Heart_Disease_Prediction

DATASET COLUMNS

1. Age (age in years)
2. Sex (1 = male; 0 = female)
3. CP (chest pain type)
4. TRESTBPS (resting blood pressure (in mm Hg on admission to the hospital))
5. CHOL (serum cholestoral in mg/dl)
6. FPS (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. RESTECH (resting electrocardiographic results)
8. THALACH (maximum heart rate achieved)
9. EXANG (exercise induced angina (1 = yes; 0 = no))
10. OLDPEAK (ST depression induced by exercise relative to rest)
11. SLOPE (the slope of the peak exercise ST segment)
12. CA (number of major vessels (0-3) colored by flourosopy)
13. THAL (3 = normal; 6 = fixed defect; 7 = reversable defect)
14. TARGET (1 or 0)

Import necessary Python modules and Read the data

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
Heart_Disease =pd.read_csv('C:/Users/Lenovo/Desktop/heart_disease.csv')
```

In [3]:

```
Heart_Disease.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Exploratory Data Analysis (EDA)

In [4]:

```
Heart_Disease.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	14
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	2
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	7
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	13
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	15
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	16
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	20.

In [5]:

```
Heart_Disease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [6]:

```
Heart_Disease.columns
```

Out[6]:

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [7]:

```
Heart_Disease.shape
```

Out[7]:

```
(303, 14)
```

In [8]:

```
Heart_Disease.isnull().sum()
```

Out[8]:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [9]:

```
Heart_Disease.isnull().values.any()
```

Out[9]:

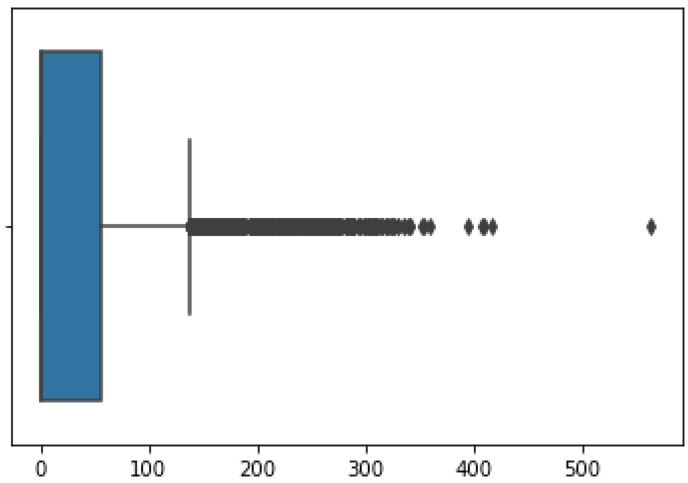
```
False
```

In [10]:

```
#checking for Outlier's
sns.boxplot(x=Heart_Disease)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d3c4042d08>
```



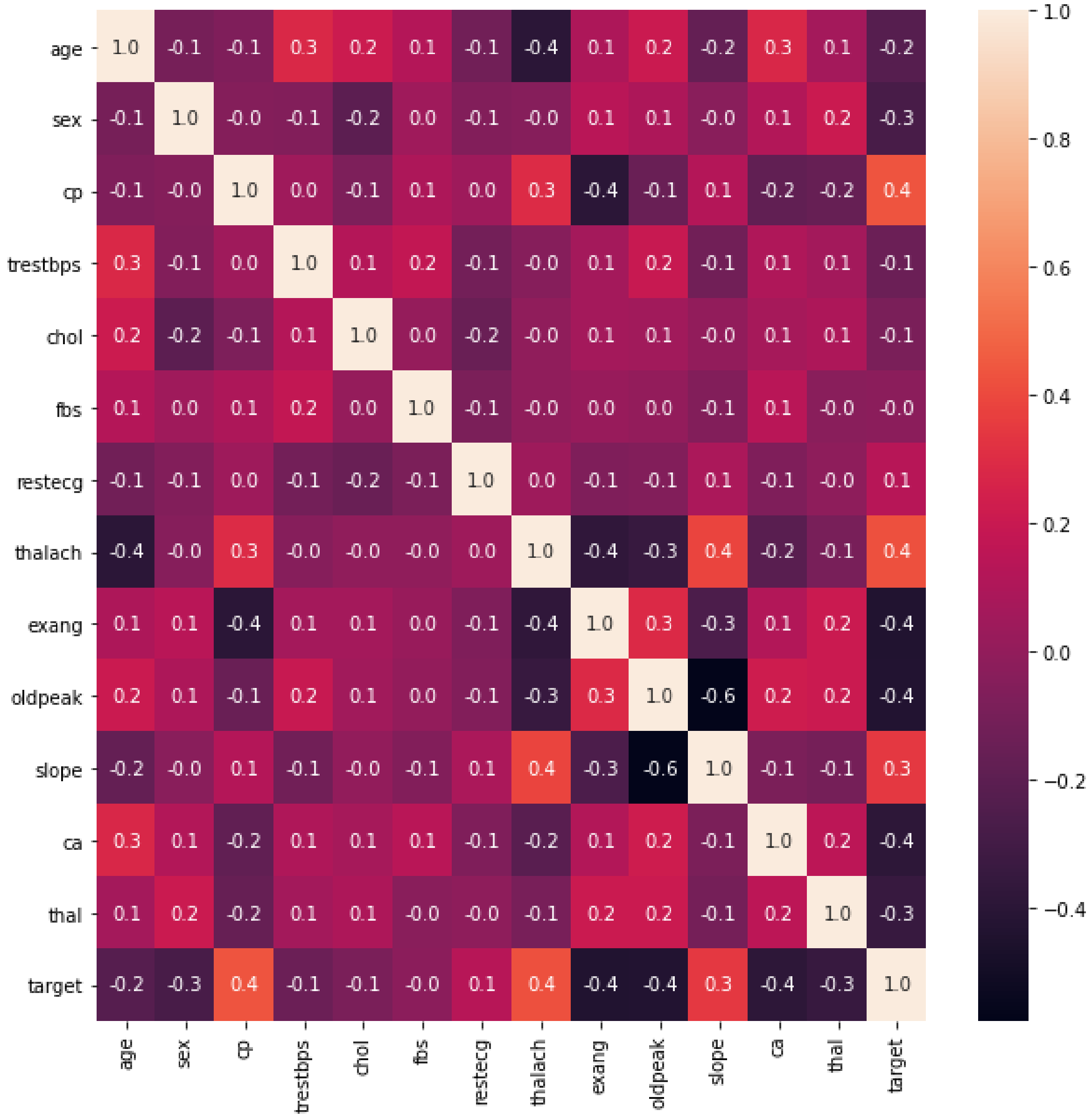
In [11]:

```
#Discover outliers with mathematical function
#Z-Score
#if the Z-score value is greater than or less than 3 or -3 respectively, that data poir
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(Heart_Disease))
print(z)
```

```
[[0.9521966  0.68100522 1.97312292 ... 0.71442887 2.14887271 0.91452919]
 [1.91531289 0.68100522 1.00257707 ... 0.71442887 0.51292188 0.91452919]
 [1.47415758 1.46841752 0.03203122 ... 0.71442887 0.51292188 0.91452919]
 ...
 [1.50364073 0.68100522 0.93851463 ... 1.24459328 1.12302895 1.09345881]
 [0.29046364 0.68100522 0.93851463 ... 0.26508221 1.12302895 1.09345881]
 [0.29046364 1.46841752 0.03203122 ... 0.26508221 0.51292188 1.09345881]]
```

