Breast Cancer (Diagnostic) Data Set

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

warnings.filterwarnings("ignore")
import datetime

data=pd.read_csv(r"C:\Users\Admin\Downloads\breast_cancer.csv")
data.head()
```

```
   Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape
\
0                5                        1                         1

1                5                        4                         4

2                3                        1                         1

3                6                        8                         8

4                4                        1                         1


   Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                  1                            2            1
1                  5                            7           10
2                  1                            2            2
3                  1                            3            4
4                  3                            2            1

   Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                3                1        1      2
1                3                2        1      2
2                3                1        1      2
3                3                7        1      2
4                3                1        1      2
```

```
data.shape
```

```
(683, 10)
```

```
#description of dataset
data.describe()
```

```
       Clump Thickness  Uniformity of Cell Size  Uniformity of Cell
Shape  \
count       683.000000               683.000000
683.000000
mean          4.442167                 3.150805
3.215227
std           2.820761                 3.065145
2.988581
min           1.000000                 1.000000
1.000000
25%           2.000000                 1.000000
1.000000
50%           4.000000                 1.000000
1.000000
75%           6.000000                 5.000000
5.000000
max          10.000000                10.000000
10.000000

       Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
count         683.000000                   683.000000   683.000000
mean            2.830161                     3.234261     3.544656
std             2.864562                     2.223085     3.643857
min             1.000000                     1.000000     1.000000
25%             1.000000                     2.000000     1.000000
50%             1.000000                     2.000000     1.000000
75%             4.000000                     4.000000     6.000000
max            10.000000                    10.000000    10.000000

       Bland Chromatin  Normal Nucleoli     Mitoses       Class
count       683.000000       683.000000  683.000000  683.000000
mean          3.445095         2.869693    1.603221    2.699854
std           2.449697         3.052666    1.732674    0.954592
min           1.000000         1.000000    1.000000    2.000000
25%           2.000000         1.000000    1.000000    2.000000
50%           3.000000         1.000000    1.000000    2.000000
75%           5.000000         4.000000    1.000000    4.000000
max          10.000000        10.000000   10.000000    4.000000

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683 entries, 0 to 682
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Clump Thickness              683 non-null    int64
 1   Uniformity of Cell Size      683 non-null    int64
 2   Uniformity of Cell Shape     683 non-null    int64
 3   Marginal Adhesion            683 non-null    int64
```

```
 4   Single Epithelial Cell Size  683 non-null    int64
 5   Bare Nuclei                  683 non-null    int64
 6   Bland Chromatin              683 non-null    int64
 7   Normal Nucleoli              683 non-null    int64
 8   Mitoses                      683 non-null    int64
 9   Class                        683 non-null    int64
dtypes: int64(10)
memory usage: 53.5 KB
```

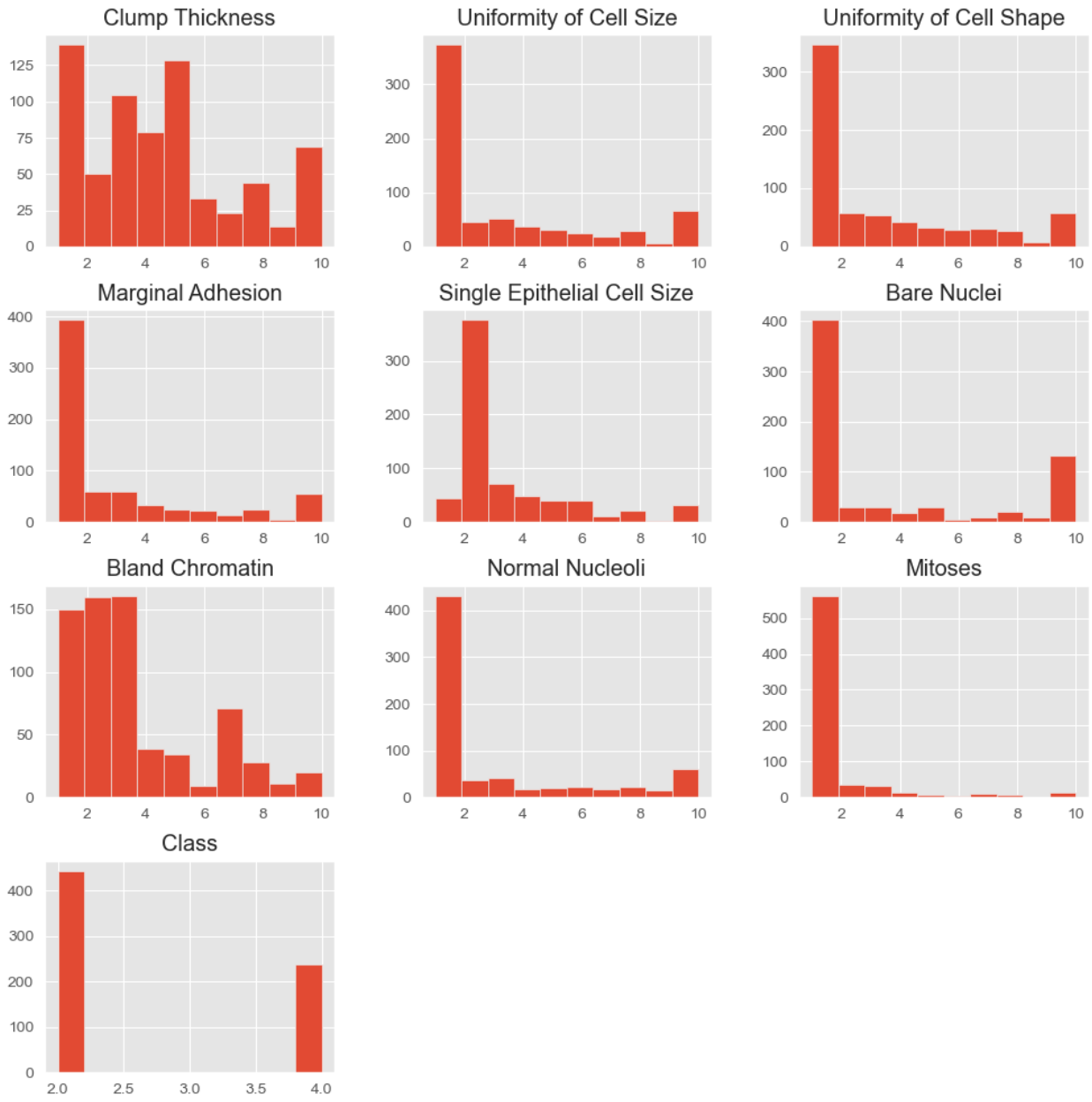
```
data.columns
```

