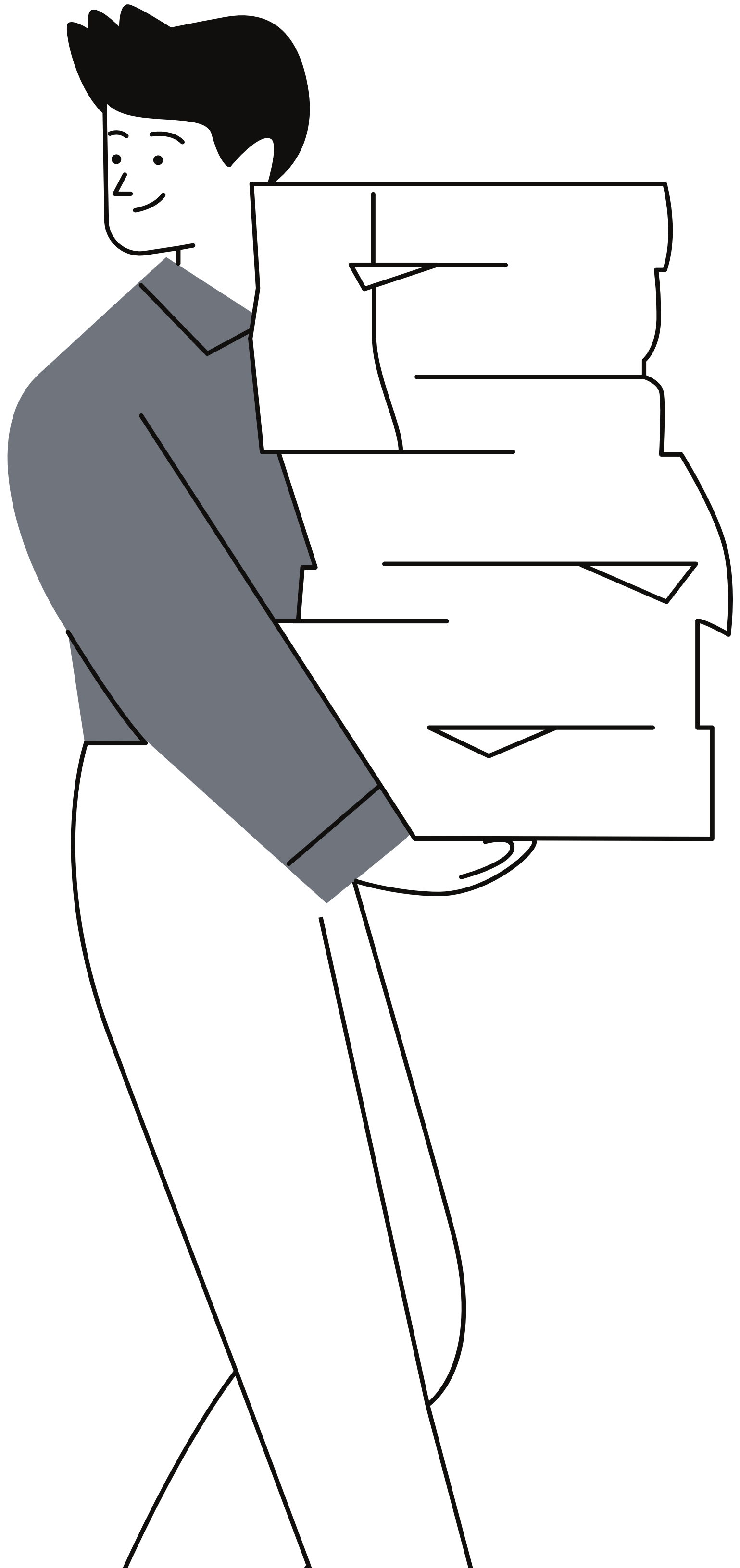
Breast Cancer Prediction

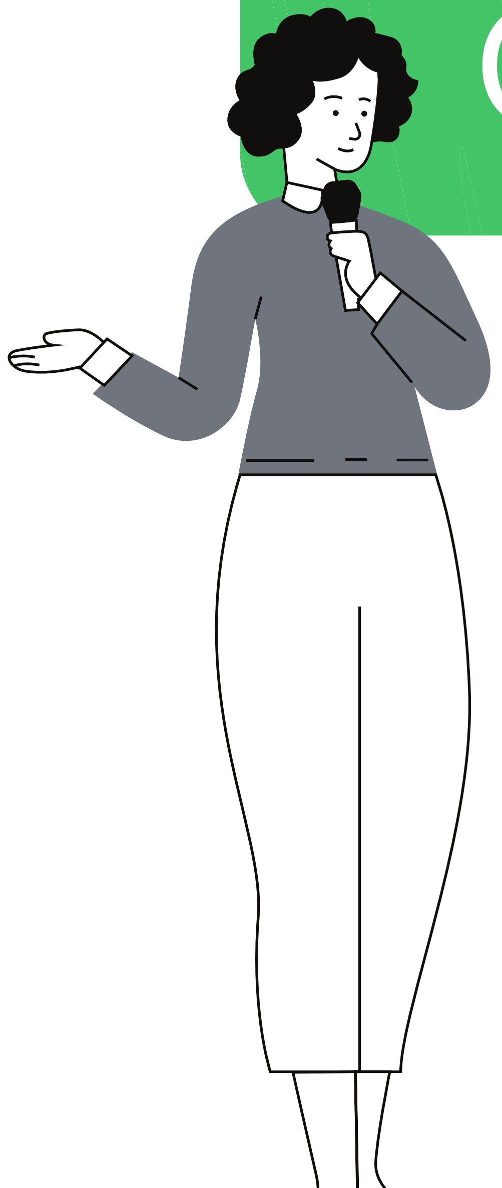
Machine Learning

By Kiruthika



Breast Cancer Detection

Today's Content



Introduction

The **breast cancer dataset** is a classic and very easy binary classification dataset. It contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass and describe characteristics of the cell nuclei present in the image.

Classes	2
Samples per class	212(M),357(B)
Samples total	569
Dimensionality	30
Features	real, positive

```
:Summary Statistics:

===================================== ====== ======
                                       Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
```