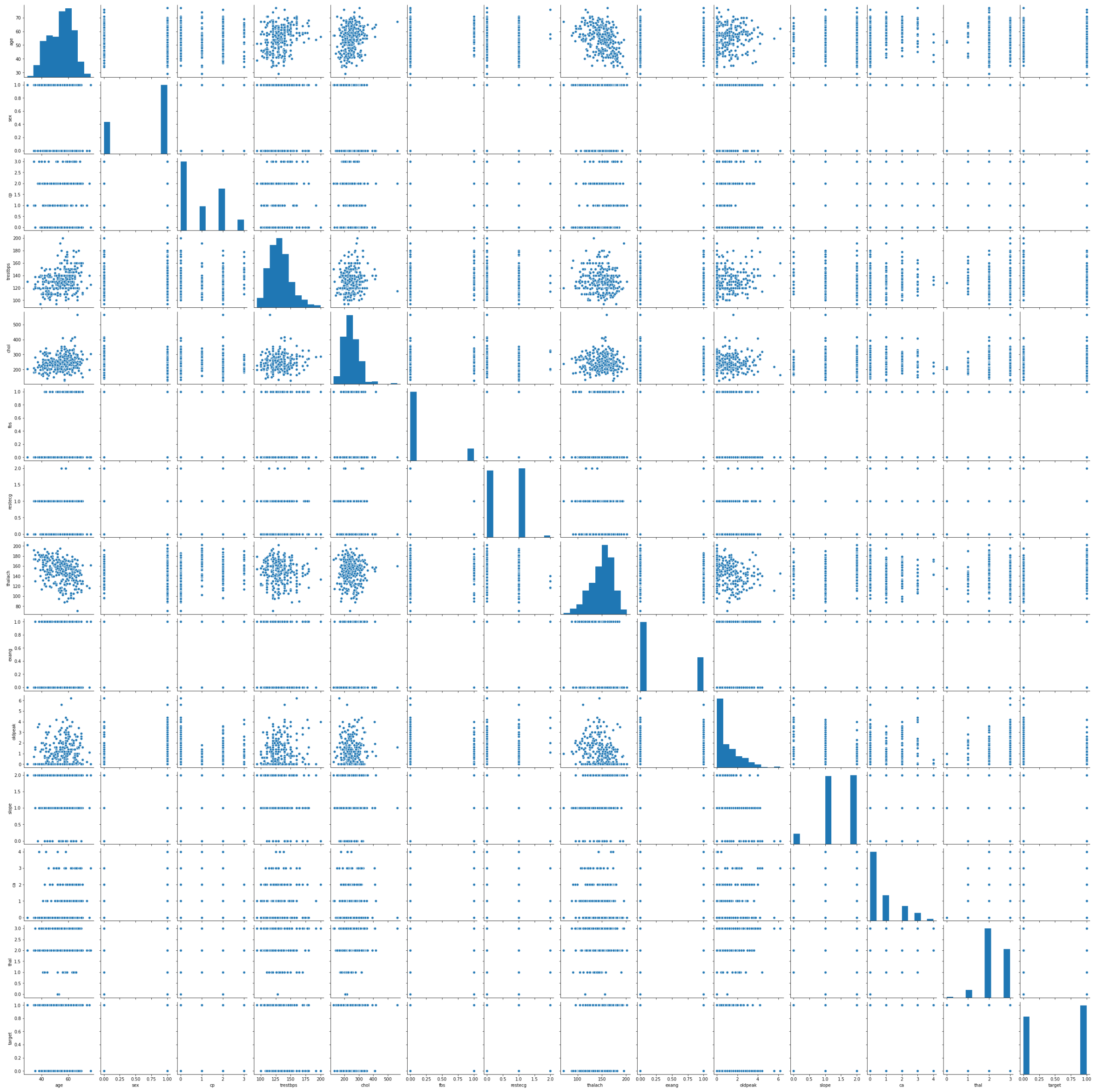
In [12]:

```
plt.figure(figsize=(10,10))
sns.heatmap(Heart_Disease.corr(),annot=True,fmt='.1f')
plt.show()
```