```
Index(['Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland
Chromatin',
       'Normal Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

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

```
Clump Thickness                0
Uniformity of Cell Size        0
Uniformity of Cell Shape       0
Marginal Adhesion              0
Single Epithelial Cell Size    0
Bare Nuclei                    0
Bland Chromatin                0
Normal Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

```
data.hist(figsize=(12,12))
plt.show()
```




```
data.corr()
```

	Clump Thickness	Uniformity of Cell Size
Clump Thickness	1.000000	0.642481
Uniformity of Cell Size	0.642481	1.000000
Uniformity of Cell Shape	0.653470	0.907228
Marginal Adhesion	0.487829	0.706977

Single Epithelial Cell Size	0.523596	0.753544
Bare Nuclei	0.593091	0.691709
Bland Chromatin	0.553742	0.755559
Normal Nucleoli	0.534066	0.719346
Mitoses	0.350957	0.460755
Class	0.714790	0.820801

	Uniformity of Cell Shape	Marginal Adhesion
Clump Thickness	0.653470	0.487829
Uniformity of Cell Size	0.907228	0.706977
Uniformity of Cell Shape	1.000000	0.685948
Marginal Adhesion	0.685948	1.000000
Single Epithelial Cell Size	0.722462	0.594548
Bare Nuclei	0.713878	0.670648
Bland Chromatin	0.735344	0.668567
Normal Nucleoli	0.717963	0.603121
Mitoses	0.441258	0.418898
Class	0.821891	0.706294

	Single Epithelial Cell Size	Bare Nuclei
Clump Thickness	0.523596	0.593091
Uniformity of Cell Size	0.753544	0.691709
Uniformity of Cell Shape	0.722462	0.713878
Marginal Adhesion	0.594548	0.670648
Single Epithelial Cell Size	1.000000	0.585716
Bare Nuclei	0.585716	1.000000
Bland Chromatin	0.618128	0.680615

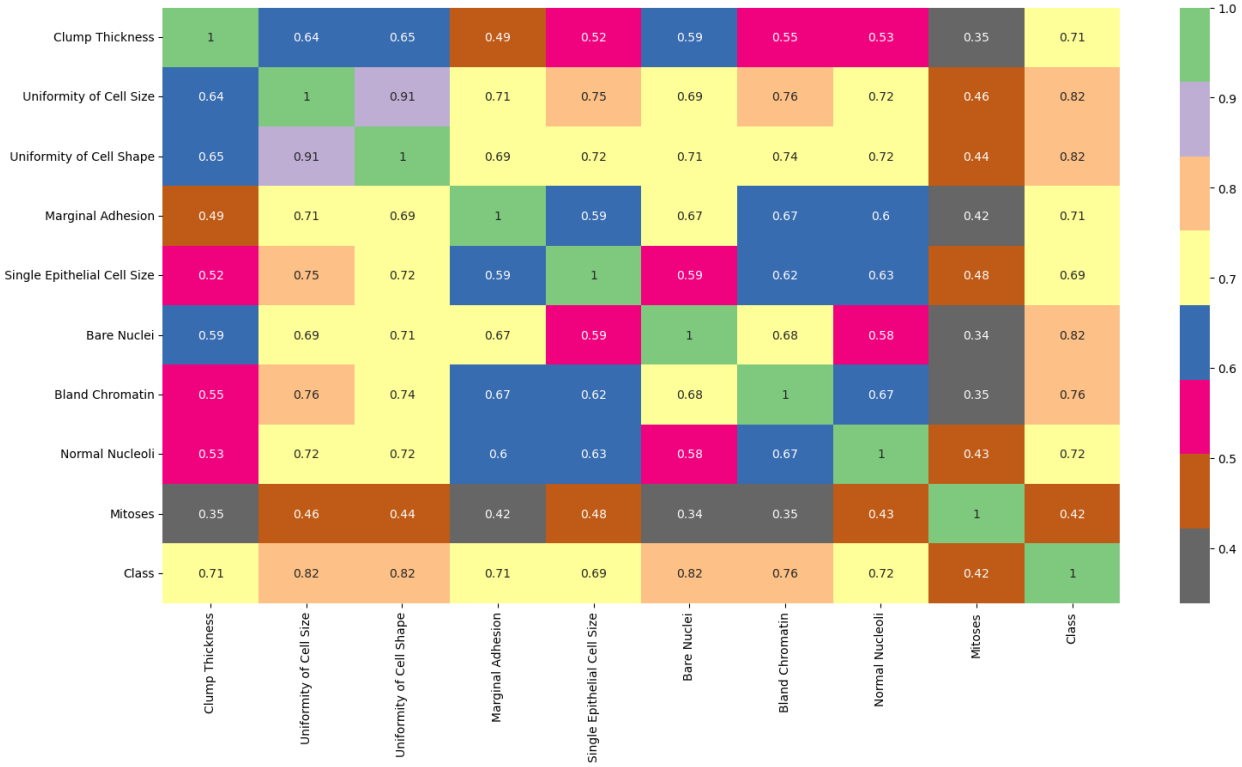
```
Normal Nucleoli                                 0.628926     0.584280

Mitoses                                         0.480583     0.339210

Class                                           0.690958     0.822696
```

```
                             Bland Chromatin  Normal Nucleoli
Mitoses  \
Clump Thickness                     0.553742         0.534066
0.350957
Uniformity of Cell Size             0.755559         0.719346
0.460755
Uniformity of Cell Shape            0.735344         0.717963
0.441258
Marginal Adhesion                   0.668567         0.603121
0.418898
Single Epithelial Cell Size         0.618128         0.628926
0.480583
Bare Nuclei                         0.680615         0.584280
0.339210
Bland Chromatin                     1.000000         0.665602
0.346011
Normal Nucleoli                     0.665602         1.000000
0.433757
Mitoses                             0.346011         0.433757
1.000000
Class                               0.758228         0.718677
0.423448
```

