AIMD Hospital Cancer Prediction (K-Nearest Neighbours)

In [1]:

```
# Import required libs
import seaborn as sb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats as sts
%matplotlib inline
```

In [2]:

```
# Experimenting own code, ignore for grading
# Distance metric for K-NN
def calcMinkowskiDist(norm, vec1, vec2):
    if(norm < 1 or not(isinstance(norm, int))):
        raise Exception('Norm must be a postive integer!')
    dist = 0
    for i in range(len(vec1)):
        dist += (abs(vec1[i] - vec2[i]))**norm
    return dist**(1/norm)

# Distance sorted neighbours
def neighbours(norm, test_vec, dataset):
    dist = list()
    for each_row in dataset:
        dist.append((each_row, calcMinkowskiDist(norm, each_row, test_vec)))
    dist.sort(key=lambda dist:dist[1])
    return dist

# Test Manhatten, Euclidean, Supremum (L1, L2 and L_inf norms)
calcMinkowskiDist(1, [0,0], [3,4]), calcMinkowskiDist(2, [0,0], [3,4]), calcMinkowskiDi
st(110, [0,0], [3,4])
```

Out[2]:

```
(7.0, 5.0, 4.0)
```

In [3]:

```
# Get neighbours for a dataset and test the algorithm
dataset = [[1,2,3],[4,5,6],[7,8,9]]
neighbours(2, dataset[1], dataset)
```

Out[3]:

```
[([4, 5, 6], 0.0),
 ([1, 2, 3], 5.196152422706632),
 ([7, 8, 9], 5.196152422706632)]
```

In [4]:

```
# Read cancer data-set
aimd = pd.read_csv("Cancer_Dataset.csv")
```

In [5]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Pregnancies     768 non-null    int64  
 1   Glucose         768 non-null    int64  
 2   BloodPressure   768 non-null    int64  
 3   SkinThickness   768 non-null    int64  
 4   Insulin         768 non-null    int64  
 5   BMI             768 non-null    float64
 6   Cancer_Markers  768 non-null    float64
 7   Age             768 non-null    int64  
 8   Outcome         768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [ ]:

```

```

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

In [6]:

```
# Check coloumn set
list(aimd.columns)
```

Out[6]:

```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'Cancer_Markers',
 'Age',
 'Outcome']
```

In [7]:

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

Out[7]:

```
Pregnancies          int64
Glucose              int64
BloodPressure        int64
SkinThickness        int64
Insulin              int64
BMI                float64
Cancer_Markers     float64
Age                  int64
Outcome              int64
dtype: object
```

In [8]:

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

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Cancer_Markers	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32

In [9]:

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

```
DataFrame Dimensions = 2
Dataframe Shape = (768, 9)
```

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%	7
Pregnancies	768.0	3.845052	3.369578	0.000	0.00000	1.00000	3.0000	6.00
Glucose	768.0	120.894531	31.972618	0.000	79.00000	99.00000	117.0000	140.25
BloodPressure	768.0	69.105469	19.355807	0.000	38.70000	62.00000	72.0000	80.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	0.00000	23.0000	32.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	0.00000	30.5000	127.25
BMI	768.0	31.992578	7.884160	0.000	21.80000	27.30000	32.0000	36.60
Cancer_Markers	768.0	0.471876	0.331329	0.078	0.14035	0.24375	0.3725	0.62
Age	768.0	33.240885	11.760232	21.000	21.00000	24.00000	29.0000	41.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.00000	0.0000	1.00

In [11]:

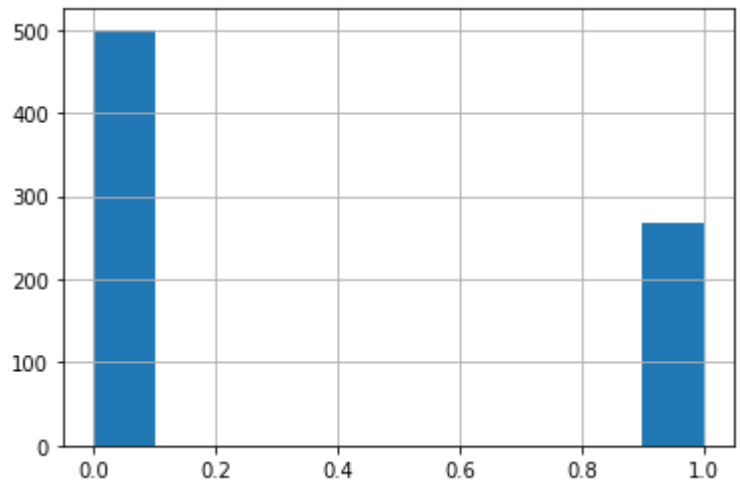
```
# Check that Outcome is either 0/1
print(aimd.Outcome.unique())
```

```
[1 0]
```

