In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [3]:
```
df = pd.read_csv("E:\level 4 term 1\CSE 441\83\heart.csv")
```

In [4]:
```
df.head()
```

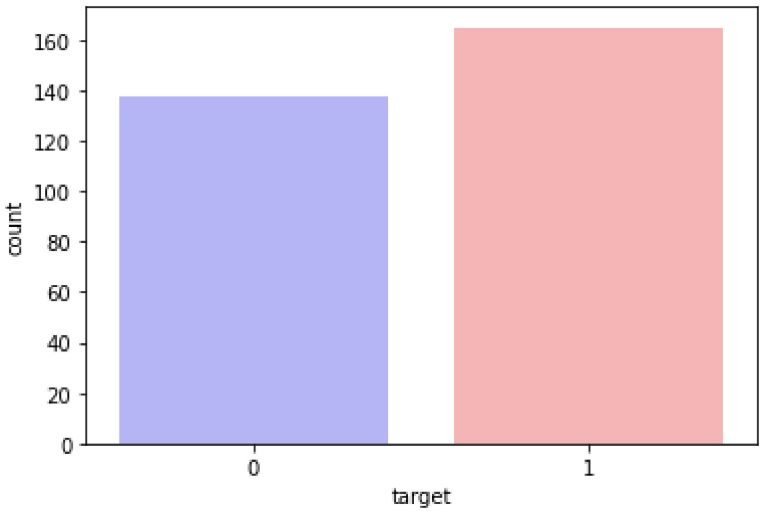
Out[4]:

	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*

In [5]:
```
#_Data Exploration
df.target.value_counts()
```

Out[5]:
```
1    165
0    138
Name: target, dtype: int64
```

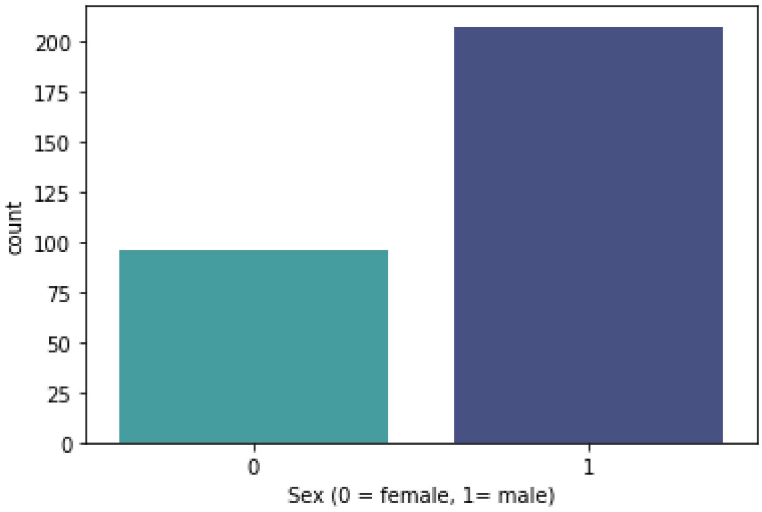
In [6]:
```
sns.countplot(x="target", data=df, palette="bwr")
plt.show()
```



In [7]:
```
countNoDisease = len(df[df.target == 0])
countHaveDisease = len(df[df.target == 1])
print("Percentage of Patients Haven't Heart Disease: {:.2f}%".format((countNoD
isease / (len(df.target))*100)))
print("Percentage of Patients Have Heart Disease: {:.2f}%".format((countHaveDi
sease / (len(df.target))*100)))
```

```
Percentage of Patients Haven't Heart Disease: 45.54%
Percentage of Patients Have Heart Disease: 54.46%
```

In [8]:
```
sns.countplot(x='sex', data=df, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



In [9]:
```
countFemale = len(df[df.sex == 0])
countMale = len(df[df.sex == 1])
print("Percentage of Female Patients: {:.2f}%".format((countFemale / (len(df.s
ex))*100)))
print("Percentage of Male Patients: {:.2f}%".format((countMale / (len(df.sex))
*100)))
```

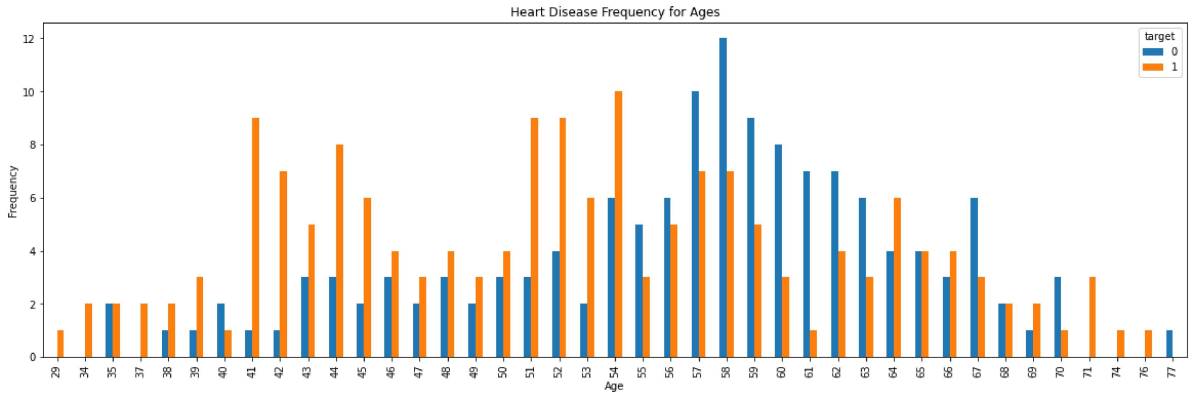
```
Percentage of Female Patients: 31.68%
Percentage of Male Patients: 68.32%
```

In [10]:
```
df.groupby('target').mean()
```

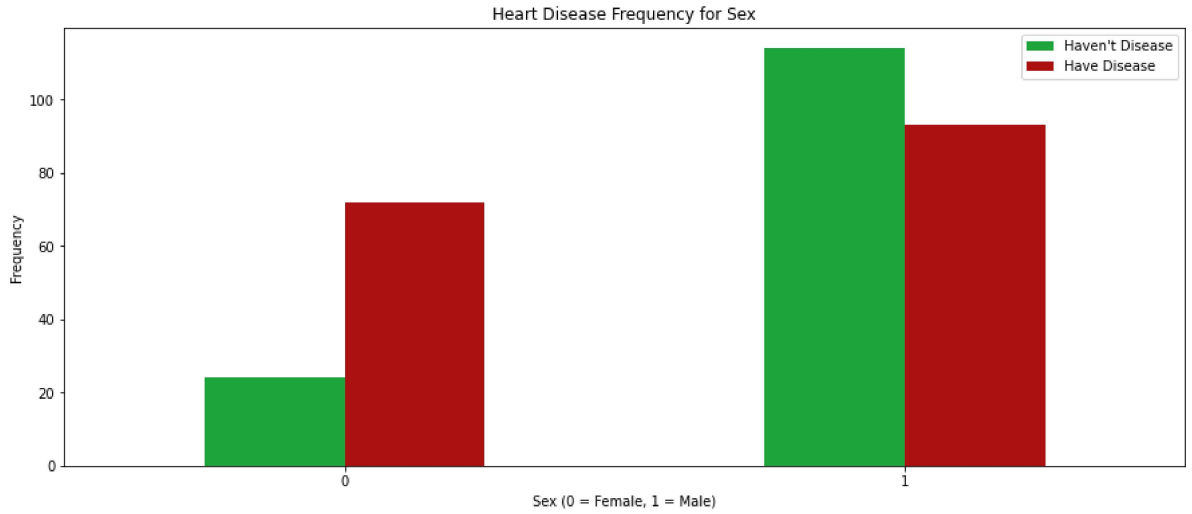
Out[10]:

target	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	o
0	56.601449	0.826087	0.478261	134.398551	251.086957	0.159420	0.449275	139.101449	0.550725	1.
1	52.496970	0.563636	1.375758	129.303030	242.230303	0.139394	0.593939	158.466667	0.139394	0.

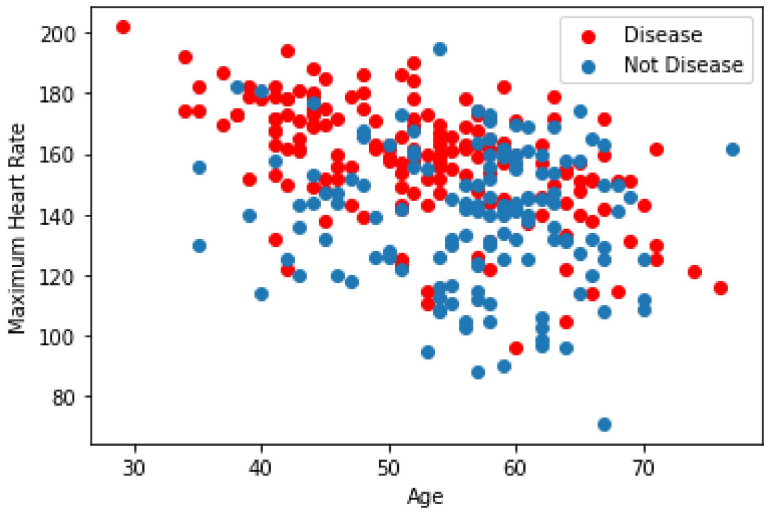
In [11]:
```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



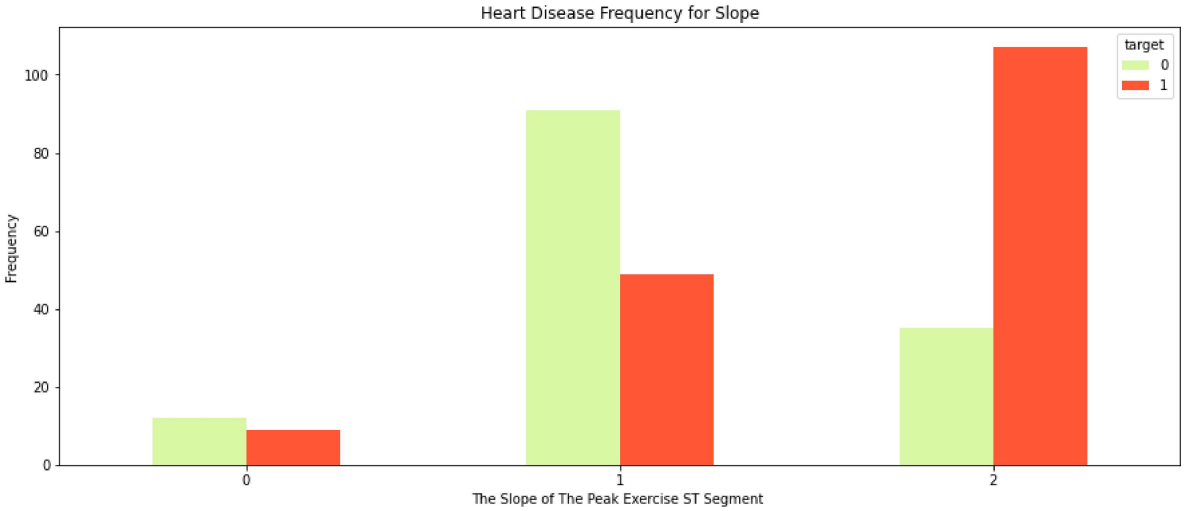
In [12]:
```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#1CA53B',
'#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()
```



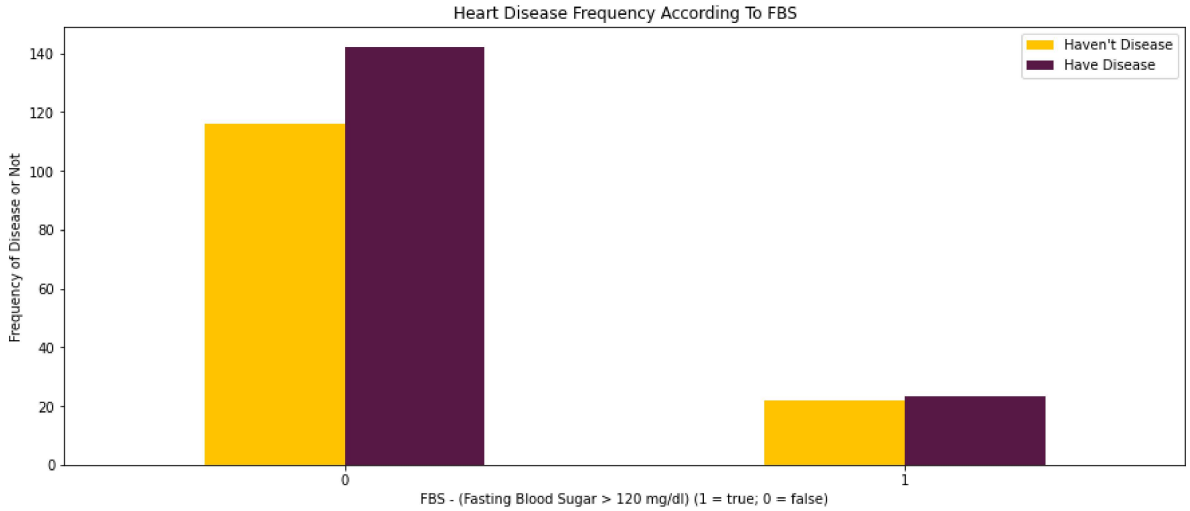
In [13]:
```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



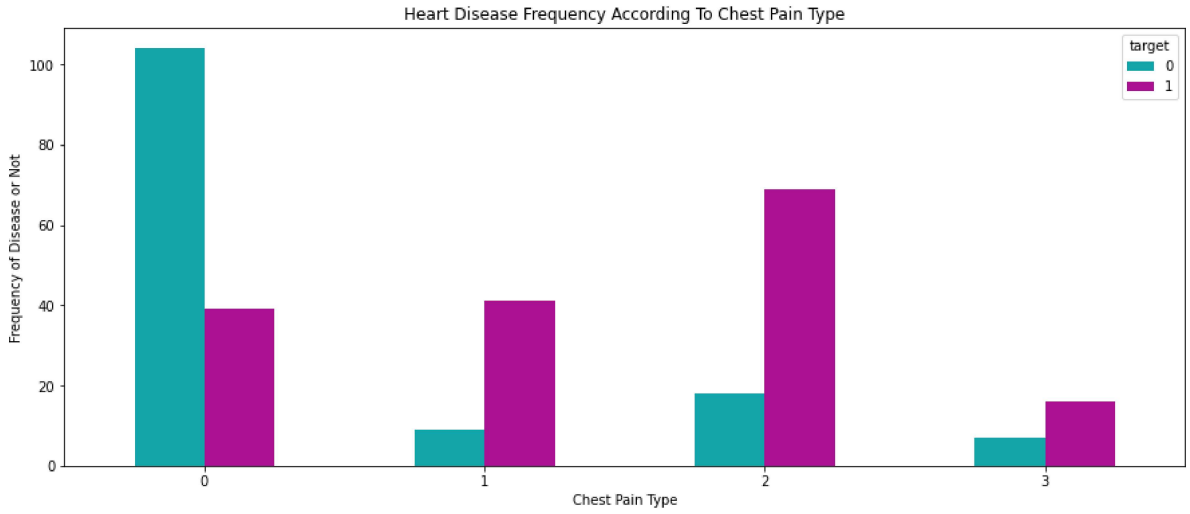
In [14]:
```
pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#DAF7A6','#FF5733' ])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



In [15]:
```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#FFC300',
'#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



In [16]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA',
'#AA1190' ])
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



In [18]:
```
#Creating_Dummy_Variables

a = pd.get_dummies(df['cp'], prefix = "cp")
b = pd.get_dummies(df['thal'], prefix = "thal")
c = pd.get_dummies(df['slope'], prefix = "slope")
```

In [19]:
```
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()
```

Out[19]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	...	cp_1	cp_2	cp_3	thal_0	t
0	63	1	3	145	233	1	0	150	0	2.3	...	0	0	1	0	
1	37	1	2	130	250	0	1	187	0	3.5	...	0	1	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	...	1	0	0	0	
3	56	1	1	120	236	0	1	178	0	0.8	...	1	0	0	0	
4	57	0	0	120	354	0	1	163	1	0.6	...	0	0	0	0	

5 rows × 25 columns

In [20]:
```
df = df.drop(columns = ['cp', 'thal', 'slope'])
df.head()
```

Out[20]:

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	...	cp_1	cp_2	cp_3	thal_0	t
0	63	1	145	233	1	0	150	0	2.3	0	...	0	0	1	0	
1	37	1	130	250	0	1	187	0	3.5	0	...	0	1	0	0	
2	41	0	130	204	0	0	172	0	1.4	0	...	1	0	0	0	
3	56	1	120	236	0	1	178	0	0.8	0	...	1	0	0	0	
4	57	0	120	354	0	1	163	1	0.6	0	...	0	0	0	0	

5 rows × 22 columns

In [21]:
```
#Creating Model for Logistic Regression

y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

In [22]:
```
#Normalize Data

x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

In [23]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random
_state=0)
```

In [24]:
```
#transpose matrices

x_train = x_train.T
y_train = y_train.T
x_test = x_test.T
y_test = y_test.T
```

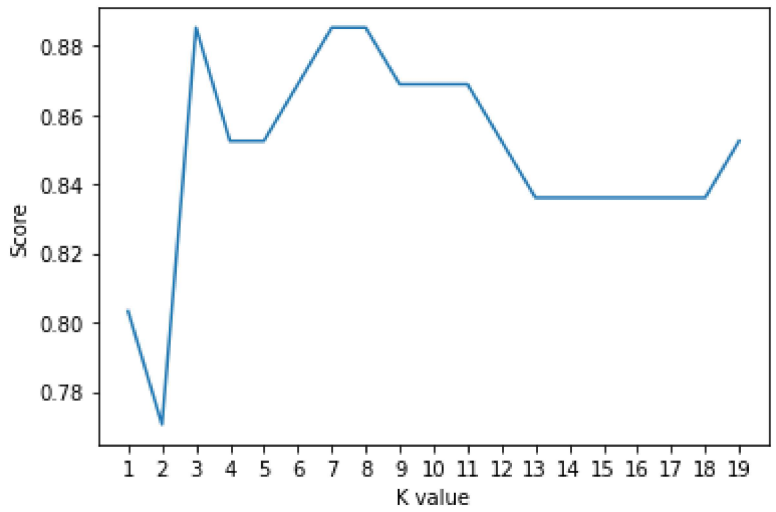
In [25]:
```
#initialize

def initialize(dimension):

    weight = np.full((dimension,1),0.01)
    bias = 0.0
    return weight,bias
```

In [26]:
```
def sigmoid(z):

    y_head = 1/(1+ np.exp(-z))
    return y_head
```

In [27]:
```
#Sklearn Logistic Regression

accuracies = {}

lr = LogisticRegression()
lr.fit(x_train.T,y_train.T)
acc = lr.score(x_test.T,y_test.T)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))
```

```
Test Accuracy 86.89%
```

In [32]:
```
#K-Nearest Neighbour (KNN) Classification
# find best k value
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
    scoreList.append(knn2.score(x_test.T, y_test.T))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))
```




```
Maximum KNN Score is 88.52%
```

In [33]:
```
#Support Vector Machine Algorithm

from sklearn.svm import SVC
```

In [34]:
```
svm = SVC(random_state = 1)
svm.fit(x_train.T, y_train.T)

acc = svm.score(x_test.T,y_test.T)*100
accuracies['SVM'] = acc
print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
```

```
Test Accuracy of SVM Algorithm: 88.52%
```

In [35]:
```
#Naive Bayes Algorithm

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train.T, y_train.T)

acc = nb.score(x_test.T,y_test.T)*100
accuracies['Naive Bayes'] = acc
print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
```

```
Accuracy of Naive Bayes: 86.89%
```

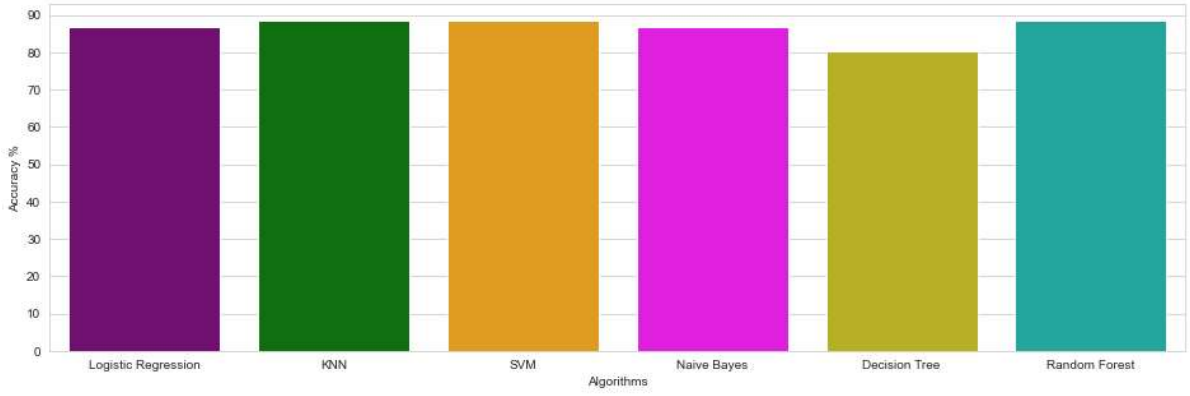
In [36]:
```
#Decision Tree Algorithm

from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train.T, y_train.T)

acc = dtc.score(x_test.T, y_test.T)*100
accuracies['Decision Tree'] = acc
print("Decision Tree Test Accuracy {:.2f}%".format(acc))
```

```
Decision Tree Test Accuracy 80.33%
```

In [37]:
```
# Random Forest Classification

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train.T, y_train.T)

acc = rf.score(x_test.T,y_test.T)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
```

```
Random Forest Algorithm Accuracy Score : 88.52%
```