In [13]:

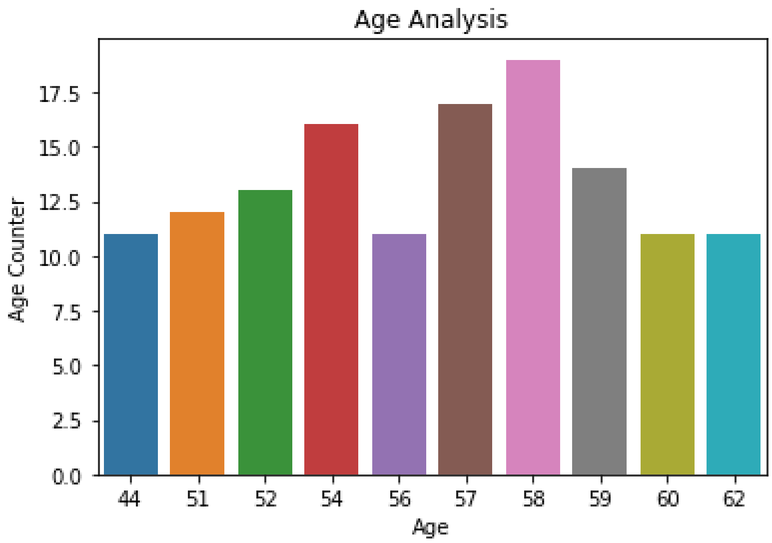
```
sns.pairplot(Heart_Disease)
plt.show()
```



In [14]:

```
sns.barplot(x=Heart_Disease.age.value_counts()[:10].index,y=Heart_Disease.age.value_cou
plt.xlabel('Age')
plt.ylabel('Age Counter')
plt.title('Age Analysis')
plt.show()

```



In [15]:

```
minAge=min(Heart_Disease.age)
maxAge=max(Heart_Disease.age)
meanAge=Heart_Disease.age.mean()
print('Min Age :',minAge)
print('Max Age :',maxAge)
```

```
Min Age : 29
Max Age : 77
```

In [16]:

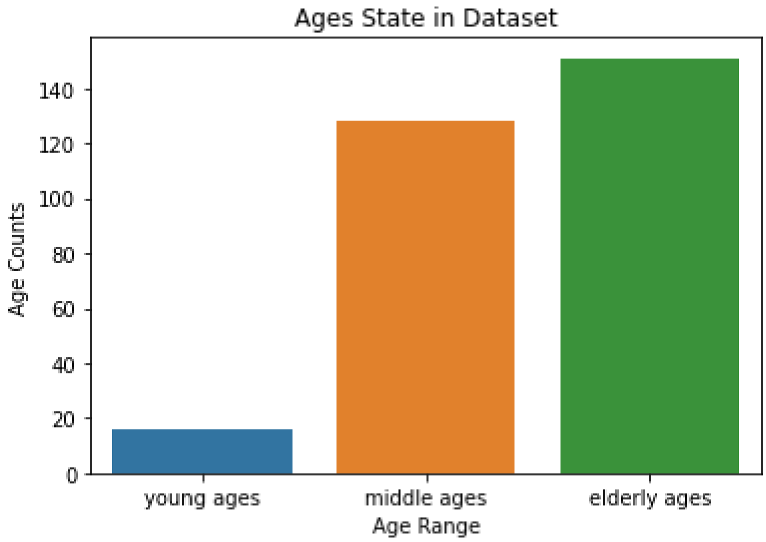
```
young_ages=Heart_Disease[(Heart_Disease.age>=29)&(Heart_Disease.age<40)]
middle_ages=Heart_Disease[(Heart_Disease.age>=40)&(Heart_Disease.age<55)]
elderly_ages=Heart_Disease[(Heart_Disease.age>55)]
print('Young Ages :',len(young_ages))
print('Middle Ages :',len(middle_ages))
print('Elderly Ages :',len(elderly_ages))