In [12]:

```
# Check class balance
print(aimd.Outcome.value_counts())
aimd.Outcome.hist()
plt.show()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



In [13]:

```
# Check duplicates, no duplicates if shape = (0, M)
aimd[aimd.duplicated()].shape
```

Out[13]:

```
(0, 9)
```

In [14]:

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

Out[14]:

```
Pregnancies       0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
Cancer_Markers    0
Age               0
Outcome           0
dtype: int64
```

In [15]:

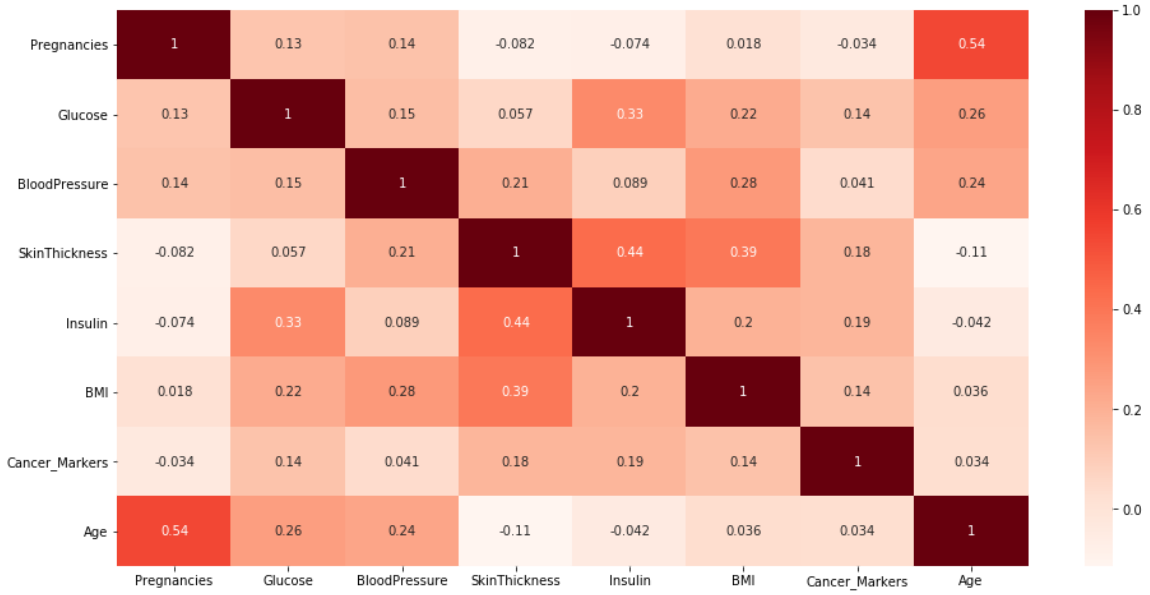
```
# Check the correlation matrix (exclude Outcome)
corr = aimd.iloc[:,:-1].corr()
corr
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000
Cancer_Markers	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242

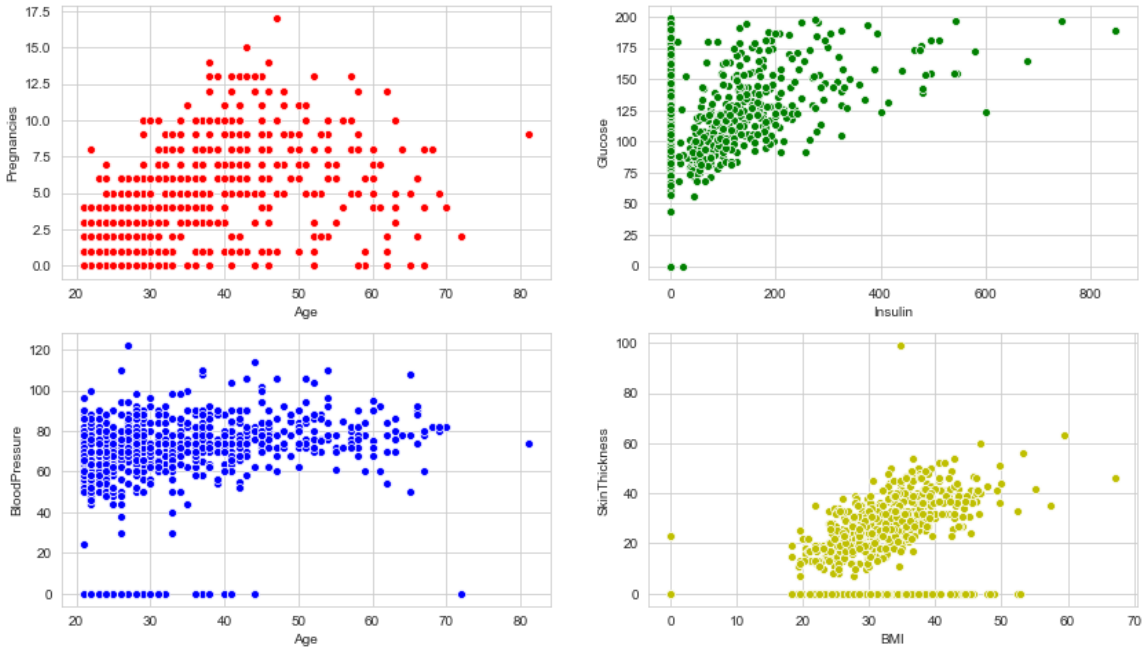
In [16]:

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



In [17]:

```
# Check correlations for >30% above using multi-variate plots
sb.set_style('whitegrid')
plt.figure(figsize=[14,8])
plt.subplot(2,2,1)
sb.scatterplot(x=aimd.Age, y=aimd.Pregnancies, color='r')
plt.subplot(2,2,2)
sb.scatterplot(x=aimd.Insulin, y=aimd.Glucose, color='g')
plt.subplot(2,2,3)
sb.scatterplot(x=aimd.Age, y=aimd.BloodPressure, color='b')
plt.subplot(2,2,4)
sb.scatterplot(x=aimd.BMI, y=aimd.SkinThickness, color='y')
plt.show()
```



In [18]:

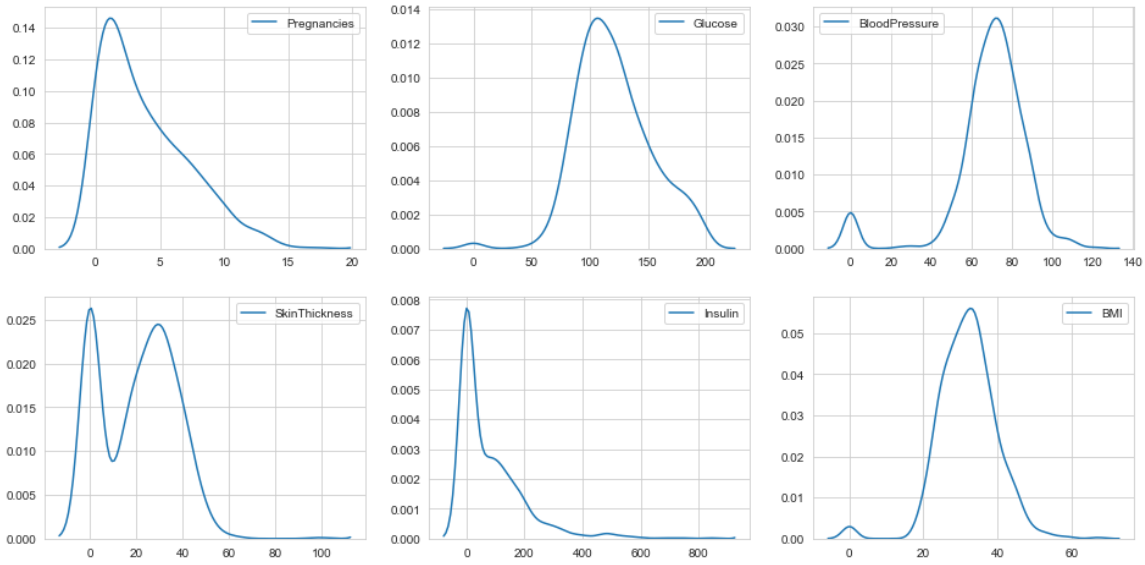
```
# Scatter and pair plots of data
sb.pairplot(aimd, diag_kind='kde', hue = 'Outcome')
plt.show()
```

In [19]:

```
# Rename columns to short values
#aimd = aimd.rename(columns={'Pregnancies':'PG', 'Glucose':'GLU', 'BloodPressure':'BP',
'SkinThickness':'ST'})
#aimd = aimd.rename(columns={'Insulin':'INS', 'Age':'AGE', 'Cancer_Markers':'CMR', 'Out
come':'CANCER'})
```

In [20]:

```
# Minimum values appears as 0 for few features, check the distributions
plt.figure(figsize=[16,8])
plt.subplot(2,3,1)
sb.kdeplot(aimd.Pregnancies)
plt.subplot(2,3,2)
sb.kdeplot(aimd.Glucose)
plt.subplot(2,3,3)
sb.kdeplot(aimd.BloodPressure)
plt.subplot(2,3,4)
sb.kdeplot(aimd.SkinThickness)
plt.subplot(2,3,5)
sb.kdeplot(aimd.Insulin)
plt.subplot(2,3,6)
sb.kdeplot(aimd.BMI)
plt.show()
```

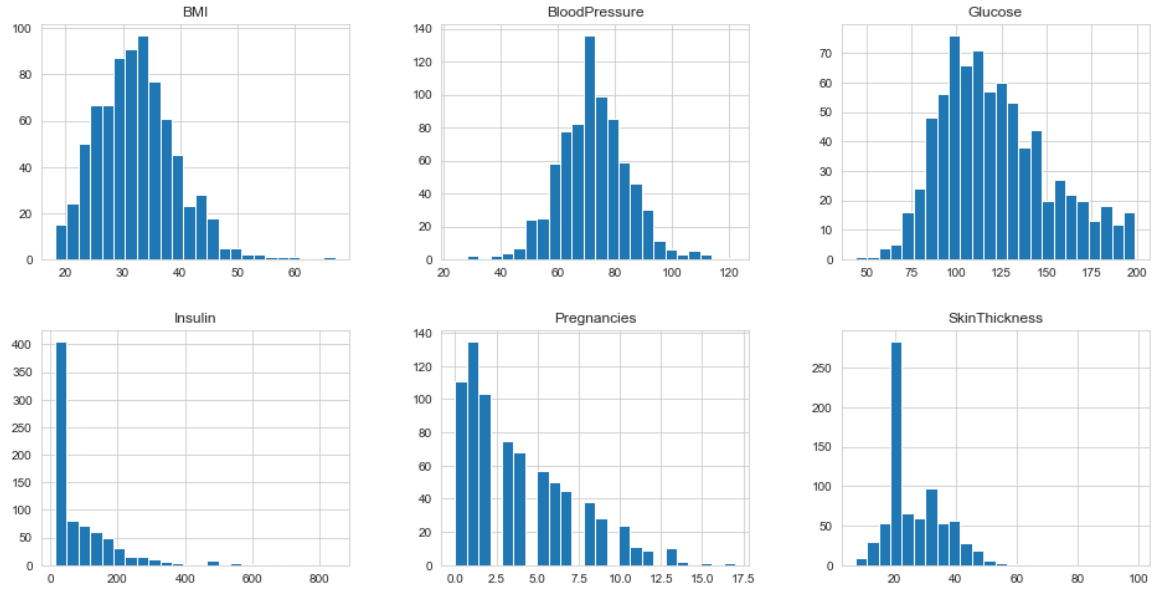


In [21]:

```
# Glucose, BP, SkinThickness and BMI are distributed normally, replace with mean
aimd.loc[aimd[aimd['Glucose'] == 0].index, 'Glucose'] = aimd['Glucose'].mean()
aimd.loc[aimd[aimd['BloodPressure'] == 0].index, 'BloodPressure'] = aimd['BloodPressur
e'].mean()
aimd.loc[aimd[aimd['SkinThickness'] == 0].index, 'SkinThickness'] = aimd['SkinThicknes
s'].mean()
aimd.loc[aimd[aimd['BMI'] == 0].index, 'BMI'] = aimd['BMI'].mean()

# Insulin is right-skewed, replace this with median
aimd.loc[aimd[aimd['Insulin'] == 0].index, 'Insulin'] = aimd['Insulin'].median()
```

In [22]:

```
aimd[['Pregnancies','Glucose','BloodPressure','SkinThickness','BMI','Insulin']].hist(fi
gsize=[16,8], bins=24, layout=(2,3))
plt.show()
```