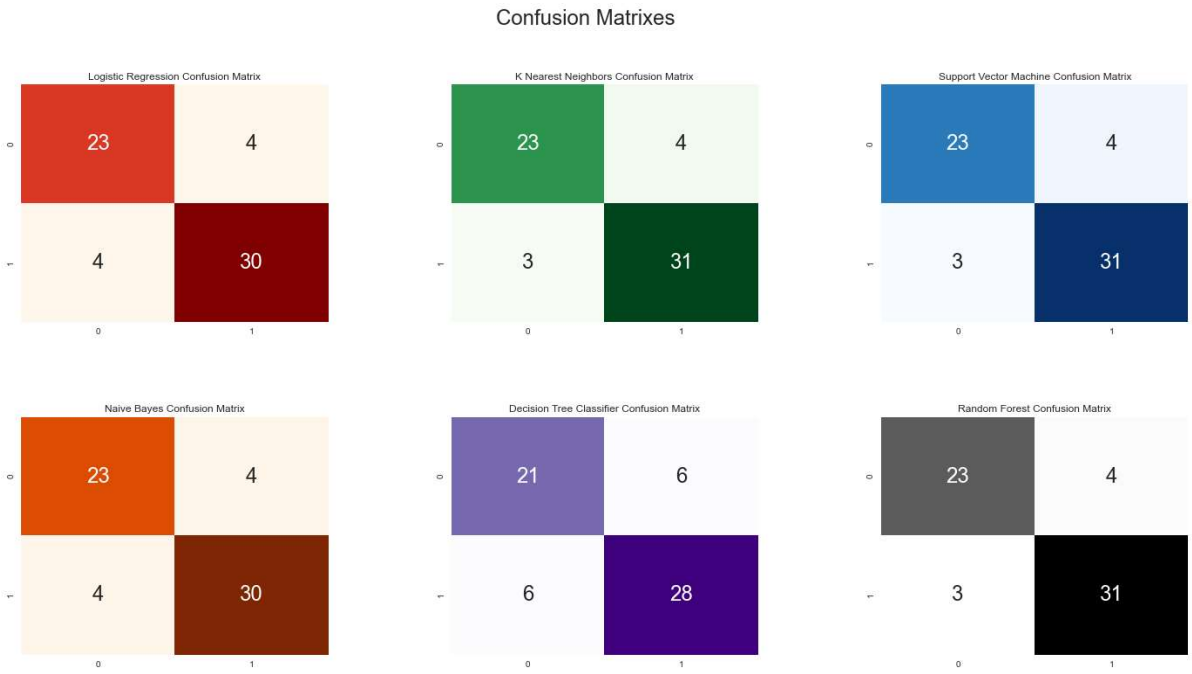
In [38]:
```
#Comparing Models

colors = ["purple", "green", "orange", "magenta","#CFC60E","#0FBBAE"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=co
lors)
plt.show()
```



In [39]:
```
# Predicted values
y_head_lr = lr.predict(x_test.T)
knn3 = KNeighborsClassifier(n_neighbors = 3)
knn3.fit(x_train.T, y_train.T)
y_head_knn = knn3.predict(x_test.T)
y_head_svm = svm.predict(x_test.T)
y_head_nb = nb.predict(x_test.T)
y_head_dtc = dtc.predict(x_test.T)
y_head_rf = rf.predict(x_test.T)
```

In [40]:
```
#Confusion Matrix

from sklearn.metrics import confusion_matrix

cm_lr = confusion_matrix(y_test,y_head_lr)
cm_knn = confusion_matrix(y_test,y_head_knn)
cm_svm = confusion_matrix(y_test,y_head_svm)
cm_nb = confusion_matrix(y_test,y_head_nb)
cm_dtc = confusion_matrix(y_test,y_head_dtc)
cm_rf = confusion_matrix(y_test,y_head_rf)
```

In [51]:
```
plt.figure(figsize=(24,12))

plt.suptitle("Confusion Matrixes",fontsize=24)
plt.subplots_adjust(wspace = 0.4, hspace= 0.4)

plt.subplot(2,3,1)
plt.title("Logistic Regression Confusion Matrix")
sns.heatmap(cm_lr,annot=True,cmap="OrRd",fmt="d",cbar=False, annot_kws={"size"
: 24})

plt.subplot(2,3,2)
plt.title("K Nearest Neighbors Confusion Matrix")
sns.heatmap(cm_knn,annot=True,cmap="Greens",fmt="d",cbar=False, annot_kws={"si
ze": 24})

plt.subplot(2,3,3)
plt.title("Support Vector Machine Confusion Matrix")
sns.heatmap(cm_svm,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"siz
e": 24})

plt.subplot(2,3,4)
plt.title("Naive Bayes Confusion Matrix")
sns.heatmap(cm_nb,annot=True,cmap="Oranges",fmt="d",cbar=False, annot_kws={"si
ze": 24})

plt.subplot(2,3,5)
plt.title("Decision Tree Classifier Confusion Matrix")
sns.heatmap(cm_dtc,annot=True,cmap="Purples",fmt="d",cbar=False, annot_kws={"s
ize": 24})

plt.subplot(2,3,6)
plt.title("Random Forest Confusion Matrix")
sns.heatmap(cm_rf,annot=True,cmap="Greys",fmt="d",cbar=False, annot_kws={"siz
e": 24})

plt.show()
```



In [ ]: