In [1]:
```
import numpy as np
import pandas as pd

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

In [2]:
```
data = pd.read_csv("breat_cancer_data.csv")
```

In [3]:
```
len(data.index), len(data.columns)
```

Out[3]:
```
(569, 33)
```

In [4]:
```
data.shape
```

Out[4]:
```
(569, 33)
```

In [5]:
```
data.head
```

Out[5]:
```
<bound method NDFrame.head of          id diagnosis  radius_mean  texture_mean  per
imeter_mean  area_mean  \
0      842302         M        17.99         10.38          122.80     1001.0
1      842517         M        20.57         17.77          132.90     1326.0
2    84300903         M        19.69         21.25          130.00     1203.0
3    84348301         M        11.42         20.38           77.58      386.1
4    84358402         M        20.29         14.34          135.10     1297.0
..        ...       ...          ...           ...             ...        ...
564    926424         M        21.56         22.39          142.00     1479.0
565    926682         M        20.13         28.25          131.20     1261.0
566    926954         M        16.60         28.08          108.30      858.1
567    927241         M        20.60         29.33          140.10     1265.0
568     92751         B         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220
1    ...          23.41           158.80      1956.0           0.12380
2    ...          25.53           152.50      1709.0           0.14440
3    ...          26.50            98.87       567.7           0.20980
4    ...          16.67           152.20      1575.0           0.13740
..   ...            ...              ...         ...               ...
564  ...          26.40           166.10      2027.0           0.14100
565  ...          38.25           155.00      1731.0           0.11660
566  ...          34.12           126.70      1124.0           0.11390
567  ...          39.42           184.60      1821.0           0.16500
568  ...          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 ...              ...                   ...             ...
564            0.21130           0.4107                0.2216          0.2060
565            0.19220           0.3215                0.1628          0.2572
566            0.30940           0.3403                0.1418          0.2218
567            0.86810           0.9387                0.2650          0.4087
568            0.06444           0.0000                0.0000          0.2871

     fractal_dimension_worst  Unnamed: 32
0                    0.11890          NaN
1                    0.08902          NaN
2                    0.08758          NaN
3                    0.17300          NaN
4                    0.07678          NaN
..                       ...          ...
```

```
564                 0.07115          NaN
565                 0.06637          NaN
566                 0.07820          NaN
567                 0.12400          NaN
568                 0.07039          NaN

[569 rows x 33 columns]>
```

In [7]:
```
data.tail()
```

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

5 rows × 33 columns

Data Analysis

In [8]:
```
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
```

In [9]:

```
data.isnull().sum()
```

Out[9]:
```
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
```

In [10]:
```
data = data.dropna(axis='columns')
data.describe(include="O")
```

Out[10]:

	diagnosis
count	569
unique	2
top	B
freq	357

In [11]:
```
data.diagnosis.value_counts()
```

Out[11]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [12]:
```
diagnosis_unique = data.diagnosis.unique()
diagnosis_unique
```

Out[12]:
```
array(['M', 'B'], dtype=object)
```

Data Visualization

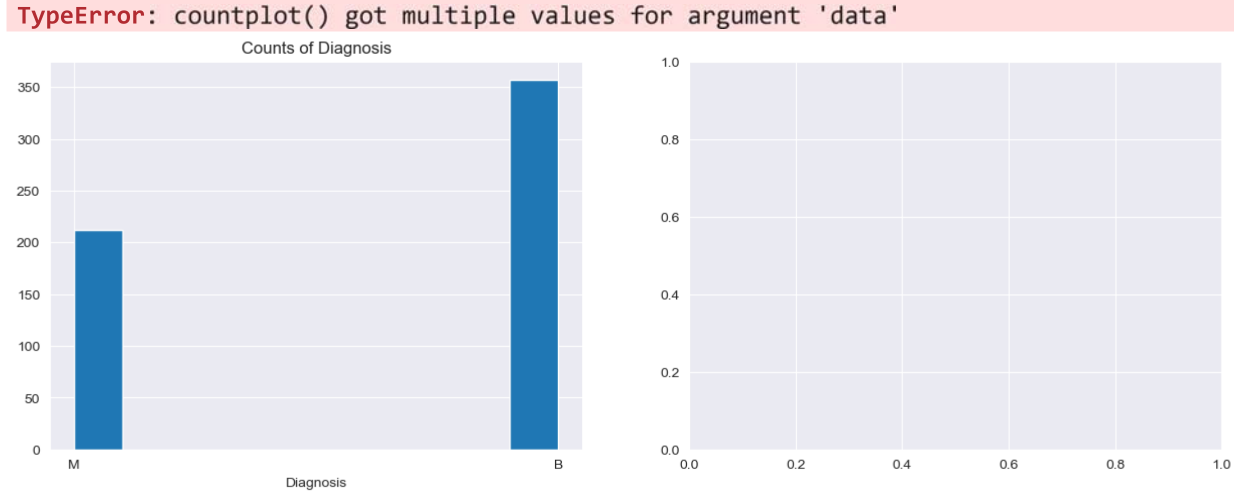
In [13]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in c:\users\admin\anaconda3\lib\site-packages
(5.9.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\admin\anaconda3\lib\site-p
ackages (from plotly) (8.0.1)
```