```
                                Class
Clump Thickness              0.714790
Uniformity of Cell Size      0.820801
Uniformity of Cell Shape     0.821891
Marginal Adhesion            0.706294
Single Epithelial Cell Size  0.690958
Bare Nuclei                  0.822696
Bland Chromatin              0.758228
Normal Nucleoli              0.718677
Mitoses                      0.423448
Class                        1.000000
```

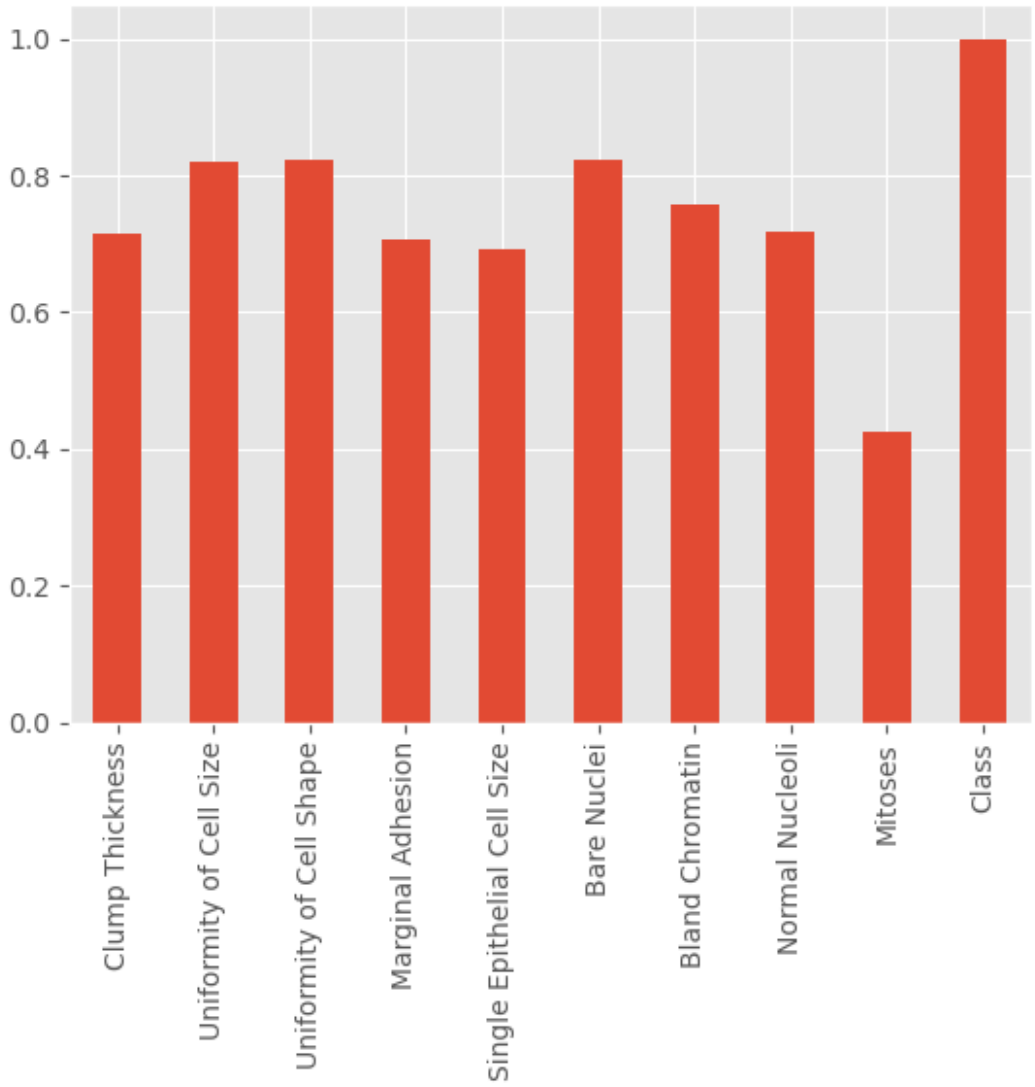
```
plt.figure(figsize=(18,9))
sns.heatmap(data.corr(),annot = True, cmap ="Accent_r")
```

