#!pip install sklearn

#Import all libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

#Inline matplotlib to view inside this notebook directly %matplotlib inline

#Models

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

#Evaluation

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

from sklearn.metrics import confusion_matrix, classification_report

from sklearn.metrics import precision_score, recall_score, f1_score

#Importing and understanding our dataset

df = pd.read_csv("/content/heart_dis_data.csv")

#Shape of dataset

df.shape

#Print first five rows of the dataset

df.head()

#Print last five rows of the dataset

df.tail()

 # checking the distribution of Target

df["target"].value_counts()

df["target"].value_counts().plot(kind='bar
',color=["salmon","lightblue"])

 # getting some info about the data

df.info()

**# checking for
missing values**

df.isna().sum()

**# statistical
measures about the
data**

df.describe()

**# count of the
gender**

df.sex.value_counts()

pd.crosstab(df.target,
df.sex)

pd.crosstab(df.target,
df.sex).plot(kind="ba
r",figsize=(10,6),colo
r=["salmon","lightblu
e"])

plt.title("Heart
Disease Frerquency
for Sex")

plt.xlabel("0 = No
Disease, 1=Disease")

plt.ylabel("Amount")

```
plt.legend(["Female",
"Male"]);
plt.xticks(rotation=0)
;

plt.figure(figsize=(10
,6))

#Scatter with positive
examples
plt.scatter(df.age[df.t
arget==1],
df.thalach[df.target==
1],
c="salmon")
```

#Scatter with negative examples

```
plt.scatter(df.age[df.t
arget==0],
        df.thalach[df.t
arget==0],
        c="lightblue");
```

#Add some helpful info

```
plt.title("Heart
Disease in function
of Age and Max
Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart
Rate")
plt.legend(["Disease"
,"No Disease"])
```

```
df.age.plot.hist()

pd.crosstab(df.cp,df.target)

pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(10,6),color=["salmon","lightblue"])

plt.title("Heart Disease Frequency Per Chest Pain Type")

plt.xlabel("CHest Pain Type")

plt.ylabel("Amount")

plt.legend(["No Disease","Disease"])

plt.xticks(rotation=0)

df.corr()

corr_matrix = df.corr()

fig,ax = plt.subplots(figsize=(15,10))

ax=sns.heatmap(corr_matrix,annot=True,linewidths=0.5,fmt=".2f",cmap="YlGnBu");

bottom, top = ax.get_ylim()

ax.set_ylim(bottom + 0.5, top

0.5)
```

X=df.drop("target",a
xis=1)

y=df["target"]

np.random.seed(42)

X_train,X_test,y_trai
n,y_test =
train_test_split(X,y,t
est_size=0.2)

models = {"Logistic
Regression":
LogisticRegression(),

"KNN":
KNeighborsClassifier
(),

"Random
Forest":
RandomForestClassif
ier()}

#Create function to fit and score models

def
fit_and_score(models
, X_train, X_test,
y_train, y_test):

""""

#Fits and evaluates given machine learning models.

models : a dict of
differetn Scikit-Learn
machine learning
models

X_train : training
data (no labels)

X_test : testing data (no labels)

y_train : training labels

y_test : test labels

#set random seed

np.random.seed(42)

#dictionary to keep model scores

model_scores = {}

#loop thru models

for name, model in models.items():

#fit model

model.fit(X_train, y_train

#evaluate model and append score

model_scores[name]=model.score(X_test, y_test)

return model_scores

model_scores = fit_and_score(models=models, X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)

model_scores

```
model_scores = fit_and_score(models=models, X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
model_scores

model_compare = pd.DataFrame(model_scores, index=["accuracy"])
model_compare.T.plot.bar()
```

#Tune knn

```
train_scores = []
test_scores = []
```

#list for different values of n-neighbors

```
neighbors = range(1,21)
```

#set up knn instance

```
knn = KNeighborsClassifier()
```

#loop thru list

```
for i in neighbors:
 knn.set_params(n_neighbors=i)
```

#fit the model

```python
  knn.fit(X_train,
y_train)
```

 # Update the training scores list

```python
    train_scores.append(knn.score(X_train,
y_train))
```

 # Update the test scores list

```python
  test_scores.append(
knn.score(X_test,
y_test))

test_scores


plt.plot(neighbors, train_scores, label="Train score")
plt.plot(neighbors, test_scores, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_scores)*100:.2f}%")


log_reg_grid = {"C": np.logspace(-4,4,20),
                "solver": ["liblinear"]}

#Create a hyperparameter grid for RF
rf_grid = {"n_estimators": np.arange(10,1000,50),
           "max_depth": [None,3,5,10],
           "min_samples_split": np.arange(2,20,2),
           "min_samples_leaf": np.arange(1,20,2)}
```

