APPENDIX (CODE)

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
data = pd.read_csv("/content/archive/healthcare-dataset-stroke-data.csv")
df = pd.DataFrame(data, index=None)
rows = len(df.axes[0])
cols = len(df.axes[1])
print("Number of Instance: ", rows)
print("Number of Attributes: ", cols)
colCon=[]
for i in range(cols-1):
  if(data[df.columns[i]].dtype == 'object'):
    colCon.append(df.columns[i])
print("Categorical Attributes : ",colCon)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
data[colCon] = data[colCon].apply(label_encoder.fit_transform)
print("Converted Dataset")
print(data)
df = pd.DataFrame(data, index=None)
df.fillna(df.mean(), inplace=True)
X = data.iloc[:, :cols-1]Y=data.iloc[:,cols-1:]

from sklearn.feature_selection import RFECV
from sklearn.ensemble import RandomForestClassifier
estimator = RandomForestClassifier()
selector = RFECV(estimator, step=1, cv=5)
selector = selector.fit(X, Y)
X_new = selector.transform(X)
print('Selected features Count = ',selector.n_features_)
print('Selected Features = ',selector.get_support(indices=True))
print(X.columns[selector.get_support()].to_list())
clsname=df.columns[cols-1]
clsuni=data[clsname].unique()
print(clsname)
print(clsuni)
print(clsuni.size)
print(data[df.columns[1]].dtype)
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_sm, Y_sm = sm.fit_resample(X_new, Y)
Y_sm.value_counts()
X_sm.shape
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier, HistGradientBoostingClassifier,
GradientBoostingClassifier, RandomForestClassifier, ExtraTreesClassifier,
AdaBoostClassifier
from imblearn.over_sampling import SMOTE
vc = VotingClassifier(estimators=[

    ('abc', AdaBoostClassifier()),
    ('ets', ExtraTreesClassifier()),
    ('rf', RandomForestClassifier()),
    ('dt', DecisionTreeClassifier()),
    ('gbc', GradientBoostingClassifier()),
    ('hgb', HistGradientBoostingClassifier()),
    ('knn', KNeighborsClassifier()),
    ('svc', SVC(probability=True)),
    ('nb', GaussianNB()),
    ('lr', LogisticRegression())
], voting='soft')
vc.fit(X_sm, Y_sm)
```

```
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
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, shuffle=False)
X_train, X_test, Y_train,Y_test = train_test_split(X_sm, Y_sm, random_state=0, test_size =
0.2)
#kmeans = KMeans(n_clusters=clsuni.size)
kmeans = Birch(n_clusters=clsuni.size)
kmeans.fit(X_train)
yhat = kmeans.predict(X_test)
print(yhat)
```

```
import sklearn.metrics as metrics
score = metrics.accuracy_score(Y_test,yhat)
print(score)
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
gbc = GradientBoostingClassifier(n_estimators=300,
                                 learning_rate=0.5,
                                 random_state=100,
                                max_features=5 )

gbc.fit(X_train, Y_train)

pred_y_gbc = gbc.predict(X_test)

boost_acc = metrics.accuracy_score(Y_test, pred_y_gbc)
print("GradientBoosting = ",boost_acc)
```

```
import matplotlib as mpl
from matplotlib import pyplot as plt
import seaborn as sns
boosting_classification_models = ['Boosting_DT-1', 'Boosting_KNN',  'Boosting_RF-1',
'Boosting_DT-2', 'Boosting_RF-2']
boosting_classifiers = []
boosting_classifiers.append(AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=10,
learning_rate=0.02))
boosting_classifiers.append(AdaBoostClassifier(GradientBoostingClassifier(),
n_estimators=30, learning_rate=0.02))
boosting_classifiers.append(AdaBoostClassifier(RandomForestClassifier(), n_estimators=50,
learning_rate=0.02))
boosting_classifiers.append(AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=50,
learning_rate=0.03))
boosting_classifiers.append(AdaBoostClassifier(RandomForestClassifier(), n_estimators=50,
learning_rate=0.04))

boosting_train_accuracies = []
boosting_test_accuracies = []

for boosting_classifier in boosting_classifiers:
    boosting_pipeline = Pipeline(steps = [
              ('boosting_classifier',boosting_classifier)
           ])

    print(f'--------------START OF THE {boosting_classifier} MODEL--------------- \n')

    print(f'Training the {boosting_classifier} model')
    boosting_model = boosting_pipeline.fit(X_train, Y_train)
    print('\nTraining Info:')

    boosting_train_predictions = boosting_model.predict(X_train)
    boosting_train_accuracy = metrics.accuracy_score(Y_train, boosting_train_predictions)
    boosting_train_accuracies.append(boosting_train_accuracy)
    print(f'{boosting_classifier} model training accuracy: {boosting_train_accuracy}')

    print(f'\nTesting the {boosting_classifier} model')

    print('\nTesting Info:')