```
<AxesSubplot:>
```



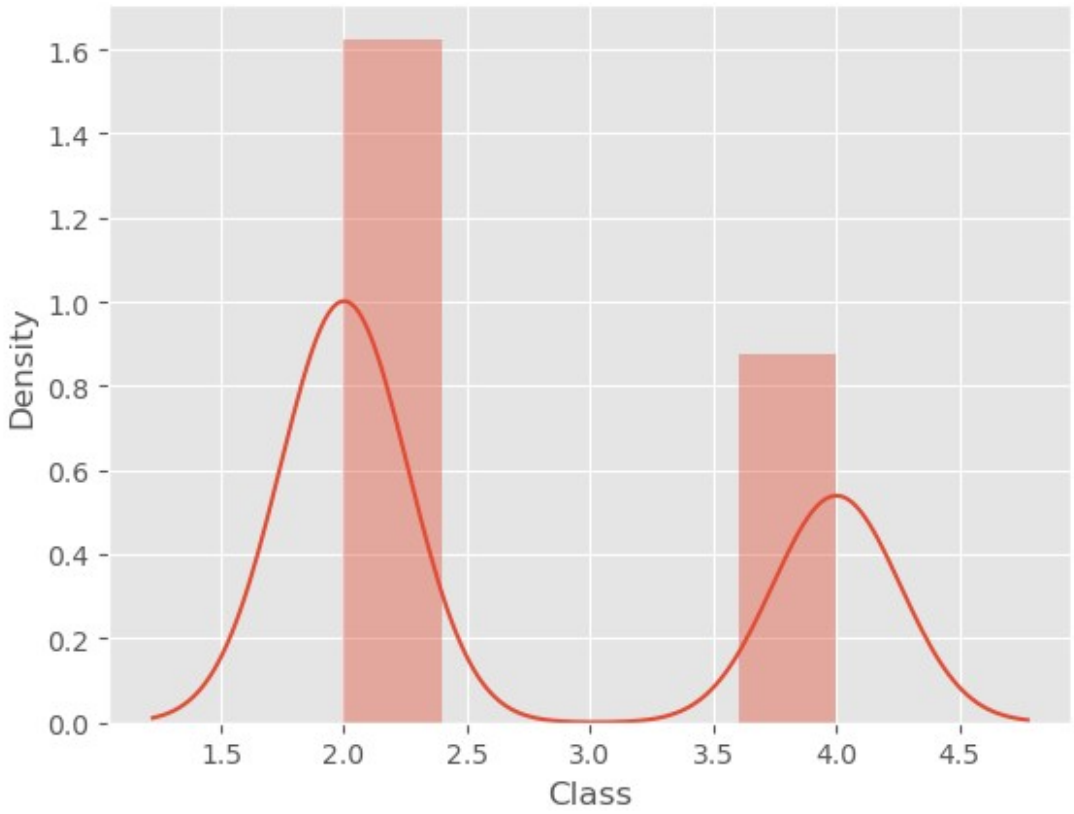

```
data.corr()['Class'].plot(kind='bar')
```

```
<AxesSubplot:>
```



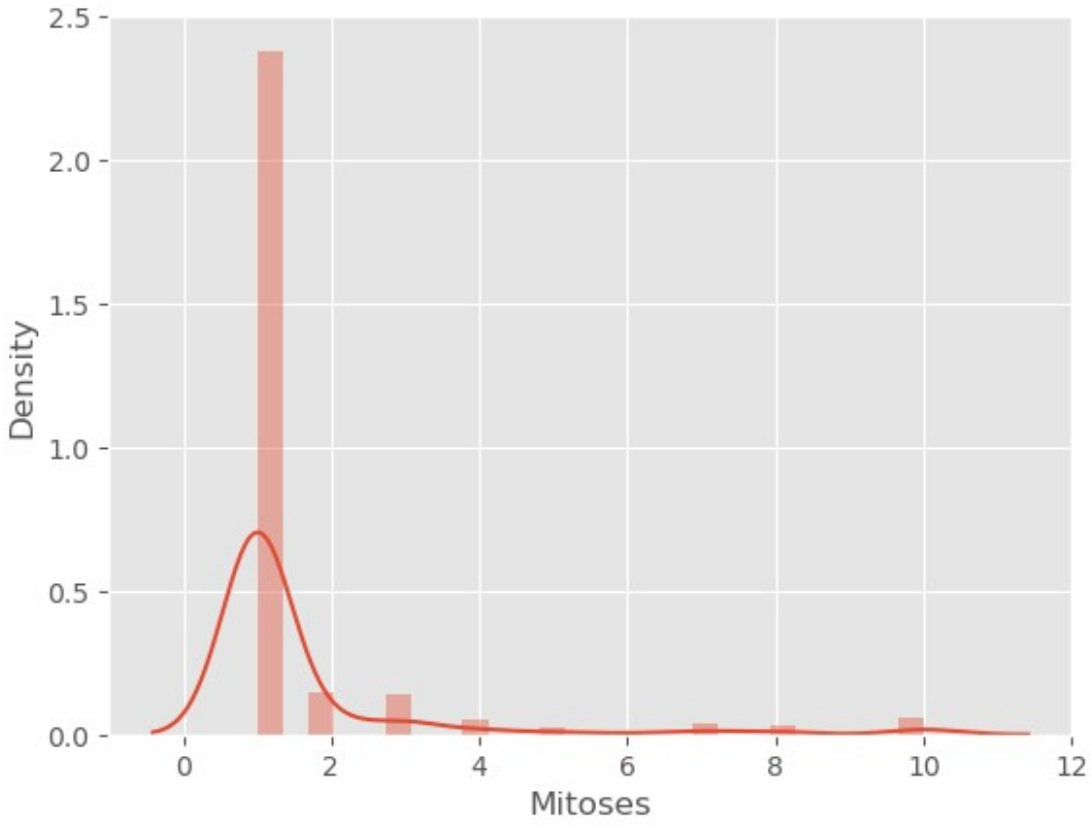

```
sns.distplot(data['Class'])
```

```
<AxesSubplot:xlabel='Class', ylabel='Density'>
```



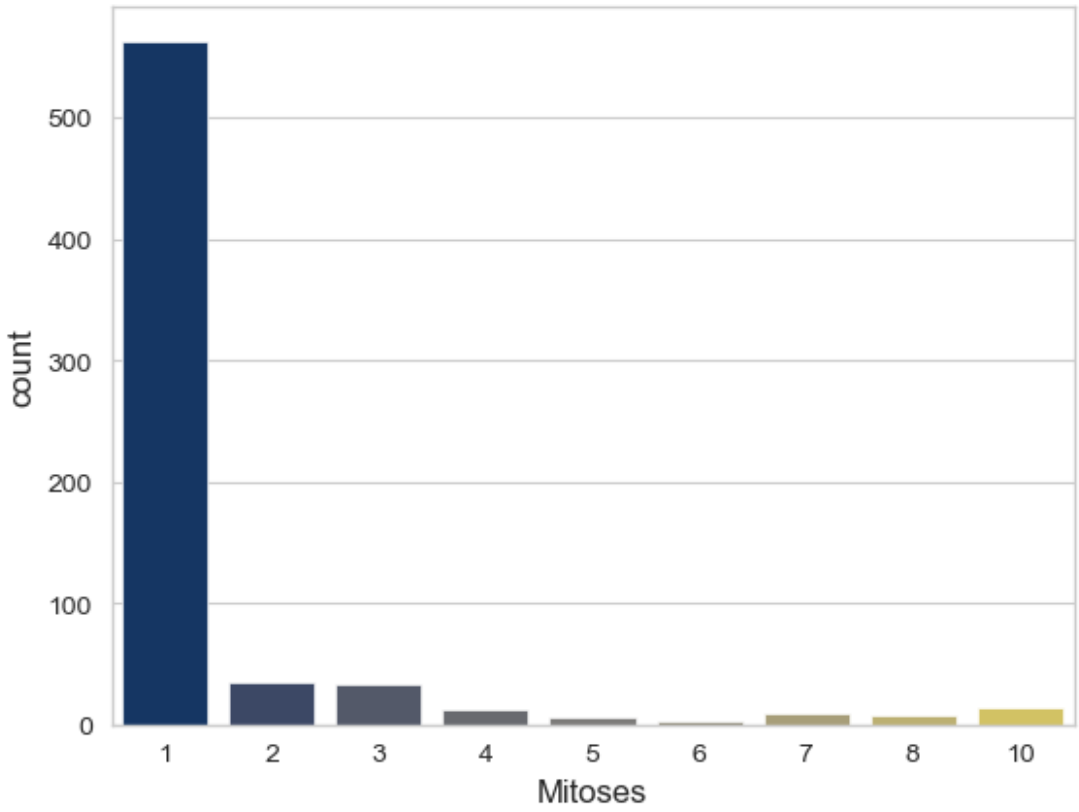

```
sns.distplot(data['Mitoses'])
```

```
<AxesSubplot:xlabel='Mitoses', ylabel='Density'>
```

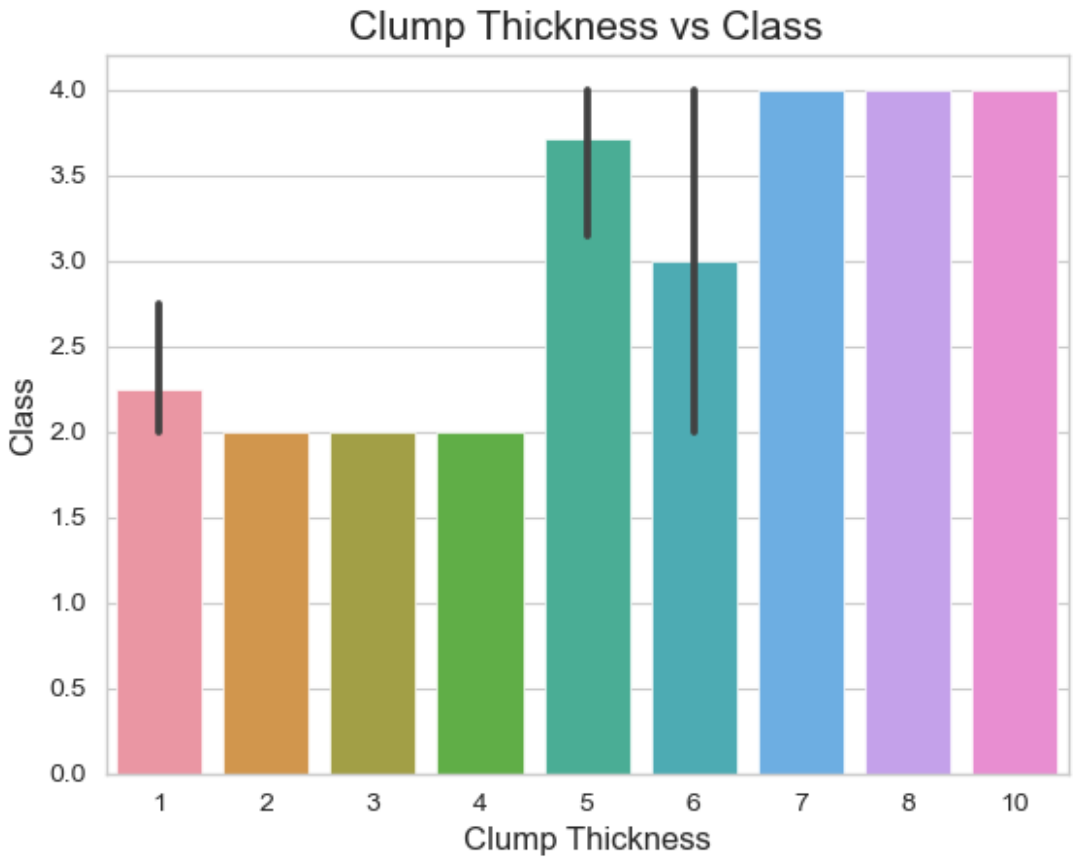



```
sns.set_style('whitegrid')
sns.countplot(x='Mitoses',data=data,palette='cividis')
```

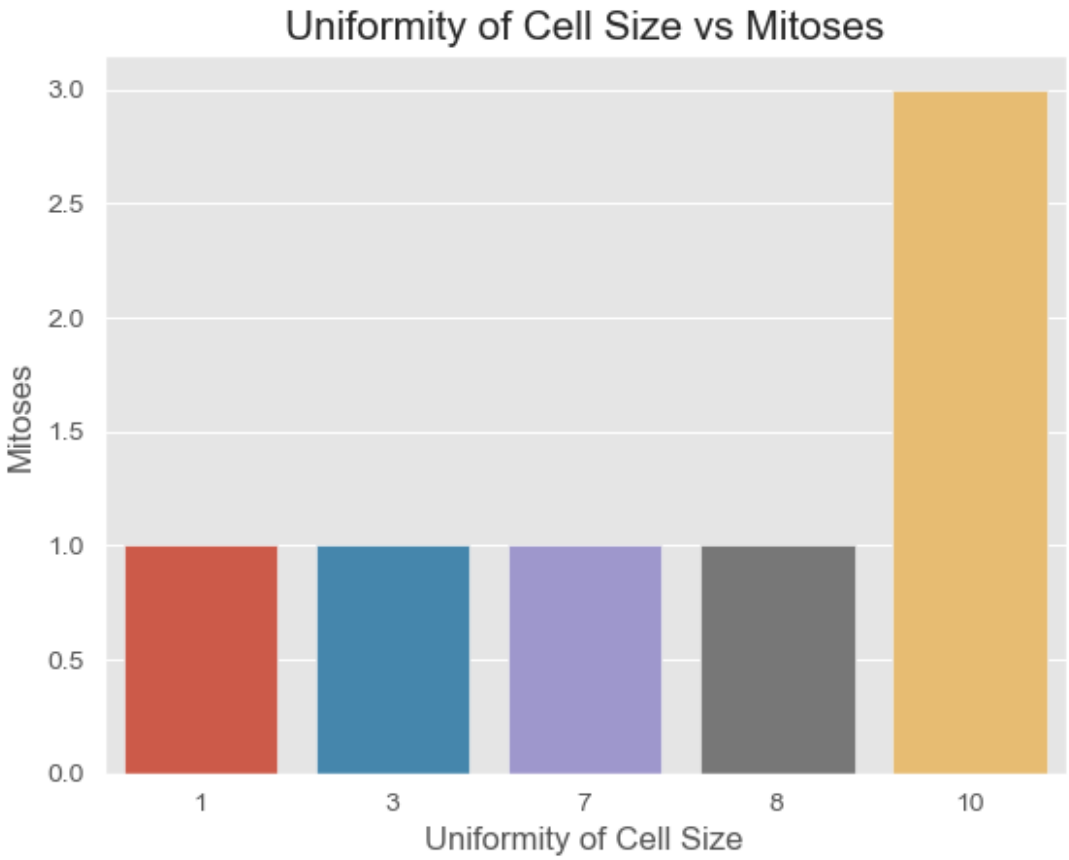
```
<AxesSubplot:xlabel='Mitoses', ylabel='count'>
```




```
sns.barplot(x="Clump Thickness", y="Class",data=data[160:190])
plt.title("Clump Thickness vs Class",fontsize=15)
plt.xlabel("Clump Thickness")
plt.ylabel("Class")
plt.show()
plt.style.use("ggplot")
```

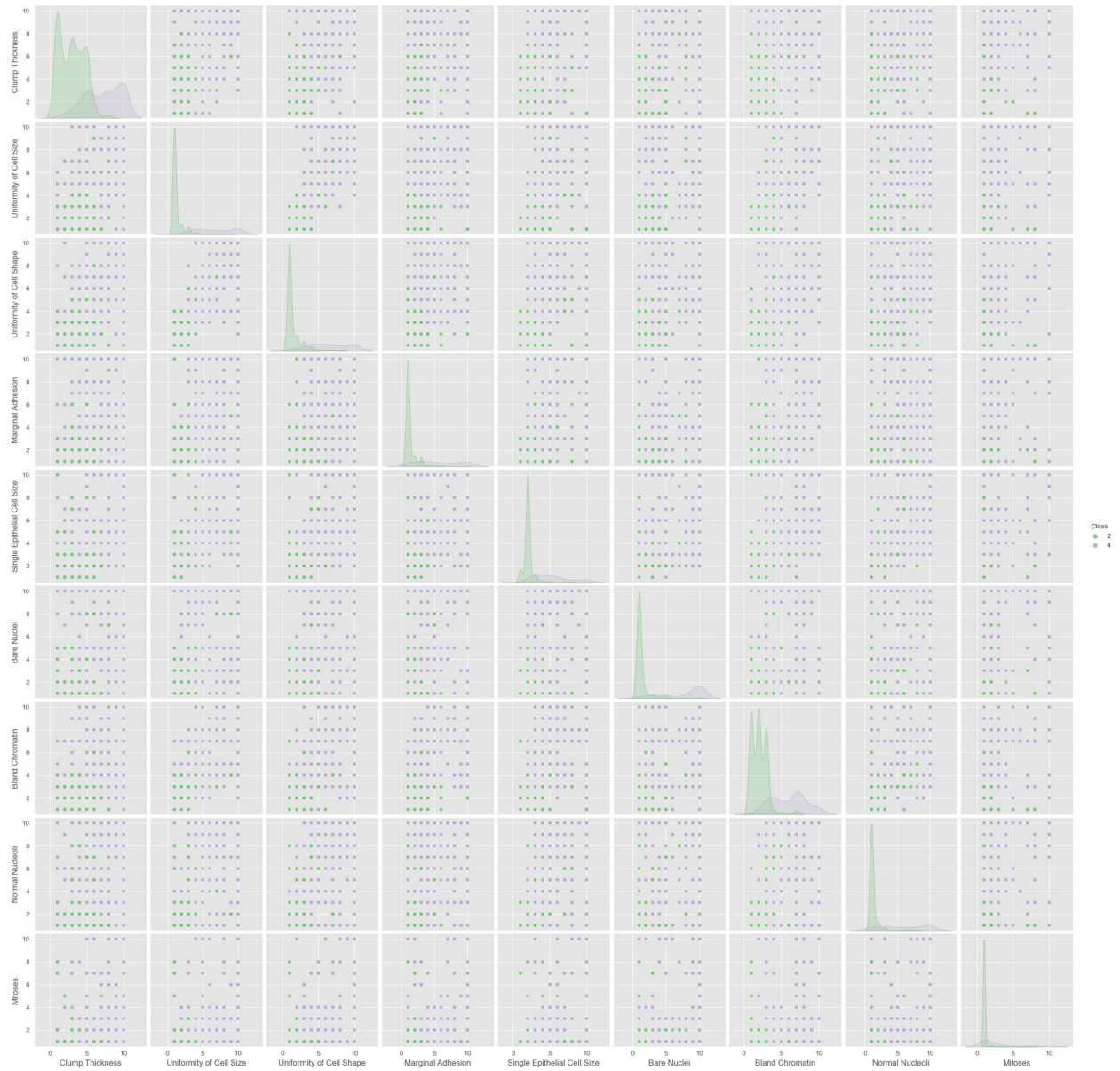



```
sns.barplot(x="Uniformity of Cell Size", y="Mitoses",
data=data[170:180])
plt.title("Uniformity of Cell Size vs Mitoses",fontsize=15)
plt.xlabel("Uniformity of Cell Size")
plt.ylabel("Mitoses")
plt.show()
plt.style.use("ggplot")
```



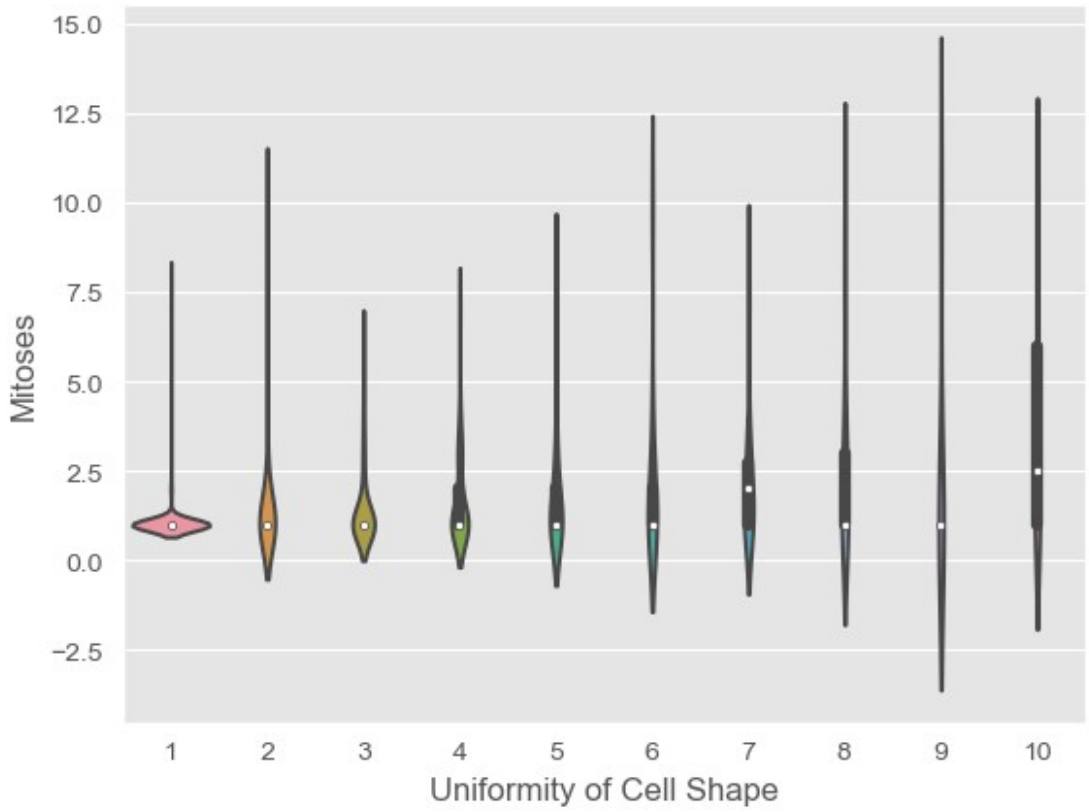

```
mean_col = ['Clump Thickness','Uniformity of Cell Size', 'Uniformity
of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland
Chromatin', 'Normal Nucleoli',
       'Mitoses', 'Class']

sns.pairplot(data[mean_col],hue = 'Class', palette='Accent')
```

```
<seaborn.axisgrid.PairGrid at 0x22041ebde80>
```

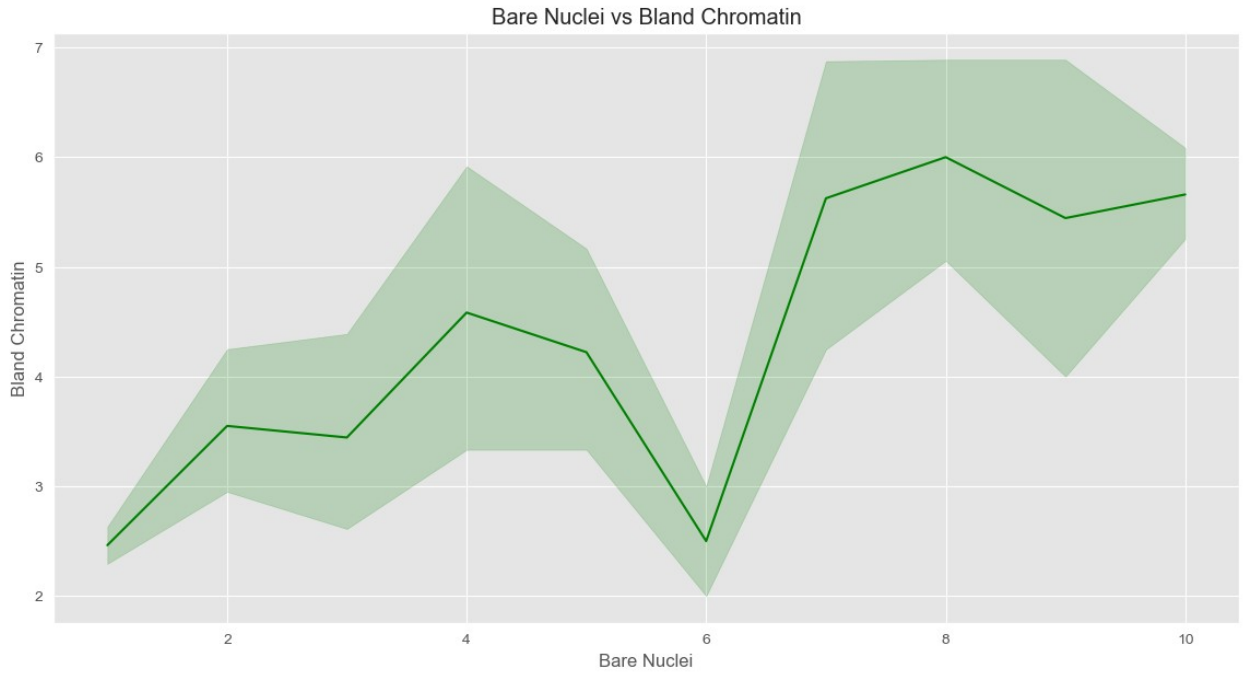



```
sns.violinplot(x="Uniformity of Cell Shape",y="Mitoses",data=data)
```

```
<AxesSubplot:xlabel='Uniformity of Cell Shape', ylabel='Mitoses'>
```




```
plt.figure(figsize=(14,7))
sns.lineplot(x = "Bare Nuclei",y = "Bland Chromatin",data =
data[0:400], color='green')
plt.title("Bare Nuclei vs Bland Chromatin")
plt.xlabel("Bare Nuclei")
plt.ylabel("Bland Chromatin")
plt.show()
```




```
x = data.drop(columns = 'Class')

# Getting Predicting Value
y = data['Class']

#train_test_splitting of the dataset
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
```

MODELS

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(x_train,y_train)

LogisticRegression()

y_pred=reg.predict(x_test)
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,r2_score
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           2       0.97      0.97      0.97        87
           4       0.94      0.94      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137


print(confusion_matrix(y_test,y_pred))
print("Training Score: ",reg.score(x_train,y_train)*100)

[[84  3]
 [ 3 47]]
Training Score:  97.06959706959707

data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data.head()

     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2

print(accuracy_score(y_test,y_pred)*100)

95.62043795620438
```

So we get a accuracy score of 95.620.73 % using Logistic Regression

```
from sklearn.model_selection import GridSearchCV
param = {
         'penalty':['l1','l2'],
         'C':[0.001, 0.01, 0.1, 1, 10, 20,100, 1000]
}
lr= LogisticRegression(penalty='l1')
cv=GridSearchCV(reg,param,cv=5,n_jobs=-1)
cv.fit(x_train,y_train)
cv.predict(x_test)

array([2, 2, 4, 4, 2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2,
2,
       4, 2, 4, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2,
2,
       2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 2, 2,
2,
       2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 4, 2,
4,
```

```
       4, 2, 4, 2, 4, 2, 2, 4, 4, 4, 2, 2, 2, 2, 4, 4, 2, 2, 4, 2, 2,
2,
       4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 4, 2, 4, 2,
2,
       4, 2, 2, 4, 2], dtype=int64)
```

```
print("Best CV score", cv.best_score_*100)
```

```
Best CV score 96.8907422852377
```

2. DECISION TREE CLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=6, random_state=123)

dtree.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=6, random_state=123)
```

```
y_pred1=dtree.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           2       0.96      0.94      0.95        87
           4       0.90      0.94      0.92        50

    accuracy                           0.94       137
   macro avg       0.93      0.94      0.94       137
weighted avg       0.94      0.94      0.94       137
```

```
print(confusion_matrix(y_test,y_pred1))
print("Training Score: ",dtree.score(x_train,y_train)*100)
```

```
[[82  5]
 [ 3 47]]
Training Score:  99.08424908424908
```

```
print(accuracy_score(y_test,y_pred1)*100)
```

```
94.16058394160584
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred1})
data.head()
```

```
     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2
```

So we get a accuracy score of 94.160 % using Decision Tree Classifier

3. Random Forest Classifier

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

```
RandomForestClassifier()
```

```
y_pred2=rfc.predict(x_test)
```

```
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           2       0.99      0.97      0.98        87
           4       0.94      0.98      0.96        50

    accuracy                           0.97       137
   macro avg       0.97      0.97      0.97       137
weighted avg       0.97      0.97      0.97       137
```

```
print(confusion_matrix(y_test,y_pred2))
print("Training Score: ",rfc.score(x_train,y_train)*100)
```

```
[[84  3]
 [ 1 49]]
Training Score:  100.0
```

```
print(accuracy_score(y_test,y_pred2)*100)
```

```
97.08029197080292
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred2})
data.head()
```

```
     Actual  Predicted
113       2          2
```

```
378        2           2
303        4           4
504        4           4
301        2           2
```

So we get a accuracy score of 97.080 % using Random Forest Classifier

4. KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=7)

knn.fit(x_train,y_train)

KNeighborsClassifier(n_neighbors=7)

y_pred3=knn.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

           2       0.99      0.97      0.98        87
           4       0.94      0.98      0.96        50

    accuracy                           0.97       137
   macro avg       0.97      0.97      0.97       137
weighted avg       0.97      0.97      0.97       137
```

```
print(confusion_matrix(y_test,y_pred3))

[[84  3]
 [ 1 49]]

print("Training Score: ",knn.score(x_train,y_train)*100)

Training Score:  97.8021978021978

print(knn.score(x_test,y_test))

0.9708029197080292

print(accuracy_score(y_test,y_pred3)*100)

97.08029197080292

data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred3})
data.head()
```

```
     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2
```

So we get a accuracy score of 97.080 % using KNeighborsClassifier

5. SVC

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
```

```
SVC()
```

```
y_pred4=svc.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

           2       0.99      0.95      0.97        87
           4       0.92      0.98      0.95        50

    accuracy                           0.96       137
   macro avg       0.96      0.97      0.96       137
weighted avg       0.96      0.96      0.96       137
```

```
print(confusion_matrix(y_test,y_pred4))
```

```
[[83  4]
 [ 1 49]]
```

```
print("Training Score: ",svc.score(x_train,y_train)*100)
print(svc.score(x_test,y_test))
```

```
Training Score:  97.8021978021978
0.9635036496350365
```

```
print(accuracy_score(y_test,y_pred4)*100)
```

```
96.35036496350365
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred4})
data.head()
```

```
     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2
```

So we get a accuracy score of 97.802 % using SVC

6. AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier(base_estimator = None)
adb.fit(x_train,y_train)
```

```
AdaBoostClassifier()
```

```
y_pred5=adb.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred5))
```

```
              precision    recall  f1-score   support

           2       0.98      0.95      0.97        87
           4       0.92      0.96      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.96      0.95       137
weighted avg       0.96      0.96      0.96       137
```

```
print(confusion_matrix(y_test,y_pred5))
```

```
[[83  4]
 [ 2 48]]
```

```
print("Training Score: ",adb.score(x_train,y_train)*100)
```

```
Training Score:  99.63369963369964
```

```
print(accuracy_score(y_test,y_pred5)*100)
```

```
95.62043795620438
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred5})
data.head()
```

	Actual	Predicted
113	2	2
378	2	2
303	4	4
504	4	4
301	2	2

So we get a accuracy score of 95.620 % using AdaBoostClassifier

7. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier()
gbc.fit(x_train,y_train)
```

```
GradientBoostingClassifier()
```

```
y_pred6=gbc.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred6))
```

```
              precision    recall  f1-score   support

           2       0.97      0.97      0.97        87
           4       0.94      0.94      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137
```

```
print(confusion_matrix(y_test,y_pred6))
```

```
[[84  3]
 [ 3 47]]
```

```
print("Training Score: ",gbc.score(x_train,y_train)*100)
```

```
Training Score:  100.0
```

```
print(gbc.score(x_test,y_test))
```

```
0.9562043795620438
```

```
print(accuracy_score(y_test,y_pred6)*100)
```

```
95.62043795620438
```

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred6})
data.head()
```

```
     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2
```

So we get a accuracy score of 95.620 % using GradientBoostingClassifier

8. XGBClassifier

```
from xgboost import XGBClassifier

xgb =XGBClassifier(objective ='reg:linear', colsample_bytree = 0.3,
learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)

xgb.fit(x_train, y_train)
```

```
XGBClassifier(alpha=10, colsample_bytree=0.3, max_depth=5,
n_estimators=10,
              objective='reg:linear')
```

```
y_pred7=xgb.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred7))
```

```
              precision    recall  f1-score   support

           2       0.99      0.95      0.97        87
           4       0.92      0.98      0.95        50

    accuracy                           0.96       137
   macro avg       0.96      0.97      0.96       137
weighted avg       0.96      0.96      0.96       137
```

```
print(confusion_matrix(y_test,y_pred7))
```

```
[[83  4]
 [ 1 49]]
```

```
print("Training Score: ",xgb.score(x_train,y_train)*100)
print(xgb.score(x_test,y_test))
```

```
Training Score:  97.98534798534799
0.9635036496350365
```

```
print(accuracy_score(y_test,y_pred7)*100)
```

```
96.35036496350365
```

So we get a accuracy score of 96.350 % using XGBClassifier

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred7})
data.head()
```

```
     Actual  Predicted
113       2          2
378       2          2
303       4          4
504       4          4
301       2          2
```

9. Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

```
GaussianNB()
```

```
y_pred8=gnb.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,mean_squared_err
or,r2_score
print(classification_report(y_test,y_pred8))
```

```
              precision    recall  f1-score   support

           2       1.00      0.92      0.96        87
           4       0.88      1.00      0.93        50

    accuracy                           0.95       137
   macro avg       0.94      0.96      0.95       137
weighted avg       0.96      0.95      0.95       137
```

```
print(confusion_matrix(y_test,y_pred8))
```

```
[[80  7]
 [ 0 50]]

print(accuracy_score(y_test,y_pred8)*100)

94.8905109489051

print("Training Score: ",gnb.score(x_train,y_train)*100)
print(gnb.score(x_test,y_test))

Training Score:  96.52014652014653
0.948905109489051
```

So we get a accuracy score of 94.890 % using Naive Bayes

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred8})
data.head()
```

	Actual	Predicted
113	2	2
378	2	2
303	4	4
504	4	4
301	2	2

So now we conclude the accuracy of different models:

```
data=pd.DataFrame({'Models':
['LOGREG','DT','RF','KNN','SVC','ADABOOST','GRADIENT','XGB','NAIVE'],
                   'Accuracy':
[accuracy_score(y_test,y_pred)*100,accuracy_score(y_test,y_pred1)*100,

accuracy_score(y_test,y_pred2)*100,accuracy_score(y_test,y_pred3)*100,

accuracy_score(y_test,y_pred4)*100,accuracy_score(y_test,y_pred5)*100,

accuracy_score(y_test,y_pred6)*100,accuracy_score(y_test,y_pred7)*100,
                            accuracy_score(y_test,y_pred8)*100]})
data
```

	Models	Accuracy
0	LOGREG	95.620438
1	DT	94.160584
2	RF	97.080292
3	KNN	97.080292
4	SVC	96.350365
5	ADABOOST	95.620438
6	GRADIENT	95.620438
7	XGB	96.350365
8	NAIVE	94.890511

Random Forest and KNN got the highest accuracy

```
sns.barplot(data['Models'],data['Accuracy'])
plt.xticks(rotation=90)
plt.show()
```