```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv('/content/Yash Markad - breast-cancer-wisconsin.data',header=None)
df
```

	0	1	2	3	4	5	6	7	8	9	10
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2	1	1	1	2
695	841769	2	1	1	1	2	1	1	1	1	2
696	888820	5	10	10	3	7	3	8	10	2	4
697	897471	4	8	6	4	3	4	10	6	1	4
698	897471	4	8	8	5	4	5	10	4	1	4

699 rows × 11 columns

```
df.columns = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Ce
df.head()
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli
0	1000025	5	1	1	1	2	1	3	1
1	1002945	5	4	4	5	7	10	3	2
2	1015425	3	1	1	1	2	2	3	1
3	1016277	6	8	8	1	3	4	3	7
4	1017023	4	1	1	3	2	1	3	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
```

```
 0   Sample code number           699 non-null    int64
 1   Clump Thickness              699 non-null    int64
 2   Uniformity of Cell Size      699 non-null    int64
 3   Uniformity of Cell Shape     699 non-null    int64
 4   Marginal Adhesion            699 non-null    int64
 5   Single Epithelial Cell Size  699 non-null    int64
 6   Bare Nuclei                  699 non-null    object
 7   Bland Chromatin              699 non-null    int64
 8   Normal Nucleoli              699 non-null    int64
 9   Mitoses                      699 non-null    int64
 10  Class                        699 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.2+ KB
```

```
df['Bare Nuclei'] = df['Bare Nuclei'].replace('?', np.nan)
df['Bare Nuclei'] = pd.to_numeric(df['Bare Nuclei'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Sample code number           699 non-null    int64
 1   Clump Thickness              699 non-null    int64
 2   Uniformity of Cell Size      699 non-null    int64
 3   Uniformity of Cell Shape     699 non-null    int64
 4   Marginal Adhesion            699 non-null    int64
 5   Single Epithelial Cell Size  699 non-null    int64
 6   Bare Nuclei                  683 non-null    float64
 7   Bland Chromatin              699 non-null    int64
 8   Normal Nucleoli              699 non-null    int64
 9   Mitoses                      699 non-null    int64
 10  Class                        699 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 60.2 KB
```

```
df.isnull().sum()
```

	0
Sample code number	0
Clump Thickness	0
Uniformity of Cell Size	0
Uniformity of Cell Shape	0
Marginal Adhesion	0
Single Epithelial Cell Size	0
Bare Nuclei	16
Bland Chromatin	0
Normal Nucleoli	0
Mitoses	0
Class	0

dtype: int64

```
da=df.dropna(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 683 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Sample code number           683 non-null    int64
 1   Clump Thickness              683 non-null    int64
 2   Uniformity of Cell Size      683 non-null    int64
 3   Uniformity of Cell Shape     683 non-null    int64
 4   Marginal Adhesion            683 non-null    int64
 5   Single Epithelial Cell Size  683 non-null    int64
 6   Bare Nuclei                  683 non-null    float64
 7   Bland Chromatin              683 non-null    int64
 8   Normal Nucleoli              683 non-null    int64
 9   Mitoses                      683 non-null    int64
 10  Class                        683 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 64.0 KB
```

```
correlation=df.corr()
correlation
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin
Sample code number	1.000000	-0.056350	-0.041396	-0.042221	-0.069630	-0.048644	-0.099248	-0.061966
Clump Thickness	-0.056350	1.000000	0.642481	0.653470	0.487829	0.523596	0.593091	0.553742
Uniformity of Cell Size	-0.041396	0.642481	1.000000	0.907228	0.706977	0.753544	0.691709	0.755559
Uniformity of Cell Shape	-0.042221	0.653470	0.907228	1.000000	0.685948	0.722462	0.713878	0.735344
Marginal Adhesion	-0.069630	0.487829	0.706977	0.685948	1.000000	0.594548	0.670648	0.668567
Single Epithelial Cell Size	-0.048644	0.523596	0.753544	0.722462	0.594548	1.000000	0.585716	0.618128
Bare Nuclei	-0.099248	0.593091	0.691709	0.713878	0.670648	0.585716	1.000000	0.680615
Bland Chromatin	-0.061966	0.553742	0.755559	0.735344	0.668567	0.618128	0.680615	1.000000
Normal Nucleoli	-0.050699	0.534066	0.719346	0.717963	0.603121	0.628926	0.584280	0.665602
Mitoses	-0.037972	0.350957	0.460755	0.441258	0.418898	0.480583	0.339210	0.346011
Class	-0.084701	0.714790	0.820801	0.821891	0.706294	0.690958	0.822696	0.758228

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
```

```
x=df.drop(['Class'],axis=1)
y=df['Class']
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x)
x=scaler.transform(x)
```

```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

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

```
model1=LogisticRegression()
```

```
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'penalty': ['l1', 'l2']  # Penalty type
}
```

```
grid_search = GridSearchCV(estimator=model1, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
best_estimator_: LogisticRegression
LogisticRegression
```

```
print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_*100)
```

```
Best parameters: {'C': 0.1, 'penalty': 'l2'}
Best score: 96.70391993327773
```

```
test_accuracy = grid_search.score(X_test, y_test)
print("Test accuracy:", test_accuracy*100)
```

```
Test accuracy: 95.62043795620438
```

```
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

y_pred=grid_search.predict(X_test)

precision = precision_score(y_test, y_pred,pos_label=2)
recall = recall_score(y_test, y_pred,pos_label=2)
f1 = f1_score(y_test, y_pred,pos_label=2)
roc_auc = roc_auc_score(y_test, y_pred)

print("Precision:", precision*100)
print("Recall:", recall*100)
print("F1-Score:", f1*100)
print("ROC AUC:", roc_auc*100)
```

```
Precision: 90.69767441860465
Recall: 98.73417721518987
F1-Score: 94.54545454545455
ROC AUC: 92.470536883457
```

```
model2=LogisticRegression()
param_grid = {
    'C':[0.1] ,'penalty':['l2']
}
grid_search2 = GridSearchCV(estimator=model2, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search2.fit(X_train, y_train)
```

```
GridSearchCV
best_estimator_: LogisticRegression
LogisticRegression
```

```
print("train_score:",grid_search2.score(X_train,y_train)*100)
print("test_score:",grid_search2.score(X_test,y_test)*100)
```

```
train_score: 97.06959706959707
test_score: 95.62043795620438
```

```
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

y_pred=grid_search2.predict(X_test)

precision = precision_score(y_test, y_pred,pos_label=2)
recall = recall_score(y_test, y_pred,pos_label=2)
f1 = f1_score(y_test, y_pred,pos_label=2)
roc_auc = roc_auc_score(y_test, y_pred)

print("Precision:", precision*100)
print("Recall:", recall*100)
print("F1-Score:", f1*100)
print("ROC AUC:", roc_auc*100)
```

```
Precision: 93.97590361445783
Recall: 98.73417721518987
F1-Score: 96.29629629629629
ROC AUC: 95.05674378000873
```

lets try with knn model

```
from sklearn.neighbors import KNeighborsClassifier


param_grid = {
    'n_neighbors': list(range(1, 31)),  # Range of k values to try
    'weights': ['uniform', 'distance'],  # Weighting schemes
    'metric': ['euclidean', 'manhattan']  # Distance metrics
}


model3=KNeighborsClassifier()
grid_search3 = GridSearchCV(estimator=model3, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search3.fit(X_train, y_train)
best_params = grid_search3.best_params_
best_score = grid_search3.best_score_
print("Best Parameters:", best_params)
print("Best Accuracy:", best_score*100)
print("test_score",grid_search3.score(X_test,y_test)*100)
```

```
Best Parameters: {'metric': 'euclidean', 'n_neighbors': 4, 'weights': 'uniform'}
Best Accuracy: 96.70391993327773
test_score 93.43065693430657
```

```
y_pred2=grid_search3.predict(X_test)

precision = precision_score(y_test, y_pred,pos_label=2)
recall = recall_score(y_test, y_pred,pos_label=2)
f1 = f1_score(y_test, y_pred,pos_label=2)
roc_auc = roc_auc_score(y_test, y_pred)

print("Precision:", precision*100)
print("Recall:", recall*100)
print("F1-Score:", f1*100)
print("ROC AUC:", roc_auc*100)
```

```
Precision: 93.97590361445783
Recall: 98.73417721518987
F1-Score: 96.29629629629629
ROC AUC: 95.05674378000873
```

```
param_grid1 = {
    'n_neighbors': [4],
    'weights': ['uniform'],
    'metric': ['euclidean']
}
model4=KNeighborsClassifier()
grid_search4 = GridSearchCV(estimator=model4, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search4.fit(X_train, y_train)
print("best acuuracy",grid_search4.score(X_train,y_train)*100)
print("test_score",grid_search4.score(X_test,y_test)*100)
```

```
best acuuracy 97.06959706959707
test_score 93.43065693430657
```

```
y_pred3=grid_search4.predict(X_test)

precision = precision_score(y_test, y_pred,pos_label=2)
recall = recall_score(y_test, y_pred,pos_label=2)
f1 = f1_score(y_test, y_pred,pos_label=2)
roc_auc = roc_auc_score(y_test, y_pred)

print("Precision:", precision*100)
print("Recall:", recall*100)
print("F1-Score:", f1*100)
print("ROC AUC:", roc_auc*100)
```

```
Precision: 93.97590361445783
Recall: 98.73417721518987
F1-Score: 96.29629629629629
ROC AUC: 95.05674378000873
```

Double-click (or enter) to edit

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

# Assuming you have trained multiple models using GridSearchCV and stored
# their results in variables like grid_search1, grid_search2, grid_search3, etc.
models = ['Model2', 'Model4']  # Replace with your model names
accuracies = [grid_search2.score(X_test,y_test), grid_search4.score(X_test,y_test),
```

```
# Create a bar chart
plt.figure(figsize=(8, 6))
plt.bar(models, accuracies, color=['blue', 'green'])

plt.title('Model_testing Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xlabel('Models')
plt.ylim(0, 1)  # Set y-axis limits to 0-1 for accuracy

# Add accuracy values on top of the bars
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.01, f'{acc:.2f}', ha='center', va='bottom')
```