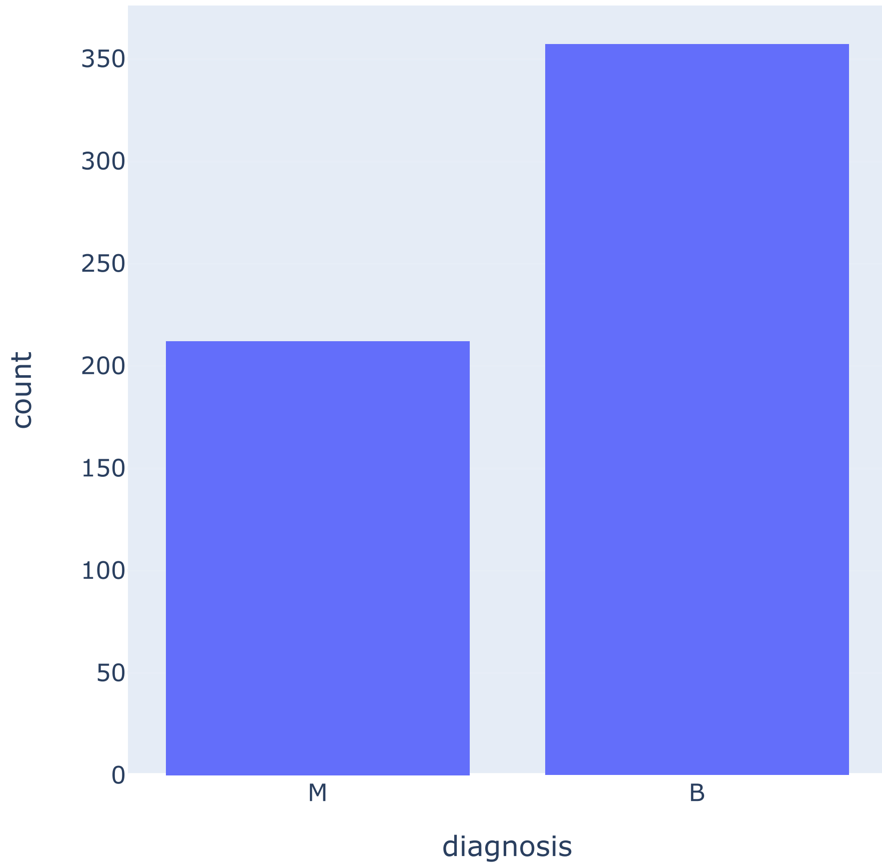
In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

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

In [19]:
```
plt.figure(figsize=(15, 5))

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

plt.subplot(1, 2, 2)

sns.countplot('diagnosis', data=data)
```