In [23]:

```
# Keep a copy for SKLearn
aimd_orig = aimd.copy()
```

In [24]:

```
# Normalize features to have KNN work properly
X = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','Cancer_Ma
rkers','Age',]
aimd[X] = (aimd[X] - aimd[X].mean())/aimd[X].std()
display(aimd.head(3), aimd.tail(2))
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Cancer_Mark
0	0.639530	0.864712	-0.021031	0.871489	-0.607805	0.167147	0.468
1	-0.844335	-1.205204	-0.516246	0.248516	-0.607805	-0.850980	-0.364
2	1.233077	2.014666	-0.681318	-0.630243	-0.607805	-1.330954	0.604

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Cancer_Ma
766	-0.844335	0.141884	-1.011462	-0.630243	-0.607805	-0.341917	-0.3
767	-0.844335	-0.942357	-0.186103	0.456174	-0.607805	-0.298283	-0.4

In [ ]:

```
```

Modelling using SKLearn

In [25]:

```
# Import relevant sklearn libs
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1
_score, roc_curve, roc_auc_score
```

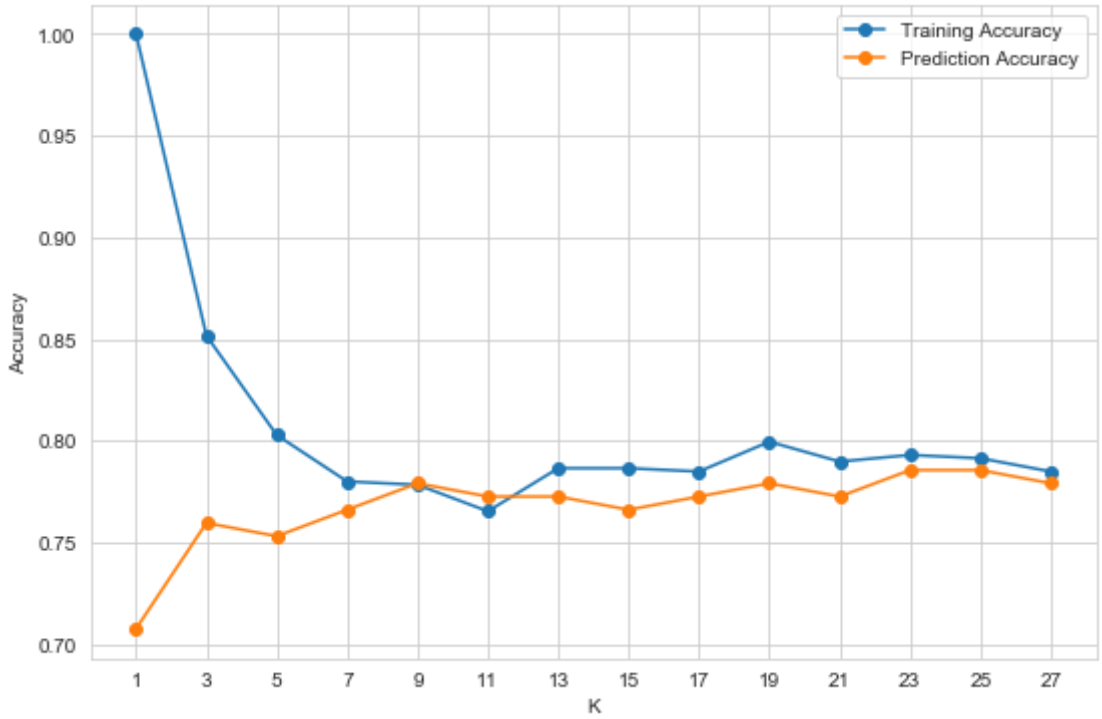
In [26]:

```
# Gather data and take the features for modelling
X = aimd_orig.loc[:, ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin'
,'BMI','Cancer_Markers','Age',]]
y = aimd_orig['Outcome']

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

# Split to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2
, stratify=y, random_state=0)
```

In [27]:

```
# Check and plot training and prediction accuracies to get the best K-value
# Checking only odd K-values as even values don't make sense, use K=27 as the root of e
lements(768) in dataset
train_acc = list()
test_acc = list()
n_range = np.arange(1,28,2)
for n,e in enumerate(n_range):
    knn = KNeighborsClassifier(n_neighbors=e, p=2).fit(X_train, y_train)
    train_acc.append(knn.score(X_train, y_train))
    test_acc.append(knn.score(X_test, y_test))
    print('K = {:<2}, Train = {:.5f}, Test = {:.5f}'.format(e, train_acc[n], test_acc[n
]))

# Plot the elbow curve to find train/test accuracy convergence and best K-value
plt.figure(figsize=(9,6))
plt.plot(n_range, train_acc, marker='o', label="Training Accuracy")
plt.plot(n_range, test_acc, marker='o', label="Prediction Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K")
plt.xticks(n_range)
plt.legend()
plt.show()
```