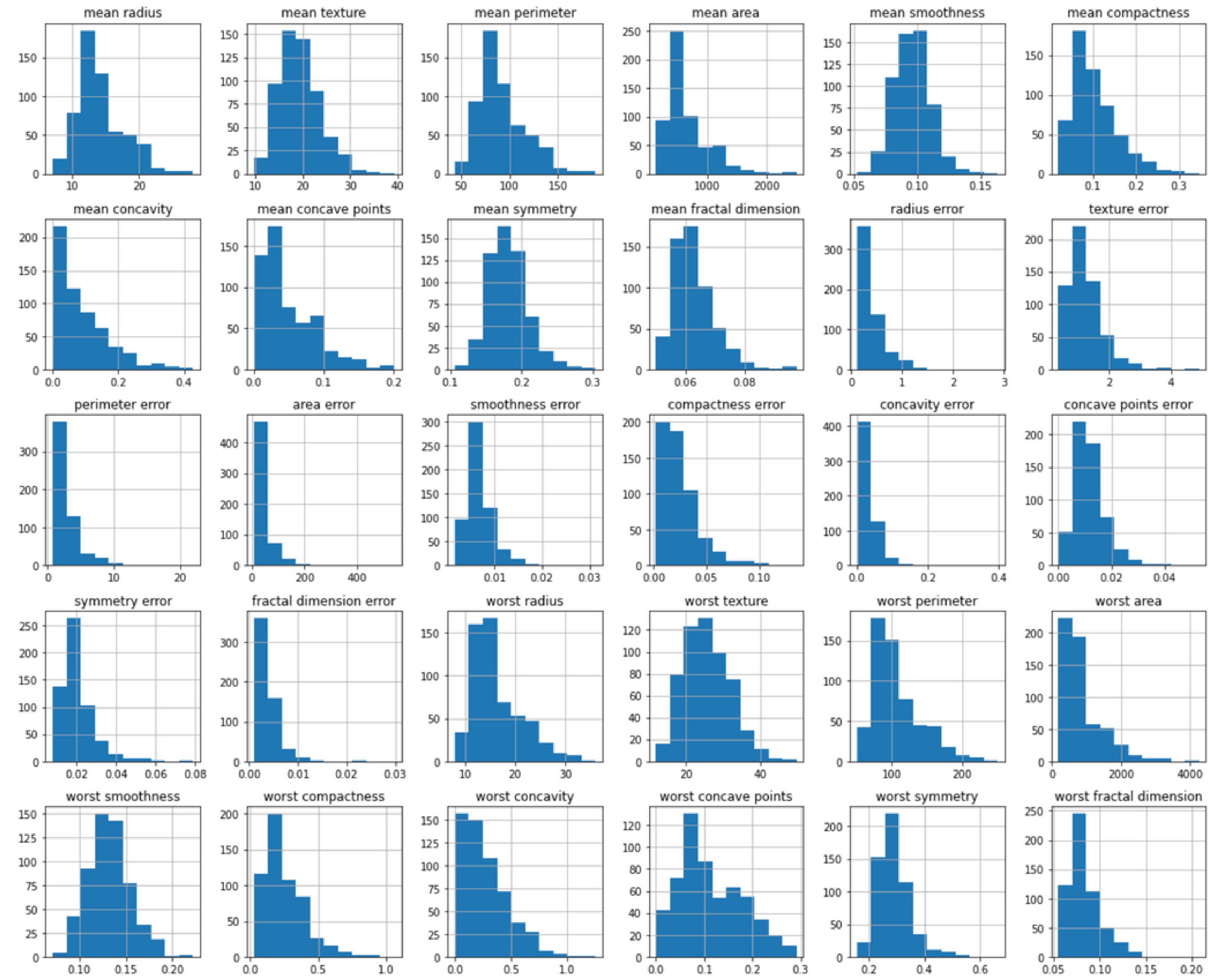
Link to Dataset: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

Visualization

The following is a visualization image of breast cancer attributes from the given dataset.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```



