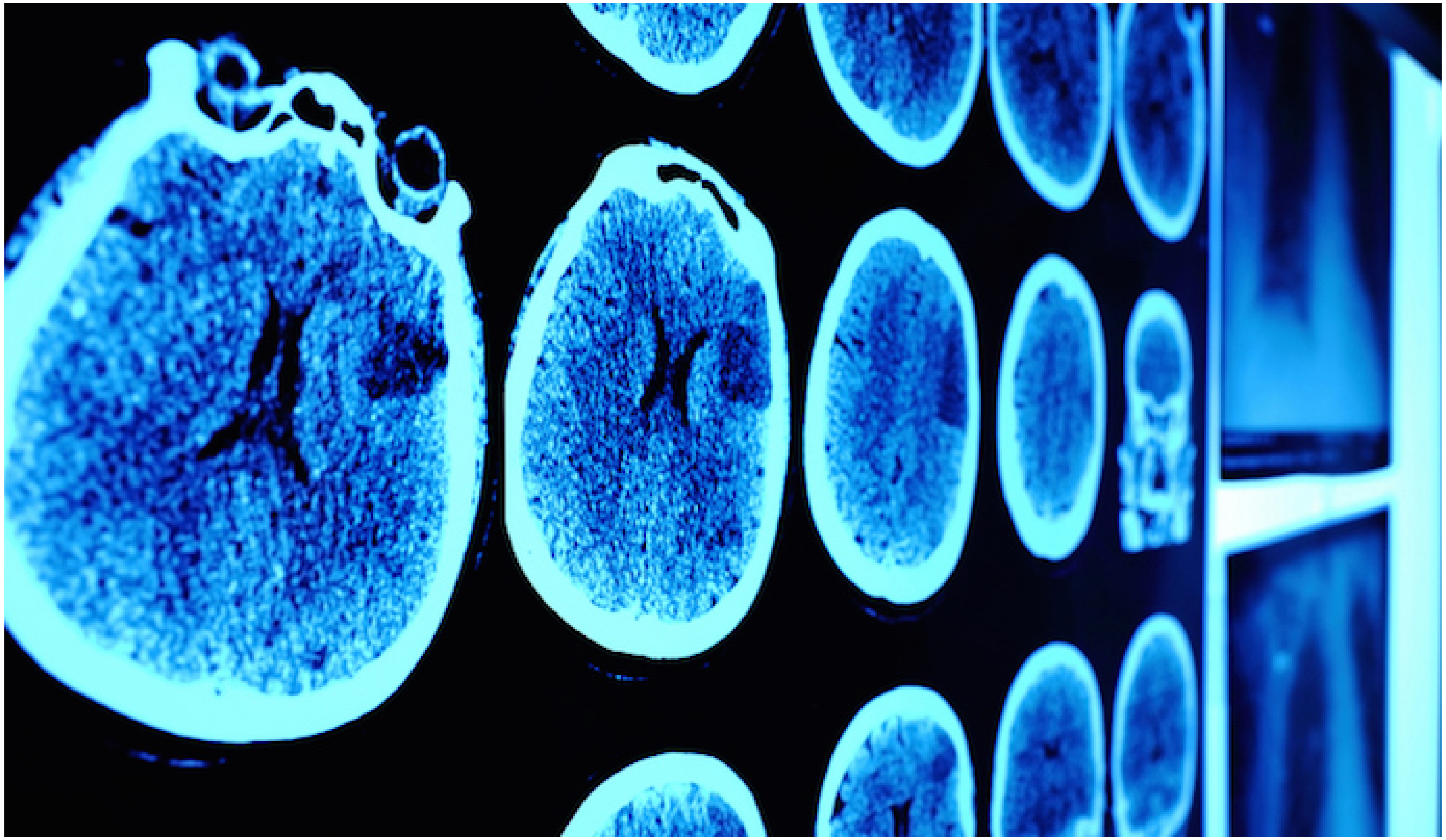
Breast Cancer Prediction Using Machine Learning.

Table Of Contains

> Steps are:



Attribute Information:

1. ID number

- Diagnosis (M = malignant, B = benign)

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

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)

8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

```
import numpy as np
import pandas as pd

pd.options.display.max_columns = 100
```

After installing numpy and pandas package, we are ready to fetch data using pandas package, Befor we use it, We need to know where's our dataset located. Means what is the path of our dataset

1. Data Collection.

```
from google.colab import files
uploaded = files.upload()
```

Choose files breastCancer.csv

- **breastCancer.csv**(text/csv) - 125204 bytes, last modified: 03/01/2024 - 100% done

```
Saving breastCancer.csv to breastCancer.csv
```

```
data = pd.read_csv("breastCancer.csv")
```

After collecting data, we need to know what are the shape of this dataset, Here we have attribute(`property`) called `data.shape`

For that we have 2 type of methods to show the shape of the datasets.