```
K = 1 , Train = 1.00000, Test = 0.70779
K = 3 , Train = 0.85179, Test = 0.75974
K = 5 , Train = 0.80293, Test = 0.75325
K = 7 , Train = 0.78013, Test = 0.76623
K = 9 , Train = 0.77850, Test = 0.77922
K = 11, Train = 0.76547, Test = 0.77273
K = 13, Train = 0.78664, Test = 0.77273
K = 15, Train = 0.78664, Test = 0.76623
K = 17, Train = 0.78502, Test = 0.77273
K = 19, Train = 0.79967, Test = 0.77922
K = 21, Train = 0.78990, Test = 0.77273
K = 23, Train = 0.79316, Test = 0.78571
K = 25, Train = 0.79153, Test = 0.78571
K = 27, Train = 0.78502, Test = 0.77922
```



In [28]:

```
# Check best K using sklearn
best_knn = GridSearchCV(KNeighborsClassifier(), param_grid = {'n_neighbors':range(1,27
), 'p':[1, 2]}, cv=10)
best_knn.fit(X,y)
best_knn.best_score_, best_knn.best_params_
```

Out[28]:

```
(0.7721462747778538, {'n_neighbors': 23, 'p': 2})
```

In [29]:

```
# As seen from the Elbow graph, and GridSearchCV, K=23 presents a good accuracy trade-o
ff and can be used to model with KNN
model = KNeighborsClassifier(algorithm='auto', n_jobs=1, n_neighbors=23, metric='minkow
ski', p=2, weights='uniform')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('Prediction Accuracy:', round(accuracy_score(y_test, y_pred), 5) * 100, '%')
```

```
Prediction Accuracy: 78.571 %
```

In [ ]:

```
```

Evaluate Model Performance

In [30]:

```
# Print the confusion matrix
pd.DataFrame(pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], m
argins_name='Total', margins=True))
```

Out[30]:

Predicted	0	1	Total
Actual			
0	91	9	100
1	24	30	54
Total	115	39	154

In [31]:

```
# Considering that not having cancer is positive -
# True Positives = 91
# True Negatives = 30
# False Positives = 24
# False Negatives = 9

# Considering that having cancer is positive -
# True Positives = 30
# True Negatives = 91
# False Positives = 9
# False Negatives = 24

# Print F1-Score for both the cases
round(f1_score(y_test, y_pred, pos_label=0),3), round(f1_score(y_test, y_pred, pos_labe
l=1),3)
```

Out[31]:

```
(0.847, 0.645)
```

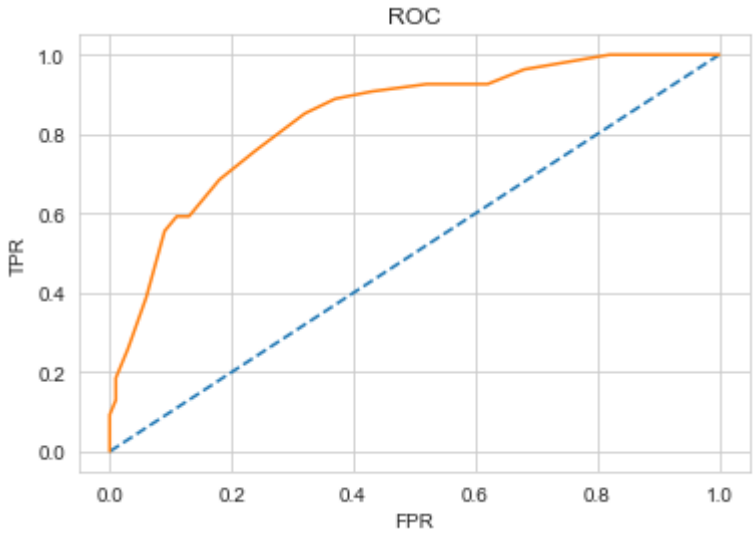
In [32]:

```
# Print full classification report showing precision, recall, etc.
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       100
           1       0.77      0.56      0.65        54

    accuracy                           0.79       154
   macro avg       0.78      0.73      0.75       154
weighted avg       0.78      0.79      0.78       154
```

In [33]:

```
# ROC Plot
y_pred_prob = model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0,1],[0,1],'--')
plt.plot(fpr, tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()

# AUC
round(roc_auc_score(y_test, y_pred_prob), 3)
```



Out[33]:

0.839