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

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

```
.read_csv("https://raw.githubusercontent.com/ingledarshan/AIML-B2/main/data.csv")
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	0
1	842517	M	20.57	17.77	132.90	1326.0	0
2	84300903	M	19.69	21.25	130.00	1203.0	0
3	84348301	M	11.42	20.38	77.58	386.1	0
4	84358402	M	20.29	14.34	135.10	1297.0	0

```
df.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', 'Unnamed: 32'],
      dtype='object')
```



```
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
 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
```

```
df['Unnamed: 32']
```

```
0     NaN
1     NaN
2     NaN
...
564   NaN
565   NaN
566   NaN
567   NaN
568   NaN
Name: Unnamed: 32, Length: 569, dtype: float64
```

Saved successfully!

```
df = df.drop("Unnamed: 32", axis=1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-71-1f8131bd2929> in <module>()
----> 1 df = df.drop("Unnamed: 32", axis=1)

                                 3 frames
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   5285         if mask.any():
   5286             if errors != "ignore":
-> 5287                 raise KeyError(f"{labels[mask]} not found in axis")
   5288             indexer = indexer[~mask]
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	0
1	842517	M	20.57	17.77	132.90	1326.0	0
2	84300903	M	19.69	21.25	130.00	1203.0	0
3	84348301	M	11.42	20.38	77.58	386.1	0
4	84358402	M	20.29	14.34	135.10	1297.0	0

```
df.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')
```

Saved successfully!

```
df.drop('id' , axis=1, inplace=True)
#df = df.drop('id', axis=1)
```

```
df.columns
```

```
Index(['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')
```

```
type(df.columns)
```

```
pandas.core.indexes.base.Index
```

```
l = list(df.columns)
print(l)
```

```
['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_
```

```
features_mean = l[1:11]

features_se = l[11:21]

features_worst = l[21:]
```

```
print(features_mean)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compa
```

```
print(features_se)
```

```
['radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se
```

Saved successfully!

```
['radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
```

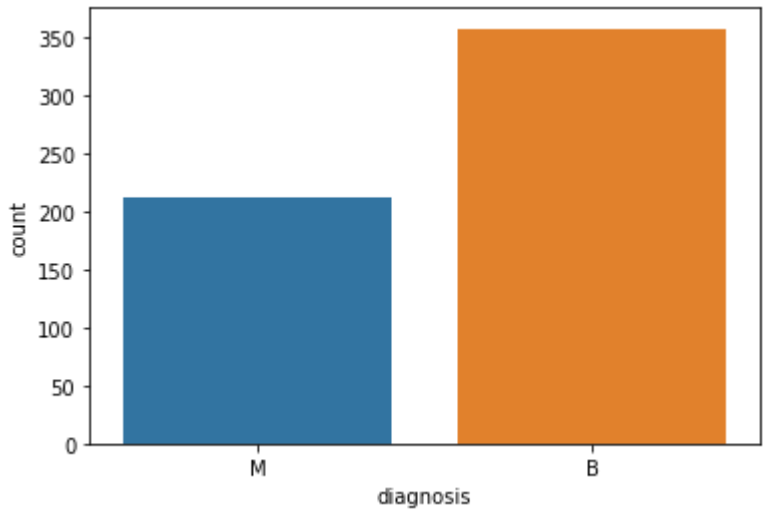
```
df.head(2)
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	M	17.99	10.38	122.8	1001.0	0.11840	
1	M	20.57	17.77	132.9	1326.0	0.08474	

```
df['diagnosis'].unique()
# M=Malignant, B= Benign
```

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

```
sns.countplot(df['diagnosis'], label="count,");
```




```
df['diagnosis'].value_counts()
```

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

```
df.shape
```

```
(569, 31)
```

Saved successfully!

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactr

```
len(df.columns)
```

```
31
```

std	3.524049	4.301036	24.298981	351.914129	0.014064	

```
#Correlation Plot
corr = df.corr()
corr
```

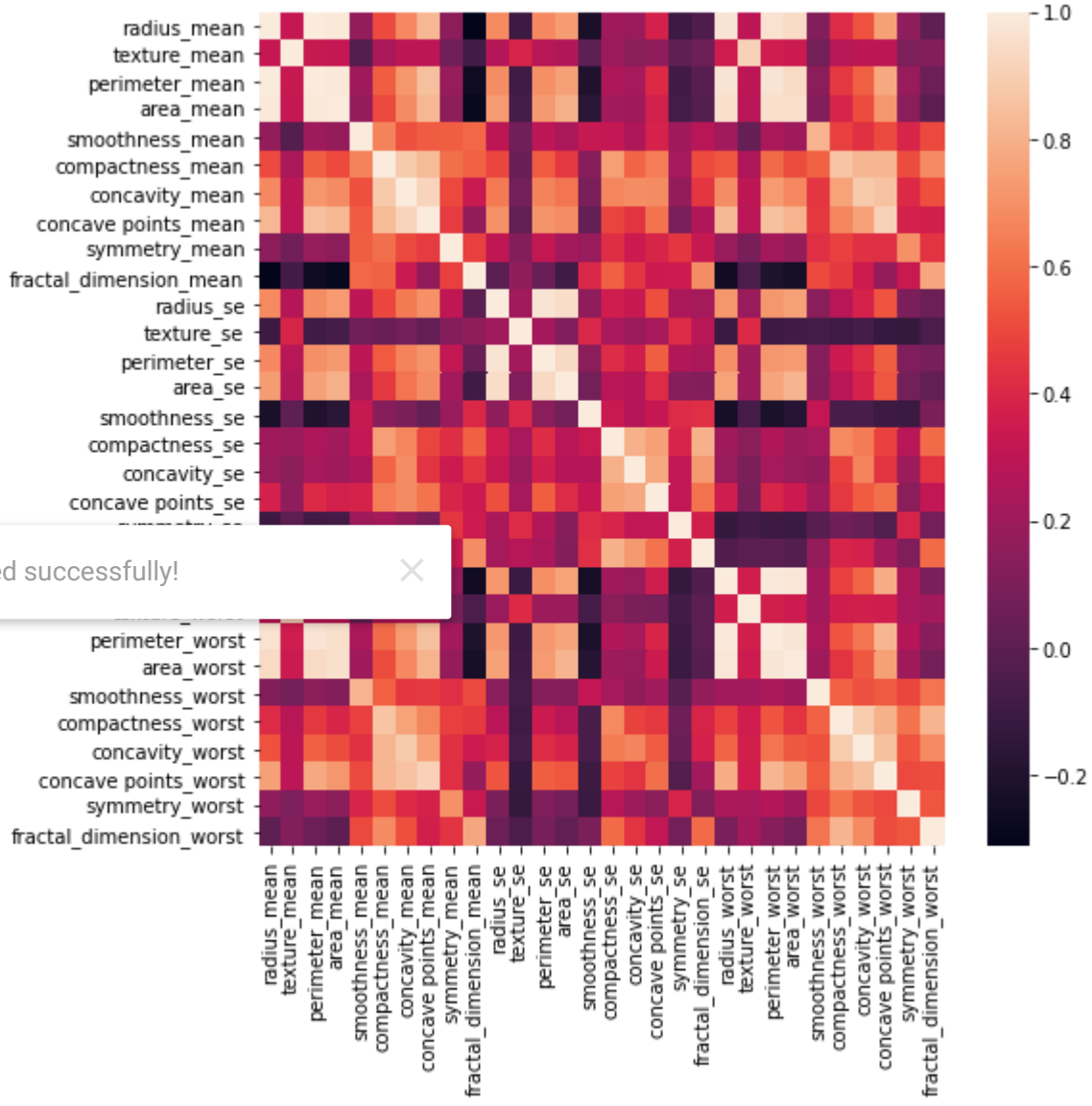
Saved successfully!

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
radius_mean	1.000000	0.323782	0.997855	0.987357	0.1
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.0
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.2
area_mean	0.987357	0.321086	0.986507	1.000000	0.1
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.0

```
corr.shape
```

```
(30, 30)
```

```
plt.figure(figsize=(8,8))
sns.heatmap(corr);
```



Saved successfully!

fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.4

```
df.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

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

```
df.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	

```
df['diagnosis'].unique()
```

Saved successfully!

Double-click (or enter) to edit

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

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_m
0	17.99	10.38	122.80	1001.0	0.11840	0.27

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

```
0    1
1    1
2    1
3    1
4    1
Name: diagnosis, dtype: int64
```

```
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)
```

```
df.shape
```

```
(569, 31)
```

```
X_train.shape
```

```
(398, 30)
```

```
X_test.shape
```

```
(171, 30)
```

```
y_train.shape
```



```
y_test.shape
```

```
(171,)
```

```
X_train.head(1)
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
425	10.03	21.28	63.19	307.3	0.08117	0.

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

```
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

```
X_train
```

```
array([[-1.18243367,  0.41639142, -1.20666942, ..., -1.39224566,
        -0.87931159, -0.2301983 ],
       [-0.89153024, -1.02371423, -0.91535792, ..., -0.94047776,
        -0.50647409,  0.34232864],
       [ 1.77743813,  1.87435885,  1.70685478, ...,  0.58827933,
        -0.74317176, -0.09085036],
       ...,
       [ 0.0094231 ,  0.61063823,  0.15292065, ...,  0.92517041,
        -0.11352501,  1.47068164],
       [-0.86328719,  0.44541681, -0.88917261, ..., -1.13016868,
         0.09068474, -0.71974276],
       [ 0.54321678, -1.01701606,  0.47532721, ..., -0.07032756,
        -0.21562989, -0.77767392]])
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_pred = lr.predict(X_test)
```

```
y_pred
```

```
array([1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

Saved successfully!

```
y_test
```

```
489    1
217    0
127    1
134    1
99     1
      ..
214    1
```

```
144    0
548    0
293    0
229    1
Name: diagnosis, Length: 171, dtype: int64
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9824561403508771
```

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

```
0.9824561403508771
```

```
results = pd.DataFrame()
results
```

```
tempResults = pd.DataFrame({'Algorithm':['Logistic Regression Method'], 'Accuracy
results = pd.concat([results, tempResults])
results = results[['Algorithm','Accuracy']]
results
```

	Algorithm	Accuracy
0	Logistic Regression Method	0.982456

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
```

Saved successfully!

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
y_pred = dtc.predict(X_test)
y_pred
```

```
array([0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
```

```
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.935672514619883
```

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

```
0.935672514619883
```

```
tempResults = pd.DataFrame({'Algorithm':['Decision tree Classifier Method'], 'Acc
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

	Algorithm	Accuracy
0	Logistic Regression Method	0.982456
0	Decision tree Classifier Method	0.935673

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       les_leaf=1, min_samples_split=2,
                       ht_fraction_leaf=0.0, n_estimators=100,
                       one, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```



```
y_pred = rfc.predict(X_test)
y_pred
```

```
array([1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9532163742690059
```

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

```
0.9532163742690059
```

```
tempResults = pd.DataFrame({'Algorithm' :['Random Forest Classifier Method'], 'Ac
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

	Algorithm	Accuracy
0	Logistic Regression Method	0.982456
0	Decision tree Classifier Method	0.935673
0	Random Forest Classifier Method	0.953216

```
from sklearn import svm
svc = svm.SVC()
svc.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred = svc.predict(X_test)
```

Saved successfully!

```
1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9824561403508771
```

```
svc_acc = accuracy_score(y_test, y_pred)
```

```
print(svc_acc)
```

```
0.9824561403508771
```

```
tempResults = pd.DataFrame({'Algorithm':['Support Vector Classifier Method'], 'Ac
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

	Algorithm	Accuracy
0	Logistic Regression Method	0.982456
0	Decision tree Classifier Method	0.935673
0	Random Forest Classifier Method	0.953216
0	Support Vector Classifier Method	0.982456

Saved successfully!