CardioVascular Disease Prediction using Machine Learning models

A heart attack is a medical emergency when a blood clot blocks blood flow to the heart. Without blood, tissue loses oxygen and dies.It is estimated that dietary risk factors are associated with 53% of cardiovascular deaths. Cardiovascular diseases are the leading cause of death worldwide. As per WHO , 17.9 million people every year die from heart attack. According to a medical study, human lifestyle is the main reason behind this heart problem. Apart from this there are many key factors which warns that the person may/maynot getting chance of heart attack.

The early prognosis of cardiovascular diseases can aid in making decisions on lifestyle changes in high risk patients and in turn reduce the complications.

The goal of this project is to pinpoint the most relevant/risk factors of heart disease as well as predict the overall risk using logistic regression and other classifier models.

This dataset was chosen from Kaggle website and this consists of 12 attributes,which include:

```
1.  age: age in years
2.  sex: sex (1 = male; 0 = female)
3.Chestpaintype: chest pain type( 0= typical angina; 1= atypical angina; 2=non-anginal pain; 3=

4.RestingBP: resting blood pressure (in mm Hg on admission to the hospital)
5.Cholesterol : serum cholestoral in mg/dl
6.FastingBS : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7.RestingECG  : resting electrocardiographic results
( 0: normal
  1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.0
  2: showing probable or definite left ventricular hypertrophy by Estes' criteria)

8.MaxHR  : maximum heart rate achieved
9.ExerciseAngina : exercise induced angina (1 = yes; 0 = no)
10.oldpeak = ST depression induced by exercise relative to rest
11.ST_slope: the slope of the peak exercise ST segment ( 0: upsloping, 1: flat, 2: downsloping)

12.HeartDisease: 0 = no disease, 1 = disease
```

[75]:
```python
#Import the neccessary libraries
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
import plotly
import plotly.express as px

from pandas_profiling import ProfileReport
from sklearn.model_selection import␣
↪train_test_split,cross_val_score,GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score,roc_curve,roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,␣
↪confusion_matrix,f1_score,accuracy_score,precision_score,recall_score
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Lasso
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from numpy import std
%matplotlib inline
```

[76]:
```python
#Import the Dataset
df = pd.read_csv('C:/Users/chait/Desktop/CPS-NEU/git/heart.csv')
df.head()
```

[76]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	\
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	N	0.0	Up	0
1	N	1.0	Flat	1
2	N	0.0	Up	0

```
3              Y      1.5     Flat            1
4              N      0.0       Up            0
```

[77]:
```python
#checking for nulls
df.isnull().sum()
```

[77]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[78]:
```python
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

[78]:
```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
```

```
25%     47.000000  120.000000  173.250000    0.000000  120.000000
50%     54.000000  130.000000  223.000000    0.000000  138.000000
75%     60.000000  140.000000  267.000000    0.000000  156.000000
max     77.000000  200.000000  603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

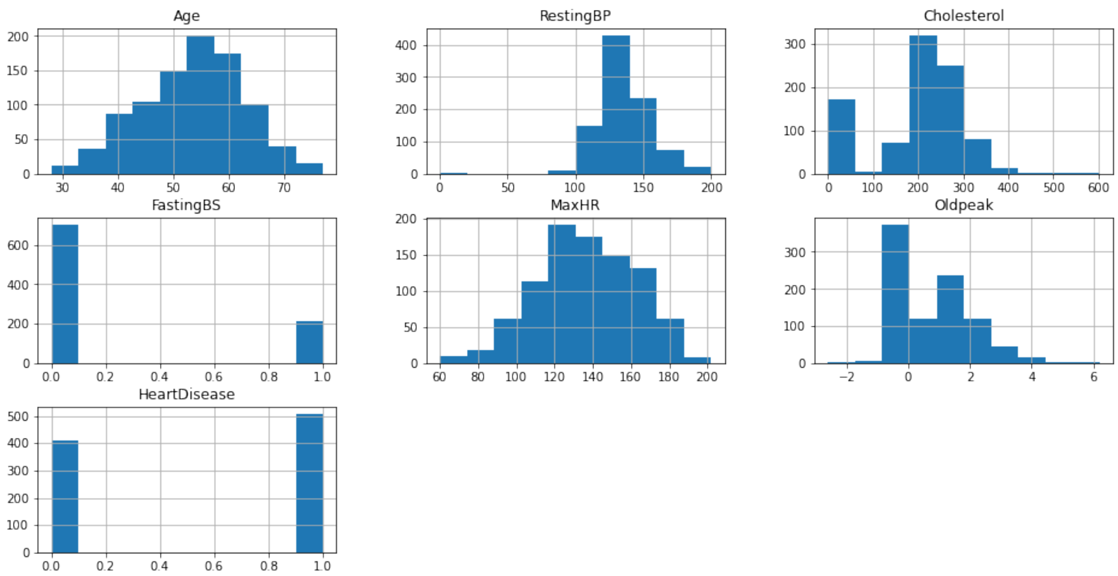
From this analysis,we see that there is a vast difference in the values of the variables.The max for Age is 77 and the max for Cholestrol is 603.This indicates the need for FeatureScaling.

2 Exploratory Data Analysis

Lets do an exploratory data analysis of the variables used in the dataset which can aid our study

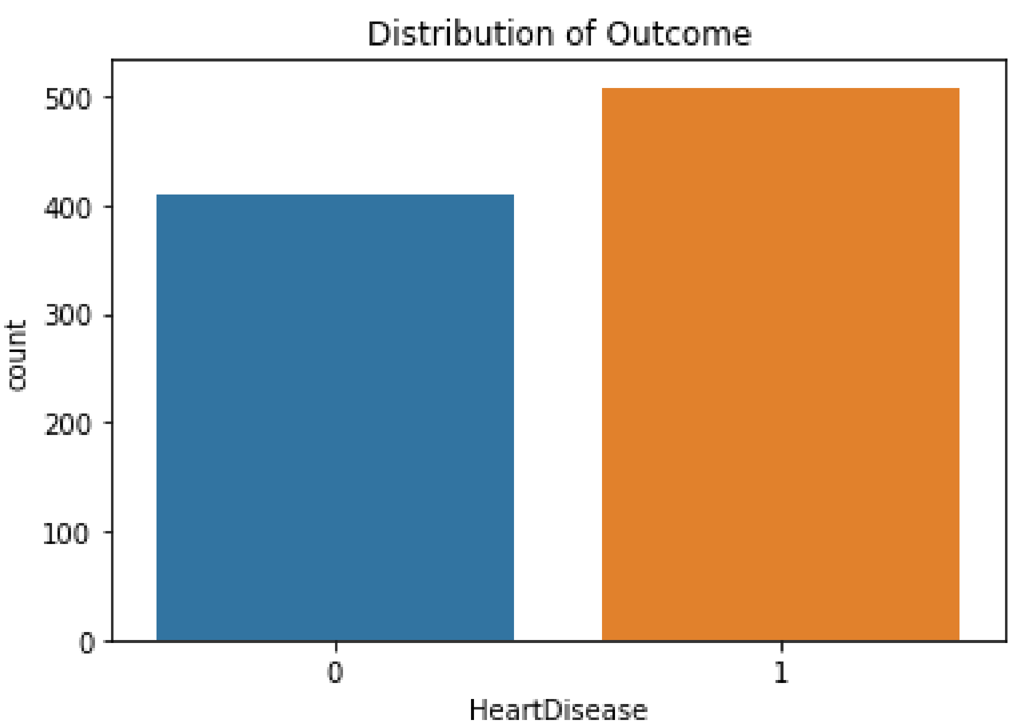
[79]:
```
df.hist(figsize=(16, 8))
```

[79]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'RestingBP'}>,
        <AxesSubplot:title={'center':'Cholesterol'}>],
       [<AxesSubplot:title={'center':'FastingBS'}>,
        <AxesSubplot:title={'center':'MaxHR'}>,
        <AxesSubplot:title={'center':'Oldpeak'}>],
       [<AxesSubplot:title={'center':'HeartDisease'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



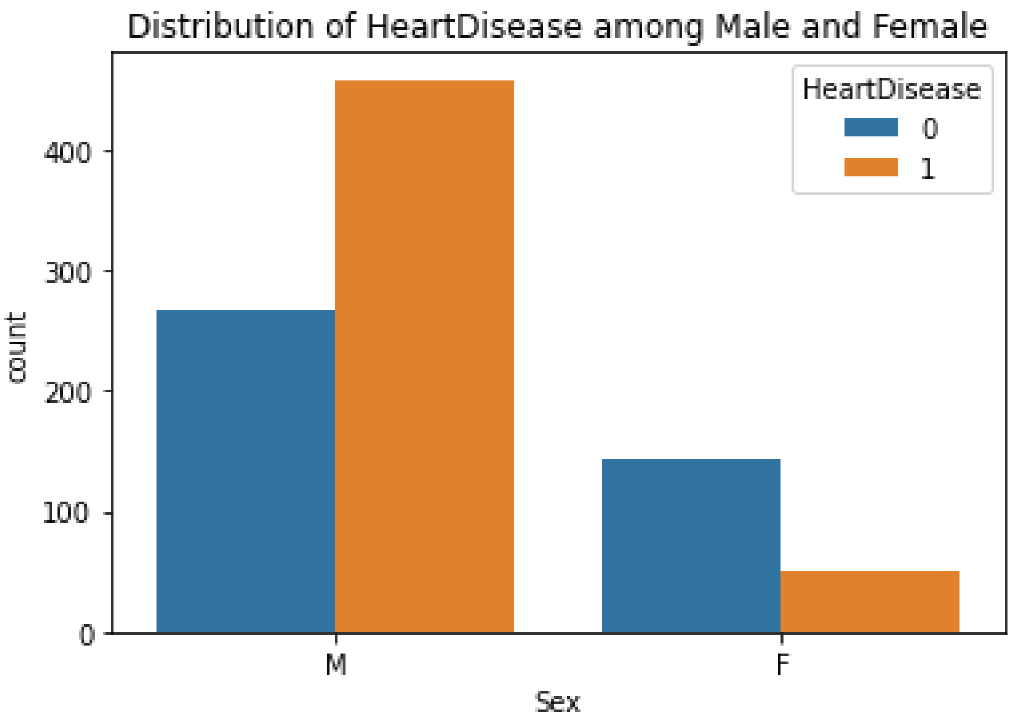
[80]:
```
#Distribution of Outcome variable
df['HeartDisease'].value_counts()

sns.countplot(x='HeartDisease',data=df).set_title("Distribution of Outcome")
```

[80]: `Text(0.5, 1.0, 'Distribution of Outcome')`



from this plot, we see that our target variable is balanced and we dont see much difference.

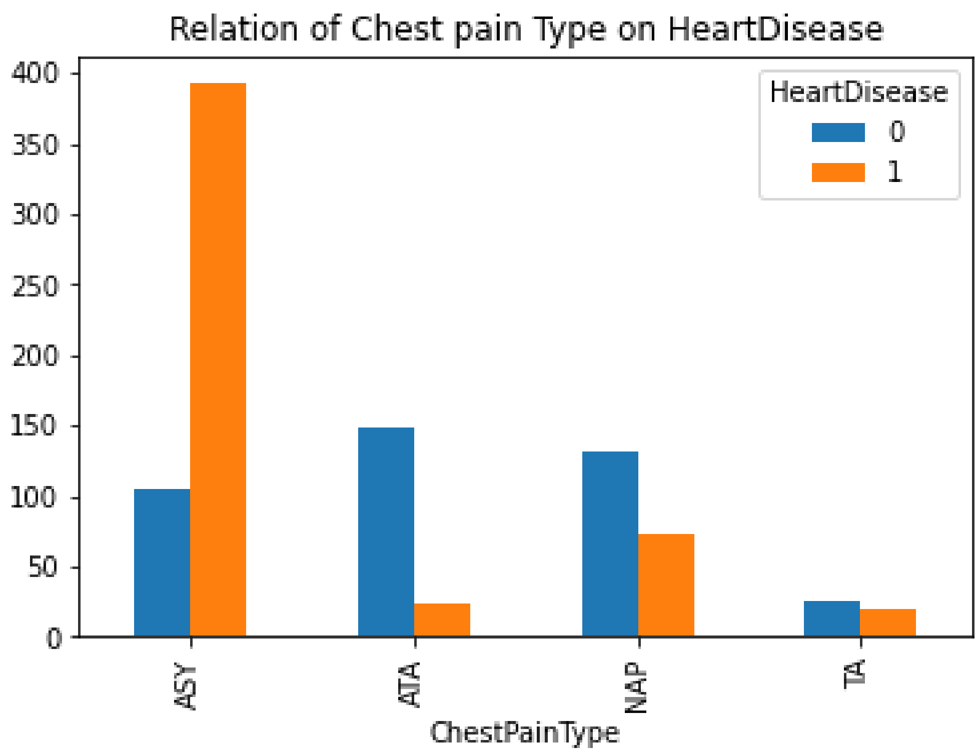
[81]:
```
df.groupby('Sex').count()
sns.countplot(x = df['Sex'], hue = df['HeartDisease']).set_title("Distribution␣
↪of HeartDisease among Male and Female")
plt.show()
```



We see that,Males are more prone to heart attacks as compared to women.

[83]:
```
pd.crosstab(df.ChestPainType,df.HeartDisease).plot(kind='bar',title="Relation␣
↪of Chest pain Type on HeartDisease")
```

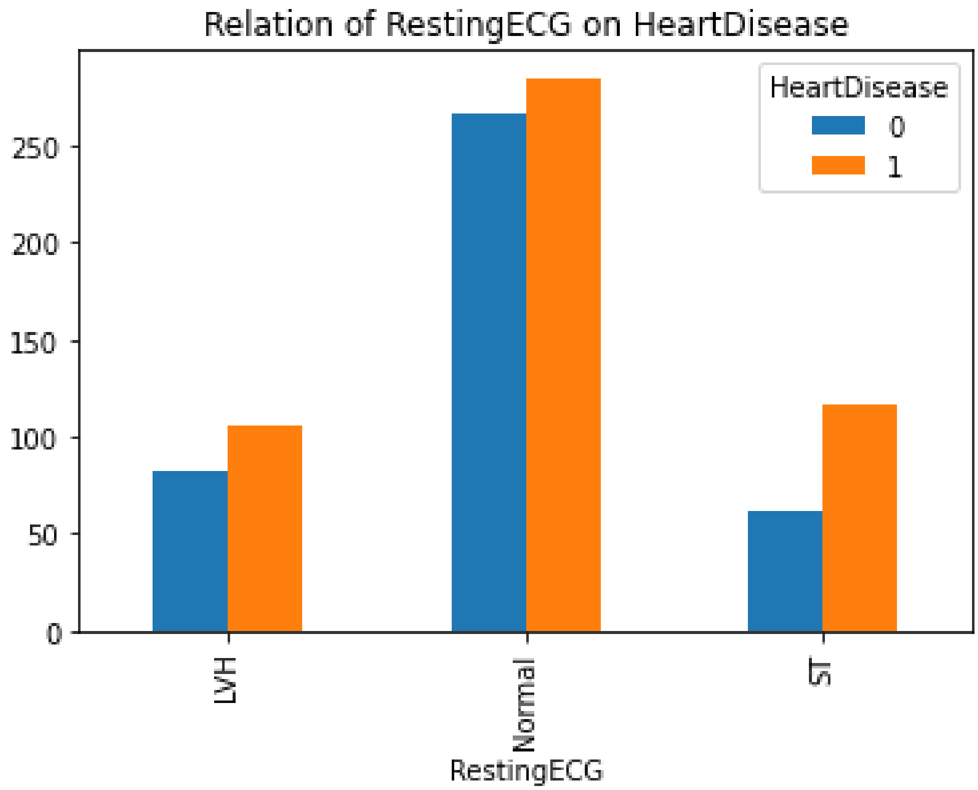
[83]:
```
<AxesSubplot:title={'center':'Relation of Chest pain Type on HeartDisease'},
xlabel='ChestPainType'>
```



[27]:
```
plt.figure(figsize = (10,10))
pd.crosstab(df.RestingECG,df.HeartDisease).plot(kind='bar',title="Relation of␣
↪RestingECG on HeartDisease")
```

[27]:
```
<AxesSubplot:title={'center':'Relation of RestingECG on HeartDisease'},
xlabel='RestingECG'>
```

```
<Figure size 720x720 with 0 Axes>
```