Preparing our Data

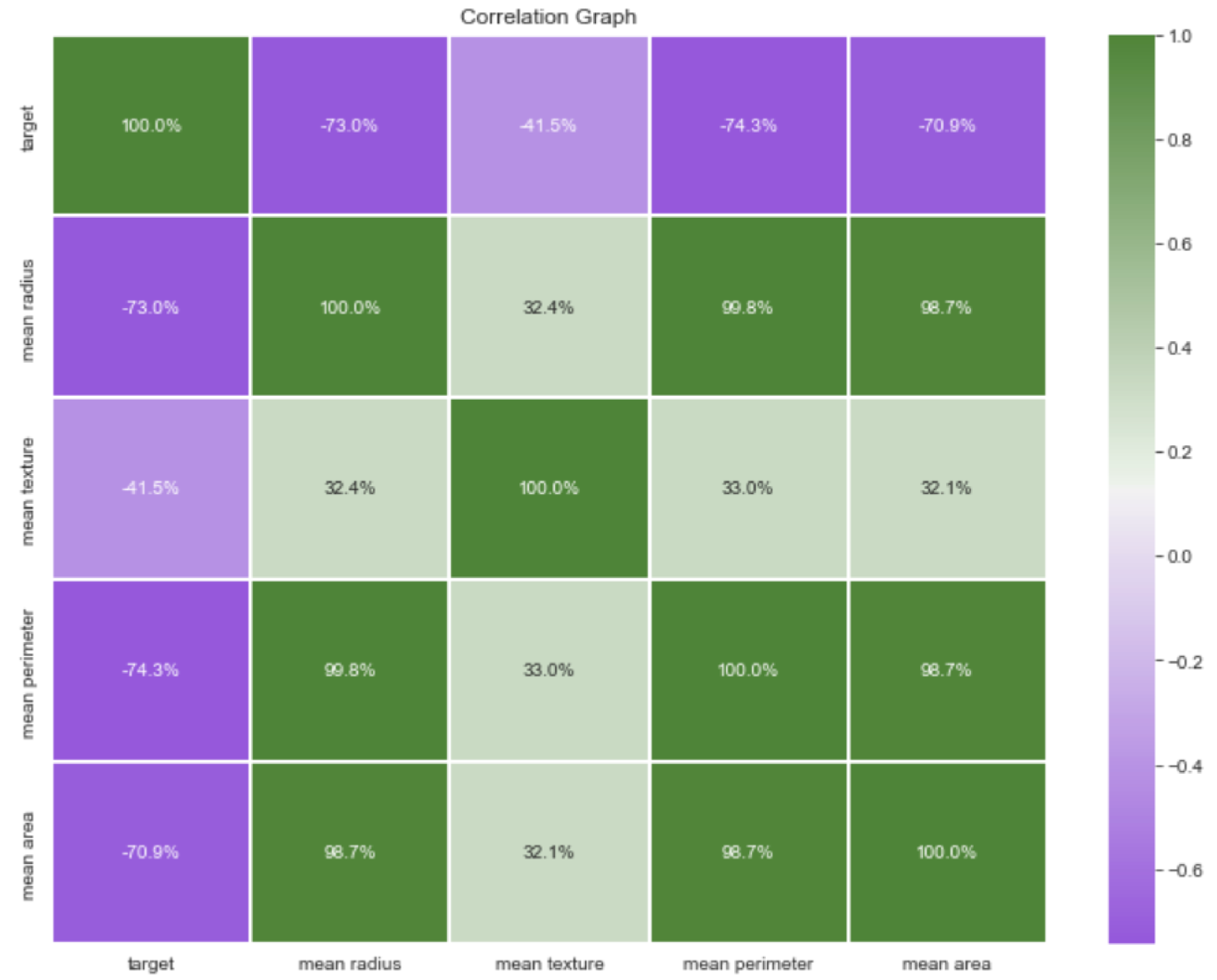
Splitting our dataset

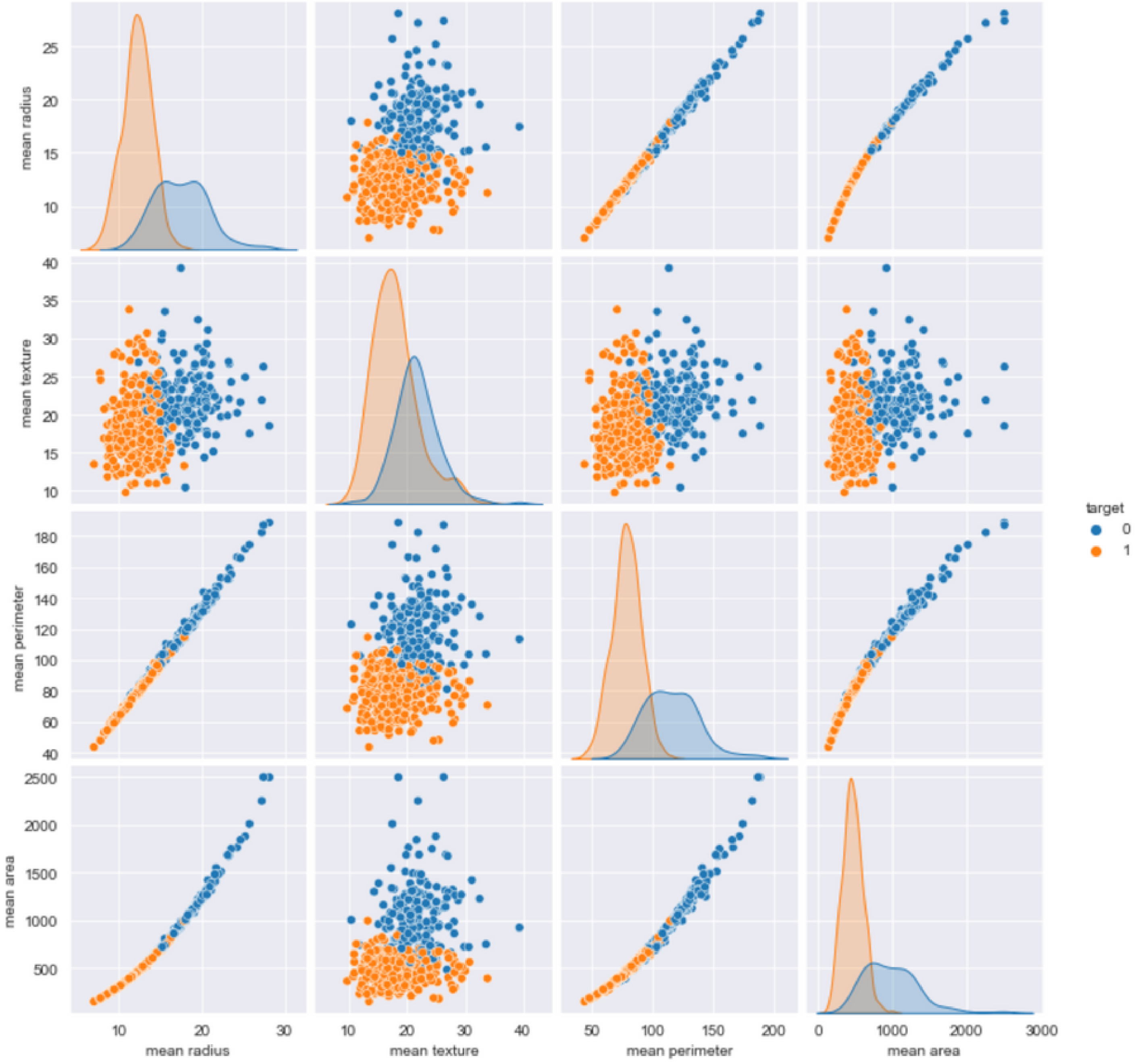
```
X = data.data
# Store the target data
y = data.target
# split the data using Scikit-Learn's train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Visualization





Algorithm

Choosing our Algorithm

In our dataset we have the outcome variable or Dependent variable i.e Y having only two set of values, either M (Malign) or B(Benign). So we will use **Classification algorithm** of **supervised learning**.

- Logistic Regression
- Nearest Neighbor
- Support Vector Machines
- Kernel SVM
- Naïve Bayes
- Decision Tree Algorithm
- Random Forest Classification

Results

Choosing our Algorithm

In our dataset we have the outcome variable or Dependent variable i.e Y having only two set of values, either M (Malign) or B(Benign). So we will use **Classification algorithm** of **supervised learning**.

- Logistic Regression
- Nearest Neighbor
- Support Vector Machines
- Naïve Bayes
- Decision Tree Algorithm
- Random Forest Classification

Results

```
#Using Logistic Regression Algorithm to the Training Set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
predictions = classifier.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.958041958041958
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        53
           1       0.97      0.97      0.97        90

    accuracy                           0.96       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143
```

Logistic Regression

```
#Using KNeighborsClassifier Method o
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 6, metric
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
predictions = classifier.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.951048951048951
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        53
           1       0.95      0.98      0.96        90

    accuracy                           0.95       143
   macro avg       0.95      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```

K Neighbour

Results

```
from sklearn.svm import SVC
classifier_svm = SVC (kernel = 'rbf', random_state = SEED)
classifier_svm.fit (X_train, y_train)
predictions = classifier_svm.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
cm_svm = confusion_matrix (y_test, predictions)
acc_svm = accuracy_score (y_test, predictions)
```

```
0.965034965034965
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        53
           1       0.97      0.98      0.97        90

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.97      0.96       143
```