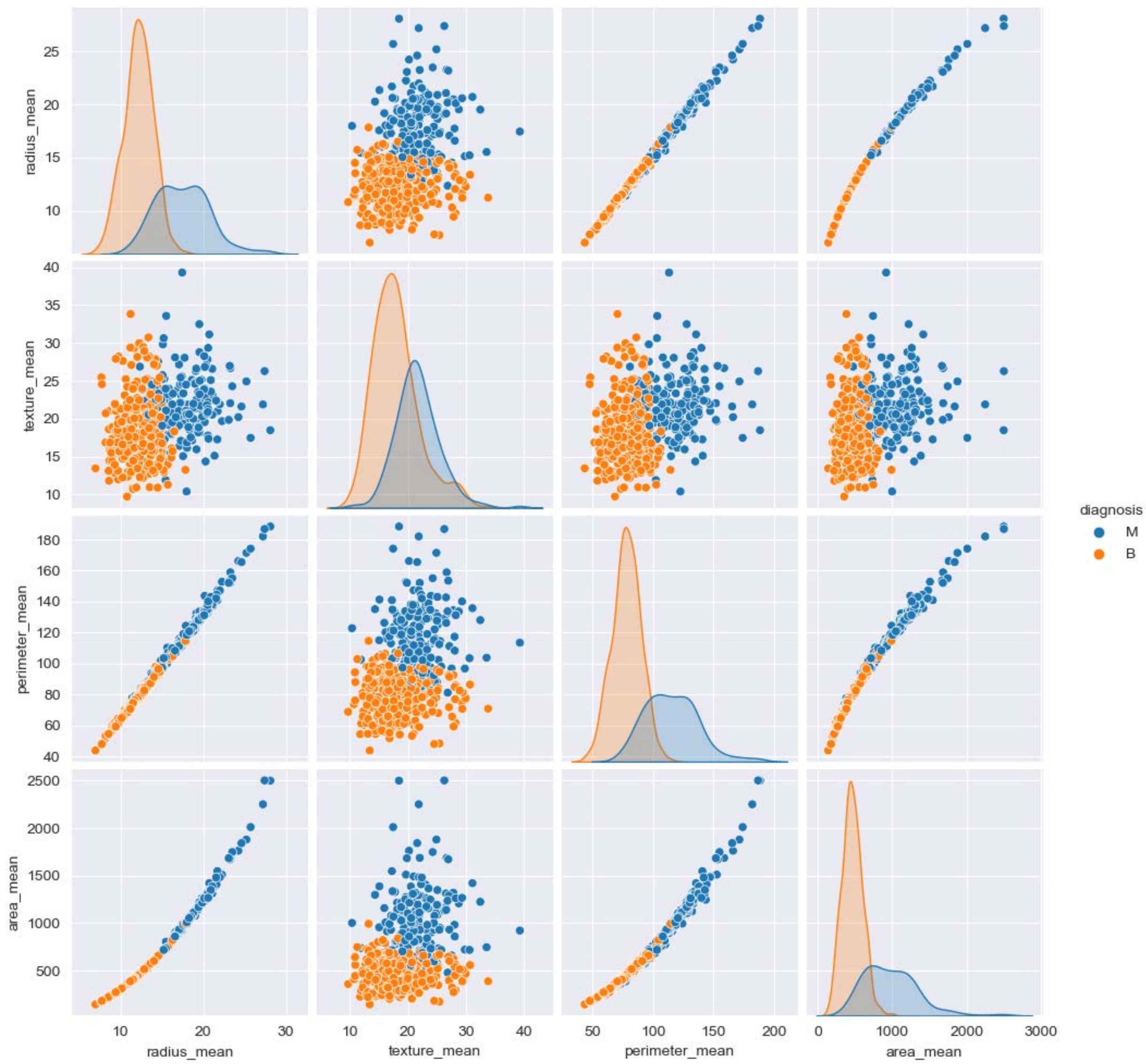
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 10
      6 plt.xlabel("Diagnosis")
      8 plt.subplot(1, 2, 2)
---> 10 sns.countplot('diagnosis', data=data)

TypeError: countplot() got multiple values for argument 'data'
```



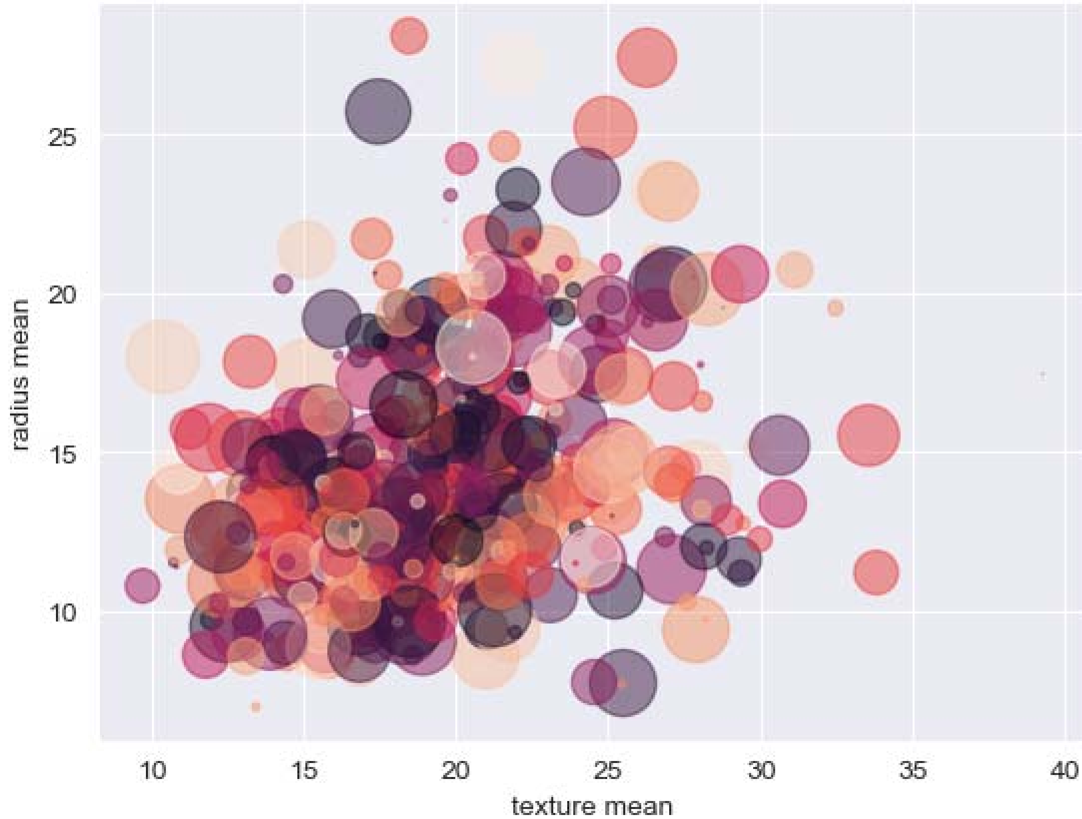
In [20]:
```
# plt.figure(figsize=(7,12))
px.histogram(data, x='diagnosis')
# plt.show()
```



In [21]:
```
cols = ["diagnosis", "radius_mean", "texture_mean", "perimeter_mean", "area_mean"]