```

```
Young Ages : 16
Middle Ages : 128
Elderly Ages : 151
```

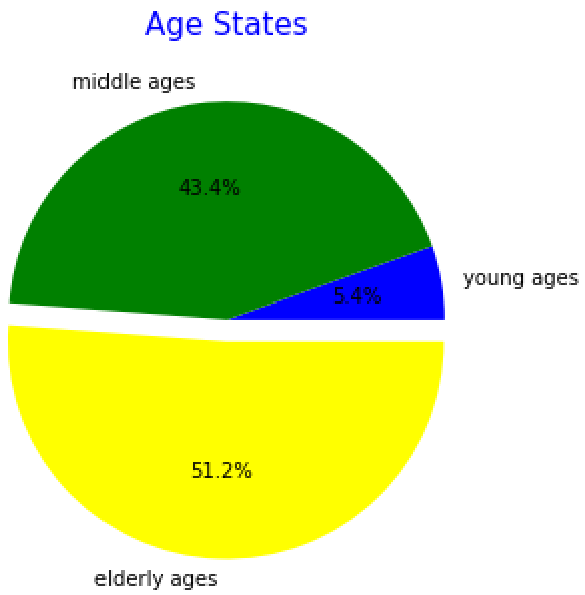
In [17]:

```
sns.barplot(x=['young ages','middle ages','elderly ages'],y=[len(young_ages),len(middle
plt.xlabel('Age Range')
plt.ylabel('Age Counts')
plt.title('Ages State in Dataset')
plt.show()
```



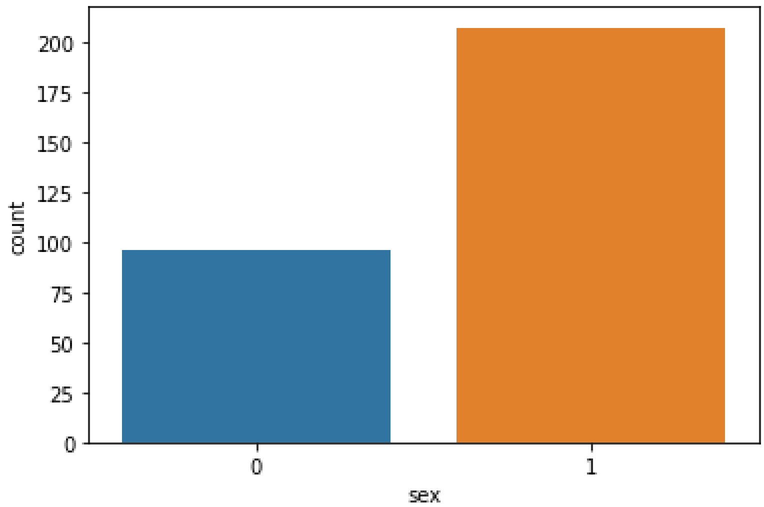
In [18]:

```
colors = ['blue','green','yellow']
explode = [0,0,0.1]
plt.figure(figsize = (5,5))
#plt.pie([target_0_agerang_0,target_1_agerang_0], explode=explode, labels=['Target 0 Ag
plt.pie([len(young_ages),len(middle_ages),len(elderly_ages)],labels=['young ages','midd
plt.title('Age States',color = 'blue',fontsize = 15)
plt.show()
```



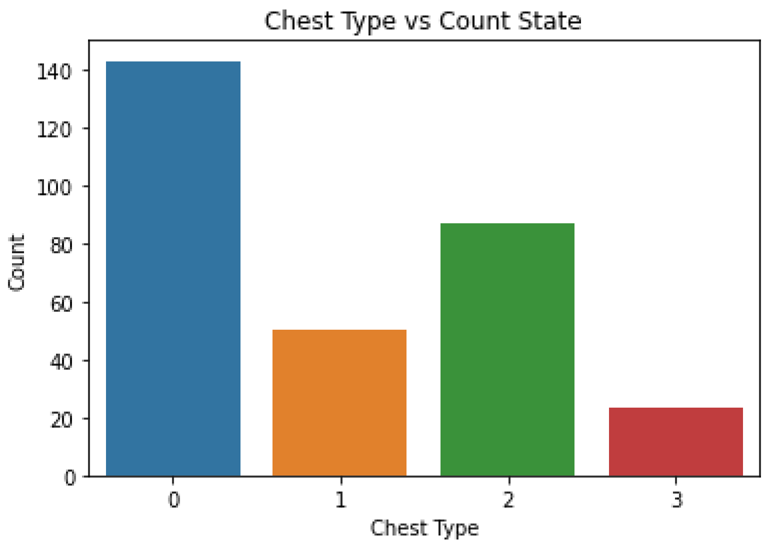
In [19]:

```
#Sex (1 = male; 0 = female)
sns.countplot(Heart_Disease.sex)
plt.show()
```



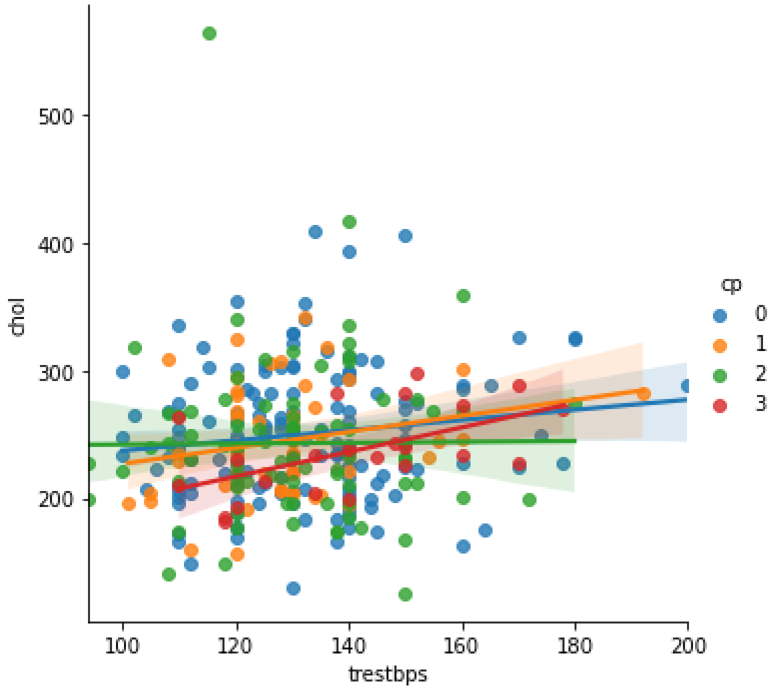
In [20]:

```
sns.countplot(Heart_Disease.cp)
plt.xlabel('Chest Type')
plt.ylabel('Count')
plt.title('Chest Type vs Count State')
plt.show()
#0 status at least
#1 condition slightly distressed
#2 condition medium problem
#3 condition too bad
```



In [21]:

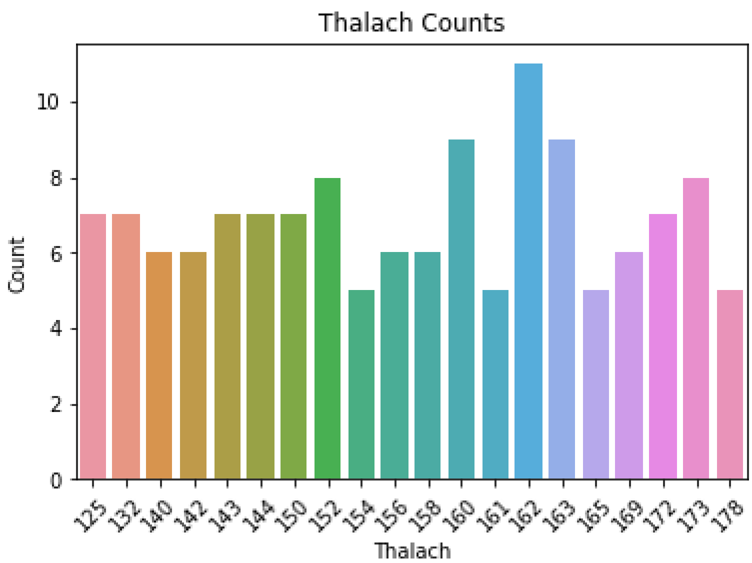
```
# Show the results of a linear regression within each dataset
sns.lmplot(x="trestbps", y="chol",data=Heart_Disease,hue="cp")
plt.show()
```



As a result of the above analyzes, it can be seen that 0 cases with chest pain are less common with heart disease. But on the other hand, there are problems in all cases of chest pain, such as 1,2,3.

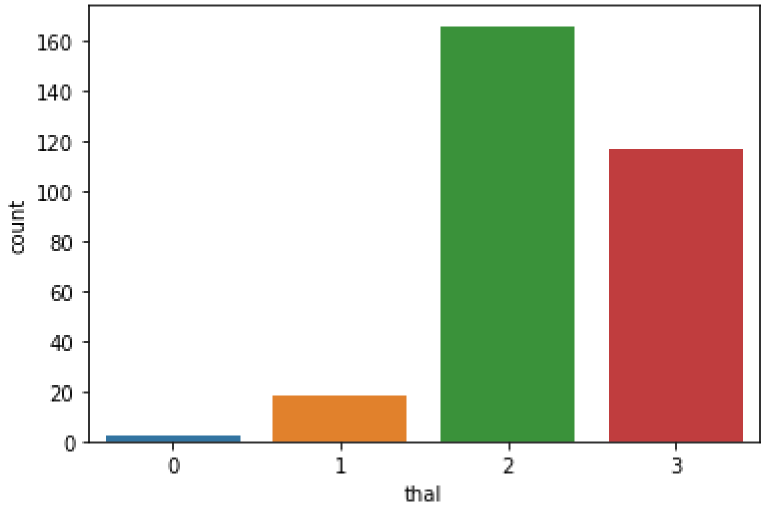
In [22]:

```
sns.barplot(x=Heart_Disease.thalach.value_counts()[:20].index,y=Heart_Disease.thalach.
plt.xlabel('Thalach')
plt.ylabel('Count')
plt.title('Thalach Counts')
plt.xticks(rotation=45)
plt.show()
```



In [23]:

```
sns.countplot(Heart_Disease.thal)
plt.show()
```



In [24]:

```
#Let's see the correlation values between them
Heart_Disease.corr()
```

Out[24]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741

Defining Dependent and Independent variables

Splitting Data into train and test with 70% and 20% respectively

In [25]:

```
X=Heart_Disease.drop(['target','slope'],axis=1)
#removing 'slope' to reduce the strong negative multicollinearity between 'slope' and
Y=Heart_Disease['target']
```

All Classification Algorithms with Default Parameters

In [26]:

```
import statsmodels.api as sm
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, random_state
logit = sm.Logit(y_train, X_train).fit()
print(logit.summary())
# attributes with p value less than 0.05 are statistically significant
```

```
Optimization terminated successfully.
         Current function value: 0.360832
         Iterations 7
                           Logit Regression Results
==========================================================================
====
Dep. Variable:                 target   No. Observations:
242
Model:                          Logit   Df Residuals:
230
Method:                           MLE   Df Model:
11
Date:                Wed, 07 Oct 2020   Pseudo R-squ.:                  0.
4757
Time:                        12:40:15   Log-Likelihood:                -8
7.321
converged:                       True   LL-Null:                       -16
6.55
Covariance Type:            nonrobust   LLR p-value:                 2.772
e-28
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
age            0.0128      0.020      0.628      0.530      -0.027
0.053
sex           -1.4981      0.481     -3.111      0.002      -2.442      -
0.554
cp             0.8414      0.208      4.052      0.000       0.434
1.248
trestbps      -0.0099      0.011     -0.911      0.362      -0.031
0.011
chol          -0.0020      0.004     -0.487      0.627      -0.010
0.006
fbs           -0.0504      0.603     -0.084      0.933      -1.232
1.131
restecg        0.7180      0.386      1.861      0.063      -0.038
1.474
thalach        0.0322      0.009      3.412      0.001       0.014
0.051
exang         -1.1158      0.446     -2.504      0.012      -1.989      -
0.243
oldpeak       -0.8555      0.225     -3.808      0.000      -1.296      -
0.415
ca            -0.7468      0.210     -3.558      0.000      -1.158      -
0.335
thal          -0.9405      0.335     -2.804      0.005      -1.598      -
0.283
```

```
==============================================================================
====
```

In [27]:

```
X=X.drop(['restecg','fbs','chol','trestbps','age'],axis=1)
```

In [28]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state
logit = sm.Logit(y_train, X_train).fit()
print(logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.371069
         Iterations 7
                           Logit Regression Results
==============================================================================
====
Dep. Variable:                 target   No. Observations:
242
Model:                          Logit   Df Residuals:
235
Method:                           MLE   Df Model:
6
Date:                Wed, 07 Oct 2020   Pseudo R-squ.:                  0.
4608
Time:                        12:40:15   Log-Likelihood:                -8
9.799
converged:                       True   LL-Null:                       -16
6.55
Covariance Type:            nonrobust   LLR p-value:                 1.406
e-30
==============================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
----
sex           -1.3929      0.443     -3.144      0.002      -2.261      -
0.524
cp             0.7918      0.196      4.040      0.000       0.408
1.176
thalach        0.0264      0.005      5.097      0.000       0.016
0.037
exang         -1.1179      0.422     -2.648      0.008      -1.945      -
0.290
oldpeak       -0.8437      0.210     -4.012      0.000      -1.256      -
0.432
ca            -0.7163      0.200     -3.585      0.000      -1.108      -
0.325
thal          -0.8721      0.305     -2.855      0.004      -1.471      -
0.273
==============================================================================
====
```

In [29]:

```
y_pred = logit.predict(X_test)
prediction = list(map(round, y_pred))
```

In [30]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
cm1= confusion_matrix(y_test,prediction)
print('Confusion Matrix : ')
print(cm1)
from sklearn.metrics import accuracy_score
print ("Accuracy Score : ", accuracy_score(y_test, prediction))
print ('Report : ')
print (classification_report(y_test, prediction))
```

```
Confusion Matrix :
[[25  4]
 [ 3 29]]
Accuracy Score :  0.8852459016393442
Report :
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

In [31]:

```
#Sensitivity and Specificity
#A test with a sensitivity and specificity of around 90% would be considered to have gc

sensitivity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity )

specificity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity)
```

```
Sensitivity :  0.8620689655172413
Specificity :  0.90625
```

In [32]:

```
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,random_state=42)
```

1. Logistic Regression Algorithm

In [33]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

Out[33]:

```
LogisticRegression()
```

In [34]:

```
y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curve, a
acclog = accuracy_score(y_test, y_pred)*100
reclog = recall_score(y_test, y_pred)*100
preclog = precision_score(y_test, y_pred)*100
fprlog, tprlog, _ = roc_curve(y_test, y_pred)
auclog=auc(fprlog, tprlog)*100
```

In [35]:

```
from sklearn.metrics import confusion_matrix
cm1= confusion_matrix(y_test,y_pred)
print('Confusion Matrix :' )
print(cm1)
from sklearn.metrics import accuracy_score
print ("Accuracy Score: ", accuracy_score(y_test, y_pred))
print ('Report : ')
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix :
[[25  4]
 [ 4 28]]
Accuracy Score:  0.8688524590163934
Report :
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        29
           1       0.88      0.88      0.88        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

In [36]:

```
#Sensitivity and Specificity
#A test with a sensitivity and specificity of around 90% would be considered to have gc

sensitivity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity )

specificity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity)
```

```
Sensitivity :  0.8620689655172413
Specificity :  0.875
```

In [37]:

```
yl = model.predict_proba(X_test)
```

2. K Nearest Neighbor Algorithm

In [38]:

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

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'euclidean')
knn.fit(X_train, y_train)

# Predicting the Test set results
y_pred = knn.predict(X_test)

yk = knn.predict_proba(X_test)

from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curve, a
accknn = accuracy_score(y_test, y_pred)*100
recknn = recall_score(y_test, y_pred)*100
precknn = precision_score(y_test, y_pred)*100
fprknn, tprknn, _ = roc_curve(y_test, y_pred)
aucknn=auc(fprknn, tprknn)*100

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(y_test, y_pred)

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

results = confusion_matrix(y_test, y_pred)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(y_test, y_pred))
print ('Report : ')
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix :
[[26  3]
 [ 4 28]]
Accuracy Score : 0.8852459016393442
Report : 
              precision    recall  f1-score   support

           0       0.87      0.90      0.88        29
           1       0.90      0.88      0.89        32

    accuracy                           0.89        61
   macro avg       0.88      0.89      0.89        61
weighted avg       0.89      0.89      0.89        61
```

In [39]:

```
sensitivity = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity )

specificity = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity)
```

```
Sensitivity :  0.896551724137931
Specificity :  0.875
```

3.Support Vector Machine Algorithm

In [40]:

```
# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
svm = SVC(kernel = 'rbf', random_state = 0, probability=True)
svm.fit(X_train, y_train)

# Predicting the Test set results
y_pred = svm.predict(X_test)

ys = svm.predict_proba(X_test)

from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curve, a
accsvm = accuracy_score(y_test, y_pred)*100
recsvm = recall_score(y_test, y_pred)*100
precsvm = precision_score(y_test, y_pred)*100
fprsvm, tprsvm, _ = roc_curve(y_test, y_pred)
aucsvm=auc(fprsvm, tprsvm)*100

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm3 = confusion_matrix(y_test, y_pred)

results = confusion_matrix(y_test, y_pred)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(y_test, y_pred))
print ('Report : ')
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix :
[[26  3]
 [ 3 29]]
Accuracy Score : 0.9016393442622951
Report :
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        29
           1       0.91      0.91      0.91        32

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61
```

In [41]:

```
sensitivity = cm3[0,0]/(cm3[0,0]+cm3[0,1])
print('Sensitivity : ', sensitivity )

specificity = cm3[1,1]/(cm3[1,0]+cm3[1,1])
print('Specificity : ', specificity)
```

```
Sensitivity :  0.896551724137931
Specificity :  0.90625
```

4. Gaussian Naive Bayes Algorithm

In [42]:

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

# Predicting the Test set results
y_pred = gnb.predict(X_test)

yg = gnb.predict_proba(X_test)

from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curve, a
accgnb = accuracy_score(y_test, y_pred)*100
recgnb = recall_score(y_test, y_pred)*100
precgnb = precision_score(y_test, y_pred)*100
fprgnb, tprgnb, _ = roc_curve(y_test, y_pred)
aucgnb=auc(fprgnb, tprgnb)*100

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm4 = confusion_matrix(y_test, y_pred)

results = confusion_matrix(y_test, y_pred)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(y_test, y_pred))
print ('Report : ')
print (classification_report(y_test, y_pred))
```

```
Confusion Matrix :
[[26  3]
 [ 6 26]]
Accuracy Score : 0.8524590163934426
Report :
              precision    recall  f1-score   support

           0       0.81      0.90      0.85        29
           1       0.90      0.81      0.85        32

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.86      0.85      0.85        61
```

In [43]:

```
sensitivity = cm4[0,0]/(cm4[0,0]+cm4[0,1])
print('Sensitivity : ', sensitivity )

specificity = cm4[1,1]/(cm4[1,0]+cm4[1,1])
print('Specificity : ', specificity)
```

```
Sensitivity :  0.896551724137931
Specificity :  0.8125
```

Comparison of all the Machine Learning Algorithms by Comparing some Evaluation Metrics

In [44]:

```
algos=["Logistic Regression","K Nearest Neighbor","Support Vector Machine","Gaussian Na
acc=[acclog,accknn,accsvm,accgnb]
auc=[auclog,aucknn,aucsvm,aucgnb]
recall=[reclog,recknn,recsvm,recgnb]
prec=[preclog,precknn,precsvm,precgnb]
comp={"Algorithms":algos,"Accuracies":acc,"Area Under the Curve":auc,"Recall":recall,"P
compdf=pd.DataFrame(comp)
display(compdf)
#display(compdf.sort_values(by=["Accuracies","Area Under the Curve","Recall","Precision
```

	Algorithms	Accuracies	Area Under the Curve	Recall	Precision
0	Logistic Regression	86.885246	86.853448	87.500	87.500000
1	K Nearest Neighbor	88.524590	88.577586	87.500	90.322581
2	Support Vector Machine	90.163934	90.140086	90.625	90.625000
3	Gaussian Naive Bayes	85.245902	85.452586	81.250	89.655172