Support Vector Machine

```
from sklearn.naive_bayes import GaussianNB
classifier_nb = GaussianNB()
classifier_nb.fit (X_train, y_train)
predictions = classifier_nb.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
cm_nb = confusion_matrix (y_test, predictions)
acc_nb = accuracy_score (y_test, predictions)
```

```
0.916083916083916
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        53
           1       0.93      0.93      0.93        90

    accuracy                           0.92       143
   macro avg       0.91      0.91      0.91       143
weighted avg       0.92      0.92      0.92       143
```

Naive Bayes

Results

```
#Using DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
predictions = classifier.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.958041958041958
              precision    recall  f1-score   support

           0       0.93      0.96      0.94        53
           1       0.98      0.96      0.97        90

    accuracy                           0.96       143
   macro avg       0.95      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143
```

Decision Tree

```
#Using RandomForestClassifier method
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criter
classifier.fit(X_train, y_train)
classifier.score(X_test, y_test)
predictions = classifier.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.972027972027972
              precision    recall  f1-score   support

           0       0.95      0.98      0.96        53
           1       0.99      0.97      0.98        90

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143
```

Random Forest

Results

	MODEL	ACCURACY
0	LOGISTIC REGRESSION	0.958042
1	K-NN	0.951049
2	NAIVE BAYES	0.916084
3	SVM	0.965035
4	DECISION TREE	0.881119
5	RANDOM FOREST	0.972028

Final Results

Hyperparameter Tuning

	NAME OF MODEL	ACCURACY SCORE	BEST ACCURACY
0	LOGISTIC REGRESSION	0.958042	0.98361
1	K-NN	0.951049	0.971059
2	NAIVE BAYES	0.916084	-
3	SVM	0.965035	0.982023
4	DECISION TREE	0.881119	0.931847
5	RANDOM FOREST	0.972028	0.826725

Hyperparameter Tuning

```
from sklearn.model_selection import RepeatedStratifiedKFold
model = LogisticRegression()
solvers = ['newton-cg']
max_iter= 1000
penalty = ['l2']
c_values = [1000, 100, 10, 1.0, 0.1, 0.01, 0.001]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=SEED)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
best_accuracy_log = grid_search.best_score_
```

```
Best: 0.981248 using {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
```

Logistic Regression

Hyperparameter Tuning

```
model = KNeighborsClassifier()
n_neighbors = range(1, 21, 2)
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']
# define grid search
grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=SEED)
grid_search = GridSearchCV(estimator=classifier_knn, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy')
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
best_accuracy_knn = grid_search.best_score_
```

```
Best: 0.971059 using {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'uniform'}
```

K Neighours

Hyperparameter Tuning

```
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [1000, 100, 10, 1.0, 0.1, 0.01, 0.001]
gamma = [0.02, 0.01]
# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=SEED)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy')
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
best_accuracy_svm = grid_search.best_score_
```

```
Best: 0.982023 using {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

Support Vector Machines

Hyperparameter Tuning

```
model = DecisionTreeClassifier()
criterion = ['gini', 'entropy', 'log_loss']
max_depth = [4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]
max_leaf_nodes = [2,4,6,10,15,30,40,50,100]
min_samples_split = [2, 3, 4]
# define grid search
cv = random_state=SEED
grid = dict(criterion=criterion, max_depth=max_depth, max_leaf_nodes=max_leaf_nodes)
grid_search = GridSearchCV(estimator=model, param_grid=grid, cv=cv)
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
best_accuracy_dtc = grid_search.best_score_
```

```
Best: 0.928055 using {'criterion': 'entropy', 'max_depth': 6, 'max_leaf_nodes': 10}
```

Decision Tree

Hyperparameter Tuning

```
model = RandomForestClassifier()
n_estimators = [10, 100, 1000]
criterion = ['gini', 'entropy', 'log_loss']
# define grid search
grid = dict(n_estimators=n_estimators, criterion=criterion)
cv = RepeatedStratifiedKFold(random_state=SEED)
grid_search = GridSearchCV(estimator=model, param_grid=grid, scoring='accuracy')
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
best_accuracy_rfc = grid_search.best_score_
```

```
Best: 0.967114 using {'criterion': 'entropy', 'n_estimators': 10}
```

Random Forest

Thank You