1. `len(data.index), len(data.columns)`

- `data.shape`

Both methods are giving us the same output, As you can see in the below cells`

```
# Cell 1
len(data.index), len(data.columns)
```

```
(569, 33)
```

```
# Cell 2
data.shape
```

```
(569, 33)
```

```
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_me
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.078
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.056
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.059
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.097
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.058

```
data.tail()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_me
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.056
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.055
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.056
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.070
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.058

2. Exploring Data Analysis

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
```

```
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
data.isna()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mea
0	False	False	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	.
564	False	False	False	False	False	False	False	False	False	False	False	Fals
565	False	False	False	False	False	False	False	False	False	False	False	Fals
566	False	False	False	False	False	False	False	False	False	False	False	Fals
567	False	False	False	False	False	False	False	False	False	False	False	Fals
568	False	False	False	False	False	False	False	False	False	False	False	Fals

569 rows × 33 columns

```
data.isna().any()
```

```
id                         False
diagnosis                  False
radius_mean                False
texture_mean               False
perimeter_mean             False
area_mean                  False
smoothness_mean            False
compactness_mean           False
concavity_mean             False
concave points_mean        False
symmetry_mean              False
fractal_dimension_mean     False
radius_se                  False
texture_se                 False
perimeter_se               False
area_se                    False
smoothness_se              False
compactness_se             False
concavity_se               False
concave points_se          False
symmetry_se                False
fractal_dimension_se       False
radius_worst               False
texture_worst              False
perimeter_worst            False
```

```
area_worst                 False
smoothness_worst           False
compactness_worst          False
concavity_worst            False
concave points_worst       False
symmetry_worst             False
fractal_dimension_worst    False
Unnamed: 32                 True
dtype: bool
```

```
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
```

```
data = data.dropna(axis='columns')
```

Get object features

- Using this method, we can see how many `object(categorical)` type of feature exists in dataset

```
data.describe(include="O")
```

	diagnosis
count	569
unique	2
top	B
freq	357

- *As we can see abouve result there are only one single feature is categorical and it's values are* `B` *and* `M`

To know how many unique values

```
data.diagnosis.value_counts()
```

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

using `value_counts` method we can see number of unique values in categorical type of feature.

Identify dependent and independent

```
data.head(2)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mea
0	842302	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.0787
1	842517	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.0566

