5% of the test data. To obtain a higher accuracy rate, various other algorithms other than logistic regression such as k-means, SVM and kNN can be implemented.

LITERATURE SURVEY AND BIBLIOGRAPHY

- Huan-Jung Chiu, Tzuu-Hseng S.Li , and Ping-Huan Kuo, "Breast Cancer–Detection System Using PCA, Multilayer Perceptron, Transfer Learning, and Support Vector Machine", November 10, 2020
- **https://www.youtube.com/watch?v=VqjJ5YYt78Y**
- **https://www.kaggle.com/jagannathrk/predicting-breast-cancer-logistic-regression**
- **https://towardsdatascience.com/conceptual-vs-inbuilt-principal-component-analysis-for-breast-cancer-diagnosis-7ec3a8455201**
- **https://dsp.stackexchange.com/questions/36908/pca-to-reduce-dimensionality-to-99-variance**