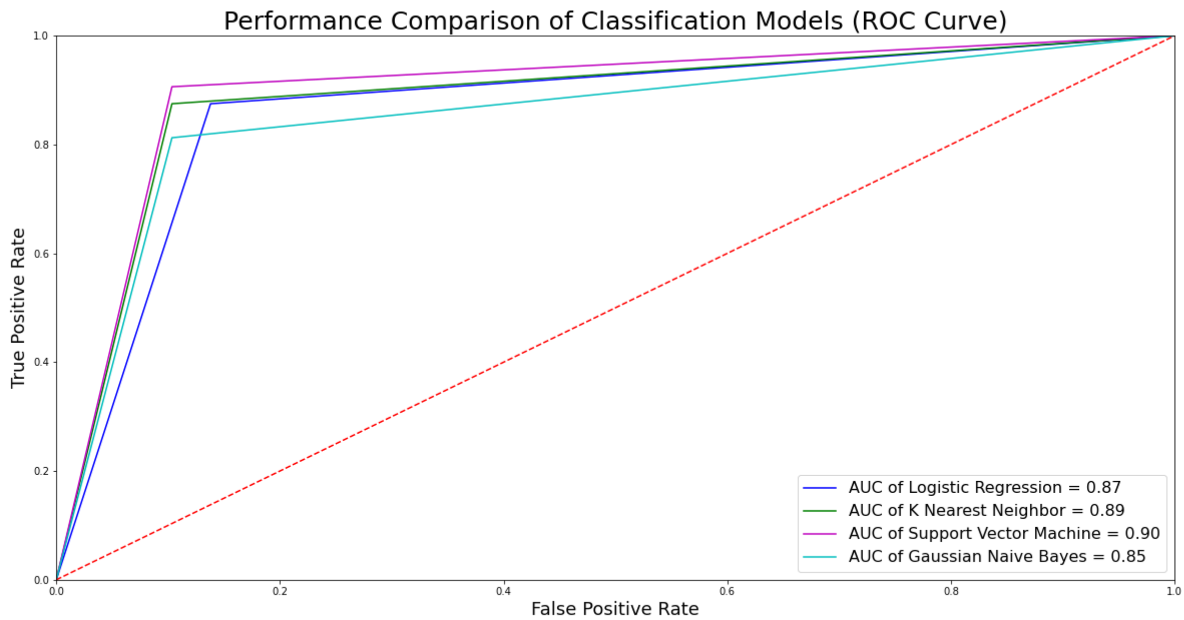
ROC of all the Machine Learning Algorithms on default parameters

In [45]:

```
import sklearn.metrics as metrics
roc_auc1=metrics.auc(fprlog,tprlog)
roc_auc2=metrics.auc(fprknn,tprknn)
roc_auc3=metrics.auc(fprsvm,tprsvm)
roc_auc4=metrics.auc(fprgnb,tprgnb)

import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(20,10))
plt.title("Performance Comparison of Classification Models (ROC Curve)", fontsize=25)
plt.plot(fprlog,tprlog,"b",label="AUC of Logistic Regression = %0.2f" % roc_auc1)
plt.plot(fprknn,tprknn,"g",label="AUC of K Nearest Neighbor = %0.2f" % roc_auc2)
plt.plot(fprsvm,tprsvm,"m",label="AUC of Support Vector Machine = %0.2f" % roc_auc3)
plt.plot(fprgnb,tprgnb,"c",label="AUC of Gaussian Naive Bayes = %0.2f" % roc_auc4)
plt.rcParams.update({'font.size': 16})
plt.legend(loc="lower right")
plt.plot([0, 1],[0, 1],"r--")
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel("True Positive Rate", fontsize = 18)
plt.xlabel("False Positive Rate", fontsize = 18)

plt.rc('axes', labelsize=15)
plt.rc('axes', titlesize=22)
```



Cross Validation Score

In [46]:

```
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
```

In [47]:

```
accuracy_log1 = cross_val_score(model, X, Y, scoring='accuracy', cv = 10)
#print('CVS for log1 : ', accuracy_svc)
print("Accuracy of LOG with Cross Validation is:",accuracy_log1.mean() * 100)
#accuracy_log = cross_val_score(log, X, Y, cv = 10)
#print('CVS for LOG : ', accuracy_svc)
#print("Accuracy of LOG with Cross Validation is:",accuracy_log.mean() * 100)
accuracy_svc = cross_val_score(svm, X, Y, cv = 10)
#print('CVS for SVC : ', accuracy_svc)
print("Accuracy of SVC with Cross Validation is:",accuracy_svc.mean() * 100)
accuracy_gnb = cross_val_score(gnb, X, Y, scoring='accuracy', cv = 10)
#print('CVS for GNB : ', accuracy_gnb)
print("Accuracy of GNB with Cross Validation is:",accuracy_gnb.mean() * 100)
accuracy_knn = cross_val_score(knn, X, Y, scoring='accuracy', cv = 10)
#print('CVS for knn : ', accuracy_gnb)
print("Accuracy of KNN with Cross Validation is:",accuracy_knn.mean() * 100)
```

```
Accuracy of LOG with Cross Validation is: 83.16129032258065
Accuracy of SVC with Cross Validation is: 69.98924731182797
Accuracy of GNB with Cross Validation is: 82.49462365591398
Accuracy of KNN with Cross Validation is: 76.89247311827957
```

In [48]:

```
algos=["Logistic Regression","K Nearest Neighbor","Support Vector Machine","Gaussian Na
acc1=[acclog,accknn,accsvm,accgnb]
acc2=[accuracy_log1.mean() * 100, accuracy_knn.mean() * 100, accuracy_svc.mean() * 100,
comp={"Algorithms":algos,"Accuracies without Cross Validation":acc1,"Accuracies with Cr
compdf=pd.DataFrame(comp)
display(compdf)
```

	Algorithms	Accuracies without Cross Validation	Accuracies with Cross Validation
0	Logistic Regression	86.885246	83.161290
1	K Nearest Neighbor	88.524590	76.892473
2	Support Vector Machine	90.163934	69.989247
3	Gaussian Naive Bayes	85.245902	82.494624

In [ ]:

```

```

In [ ]:

```
```