```
diagnosis_unique = data.diagnosis.unique()
```

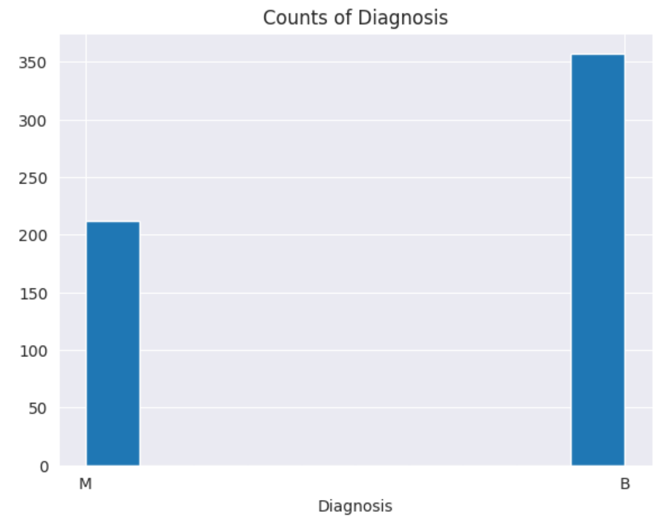
```
diagnosis_unique
```

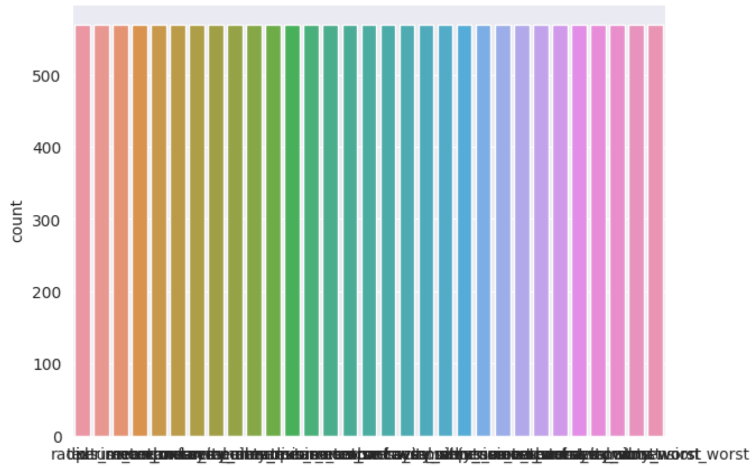
```
array(['M', 'B'], dtype=object)
```

3. Data Visualization.

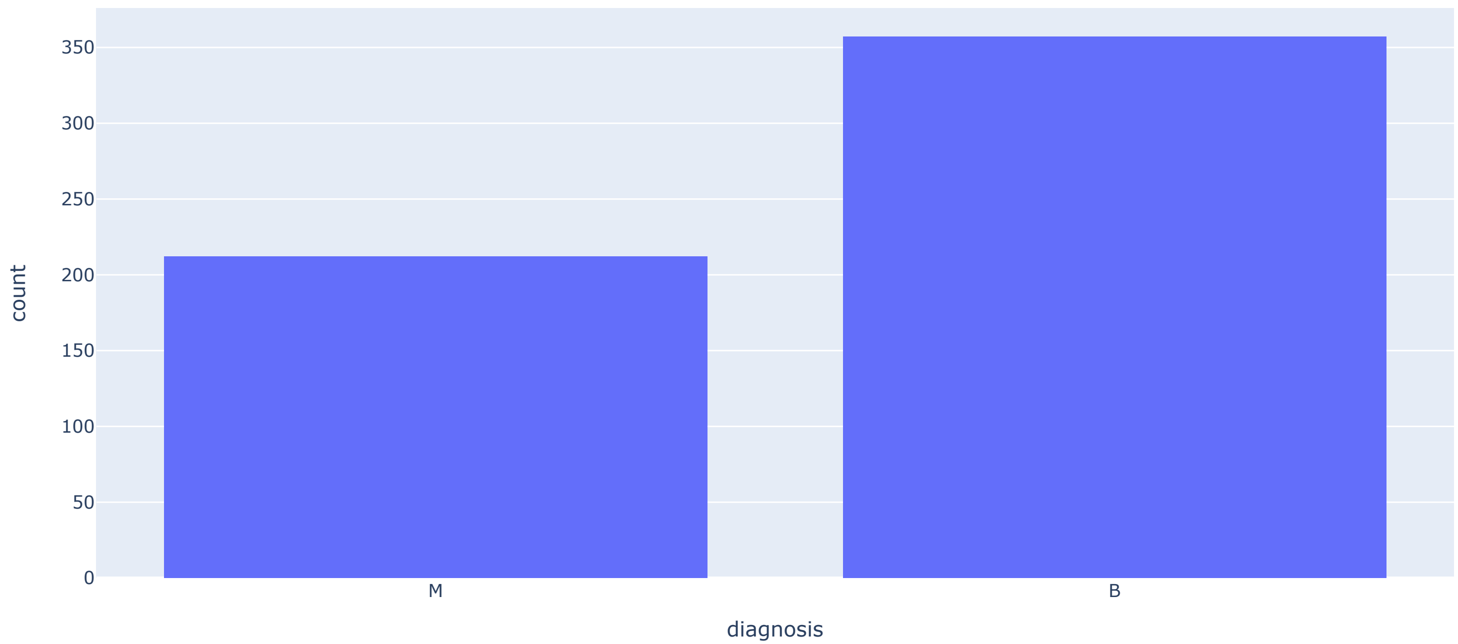
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

%matplotlib inline
sns.set_style('darkgrid')
```

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.hist( data.diagnosis)
# plt.legend()
plt.title("Counts of Diagnosis")
plt.xlabel("Diagnosis")


plt.subplot(1, 2, 2)

sns.countplot(data=data);

# plt.show()
```






```
# plt.figure(figsize=(7,12))
px.histogram(data, x='diagnosis')
# plt.show()
```

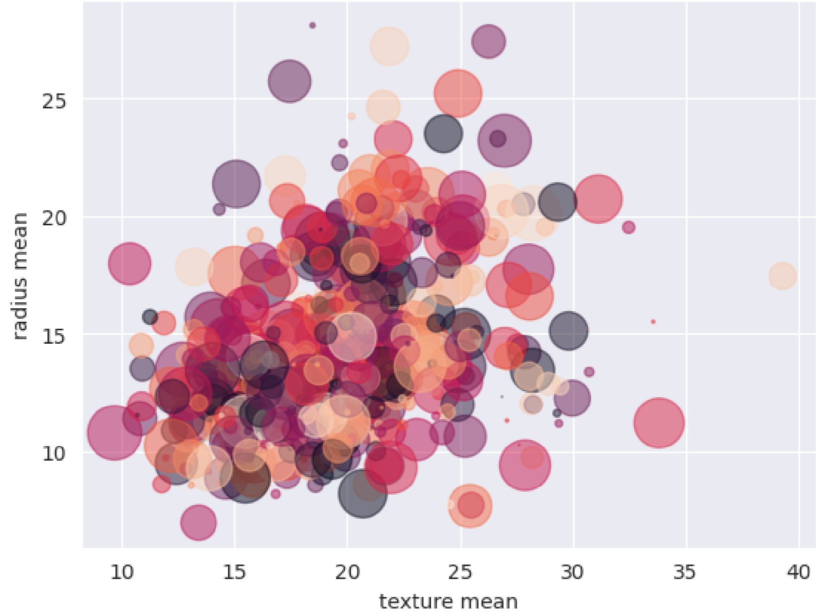



```
cols = ["diagnosis", "radius_mean", "texture_mean", "perimeter_mean", "area_mean"]

sns.pairplot(data[cols], hue="diagnosis")
plt.show()
```

radius_mean
texture_mean
perimeter_mean
area_mean
diagnosis
M
B

radius_mean texture_mean perimeter_mean area_mean

```
size = len(data['texture_mean'])

area = np.pi * (15 * np.random.rand( size ))**2
colors = np.random.rand( size )

plt.xlabel("texture mean")
plt.ylabel("radius mean")
plt.scatter(data['texture_mean'], data['radius_mean'], s=area, c=colors, alpha=0.5);
```



Data Filtering

- Now, we have one categorical feature, so we need to convert it into numeric values using `LabelEncoder` from `sklearn.preprocessing` packages

```
from sklearn.preprocessing import LabelEncoder
```

```
data.head(2)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mea
0	842302	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.0787
1	842517	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.0566

- LabelEncoder can be used to normalize labels.

```
labelencoder_Y = LabelEncoder()
data.diagnosis = labelencoder_Y.fit_transform(data.diagnosis)
```

After converting into numerical values, we can check it's values using this way,

```
data.head(2)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mea
0	842302	1	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.0787
1	842517	1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.0566

```
print(data.diagnosis.value_counts())
print("\n", data.diagnosis.value_counts().sum())
```

```
0    357
1    212
Name: diagnosis, dtype: int64

 569
```

Finnaly, We can see in this output categorical values converted into 0 and 1.

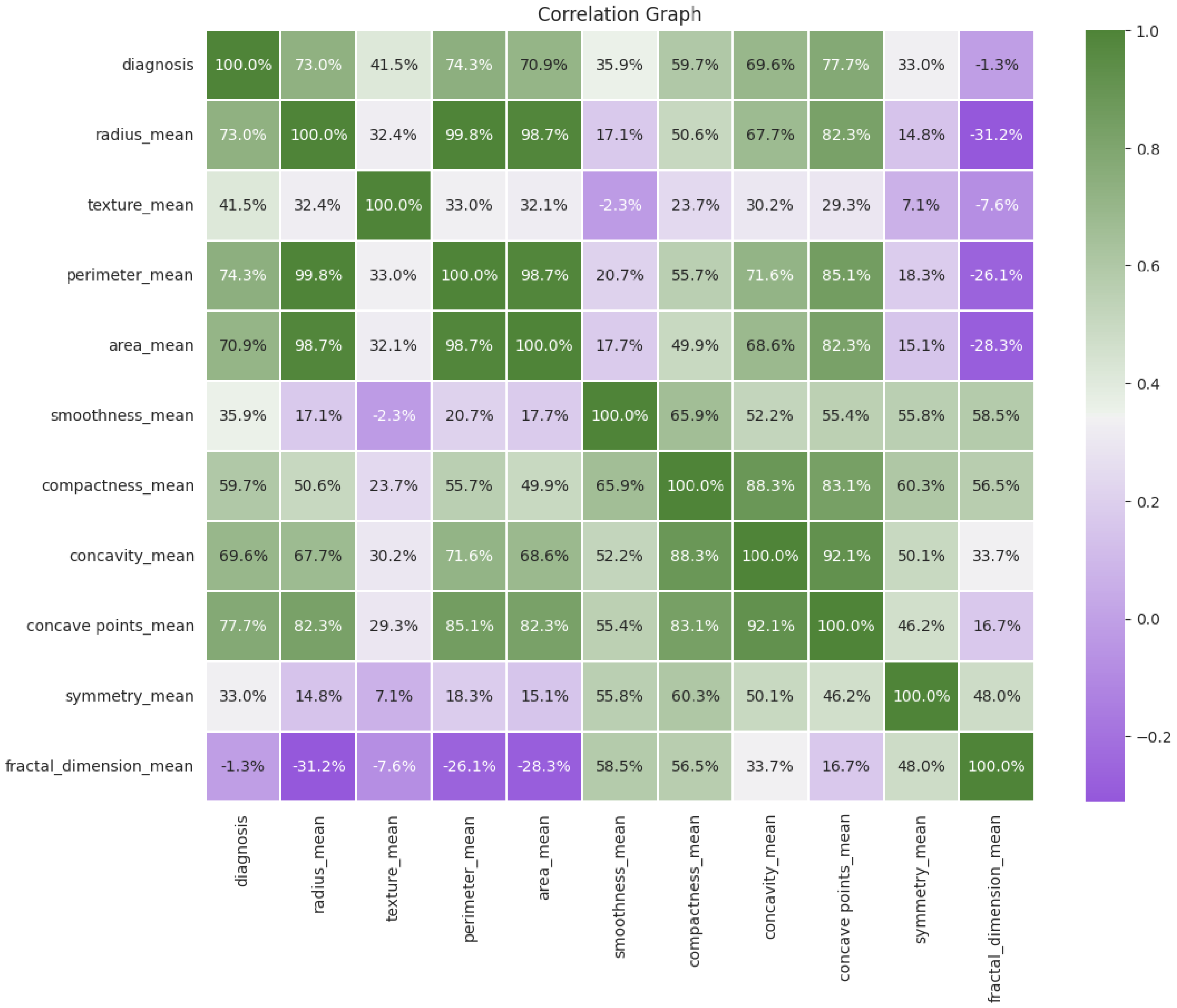
Find the correlation between other features, mean features only

```
cols = ['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
print(len(cols))
data[cols].corr()
```

```
11
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_d
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.696360	0.776614	0.330499	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	

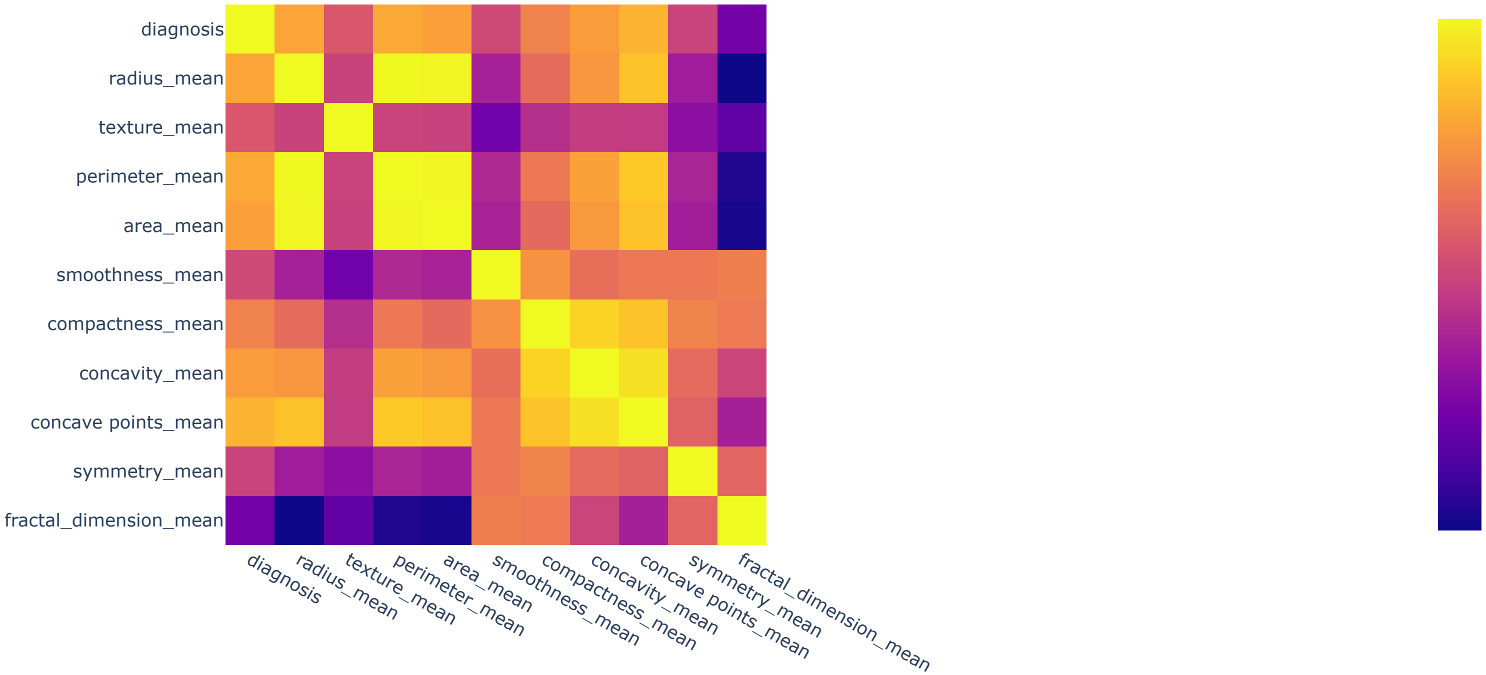
```
plt.figure(figsize=(12, 9))

plt.title("Correlation Graph")

cmap = sns.diverging_palette( 1000, 120, as_cmap=True)
sns.heatmap(data[cols].corr(), annot=True, fmt='.1%',  linewidths=.05, cmap=cmap);
```