    boosting_test_predictions = boosting_model.predict(X_test)
    boosting_test_accuracy = metrics.accuracy_score(Y_test, boosting_test_predictions)
    boosting_test_accuracies.append(boosting_test_accuracy)
    print(f'{boosting_classifier} model testing accuracy: {boosting_test_accuracy}')

    print(f'\n---------------END OF THE {boosting_classifier} MODEL---------------- \n\n')

print('Boosting Train Accuracy = ',boosting_train_accuracies)
print('Boosting Test Accuracy = ',boosting_test_accuracies)

boosting_model_accuracy_compare = pd.DataFrame({'Boosting Algorithm' :
boosting_classification_models, 'Boosting Training Accuracy' : boosting_train_accuracies,
'Boosting Testing Accuracy' : boosting_test_accuracies})
boosting_model_accuracy_compare.sort_values(by='Boosting Testing Accuracy',
ascending=False)

from sklearn.ensemble import BaggingClassifier
bc = DecisionTreeClassifier()
iter = 50

bag = BaggingClassifier(base_estimator=bc, n_estimators=iter)

bag.fit(X_train, Y_train)

pred_y_bag = bag.predict(X_test)

bag_acc = metrics.accuracy_score(Y_test, pred_y_bag)
print("Bagging Accuracy = ", bag_acc)
bagging_classification_models = ['Bag_DT-1', 'Bag_KNN', 'Bag_RF-1', 'Bag_DT-2', 'Bag_RF-
2']

bagging_classifiers = []
bagging_classifiers.append(BaggingClassifier(DecisionTreeClassifier(), n_estimators=25))
bagging_classifiers.append(BaggingClassifier(KNeighborsClassifier(), n_estimators=50))
bagging_classifiers.append(BaggingClassifier(RandomForestClassifier(), n_estimators=100))
bagging_classifiers.append(BaggingClassifier(DecisionTreeClassifier(), n_estimators=100))
bagging_classifiers.append(BaggingClassifier(RandomForestClassifier(), n_estimators=100))

bagging_train_accuracies = []
bagging_test_accuracies = []

for bagging_classifier in bagging_classifiers:
    bagging_pipeline = Pipeline(steps = [
              ('bagging_classifier',bagging_classifier)
           ])

    print(f'--------------START OF THE {bagging_classifier} MODEL--------------- \n')

    print(f'Training the {bagging_classifier} model')
    bagging_model = bagging_pipeline.fit(X_train, Y_train)

    print('\nTraining Info:')

    bagging_train_predictions = bagging_model.predict(X_train)
    bagging_train_accuracy = metrics.accuracy_score(Y_train, bagging_train_predictions)
    bagging_train_accuracies.append(bagging_train_accuracy)
    print(f'{bagging_classifier} model training accuracy: {bagging_train_accuracy}')

    print(f'\nTesting the {bagging_classifier} model')

    print('\nTesting Info:')

    bagging_test_predictions = bagging_model.predict(X_test)
    bagging_test_accuracy = metrics.accuracy_score(Y_test, bagging_test_predictions)
    bagging_test_accuracies.append(bagging_test_accuracy)
    print(f'{bagging_classifier} model testing accuracy: {bagging_test_accuracy}')

    print(f'\n--------------END OF THE {bagging_classifier} MODEL--------------- \n\n')

print('Bagging Train Accuracy = ',bagging_train_accuracies)
print('Bagging Test Accuracy = ',bagging_test_accuracies)

bagging_model_accuracy_compare = pd.DataFrame({'Bagging Algorithm' :
bagging_classification_models, 'Bagging Training Accuracy' : bagging_train_accuracies,
'Bagging Testing Accuracy' : bagging_test_accuracies})
bagging_model_accuracy_compare.sort_values(by='Bagging Testing Accuracy', ascending=False)

br1 = np.arange(len(bagging_train_accuracies))
br2 = [x + barWidth for x in br1]
```

```
plt.bar(br1, bagging_train_accuracies, color ='r', width = barWidth,
        edgecolor ='grey', label ='Bagging Training Accuracy')
plt.bar(br2, bagging_test_accuracies, color ='g', width = barWidth,
        edgecolor ='grey', label ='Bagging Testing Accuracy')

plt.xticks([r + barWidth for r in range(len(bagging_train_accuracies))],
        ['Bag_DT-1', 'Bag_KNN', 'Bag_RF-1', 'Bag_DT-2', 'Bag_RF-2'])

plt.legend()
plt.show()
```

```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators = [('gbc', GradientBoostingClassifier()),('bag',
BaggingClassifier())],voting='soft')
vc.fit(X_train, Y_train)
pred_y_vc = vc.predict(X_test)
vc_acc = metrics.accuracy_score(Y_test, pred_y_vc)
print('VC Accuracy = ',vc_acc)
data = {'Boosting': boost_acc, 'Bagging':bag_acc, 'Voting':vc_acc}
alg = list(data.keys())
values = list(data.values())
fig = plt.figure(figsize = (8, 4))
plt.bar(alg, values, color ='maroon',
        width = 0.25)
plt.xlabel("Algorithm")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison")
plt.show()
```