 # Tune LogisticRegression

```
np.random.seed(42)
```

 # Setup random hyperparameter search for LogisticRegression

```
rs_log_reg = RandomizedSearchCV(LogisticRegression(),
 param_distributions=log_reg_grid,
   cv=5,
   n_iter=20,
   verbose=True)
```

 # Fit random hyperparameter search model for LogisticRegression

```
 rs_log_reg.fit(X_train, y_train)


rs_log_reg.best_params_

rs_lr_score = rs_log_reg.score(X_test,y_test)
```

#Tune RF

#Set random parameter search for RF

```
np.random.seed(42)
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
 param_distributions=rf_grid,
 cv=5,
 n_iter=20,
 verbose=True)
```

#Fit random hyperparameter search model for RF

```
rs_rf.fit(X_train,y_train)

rs_rf.best_params_

rs_rf_score =rs_rf.score(X_test, y_test)

model_scores
```

#Different hyperparameters for LR model

```
log_reg_grid = {"C": np.logspace(-4,4,30),
               "solver": ["liblinear"]}
```

#Setup grid hyperparameter for LR

```
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid=log_reg_grid,
                          cv=5,
                          verbose=True)
```

#Fit into model

```
gs_log_reg.fit(X_train,y_train)

gs_log_reg.best_params_

gs_lr_score = gs_log_reg.score(X_test,y_test)

model_scores.update( [('RandomizedS LR',rs_lr_score),('RandomizedS
RF',rs_rf_score),('GridS LR',gs_lr_score)] )
model_compare2 = pd.DataFrame(model_scores, index=["accuracy"])
model_compare2.T.plot.bar(legend=False)

y_preds = gs_log_reg.predict(X_test)

y_preds

def plot_roc_curve(classifier, X_test, y_test):
  """Plots the ROC curve for a given classifier.

 Args:
 classifier: The classifier to plot the ROC curve for.
  X_test: The test data.
  y_test: The test labels.
  """

  y_pred = classifier.predict(X_test)
  fpr, tpr, thresholds = roc_curve(y_test, y_pred)
  plt.plot(fpr, tpr)

  print(confusion_matrix(y_test, y_preds))

  sns.set(font_scale=1.5)

  def plot_conf_mat(y_test,y_preds):
    """
    Plot a nice looking confusion matrix using Seaborn's heatmap
```

```
    """
    fig,ax = plt.subplots(figsize=(3,3))
    ax=sns.heatmap(confusion_matrix(y_test,y_preds),
                   annot=True,
                   cbar=False)
    plt.xlabel("Predicted label")
    plt.ylabel("True label")

    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)

plot_conf_mat(y_test, y_preds)

print(classification_report(y_test,y_preds))
```

#Check best hyperparameters

```
gs_log_reg.best_params_
```

#Create a new classifier with best parameters

```
clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")
```

#Cross validated accuracy

```
cv_acc= cross_val_score(clf,
X, y,cv=5, scoring="accuracy")
cv_acc

cv_acc = np.mean(cv_acc)
cv_acc
```

#Cross-validated precision

```
cv_precision= cross_val_score(clf,
 X,y,cv=5, scoring="precision")
cv_precision = np.mean(cv_precision)
cv_precision
```

#Cross-validated recall

```
cv_recall= cross_val_score(clf,
                         X,
                         y,
```

```
                    cv=5,
                    scoring="recall")
cv_recall = np.mean(cv_recall)
cv_recall
```

#Cross-validated f1

```
cv_f1= cross_val_score(clf,
                    X,
                    y,
                    cv=5,
                    scoring="f1")
cv_f1 = np.mean (cv_f1)
cv_f1
```

#Visualise cross-validated metrics

```
cv_metrics = pd.DataFrame({"Accuracy": cv_acc,
                      "Precision": cv_precision,
                      "Recall": cv_recall,
                      "F1": cv_f1},
                      index=[0])

cv_metrics.T.plot.bar(title="Cross-validated classification metrics",
                   legend=False);
```

#Fit an instance of LR

```
clf = LogisticRegression(C=0.20433597178569418,
                     solver="liblinear")

clf.fit(X_train,y_train);

clf.coef_
```

#Match coef to columns

```
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

#Visualise feature importance

```
feature_df = pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot.bar(title="Feature Importance",legend=False)
```