Correlation Graph
diagnosis 100.0% 73.0% 41.5% 74.3% 70.9% 35.9% 59.7% 69.6% 77.7% 33.0% -1.3%
radius_mean 73.0% 100.0% 32.4% 99.8% 98.7% 17.1% 50.6% 67.7% 82.3% 14.8% -31.2%
texture_mean 41.5% 32.4% 100.0% 33.0% 32.1% -2.3% 23.7% 30.2% 29.3% 7.1% -7.6%
perimeter_mean 74.3% 99.8% 33.0% 100.0% 98.7% 20.7% 55.7% 71.6% 85.1% 18.3% -26.1%
area_mean 70.9% 98.7% 32.1% 98.7% 100.0% 17.7% 49.9% 68.6% 82.3% 15.1% -28.3%
smoothness_mean 35.9% 17.1% -2.3% 20.7% 17.7% 100.0% 65.9% 52.2% 55.4% 55.8% 58.5%
compactness_mean 59.7% 50.6% 23.7% 55.7% 49.9% 65.9% 100.0% 88.3% 83.1% 60.3% 56.5%
concavity_mean 69.6% 67.7% 30.2% 71.6% 68.6% 52.2% 88.3% 100.0% 92.1% 50.1% 33.7%
concave points_mean 77.7% 82.3% 29.3% 85.1% 82.3% 55.4% 83.1% 92.1% 100.0% 46.2% 16.7%
symmetry_mean 33.0% 14.8% 7.1% 18.3% 15.1% 55.8% 60.3% 50.1% 46.2% 100.0% 48.0%
fractal_dimension_mean -1.3% -31.2% -7.6% -26.1% -28.3% 58.5% 56.5% 33.7% 16.7% 48.0% 100.0%
diagnosis
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Using, Plotly Pacage we can show it in interactive graphs like this,

```
plt.figure(figsize=(15, 10))


fig = px.imshow(data[cols].corr());
fig.show()
```



Model Implementation

Train Test Splitting

Preprocessing and model selection

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

Import Machine Learning Models

```
from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier
```

Check the Model Accuracy, Errors and it's Validations

```
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score

from sklearn.metrics import classification_report

from sklearn.model_selection import KFold

from sklearn.model_selection import cross_validate, cross_val_score

from sklearn.svm import SVC

from sklearn import metrics
```

Feature Selection

Select feature for predictions

```
data.columns
```

```
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'],
      dtype='object')
```

- Take the dependent and independent feature for prediction

```
prediction_feature = [ "radius_mean",  'perimeter_mean', 'area_mean', 'symmetry_mean', 'compactness_mean', 'concave points_mean']

targeted_feature = 'diagnosis'

len(prediction_feature)
```

```
6
```

```
X = data[prediction_feature]
X

# print(X.shape)
# print(X.values)
```

	radius_mean	perimeter_mean	area_mean	symmetry_mean	compactness_mean	concave points_mean
0	17.99	122.80	1001.0	0.2419	0.27760	0.14710
1	20.57	132.90	1326.0	0.1812	0.07864	0.07017
2	19.69	130.00	1203.0	0.2069	0.15990	0.12790
3	11.42	77.58	386.1	0.2597	0.28390	0.10520
4	20.29	135.10	1297.0	0.1809	0.13280	0.10430
...	...	...	...	...	...	...
564	21.56	142.00	1479.0	0.1726	0.11590	0.13890
565	20.13	131.20	1261.0	0.1752	0.10340	0.09791
566	16.60	108.30	858.1	0.1590	0.10230	0.05302
567	20.60	140.10	1265.0	0.2397	0.27700	0.15200
568	7.76	47.92	181.0	0.1587	0.04362	0.00000

569 rows × 6 columns

```
y = data.diagnosis
y

# print(y.values)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

- Splite the dataset into TrainingSet and TestingSet by 33% and set the 15 fixed records

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=15)

print(X_train)
# print(X_test)
```

```
     radius_mean  perimeter_mean  area_mean  symmetry_mean  compactness_mean  \
274        17.93          115.20      998.9         0.1538           0.07027
189        12.30           78.83      463.7         0.1667           0.07253
158        12.06           76.84      448.6         0.1590           0.05241
257        15.32          103.20      713.3         0.2398           0.22840
486        14.64           94.21      666.0         0.1409           0.06698
..           ...             ...        ...            ...               ...
85         18.46          121.10     1075.0         0.2132           0.10530
199        14.45           94.49      642.7         0.1950           0.12060
156        17.68          117.40      963.7         0.1971           0.16650
384        13.28           85.79      541.8         0.1617           0.08575
456        11.63           74.87      415.1         0.1799           0.08574

     concave points_mean
274              0.04744
189              0.01654
158              0.01963
257              0.12420
486              0.02791
..                   ...
85               0.08795
199              0.05980
156              0.10540
384              0.02864
456              0.02017

[381 rows x 6 columns]
```

Perform Feature Standerd Scalling

Standardize features by removing the mean and scaling to unit variance

The standard score of a sample x is calculated as:

- z = (x - u) / s

```
# Scale the data to keep all the values in the same magnitude of 0 -1

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

ML Model Selecting and Model PredPrediction

Model Building

Now, we are ready to build our model for prediction, for the I made function for model building and preforming prediction and measure it's prediction and accuracy score.

Arguments

1. model => ML Model Object
2. Feature Training Set data
3. Feature Testing Set data
4. Targetd Training Set data
5. Targetd Testing Set data

```

```

```
def model_building(model, X_train, X_test, y_train, y_test):
    """

    Model Fitting, Prediction And Other stuff
    return ('score', 'accuracy_score', 'predictions' )
    """

    model.fit(X_train, y_train)
    score = model.score(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(predictions, y_test)

    return (score, accuracy, predictions)
```

Let's make a dictionary for multiple models for bulk predictions

```
models_list = {
    "LogisticRegression" :  LogisticRegression(),
    "RandomForestClassifier" :  RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=5),
    "DecisionTreeClassifier" :  DecisionTreeClassifier(criterion='entropy', random_state=0),
    "SVC" :  SVC(),
}

# print(models_list)
```

Before, sending it to the prediction check the key and values to store it's values in DataFrame below.

```
print(list(models_list.keys()))
print(list(models_list.values()))

# print(zip(list(models_list.keys()), list(models_list.values())))
```

```
['LogisticRegression', 'RandomForestClassifier', 'DecisionTreeClassifier', 'SVC']
[LogisticRegression(), RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=5), DecisionTreeClassifier(criterion='entropy', random_state=0), SVC()]
```

Model Implementing

Now, Train the model one by one and show the classification report of perticular models wise.

```
# Let's Define the function for confision metric Graphs

def cm_metrix_graph(cm):

    sns.heatmap(cm,annot=True,fmt="d")
    plt.show()
```

```
df_prediction = []
confusion_matrixs = []
df_prediction_cols = [ 'model_name', 'score', 'accuracy_score' , "accuracy_percentage"]

for name, model in zip(list(models_list.keys()), list(models_list.values())):

    (score, accuracy, predictions) = model_building(model, X_train, X_test, y_train, y_test )

    print("\n\nClassification Report of '"+ str(name), "'\n")

    print(classification_report(y_test, predictions))

    df_prediction.append([name, score, accuracy, "{0:.2%}".format(accuracy)])

    # For Showing Metrics
    confusion_matrixs.append(confusion_matrix(y_test, predictions))


df_pred = pd.DataFrame(df_prediction, columns=df_prediction_cols)
```

```
Classification Report of 'LogisticRegression '

              precision    recall  f1-score   support

           0       0.90      0.96      0.93       115
           1       0.92      0.84      0.88        73

    accuracy                           0.91       188
   macro avg       0.91      0.90      0.90       188
weighted avg       0.91      0.91      0.91       188



Classification Report of 'RandomForestClassifier '

              precision    recall  f1-score   support

           0       0.92      0.96      0.94       115
           1       0.93      0.88      0.90        73
```

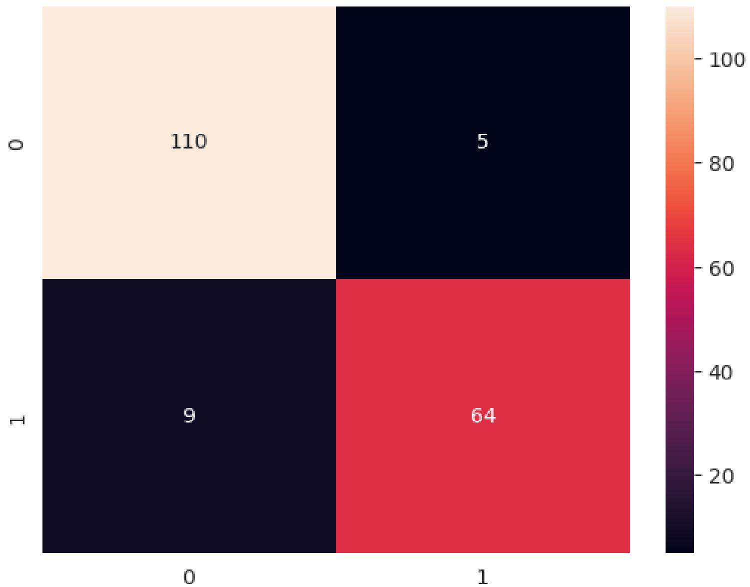
```
    accuracy                           0.93       188
   macro avg       0.93      0.92      0.92       188
weighted avg       0.93      0.93      0.93       188


Classification Report of 'DecisionTreeClassifier '

              precision    recall  f1-score   support

           0       0.90      0.96      0.93       115
           1       0.92      0.84      0.88        73

    accuracy                           0.91       188
   macro avg       0.91      0.90      0.90       188
weighted avg       0.91      0.91      0.91       188


Classification Report of 'SVC '

              precision    recall  f1-score   support

           0       0.90      0.97      0.93       115
           1       0.94      0.84      0.88        73

    accuracy                           0.91       188
   macro avg       0.92      0.90      0.91       188
weighted avg       0.92      0.91      0.91       188
```

```
print(len(confusion_matrixs))
```

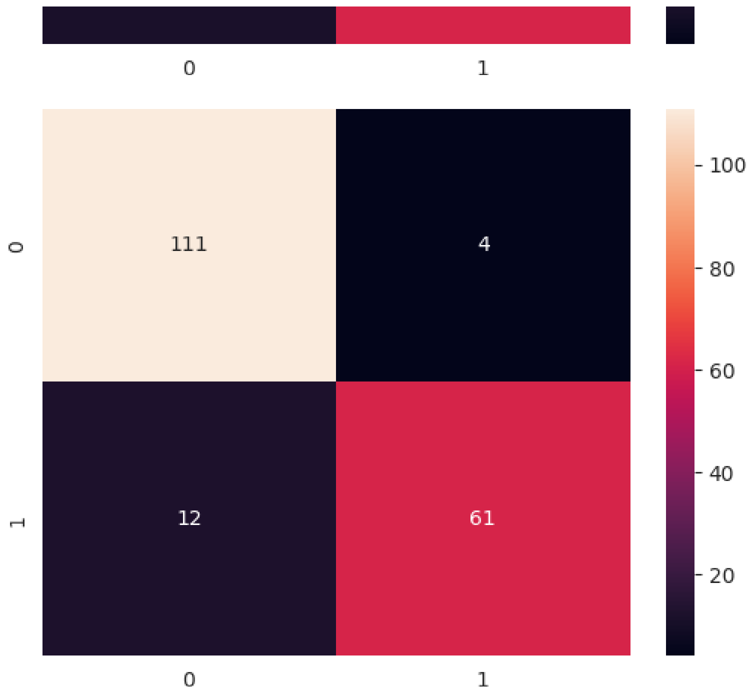
```
4
```

```
plt.figure(figsize=(10, 2))
# plt.title("Confusion Metric Graph")


for index, cm in enumerate(confusion_matrixs):

#     plt.xlabel("Negative Positive")
#     plt.ylabel("True Positive")



    # Show The Metrics Graph
    cm_metrix_graph(cm) # Call the Confusion Metrics Graph
    plt.tight_layout(pad=True)
```


0
1
110
5
12
61
0
1
100
75
50
25


0
1
110
5
9
64
0
1
100
80
60
40
20


0
1
110
5
12
61
100
80
60
40
20




```
<Figure size 640x480 with 0 Axes>
```

While Predicting we can store model's score and prediction values to new generated dataframe

```
df_pred
```

	model_name	score	accuracy_score	accuracy_percentage
0	LogisticRegression	0.916010	0.909574	90.96%
1	RandomForestClassifier	0.992126	0.925532	92.55%
2	DecisionTreeClassifier	1.000000	0.909574	90.96%
3	SVC	0.923885	0.914894	91.49%

- print the hightest accuracy score using sort values

```
df_pred.sort_values('score', ascending=False)
# df_pred.sort_values('accuracy_score', ascending=False)
```

	model_name	score	accuracy_score	accuracy_percentage
2	DecisionTreeClassifier	1.000000	0.909574	90.96%
1	RandomForestClassifier	0.992126	0.925532	92.55%
3	SVC	0.923885	0.914894	91.49%
0	LogisticRegression	0.916010	0.909574	90.96%

HyperTunning the ML Model

Tuning Parameters applying...

```
import warnings
warnings.filterwarnings('ignore')
```

```
from  sklearn.model_selection import GridSearchCV
```

For HyperTunning we can use `GridSearchCV` to know the best performing parameters

- GridSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.
- The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

```
# Let's Implement Grid Search Algorithm

# Pick the model
model = DecisionTreeClassifier()

# Tunning Params
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'min_samples_split': [2,3,4,5,6,7,8,9,10],
              'min_samples_leaf':[2,3,4,5,6,7,8,9,10] }


# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=10) # For 10 Cross-Validation

gsc.fit(X_train, y_train) # Model Fitting

print("\n Best Score is ")
print(gsc.best_score_)

print("\n Best Estinator is ")
print(gsc.best_estimator_)

print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is 
0.9237516869095816

 Best Estinator is 
DecisionTreeClassifier(max_features='sqrt', min_samples_leaf=3)

 Best Parametes are
{'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 2}
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Assuming you have X_train, y_train defined somewhere

# Create a Random Forest classifier
model = RandomForestClassifier()

# Simplified Tuning Params
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=5)

# Model Fitting
gsc.fit(X_train, y_train)

print("\n Best Score is ")
print(gsc.best_score_)

print("\n Best Estimator is ")
print(gsc.best_estimator_)

print("\n Best Parameters are")
print(gsc.best_params_)
```

```
 Best Score is
0.9132604237867396

 Best Estimator is
RandomForestClassifier(min_samples_leaf=2, min_samples_split=5)

 Best Parameters are
{'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
```

```

```

```
# Pick the model
model = SVC()


# Tunning Params
param_grid = [
              {'C': [1, 10, 100, 1000],
               'kernel': ['linear']
              },
              {'C': [1, 10, 100, 1000],
               'gamma': [0.001, 0.0001],
               'kernel': ['rbf']
              }
]


# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=10) # 10 Cross Validation

# Model Fitting
gsc.fit(X_train, y_train)

print("\n Best Score is ")
```