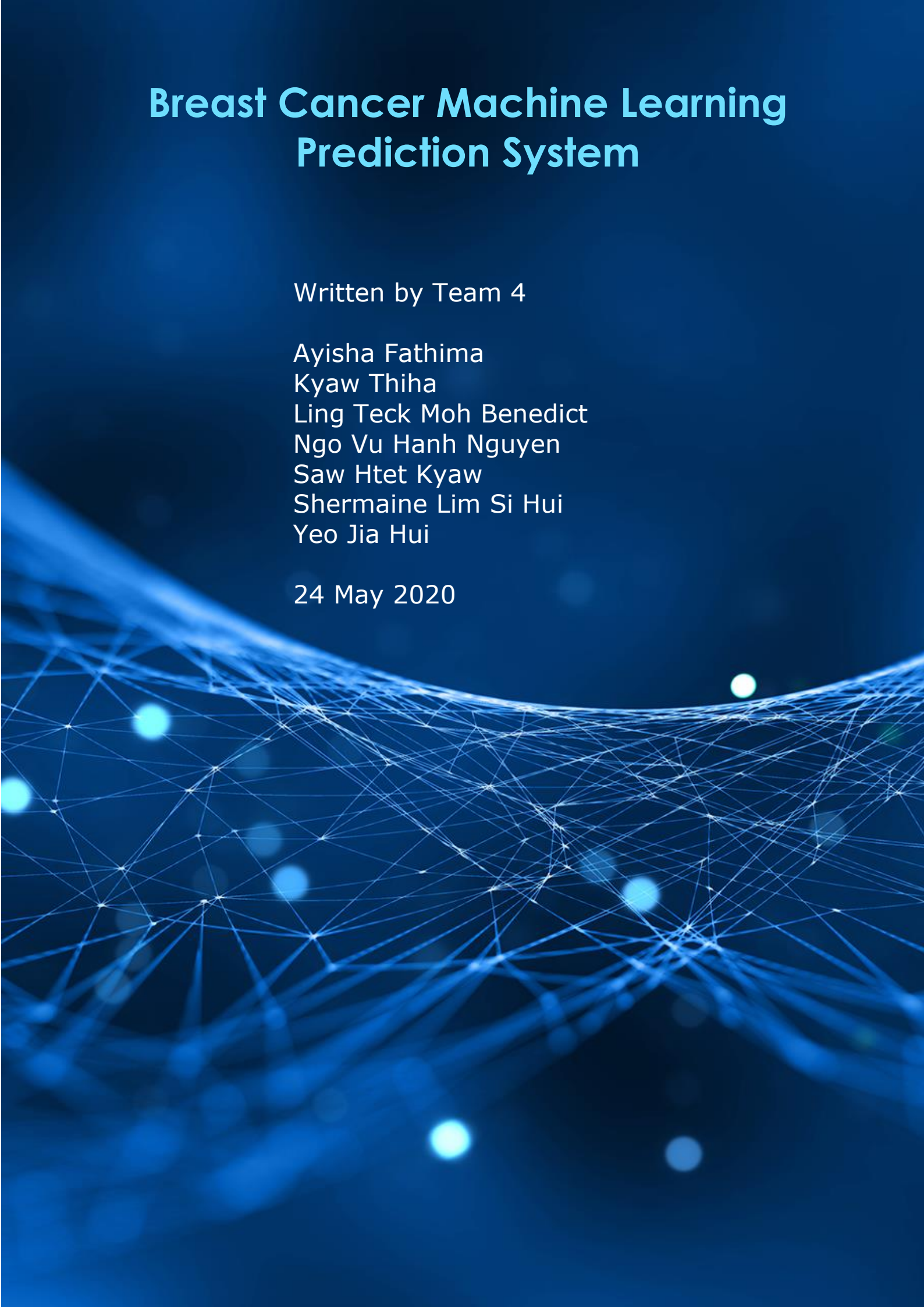
Breast Cancer Machine Learning Prediction System

Written by Team 4

Ayisha Fathima
Kyaw Thiha
Ling Teck Moh Benedict
Ngo Vu Hanh Nguyen
Saw Htet Kyaw
Shermaine Lim Si Hui
Yeo Jia Hui

24 May 2020



Contents

Abstract

In Part A, we will use linear regression. The data has been experimented with 2 types of linear regression model: Multiple Linear Regression and Polynomial Regression. In Part B and C, we will use a different dataset which is on Breast Cancer data to apply to supervised and unsupervised machine learning models. Breast cancer is the second highest cause of death from cancer among women, in the case of United States. Breast cancer occurs as a result of tumours in the breast tissue. The intention of this study is to design an accurate prediction system through various machine learning models of Python to predict whether a tumour is malignant or benign. The usage of machine learning and deep learning techniques in healthcare analysis and diagnostics is growing rapidly. Most of the possible medical flaws can be minimized by using **classification models** that also allow the data to be analysed faster, more accurately and more comprehensively (Kumari and Singh,2018).

Abbreviations

AUC- Area under curve

KNN- K-Nearest Neighbours

RMSE- Root Mean Square Error

ROC- Receiver operating characteristic

PCA- Principal Component Analysis

RFECV- Recursive feature elimination with cross-validation

Classification and Regression (Identifying datasets to be a classification or regression task)

There are two major types of supervised machine learning problems, called classification and regression. The aim in classification is to predict a class label, which is a choice from a predefined list of possibilities. The goal in regression is to predict continuous number. To distinguish between classification and regression is to analyse whether there is some form of continuity in the output. If there is continuity between possible outcomes, then it is a regression problem (MÜLLER and Guido, 2017)

.

Major Types of Supervised Learning

Regression

- To predict a continuous number

Classification

- To predict a class label

For example, in our Part A) predicting house prices shows clear continuity in the output and whether the algorithm predicts $29,999 or $30,001 when it should be $30,000 doesn't really matter therefore it is clear this is a regression task. On the other hand, for example, in our Part B), the task of recognizing the type of tumour as "Benign" or "Malignant" is a classification task as there is no matter of degree or continuity.

Supervised (Part B) versus Unsupervised Learning (Part C)

The difference between supervised and unsupervised learning is that for supervised learning, all data is labelled and the algorithms learn to predict the output from the input data while unsupervised learning is where all the data is unlabelled and the algorithms learn to discover the structure and categories from the input data (Brownlee, 2016) .

Therefore, for Part B, we will apply supervised learning where we test out on various supervised learning models to predict how well the models can distinguish Benign from Malignant tumours from the provided dataset with the diagnosis column input containing Benign and Malignant type tumours. For Part C, we will use unsupervised learning for the machine learning where the datapoints which has already been categorized into benign and malignant, so we will use the data with clustering algorithms to see if the model can identify subcategories within the benign and malignant categories.

Part A) Linear Regression

Data Source

We chose to predict housing price and we have picked a dataset from Kaggle. The dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. Data Source Link : https://www.kaggle.com/harlfoxem/housesalesprediction

Problem Statement

We want to predict The King County's home prices based on a variety of attributes such as bedrooms, bathrooms and so on, etc.

Solution Statement

As the dataset has many attributes, we will use multiple regression so that we can compare multiple X variables against a Y variable (price) and improve the multiple regression model by rebuilding after analysing the p-value. After trying the multiple linear regression, we further improve the scores by using Polynomial Regression model which is another form of linear regression model.

KC House Data Dictionary

1. Number of instances: 21,613
2. Number of attributes: 21

Attributes	Type	Definition
Id	Int64	Unique ID of house
Date	Object	Date
Price	Float64	Price of house (USD)
Bedrooms	Int64	The number of bedrooms in a house
Bathrooms	Float64	The number of bathrooms in a house
Sqft_living	Int64	Square feet living area
sqft_lot	Int64	Square feet lot
floors	Float64	The number of floors in a house
waterfront	Int64	Whether the house is near waterfront
view	Int64	The number of views
condition	Int64	The level of condition of the house
grade	Int64	Grade of the house
sqft_above	Int64	In real estate, above grade means the portion of a home that is above the ground
sqft_basement	Int64	Square feet basement
yr_built	Int64	The year when the house was built
yr_renovated	Int64	The year when the house was renovated
zipcode	Int64	Zipcode
lat	Float64	Latitude
long	Float64	Longitude
sqft_living15	Int64	Square feet living 15
sqft_lot15	Int64	Square feet lot 15

Multiple Linear Regression is the most common model for linear regression analysis. Multiple Linear Regression is a form of predictive analysis that is used to explain the relation between one continuous dependent variable(in this case, y = price) , and two or more independent variables (in this case, x= all the other variables after dropping data and ID and excluding price).

Code Guide for Multiple Linear Regression

The code contains headings where we explain what we do for each section of the code.

MultipleLinearRegression-Kyaw_Shermaine

Import Libraries

In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Read File

In [2]:
```
house = pd.read_csv("C:/Users/Shermaine/Documents/Python ML II (Cher Wah)/kc_house_data.csv")

df = pd.DataFrame(house)
```

Data Visualization

In [3]:
```
df
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	

21613 rows × 21 columns

maine.ipynb#

Data Cleaning

Remove ID and Date (Redundant) They can't be calculated.

In [4]:
```
#removed date
df = df.drop(['id','date'],axis=1)

print(df)

x = df.drop(['price'],axis=1)
y = df[['price']]
```

```
          price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
0      221900.0         3       1.00         1180      5650     1.0
1      538000.0         3       2.25         2570      7242     2.0
2      180000.0         2       1.00          770     10000     1.0
3      604000.0         4       3.00         1960      5000     1.0
4      510000.0         3       2.00         1680      8080     1.0
...         ...       ...        ...          ...       ...     ...
21608  360000.0         3       2.50         1530      1131     3.0
21609  400000.0         4       2.50         2310      5813     2.0
21610  402101.0         2       0.75         1020      1350     2.0
21611  400000.0         3       2.50         1600      2388     2.0
21612  325000.0         2       0.75         1020      1076     2.0

       waterfront  view  condition  grade  sqft_above  sqft_basement  \
0               0     0          3      7        1180              0
1               0     0          3      7        2170            400
2               0     0          3      6         770              0
3               0     0          5      7        1050            910
4               0     0          3      8        1680              0
...           ...   ...        ...    ...         ...            ...
21608           0     0          3      8        1530              0
21609           0     0          3      8        2310              0
21610           0     0          3      7        1020              0
21611           0     0          3      8        1600              0
21612           0     0          3      7        1020              0

       yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0          1955             0    98178  47.5112 -122.257           1340
1          1951          1991    98125  47.7210 -122.319           1690
2          1933             0    98028  47.7379 -122.233           2720
3          1965             0    98136  47.5208 -122.393           1360
4          1987             0    98074  47.6168 -122.045           1800
...         ...           ...      ...      ...      ...            ...
21608      2009             0    98103  47.6993 -122.346           1530
21609      2014             0    98146  47.5107 -122.362           1830
21610      2009             0    98144  47.5944 -122.299           1020
21611      2004             0    98027  47.5345 -122.069           1410
21612      2008             0    98144  47.5941 -122.299           1020

       sqft_lot15
0            5650
1            7639
```

Train_Test_Split= 80/20 and Time Taken

In [5]:
```
import time

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=19)

lr = LinearRegression()

start_time=time.time()

model = lr.fit(x_train, y_train)

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))

y_predict= lr.predict(x_test)
```

```
---0.23113656044006348 seconds ---
```

Plot and Print

In [6]:
```
print("Train score:")
print(lr.score(x_train, y_train))

print("Test score:")
print(lr.score(x_test, y_test))

plt.scatter(y_test, y_predict)

plt.plot(range(2000000), range(2000000),color='red')


plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Rent vs Predicted Rent")

plt.show()

print(lr.coef_)
```

```
Train score:
0.6991423015412856
Test score:
0.7008083566026718
```


```
[[-3.35294641e+04  4.07327629e+04  1.05222109e+02  1.58103951e-01
   6.52047359e+03  6.06570370e+05  5.40751847e+04  2.77180088e+04
   9.63071796e+04  7.04695529e+01  3.47525562e+01 -2.60004489e+03
   1.95994527e+01 -5.89049919e+02  6.03190978e+05 -2.18376207e+05
   2.16581949e+01 -3.64617240e-01]]
```

Import Libraries

In [7]:
```
# import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats
```

P value

We will observe the p-value from the OLS summary results and later rebuild the model to improve the scores by removing p-values above 0.005 from x variables. An explanation on what p-value and coefficients are written inside the code.

Print Summary of OLS Regression Results and Observe P-value. We will remove P-values above 0.005 when we rebuild the model later to improve the scores. ¶

In [8]:
```
X2 = sm.add_constant(x)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.700
Model:                            OLS   Adj. R-squared:                  0.700
Method:                 Least Squares   F-statistic:                     2961.
Date:                Sat, 23 May 2020   Prob (F-statistic):               0.00
Time:                        01:00:57   Log-Likelihood:            -2.9462e+05
No. Observations:               21613   AIC:                         5.893e+05
Df Residuals:                   21595   BIC:                         5.894e+05
Df Model:                          17
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          6.677e+06   2.93e+06      2.276      0.023    9.27e+05    1.24e+07
bedrooms       -3.58e+04   1893.037    -18.910      0.000   -3.95e+04   -3.21e+04
bathrooms      4.117e+04   3255.732     12.644      0.000    3.48e+04    4.75e+04
sqft_living     110.5125      2.271     48.662      0.000     106.061     114.964
sqft_lot          0.1284      0.048      2.678      0.007       0.034       0.222
floors         6695.1575   3598.129      1.861      0.063    -357.441    1.37e+04
waterfront      5.83e+05   1.74e+04     33.563      0.000    5.49e+05    6.17e+05
view           5.293e+04   2141.405     24.719      0.000    4.87e+04    5.71e+04
condition      2.641e+04   2352.946     11.225      0.000    2.18e+04     3.1e+04
grade          9.599e+04   2154.148     44.559      0.000    9.18e+04      1e+05
sqft_above        70.8285      2.255     31.412      0.000      66.409      75.248
sqft_basement     39.6882      2.648     14.987      0.000      34.497      44.879
yr_built       -2622.4105     72.705    -36.069      0.000   -2764.918   -2479.903
yr_renovated      19.8242      3.658      5.420      0.000      12.654      26.994
zipcode         -582.5717     33.007    -17.650      0.000    -647.267    -517.876
lat            6.028e+05   1.07e+04     56.121      0.000    5.82e+05    6.24e+05
long           -2.15e+05   1.31e+04    -16.357      0.000   -2.41e+05   -1.89e+05
sqft_living15     21.6758      3.450      6.283      0.000      14.914      28.438
sqft_lot15        -0.3825      0.073     -5.217      0.000      -0.526      -0.239
==============================================================================
Omnibus:                    18359.449   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1856934.171
Skew:                           3.560   Prob(JB):                         0.00
Kurtosis:                      47.848   Cond. No.                     7.14e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.3e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Find Mean_Squared Error, R_Square Value and Root of Mean_Square Error

In [10]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_predict)
r_squared = r2_score(y_test, y_predict)
```

In [11]:
```
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 45340773879.049095
r_square_value : 0.7008083566026718
```

In [12]:
```
from math import sqrt

rms = sqrt(mse)
rms
```

Out[12]: 212933.73119130067

R Square and RMSE

A well-fitting regression model generates results in predicted values that are close to the observed data values. R-squared is useful and range from 0-1. 0 will mean that the proposed model does not improve prediction over the mean model, while 1 indicates perfect prediction. An improvement in the regression model will also increase the R-squared in proportion (The Analysis Factor, 2008).

We also use RMSE which is the square root of the variance of the residuals to observe the results. RMSE indicates the absolute fit of the model of how close the observed data points are to the model's predicted values. While R-squared is a relative measure of fit, RMSE is an absolute measure of fit. RMSE is also good to see how accurately the model predicts the results and it is the most crucial criterion for fit if the main goal in the model is prediction which supports our objectives since our main objective is to predict house prices. Lower values of RMSE implies better fit (The Analysis Factor, 2008).

Rebuilding and Improving the Multiple Linear Regression Model

We improve the accuracy score by removing values based on P-value above 0.005 from previous summary. We will remove sqft_lot that has P-value of 0.007 and floors which has P-value of 0.063 as seen in the previous OLS results below.

OLS Regression Results

Dep. Variable:	price	R-squared:	0.700
Model:	OLS	Adj. R-squared:	0.700
Method:	Least Squares	F-statistic:	2961.
Date:	Sat, 23 May 2020	Prob (F-statistic):	0.00
Time:	01:00:57	Log-Likelihood:	-2.9462e+05
No. Observations:	21613	AIC:	5.893e+05
Df Residuals:	21595	BIC:	5.894e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.677e+06	2.93e+06	2.276	0.023	9.27e+05	1.24e+07
bedrooms	-3.58e+04	1893.037	-18.910	0.000	-3.95e+04	-3.21e+04
bathrooms	4.117e+04	3255.732	12.644	0.000	3.48e+04	4.75e+04
sqft_living	110.5125	2.271	48.662	0.000	106.061	114.964
sqft_lot	0.1284	0.048	2.678	0.007	0.034	0.222
floors	6695.1575	3598.129	1.861	0.063	-357.441	1.37e+04
waterfront	5.83e+05	1.74e+04	33.563	0.000	5.49e+05	6.17e+05
view	5.293e+04	2141.405	24.719	0.000	4.87e+04	5.71e+04
condition	2.641e+04	2352.946	11.225	0.000	2.18e+04	3.1e+04
grade	9.599e+04	2154.148	44.559	0.000	9.18e+04	1e+05
sqft_above	70.8285	2.255	31.412	0.000	66.409	75.248
sqft_basement	39.6882	2.648	14.987	0.000	34.497	44.879
yr_built	-2622.4105	72.705	-36.069	0.000	-2764.918	-2479.903
yr_renovated	19.8242	3.658	5.420	0.000	12.654	26.994
zipcode	-582.5717	33.007	-17.650	0.000	-647.267	-517.876
lat	6.028e+05	1.07e+04	56.121	0.000	5.82e+05	6.24e+05
long	-2.15e+05	1.31e+04	-16.357	0.000	-2.41e+05	-1.89e+05
sqft_living15	21.6758	3.450	6.283	0.000	14.914	28.438
sqft_lot15	-0.3825	0.073	-5.217	0.000	-0.526	-0.239

Omnibus:	18359.449	Durbin-Watson:	1.990
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1856934.171
Skew:	3.560	Prob(JB):	0.00
Kurtosis:	47.848	Cond. No.	7.14e+17

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.3e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Rebuild the Model

P- values help identify whether the relationships in the sample also exist in the larger population. The p-value for each independent variable tests the null hypothesis that the variable has no correlation with the dependent variable. If there is no correlation then there is no association between the changes in the independent variable and the shifts in the dependent variable. Therefore, there is insufficient evidence to conclude that there is effect at the population level (Frost,2017).

If the p-value for a variable is less than our signifance level, our sample data provides enough evidence to reject the null hypothesis for the entire population. Our data favor the hypothesis that there is non-zero correlation (Frost,2017)

On the other hand, a p-value that is greater than the significance level indicates that there is insffucient evidence in our sasmple to conclude that a non-zero correlation exists (Frost,2017)

Coefficients are most helpful in determining which independent variable carries more weight. For example, a coefficient of -1.345 will impact the rent more than a coefficient of 0.238, with the former impacting prices negatively and latter positively.

Remove Price, sqft_lot and floors

In [13]:
```
x = df.drop(['price', 'floors', 'sqft_lot'],axis=1)

print(x)

y = df[['price']]
```

```
       bedrooms  bathrooms  sqft_living  waterfront  view  condition  grade  \
0             3       1.00         1180           0     0          3      7
1             3       2.25         2570           0     0          3      7
2             2       1.00          770           0     0          3      6
3             4       3.00         1960           0     0          5      7
4             3       2.00         1680           0     0          3      8
...         ...        ...          ...         ...   ...        ...    ...
21608         3       2.50         1530           0     0          3      8
21609         4       2.50         2310           0     0          3      8
21610         2       0.75         1020           0     0          3      7
21611         3       2.50         1600           0     0          3      8
21612         2       0.75         1020           0     0          3      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
0            1180              0      1955             0    98178  47.5112
1            2170            400      1951          1991    98125  47.7210
2             770              0      1933             0    98028  47.7379
3            1050            910      1965             0    98136  47.5208
4            1680              0      1987             0    98074  47.6168
...           ...            ...       ...           ...      ...      ...
21608        1530              0      2009             0    98103  47.6993
21609        2310              0      2014             0    98146  47.5107
21610        1020              0      2009             0    98144  47.5944
21611        1600              0      2004             0    98027  47.5345
21612        1020              0      2008             0    98144  47.5941

          long  sqft_living15  sqft_lot15
0     -122.257           1340        5650
1     -122.319           1690        7639
2     -122.233           2720        8062
3     -122.393           1360        5000
4     -122.045           1800        7503
...        ...            ...         ...
21608 -122.346           1530        1509
21609 -122.362           1830        7200
21610 -122.299           1020        2007
21611 -122.069           1410        1287
21612 -122.299           1020        1357

[21613 rows x 16 columns]
```

Train_Test_Split= 80/20 and Time Taken

In [14]:
```
import time

x_train, x_test, y_train, y_test2 = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=19)

lr = LinearRegression()

start_time=time.time()

model = lr.fit(x_train, y_train)

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))

y_predict2= lr.predict(x_test)
```

```
---0.011967182159423828 seconds ---
```

Plot and Print

In [15]:
```
print("Train score:")
print(lr.score(x_train, y_train))

print("Test score:")
print(lr.score(x_test, y_test))

plt.scatter(y_test, y_predict)
plt.plot(range(2000000), range(2000000),color='red')

plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Rent vs Predicted Rent")

plt.show()

print(lr.coef_)
```

```
Train score:
0.6989415742237948
Test score:
0.7008815632093645
```


```
[[-3.37822023e+04  4.22410747e+04  1.05136583e+02  6.05441907e+05
   5.44236489e+04  2.73690807e+04  9.68445850e+04  7.23363171e+01
   3.28002658e+01 -2.58097575e+03  1.98137876e+01 -5.82600857e+02
   6.03316989e+05 -2.17267152e+05  2.01773889e+01 -2.03878103e-01]]
```

Upon printing OLS Regression Results, we now see that all p-values have become 0.

Print Summary of OLS Regression Results

In [17]:
```
X2 = sm.add_constant(x)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.700
Model:                            OLS   Adj. R-squared:                  0.699
Method:                 Least Squares   F-statistic:                     3353.
Date:                Sat, 23 May 2020   Prob (F-statistic):               0.00
Time:                        01:00:58   Log-Likelihood:            -2.9462e+05
No. Observations:               21613   AIC:                         5.893e+05
Df Residuals:                   21597   BIC:                         5.894e+05
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.095e+06   2.89e+06      2.112      0.035    4.39e+05    1.18e+07
bedrooms      -3.602e+04   1892.067    -19.039      0.000   -3.97e+04   -3.23e+04
bathrooms      4.277e+04   3144.355     13.602      0.000    3.66e+04    4.89e+04
sqft_living     110.3399      2.255     48.937      0.000     105.920     114.759
waterfront     5.823e+05   1.74e+04     33.522      0.000    5.48e+05    6.16e+05
view           5.322e+04   2140.012     24.868      0.000     4.9e+04    5.74e+04
condition      2.606e+04   2349.271     11.091      0.000    2.15e+04    3.07e+04
grade          9.644e+04   2145.844     44.944      0.000    9.22e+04    1.01e+05
sqft_above       72.6613      2.091     34.757      0.000      68.564      76.759
sqft_basement    37.6803      2.413     15.613      0.000      32.950      42.411
yr_built      -2602.2537     70.887    -36.710      0.000   -2741.197   -2463.310
yr_renovated     20.0405      3.653      5.486      0.000      12.880      27.201
zipcode        -576.8939     32.869    -17.551      0.000    -641.319    -512.469
lat            6.035e+05   1.07e+04     56.400      0.000    5.82e+05    6.24e+05
long          -2.146e+05   1.31e+04    -16.429      0.000   -2.4e+05   -1.89e+05
sqft_living15    20.3873      3.422      5.957      0.000      13.680      27.095
sqft_lot15       -0.2539      0.053     -4.787      0.000      -0.358      -0.150
==============================================================================
Omnibus:                    18286.453   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1826325.832
Skew:                           3.540   Prob(JB):                         0.00
Kurtosis:                      47.474   Cond. No.                     5.18e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 7.9e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Find Mean_Squared Error, R_Square Value and Square Root of Mean_Square Error

In [18]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse2 = mean_squared_error(y_test2, y_predict2)
r_squared2 = r2_score(y_test2, y_predict2)
```

In [19]:
```
print('Mean_Squared_Error :' ,mse2)
print('r_square_value :',r_squared2)
rms2 = sqrt(mse2)
rms2
```

```
Mean_Squared_Error : 45329679838.577866
r_square_value : 0.7008815632093645
```

Out[19]:
```
212907.67914421938
```

Polynomial Regression and Code Guide

Next, we can further improve by using polynomial regression model. The code guide also provides explanation on what is polynomial regression. Polynomial Regression a form of linear regression.

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x).

For degree=2, the linear modelis built. The mean squared error is calculated and r squared is found for training and testing.

Import Libraries

In [1]:
```
#importing numpy and pandas, seaborn

import numpy as np #linear algebra
import pandas as pd #datapreprocessing, CSV file I/O
import seaborn as sns #for plotting graphs
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

Read File

In [2]:
```
house = pd.read_csv("C:/Users\Shermaine/Documents/Python ML II (Cher Wah)/kc_house_data.csv")

df = pd.DataFrame(house)
```

Train_Test_Split= 80/20

In [3]:
```
train_data,test_data=train_test_split(df,train_size=0.8,random_state=3)

#same as in multiple linear regression
#x = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot','floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', '
#features1= ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot','floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above

features= df.drop(['id','date','price', 'floors', 'sqft_lot'],axis=1)
print(features)

features1=x = ['bedrooms', 'bathrooms', 'sqft_living', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement',
```

```
       bedrooms  bathrooms  sqft_living  waterfront  view  condition  grade  \
0             3       1.00         1180           0     0          3      7
1             3       2.25         2570           0     0          3      7
2             2       1.00          770           0     0          3      6
3             4       3.00         1960           0     0          5      7
4             3       2.00         1680           0     0          3      8
...         ...        ...          ...         ...   ...        ...    ...
21608         3       2.50         1530           0     0          3      8
21609         4       2.50         2310           0     0          3      8
21610         2       0.75         1020           0     0          3      7
21611         3       2.50         1600           0     0          3      8
21612         2       0.75         1020           0     0          3      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
0            1180              0      1955             0    98178  47.5112
1            2170            400      1951          1991    98125  47.7210
2             770              0      1933             0    98028  47.7379
3            1050            910      1965             0    98136  47.5208
4            1680              0      1987             0    98074  47.6168
...           ...            ...       ...           ...      ...      ...
21608        1530              0      2009             0    98103  47.6993
21609        2310              0      2014             0    98146  47.5107
21610        1020              0      2009             0    98144  47.5944
21611        1600              0      2004             0    98027  47.5345
21612        1020              0      2008             0    98144  47.5941

          long  sqft_living15  sqft_lot15
0     -122.257           1340        5650
1     -122.319           1690        7639
2     -122.233           2720        8062
3     -122.393           1360        5000
4     -122.045           1800        7503
...        ...            ...         ...
21608 -122.346           1530        1509
21609 -122.362           1830        7200
21610 -122.299           1020        2007
21611 -122.069           1410        1287
21612 -122.299           1020        1357

[21613 rows x 16 columns]
```

Train and Time Taken, Degree= 2

In [4]:
```
import time

polyfeat=PolynomialFeatures(degree=2)

start_time=time.time()

xtrain_poly=polyfeat.fit_transform(train_data[features1])

xtest_poly=polyfeat.fit_transform(test_data[features1])

poly=linear_model.LinearRegression()
poly.fit(xtrain_poly,train_data['price'])

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))

polypred=poly.predict(xtest_poly)
```

```
---0.19698071479797363 seconds ---
```

Print Results

In [5]:
```
print('PolymomialRegression')
mean_squared_error = metrics.mean_squared_error(test_data['price'], polypred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(poly.score(xtrain_poly, train_data['price']), 3))
print('R-squared (testing) ', round(poly.score(xtest_poly, test_data['price']), 3))
```

```
PolymomialRegression
Mean Squared Error (MSE)  162705.19
R-squared (training)  0.799
R-squared (testing)  0.794
```

R-squared testing: 0.794

Rebuild Model for degree=3.The mean squared error is calculated and r squared is found for training and testing.

Train and Time Taken, Degree= 3

In [6]:
```
import time

polyfeat=PolynomialFeatures(degree=3)

start_time=time.time()

xtrain_poly=polyfeat.fit_transform(train_data[features1])

xtest_poly=polyfeat.fit_transform(test_data[features1])

poly=linear_model.LinearRegression()
poly.fit(xtrain_poly,train_data['price'])

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))

polypred=poly.predict(xtest_poly)
```

```
---1.5800795555114746 seconds ---
```

Print Results

In [7]:
```
print('PolynomialRegressionRebuild')
mean_squared_error=metrics.mean_squared_error(test_data['price'],polypred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(poly.score(xtrain_poly, train_data['price']), 3))
print('R-squared (testing) ', round(poly.score(xtest_poly, test_data['price']), 3))
```

```
PolynomialRegressionRebuild
Mean Squared Error (MSE)  206770.69
R-squared (training)  0.826
R-squared (testing)  0.667
```

R-squared testing: 0.667

Table of Comparison and Observations – Linear Regression

Multiple Linear Regression and Polynomial Regression			
Original Multiple Linear Regression	Rebuild and Improved Multiple Linear Regression	Polynomial with Degree= 2	Polynomial with Degree= 3
Never remove p-value	Removed sqft_lot of p-value =0.007 and floors of p-value = 0.063	Degree= 2	Degree= 3
Test score: 0.7008083566026718	Test score: 0.7008815632093645	-	-
Time Taken: 0.2311365 6044006348 seconds	Time Taken: 0.0119671 82159423828 seconds	Time Taken: 0.19698071479797363 seconds ---	Time Taken: 1.5800795555114746 s econds
Mean_Squared_Error: 4 5340773879.049095	Mean_Squared_Error: 4 5329679838.577866	Mean Squared Error:16 2705.19	Mean Squared Error:20 6770.69
R_Square_value: 0.700 8083566026718	R_Square_value: 0.700 8815632093645	R_Square_value: 0.794	R_Square_value: 0.667
RMSE: 212933.731191 30067	RMSE: 212907.679144 21938	-	-
Comparison of both Multiple Linear Regression Models		Comparison of Polynomial Degree= 2 and Polynomial Degree = 3	
Observations: The improved multiple linear regression did better i n all the metrics recorded before upon the removal of variables with p-values above 0.005 compared t o the original multiple linear regression model witho ut the p-value removal. The improved multiple linear regression model had a slightly higher test score at 0.70088 compared to the original with a test score at 0.70080. The most significant was that the time taken for the improved linear regression was also nearly twice a s short compared to the original model. The improved multiple linear regression model had slightly lower mean squared error than original. The improved multiple linear regression had slightl y higher R square value and lower RMSE than the original.		Observations: Degree 2 performed better in all the metrics such as getting lower mean squared error, higher R square value and shorter time compared to Degree 3 and the all the differences in value comparison is quite significant.	

Therefore, by observing and removing relevant p-values of the x variables and rebuilding the model has shown improvement where test score and R square score is higher and has a lower Mean Squared Error and RMSE which is a better fit for our prediction.	

Multiple Linear Regression Versus Polynomial Regression
Final Observations Polynomial regression has advantages such as being able to fit a wide range of curvature, a broad range of function can also fit under it and it can generate the best approximation of the relationship between the dependent and independent variable (Introduction to Linear Regression and Polynomial Regression ,2019). A note is that the presence of outliers can affect the results of this nonlinear analysis but in this case scenario, is not applicable here. The most distinct difference between multiple linear regression and polynomial regression will the mean squared error. Polynomial Regression has significantly lower mean squared error values compared to that in the multiple linear regression. Polynomial Regression at degree = 2 gives us the best R-squared score of 0.79 and significantly lower mean square error which we conclude as the best solution out of multiple linear and polynomial regression.

Part B) Classification Techniques

Data Source

This is an analysis of the Breast Cancer Wisconsin (Diagnostic) Data Set, obtained from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic). The data set contains 569 observations across 32 attributes. The attribute information includes ID number, Diagnosis (M= malignant, B= benign) and 30 real-valued numerical variables derived from ten features that are computed for each cell nucleus.

The ten features of the cell nucleus consist of:

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

Picture 1: *Attribute Information, taken from:*
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Data Dictionary for Breast Cancer Wisconsin (Diagnostic) Dataset

Range Index: 569 entries, 0 to 568

Data columns (total 33 columns):

Attributes	Type	Definition
id	Int64	Unique code of an instance
diagnosis	Object	Diagnosis (M = malignant, B = benign)
radius_mean	Float64	mean of distances from centre to points on the perimeter
texture_mean	Float64	mean of standard deviation of gray-scale values
perimeter_mean	Float64	mean of perimeter
area_mean	Float64	mean of area
smoothness_mean	Float64	mean of local variation in radius lengths
compactness_mean	Float64	mean of perimeter^2 / area - 1.0
concavity_mean	Float64	mean of severity of concave portions of the contour
concave points_mean	Float64	mean of number of concave portions of the contour
symmetry_mean	Float64	mean of symmetry
fractal_dimension_mean	Float64	mean of ("coastline approximation" - 1)
radius_se	Float64	standard error of distances from centre to points on the perimeter
texture_se	Float64	standard error of standard deviation of gray-scale values
perimeter_se	Float64	standard error of perimeter
area_se	Float64	standard error of area
smoothness_se	Float64	standard error of local variation in radius lengths
compactness_se	Float64	standard error of perimeter^2 / area - 1.0
concavity_se	Float64	standard error of severity of concave portions of the contour
concave points_se	Float64	standard error of number of concave portions of the contour
symmetry_se	Float64	standard error of symmetry

fractal_dimension_se	Float64	standard error of ("coastline approximation" - 1)
radius_worst	Float64	worst of distances from centre to points on the perimeter
texture_worst	Float64	worst of standard deviation of gray-scale values
perimeter_worst	Float64	worst of perimeter
area_worst	Float64	worst of area
smoothness_worst	Float64	worst of local variation in radius lengths
compactness_worst	Float64	worst of perimeter^2 / area - 1.0
concavity_worst	Float64	worst of severity of concave portions of the contour
concave points_worst	Float64	worst of number of concave portions of the contour
symmetry_worst	Float64	worst of symmetry
fractal_dimension_worst	Float64	worst of ("coastline approximation" - 1)
Unnamed: 32	Float64	NaN

Main Objectives

The analysis aims to identify which features are best in predicting malignant or benign cancer to help us in model selection. The main goal is to correct classify whether the breast tumour is benign or malignant. The more critical objective will be to minimize wrong predictions for malignant tumours than benign tumours as wrongly predicting a malignant tumour as a benign tumour holds heavier penalties (also known as Type II error) because if you misdiagnose a "Malignant" tumour, which is aggressive and dangerous tumour as a "Benign" which is harmless tumour then the person will think he/she is safe when in fact his/her health is in danger due to aggressive tumour. Then thinking it is a harmless tumour and not proceed for further treatment will eventually cause health to deteriorate at a faster rate due to the aggressive nature of the "Malignant tumour".

Solution Statement

To find the best solution, we first do data engineering to clean the data, then apply feature engineering or PCA to experiment with feature scaling and dimensionality to through different algorithms that will be tested on various hypotheses(models).Based on the objectives, the most appropriate model is chosen based on metrics such as accuracy, ROC, AUC, precision, recall, F1 score and error rate.

Data Exploration, Cleaning and Engineering

The data is first explored before implementing data cleaning and engineering. The first feature, ‘ID ‘, is irrelevant and will be removed. There is also a column with all NaN values which we do not want as it will affect the machine learning models. Therefore, we do data cleaning by removing these 2 columns.

Feature Selection/ Engineering

Feature selection is the method/process of reducing the number of input variables to those that can be most useful in predicting the target variable when developing a predictive model. They help by choosing features that will give better accuracy while requiring less data (Brownlee,2019). It is one of the core concepts in machine learning which impacts hugely on the performance of the machine learning models. The features that is used to train the machine learning models have a huge influence on the performance. Feature selection is the process where one can automatically or manually select those features which contribute most to the prediction variables or output. Having irrelevant/redundant features in the data can decrease the accuracy of the machine learning models as the model learns based on irrelevant/irrelevant features.

Benefits of Feature Selection/Engineering

Feature selection reduces overfitting, improves accuracy and reduces training time. Less irrelevant/redundant data will mean less opportunity to make decisions based on noise. Less misleading data also improves the model's accuracy. Having fewer data points also reduces algorithm complexity and algorithms can train faster.

Feature Selection Techniques Used

We will be discussing the pros and cons between Filter methods and Wrapper methods. The filter method we have selected is Pearson Correlation and the wrapper method will be Recursive Feature Elimination with additional Cross Validation (RFECV) added. The importance of feature selection comes in especially when dealing with a dataset with large number of features. This type of dataset will be known as a high dimensional dataset and with high dimensionality will significantly increase the training time of the machine learning model, increasing complexity and possibly overfitting. In a high dimensional feature set, there are often some features that are redundant and merely extensions of other essential features so extracting the most important and relevant features for a dataset is important to obtain the most effective predictive modelling performance. RFECV has limitations and one will be it does not work for all machine learning models, for example in this case scenario, it can be applied to the Logistics Regression model but not the KNN Classification model, therefore we will not use RFECV as a comparison due to lack of compatibility among machine learning models but instead use Pearson Correlation for the supervised learning models.

Filter method and Pearson Correlation

Filter method uses the general uniqueness of the dataset for evaluation and pick feature subset, without any mining algorithm. The filter method uses exact assessment criterion such as distance, information, dependency, consistency. Utilizing principal criteria of ranking technique and rank ordering for variable selection, this simple method is popular for filtering irrelevant features and yielding excellent and relevant features before classification process is initiated. The selection of features is independent of any machine learning model and algorithm. Features are ranked based on statistical score that determine the feature's correlation with the outcome variable (Feature Selection in Python, 2020). Diagram 1 shows an example of how filter method works.

One common and simple filtering method done using correlation matrix is the Pearson Correlation, which measures linear correlation between two variables. The resulting value lies in [-1:1], a -1 means perfect negative correlation (as one variable decreases, the other increases), +1 will mean it has perfect positive correlation and 0 indicates there is no linear correlation between the two. Features with high correlation are more linearly dependent, meaning that feature is almost the same as the other feature so if both features have high correlation, one of them is dropped from the feature set variables (Diving into data, 2014).

Diagram 1 – Filter Method, Source: Analytics Vidhya

Wrapper method and RFECV

A wrapper method **needs to use one machine learning model algorithm** and uses its performance as evaluation criteria. This method will search for a feature that is most suitable for the machine learning model algorithm and strives to improve mining performance. The features are evaluated through predictive accuracy utilised for classification tasks and cluster evaluation (Diving into data, 2014). Diagram 2 displays how wrapper method works.

An example of a wrapper method will be the REF which can be further cross-validated for selection of best number of features according to the machine learning model algorithm. The Recursive Feature Elimination (RFE) method performs by recursively removing attributes and building a model based on attributes left. It utilises accuracy metric for ranking features according to their importance. RFE takes machine learning model algorithm and the number of required features as input then generates a ranking of all the variables, with 1 being the most important *(Feature Selection with sklearn and Pandas* ,2019). RFECV uses feature ranking of RFE with additional cross-validated selection of the best number of features (scikit-learn 0.23.1 documentation ,2020).

Therefore, RFE can generate the best features based on how many features you want while RFECV will generate optimal number of features and the best features in rank for that machine learning model algorithm. E.g. For 30 features in our file, if we apply RFE and only want the best 5 features, we can input that to RFE and RFE will generate only the best 5 features for that model algorithm. For RFECV, it generates an optimal number (e.g. 11 features out of 30 features) and rank the feature in order of importance.

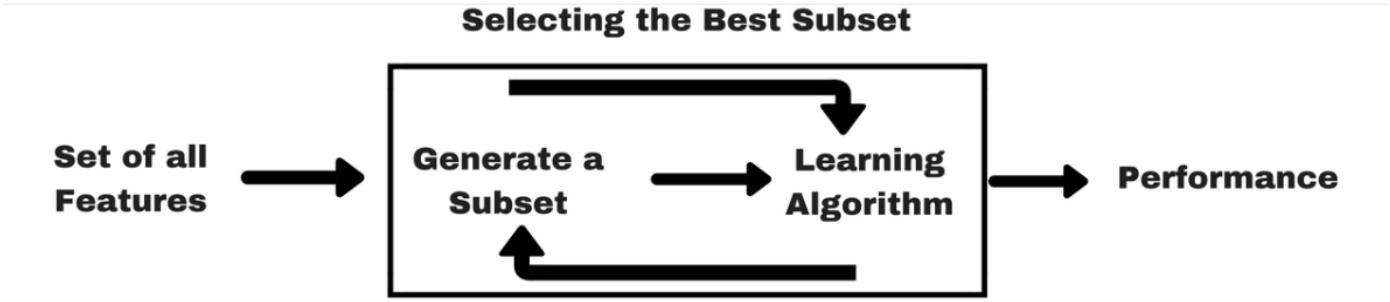


Diagram 2 – Filter Method, Source: Analytics Vidhya

Filter versus Wrapper Method

Filter methods are more flexible as they do not need to use a machine learning model first to determine if a feature is good or bad, but a wrapper method requires a machine learning model to train it to see if the feature is important or not. Filter methods are also faster than wrapper method as they not involve training models. Wrapper methods are computationally expensive and may not be the most efficient feature selection method especially when dealing with massive datasets. When there is insufficient data to model the statistical correlation of features, filter methods may fail to find the best subset of features, but wrapper methods are always able to generate the best subset of features with their exhaustive nature. Using features from wrapper methods in the final machine learning can lead to overfitting since the method has

already trained the machine learning models with the features, affecting the learning. However, features from filter methods will not lead to overfitting for most case scenarios (Feature Selection in Python, 2020).

Since the wrapper method is not applicable to all machine learning models for supervised learning, which in this case, as discussed, RFE and RFECV may be applicable for Logistics Regression model but not for the KNN Classification Model, the simplicity, sufficient data and speed of filter method like Pearson Correlation is more suitable for this Breast Cancer dataset for a fair comparison against the classification models.

Dimension Reduction and PCA

Feature selection should not be confused with dimension reduction. Both methods tend to reduce the number of attributes in dataset but dimension reduction method will do so by creating new combinations of attributes (or feature transformation), versus feature selection where include and exclude attributes are already present in the dataset without transforming to new features (Feature Selection in Python, 2020). For Part C, we will apply a dimension reduction method- Principal Component Analysis (PCA) to unsupervised learning models.

Data Engineering and Feature Selection (with Pearson Correlation to 16 Features)

The following code will reduce the original 30 features of the dataset into 16 optimal features.

Import Libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualization library
import matplotlib.pyplot as plt

import time
from subprocess import check_output
```

Read Dataset

In [2]:
```
data = pd.read_csv('C:/Users/Shermaine/Documents/Python ML II (Cher Wah)/caknn/fs/breast_cancer_30.csv')
```

Before making anything like feature selection,feature extraction and classification, firstly we start with basic data analysis. Lets look at features of data.

In [3]:
```
data.head()  # head method show only first 5 rows
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

Data Engineering - Cleaning

The first feature, 'ID ' is irrelevant and will be removed. There is also a column with all NaN values which we do not want as it will affect the machine learning models. Therefore, we do data cleaning by removing these 2 columns.

In [4]:
```
# feature names as a list
col = data.columns       # .columns gives columns names in data
print(col)
```

```
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')
```

In [5]:
```
# y includes our labels and x includes our features
y = data.diagnosis                          # M or B
list = ['Unnamed: 32','id','diagnosis']
x = data.drop(list,axis = 1 )
x.head()
```

Out[5]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dir
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

Normalization/ Standardization before Visualization, Feature Selection or Classification

The describe functions shows helpful information such as variance, standart deviation, number of sample (count) or max min values. Such information helps to understand about what is going on data. For example , "area_mean" feature's max value is 2501 while "smoothness_mean" features' max is around 0.16340. Thus we need to normalize/standardize the data first before before visualization, feature selection, feature extraction or classificaiton.

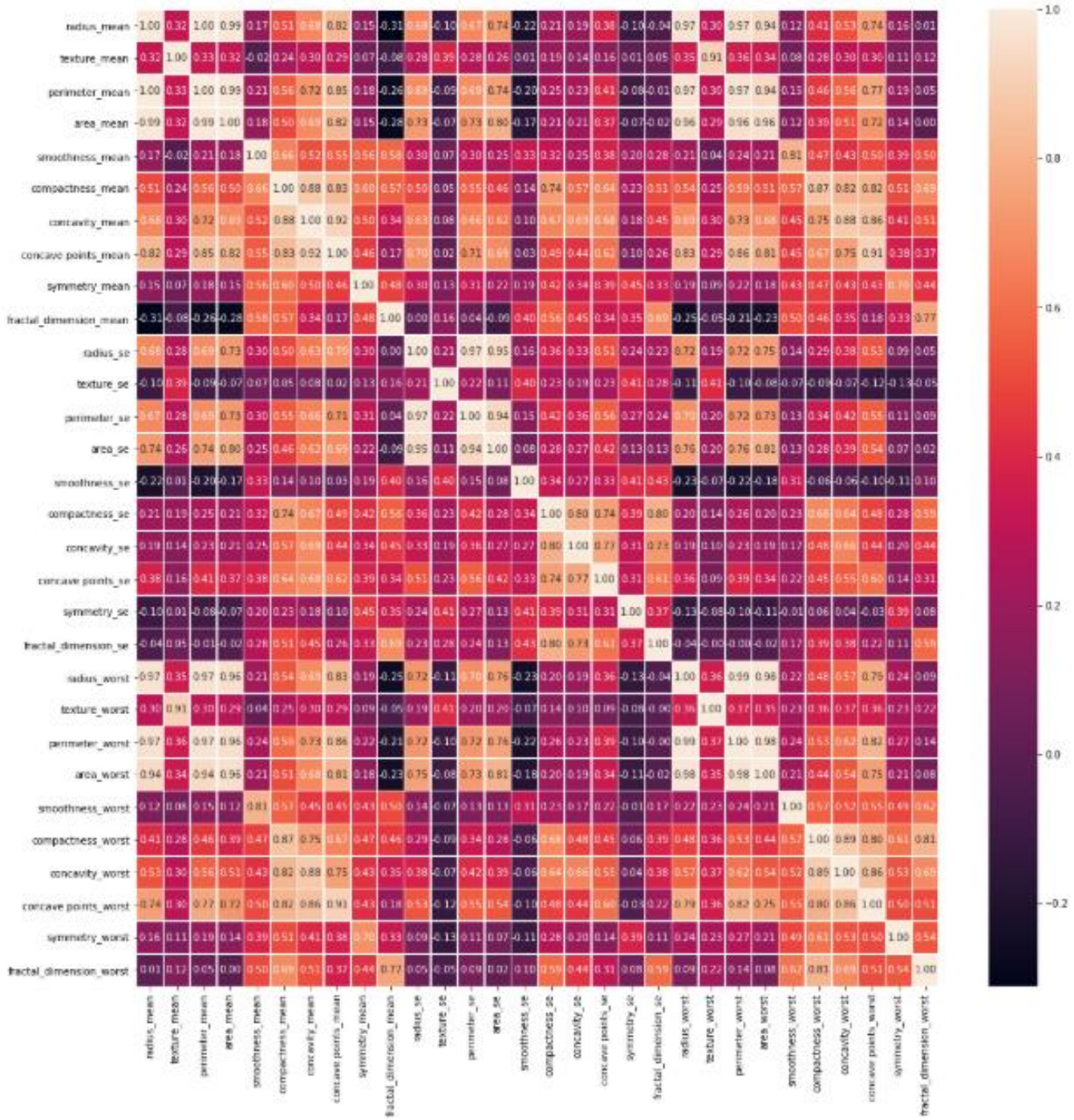
In [11]:
```
data = x
# standardization
data_n_2 = (data - data.mean()) / (data.std())
```

Visualization of Correlation Map

We observe all correlation between features through the heatmap.

In [12]:
```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
corr_mat = x.corr()
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt= '.2f',ax=ax)
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d8f2a7a1f0>
```


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
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Based on Pearson Correlation Matrix, a lot of values fall below 0.8 so a number above 0.8 is safe to say that it is highly correlated and the -0.8 is the inverse of the high correlation. After various feature selection experimentation with different correlation number, we have decided on 0.8 and -0.8 for the feature selection algorithm.

Feature Selection Algorithm

In [17]:
```
for target in x:

    #target = 'diagnosis'
    candidates = corr_mat.index[
        ((corr_mat[target] >= 0.8) | (corr_mat[target]<=-0.8))
    ].values
    candidates = candidates[candidates != target]
    print('Correlated to', target, ': ')
    print(candidates)

    removed = []
    for c1 in candidates:
        for c2 in candidates:
            if (c1 not in removed) and (c2 not in removed):
                if c1 != c2:
                    coef = corr_mat.loc[c1, c2]
                    if coef >= 0.8 or coef <= -0.8:
                        removed.append(c1)
    print('Removed: ', removed)

    selected_features = [x for x in candidates if x not in removed]
    print('Selected features: ', selected_features)
    print(" ")
```

```
Correlated to radius_mean :
['perimeter_mean' 'area_mean' 'concave points_mean' 'radius_worst'
 'perimeter_worst' 'area_worst']
Removed:  ['perimeter_mean', 'area_mean', 'concave points_mean', 'radius_worst', 'perimeter_worst']
Selected features:  ['area_worst']

Correlated to texture_mean :
['texture_worst']
Removed:  []
Selected features:  ['texture_worst']

Correlated to perimeter_mean :
['radius_mean' 'area_mean' 'concave points_mean' 'radius_worst'
 'perimeter_worst' 'area_worst']
Removed:  ['radius_mean', 'area_mean', 'concave points_mean', 'radius_worst', 'perimeter_worst']
Selected features:  ['area_worst']

Correlated to area_mean :
['radius_mean' 'perimeter_mean' 'concave points_mean' 'area_se'
 'radius_worst' 'perimeter_worst' 'area_worst']
Removed:  ['radius_mean', 'perimeter_mean', 'concave points_mean', 'area_se', 'radius_worst', 'perimeter_worst']
```

DropList of Features based on Algorithm

Alot of values that fall below 0.8 so a number above 0.8 is safe to say that it is highly correlated and the -0.8 is the inverse of the high correlation. So after various feature selection experimentation with different correlation number, we have decided on 0.8 for Pearson Correlation Matrix.

In [25]:
```
drop_list1 = ['perimeter_mean','radius_mean','compactness_mean','concave points_mean','radius_se','perimeter_se','radius_worst','
x_1 = x.drop(drop_list1,axis = 1 )        # do not modify x, we will use it later
x_1.head()
```

Out[25]:

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean	texture_se	area_se	smoothness_se	concavity_se
0	10.38	1001.0	0.11840	0.3001	0.2419	0.07871	0.9053	153.40	0.006399	0.05373
1	17.77	1326.0	0.08474	0.0869	0.1812	0.05667	0.7339	74.08	0.005225	0.01860
2	21.25	1203.0	0.10960	0.1974	0.2069	0.05999	0.7869	94.03	0.006150	0.03832
3	20.38	386.1	0.14250	0.2414	0.2597	0.09744	1.1560	27.23	0.009110	0.05661
4	14.34	1297.0	0.10030	0.1980	0.1809	0.05883	0.7813	94.44	0.011490	0.05688

Save New File to use the new 16 Features

In [27]:
```
x_1.to_csv("NewFeatures_16_SherFinal1.csv")
# But remember to add back the diagnosis column from the original file
```

In [ ]:

Metrics

In order to choose the suitable metrics, we must consider the objectives. Since this is a cancer diagnostic system, the 2 kinds of prediction errors have different level of importance. Type II errors (False Negative) should be very heavily penalized while Type I errors (False Positives) can be somewhat tolerated. In this case scenario, False Negatives will be misidentifying aggressive malignant tumours as harmless benign tumours and False Positives are misidentifying harmless benign tumours as aggressive malignant tumours.

Consider that it is much more critical to get correctly diagnosed when a person has cancerous tumours and start treatment as soon as possible. If a person with cancer gets a “healthy” benign tumour diagnosis and goes home, this will quickly result in fatal consequences. On the other hand, if the system predicts incorrectly that a healthy person has malignant tumour cancer, then additional diagnostic testing can quickly show that there is no problem and it is simply a false alarm.

Accuracy Paradox and Confusion Matrix

Accuracy is one metric for evaluating classification models. However, accuracy has its pitfalls too. The following is a screenshot of the accuracy score and the confusion matrix for KNN Classification on the **original 30 features of the dataset**. Accuracy score seems to come out great with 0.9% and above, however, upon closer analysis of the positives and negatives in the negative metrics will allow us to gain more insight into our model’s performance.

```
Accuracy Score of test 0.9649122807017544
AUC od the test  0.9920634920634921
 confusion matrix
 [[108   0]
 [  6  57]]
```

True Negative= TN, False Positive= FP, False Negative= FN, True Positive= TP, N= Total number of predictions.

TN= 108, FP=0, FN= 6, TP=57, n= 171

In the above matrix, the columns are those which have been predicted by the classifier. On the other hand, the rows are the actual classes of the dataset. There are two possible predicted classes: "Benign (First row class)" and "Malignant (Second row class)".

The classifier made a total of 171 predictions. 171 patients were being tested for the presence of either Benign or Malignant tumours. Out of those 171 cases, the classifier predicted "Malignant" 57 times, and "Benign" 114 times.

Out of those 171 cases, the classifier predicted "Malignant" 63 times but of which it predicted 6 were FN (malignant predicted as Benign), the classifier predicted "Benign" 108 times but predicted 0 were FP (Benign predicted as Malignant). It is excellent that the classifier predicted all Benign tumours correctly without any mistakes, but the model was not very accurate at identifying the malignant tumours correctly which is more critical.

FP where we predict “Malignant” but is actually only “Benign” is known as Type I error which can be tolerated as the penalties are not as heavy here. FN, we had 6. This means it predicted “Benign but is actually “Malignant”. This is Type II error which holds heavier penalty because if you misdiagnose a "Malignant" tumour, which is cancerous tumour as a "Benign" which is harmless tumour then the person will think he/she is safe when in fact his/her health is in danger. Then thinking it is a harmless tumour, the sick person will not proceed for further treatment will eventually cause health to deteriorate at a faster rate due to the aggressive nature of the cancer.

Therefore, accuracy alone does not tell the full story when we are working with class-imbalanced data set like this one, since there is quite a disparity between the number of positive and negative numbers. Thus, we look at other better metrics such as precision and recall.

Precision, Recall and F1 score

As described, the importance of Type II errors should be more critical than Type I errors. Minimizing Type II error implies maximizing recall and minimizing Type I error implies maximizing precision. For this reason, recall should be emphasized as compared to precision, which is not as important here. Precision refers to the percentage of the results that are relevant while recall refers to the percentage of total relevant results classified correctly by the algorithm (Precision vs Recall, 2018).

For example:

A. recall is 0.999 and precision is 1.0

B. recall is 1.0 and precision is 0.8

A model implies that on average, 1 person in 10000 will be misdiagnosed as healthy but having malignant cancerous tumour but all the healthy people with benign tumours will be correctly diagnosed. B model implies that everyone who has malignant tumour will be correctly diagnosed but 20% of the healthy people will be wrongly diagnosed as having malignant cancerous tumour (Pechyonkin,2020).

This paper argues that B model is better than A model because the life of one person is more important than the cost of additional testing of 2000 healthy people to make sure that they do not have cancerous tumour. Then we also compare with F1 score. The F1 score seeks to balance between Precision and Recall because while accuracy can be largely contributed by a large number of True Negatives, in most objectives, we tend not to focus critically on True Negatives, but instead usually the False Negative and False Positive have more impact and consequence, thus F1 score may be better to use if we need to balance between precision and recall and when there is uneven class distribution (large number of actual negatives) (Accuracy, Precision, Recall or F1?, 2020).

ROC and AUC

Receiver Operator Characteristic (ROC) and Area Under Curve (AUC) metrics are other alternatives for prediction use instead of only parameter accuracy. AUC-ROC curve is a performance measure for classification problem at various threshold settings. ROC represents a probability curve while AUC is a degree or measure of separability. This shows how much the model can distinguish between classes. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s which in this case will be predicting Benign as Benign and Malignant as Malignant. A good AUC is near to 1 which has good measure of separability while 0 means the worst of separability. When the AUC is at 0.5, the model has no class separation capacity. To get AUC, the RUC curve must be defined first (Understanding AUC - ROC Curve, 2019).

ROC, AUC, Precision and Recall

Generally, ROC curves are better when there are about equal numbers of observations for each class whereas precision-recall works better when there is a moderate or large class imbalance (Brownlee, 2018). As for the AUC, generally the scores between 30 features and 16 features (after feature selection) did not differ much and considering there is uneven class distribution/imbalance in the Breast Cancer dataset, we will **focus more on the precision, recall and F1 score metrics.**

K Nearest Neighbour (KNN) Classification Model

KNN is a simple, versatile and one of the topmost machine learning algorithms used in many types of applications like finance, healthcare, political science, handwriting detection, image and video recognition. The KNN algorithm is based on feature similarity approach and is used for both classification and regression problems. K stands for the number of nearest neighbours. The number of neighbours is the core decision factor and K value is usually an odd number when the number of classes is 2 (KNN Classification using Scikit-learn, 2020).

Choosing the optimal value for K

To select a K value is that most optimal for the dataset, we experiment with KNN algorithm several times with different value of K and ultimately select the K value that reduces the number of errors while maintaining other metrics scores and accuracy (Machine Learning Basics with the K-Nearest Neighbours Algorithm, 2019).

Advantages of KNN

1) KNN algorithm is simple and easy to implement. It does not require building a model, tuning parameters or making additional assumptions.
2) KNN algorithm is flexible and can be used for classification, regression and search.

Disadvantages of KNN- Curse of Dimensionality

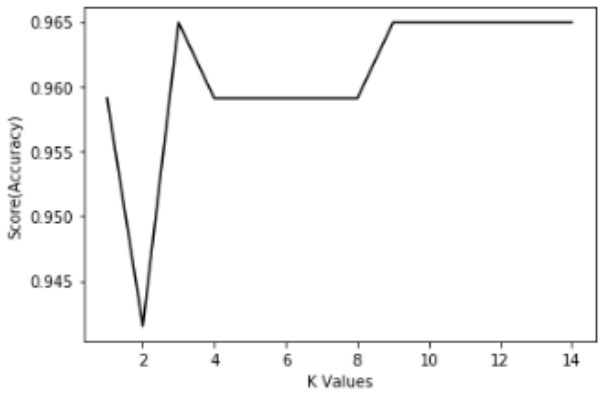
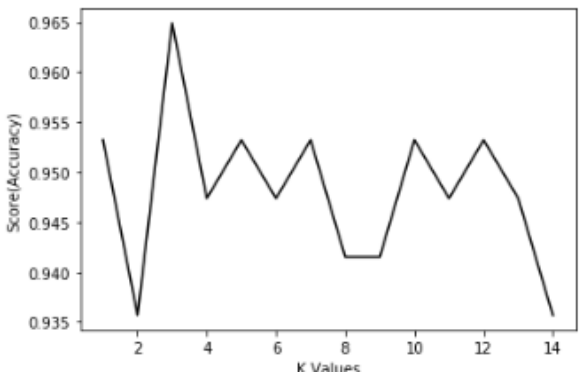
KNN algorithm performs better with a lower number of features than with many features. When the number of features or volume of data increases, it will also need more data and this increase of dimension can also lead the overfitting.

Dealing with The Curse of Dimensionality

The problem with higher dimension is known as the Curse of Dimensionality. To circumvent this issue, **applying feature selection or PCA** before using the machine learning is key. By inspecting the data, we can find optimal K value. Historically, the optimal K value for most datasets range between 3-10 which produces much better results than 1 NN (KNN Classification using Scikit-learn, 2020).

In our KNN Classification model and code, we have implemented an algorithm to plot a graph to see which is the most optimal K value, to obtain better metric scores and accuracy.

Comparison Results Table- KNN (30 features against 16 features)

<table>
<tr><td colspan="2">KNN Classification (Class 0= Benign, Class 1= Malignant)</td></tr>
<tr><td>30 Features-Original</td><td>16 Features (After Feature Selection with Pearson Correlation Matrix)</td></tr>
<tr><td>Accuracy Score of tests= 0.9649122807017544</td><td>Accuracy Score of tests = 0.9649122807017544</td></tr>
<tr><td>AUC of the test = 0.9880952380952381</td><td>AUC of the test = 0.9894179894179895</td></tr>
<tr><td>F-Measure= 0.9500000000000001</td><td>F-Measure =0.9500000000000001</td></tr>
<tr><td>Error rate =0.03508771929824561</td><td>Error rate is =0.03508771929824561</td></tr>
<tr><td>Optimal K value= 3
 
</td><td>Optimal K value= 3
 
</td></tr>
<tr><td>Time taken= 0.001995801925659177 seconds</td><td>Time taken= 0.0019948482513427734 seconds</td></tr>
<tr><td>

	precision	recall	f1-score	support
0	0.95	1.00	0.97	108
1	1.00	0.90	0.95	63
accuracy			0.96	171
macro avg	0.97	0.95	0.96	171
weighted avg	0.97	0.96	0.96	171

</td><td>

	precision	recall	f1-score	support
0	0.97	0.97	0.97	108
1	0.95	0.95	0.95	63
accuracy			0.96	171
macro avg	0.96	0.96	0.96	171
weighted avg	0.96	0.96	0.96	171

</td></tr>
<tr><td>Confusion Matrix
[[108 0]
[6 57]]

FN = 6, heavier penalties.</td><td>Confusion Matrix
[[105 3]
[3 60]]

FN= 3, not as bad as FN= 6(in 30 features)</td></tr>
</table>

Interpretation	**Interpretation**
Out of those 171 cases, the classifier predicted "Malignant" 63 times but of which it predicted 6 were FN (malignant predicted as Benign), the classifier predicted "Benign" 108 times but predicted 0 were FP (Benign predicted as Malignant). It is excellent that the classifier predicted all Benign tumours correctly without any mistake, but the model was not very accurate at identifying the malignant tumours correctly which is more critical.	Out of those 171 cases, the classifier predicted "Malignant" 63 times but of which it predicted 3 were FN (malignant predicted as Benign), the classifier predicted "Benign" 108 times but predicted 3 were FP (Benign predicted as Malignant). We argue that this result is better even though it was not as accurate in identifying the Benign tumours correct (made 3 misdiagnosis), this result was more accurate in identifying the malignant tumours which is more critical and our key objective.

Comparison of 30 against 16 features

Similarities: Both 30 and 16 features acquired similar F-Measure, Error rate and Optimal K value =3.

Differences:

Time taken: 16 Features took slightly faster for time at 0.001994 while 30 features took slightly slower at 0.001995. AUC

AUC Score: 16 features had higher AUC score at 0.989 compared to 30 features at 0.988. There is not much difference therefore we will focus more on the precision and recall metrics that has more distinction is the scores.

Another observation is that ROC/AUC curves work better where there are about equal numbers of observations for each class, but precision-recall will be better in this case as there is quite a moderate class imbalance between Benign and Malignant class.

For Precision and Recall:

30 features has:

	precision	recall	f1-score
0	0.95	1.00	0.97
1	1.00	0.90	0.95

16 Features has:

	precision	recall	f1-score
0	0.97	0.97	0.97
1	0.95	0.95	0.95

The F1 scores for both B and M classes for 30 and 16 features are the same at 0.97 and 0.95 respectively.

30 Features:
Benign (Class 0). recall is 1.00, precision is 0.95
Malignant (Class 1). recall is 0.90, precision is 1.00

16 Features:
Benign (Class 0). recall is 0.97, precision is 0.97
Malignant (Class 1). recall is 0.95, precision is 0.95

Justification on choosing 16 features model for KNN Classification

This paper argues that 16 features model is better than 30 features model because the recall for Malignant(Class 1) is higher at 0.95 compared to 0.90 30 features because the lives of people who have

cancer is more critical to detecting and save than the cost of additional testing of other misdiagnosed healthy people to make sure they do not have the cancerous malignant tumour.

Therefore although the 16 features may make slightly more misdiagnosis of healthy people with benign tumours as malignant tumours compared to 30 features, the small mistakes can be tolerated since the percentage is difference is slightly different only and also because our key objective is to minimize the misdiagnosis of misclassifying malignant as benign tumours which 16 features performs much better than 30 features in this field with a 10% improvement.

Impact of Data Engineering and Feature Engineering

The improved model for 16 features improved the AUC Score, the time taken and most importantly minimizing FN for misdiagnosis of malignant cancerous tumours as benign tumours with a distinct 10% improvement against the 30 features that apply feature selection. With feature selection, the KNN algorithm performed better with a lower number of features than with many features.

KNN Classification at 30 features Code Guide

For 16 features, just change read file path to 16 features file. K value does not need to change because both 30 and 16 features use K value= 3.

KNN CLassification for Original 30 Features- Shermaine

Import Libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import time
import matplotlib.pyplot as plt
import seaborn as sns
```

Read File/ Data Exploration

In [2]:
```
df= pd.read_csv("C:/Users/Shermaine/Documents/Python ML II (Cher Wah)/caknn/fs/breast_cancer_30.csv")
```

In [3]:
```
df.head(5)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

In [4]:
```
df.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
```

Data Engineering/Cleaning

Lets drop the unneccasary Varaibles- ID, Unnamed

In [5]:
```
df.drop(["Unnamed: 32"],axis=1,inplace=True)
df.drop(["id"],axis=1,inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

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

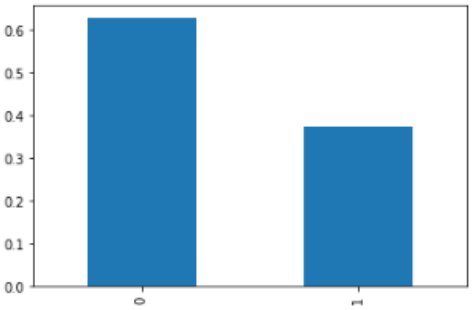
5 rows × 31 columns

Encoding

The second feature 'diagnosis' is binary and categorical with the values 'B' for Benign and 'M' for Malignant. The variable will be the label for supervised learning problem and the values will be encoded to 0 and 1 respectively for the training.

In [7]:
```
df['diagnosis']=df['diagnosis'].map({'B':0,'M':1})
```

In [8]:
```
df["diagnosis"].value_counts(normalize=True).plot(kind='bar')
plt.show()
```



Train_Test_Split= 70/30

In [9]:
```
X= df.drop("diagnosis",axis=1)
y= df["diagnosis"]

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=1)
```

KNN Classifier

Since KNN is a distance based Algorithm- we need to do standardization of values

In [10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=3)

start_time=time.time()

clf.fit(X_train, y_train)

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))

print(clf.score(X_test, y_test))
```

```
---0.0019931793212890625 seconds ---
0.9649122807017544
```

Import Metrics and Matrix Libraries

In [12]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,roc_auc_score
```

In [13]:
```
# Validating the train on the model
y_train_pred =clf.predict(X_train)
y_train_prob =clf.predict_proba(X_train)[:,1]

print("Accuracy Score of train", accuracy_score(y_train,y_train_pred))
print("AUC of the train ", roc_auc_score(y_train,y_train_prob))
print(" confusion matrix \n" , confusion_matrix(y_train,y_train_pred))
```

```
Accuracy Score of train 0.9824120603015075
AUC of the train  0.9985849438020539
 confusion matrix
 [[247   2]
 [  5 144]]
```

In [14]:
```
# Model on Test data
y_test_pred =clf.predict(X_test)
y_test_prob =clf.predict_proba(X_test)[:,1]

print("Accuracy Score of test", accuracy_score(y_test,y_test_pred))
print("AUC of the test ", roc_auc_score(y_test,y_test_prob))
print(" confusion matrix \n" , confusion_matrix(y_test,y_test_pred))
```

```
Accuracy Score of test 0.9649122807017544
AUC of the test  0.9880952380952381
 confusion matrix
 [[108   0]
 [  6  57]]
```

In [15]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       108
           1       1.00      0.90      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.95      0.96       171
weighted avg       0.97      0.96      0.96       171
```

Classification Rate/Accuracy(Manual Calculation to cross-check against classification_report):

TN= 108, FP=0,

FN= 6, TP=57

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Recall: Recall gives us an idea about when it's actually yes, how often does it predict yes. Recall = TP / (TP + FN) = 100 / (100 + 5) = 0.95

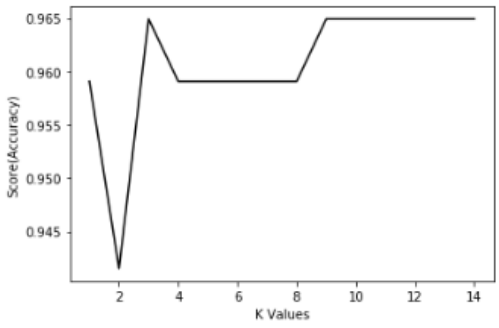
Precision: Precsion tells us about when it predicts yes, how often is it correct. Precision = TP / (TP + FP)=100/ (100+10) = 0.91

F-measure: Fmeasure = (2 * Recall * Precision) / (Recall + Presision) = (2 * 0.95 * 0.91) / (0.91 + 0.95) = 0.92

In [16]:
```
#Shermaine- cross verified with data from classification report in In[29]
TN= 108
FP=0
FN= 6
TP=57
TOTAL= 171 #n

Accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy", Accuracy)

#ecall: Recall gives us an idea about when it's actually yes, how often does it predict yes.
Recall = TP / (TP + FN)
print("Recall", Recall)

#Precision: Precsion tells us about when it predicts yes, how often is it correct.
Precision = TP / (TP + FP)
print("Precision", Precision)

F_measure = (2 * Recall * Precision) / (Recall + Precision)
print("F-Measure", F_measure)

#Misclassification Rate: Overall, how often is it wrong?
Misclassification_Rate=(FP+FN)/TOTAL
print("Error rate is",Misclassification_Rate)
#equivalent to 1 minus Accuracy
#also known as "Error Rate"
# https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
```

```
Accuracy 0.9649122807017544
Recall 0.9047619047619048
Precision 1.0
F-Measure 0.9500000000000001
Error rate is 0.03508771929824561
```

In [17]:
```
#Find Optimum K value
scores = []
for each in range(1,15):
    KNNfind = KNeighborsClassifier(n_neighbors = each)
    KNNfind.fit(X_train,y_train)
    scores.append(KNNfind.score(X_test,y_test))

plt.plot(range(1,15),scores,color="black")
plt.xlabel("K Values")
plt.ylabel("Score(Accuracy)")
plt.show()
```



Logistic Regression

Logistic Regression is a classification algorithm that is used to assign observations to a discrete set of classes. There are 2 types of logistics regression- binary and multi-linear functions failsClass. We are using binary logistics regression for malignant and benign tumours. Logistics Regression transforms its output using the sigmoid function to return a probability value (Introduction to Logistic Regression, 2019).

Duration of training for each model

Model 1: 1.1936 seconds

Model 2: 0.0550 seconds

Data Engineering

Data before Data Engineering:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

5 rows × 32 columns

Data after Data Engineering:

Data Engineering

Data Cleaning

```
df=df.drop("id",axis=1)
df.head()
```

[3]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.

5 rows × 31 columns

Label Encoding

Label Encoding

```
def myfunction(x):
    if x == "B":
        return 0
    elif x == "M":
        return 1


df["diagnosis"] = df["diagnosis"].apply(myfunction)
```

```
df.head()
```

5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_r
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.

5 rows × 31 columns

In terms of data engineering, redundant features like “id” has been removed from the dataset. In addition, label encoding had been applied to the feature “diagnosis” such that the values are converted to integers allowing this feature to be used for training the Logistic Regression data model.

Model 1 (30 Features)

```
X=df.drop("diagnosis",axis=1)
X.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fracti
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

```
y=df["diagnosis"]
y.head()
```

This training model includes using all the features in the dataset except “id” and “diagnosis” as its independent variables to predict the diagnosis of breast cancer.

Duration of Training for Model 1

Divide the data into training and test sets, 70% training and 30% test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 42)
```

Train the model and calculate the duration of time taken

```
logReg = LogisticRegression(solver = 'lbfgs', max_iter=9000,multi_class = 'multinomial', random_state = 42)

start_time=time.time()

logReg.fit(X_train, y_train)

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))
```

```
---1.1936018466949463 seconds ---
```

Duration of training for model 1: 1.1936 seconds

Accuracy score for Model 1

```
print(accuracy_score(y_test, y_pred))
```

```
0.9707602339181286
```

Accuracy score for model 1: 97.08%

Confusion Matrix for Model 1

```
print(confusion_matrix(y_test, y_pred))
```

```
[[106   2]
 [  3  60]]
```

Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	106	2
Actual Negative	3	60

AUC of the train for Model 1

```
# Validating the train on the model
y_train_pred =logReg.predict(X_train)
y_train_prob =logReg.predict_proba(X_train)[:,1]

print("Accuracy Score of train", accuracy_score(y_train,y_train_pred))
print("AUC of the train ", roc_auc_score(y_train,y_train_prob))
print(" confusion matrix \n" , confusion_matrix(y_train,y_train_pred))
```

```
Accuracy Score of train 0.9597989949748744
AUC of the train  0.9940971941457104
 confusion matrix
 [[243   6]
 [ 10 139]]
```

AUC of test for Model 1

```
# Model on Test data
#Model on Test data
y_test_pred=logReg.predict(X_test)
y_test_prob=logReg.predict_proba(X_test)[:,1]

print("Accuracy Score of test", accuracy_score(y_test,y_test_pred))
print("AUC od the test ", roc_auc_score(y_test,y_test_prob))
print(" confusion matrix \n" , confusion_matrix(y_test,y_test_pred))
```

```
Accuracy Score of test 0.9707602339181286
AUC od the test  0.9976484420928866
 confusion matrix
 [[106   2]
 [  3  60]]
```

Classification Report for Model 1

```
from sklearn.metrics import classification_report

print(classification_report(y_test,y_test_pred))
```

```
              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
```

Feature Engineering

Model 2 (16 Features)

```
dataset=dataset.drop(["perimeter_mean","radius_mean","compactness_mean","concave points_mean","radius_se","perimeter_se",
                      "radius_worst","perimeter_worst","compactness_worst","concave points_worst","compactness_se",
                      "concave points_se","texture_worst","area_worst"],axis=1)
```

```
dataset.head()
```

	diagnosis	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean	texture_se	area_se	smoothness_s
0	1	10.38	1001.0	0.11840	0.3001	0.2419	0.07871	0.9053	153.40	0.00639
1	1	17.77	1326.0	0.08474	0.0869	0.1812	0.05667	0.7339	74.08	0.00522
2	1	21.25	1203.0	0.10960	0.1974	0.2069	0.05999	0.7869	94.03	0.00615
3	1	20.38	386.1	0.14250	0.2414	0.2597	0.09744	1.1560	27.23	0.00911
4	1	14.34	1297.0	0.10030	0.1980	0.1809	0.05883	0.7813	94.44	0.01149

```
X1=dataset.drop("diagnosis",axis=1)
X1.head()
```

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean	texture_se	area_se	smoothness_se	concavit
0	10.38	1001.0	0.11840	0.3001	0.2419	0.07871	0.9053	153.40	0.006399	0.0
1	17.77	1326.0	0.08474	0.0869	0.1812	0.05667	0.7339	74.08	0.005225	0.0

This training model includes using texture_mean, area_mean, smoothness_mean, concavity_mean, symmetry_mean, fractal_dimension_mean, texture_se, area_se, smoothness_se, concavity_se, symmetry_se, fractal_dimension_se, smoothness_worst, concavity_worst, symmetry_worst, fractal_dimension_worst as its independent variables.

Duration of Training for Model 2

Divide the data into training and test sets, 70% training and 30% test

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.3, random_state = 42)
```

Train the model and calculate the duration of time taken

```
logReg1 = LogisticRegression(solver = 'lbfgs', max_iter=9000,multi_class = 'multinomial', random_state = 42)

start_time=time.time()

logReg1.fit(X1_train, y1_train)

end_time=time.time()

print("---%s seconds ---" % (end_time - start_time))
```

```
---0.05499911308288574 seconds ---
```

Duration of training for model 2: 0.0550 seconds

Accuracy score for Model 2

```
print(accuracy_score(y1_test, y1_pred))
```

```
0.9707602339181286
```

Accuracy score for model 2: 97.08%

Confusion Matrix for Model 2

```
print(confusion_matrix(y1_test, y1_pred))
```

```
[[106   2]
 [  3  60]]
```

Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	106	2
Actual Negative	3	60

AUC of the train for Model 2

```
# Validating the train on the model
y1_train_pred =logReg1.predict(X1_train)
y1_train_prob =logReg1.predict_proba(X1_train)[:,1]

print("Accuracy Score of train", accuracy_score(y1_train,y1_train_pred))
print("AUC of the train ", roc_auc_score(y1_train,y1_train_prob))
print(" confusion matrix \n" , confusion_matrix(y1_train,y1_train_pred))
```

```
Accuracy Score of train 0.9447236180904522
AUC of the train  0.9861998328886014
 confusion matrix
 [[241   8]
 [ 14 135]]
```

AUC of the test for Model 2

```
#Model on Test data
y1_test_pred=logReg1.predict(X1_test)
y1_test_prob=logReg1.predict_proba(X1_test)[:,1]

print("Accuracy Score of test", accuracy_score(y1_test,y1_test_pred))
print("AUC od the test ", roc_auc_score(y1_test,y1_test_prob))
print(" confusion matrix \n" , confusion_matrix(y1_test,y1_test_pred))
```

```
Accuracy Score of test 0.9707602339181286
AUC od the test  0.9936801881246327
 confusion matrix
 [[106   2]
 [  3  60]]
```

Classification Report for Model 2

```
from sklearn.metrics import classification_report

print(classification_report(y1_test,y1_test_pred))
```

```
              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
```

Comparison Results Table -Logistics Regression (30 features against 16 features)

Logistics Regression (Class 0= Benign, Class 1= Malignant)	
Model 1-30 Features	Model 2- 16 Features (with Feature Selection)
Accuracy Score of test= 0.9707602339181286	Accuracy Score of test= 0.9707602339181286
AUC of the test = 0.9976484420928866	AUC of the test = 0.9936801881246327
Time taken= 1.1936 seconds	Time taken= 0.0550 seconds
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	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
Confusion Matrix [[106 2] [3 60]]	Confusion Matrix [[106 2] [3 60]]
Model 1 Interpretation Out of those 171 cases, the classifier predicted "Malignant" 63 times but of which it predicted 3 were	**Model 2 Interpretation** Out of those 171 cases, the classifier predicted "Malignant" 63 times but of which it predicted 3 were

FN (malignant predicted as Benign), the classifier predicted "Benign" 108 times but predicted 2 were FP (Benign predicted as Malignant).	FN (malignant predicted as Benign), the classifier predicted "Benign" 108 times but predicted 2 were FP (Benign predicted as Malignant).

Comparison of 30 (Model 1) against 16 features (Model 2)

Model 2 (16 features) is the better model here as compared to Model 1 (30 features).

Comparison and impact of data and feature engineering

In comparison, both model 1 and model 2 generated the same accuracy score of 97.08%, confusion matrix, precision, recall and f1 scores. Both had very high AUC scores at 0.99. For model 1 which had a total of 30 features, this might be due to the fact that model 1 had too many highly correlated features in its model and also due to overfitting for having too many features and resulting in the model to be overly complex since feature engineering had not yet been applied for model 1.

Whereas for model 2 which had a total of 16 features, feature engineering had been applied for model 2 which resulted in highly correlated features being removed and the number of features being used in the model being dropped to 16 features from the original total of 30 features. Despite the decrease in the number of features being used for model 2, yet model 2 is still able to generate the same accuracy score as model 1 which also implied that the 16 features being used in model 2 actually contains the same amount of information as the 30 features being used in model 1.

Other observations- Time

In comparison, the duration needed to train model 1 is 1.1936 seconds whereas the duration needed to train model 2 is 0.0550 seconds. There is a big difference in the duration needed to train the models due to the size of the dataset. Having too many features would result in a longer duration needed to train the model and therefore it would be more efficient to choose the appropriate number of features to train the model such that less time is taken and yet still be able maintain the same amount of accuracy score. Therefore, Model 2 with 16 features is the better model for Logistics Regression.

Decision Trees

Decision trees are widely used models for classification and regression tasks. They learn a hierarchy of if/else questions, leading to a decision (MÜLLER and GUIDO, 2017).

Accuracy and Duration of training

30 Features (Model 1): 0.008008241653442383 seconds
Accuracy Score: 0.9385964912280702

16 Features (Model 2): 0.004988193511962891 seconds
Accuracy Score: 0.956140350877193

Process of training model

Our team decided to train each model with two split features- before feature engineering and after feature engineering. So firstly, we train the model with 30 features and then we train a new model with 16 features.

30 Features

In [4]:
```
from sklearn.model_selection import train_test_split
```

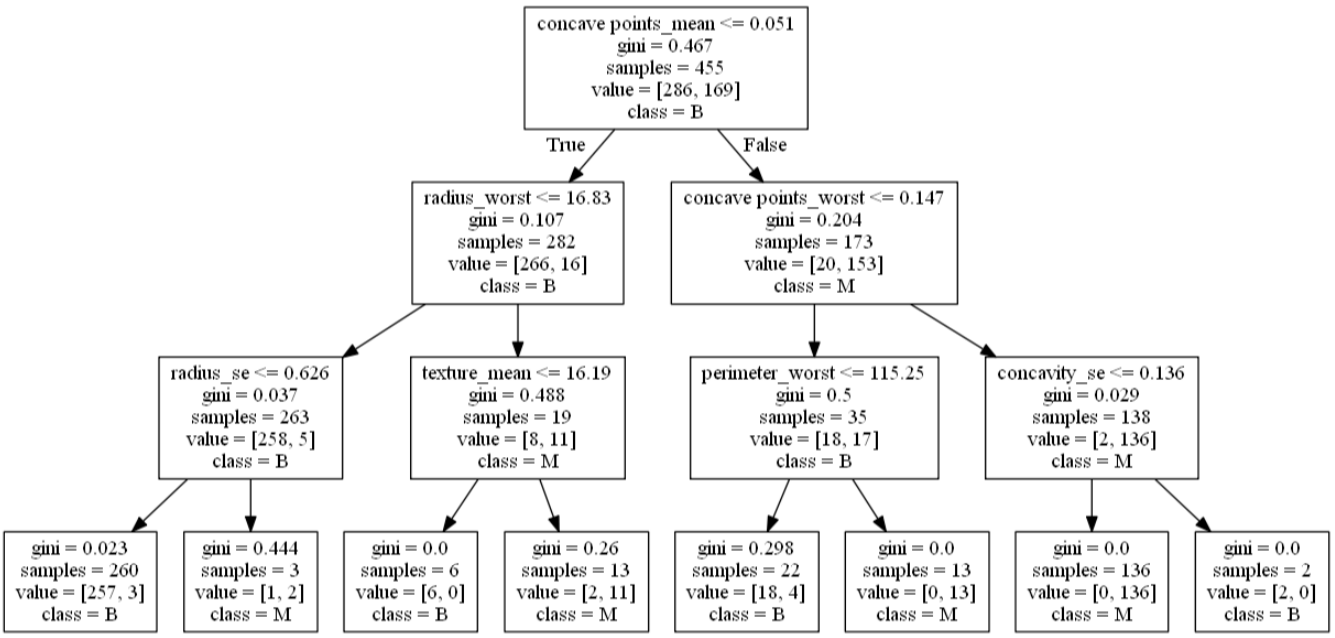
In [5]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

In [6]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [7]:
```
start_time = time.time()
dt = DecisionTreeClassifier(max_depth = 3)
dt.fit(x_train, y_train)
end_time = time.time()
print("---%s seconds ---" % (end_time - start_time))
```

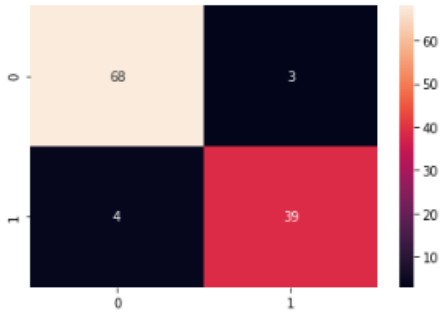
```
---0.008008241653442383 seconds ---
```

In [8]:
```
from sklearn import tree
import graphviz
from graphviz import Source
Source(tree.export_graphviz(dt, out_file=None, class_names=y_train.values ,feature_names= x_train.columns))
```



According to the tree we can say that concave points mean have the most correlation with others features and label.

```
In [10]: y_pred = dt.predict(x_test)

In [11]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_pred)

Out[11]: 0.9385964912280702

In [12]: from sklearn.metrics import f1_score,confusion_matrix

In [13]: cm = confusion_matrix(y_test,y_pred)
         sns.heatmap(cm,annot=True,fmt="d")

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2c3b1dc49c8>
```



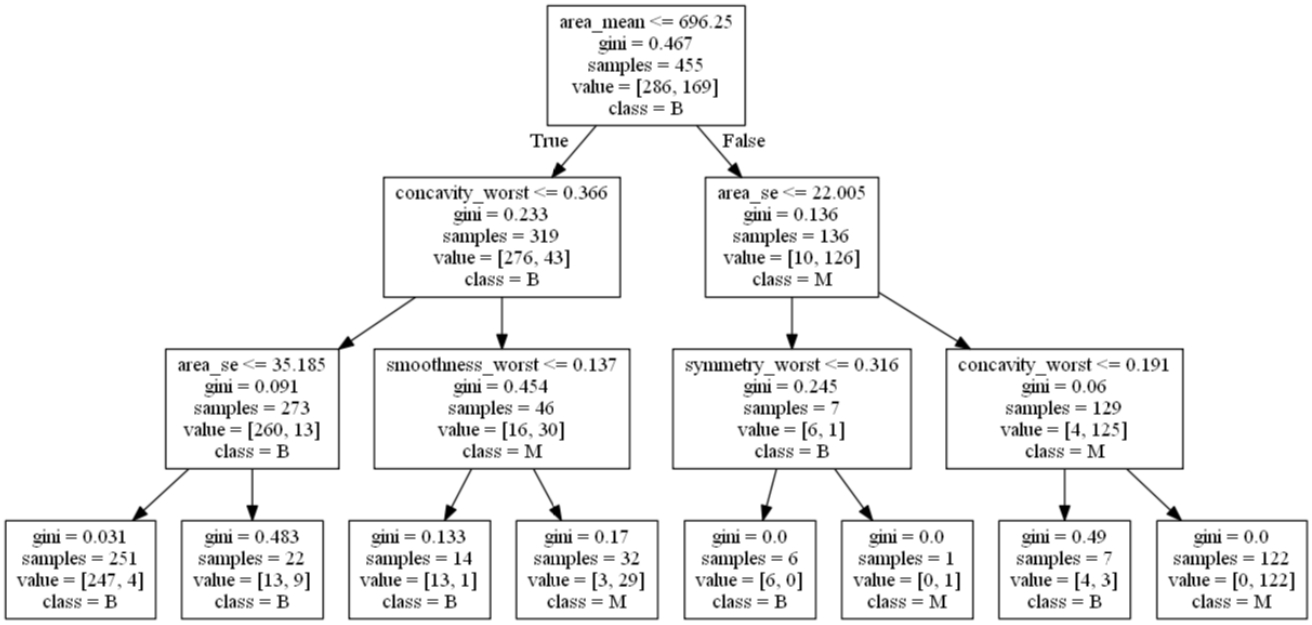
As you can see that the accuracy score is over 93%, so we can say our model is quite accurate. But when we see the confusion matrix for deeper analysis, we see there is 4 false negative, which is not good for this data set, because 4 of the patients will not know that their tumour is malignant. Thus, we are not satisfied with these features so we will implement feature selection.

With 16 Features

With 16 Features

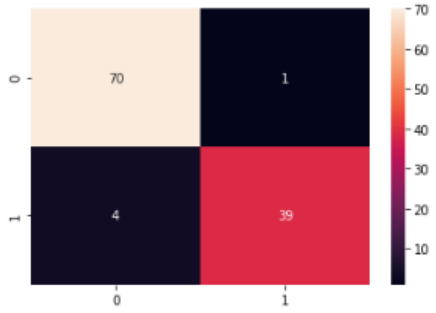
```
In [14]: drop_list1 = ['perimeter_mean','radius_mean','compactness_mean','concave points_mean','radius_se',
                       'perimeter_se','radius_worst','perimeter_worst','compactness_worst','concave points_worst',
                       'compactness_se','concave points_se','texture_worst','area_worst','id','diagnosis']
```

We drop the above 14 features and train the model with the remaining 16 features.



Now the root has been changed because we already dropped concave point mean which has the most correlation among the features. So, we can know that area mean has the best correlation between the remaining features.

```
In [27]: y1_pred = dt1.predict(x1_test)

In [28]: accuracy_score(y1_test, y1_pred)
Out[28]: 0.956140350877193

In [29]: cm1 = confusion_matrix(y1_test,y1_pred)
         sns.heatmap(cm1,annot=True,fmt="d")
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2c3b1c7f108>
```



Although we still have 4 false negatives like the Model 1, we managed to reduce the false positive and cut down the time by nearly half. The accuracy score has also increased. Thus, we can say that feature selection is important in machine learning, as it reduce the features and data we need to use and increase the accuracy score if done correctly.

NEURAL NETWORKS

Duration of training:

- ✧ 0s 52us/sample for original 30 features
- ✧ 0s 52us/sample for 16 features

Sharing of Process:

1.1. We work with original 30 features and 1 label named “Diagnosis”

Firstly, we import all neccessary libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from tensorflow.keras.utils import to_categorical
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D

In [2]: df = pd.read_csv("BreastCancerData.csv")

In [3]: df = df.drop(columns =["id"])

In [4]: df.tail()
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mea
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.17
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.17
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.15
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.23
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.15

5 rows × 31 columns

```
In [5]: df1 = df.replace(to_replace =["M","B"],value=[1,0])

In [6]: df1.head()
```

Out[6]:

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

5 rows × 31 columns

```
In [7]: x = df1.iloc[:,1:]
        x.shape
Out[7]: (569, 30)

In [8]: y=df1["diagnosis"]

In [9]: y.shape
Out[9]: (569,)
```

We split 90% for training, 10% for testing

In [10]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.1,random_state=42)
print ('Train set:', x_train.shape,  y_train.shape)
print ('Test set:', x_test.shape,  y_test.shape)
```

```
Train set: (512, 30) (512,)
Test set: (57, 30) (57,)
```

We fit the scaler on your training data only, then standardise both training and test sets with that scaler

In [11]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [12]:
```
x_train.shape
```

Out[12]:
```
(512, 30)
```

In [13]:
```
x_test.shape
```

Out[13]:
```
(57, 30)
```

One-Hot Encoding: Converting the Labels From Integers to Categorical Data ¶

In [14]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [15]:
```
y_train.shape
```

Out[15]:
```
(512, 2)
```

In [16]:
```
y_test.shape
```

Out[16]:
```
(57, 2)
```

In [17]:
```
# Create a neural network
model = Sequential()
```

Add layers to the neural network

In [18]:
```
model.add(Dense(100,input_shape =(30,),activation="relu"))
model.add(Dense(50,activation="sigmoid"))
model.add(Dense(2,activation="softmax"))
```

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               3100
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 102
=================================================================
Total params: 8,252
Trainable params: 8,252
Non-trainable params: 0
_________________________________________________________________
```

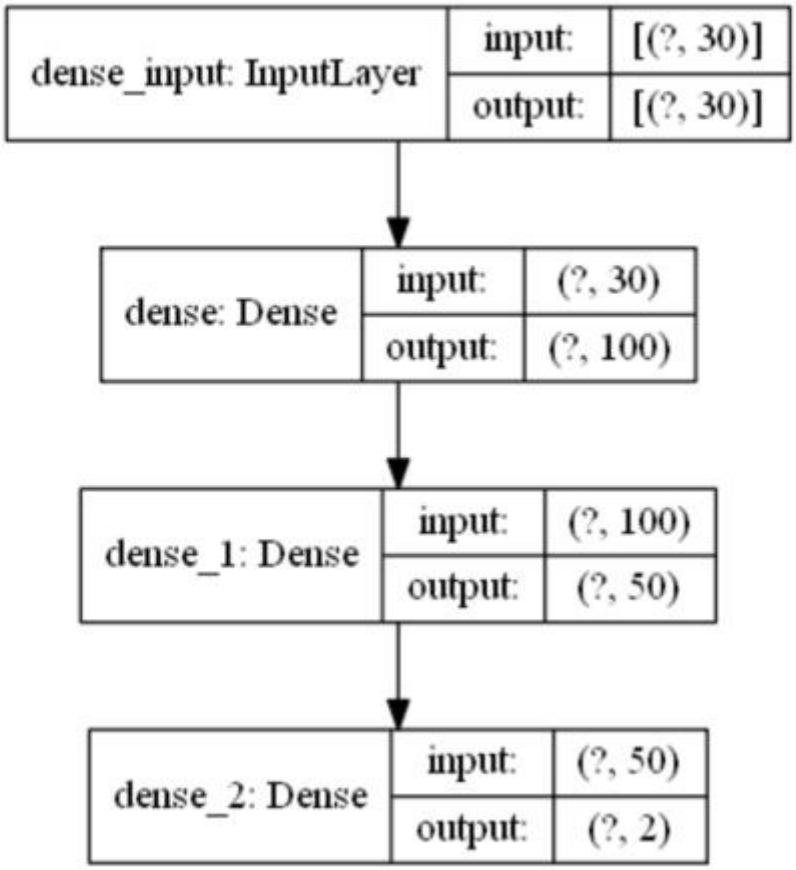
Visualizing the model's structure

In [20]:
```
from tensorflow.keras.utils import plot_model
```

In [21]:
```
from IPython.display import Image
```

In [22]:
```
plot_model(model, to_file='neuralNetwork.png', show_shapes=True,
           show_layer_names=True)
```

Out[22]:



Compiling the model

In [23]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

training with 90% for training and 10% for validation ¶

```
In [24]: model.fit(x_train, y_train, epochs=50, batch_size=100,validation_split=0.1)
```

```
Train on 460 samples, validate on 52 samples
Epoch 1/50
460/460 [==============================] - 0s 865us/sample - loss: 0.6095 - accuracy: 0.6565 - val_loss: 0.5084 - val_accurac
y: 0.8269
Epoch 2/50
460/460 [==============================] - 0s 28us/sample - loss: 0.4572 - accuracy: 0.8587 - val_loss: 0.3948 - val_accurac
y: 0.8846
Epoch 3/50
460/460 [==============================] - 0s 35us/sample - loss: 0.3495 - accuracy: 0.9261 - val_loss: 0.3133 - val_accurac
y: 0.8846
Epoch 4/50
460/460 [==============================] - 0s 30us/sample - loss: 0.2736 - accuracy: 0.9391 - val_loss: 0.2555 - val_accurac
y: 0.9038
Epoch 5/50
460/460 [==============================] - 0s 28us/sample - loss: 0.2189 - accuracy: 0.9478 - val_loss: 0.2160 - val_accurac
y: 0.9038
Epoch 6/50
460/460 [==============================] - 0s 30us/sample - loss: 0.1805 - accuracy: 0.9522 - val_loss: 0.1885 - val_accurac
y: 0.9038
```

Calculate Loss and Accuracy of the neural model

```
In [25]: loss, accuracy = model.evaluate(x_test, y_test, batch_size=100)
         print("Loss = " ,loss, "Accuracy = ",accuracy)
```

```
57/57 [==============================] - 0s 52us/sample - loss: 0.0894 - accuracy: 0.9825
Loss =  0.0894215926527977 Accuracy =  0.98245615
```

Perform Prediction ¶

```
In [26]: predictions = model.predict(x_test)
         for i in np.arange(len(predictions)):
             print(print("Actual: " , y_test[i],", Predicted:" , predictions[i]))
```

```
Actual:  [1. 0.] , Predicted: [0.95990884 0.0400911 ]
None
Actual:  [0. 1.] , Predicted: [6.4606476e-04 9.9935395e-01]
None
Actual:  [0. 1.] , Predicted: [0.00172552 0.99827445]
None
Actual:  [1. 0.] , Predicted: [9.993672e-01 6.328802e-04]
None
Actual:  [1. 0.] , Predicted: [9.9962294e-01 3.7699513e-04]
None
Actual:  [0. 1.] , Predicted: [2.9765264e-04 9.9970227e-01]
None
Actual:  [0. 1.] , Predicted: [3.2104301e-04 9.9967897e-01]
None
Actual:  [0. 1.] , Predicted: [0.03899173 0.9610082 ]
None
Actual:  [1. 0.] , Predicted: [0.6690027  0.33099735]
None
Actual:  [1. 0.] , Predicted: [0.99875414 0.00124588]
```

2.2. We work with original 16 features and 1 label named “Diagnosis”

Try with 16 features ¶

In [27]:
```
x1 = df1[['texture_mean', 'area_mean', 'smoothness_mean','concavity_mean',
       'symmetry_mean', 'fractal_dimension_mean',
       'texture_se', 'area_se', 'smoothness_se',
       'concavity_se', 'symmetry_se',
       'fractal_dimension_se', 'smoothness_worst',
       'concavity_worst', 'symmetry_worst', 'fractal_dimension_worst']]
x1.shape
```

Out[27]: (569, 16)

In [28]:
```
y1=df1["diagnosis"]
```

In [29]:
```
y1.shape
```

Out[29]: (569,)

We split 90% for training, 10% for testing

In [30]:
```
from sklearn.model_selection import train_test_split
x1_train,x1_test,y1_train,y1_test = train_test_split(x1,y1,test_size=0.1,random_state=42)
print ('Train set:', x1_train.shape,  y1_train.shape)
print ('Test set:', x1_test.shape,  y1_test.shape)
```

```
Train set: (512, 16) (512,)
Test set: (57, 16) (57,)
```

We fit the scaler on your training data only, then standardise both training and test sets with that scaler

In [31]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x1_train = sc.fit_transform(x1_train)
x1_test = sc.transform(x1_test)
```

In [32]:
```
x1_train.shape
```

Out[32]: (512, 16)

In [33]:
```
x1_test.shape
```

Out[33]: (57, 16)

One-Hot Encoding: Converting the Labels From Integers to Categorical Data

In [34]:
```
y1_train = to_categorical(y1_train)
y1_test = to_categorical(y1_test)
```

In [35]:
```
y1_train.shape
```

Out[35]: (512, 2)

In [36]:
```
y1_test.shape
```

Out[36]: (57, 2)

Create a neural network model

In [37]:
```
model1 = Sequential()
```

Add layers to the neural network

In [38]:
```
model1.add(Dense(100,input_shape =(16,),activation="relu"))
model1.add(Dense(50,activation="sigmoid"))
model1.add(Dense(2,activation="softmax"))
```

In [39]:
```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 100)               1700
_________________________________________________________________
dense_4 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 102
=================================================================
Total params: 6,852
Trainable params: 6,852
Non-trainable params: 0
_________________________________________________________________
```

Compiling the model

In [40]:
```
model1.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

Training with 90% for training and 10% for validation

In [41]:
```
model1.fit(x1_train, y1_train, epochs=50, batch_size=100,validation_split=0.1)
```

```
Epoch 45/50
460/460 [==============================] - 0s 28us/sample - loss: 0.0541 - accuracy: 0.9891 - val_loss: 0.1061 - val_accurac
y: 0.9423
Epoch 46/50
460/460 [==============================] - 0s 26us/sample - loss: 0.0531 - accuracy: 0.9891 - val_loss: 0.1052 - val_accurac
y: 0.9423
Epoch 47/50
460/460 [==============================] - 0s 28us/sample - loss: 0.0522 - accuracy: 0.9870 - val_loss: 0.1042 - val_accurac
y: 0.9423
Epoch 48/50
460/460 [==============================] - 0s 26us/sample - loss: 0.0514 - accuracy: 0.9891 - val_loss: 0.1030 - val_accurac
y: 0.9423
Epoch 49/50
460/460 [==============================] - 0s 28us/sample - loss: 0.0506 - accuracy: 0.9891 - val_loss: 0.1020 - val_accurac
y: 0.9423
Epoch 50/50
460/460 [==============================] - 0s 28us/sample - loss: 0.0498 - accuracy: 0.9870 - val_loss: 0.1013 - val_accurac
y: 0.9423
```

Out[41]:
```
<tensorflow.python.keras.callbacks.History at 0x1746d88da88>
```

Perform Prediction

```
In [43]: predictions = model1.predict(x1_test)
         for i in np.arange(len(predictions)):
             print(print("Actual: " , y1_test[i],", Predicted:" , predictions[i]))
```

```
Actual:  [1. 0.] , Predicted: [0.90762866 0.09237137]
None
Actual:  [0. 1.] , Predicted: [0.00486355 0.99513644]
None
Actual:  [0. 1.] , Predicted: [0.00633942 0.9936606 ]
None
Actual:  [1. 0.] , Predicted: [0.9931323  0.00686772]
None
Actual:  [1. 0.] , Predicted: [0.9969212 0.0030788]
None
Actual:  [0. 1.] , Predicted: [0.00126945 0.9987306 ]
None
Actual:  [0. 1.] , Predicted: [0.00147325 0.9985267 ]
None
Actual:  [0. 1.] , Predicted: [0.39952004 0.60047996]
None
Actual:  [1. 0.] , Predicted: [0.63182354 0.36817646]
None
Actual:  [1. 0.] , Predicted: [0.9931678  0.00683221]
```

Observations for Neural Networks:

The Neural Network Model has good prediction with accuracy score around 98% but takes more time than other model to train the data. Furthermore, after applying feature selection with 16 features, the model still yields a good prediction with the same accuracy score = 0.9825 compared with 30 features, yet the loss is slightly higher than the loss with original 30 features (0.1148 and 0.0894).

Part C) Unsupervised Machine Learning

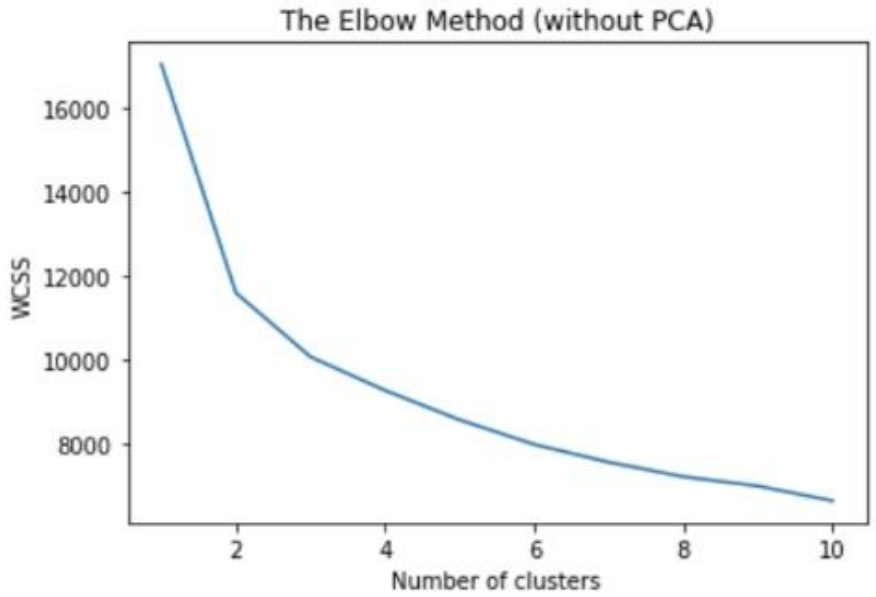
Problem Statement

The dataset that we will be using is the breast cancer dataset. Although the datapoints has already been categorized into benign and malignant, we decided to use the data for our clustering algorithm to see if we can identify subcategories within the benign and malignant categories, which can have implications for treatment of different types of malignant tumours, for example.

Our goal is to use various clustering algorithms such as K-Means and DBScan to find datapoints in our dataset with similar features and place them into groups. We will also be comparing the impact of training our models using principal components extracted from the original features using principal component analysis.

K-Means without PCA

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method (without PCA)')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



K-Means (with and without PCA)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
dataset = pd.read_csv('BreastCancerData.csv')
```

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

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

- As part of data engineering, we did feature scaling on the dataset

K-Means with PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
pca.fit_transform(X)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[4.42720256e-01 1.89711820e-01 9.39316326e-02 6.60213492e-02
 5.49576849e-02 4.02452204e-02 2.25073371e-02 1.58872380e-02
 1.38964937e-02 1.16897819e-02 9.79718988e-03 8.70537901e-03
 8.04524987e-03 5.23365745e-03 3.13783217e-03 2.66209337e-03
 1.97996793e-03 1.75395945e-03 1.64925306e-03 1.03864675e-03
 9.99096464e-04 9.14646751e-04 8.11361259e-04 6.01833567e-04
 5.16042379e-04 2.72587995e-04 2.30015463e-04 5.29779290e-05
 2.49601032e-05 4.43482743e-06]
```

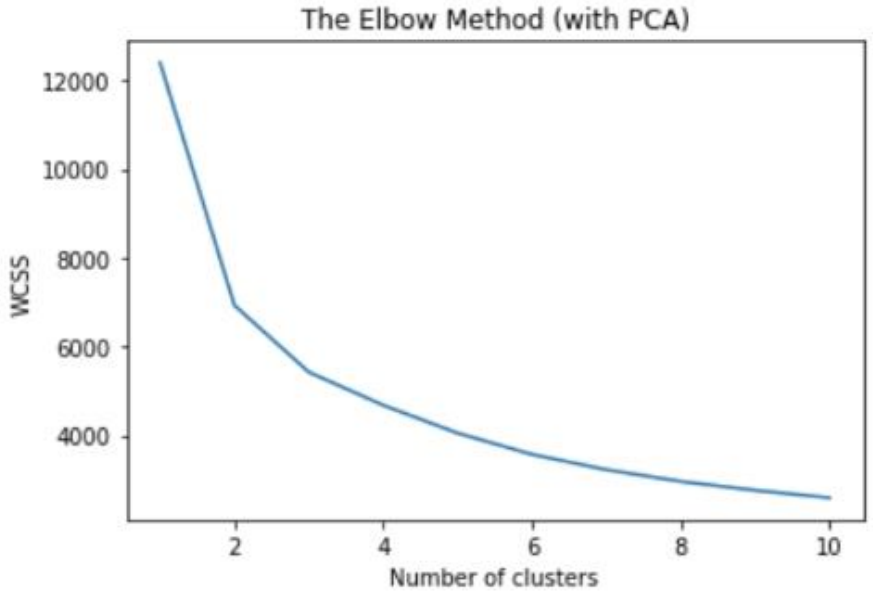
```
pca = PCA(n_components=3)
X = pca.fit_transform(X)
```

```
print(f"WCSS for no. of clusters = 2: {wcss[2]}")
```

```
WCSS for no. of clusters = 2: 10061.797818243695
```

- Using the elbow method, we can see that the optimal number of clusters is 2

```python
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method (with PCA)')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```python
print(f"WCSS for no. of clusters = 2: {wcss[2]}")
```

```
WCSS for no. of clusters = 2: 5422.816606122149
```

Analysis of K-Means with and without PCA

- When all 30 features from the dataset were used in the K-Means model, the WCSS for 2 clusters was 10061.80
- However, when PCA was performed on the 30 features, and the top 3 principal components were extracted and used in the K-Means model, the WCSS for 2 clusters was 5422.82
- Therefore, we conclude that PCA produced better results for our K-Means model

Training of K-Means model on top 3 principal components

```
import time

kmeans = KMeans(n_clusters = 2)

start = time.time()
clusters = kmeans.fit_predict(X)
```

K-Means model training time

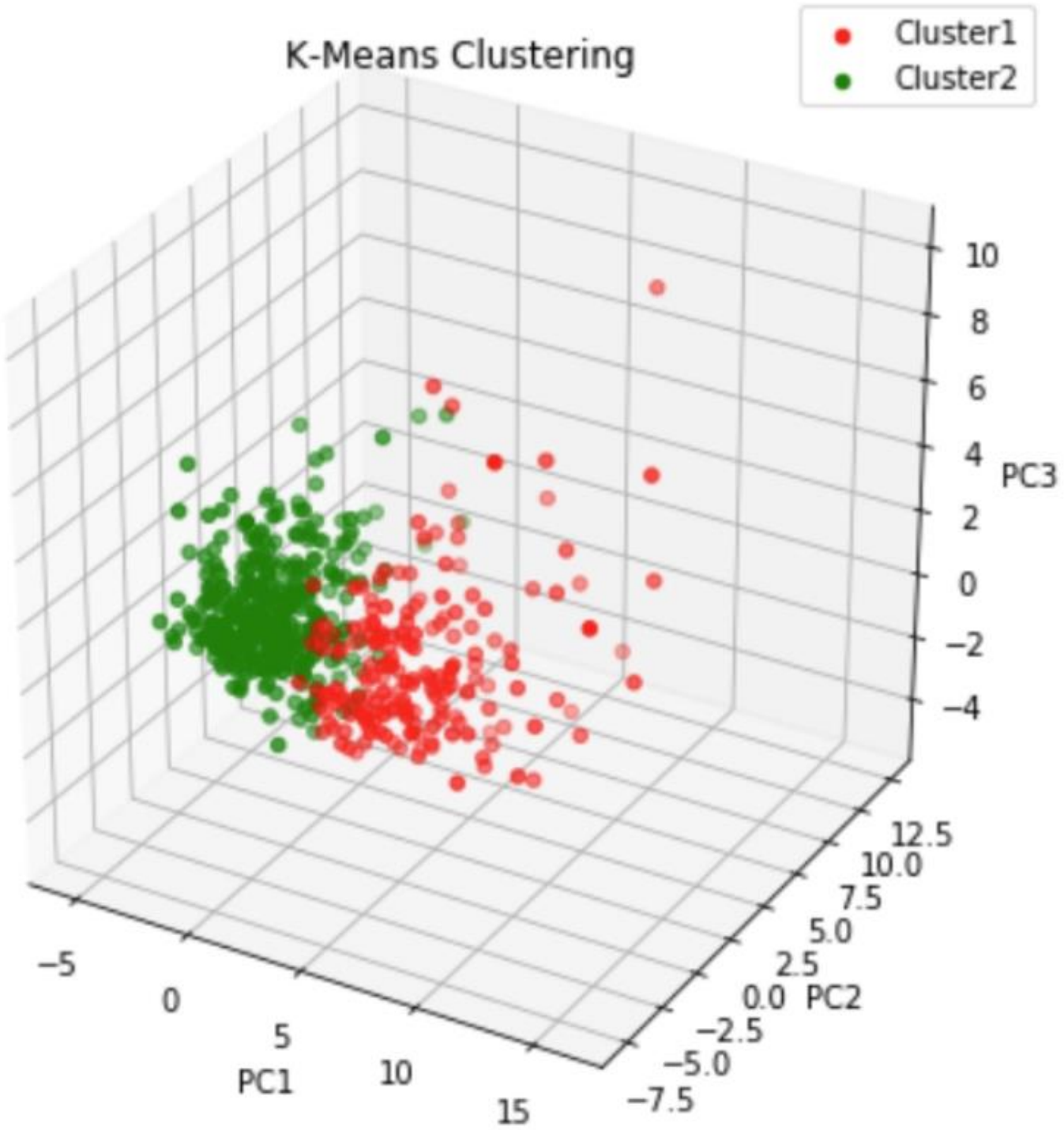
```
print(f"Training time: {stop - start}s")
```

```
Training time: 0.024625539779663086s
```

Plotting of the 3 principal comonents

```
from mpl_toolkits.mplot3d import Axes3D

colors = 'rgbkcmy'

fig = plt.figure(figsize=(7,7))
ax = plt.axes(projection='3d')

for i in np.unique(clusters):
    ax.scatter3D(X[clusters==i,0],
                 X[clusters==i,1],
                 X[clusters==i,2],
                 color=colors[i], label='Cluster' + str(i + 1))

ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")

plt.legend()
plt.title('K-Means Clustering')
plt.show()
```

K-Means Clustering Plot



Hierarchical Clustering

Standardise Data

```
data.head()
```

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

5 rows × 31 columns

```
#standardise data
from sklearn.preprocessing import StandardScaler
X=StandardScaler().fit_transform(data.iloc[:,1:])
X.shape
```

```
(569, 30)
```

The result of standardization is that the features will be rescaled so that they will have the properties of a standard normal distribution with:

- μ=0 and σ=1 (Where μ is the mean (average) and σ is the standard deviation from the mean)

Standardizing the features so that they are centred around 0 with a standard deviation of 1 is important here because:

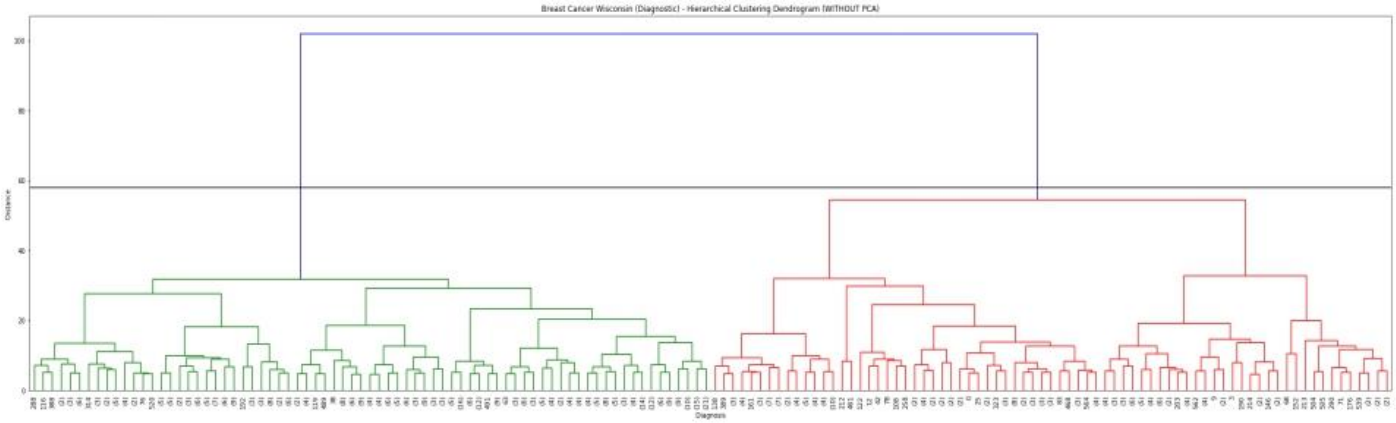
- we are comparing measurements that have different units.
- features of our input data set have large differences between their ranges
- Clustering models are distance-based algorithms. This means they use distance-based metrics to measure similarities between observations and form clusters. So, features with high ranges will have a bigger influence on the clustering. Hence, standardization is required before building a clustering model.

Hierarchical Clustering (Without PCA)

There are two types of tumours (in relevance to our dataset) – benign and malignant. Hence, the number of clusters formed from our data should be 2 (n_clusters =2).

```
from scipy.cluster.hierarchy import dendrogram,linkage
```

```
plt.figure(figsize = (40,10))
plt.title('Breast Cancer Wisconsin (Diagnostic) - Hierarchical Clustering Dendrogram (WITHOUT PCA)')
plt.xlabel('Diagnosis')
plt.ylabel('Distance')

dendrogram(
    linkage(X,'ward'),
    truncate_mode='lastp',
    p=150,
    leaf_rotation=90,
    leaf_font_size=10
)

plt.axhline(y=58, c='k')
plt.show()
```



- Usage of ward algorithm to calculate distance between newly formed clusters. This algorithm involves variance minimization.
- Original dendrogram is difficult to read, because the generated linkage matrix is large, so we made use of truncate mode to condense the dendrogram.
- lastp and p refer to the last p merged clusters. In this dendrogram, the last 150 merged clusters have been displayed.
- A max distance of almost 60 is required to separate the dataset into 2.

Model Training

```
from sklearn.cluster import AgglomerativeClustering
```

```
clustering = AgglomerativeClustering(linkage='ward',n_clusters=2)
```

```
#record time take to train model
import time
#start timer
start_time=time.time()
clustering.fit(X)
#stop timer
end_time=time.time()
duration=end_time-start_time
#printing duration of timing
print("Duration of model training: %s"%(duration))
```

```
Duration of model training: 0.02998518943786621
```

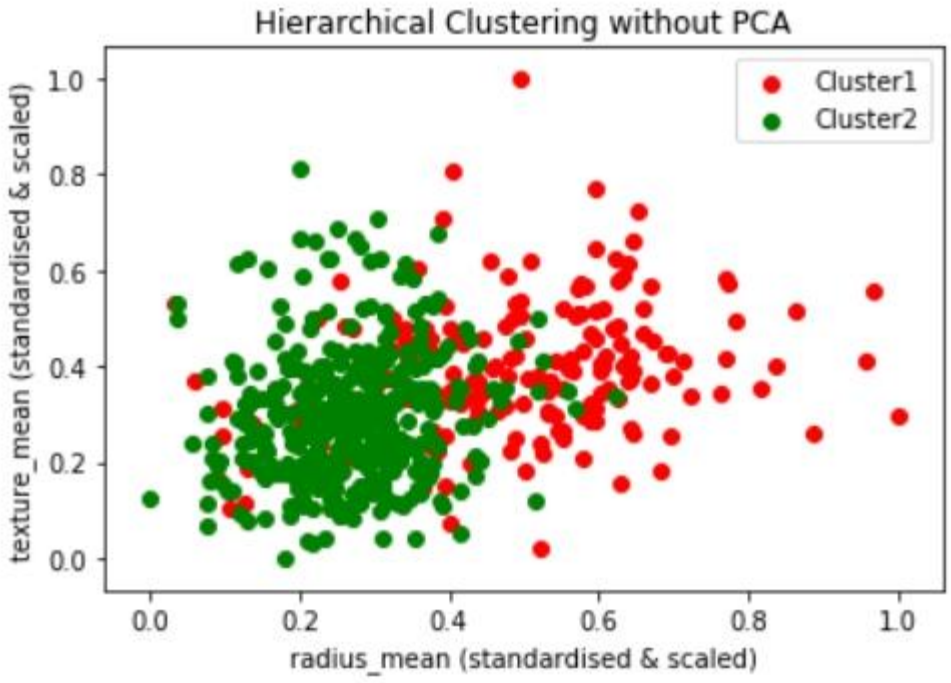
```
print(np.unique(clustering.labels_))
```

```
[0 1]
```

```
from sklearn import preprocessing
q=preprocessing.MinMaxScaler().fit_transform(X)
q.shape
```

```
(569, 30)
```

```
colors="rgb"
for i in np.unique(clustering.labels_):
    plt.scatter(q[clustering.labels_==i,0],
                q[clustering.labels_==i,1],
                color=colors[i],label='Cluster'+str(i+1))

plt.legend()
plt.title('Hierarchical Clustering without PCA')
plt.xlabel(data.columns[1]+' (standardised & scaled)')
plt.ylabel(data.columns[2]+' (standardised & scaled)')
plt.show()
```



Two distinct clusters can be observed from the above scatter plot. However, a significant area of cluster 1 does overlap with cluster 2. Most of the data points belonging to cluster 1 tend to have a higher texture mean and higher radius mean than those belonging to cluster 2. In general, the agglomerative clustering algorithm was able to cluster the data effectively. This is also an indication that the data is well suited for training classification models.

Hierarchical Clustering (With PCA)

Data has already been standardised

```
from sklearn.decomposition import PCA
pca=PCA(n_components=None)
pca.fit_transform(X)
explained_variance=pca.explained_variance_ratio_
print(explained_variance)
```

```
[4.42720256e-01 1.89711820e-01 9.39316326e-02 6.60213492e-02
 5.49576849e-02 4.02452204e-02 2.25073371e-02 1.58872380e-02
 1.38964937e-02 1.16897819e-02 9.79718988e-03 8.70537901e-03
 8.04524987e-03 5.23365745e-03 3.13783217e-03 2.66209337e-03
 1.97996793e-03 1.75395945e-03 1.64925306e-03 1.03864675e-03
 9.99096464e-04 9.14646751e-04 8.11361259e-04 6.01833567e-04
 5.16042379e-04 2.72587995e-04 2.30015463e-04 5.29779290e-05
 2.49601032e-05 4.43482743e-06]
```

From this we can notice that with just the first two components we can explain more than 60% of the variance. This is impressive especially given that we are working with a 30-dimension data set and trying to plot something meaningful with all thirty features would be very difficult.

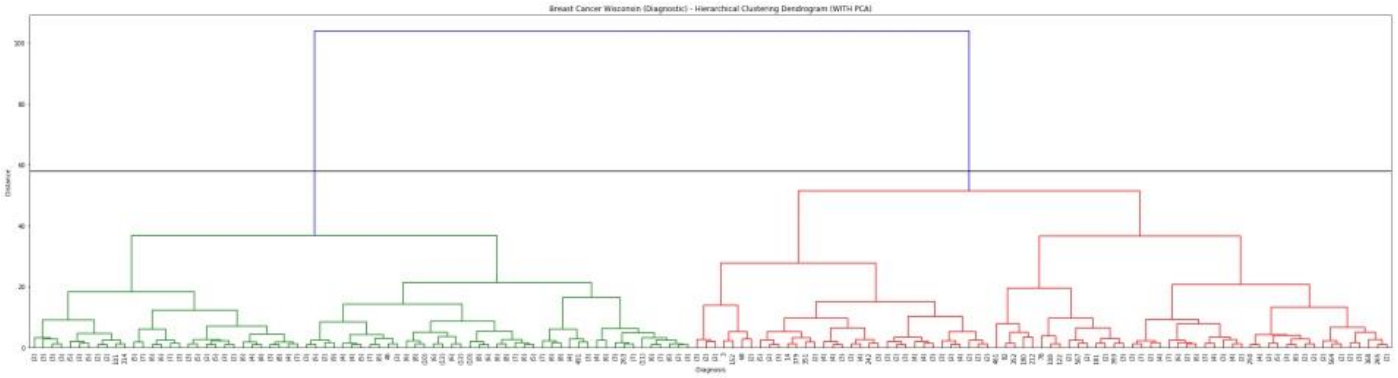
Applying PCA

```
pca=PCA(n_components=2)
X2=pca.fit_transform(X)
X2.shape
```

```
(569, 2)
```

Dendrogram after Applying PCA

```
plt.figure(figsize = (40,10))
plt.title('Breast Cancer Wisconsin (Diagnostic) - Hierarchical Clustering Dendrogram (WITH PCA)')
plt.xlabel('Diagnosis')
plt.ylabel('Distance')

dendrogram(
    linkage(X2,'ward'),
    truncate_mode='lastp',
    p=150,
    leaf_rotation=90,
    leaf_font_size=10
)

plt.axhline(y=58, c='k')
plt.show()
```



Model Training

```
start_time=time.time()
clustering.fit(X2)
#stop timer
end_time=time.time()
duration=end_time-start_time
#printing duration of timing
print("Duration of model training: %s"%(duration))
```

```
Duration of model training: 0.017989635467529297
```

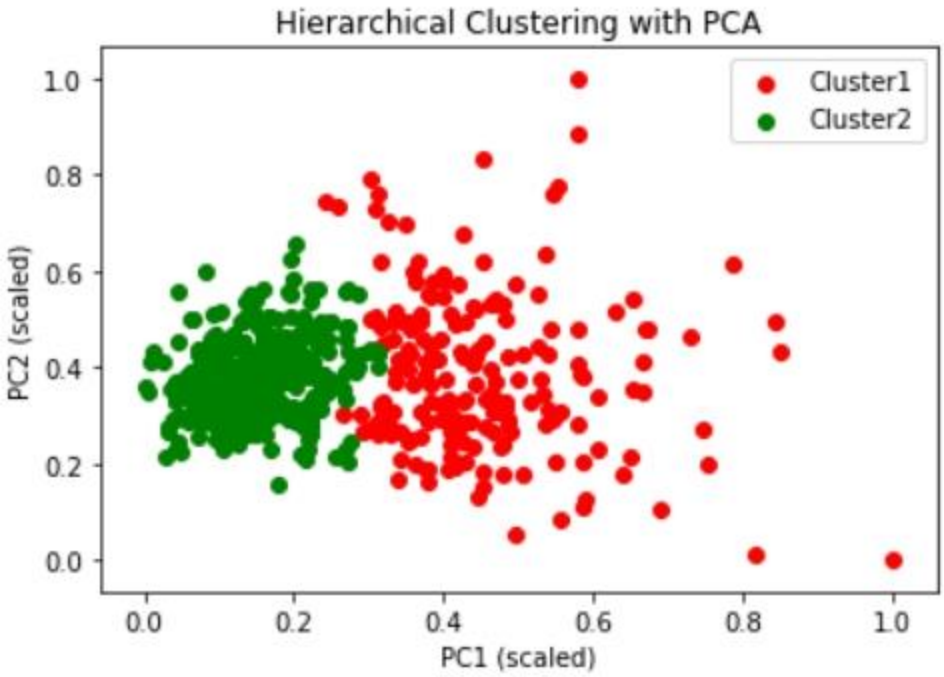
Time to taken to train model after performing PCA on data was significantly shorter. Which makes sense since we are using much lesser features to train the model now.

```
g=preprocessing.MinMaxScaler().fit_transform(X2)
g.shape
```

```
(569, 2)
```

```
colors="rgb"
for i in np.unique(clustering.labels_):
    plt.scatter(g[clustering.labels_==i,0],
                g[clustering.labels_==i,1],
                color=colors[i],label='Cluster'+str(i+1))

plt.legend()
plt.title('Hierarchical Clustering with PCA')
plt.xlabel('PC1 (scaled)')
plt.ylabel('PC2 (scaled)')
plt.show()
```

Plot- Hierarchical Clustering with PCA



Again, two distinct clusters can be observed from the above scatter plot. However, this time a significant portion of cluster 1 does not seem overlap with cluster 2. In general, the agglomerative clustering algorithm has clustered the data more effectively when the data had undergone PCA (more distinct clusters) compared to the data which had not undergone PCA.

It should also be noted that PCA derives principal components from original features of the data. And though the number of principal components is lesser than the original features they can explain maximal variance of the data. It reduces dimensionality and not features (the features are needed to construct the principal components). As such, since the dimension of the data is reduced, it helped reduce the time required to train the model. With PCA, the training time decreased from 0.029985 (5s.f.) seconds to 0.017990 (5s.f.) seconds.

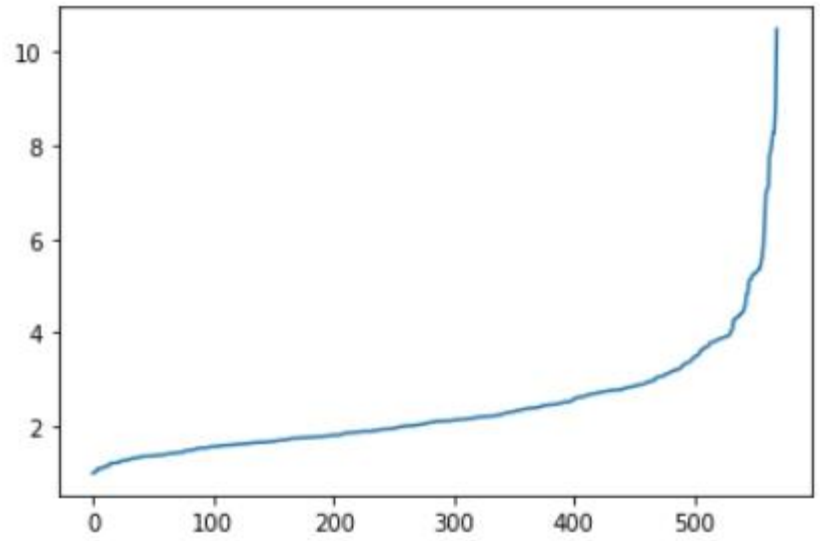
DBSCAN

Determining a suitable epsilon & minPts

We find a suitable value for epsilon by calculating the distance to the nearest n points for each point, sorting and plotting the results. Then we look to see where the change is most pronounced (greatest change in gradient) and select that as epsilon.
We can calculate the distance from each point to its closest neighbour using NearestNeighbors. The point itself is included in n_neighbors.

```
from sklearn.neighbors import NearestNeighbors

neigh=NearestNeighbors(n_neighbors=2)
nbrs=neigh.fit(X)
distances, indices = nbrs.kneighbors(X)
```

```
distances=np.sort(distances, axis=0)
distances=distances[:,1]
plt.plot(distances)
```



From the above graph we will take eps to be 4.

Determining number of minimum points with heuristic technique:

- minPts = ln(n) - where n is the total number of points to be clustered.
- Therefore minPts = ln(569) ≈ 6 (To nearest whole number)

DBSCAN (Without PCA)

```
from sklearn.cluster import DBSCAN
```

```
import time
#start timer
start_time=time.time()
dbscan=DBSCAN(eps=4,min_samples=6)
clusters=dbscan.fit_predict(X)
#stop timer
end_time=time.time()
duration=end_time-start_time
#printing duration of timing
print("Duration of model training: %s"%(duration))
```

```
Duration of model training: 0.07295703887939453
```

```
colors='rgbkcmy'
```

```
np.unique(clusters)
```

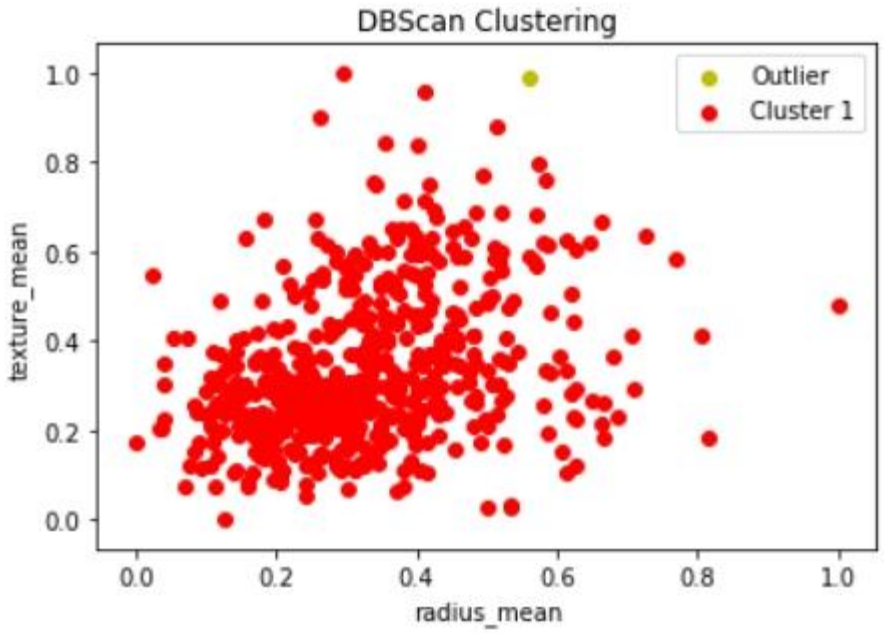
```
array([-1,  0], dtype=int64)
```

This means, there is only one cluster and outliers. The model has performed very poorly. Tried with 7,12 and 60 minPts and still gave same number of results.

```
from sklearn import preprocessing
j=preprocessing.MinMaxScaler().fit_transform(X)
j.shape
```

```
(569, 30)
```

```
for i in np.unique(clusters):
    label = "Outlier" if i==-1 else 'Cluster '+str(i+1)
    plt.scatter(j[clusters==i,1],j[clusters==i,2],
                color=colors[i],label=label)

plt.xlabel(data.columns[1])
plt.ylabel(data.columns[2])

plt.legend()
plt.title('DBScan Clustering')
plt.show()
```



DBSCAN (With PCA)

```
from sklearn.decomposition import PCA
pca=PCA(n_components=None)
pca.fit_transform(X)
explained_variance=pca.explained_variance_ratio_
print(explained_variance)
```

```
[4.42720256e-01 1.89711820e-01 9.39316326e-02 6.60213492e-02
 5.49576849e-02 4.02452204e-02 2.25073371e-02 1.58872380e-02
 1.38964937e-02 1.16897819e-02 9.79718988e-03 8.70537901e-03
 8.04524987e-03 5.23365745e-03 3.13783217e-03 2.66209337e-03
 1.97996793e-03 1.75395945e-03 1.64925306e-03 1.03864675e-03
 9.99096464e-04 9.14646751e-04 8.11361259e-04 6.01833567e-04
 5.16042379e-04 2.72587995e-04 2.30015463e-04 5.29779290e-05
 2.49601032e-05 4.43482743e-06]
```

```
pca=PCA(n_components=2)
X2=pca.fit_transform(X)
X2.shape
```

```
(569, 2)
```

Model Training

```
start_time=time.time()
dbscan=DBSCAN(eps=4,min_samples=6)
clusters=dbscan.fit_predict(X2)
#stop timer
end_time=time.time()
duration=end_time-start_time
#printing duration of timing
print("Duration of model training: %s"%(duration))
```

```
Duration of model training: 0.017990589141845703
```

```
np.unique(clusters)
```

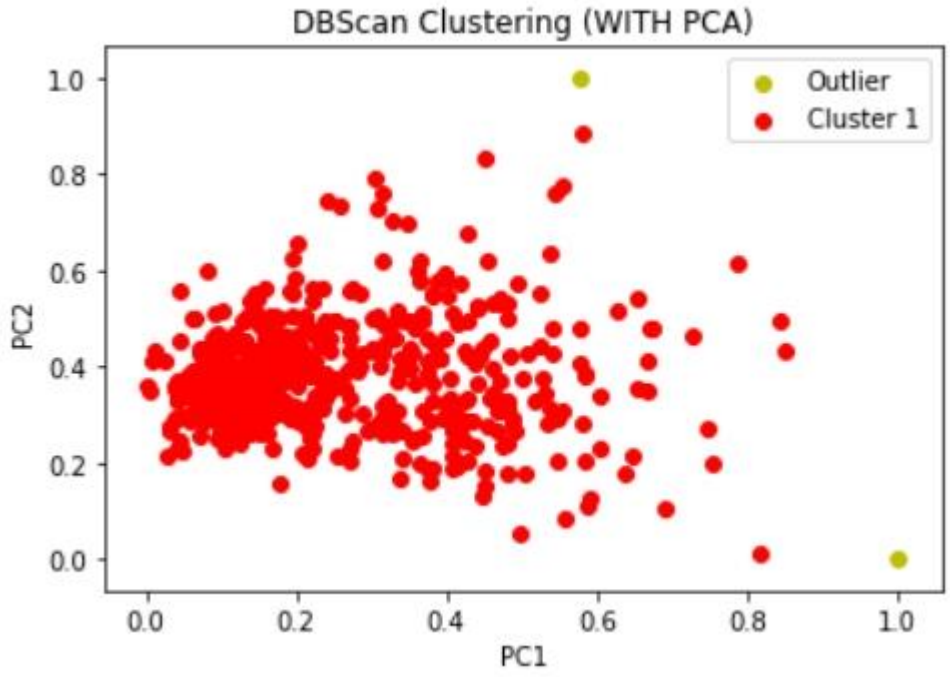
```
array([-1,  0], dtype=int64)
```

Much shorter training duration. But same disappointing results.

```
k=preprocessing.MinMaxScaler().fit_transform(X2)
k.shape
```

```
(569, 2)
```

```
for i in np.unique(clusters):
    label = "Outlier" if i==-1 else 'Cluster '+str(i+1)
    plt.scatter(k[clusters==i,0],k[clusters==i,1],
                color=colors[i],label=label)

plt.xlabel('PC1')
plt.ylabel('PC2')

plt.legend()
plt.title('DBScan Clustering (WITH PCA)')
plt.show()
```



The DBSCAN clustering performed very poorly because it could not identify the two clusters we had expected from it. After performing feature selection and feature extraction on our dataset, performance was still poor, but it can also be noted that more outliers were identified.
There are some possible explanations to regarding the poor performance. Firstly, it could be due to our data containing clusters of similar density. DBSCAN is effective in separating high density clusters from low density clusters and struggles with clusters of similar density. Secondly, it could have also been due to the high dimensionality of our dataset.

Overall Table Comparison of Results for Supervised Learning Models

The results from KNN, Logistics Regression, Decision Trees and Neural Networks are consolidated into a single table for comparison to choose the best model for the Breast Cancer dataset. The best model is taken from each supervised machine learning model and compared against all the other best supervised learning model results.

KNN Classification	Logistics Regression	Decision Tree	Neural Networks
16 features	16 features	16 features	30 features
Accuracy Score of tests = 0.9649122807017544	Accuracy Score of test= 0.9707602339181286	Accuracy Score of tests = 0.956140350877193	Accuracy score = 0.9825
Accuracy Score of tests = 0.9649122807017544	AUC of the test = 0.9936801881246327	-	-
Time taken= 0.0019948482513427734 seconds	Time taken= 0.0550 seconds	Time taken= 0.004988193511962891 seconds	Time taken= 0s 52us/sample
Confusion Matrix [[105 3] [3 60]]	Confusion Matrix [[106 2] [3 60]]	Confusion Matrix [[70 1] [4 39]]	-
precision recall f1-score 0 0.97 0.97 0.97 1 0.95 0.95 0.95 accuracy 0.96 macro avg 0.96 0.96 0.96 weighted avg 0.96 0.96 0.96	precision recall f1-score 0 0.97 0.98 0.98 1 0.97 0.95 0.96 accuracy 0.97 macro avg 0.97 0.97 0.97 weighted avg 0.97 0.97 0.97	-	-

Observations on the overall comparison of supervised learning models

Although the Neural Networks model had the highest accuracy, this is not accurate as it could be due to overfitting for having too many features since it has 30 features. Therefore, we have decided to focus on the other models that returned better scores for 16 features. A further comparison among the KNN, Logistics Regression and Decision Tree reveals that the Logistics Regression model has the best accuracy score and a better confusion matrix that is best suited towards our main objectives.

Our main objective is to reduce the FN as much as possible as it is Type II error and more critical to reduce mistakes in misidentifying malignant as benign tumours while being able to tolerate slightly higher scores for FP as this is Type I error for misidentifying benign as malignant tumours .

We eliminate Decision Tree among Logistics Regression and KNN as Decision Tree has FN score of 4 compared to Logistics Regression and KNN having and FN of 3. Upon comparing Logistics Regression and KNN, KNN is eliminated as it has FP score of 3 compared to Logistics Regression having a FP score of 2.

A closer analysis at the precision, recall and F1-scores between KNN and Logistics Regression also reveals **the clear winner to be Logistics Regression.** Although Logistics Regression took slightly more time than KNN, Logistics Regression clearly has higher scores for all the scores in precision, recall and F1-score for both class 0 and 1 in addition to accuracy and AUC score.

Summary and Conclusion

The following summarises our research and observations in applying various algorithms, concepts and metrics to various types of machine learning models.

Part A- Linear Regression (Multiple Linear Regression and Polynomial Regression)

For Part A on Linear Regression, we can improve the accuracy score of Multiple Linear Regression by removing values based on P-value above 0.005. We can further improve Multiple Linear Regression in this case with Polynomial Regression, which has advantages such as being able to fit a wide range of curvature, a broad range of function can also fit under it and it can generate the best approximation of the relationship between the dependent and independent variable The presence of outliers can affect the results of this Polynomial Regression but in this case scenario, is not applicable here.

Part B- Supervised Learning Models, Metrics, Data Engineering and Feature Selection

For Part B on Supervised Learning Models, we conclude that accuracy alone does not reveal the full story when we are working with class-imbalanced set like this Breast Cancer dataset, a closer analysis of the confusion matrix helps to see whether the model is a good fit in fulfilling the main objectives and we have to look at other better metrics such as precision and recall too. Another observation is that ROC/AUC curves work better when there are about equal number of observations for each-class but precision and recall work better in this case since there is quite an amount of class imbalance between Benign and Malignant class.

As stated in our main objectives, the importance of Type II errors should be more critical than Type I errors. For this reason, recall and precision should be emphasized. Precision refers to the percentage of the results that are relevant while recall refers to the percentage of total relevant results classified correctly by the algorithm.

Before doing feature selection, we must do data engineering to explore and clean the data to remove redundant fields like NaN in Part B or fields such as ID and Date that can't be calculated in Linear Regression for Part A. Feature selection helps reduce overfitting, improve accuracy and reduce training time. Less irrelevant/redundant data will mean less opportunity to make decisions based on noise. Less misleading data also improves the model's accuracy. Having fewer data points also reduces algorithm complexity and algorithms can train faster. Feature selection helps by choosing features that will give better accuracy while requiring less data. Feature selection is one of the core concepts in machine learning that impacts greatly on the machine learning model's performance.

Filter methods are more flexible and faster as they not require a machine learning model first to determine if a feature is good or bad, whereas a wrapper method needs a machine learning model to train it to see if the feature is important or not.

Wrapper methods are also computationally expensive and may not be the most efficient feature selection method especially when dealing with massive datasets. Since the wrapper method is not applicable to all machine learning models for supervised learning, which in this case, as discussed, RFE and RFECV may be applicable for Logistics Regression model but not for the KNN Classification Model, the simplicity, sufficient data and speed of filter method like Pearson Correlation is more suitable for this Breast Cancer dataset for a fair comparison against the classification models.

Part C- Unsupervised Learning and Clustering

For Part C on Unsupervised Learning, attempts to derive clusters from the Wisconsin Breast Cancer (Diagnostic) data was done so using three unsupervised learning models: K-Means, Hierarchical (Agglomerative) Clustering and DBSCAN. Among the three models, it was evident that DBSCAN performed the worst because it only identified one cluster. This can be attributed to the fact that the data does not have distinct densities. DBSCAN is effective in separating high-density clusters from low-density clusters but struggles with clusters of similar density. The high dimensionality of the dataset could also be another reason why the DBSCAN model performed poorly.

Both K-Means and Hierarchical clustering models were effective in producing at least 2 distinct clusters and both performed significantly better when they were trained with data that underwent PCA. It should be noted the PCA can help with denoising which in turn can lead to more stable clustering. Key differences between K Means and Hierarchical clustering include results are reproducible in Hierarchical clustering unlike in K Means where we start with a random choice of clusters. Hence running the K-Means algorithm multiple times may produce differing results. Also, it is easier to visualise clusters in hierarchical clustering than in K-Means. This is because in hierarchical clustering we can decide what would be an appropriate number of clusters by interpreting the dendrogram, whereas in K-Means we are expected to have prior knowledge of how many clusters we want to divide our data into.

It is important to understand that different models work differently for each dataset. Hence, it is essential for us to have good domain knowledge regarding the data and a good understanding of how each of the models works in order to decide which model will give us the best insights.

References

Assessing the Fit of Regression Models - The Analysis Factor (2008). Available at: https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/ (Accessed: 22 May 2020).

Accuracy, Precision, Recall or F1? (2020). Available at: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (Accessed: 21 May 2020).

Analytics Vidhya. (2016) *Feature Selection methods with example (Variable selection methods)*. Available at: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/ (Accessed: 21 May 2020).

Brownlee, J. (2016) *Supervised and Unsupervised Machine Learning Algorithms*, *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (Accessed: 22 May 2020).

Brownlee, J. (2019) *How to Choose a Feature Selection Method For Machine Learning*, *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/ (Accessed: 21 May 2020).

Brownlee, J. (2018) *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*, *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ (Accessed: 22 May 2020).

Feature Selection in Python (2020). Available at: https://www.datacamp.com/community/tutorials/feature-selection-python (Accessed: 21 May 2020).

Feature selection – Part I: univariate selection | *Diving into data* (2014). Available at: https://blog.datadive.net/selecting-good-features-part-i-univariate-selection/ (Accessed: 21 May 2020).

Feature Selection with sklearn and Pandas (2019). Available at: https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b (Accessed: 21 May 2020).

Frost, J. (2017) *How to Interpret P-values and Coefficients in Regression Analysis - Statistics By Jim*, *Statistics By Jim*. Available at: https://statisticsbyjim.com/regression/interpret-coefficients-p-values-regression/ (Accessed: 22 May 2020).

Introduction to Linear Regression and Polynomial Regression (2019). Available at: https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb (Accessed: 22 May 2020).

Introduction to Logistic Regression (2019). Available at: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148 (Accessed: 22 May 2020).

KNN Classification using Scikit-learn (2020). Available at: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn (Accessed: 22 May 2020).

Kumari, M. and Singh, V. (2018) "Breast Cancer Prediction system", *Procedia Computer Science*, 132, pp. 371-376. doi: 10.1016/j.procs.2018.05.197.

Machine Learning Basics with the K-Nearest Neighbors Algorithm (2019). Available at: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 (Accessed: 22 May 2020).

MÜLLER, A. C., & GUIDO, S. (2017). *Introduction to machine learning with Python: a guide for data scientists*. http://www.dawsonera.com/depp/reader/protected/external/AbstractView/S9781449369903.

Pechyonkin, M. (2020) *Predictive Modeling in Breast Cancer Diagnostics Using Supervised Machine Learning Techniques*, *Max Pechyonkin*. Available at: https://pechyonkin.me/portfolio/breast-cancer-diagnostics/ (Accessed: 21 May 2020).

Precision vs Recall (2018). Available at: https://towardsdatascience.com/precision-vs-recall-386cf9f89488 (Accessed: 21 May 2020).

sklearn.feature_selection.RFECV — scikit-learn 0.23.1 documentation (2020). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html (Accessed: 21 May 2020).

UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set (2020). Available at: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) (Accessed: 21 May 2020).

Understanding AUC - ROC Curve (2019). Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (Accessed: 22 May 2020).

Bibliography

About Feature Scaling and Normalization (2014). Available at: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html#about-standardization (Accessed: 23 May 2020).

Birant, Derya & Kut, Alp. (2007). ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data & Knowledge Engineering. 60. 208-221. 10.1016/j.datak.2006.01.013. Available at: https://www.researchgate.net/publication/223059919_ST-DBSCAN_An_algorithm_for_clustering_spatial-temporal_data (Accessed 24 May 2020)

DBSCAN: What is it? When to Use it? How to use it. (2017). Available at: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818 (Accessed: 23 May 2020).

Principal Component Analysis (PCA) 101, using R (2020). Available at: https://towardsdatascience.com/principal-component-analysis-pca-101-using-r-361f4c53a9ff (Accessed: 23 May 2020).