In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import os
import warnings
warnings.filterwarnings('ignore')
```

In [13]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics  import accuracy_score
from sklearn.metrics  import classification_report
```

Heart Disease Prediction:

Linear Regression

In [92]:

```
df = pd.read_csv('heart_disease_dataset.csv')
df.head()
```

Out[92]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
num         303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

In [7]:

```
#assign x, y

y = df['num']
```

```
X = df.drop(['num'], axis=1)
```

In [8]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

In [9]:

```
#fit the model

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)
```

Out[9]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [10]:

```
#predict the value

y_pred  = model.predict(X_test)
```

In [11]:

```
print('train score:', model.score(X_train, y_train))

print('test score:', model.score(X_test, y_test))
```

```
train score: 0.4390887691858453
test score: 0.41019101789942886
```

In [ ]:

```

```

Logistic Regression:

In [17]:

```
df.head()
```

Out[17]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

In [18]:

```
#assign x, y

y = df['num']
X = df.drop(['num'], axis=1)
```

In [19]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

In [21]:

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

Out[21]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [23]:

```
#predict model
y_pred = classifier.predict(X_test)
```

In [25]:

```
cm = confusion_matrix(y_test,y_pred)
print('confusion matrix:', cm)
```

```
confusion matrix: [[41  8]
 [ 9 33]]
```

In [27]:

```
print('accuracy', accuracy_score(y_test, y_pred))
```

```
accuracy 0.8131868131868132
```

Decision Tree

using entropy:

In [35]:

```
#split data

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

In [36]:

```
#Feature scaling

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

```
C:\Users\Krishna\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:645:
DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by
StandardScaler.
  return self.partial_fit(X, y)
C:\Users\Krishna\Anaconda3\lib\site-packages\sklearn\base.py:464: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
```

```
  return self.fit(X, **fit_params).transform(X)
```

In [37]:

```
X_test = sc.transform(X_test)
```

```
C:\Users\Krishna\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data
with input dtype int64, float64 were all converted to float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

In [31]:

```
#fitting decision tree classifcation to the training set

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion='entropy', random_state=2)
classifier.fit(X_train, y_train)
```

Out[31]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
            splitter='best')
```

In [32]:

```
print('Accuracy score: ', accuracy_score(y_test, y_pred))
```

```
Accuracy score:  0.8131868131868132
```

Decsion tree using gini:

In [38]:

```
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion='gini', random_state=2)
classifier.fit(X_train, y_train)
```

Out[38]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
            splitter='best')
```

In [39]:

```
print('Accuracy score: ', accuracy_score(y_test, y_pred))
```

```
Accuracy score:  0.8131868131868132
```

Ensemble Random Forest:

In [40]:

```
from sklearn.ensemble import RandomForestClassifier

# create gaussian classifier
```

```
clf= RandomForestClassifier(n_estimators=100)
```

In [41]:

```
# train the model using training sets

clf.fit(X_train, y_train)
#predict the model

y_pred = clf.predict(X_test)
```

In [42]:

```
print('Accuracy score: ', accuracy_score(y_test, y_pred))
```

```
Accuracy score:  0.8791208791208791
```

KNN model

In [43]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Out[43]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [44]:

```
## predict the model
y_pred = classifier.predict(X_test)
```

In [45]:

```
print('Accuracy score: ', accuracy_score(y_test, y_pred))
```

```
Accuracy score:  0.7912087912087912
```

Naive Bayes:

In [46]:

```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(X_train,y_train)
```

Out[46]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [48]:

```
ypred=model.predict(X_test)
```

In [49]:

```
accuracy_score(y_test,y_pred)
```

Out[49]:

```
0.7912087912087912
```

Bagging:

In [53]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn import model_selection
from sklearn.model_selection import KFold, cross_val_score
```

In [55]:

```
kfold = model_selection.KFold(n_splits=10, random_state=21)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=40)
results = model_selection.cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

```
0.8011827956989247
```

In [57]:

```
#fit the model

model = BaggingClassifier()
model.fit(X_train, y_train)
print(); print(model)
```

```
BaggingClassifier(base_estimator=None, bootstrap=True,
         bootstrap_features=False, max_features=1.0, max_samples=1.0,
         n_estimators=10, n_jobs=None, oob_score=False, random_state=None,
         verbose=0, warm_start=False)
```

In [58]:

```
# predict the model

predicted_y = model.predict(X_test)
```

In [60]:

```
print(accuracy_score(y_test, predicted_y))
```

```
0.8461538461538461
```

AdaBoost:

In [62]:

```
from sklearn.ensemble import AdaBoostClassifier

classifier = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200
)
classifier.fit(X_train, y_train)
```

Out[62]:

```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
```

```
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=200, random_state=None)
```

In [63]:

```
ypred = classifier.predict(X_test)
```

In [64]:

```
accuracy_score(y_test,y_pred)
```

Out[64]:

```
0.7912087912087912
```

Gradient Descent:

In [65]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [67]:

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_features=
2, max_depth=2, random_state=0)
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_test, y_test)))
```

```
Learning rate:  0.05
Accuracy score (training): 0.849
Accuracy score (validation): 0.879
Learning rate:  0.075
Accuracy score (training): 0.863
Accuracy score (validation): 0.890
Learning rate:  0.1
Accuracy score (training): 0.863
Accuracy score (validation): 0.890
Learning rate:  0.25
Accuracy score (training): 0.896
Accuracy score (validation): 0.890
Learning rate:  0.5
Accuracy score (training): 0.929
Accuracy score (validation): 0.857
Learning rate:  0.75
Accuracy score (training): 0.943
Accuracy score (validation): 0.835
Learning rate:  1
Accuracy score (training): 0.934
Accuracy score (validation): 0.791
```

In [69]:

```
#selecting the learning rate 0.75 from above


gb_clf2 = GradientBoostingClassifier(n_estimators=20, learning_rate=0.75, max_features=2,
max_depth=2, random_state=0)
gb_clf2.fit(X_train, y_train)
predictions = gb_clf2.predict(X_test)
```

```
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7912087912087912
```

Xg boost classifier:

In [71]:

```
from xgboost import XGBClassifier

classifier = XGBClassifier()
classifier.fit(X_train, y_train)
```

Out[71]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

In [72]:

```
y_pred = classifier.predict(X_test)
```

In [74]:

```
# Model Accuracy, how well the model performs
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8131868131868132
```

Here Ensemble bagging and Random Forest gives high accuracy

USL

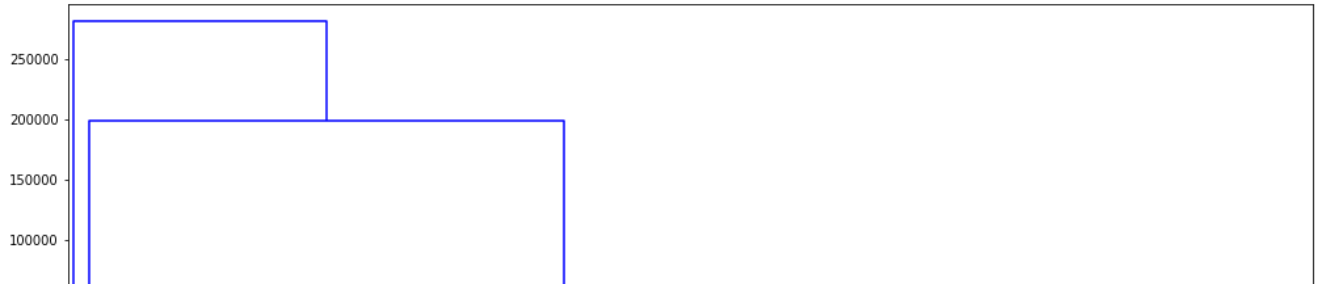
dendogram:

In [75]:

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

In [78]:

```
fig = plt.figure(figsize=(17,5))
z = linkage(df, 'ward')
dendrogram(z)
plt.show()
```



KMeans

In [88]:

```
X = df.drop(['num'], axis=1)
```

In [79]:

```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3) # K Value
kmeans.fit(X)
```

Out[79]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [80]:

```
from sklearn.cluster import KMeans
#kmeans clustering in an iteration 1 till 20 cluster size
cluster_range = range(1,20)
cluster_errors = []

for num_clusters in cluster_range:
    clusters =KMeans(num_clusters)
    clusters.fit(X)
    cluster_errors.append(clusters.inertia_)

cluster_df = pd.DataFrame( {"num_clusters":cluster_range,"cluster_errors":cluster_errors})
cluster_df[0:10]
```

Out[80]:

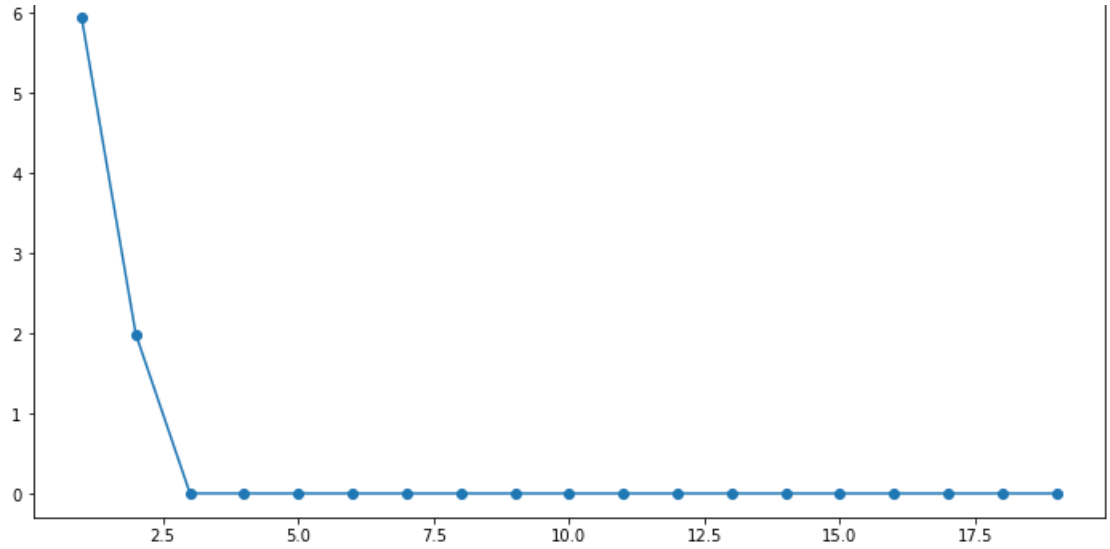
	num_clusters	cluster_errors
0	1	5.934343e+10
1	2	1.986918e+10
2	3	1.081127e+06
3	4	6.038758e+05
4	5	4.783303e+05
5	6	3.977530e+05
6	7	3.360674e+05
7	8	3.133172e+05
8	9	2.722231e+05
9	10	2.602513e+05

In [81]:

```
plt.figure(figsize=(12,6))
plt.plot(cluster_df.num_clusters,cluster_df.cluster_errors,
        marker='o')
```

Out[81]:

```
[<matplotlib.lines.Line2D at 0x261a659bd68>]
```



In [82]:

```
kmeans=KMeans(n_clusters=3)
kmeans.fit(X)
```

Out[82]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [83]:

```
kmeans.labels_
```

Out[83]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
```

In [84]:

```
y=pd.DataFrame(kmeans.labels_ ,columns=['y'])
```

In [89]:

```
df1=pd.concat([X,y],axis=1)
```

In [90]:

```
df1.head()
```

Out[90]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	y
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	y
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	0
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	0
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

PCA:

dimensionality reduction:

In [91]:

```
from sklearn.decomposition import PCA
```

In [100]:

```
#assign x, y

y = df['num']
X = df.drop(['num'], axis=1)
```

In [101]:

```
X_train, X_test,y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

In [102]:

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

```
C:\Users\Krishna\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:645:
DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by
StandardScaler.
  return self.partial_fit(X, y)
C:\Users\Krishna\Anaconda3\lib\site-packages\sklearn\base.py:464: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
C:\Users\Krishna\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DataConversionWarning: Data
with input dtype int64, float64 were all converted to float64 by StandardScaler.
  after removing the cwd from sys.path.
```

In [103]:

```
# pca

pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
```

In [104]:

```
# apply random forest

ranforest= RandomForestClassifier()

#Train the model using the training sets y_pred=clf.predict(X_test)
ranforest.fit(X_train,y_train)
```

Out[104]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
```

```
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [105]:

```
ypred=ranforest.predict(X_test)
```

In [106]:

```
accuracy_score(y_test,ypred)
```

Out[106]:

```
0.4342105263157895
```

After dimensionality reduction the ensemble Random forest gives bad accuracy

Of all the techniques the Supervised learning classification techniques and Unsuperlised learning techinques Bagging and random forest gives good accuracy.

In [ ]:

```

```