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



In [22]:
```
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);
```



In [23]:
```
from sklearn.preprocessing import LabelEncoder
```

In [24]:
```
labelencoder_Y = LabelEncoder()
data.diagnosis = labelencoder_Y.fit_transform(data.diagnosis)
```

In [25]:
```
print(data.diagnosis.value_counts())
print("\n", data.diagnosis.value_counts().sum())
```

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

 569
```

In [26]:
```
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()
```

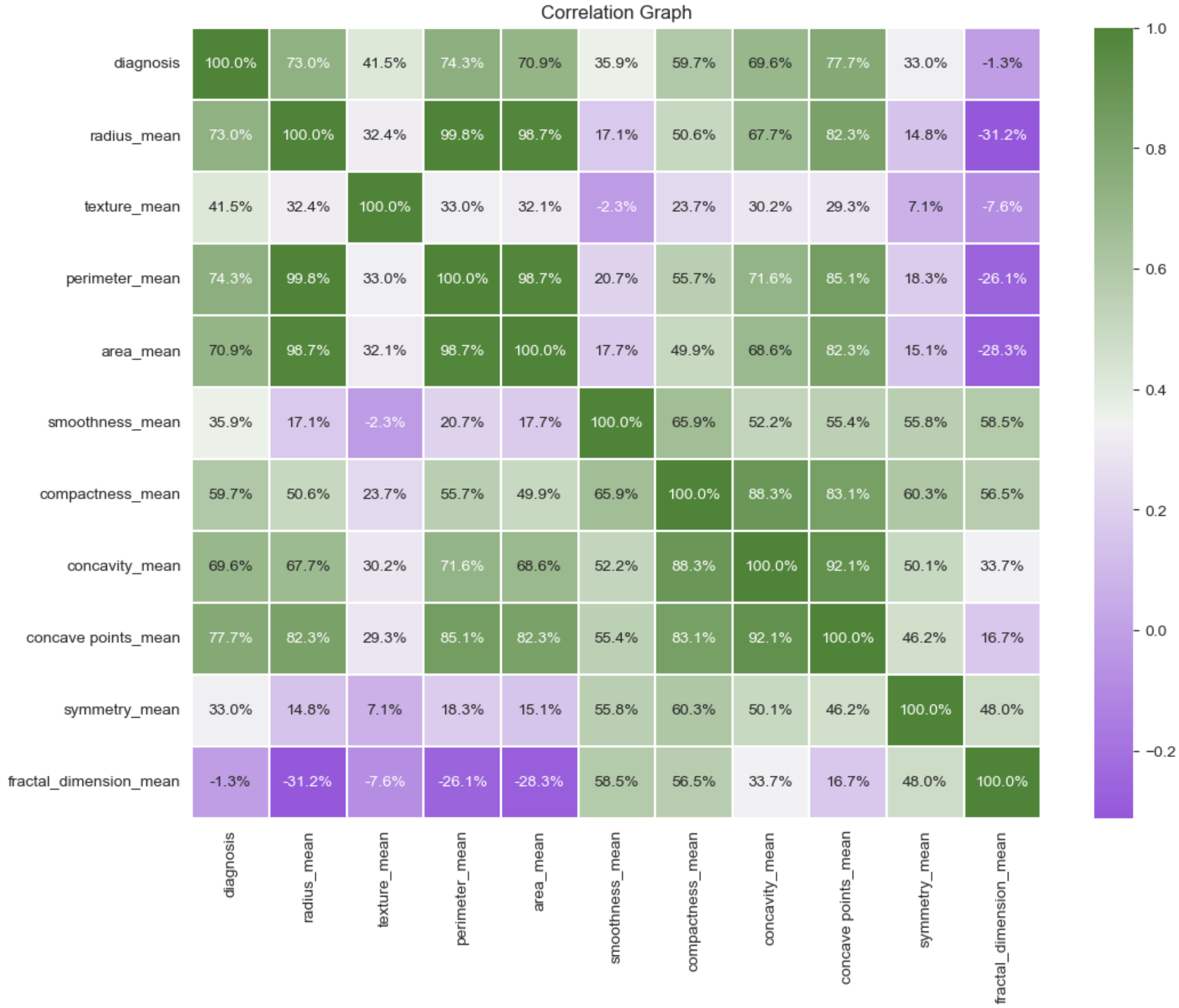
```
11
```

Out[26]:

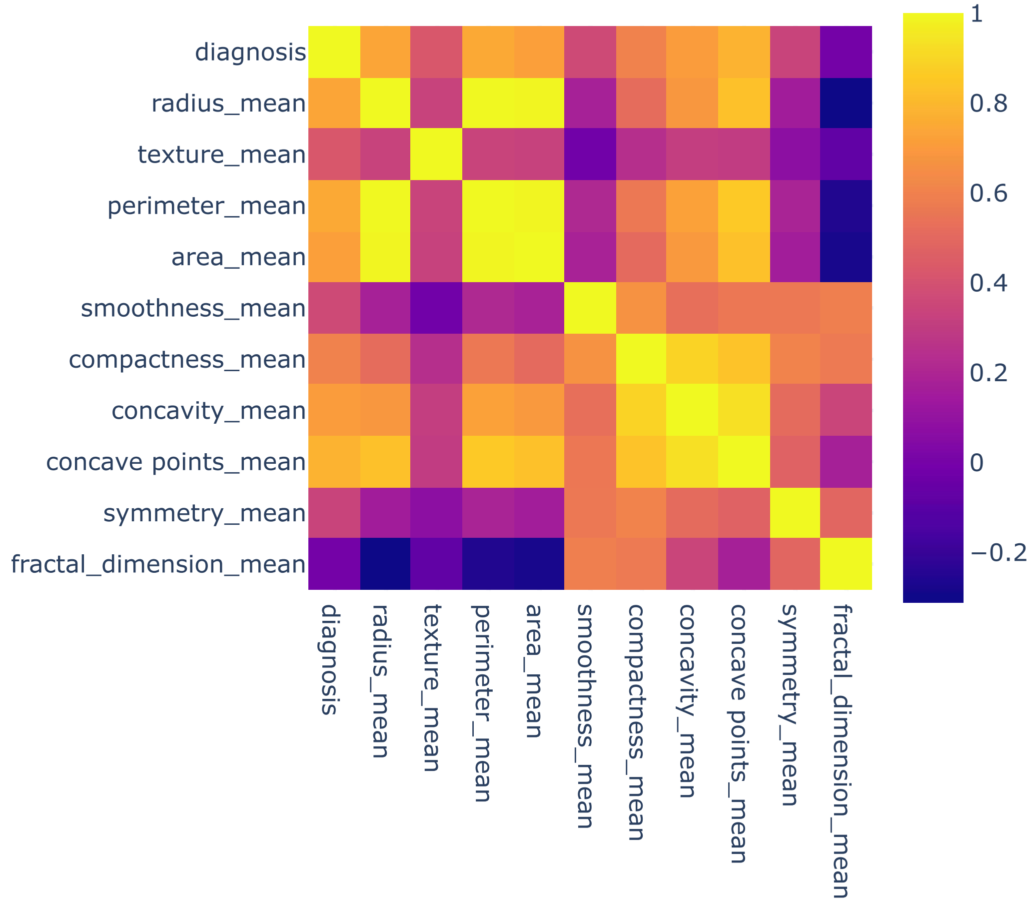
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	

In [27]:

```
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);
```



In [28]:
```
plt.figure(figsize=(15, 10))


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




```
<Figure size 1500x1000 with 0 Axes>
```

Model Implementation

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
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
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
```

In [30]:
```
data.columns
```

Out[30]:
```
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')
```

In [31]:
```
prediction_feature = [ "radius_mean",  'perimeter_mean', 'area_mean', 'symmetry_mean',
targeted_feature = 'diagnosis'

# len(prediction_feature)
```

Out[31]:
```
6
```

In [32]:
```
X = data[prediction_feature]
X
```

Out[32]:

	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

In [33]:
```
y = data.diagnosis
y
```

Out[33]:
```
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: int32
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
```

In [35]:
```
sc = StandardScaler()

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

In [36]:
```
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)
```

In [37]:
```
models_list = {
    "LogisticRegression" :  LogisticRegression(),
    "RandomForestClassifier" :  RandomForestClassifier(n_estimators=10, criterion='ent
    "DecisionTreeClassifier" :  DecisionTreeClassifier(criterion='entropy', random_sta
    "SVC" :  SVC(),
}
```

In [38]:
```
print(list(models_list.keys()))
print(list(models_list.values()))
```

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

In [39]:
```
def cm_metrix_graph(cm):

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

In [40]:
```
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())):
```

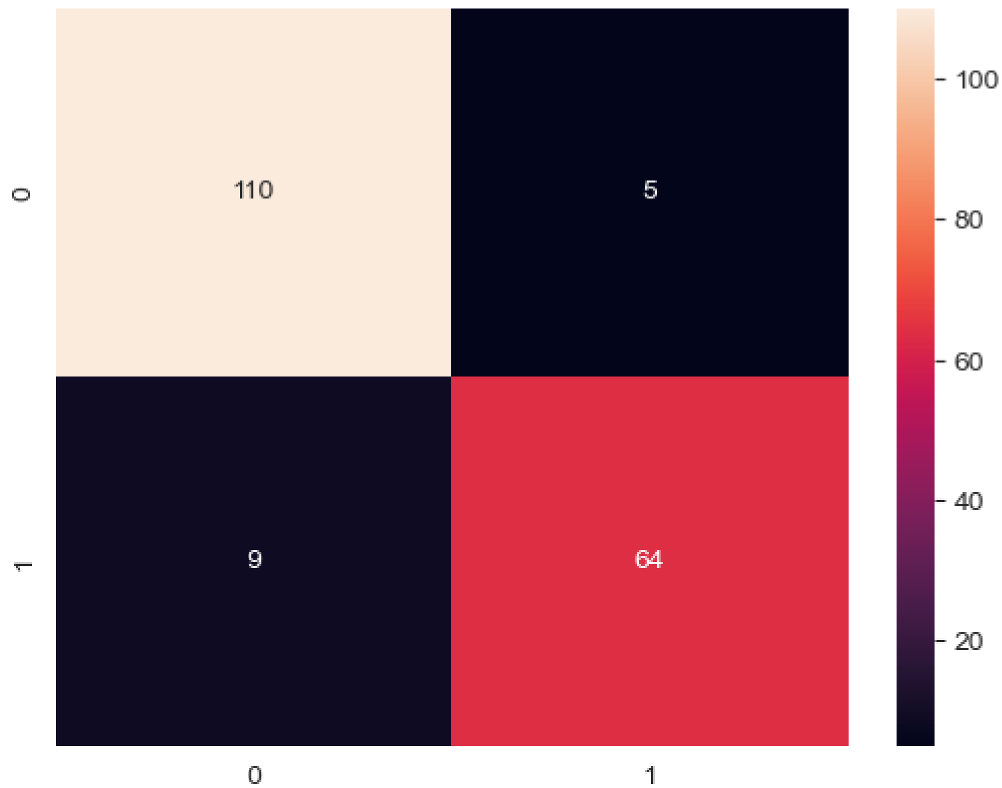
```
    (score, accuracy, predictions) = model_building(model, X_train, X_test, y_train, y

    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)
```

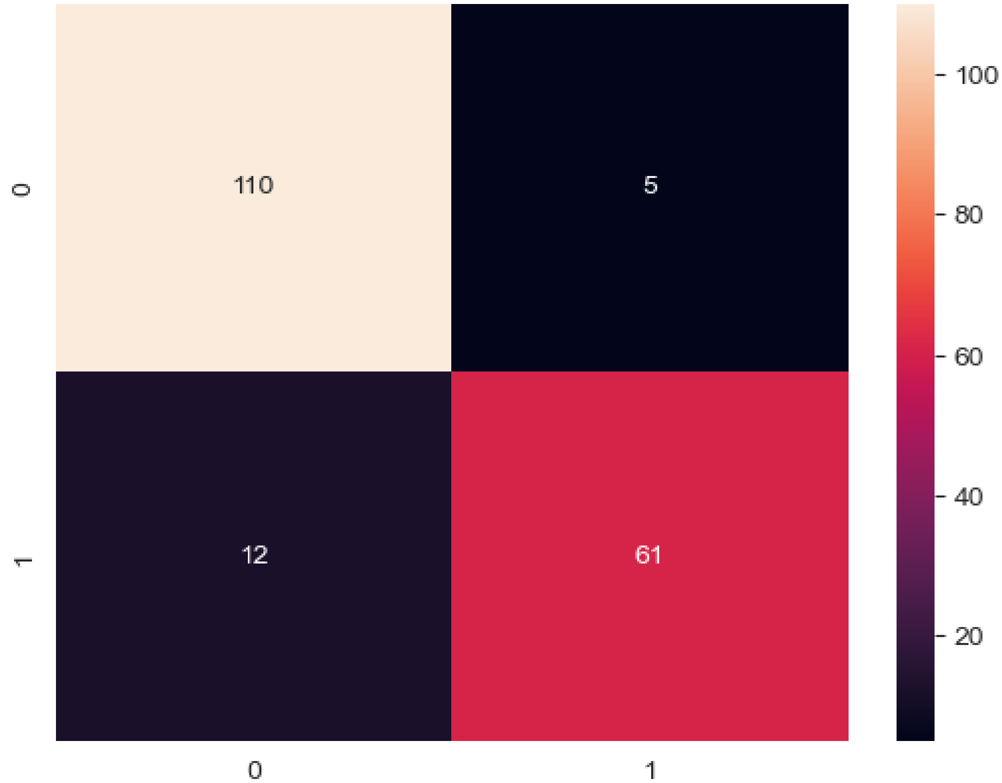
```
Classification Report of 'LogisticRegression '

              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
```

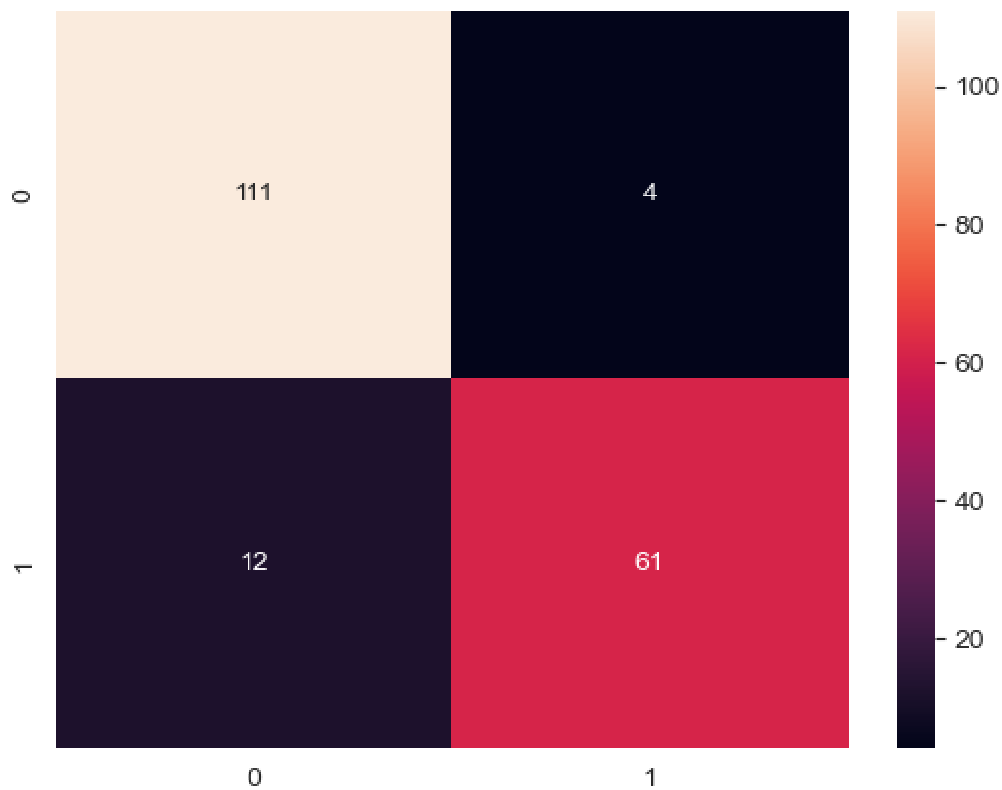
In [41]:

```python
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)
```








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

In [42]:
```
df_pred
```

Out[42]:

	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%

In [43]:

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

Out[43]:

	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%

K-fold

In [44]:

```
len(data)
```

Out[44]: 569

In [45]:

```
cv_score = cross_validate(LogisticRegression(), X, y, cv=3,
                          scoring=('r2', 'neg_mean_squared_error'),
                          return_train_score=True)

pd.DataFrame(cv_score).describe().T
```

Out[45]:

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
est_neg_mean_squared_error	3.0	-0.108902	0.043669	-0.157895	-0.126316	-0.094737	-0.084405
train_neg_mean_squared_error	3.0	-0.106321	0.012102	-0.113456	-0.113307	-0.113158	-0.102753

In [46]:

```
def cross_val_scorring(model):

#     (score, accuracy, predictions) = model_building(model, X_train, X_test, y_train,
    model.fit(data[prediction_feature], data[targeted_feature])

    # score = model.score(X_train, y_train)
    predictions = model.predict(data[prediction_feature])
    accuracy = accuracy_score(predictions, data[targeted_feature])
```

```
    print("\nFull-Data Accuracy:", round(accuracy, 2))
    print("Cross Validation Score of'"+ str(name), "'\n")

    # Initialize K folds.
    kFold = KFold(n_splits=5) # define 5 diffrent data folds

    err = []

    for train_index, test_index in kFold.split(data):
        # print("TRAIN:", train_index, "TEST:", test_index)

        # Data Spliting via fold indexes
        X_train = data[prediction_feature].iloc[train_index, :] # train_index = rows a
        y_train = data[targeted_feature].iloc[train_index] # all targeted features tra

        X_test = data[prediction_feature].iloc[test_index, :] # testing all rows and c
        y_test = data[targeted_feature].iloc[test_index] # all targeted tests

        # Again Model Fitting
        model.fit(X_train, y_train)

        err.append(model.score(X_train, y_train))

        print("Score:", round(np.mean(err),  2) )
```

In [47]:
```
for name, model in zip(list(models_list.keys()), list(models_list.values())):
    cross_val_scorring(model)
```

```
Full-Data Accuracy: 0.9
Cross Validation Score of'LogisticRegression '

Score: 0.91
Score: 0.91
Score: 0.9
Score: 0.9
Score: 0.9

Full-Data Accuracy: 1.0
Cross Validation Score of'RandomForestClassifier '

Score: 0.99
Score: 0.99
Score: 0.99
Score: 1.0
Score: 1.0

Full-Data Accuracy: 1.0
Cross Validation Score of'DecisionTreeClassifier '

Score: 1.0
Score: 1.0
Score: 1.0
Score: 1.0
Score: 1.0

Full-Data Accuracy: 0.89
Cross Validation Score of'SVC '

Score: [illegible]
Score: [illegible]
Score: [illegible]
Score: [illegible]
Score: [illegible]
```

HyperTuning the ML model

(1) Decision Tree Classifier

In [48]:
```
from  sklearn.model_selection import GridSearchCV
```

In [49]:
```
# 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 
0.9185560053981108

 Best Estinator is 
DecisionTreeClassifier(max_features='log2', min_samples_leaf=9)

 Best Parametes are
{'max_features': 'log2', 'min_samples_leaf': 9, 'min_samples_split': 2}
```

(2) KNeighborsClassifier

In [50]:

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


# Tunning Params
param_grid = {
    'n_neighbors': list(range(1, 30)),
    'leaf_size': list(range(1,30)),
    'weights': [ 'distance', 'uniform' ]
}


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

# Model Fitting
gsc.fit(X_train, y_train)

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 
0.9159244264507423

 Best Estinator is 
KNeighborsClassifier(leaf_size=1, n_neighbors=10)

 Best Parametes are
{'leaf_size': 1, 'n_neighbors': 10, 'weights': 'uniform'}
```

(3) SVC

In [51]:
```
# 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 ")
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 
0.9184885290148447

 Best Estinator is 
SVC(C=10, gamma=0.001)

 Best Parametes are
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

(4) RandomForestClassifier

In [55]:
```
# Pick the model
model = RandomForestClassifier()


# Tunning Params
random_grid = {'bootstrap': [True, False],
 'max_depth': [40, 50, None], # 10, 20, 30, 60, 70, 100,
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2], # , 4
 'min_samples_split': [2, 5], # , 10
 'n_estimators': [200, 400]} # , 600, 800, 1000, 1200, 1400, 1600, 1800, 2000
```

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

# Model Fitting
gsc.fit(X_train, y_train)

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 
0.9105937921727396

 Best Estinator is 
RandomForestClassifier(min_samples_split=5, n_estimators=400)

 Best Parametes are
{'bootstrap': True, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 400}
```

In [ ]: