AIMD Hospital Liver Disease Prediction

In [1]:

```
# Import required libs
import seaborn as sb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# Read liver disease data-set
aimd = pd.read_csv("liver_disease_1.csv")
```

In [3]:

```
# Check some information related to the imported data-set
aimd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Total_Bilirubin             583 non-null    float64
 2   Direct_Bilirubin            583 non-null    float64
 3   Alkaline_Phosphotase        583 non-null    int64  
 4   Alamine_Aminotransferase    583 non-null    int64  
 5   Aspartate_Aminotransferase  583 non-null    int64  
 6   Total_Protiens              583 non-null    float64
 7   Albumin                     583 non-null    float64
 8   Albumin_and_Globulin_Ratio  579 non-null    float64
 9   Dataset                     583 non-null    object 
dtypes: float64(5), int64(4), object(1)
memory usage: 45.7+ KB
```

In [ ]:

```

```

Exploring the data-set (EDA) with plots and stats

In [4]:

```
# Check column set
print(list(aimd.columns))
```

```
['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase', 'Al
amine_Aminotransferase', 'Aspartate_Aminotransferase', 'Total_Protiens',
'Albumin', 'Albumin_and_Globulin_Ratio', 'Dataset']
```

In [5]:

```
# Check data-types
aimd.dtypes
```

Out[5]:

```
Age                           int64
Total_Bilirubin             float64
Direct_Bilirubin            float64
Alkaline_Phosphotase          int64
Alamine_Aminotransferase      int64
Aspartate_Aminotransferase    int64
Total_Protiens              float64
Albumin                     float64
Albumin_and_Globulin_Ratio  float64
Dataset                      object
dtype: object
```

In [6]:

```
# Check some samples of the data
aimd.head(3)
```

Out[6]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	As
0	65	0.7	0.1	187	16	
1	62	10.9	5.5	699	64	
2	62	7.3	4.1	490	60	

In [7]:

```
# Check and drop duplicates
aimd[aimd.duplicated()]
```

Out[7]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
19	40	0.9	0.3	293	232
26	34	4.1	2.0	289	875
34	38	2.6	1.2	410	59
55	42	8.9	4.5	272	31
62	58	1.0	0.5	158	37
106	36	5.3	2.3	145	32
108	36	0.8	0.2	158	29
138	18	0.8	0.2	282	72
143	30	1.6	0.4	332	84
158	72	0.7	0.1	196	20
164	39	1.9	0.9	180	42
174	31	0.6	0.1	175	48
201	49	0.6	0.1	218	50

In [8]:

```
# Drop the 13 duplicates found and reset the index
aimd.drop_duplicates(inplace=True, keep='last', ignore_index=True)
aimd.duplicated().sum()
```

Out[8]:

```
0
```

In [9]:

```
# Check some information about data
print("""DataFrame Dimensions = {0}
Dataframe Shape = {1}""".format(aimd.ndim, aimd.shape))
```

```
DataFrame Dimensions = 2
Dataframe Shape = (570, 10)
```

In [10]:

```
# Check some relevant stats about the data
aimd.describe(percentiles=[.05,.25,.5,.75,.99]).T
# The data looks neatly distributed
```

Out[10]:

	count	mean	std	min	5%	25%	50%	75
Age	570.0	44.849123	16.242182	4.0	18.0	33.0	45.00	58.(
Total_Bilirubin	570.0	3.321754	6.267941	0.4	0.6	0.8	1.00	2.(
Direct_Bilirubin	570.0	1.497544	2.833231	0.1	0.1	0.2	0.30	1.3
Alkaline_Phosphotase	570.0	291.750877	245.291859	63.0	135.9	176.0	208.00	298.(
Alamine_Aminotransferase	570.0	79.728070	181.471697	10.0	15.0	23.0	35.00	60.(
Aspartate_Aminotransferase	570.0	109.380702	290.880671	10.0	15.0	25.0	41.00	86.7
Total_Protiens	570.0	6.496316	1.088300	2.7	4.6	5.8	6.60	7.2
Albumin	570.0	3.148947	0.796813	0.9	1.8	2.6	3.10	3.8
Albumin_and_Globulin_Ratio	566.0	0.948004	0.319635	0.3	0.5	0.7	0.95	1.1

In [11]:

```
# Rename columns to short values
aimd = aimd.rename(columns={'Total_Bilirubin':'Bilirubin', 'Direct_Bilirubin':'BilirubinDir',
                            'Total_Protiens':'Protiens', 'Alkaline_Phosphotase':'Phosphotase',
                            'Alamine_Aminotransferase':'AlamineAT', 'Aspartate_Aminotransferase':'AspartateAT',
                            'Albumin_and_Globulin_Ratio':'AGRatio', 'Dataset':'Disease'
})
```

In [12]:

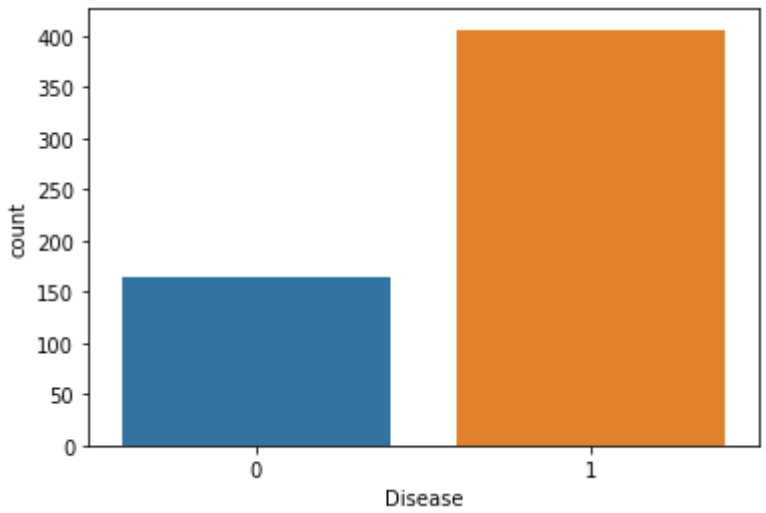
```
# Check that Disease is either Yes/No and convert to Integer 0/1
print(aimd.Disease.unique())
#aimd.Disease = aimd.Disease.astype('string', copy=False)
aimd['Disease'] = aimd['Disease'].map({'Yes': 1, 'No': 0})
print(aimd.Disease.unique())
```

```
['Yes' 'No']
[1 0]
```

In [13]:

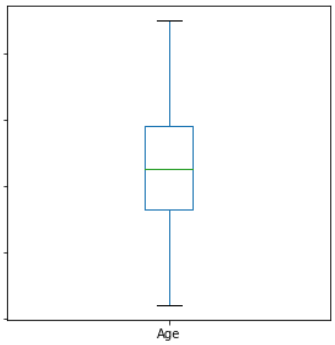
```
# Check class balance
print(aimd.Disease.value_counts())
sb.countplot(aimd.Disease)
plt.show()
# Class is roughly 70/30 split and needs to be stratified when sampling/splitting
```

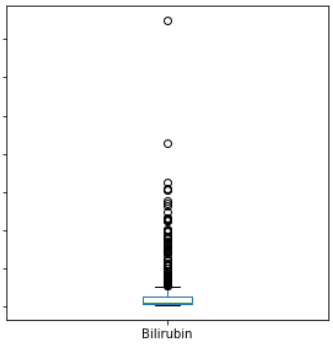
```
1    406
0    164
Name: Disease, dtype: int64
```

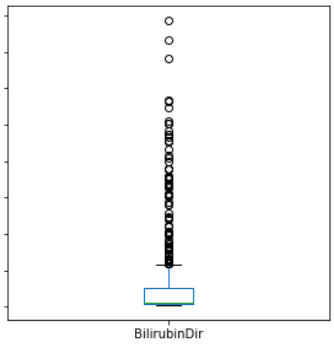


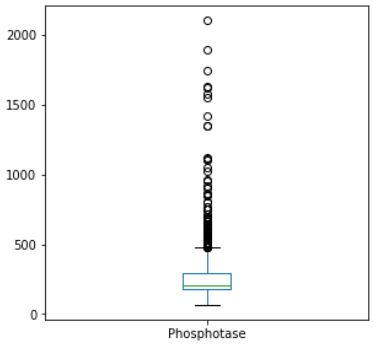
In [14]:

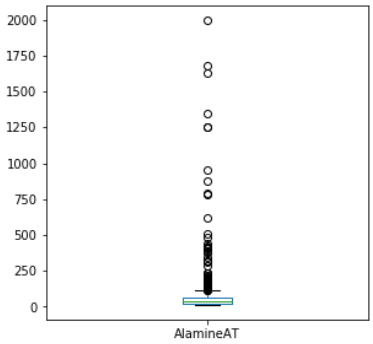
```
# Check the outliers
aimd.drop('Disease', axis=1).plot(kind='box', subplots=True, layout=(3,3), figsize=(16,
16))
plt.show()
```

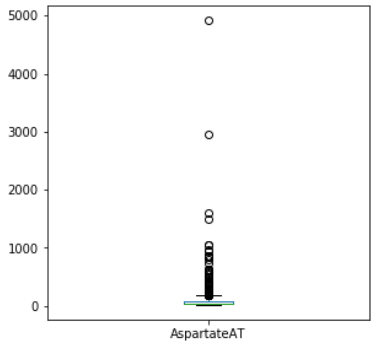

Age

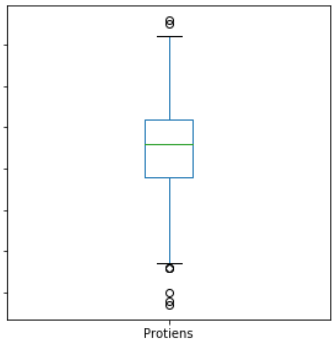

Bilirubin

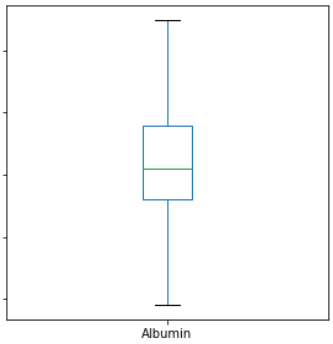

BilirubinDir

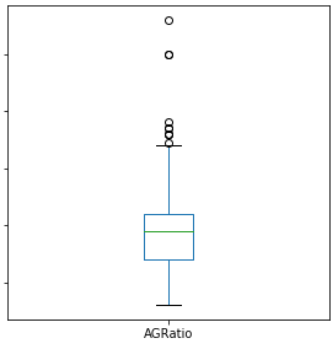

Phosphotase


AlamineAT


AspartateAT

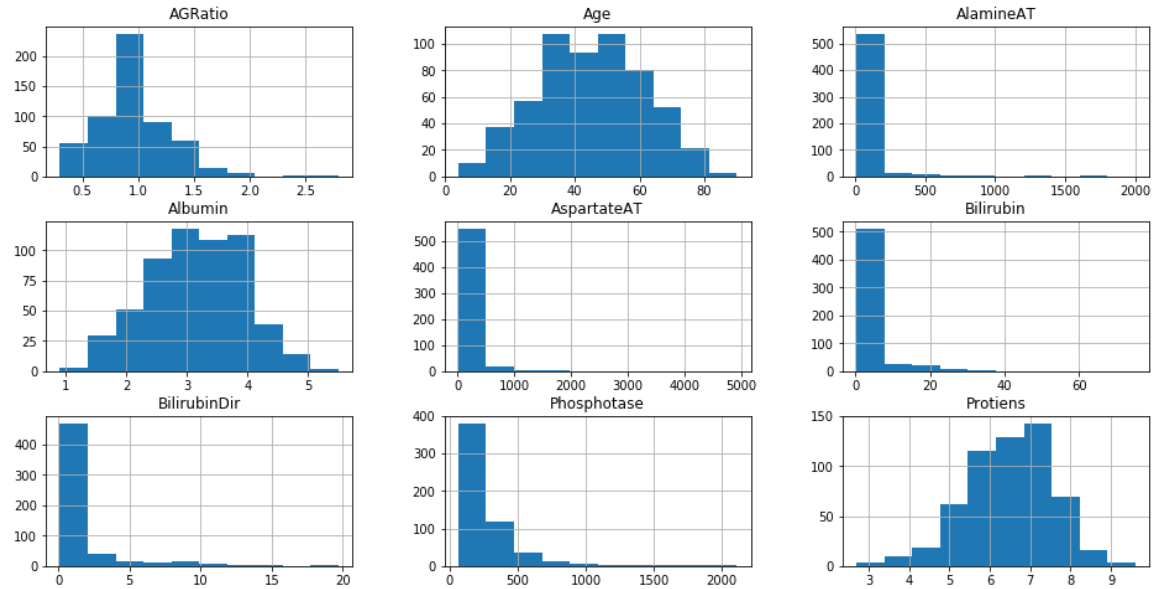

Protiens


Albumin


AGRatio

In [15]:

```
# Plot the distributions
aimd.drop(['Disease'], axis=1).hist(figsize=[16,8], bins=10, layout=(3,3))
plt.show()
```



In [16]:

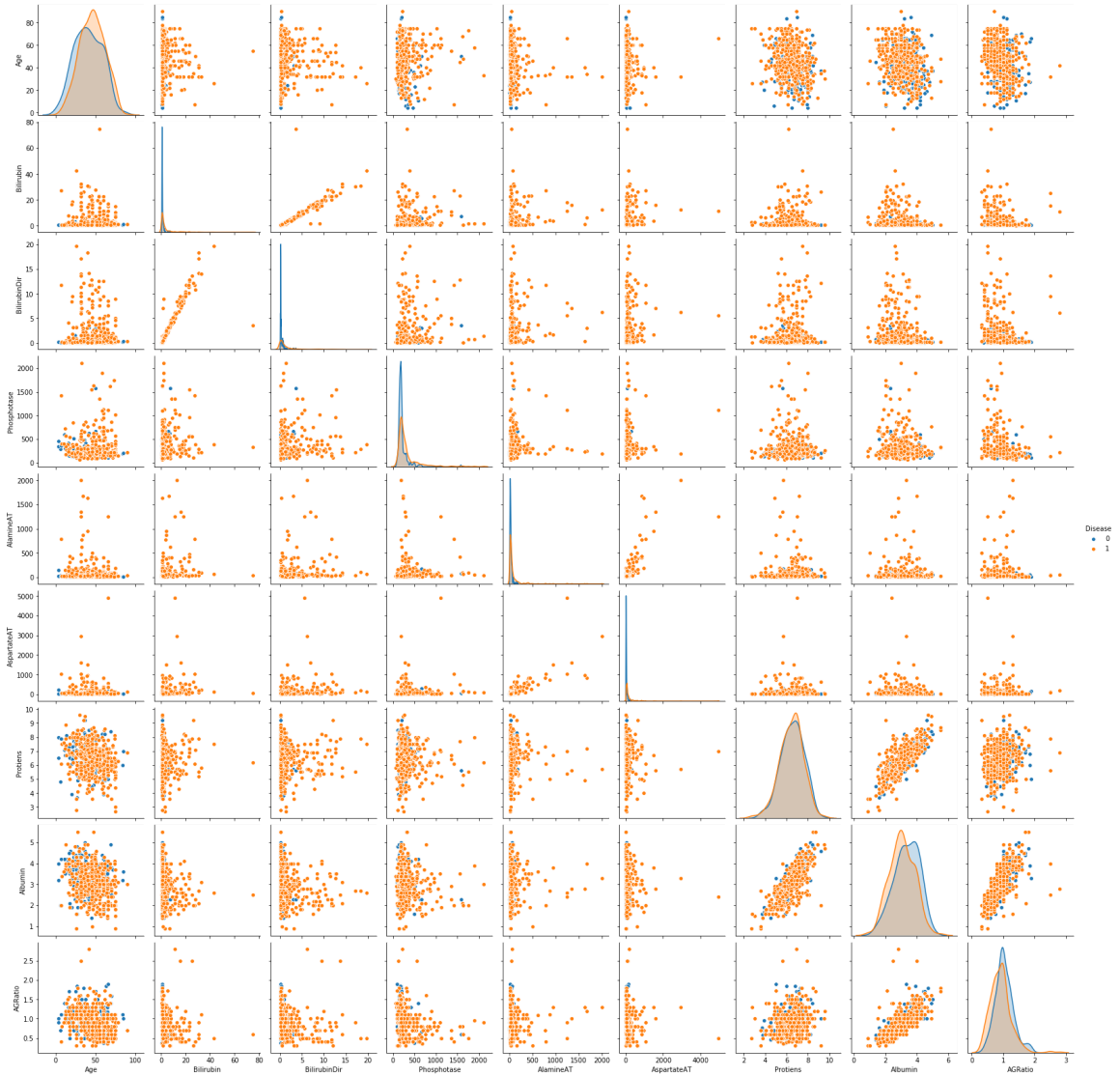
```
# Check null values
aimd.isnull().sum()
```

Out[16]:

```
Age             0
Bilirubin       0
BilirubinDir    0
Phosphotase     0
AlamineAT       0
AspartateAT     0
Protiens        0
Albumin         0
AGRatio         4
Disease         0
dtype: int64
```

In [17]:

```
# Scatter and pair plots of data
sb.pairplot(aimd, diag_kind='kde', hue='Disease')
plt.show()
```



In [18]:

```
# AGRatio is right-skewed, impute using median
from sklearn.impute import SimpleImputer
aimd['AGRatio'] = SimpleImputer(strategy='median', copy=False).fit(aimd[['AGRatio']]).t
ransform(aimd[['AGRatio']])
aimd.isna().sum()
# OR
#aimd.loc[aimd[aimd['AGRatio'].isnull()].index, 'AGRatio'] = aimd['AGRatio'].median()
```

Out[18]:

```
Age             0
Bilirubin       0
BilirubinDir    0
Phosphotase     0
AlamineAT       0
AspartateAT     0
Protiens        0
Albumin         0
AGRatio         0
Disease         0
dtype: int64
```

In [19]:

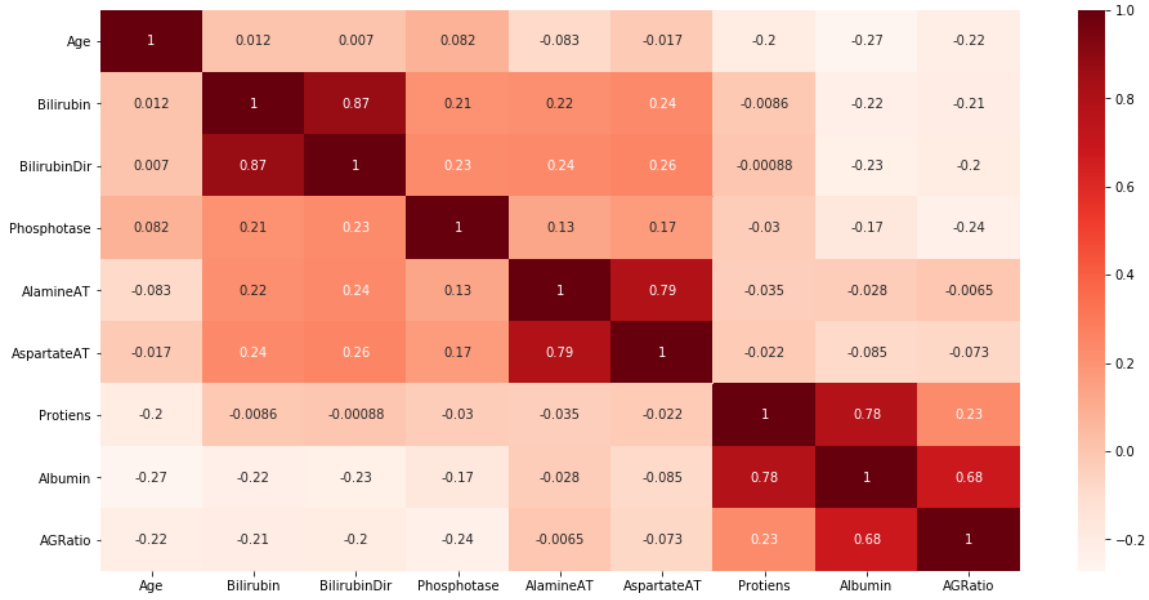
```
# Check the correlation matrix (exclude Disease)
corr = aimd.iloc[:,:-1].corr()
corr
```

Out[19]:

	Age	Bilirubin	BilirubinDir	Phosphotase	AlamineAT	AspartateAT	Prot
Age	1.000000	0.011500	0.007050	0.081673	-0.083383	-0.016753	-0.19
Bilirubin	0.011500	1.000000	0.874116	0.206239	0.217471	0.238678	-0.00
BilirubinDir	0.007050	0.874116	1.000000	0.234609	0.237450	0.258489	-0.00
Phosphotase	0.081673	0.206239	0.234609	1.000000	0.126830	0.167230	-0.03
AlamineAT	-0.083383	0.217471	0.237450	0.126830	1.000000	0.791857	-0.03
AspartateAT	-0.016753	0.238678	0.258489	0.167230	0.791857	1.000000	-0.02
Protiens	-0.197052	-0.008588	-0.000875	-0.030048	-0.035193	-0.022000	1.00
Albumin	-0.271170	-0.224124	-0.230751	-0.168318	-0.027973	-0.085180	0.78
AGRatio	-0.215654	-0.207646	-0.201412	-0.236058	-0.006537	-0.072893	0.23

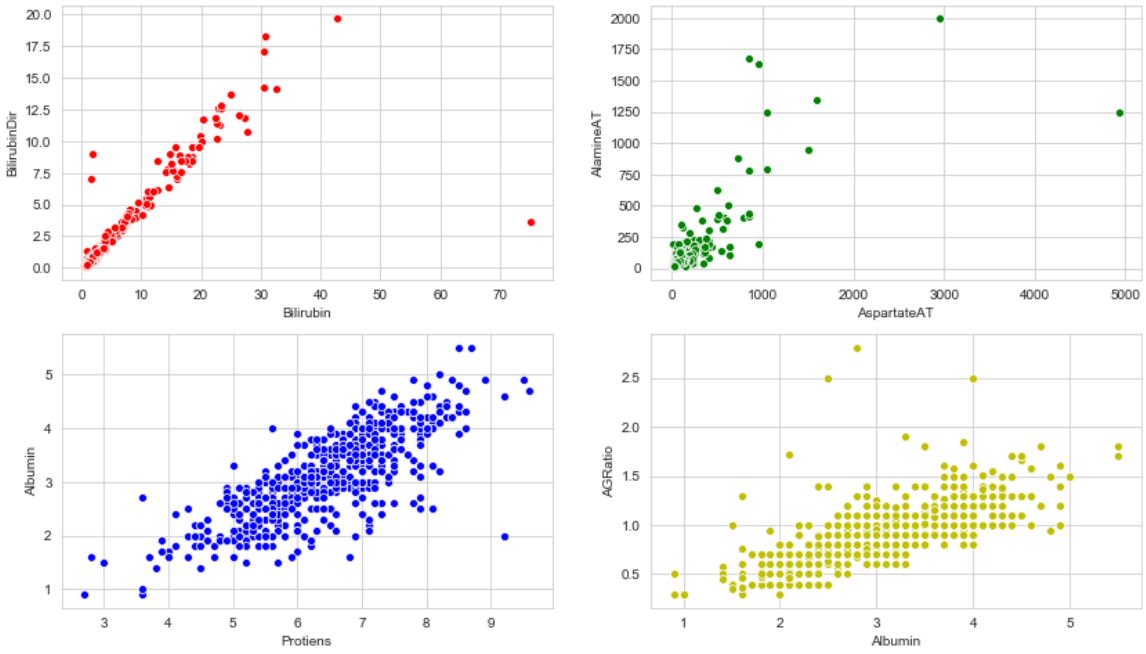
In [20]:

```
# Check multivariate correlations
plt.figure(figsize=(16,8))
sb.heatmap(corr, cmap='Reds', annot=True)
plt.show()
```



In [21]:

```
# Plot correlations for >65% above using multi-variate plots
sb.set_style('whitegrid')
plt.figure(figsize=[14,8])
plt.subplot(2,2,1)
sb.scatterplot(x=aimd.Bilirubin, y=aimd.BilirubinDir, color='r')
plt.subplot(2,2,2)
sb.scatterplot(x=aimd.AspartateAT, y=aimd.AlamineAT, color='g')
plt.subplot(2,2,3)
sb.scatterplot(x=aimd.Protiens, y=aimd.Albumin, color='b')
plt.subplot(2,2,4)
sb.scatterplot(x=aimd.Albumin, y=aimd.AGRatio, color='y')
plt.show()
```



In [22]:

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Calculate VIF scores
X = aimd.drop(['Disease'], axis=1)
vif = pd.DataFrame([variance_inflation_factor(X.values, i) for i in range(X.shape[1])],
columns=['VIF Score'], index=X.columns)
round(vif, 2).sort_values(by='VIF Score').T
# VIF scores are higher since scaling has not been performed, will check VIF again afte
r scaling and dropping features
```

Out[22]:

	Phosphotase	AspartateAT	AlamineAT	Bilirubin	BilirubinDir	Age	AGRatio	Protiens
VIF Score	2.63	3.18	3.29	5.45	5.72	7.59	24.4	96.88

In [23]:

```
# Recursive Feature Elimination
#from sklearn.feature_selection import RFE
#from sklearn.linear_model import LogisticRegression

#y=['Disease']
#X=[i for i in aimd.columns if i!='Disease']
##X=['Age', 'Bilirubin', 'Phosphotase', 'AlamineAT', 'Protiens', 'AGRatio']
#print(X)
#rfe = RFE(LogisticRegression(), 6).fit(aimd[X], aimd[y])
#print(rfe.support_)
#print(rfe.ranking_)
```

In [24]:

```
# Keep a copy for sklearn
aimd_orig = aimd.copy()
# Normalize features to have the models (GradientDescent and GaussianNB) work properly
X = ['Age', 'Phosphotase', 'Bilirubin', 'AlamineAT', 'Protiens', 'AGRatio', 'BilirubinD
ir', 'Albumin', 'AspartateAT']
aimd[X] = (aimd[X] - aimd[X].mean())/aimd[X].std()
display(aimd.head(3), aimd.tail(2))
```

	Age	Bilirubin	BilirubinDir	Phosphotase	AlamineAT	AspartateAT	Protiens	Album
0	1.240651	-0.418280	-0.493269	-0.427046	-0.351174	-0.314152	0.279044	0.18957
1	1.055947	1.209049	1.412682	1.660264	-0.086670	-0.032249	0.922249	0.06407
2	1.055947	0.634697	0.918547	0.808217	-0.108712	-0.142260	0.462817	0.18957

	Age	Bilirubin	BilirubinDir	Phosphotase	AlamineAT	AspartateAT	Protiens	Alb
568	-0.852664	-0.322555	-0.352087	-0.439276	-0.279537	-0.266022	0.279044	0.31
569	-0.421687	-0.370417	-0.422678	-0.308819	-0.323621	-0.293525	0.738476	1.57

In [25]:

```
# Build a model and check if VIF matters for the dataset
import statsmodels.api as sm
sm.Logit(aimd['Disease'].values, aimd.drop(['Disease'], axis=1).values).fit(disp=False)
.summary()
# Many p-values are >0.05 and are statistically insignificant, drop the correlated feat
ures and test again
```

Out[25]:

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	570
Model:	Logit	**Df Residuals:**	561
Method:	MLE	**Df Model:**	8
Date:	Mon, 13 Jul 2020	**Pseudo R-squ.:**	-0.06444
Time:	18:42:58	**Log-Likelihood:**	-364.10
converged:	True	**LL-Null:**	-342.06
Covariance Type:	nonrobust	**LLR p-value:**	1.000

	coef	std err	z	P>|z|	[0.025	0.975]
x1	0.2233	0.093	2.413	0.016	0.042	0.405
x2	0.0065	0.184	0.035	0.972	-0.355	0.368
x3	0.3317	0.201	1.654	0.098	-0.061	0.725
x4	0.2227	0.109	2.047	0.041	0.009	0.436
x5	0.3176	0.248	1.282	0.200	-0.168	0.803
x6	0.0294	0.299	0.098	0.922	-0.556	0.615
x7	0.3838	0.240	1.596	0.110	-0.087	0.855
x8	-0.5197	0.331	-1.569	0.117	-1.169	0.130
x9	0.0973	0.201	0.484	0.629	-0.297	0.492

In [26]:

```
# Dropping all correlated variables:
#1. BilirubinDir is heavily correlated with BilirubinTot
#2. Albumin is correlated highly with both Protiens and AGRatio
#3. AspartateAT is strongly correlated with AlamineAT
aimd.drop(['BilirubinDir', 'Albumin', 'AspartateAT'], inplace=True, axis=1)
```

In [27]:

```
# VIF after scaling and feature reduction
X = aimd.drop(['Disease'], axis=1)
vif = pd.DataFrame([variance_inflation_factor(X.values, i) for i in range(X.shape[1])],
columns=['VIF Score'], index=X.columns)
round(vif, 2).sort_values(by='VIF Score').T
# All values are in a good range without significant collinearity
```

Out[27]:

	AlamineAT	Age	Protiens	Phosphotase	Bilirubin	AGRatio
VIF Score	1.07	1.09	1.09	1.1	1.12	1.19

In [28]:

```
# Build another model and check the summary
sm.GLM(aimd['Disease'].values, aimd.drop(['Disease'], axis=1).values,
       family=sm.families.Binomial()).fit().summary()
# Now most of them are significant except Protiens and AGRatio, keep them as they don't
matter much and features are less anyway
```

Out[28]:

Generalized Linear Model Regression Results

Dep. Variable:	y	**No. Observations:**	570
Model:	GLM	**Df Residuals:**	564
Model Family:	Binomial	**Df Model:**	5
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-366.98
Date:	Mon, 13 Jul 2020	**Deviance:**	733.97
Time:	18:42:58	**Pearson chi2:**	556.
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
x1	0.2360	0.092	2.567	0.010	0.056	0.416
x2	0.3976	0.129	3.084	0.002	0.145	0.650
x3	0.2343	0.109	2.154	0.031	0.021	0.448
x4	0.3287	0.146	2.244	0.025	0.042	0.616
x5	0.0350	0.092	0.380	0.704	-0.145	0.216
x6	-0.1654	0.096	-1.716	0.086	-0.354	0.024

In [ ]:

```

```

Modelling using SKLearn

In [29]:

```
# Import relevant sklearn libs
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cr
oss_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1
_score, roc_curve, roc_auc_score, auc
```

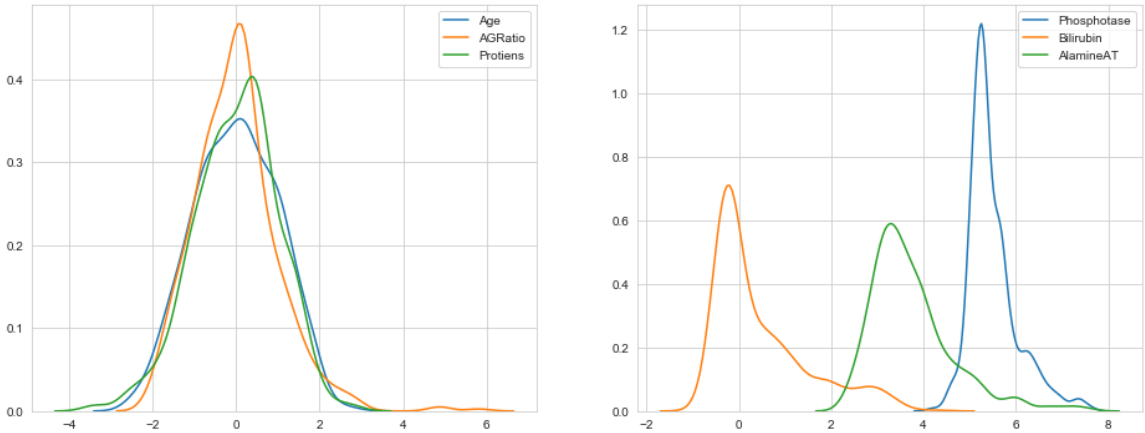
In [30]:

```
random_state=0
# Cross-Validation params
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=random_state)
target = 'Disease'
# Not considering the insignificant and correlated ones
features = ['Age','Phosphotase','Bilirubin','AlamineAT','Protiens','AGRatio']
```

Naíve-Bayes Classification

In [31]:

```
# Some of the usuable un-correlated features are not Gaussian and slightly skewed, twea
k those features to look Gaussian
# Check the feature densities after tweaking, they look more or less Gaussian now
plt.figure(figsize=[16,6])
plt.subplot(121)
sb.kdeplot(aimd.Age)
sb.kdeplot(aimd.AGRatio)
sb.kdeplot(aimd.Protiens)
plt.subplot(122)
sb.kdeplot(np.log(aimd_orig.Phosphotase))
sb.kdeplot(np.log(aimd_orig.Bilirubin))
sb.kdeplot(np.log(aimd_orig.AlamineAT))
plt.show()
```



In [32]:

```
# Prepare parameters for modelling
aimd_nb = aimd.copy()
aimd_nb.AlamineAT = np.log(aimd_orig.AlamineAT + 1/3)
aimd_nb.Bilirubin = np.log(aimd_orig.Bilirubin + 0.1)
aimd_nb.Phosphotase = np.log(aimd_orig.Phosphotase + .125)
#aimd_nb = aimd_nb.fillna(0)
X = aimd_nb.loc[:, features]
y = aimd_nb[target]
X_train, X_test, y_train, y_test1 = train_test_split(X, y, train_size=2/3, test_size=1/
3, stratify=y, random_state=random_state)

# Create the GaussianNB model
model_nb = GaussianNB()
model_nb.fit(X_train, y_train, sample_weight=None)
y_pred_nb = model_nb.predict(X_test)
y_pred_nb_prob = model_nb.predict_proba(X_test)[:,1]
print('Prediction Accuracy:', round(accuracy_score(y_test1, y_pred_nb), 3))

# CrossValidation
print("Average Cross-Validation Score: %.3f" % cross_val_score(model_nb, X, y, cv=cv).m
ean())
```

```
Prediction Accuracy: 0.626
Average Cross-Validation Score: 0.649
```

Logistic Regression #1

In [33]:

```
# Gather data and take all the features for modelling
X = aimd_orig.drop(['Disease'], axis=1)
y = aimd_orig['Disease']

# Normalize the data
X = StandardScaler(copy=False).fit(X).transform(X)

# Split to train and test
X_train, X_test, y_train, y_test2 = train_test_split(X, y, train_size=2/3, test_size=1/
3, stratify=y, random_state=random_state)

# CrossValidation and GridSearch
best_model = GridSearchCV(LogisticRegression(), param_grid={'C':[x for x in range(1,7
)], 'solver':['lbfgs','newton-cg']}, cv=cv)
best_model.fit(X,y)
print("Best Score: {:.3f}\nBest Params: {}".format(best_model.best_score_, best_model.b
est_params_))
```

```
Best Score: 0.730
Best Params: {'C': 6, 'solver': 'lbfgs'}
```

In [34]:

```
# Fit LogisticRegression as per above params and  perform cross-validation
model_logit1 = LogisticRegression(penalty='l2', tol=0.00001, C=6., solver='lbfgs', max_
iter=1000)
model_logit1.fit(X_train, y_train)
y_pred_logit1 = model_logit1.predict(X_test)
y_pred_logit1_prob = model_logit1.predict_proba(X_test)[:,1]
print('Prediction Accuracy:', round(accuracy_score(y_test2, y_pred_logit1), 3))
```

```
Prediction Accuracy: 0.705
```

K-Nearest Neighbours

In [35]:

```
# Check best K using sklearn
best_model = GridSearchCV(KNeighborsClassifier(), param_grid = {'n_neighbors':range(1,2
4), 'p':[1,2]}, cv = cv)
best_model.fit(X,y)
print("Best Score: {:.3f}\nBest Params: {}".format(best_model.best_score_, best_model.b
est_params_))

# K=23 presents a good accuracy trade-off and can be used to model with KNN
model_knn = KNeighborsClassifier(n_neighbors=23, p=1, weights='distance')
model_knn.fit(X_train, y_train)
y_pred_knn = model_knn.predict(X_test)
y_pred_knn_prob = model_knn.predict_proba(X_test)[:,1]
print('Prediction Accuracy:', round(accuracy_score(y_test2, y_pred_knn), 3))
```

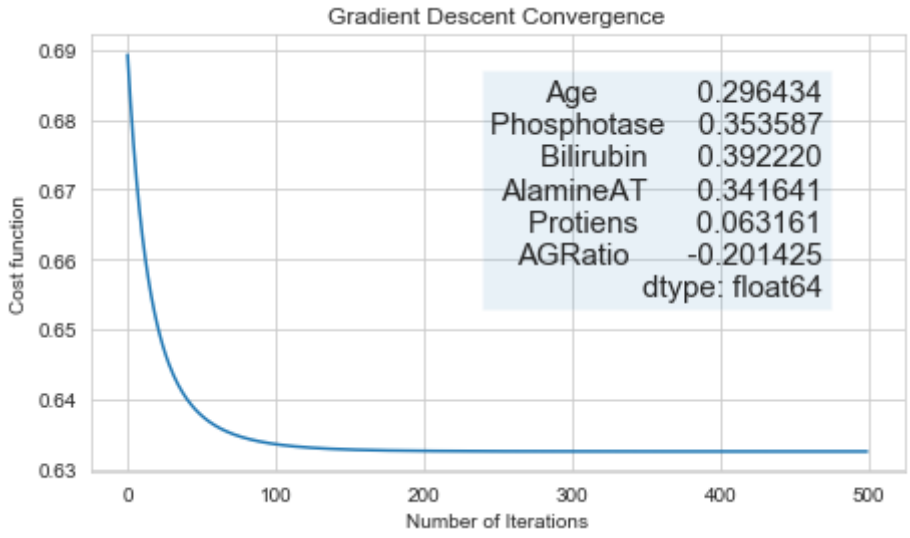
```
Best Score: 0.700
Best Params: {'n_neighbors': 23, 'p': 1}
Prediction Accuracy: 0.705
```

Logistic Regression #2

In [36]:

```
# Get the data
X = aimd.loc[:, features]
y = aimd[target]
X_train, X_test, y_train, y_test3 = train_test_split(X, y, train_size=2/3, test_size=1/
3, random_state=random_state)

# Sigmoid function
def sigmoid(Z):
    return 1/(1 + np.exp(-Z))

# Cost function
def costFunc(theta, X, y):
    return ((-1/y.size)* (y.T.dot(np.log(sigmoid(X.dot(theta)))) + (1-y).T.dot(np.log(1
-sigmoid(X.dot(theta))))))

# Gradient function
def gradient(theta, X, y):
    return ((X.T).dot(sigmoid(X.dot(theta)) - y))/y.size

# Gradient Descent for Optimization
def gradientDescent(theta, X, y, alpha=0.1, iters=500):
    samples = y.size
    costHist = []
    for i in range(iters):
        delta = gradient(theta, X, y)
        theta -= alpha * delta
        costHist.append(costFunc(theta,X,y))
    return theta, costHist

# Use advanced optimization techniques to validate the theta values
from scipy.optimize import minimize
theta_adv = minimize(fun=costFunc, x0=np.ones(X.shape[1]), args=(X_train, y_train.value
s),
                     method='Newton-CG', jac=gradient, options={'maxiter':500, 'disp':T
rue}).x

# Use the code to get the parameters at gradient convergence
theta, costHist = gradientDescent(np.zeros(X.shape[1]), X_train, y_train.values)
# Check that the thetas almost match
print(theta_adv)

# Plot Gradient Descent Convergence
plt.figure(figsize=(16,4))
sb.set_style("whitegrid")
ax = plt.subplot(1,2,1)
plt.plot(np.arange(len(costHist)),costHist)
plt.title('Gradient Descent Convergence')
plt.xlabel('Number of Iterations')
plt.ylabel('Cost function')
plt.text(0.9, 0.9, '{}'.format(theta),
         fontsize=15, horizontalalignment='right', verticalalignment='top', bbox=dict(a
lpha=0.1), transform=ax.transAxes)
plt.show()

# Predictions and results (setting threshold at 30% prob for better results)
y_pred_logit2_prob = sigmoid(X_test.dot(theta_adv))
y_pred_logit2 = pd.Series(y_pred_logit2_prob).map(lambda x: 0 if x<.3 else 1)
```

```
Optimization terminated successfully.
         Current function value: 0.632533
         Iterations: 6
         Function evaluations: 8
         Gradient evaluations: 47
         Hessian evaluations: 0
Age             0.296776
Phosphotase     0.353314
Bilirubin       0.393465
AlamineAT       0.342239
Protiens        0.063241
AGRatio        -0.201064
dtype: float64
```

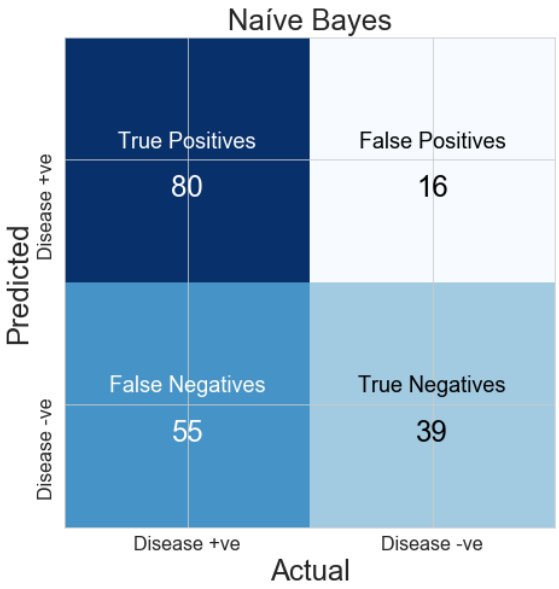


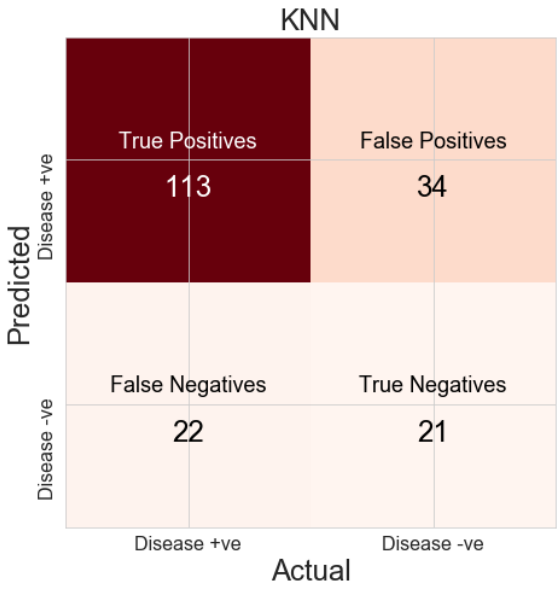
In [ ]:

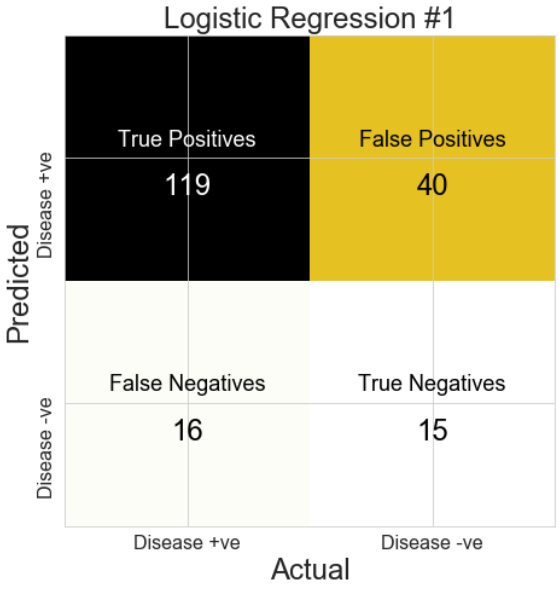
Evaluate and Compare Model Performance

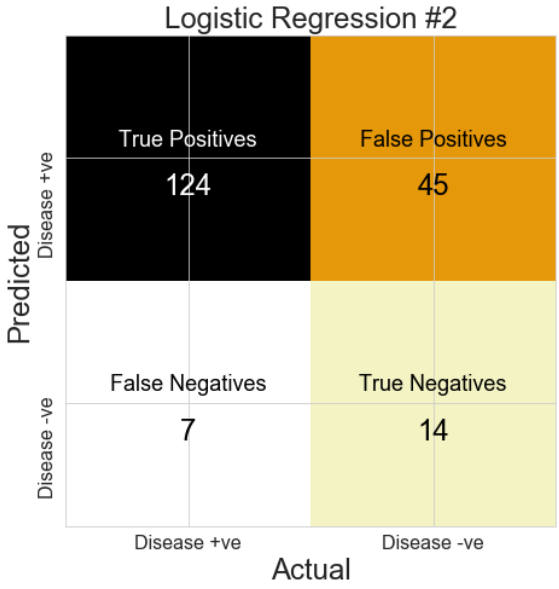
In [37]:

```
# Function to print confusion matrix with TP, FP, TN, FN
from itertools import product
def cm_axes(y_test, y_pred, title, cmap):
    conf_mat = np.rot90(confusion_matrix(y_test, y_pred), 2).T
    classes=['Disease +ve','Disease -ve']
    plt.xlabel('Actual', size = 24)
    plt.ylabel('Predicted', size = 24)
    plt.title(title, size = 24)
    plt.imshow(conf_mat, interpolation='nearest', cmap=cmap)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 0, size = 16)
    plt.yticks(tick_marks, classes, rotation = 90, size = 16)
    rows = [['True Positives', 'False Positives'],
            ['False Negatives', 'True Negatives']]
    for i, j in product(range(conf_mat.shape[0]), range(conf_mat.shape[1])):
        plt.text(j, i - 0.05, format(rows[i][j]),
        horizontalalignment="center", size = 18,
        color="white" if conf_mat[i, j] > conf_mat.max() / 2. else "black")
        plt.text(j, i + 0.15, format(conf_mat[i, j], ''),
        horizontalalignment="center", size = 24,
        color="white" if conf_mat[i, j] > conf_mat.max() / 2. else "black")
    return plt

# Draw and check the heat-map for all models to find TP, TN, FP and FN
#sb.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', xticklabels=classes,
yticklabels=classes)
plt.figure(figsize=(16,8))
plt.subplot(121)
cm_axes(y_test1, y_pred_nb, 'Naíve Bayes', plt.cm.Blues)
plt.subplot(122)
cm_axes(y_test2, y_pred_knn, 'KNN', plt.cm.Reds)
plt.show()
plt.figure(figsize=(16,8))
plt.subplot(121)
cm_axes(y_test2, y_pred_logit1, 'Logistic Regression #1', plt.cm.CMRmap_r)
plt.subplot(122)
cm_axes(y_test3, y_pred_logit2, 'Logistic Regression #2', plt.cm.CMRmap_r)
plt.show()
```


Naíve Bayes
True Positives
80
False Positives
16
False Negatives
55
True Negatives
39
Predicted
Disease +ve
Disease -ve
Disease +ve
Disease -ve
Actual


KNN
True Positives
113
False Positives
34
False Negatives
22
True Negatives
21
Predicted
Disease +ve
Disease -ve
Disease +ve
Disease -ve
Actual


Logistic Regression #1
True Positives
119
False Positives
40
False Negatives
16
True Negatives
15
Predicted
Disease +ve
Disease -ve
Disease +ve
Disease -ve
Actual


Logistic Regression #2
True Positives
124
False Positives
45
False Negatives
7
True Negatives
14
Predicted
Disease +ve
Disease -ve
Disease +ve
Disease -ve
Actual

In [38]:

```
# Print full classification report showing precision, recall, etc.
class formatter:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'

print(formatter.BOLD + formatter.UNDERLINE + 'Report for Naíve Bayes\n' + formatter.END
)
print(classification_report(y_test1, y_pred_nb, digits=3, target_names=['Disease -ve',
'Disease +ve']))
print('\n' + formatter.BOLD + formatter.UNDERLINE + 'Report for KNN\n' + formatter.END)
print(classification_report(y_test2, y_pred_knn, digits=3, target_names=['Disease -ve',
'Disease +ve']))
print('\n' + formatter.BOLD + formatter.UNDERLINE + 'Report for Logistic Regression #1
\n' + formatter.END)
print(classification_report(y_test2, y_pred_logit1, digits=3, target_names=['Disease -v
e', 'Disease +ve']))
print('\n' + formatter.BOLD + formatter.UNDERLINE + 'Report for Logistic Regression #2
\n' + formatter.END)
print(classification_report(y_test3, y_pred_logit2, digits=3, target_names=['Disease -v
e', 'Disease +ve']))
```

Report for Naíve Bayes

	precision	recall	f1-score	support
Disease -ve	0.415	0.709	0.523	55
Disease +ve	0.833	0.593	0.693	135
accuracy			0.626	190
macro avg	0.624	0.651	0.608	190
weighted avg	0.712	0.626	0.644	190

Report for KNN

	precision	recall	f1-score	support
Disease -ve	0.488	0.382	0.429	55
Disease +ve	0.769	0.837	0.801	135
accuracy			0.705	190
macro avg	0.629	0.609	0.615	190
weighted avg	0.688	0.705	0.693	190

Report for Logistic Regression #1

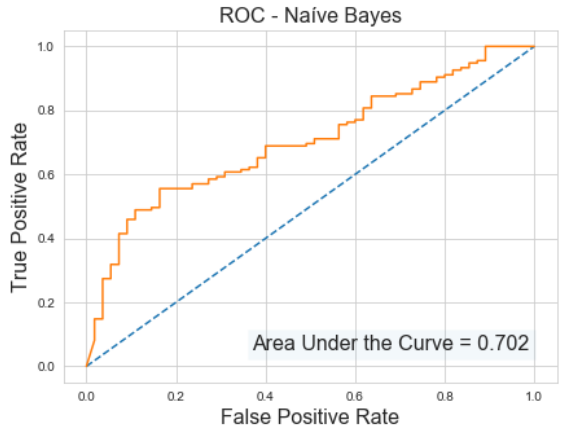
	precision	recall	f1-score	support
Disease -ve	0.484	0.273	0.349	55
Disease +ve	0.748	0.881	0.810	135
accuracy			0.705	190
macro avg	0.616	0.577	0.579	190
weighted avg	0.672	0.705	0.676	190

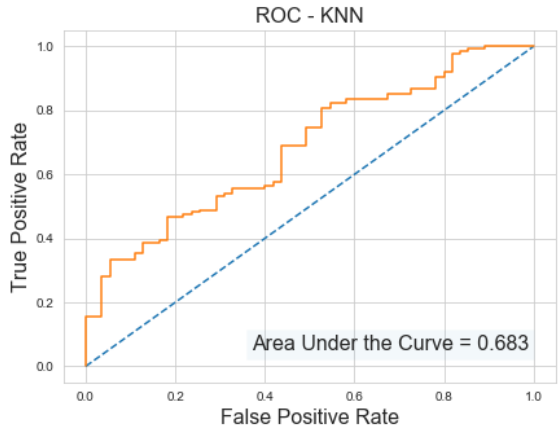
Report for Logistic Regression #2

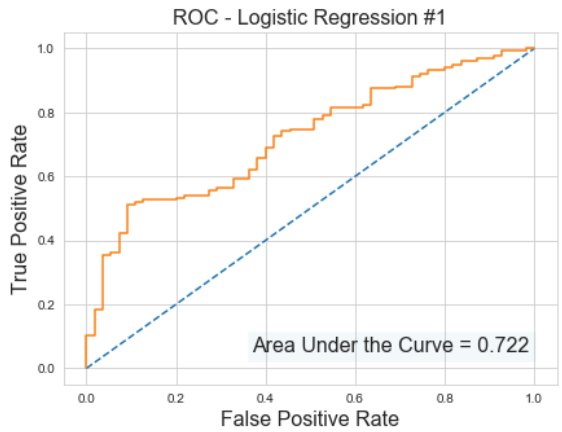
	precision	recall	f1-score	support
Disease -ve	0.667	0.237	0.350	59
Disease +ve	0.734	0.947	0.827	131
accuracy			0.726	190
macro avg	0.700	0.592	0.588	190
weighted avg	0.713	0.726	0.679	190

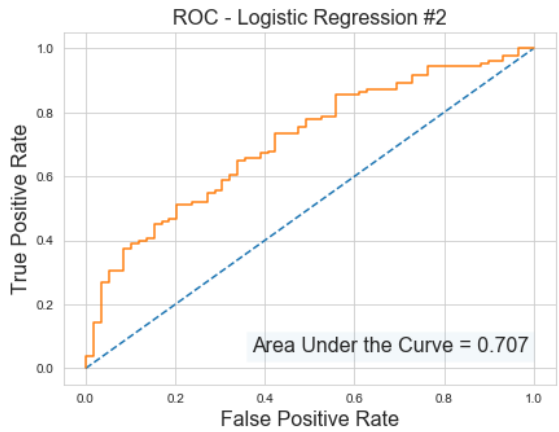
In [39]:

```
# ROC Plots for all models
def rocUtils(y_test, y_pred_prob, title,):
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    plt.plot([0,1], [0,1], '--')
    plt.text(0.99, 0.1, 'Area Under the Curve = %.3f' % roc_auc_score(y_test, y_pred_pr
ob),
             fontsize=16, horizontalalignment='right', verticalalignment='top', bbox=dic
t(alpha=0.05))
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate', size=16)
    plt.ylabel('True Positive Rate', size=16)
    plt.title('ROC - ' + title, size=16)

plt.figure(figsize=(15,5))
plt.subplot(121)
rocUtils(y_test1, y_pred_nb_prob, 'Naíve Bayes')
plt.subplot(122)
rocUtils(y_test2, y_pred_knn_prob, 'KNN')
plt.show()
plt.figure(figsize=(15,5))
plt.subplot(121)
rocUtils(y_test2, y_pred_logit1_prob, 'Logistic Regression #1')
plt.subplot(122)
rocUtils(y_test3, y_pred_logit2_prob, 'Logistic Regression #2')
plt.show()
```









In [40]:

```
# ROC Cut-Off
fpr, tpr, thresholds = roc_curve(y_test3, y_pred_logit2_prob)
roc_auc = auc(fpr, tpr)
print("Area under the ROC curve : %f" % roc_auc)

i = np.arange(len(tpr))
roc = pd.DataFrame({'fpr':pd.Series(fpr, index=i), 'tpr':pd.Series(tpr, index = i), '1-
fpr':pd.Series(1-fpr, index = i),
                    'tf':pd.Series(tpr - (1-fpr), index = i), 'thresholds':pd.Series(th
resholds, index = i)})
roc.iloc[(roc.tf-0).abs().argsort()[:1]]
```

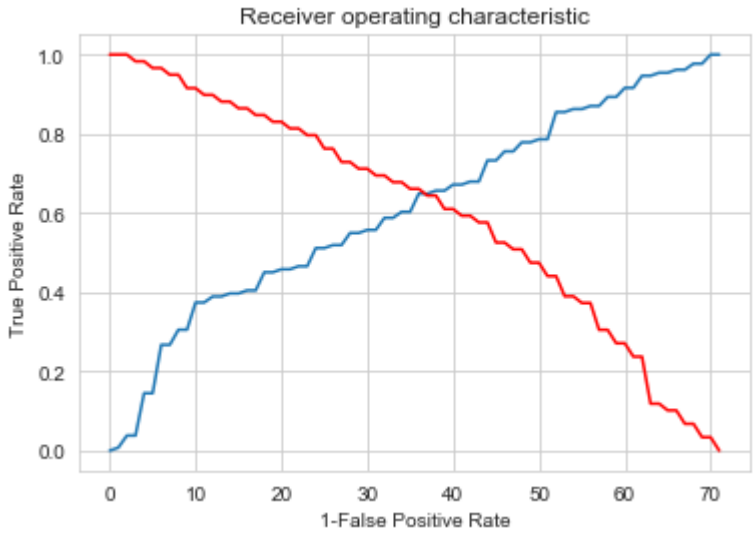
```
Area under the ROC curve : 0.707465
```

Out[40]:

	fpr	tpr	1-fpr	tf	thresholds
37	0.355932	0.648855	0.644068	0.004787	0.398038

In [41]:

```
# Plot tpr vs 1-fpr
fig, ax = plt.subplots()
plt.plot(roc['tpr'])
plt.plot(roc['1-fpr'], color = 'red')
plt.xlabel('1-False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
#ax.set_xticklabels([])
plt.show()
```



In [ ]:

Summary

Having built multiple classification models (best ones with feature engineering), the following is concluded -

1. Naíve Bayes: Generally the accuracy and recall is poor. Even with the best model, scores are poorer than K-NN or Logistic Regression although the precision is good. Without feature engineering (and considering all features), accuracy is lesser (~ 56%) and recall degrades even further (~ 0.42)
2. K-NN: Performs equally or better than Naíve Bayes and similar to Logistic Regression (although not always)
3. Logistic Regression: Performs equally or gets the better of the 3 algorithms (with/without feature engineering and optimizations). Two models were built, one with hand-written code and reduced features and the other with SKLearn with all the features and both perform very similar. Advantage of SKLearn is not having to choose a threshold for the Sigmoid which can be customized with own code

This shows that with the Naíve assumptions, Naíve Bayes perfoms poorly (non-correlation and feature independence). Logistic Regression accounts for correlations as well and performace does not degrade (shown in model #1 with all features).