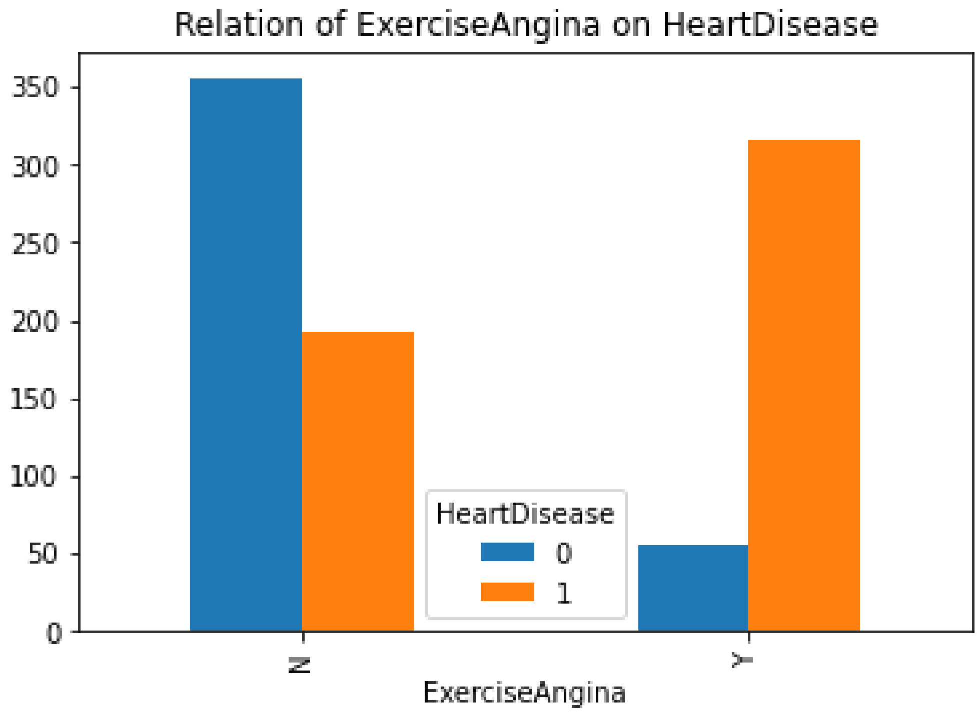


We see that people with RestingECG type "Normal" are more prone to heartDisease than LVH and ST

[84]:
```
plt.figure(figsize = (16,8))
pd.crosstab(df.ExerciseAngina,df.HeartDisease).plot(kind='bar',title="Relation␣
↪of ExerciseAngina on HeartDisease")
```

[84]:
```
<AxesSubplot:title={'center':'Relation of ExerciseAngina on HeartDisease'},
xlabel='ExerciseAngina'>
```

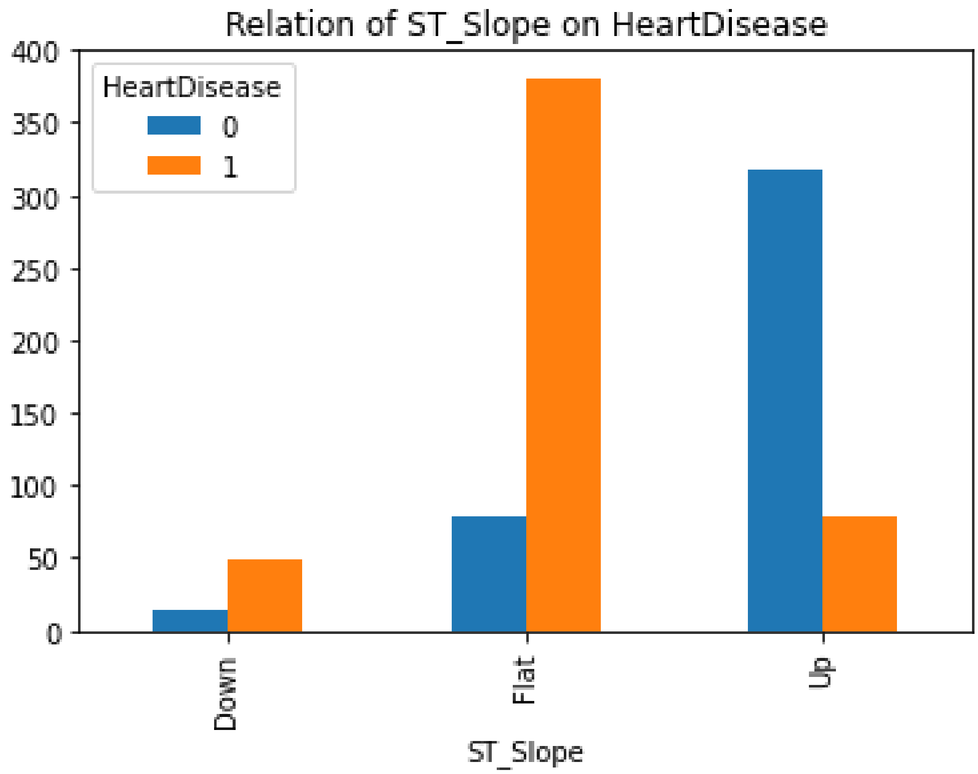
```
<Figure size 1152x576 with 0 Axes>
```



We see that people with ExerciseAngina are more prone to heartDisease than people with non ExerciseAngina

[85]:
```
pd.crosstab(df.ST_Slope,df.HeartDisease).plot(kind='bar',title="Relation of␣
↪ST_Slope on HeartDisease")
```

[85]:
```
<AxesSubplot:title={'center':'Relation of ST_Slope on HeartDisease'},
xlabel='ST_Slope'>
```



We see that people with Flat ST_slope are more prone to HeartDisease than others

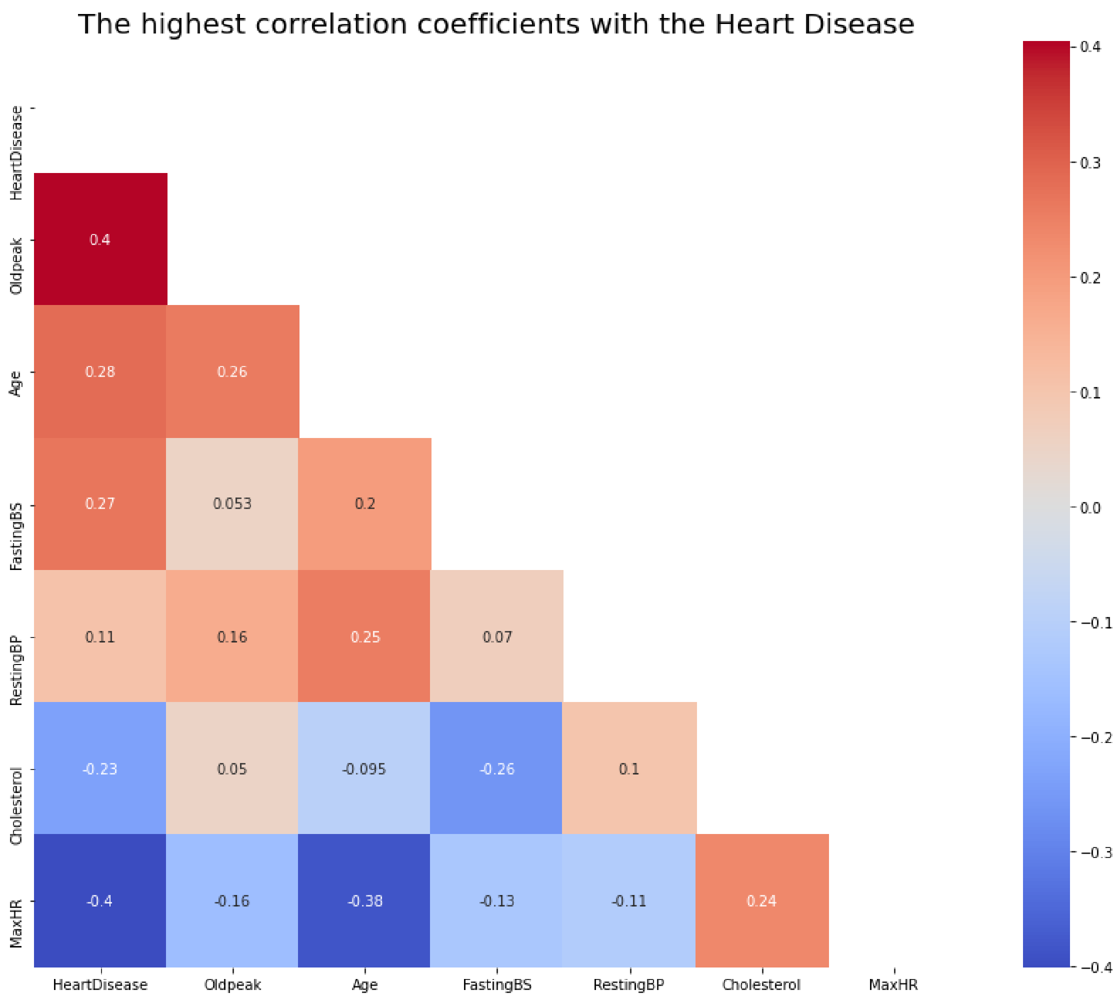
3 Correlation matrix and the highly correlated features with the target

[38]:
```
correlation=df.corr()
correlation

correlation.sort_values(["HeartDisease"], ascending = False, inplace = True)
correlation['HeartDisease'] = round(correlation['HeartDisease'],2)
correlation.drop('HeartDisease', axis = 0, inplace = True)
print("Highly correlated features with the target variable")
print(correlation.HeartDisease.head(50))
```

```
Highly correlated features with the target variable
Oldpeak        0.40
Age            0.28
FastingBS      0.27
RestingBP      0.11
Cholesterol   -0.23
MaxHR         -0.40
Name: HeartDisease, dtype: float64
```

[32]:
```python
# Plot the corrplot of top 20 highly correlated features.
top_corr = df.corr()['HeartDisease'].sort_values(ascending=False).head(20).index
print(top_corr)

#Plot heatmap of top twenty positively correlated features.
plt.figure(figsize=(16,12))
mask = np.triu(np.ones_like(df[top_corr].corr(), dtype=bool))
ax = sns.heatmap(df[top_corr].corr(), cmap='coolwarm', mask=mask, square=True,␣
 ↪annot=True)
plt.title('The highest correlation coefficients with the Heart Disease',␣
 ↪fontsize=20);
```

```
Index(['HeartDisease', 'Oldpeak', 'Age', 'FastingBS', 'RestingBP',
       'Cholesterol', 'MaxHR'],
      dtype='object')
```



We convert the categorical variables into numerical variables by using the LabelEncoder method

[39]:
```
from sklearn.preprocessing import LabelEncoder

# Ordinal categorical columns
label_encoding_cols =␣
↪["Sex","ChestPainType","RestingECG","ExerciseAngina","ST_Slope"]

# Apply Label Encoder
label_encoder = LabelEncoder()

for col in label_encoding_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

4 Recursive feature elimination

We repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. The goal of Recursive Feature Elimination is to select features by recursively considering smaller and smaller sets of features.

[41]:
```
import statsmodels.api as sm
cols=df.columns[:-1]

model=sm.Logit(df.HeartDisease,df[cols])
result=model.fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.362984
         Iterations 7
```

[41]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:           HeartDisease   No. Observations:                  918
Model:                          Logit   Df Residuals:                      907
Method:                           MLE   Df Model:                           10
Date:                Thu, 17 Mar 2022   Pseudo R-squ.:                  0.4720
Time:                        22:44:35   Log-Likelihood:                -333.22
converged:                       True   LL-Null:                       -631.07
Covariance Type:            nonrobust   LLR p-value:                1.470e-121
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
```

```
Age                0.0273      0.010      2.753      0.006       0.008
0.047
Sex                1.4336      0.253      5.673      0.000       0.938
1.929
ChestPainType     -0.6925      0.106     -6.525      0.000      -0.901
-0.485
RestingBP          0.0077      0.005      1.537      0.124      -0.002
0.018
Cholesterol       -0.0036      0.001     -3.472      0.001      -0.006
-0.002
FastingBS          1.0875      0.257      4.229      0.000       0.584
1.592
RestingECG        -0.1288      0.161     -0.799      0.424      -0.445
0.187
MaxHR             -0.0039      0.004     -1.086      0.278      -0.011
0.003
ExerciseAngina     1.1403      0.229      4.985      0.000       0.692
1.589
Oldpeak            0.3684      0.115      3.209      0.001       0.143
0.594
ST_Slope          -1.6891      0.206     -8.182      0.000      -2.094
-1.284
================================================================================
==
"""
```

The p-values for most of the variables are smaller than 0.05, except three variables, therefore, I will remove them.

[265]:
```
X = df.drop(['HeartDisease'],axis = 1) #predictor variables
Y = df.HeartDisease
def back_feature_elem (data_frame,target_var,col_list):

    while len(col_list)>0 :
        model=sm.Logit(target_var,data_frame[col_list])
        result=model.fit(disp=0)
        largest_pvalue=round(result.pvalues,3).nlargest(1)
        if largest_pvalue[0]<(0.05):
            return result
            break
        else:
            col_list=col_list.drop(largest_pvalue.index)

result=back_feature_elem(X,Y,cols)
```

[266]:
```
result.summary()
```

[266]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:           HeartDisease   No. Observations:                  918
Model:                          Logit   Df Residuals:                      910
Method:                           MLE   Df Model:                            7
Date:                Fri, 11 Mar 2022   Pseudo R-squ.:                  0.4696
Time:                        15:53:51   Log-Likelihood:                -334.69
converged:                       True   LL-Null:                       -631.07
Covariance Type:            nonrobust   LLR p-value:                8.839e-124
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
Age                0.0343      0.007      4.983      0.000       0.021
0.048
Sex                1.4244      0.245      5.819      0.000       0.945
1.904
ChestPainType     -0.7019      0.104     -6.755      0.000      -0.906
-0.498
Cholesterol       -0.0034      0.001     -3.754      0.000      -0.005
-0.002
FastingBS          1.0897      0.256      4.255      0.000       0.588
1.592
ExerciseAngina     1.1873      0.223      5.334      0.000       0.751
1.624
Oldpeak            0.3641      0.113      3.232      0.001       0.143
0.585
ST_Slope          -1.7258      0.183     -9.431      0.000      -2.084
-1.367
================================================================================
==
"""
```

5 Feature selection and Model Implementation

Now,based on the features selected, we used various models to perform regression analysis. The dataset split is 80% of the observations are train set and the remaining,test set.

Feature Scaling is performed using StandardScalar method.

SMOTE technique is applied on our dataset to remove the imbalance in the distribution.

[43]:
```
cols=['Age', 'Sex', 'ChestPainType', 'Cholesterol', 'FastingBS',
      'ExerciseAngina', 'Oldpeak', 'ST_Slope']
```

```
X=df[cols] #predictor variables
# X = df.drop('HeartDisease',axis=1)
Y = df.HeartDisease #dependant variable
print("X:",X.shape)
print("Y:",Y.shape)

sm = SMOTE(random_state=42)
X,Y = sm.fit_resample(X,Y)

#Splitting the data into train and test in to 80/20
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,␣
↪random_state=7)
print("x_train:",x_train.shape,"\ty_train:",y_train.shape)
print("\nx_test:",x_test.shape,"\ty_test:",y_test.shape)
```

```
X: (918, 8)
Y: (918,)
x_train: (812, 8)       y_train: (812,)

x_test: (204, 8)        y_test: (204,)
```

[48]:

```
# scale the data
ss = StandardScaler()
ss.fit(x_train)

X_train = ss.transform(x_train)
X_test = ss.transform(x_test)
```

6 Logistic Regression

[86]:

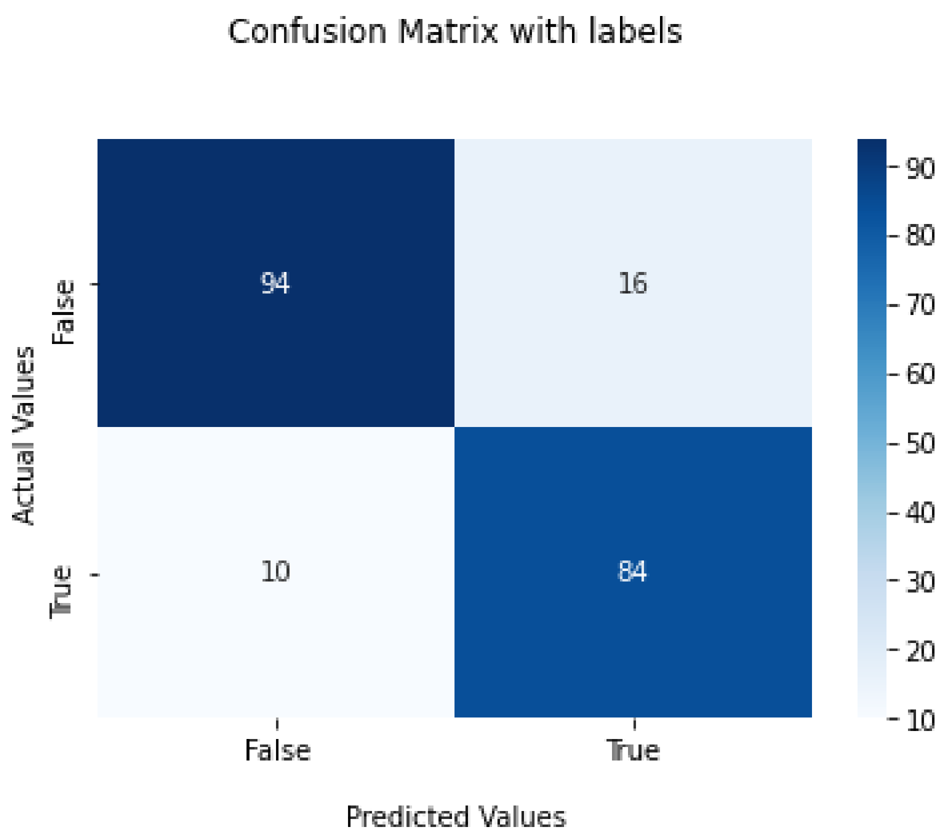
```
#Logistic Regression
lr = LogisticRegression()
lr.fit(X_train,y_train)
lr_predict = lr.predict(X_test)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)

# print(lr_conf_matrix)
lr_accscore = accuracy_score(y_test, lr_predict)

print("the accuracy of Logistic Regression is", lr_accscore)
print('F1 Score on test set:',round(f1_score(y_test, lr_predict,␣
↪average="macro"),3))
print('Precision Score:',round(precision_score(y_test, lr_predict,␣
↪average="macro"),3))
print('Recall Score:', round(recall_score(y_test, lr_predict,␣
↪average="macro"),3))
```

```python
ax = sns.heatmap(lr_conf_matrix, annot=True, cmap='Blues')
ax.set_title('Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

```
the accuracy of Logistic Regression is 0.8725490196078431
F1 Score on test set: 0.872
Precision Score: 0.872
Recall Score: 0.874
```

[86]:

```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



7 Random Forest Classifier

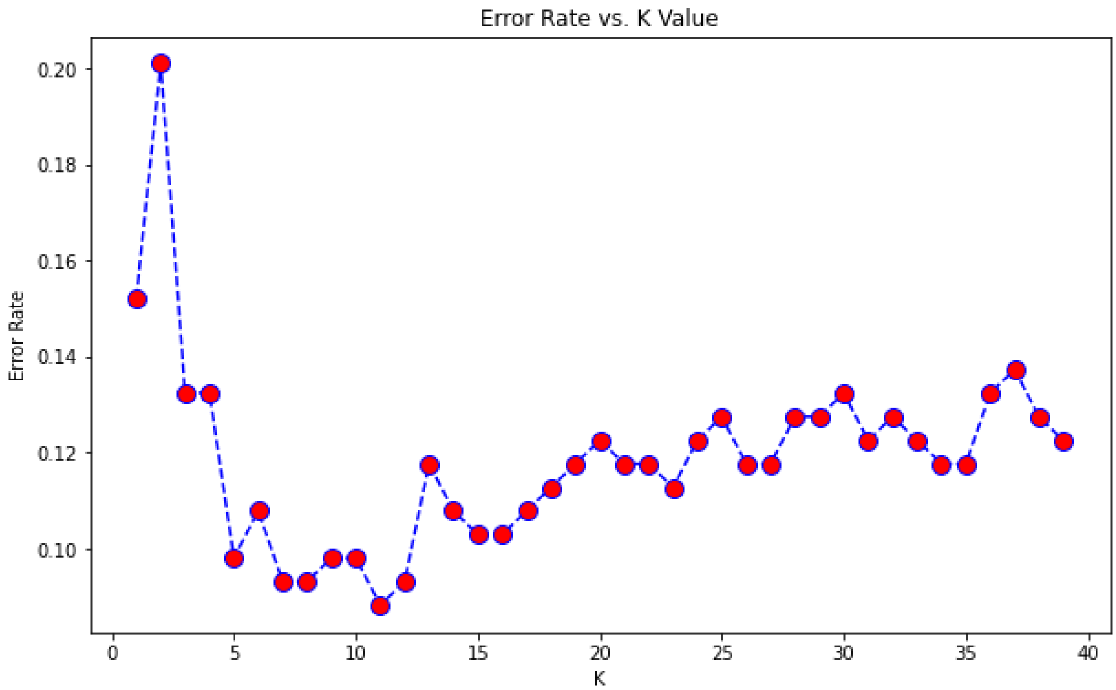
[52]:
```
m3 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=300, random_state=12,max_depth=5)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
rf_acc_score = accuracy_score(y_test, rf_predicted)

print("Accuracy of Random Forest Classifier:",round(rf_acc_score,3))
print('F1 Score of Random Forest Classifier:',round(f1_score(y_test,␣
↪rf_predicted, average="macro"),3))
print('Precision Score:',round(precision_score(y_test, rf_predicted,␣
↪average="macro"),3))
print('Recall Score:', round(recall_score(y_test, rf_predicted,␣
↪average="macro"),3))
```

```
Accuracy of Random Forest Classifier: 0.907
F1 Score of Random Forest Classifier: 0.907
Precision Score: 0.906
Recall Score: 0.908
```

8 K-Nearest Neighbors Classifier

[70]:
```
# we plot a graph of k values Vs error rate.
error_rate = []

# to find the optimal value of k
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))


plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[70]:
```
Text(0, 0.5, 'Error Rate')
```



With this graph, we see that the optimal value of k where the error rate is min is 11. hence we make use of this value.

[56]:
```
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, y_train)
knn_predicted = knn.predict(X_test)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_acc_score = accuracy_score(y_test, knn_predicted)

print("Accuracy of K-NeighborsClassifier:",round(knn_acc_score,3))
print('F1 Score of K-NeighborsClassifier:',round(f1_score(y_test,␣
 ↪knn_predicted, average="macro"),3))
print('Precision Score:',round(precision_score(y_test, knn_predicted,␣
 ↪average="macro"),3))
print('Recall Score:', round(recall_score(y_test, knn_predicted,␣
 ↪average="macro"),3))
```

```
Accuracy of K-NeighborsClassifier: 0.912
F1 Score of K-NeighborsClassifier: 0.911
Precision Score: 0.911
Recall Score: 0.911
```

9 Support Vector Classifier

[65]:
```
#Support Vector Classifier
classifier_rbf = SVC(kernel = 'rbf')
classifier_rbf.fit(X_train, y_train)
y_pred = classifier_rbf.predict(X_test)
print('Accuracy of SVC (RBF) classifier : ',round(accuracy_score(y_test,␣
 ↪y_pred),3))
print('F1 Score of SVC (RBF) classifier :',round(f1_score(y_test, y_pred,␣
 ↪average="macro"),3))
print('Precision Score:',round(precision_score(y_test, y_pred,␣
 ↪average="macro"),2))
print('Recall Score:',round(recall_score(y_test, y_pred, average="macro"),2))
```

```
Accuracy of SVC (RBF) classifier :  0.902
F1 Score of SVC (RBF) classifier : 0.902
Precision Score: 0.9
Recall Score: 0.9
```

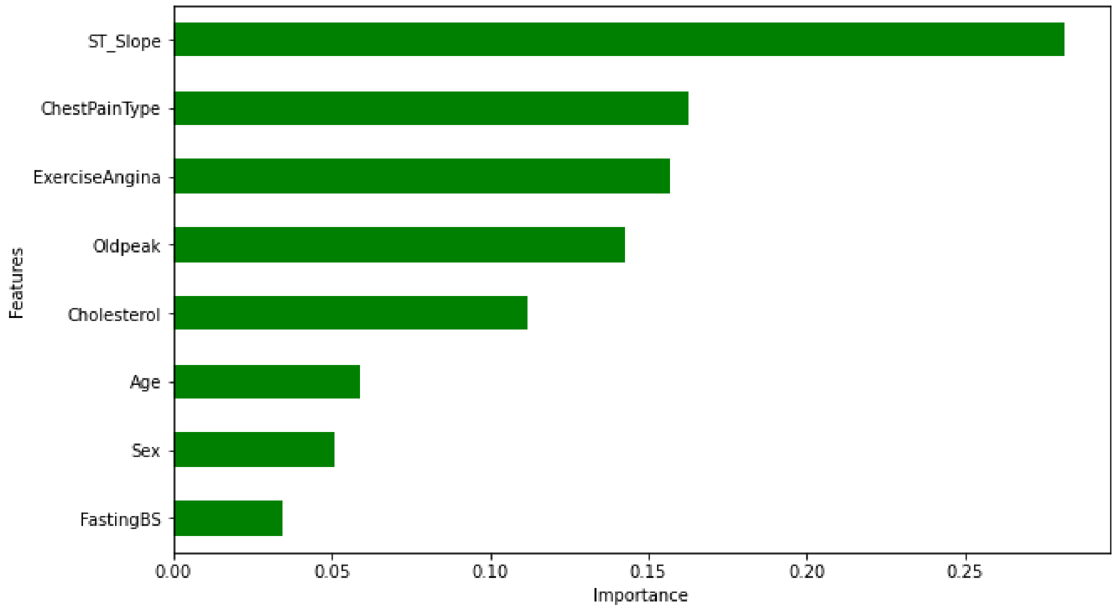
Among all the models analysed,we see that the KNN classifier yields the best results. We see highest accuracy of 91% among others such as SVC and Random Forest classifier models. The Precision score is also 91%, implying our model correctly classified observations with high risk in the high risk category 75% of the times. The Recall score is 0.75.

Conclusion: Thus, we choose KNN Classifier as the prefered model due to high accuracy,recall and precision score.

10 Importance of the features of their contribution to the model

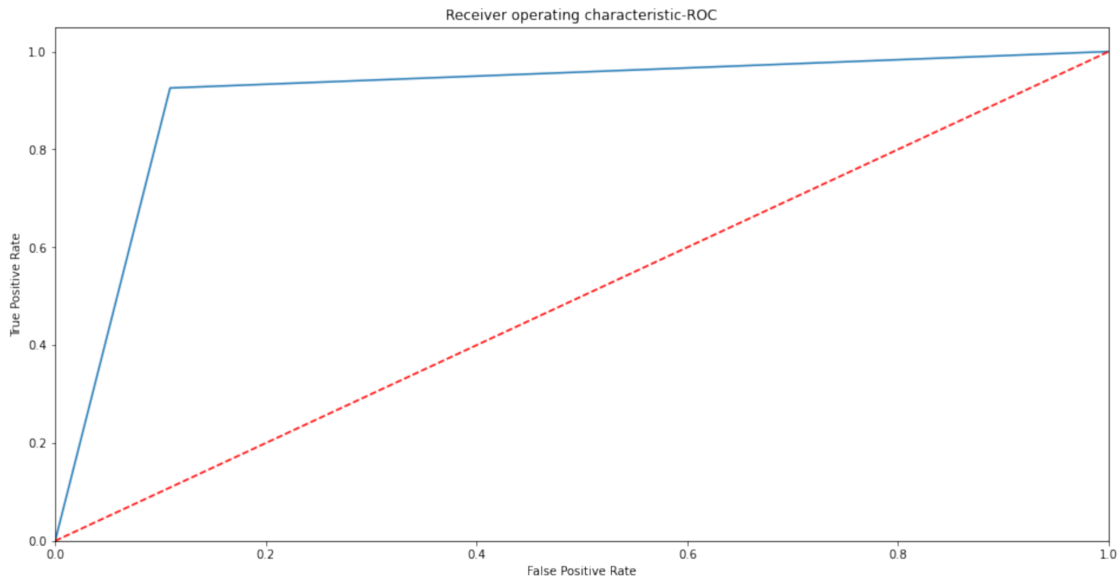
[67]:
```
#The figure  shows the relative importance of features .

from matplotlib import pyplot

rf.fit(x_train, y_train)

coeff = list(rf.feature_importances_)
labels = list(X.columns)

features = pd.DataFrame()
features['Features'] = labels
features['importance'] = coeff
features.sort_values(by=['importance'], ascending=True, inplace=True)
features['positive'] = features['importance'] > 0
features.set_index('Features', inplace=True)
features.importance.plot(kind='barh', figsize=(10,6),color = features.positive.
 ↪map({True: 'green', False: 'red'}))
plt.xlabel('Importance')
```

[67]: Text(0.5, 0, 'Importance')



11 ROC Curve and Area under the Curve

[275]:
```
logit_roc_auc = roc_auc_score(y_test, knn.predict(X_test))
# fpr, tpr, thresholds = roc_curve(y_test, rfc.predict_proba(x_test)[:,1])

fpr,tpr,rf_threshold = roc_curve(y_test,rf_predicted)

plt.figure(figsize=(16, 8))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic-ROC')
plt.savefig('Log_ROC')
plt.show()
```



[278]:
```
# AUC curve
print("Area under AUC curve", metrics.roc_auc_score(y_test, knn_predicted))
```

```
Area under AUC curve 0.9112185686653772
```