```
# Import libraries
import pandas as pd

# Display the first 5 data
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_wor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.6
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.8
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.7
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.2

5 rows × 33 columns

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Display the last 5 data
data.tail()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_wor
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2439	0.13890	...	26.40	166.1
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1440	0.09744	...	38.25	155.1
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10223	0.0951	0.05843	...	34.12	126.7
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514	0.15200	...	39.42	184.6
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37	59.1

5 rows × 33 columns

```
# count the number of rows and columns in the dataset
data.shape
```

```
(569, 33)
```

```
# count the number of empty (NaN, NAN , na) values in each column
data.isna().sum()
```

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

```
# Drop the column with all missing values (that can be 0 or ) for Integer and 'index' or 'columns' for String)
data = data.dropna(axis=1)
```

```
# Get the new count all the number of rows and columns again
data.shape
```

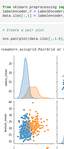
```
(569, 32)
```

```
# Get a count of the number of Malignant(M) or Benign(B) cells
data['diagnosis'].value_counts()
```

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

```
# Visualize the count of B and M in the dataset x => data['diagnosis'] , y => label='count'
sns.countplot(data['diagnosis'], label='count')
```

```
C:\Users\lata\anaconda3\Lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
```




```
# Look at the data types to see which columns need to be encoded
data.dtypes
```

```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

```
# Encode the categorical data values
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
data.iloc[:,1] = labelencoder.fit_transform(data.iloc[:,1].values)
```

```
from sklearn.pairplot import pairplot
sns.pairplot(data.iloc[:,1:6], hue='diagnosis')
```

```
<seaborn.axisgrid.PairGrid at 0x22390b8b0>
```

```
# Get the correlation of the columns
data.iloc[:,1:12].corr()
```

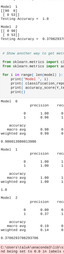
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.696360	0.776614	0.330499	-0.012838
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	0.023398	0.236702	0.302418	0.293464	0.071401	-0.076437
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716126	0.850977	0.183027	-0.261477
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823693	0.151293	-0.283110
smoothness_mean	0.358560	0.170581	0.023398	0.207278	0.177028	1.000000	0.659123	0.521984	0.553635	0.557775	0.584792
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369
concavity_mean	0.696360	0.676764	0.302418	0.716126	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823693	0.553635	0.831135	0.921391	1.000000	0.462497	0.166774
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166774	0.479921	1.000000

```
# Visualize the correlation
sns.heatmap(data.iloc[:,1:12].corr())
```

```
<matplotlib.axes._subplots.AxesSubplot>
```

```
plt.figure(figsize=(10,10))
sns.heatmap(data.iloc[:,1:12].corr(),annot=True, fmt='.0%')
```

```
<matplotlib.axes._subplots.AxesSubplot>
```




```
# Split the data set into independent (X) and dependent (Y) data sets
X = data.iloc[:,3:31].values
Y = data.iloc[:,1].values
```

```
# Split the data set into 75% training and 25% testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

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

X_train
```

```
array([[-0.38844962, -1.189e+00, -1.744e+00, ..., 1.318e-01, 9.149e-02,
        1.024e-01],
       [-0.38844962, -1.113e+00, -1.086e+00, ..., 2.000e-02, 2.012e-02,
        2.000e-02],
       [-0.38844962, 2.913e+00, 2.325e+00, ..., 2.215e-01, 1.628e-01,
        2.123e-01],
       [-0.38844962, 0.436e+00, 1.421e+00, ..., 1.144e-01, 5.952e-02,
        1.099e-01],
       [-0.38844962, 0.739e+00, 1.421e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [-0.38844962, 1.151e+00, 2.303e+00, ..., 2.638e-01, 9.653e-02,
        2.112e-01]])
```

```
# Bread Cancer Detection Machine Learning Model Building
# Create a function for the model

def models(X_train, Y_train):

  # Logistic Regression
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)

  # Decision Tree Classifier
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
  tree.fit(X_train, Y_train)

  # Random Forest Classifier
  from sklearn.ensemble import RandomForestClassifier
  forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
  forest.fit(X_train, Y_train)

  # Print the models accuracy on the training data
  print('[0]Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
  print('[1]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  print('[2]Random Forest Classifier Training Accuracy:', forest.score(X_train, Y_train))

  return log, tree, forest
```

```
# Getting all of the models
model = models(X_train,Y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.9824561403508772
[1]Decision Tree Classifier Training Accuracy: 1.0
[2]Random Forest Classifier Training Accuracy: 0.9723684210526315
C:\Users\lata\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
# Test model accuracy on test data on confusion matrix
from sklearn.metrics import confusion_matrix

for i in range(len(model)):
  print('Model ', i)
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

  TN = cm[0][0]
  TP = cm[1][1]
  FN = cm[1][0]
  FP = cm[0][1]

  print(cm)
  print('Testing Accuracy = ', (TP + TN)/ (TP + TN + FN + FP))
  print()
```

```
Model  0
[[86 2]
 [ 6 50]]
Testing Accuracy =  0.9066666666666666

Model  1
[[84 4]
 [ 7 49]]
Testing Accuracy =  1.0

Model  2
[[ 0 90]
 [ 0 56]]
Testing Accuracy =  0.3766666666666667
```

```
# Show another way to get metrics of the models
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

for i in range(len(model)):
  print('Model ',i,')')
  print(classification_report(Y_test, model[i].predict(X_test)))
  print( accuracy_score(Y_test, model[i].predict(X_test)))
  print()
```

```
Model  0
              precision    recall  f1-score   support

           0       0.93      0.98      0.96        88
           1       0.96      0.89      0.92        56

    accuracy                           0.95       144
   macro avg       0.94      0.93      0.94       144
weighted avg       0.95      0.95      0.95       144

0.9444444444444444

Model  1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        90
           1       1.00      1.00      1.00        54

    accuracy                           1.00       144
   macro avg       1.00      1.00      1.00       144
weighted avg       1.00      1.00      1.00       144

1.0

Model  2
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        90
           1       0.37      1.00      0.54        54

    accuracy                           0.37       144
   macro avg       0.19      0.50      0.27       144
weighted avg       0.14      0.37      0.20       144

0.3766666666666667
```

```
C:\Users\lata\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use zero_division parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```