UCI Machine Learning Repository - Adult Income Data set

For this project we will be working with the UCI Repository Wiconsin Brest Cancer Data Set. This is a very famous data set and quite many papers have been published on the same!

We'll be trying to predict a classification- Diagnosis is Malignant '4' or benign '2'. Let's begin our understanding of implementing K-Nearest Neighbour in Python for classification.

We'll use the raw version of the data set (the one provided), and perform some categorical variables encoding (dummy variables) in python for quite many features if required.

Import Libraries

Let's import some libraries to get started!

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

The Data

Let's start by reading in the adult.csv file into a pandas dataframe.

In [2]:
```
df = pd.read_csv('mammographic_masses.data', names=['bi-rads', 'age', 'shape',
 'margin', 'density', 'severity'])
```

In [3]:
```
df.head()
```

Out[3]:

	bi-rads	age	shape	margin	density	severity
0	5	67	3	5	3	1
1	4	43	1	1	?	1
2	5	58	4	5	3	1
3	4	28	1	1	3	0
4	5	74	1	5	?	1

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 961 entries, 0 to 960
Data columns (total 6 columns):
bi-rads     961 non-null object
age         961 non-null object
shape       961 non-null object
margin      961 non-null object
density     961 non-null object
severity    961 non-null int64
dtypes: int64(1), object(5)
memory usage: 45.1+ KB
```

Cleaning the Data

The data will be cleaned as many of the values are missing from the data set. Some of the values might be dropped as we cannot simply replace them with certain averag data as this is important medical observation data.

Converting the object type data to Numeric Type

In [5]:
```
df1 = df.convert_objects(convert_numeric=True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWar
ning: convert_objects is deprecated.  Use the data-type specific converters p
d.to_datetime, pd.to_timedelta and pd.to_numeric.
  """Entry point for launching an IPython kernel.
```

In [7]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 961 entries, 0 to 960
Data columns (total 6 columns):
bi-rads     959 non-null float64
age         956 non-null float64
shape       930 non-null float64
margin      913 non-null float64
density     885 non-null float64
severity    961 non-null int64
dtypes: float64(5), int64(1)
memory usage: 45.1 KB
```

After converting to Numeric, the missing values are replaced is NaN. Dropping the NaN rows as they cannot be used for prediction.

In [9]:
```
df1.dropna(inplace=True)
```

In [10]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 830 entries, 0 to 960
Data columns (total 6 columns):
bi-rads     830 non-null float64
age         830 non-null float64
shape       830 non-null float64
margin      830 non-null float64
density     830 non-null float64
severity    830 non-null int64
dtypes: float64(5), int64(1)
memory usage: 45.4 KB
```

After dropping, we are remained of the total of 830 instances in data set

In [11]:
```
df1.head()
```

Out[11]:

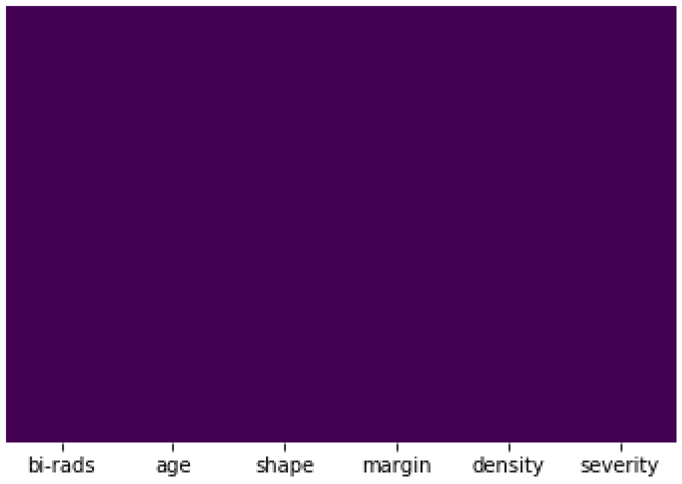
	bi-rads	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
8	5.0	57.0	1.0	5.0	3.0	1
10	5.0	76.0	1.0	4.0	3.0	1

Exploratory Data Analysis

Checking if we missed any null data to remove, i.e. checking if our data set still contains any null value.

```
In [12]: sns.heatmap(df1.isnull(), cbar=False, yticklabels=False, cmap='viridis')
```

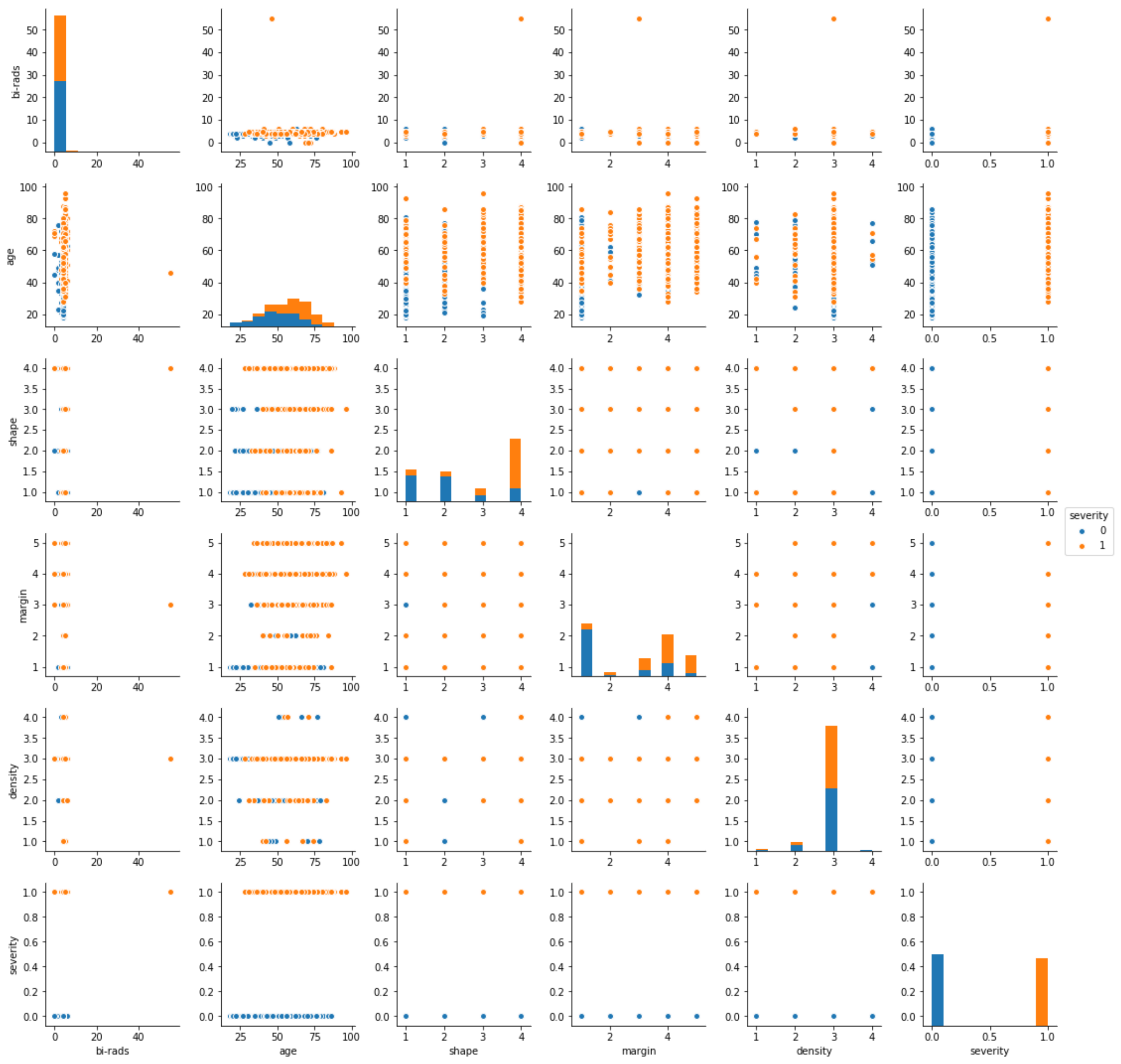
```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0xbbe9128>
```



From above heatmap, it is evident that our data does not contain any null value

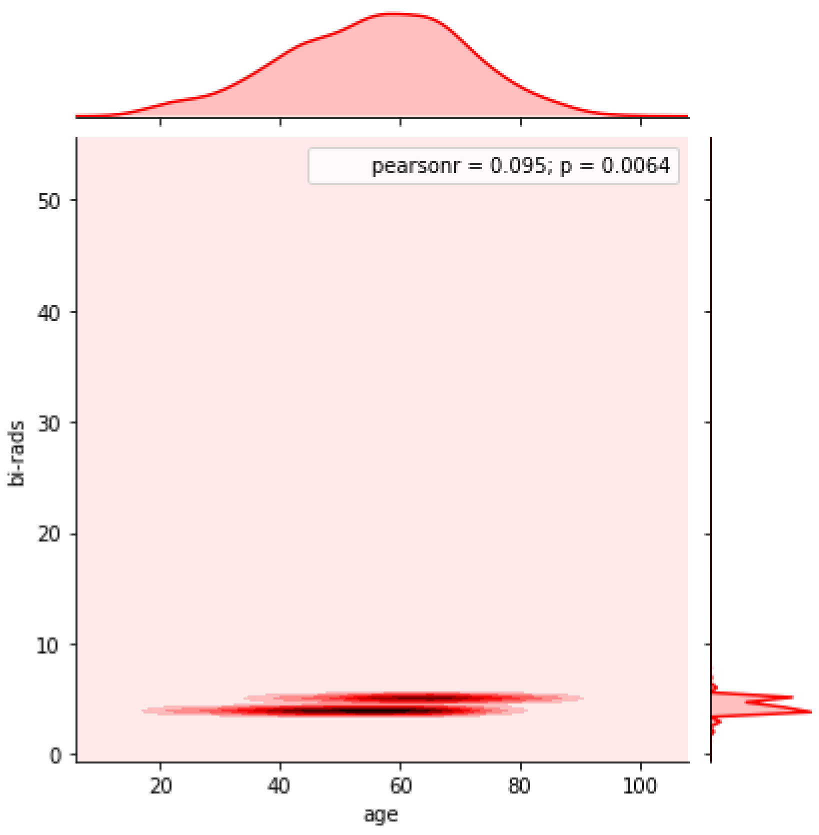
```
In [13]: sns.pairplot(df1, hue='severity')
```

```
Out[13]: <seaborn.axisgrid.PairGrid at 0xbbf4ac8>
```



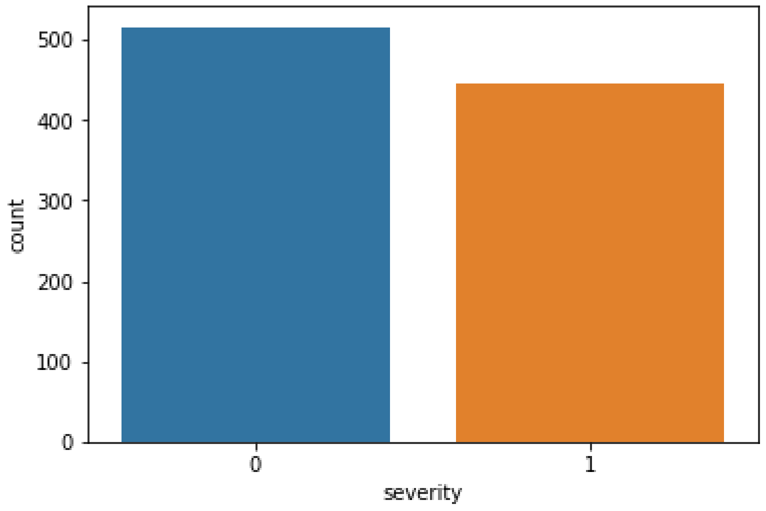
In [14]:
```
sns.jointplot(df1['age'], df1['bi-rads'], data=df1, kind='kde', color='red')
```

Out[14]:
```
<seaborn.axisgrid.JointGrid at 0xdf08710>
```



In [16]:
```
sns.countplot(df['severity'],)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc1bdd68>
```



In [35]:
```
low = df1.severity.value_counts()[0]
high= df1.severity.value_counts()[1]
total = low+high
low_per = np.round(np.divide(low, total)*100)
high_per = np.round(np.divide(high, total)*100)
```

In [38]:
```
print("Low:", low, " ","High:", high, " ", "Total:", total)
print("Low Percentage: ",low_per, "  ", "High Percentage:",high_per)
```

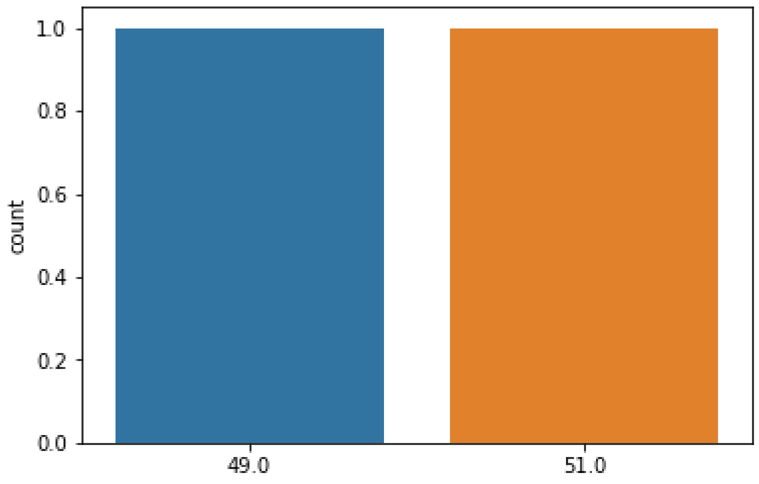
```
Low: 427   High: 403   Total: 830
Low Percentage:  51.0    High Percentage: 49.0
```

In [55]:
```
list = [low_per, high_per]
print(list)
```

```
[51.0, 49.0]
```

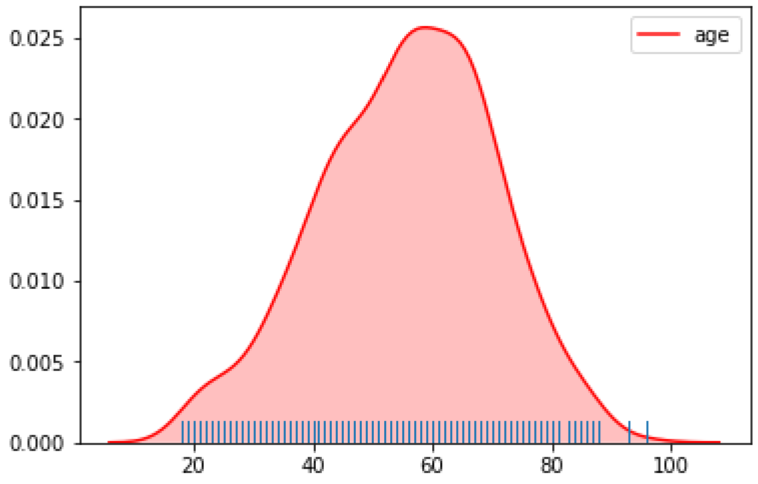
In [67]:
```
sns.countplot(x=list)
```

Out[67]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1028ccc0>
```



In [115]:
```
sns.kdeplot(df1.age, shade=True, color='r')
sns.rugplot(df1.age)
```

Out[115]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x123a5b38>
```



Feature Scaling

Normalising the data using scikit learn Standard Scaler

```
In [70]: from sklearn.preprocessing import StandardScaler
```

```
In [71]: scaler = StandardScaler()
```

```
In [72]: scaler.fit(df1.drop('severity', axis=1))

Out[72]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [73]: scaled_feat = scaler.transform(df1.drop('severity', axis=1))
```

```
In [74]: type(scaled_feat)

Out[74]: numpy.ndarray
```

Converting the scaled features to a DataFrame

```
In [75]: data = pd.DataFrame(scaled_feat, columns=df1.columns[:-1])
```

```
In [76]: data.head()
```

Out[76]:

	bi-rads	age	shape	margin	density
0	0.321118	0.765063	0.175636	1.396185	0.240466
1	0.321118	0.151271	0.981041	1.396185	0.240466
2	-0.208758	-1.894704	-1.435172	-1.157718	0.240466
3	0.321118	0.083071	-1.435172	1.396185	0.240466
4	0.321118	1.378855	-1.435172	0.757709	0.240466

Creating Training & Test Data

The feature vector has now been scaled/normalised. Now creating the training and Test Data from Data Set

```
In [77]: from sklearn.model_selection import train_test_split
```

```
In [78]: X_train, X_test, y_train, y_test = train_test_split(data, df1['severity'], tes
         t_size=0.3, random_state=101)
```

Initializing, Training & Predicting the model

Initially training the model on assumption of k='3'. Later we will use elbow method to derive the best probable value of 'K'

In [79]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [80]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [81]:
```
knn.fit(X_train, y_train)
```

Out[81]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

In [82]:
```
pred = knn.predict(X_test)
```

Evaluating the Model

Now the model has been trained and tested, we will evaluate the model for how good the model is ? with the help of certain metrics

In [83]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [84]:
```
print(confusion_matrix(y_test, pred))
print('\n')
print(classification_report(y_test, pred))
```

```
[[110  23]
 [ 33  83]]


             precision    recall  f1-score   support

          0       0.77      0.83      0.80       133
          1       0.78      0.72      0.75       116

avg / total       0.78      0.78      0.77       249
```

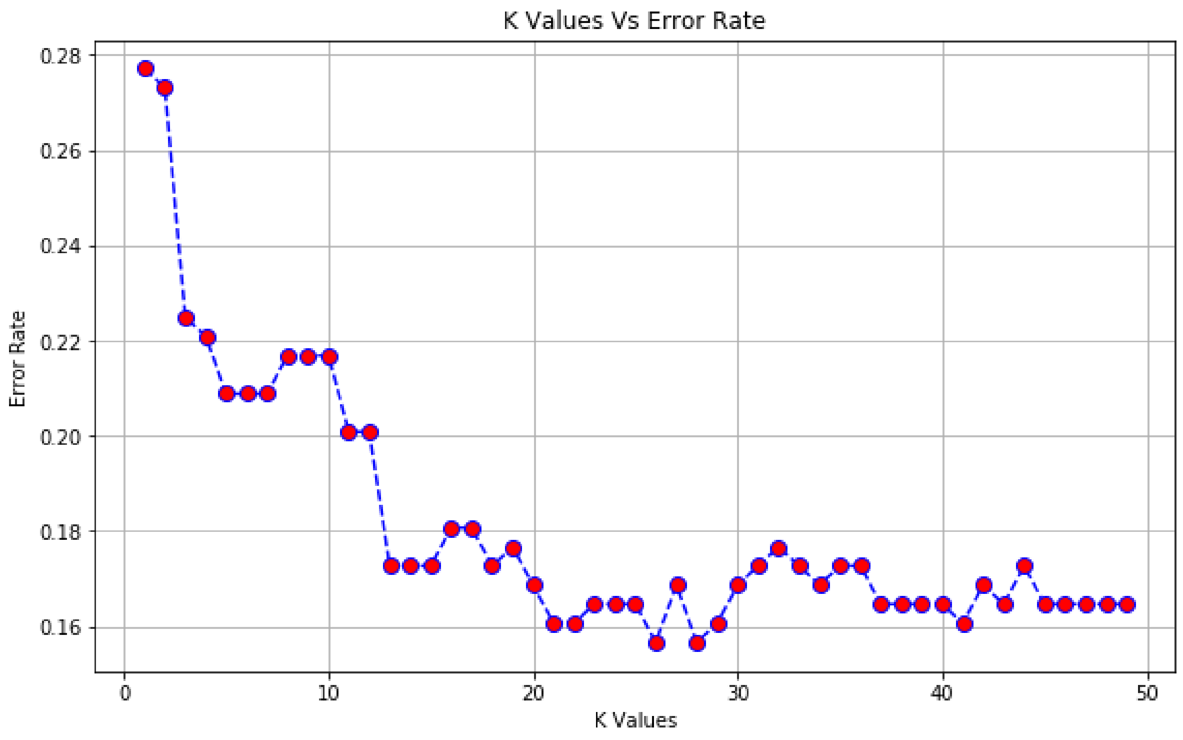
Elbow Method

Using the elbow method to find the best probable value of 'K' for which the error rate is minimal

In [101]:
```
error_rate = []
for i in range(1,50):
    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))
```

Now we have got the mean errors for values of 'K' ranging from 1 to 50. Lets plot this data as it becomes easy to look at it and find the best value of 'K' for which the error is lowest

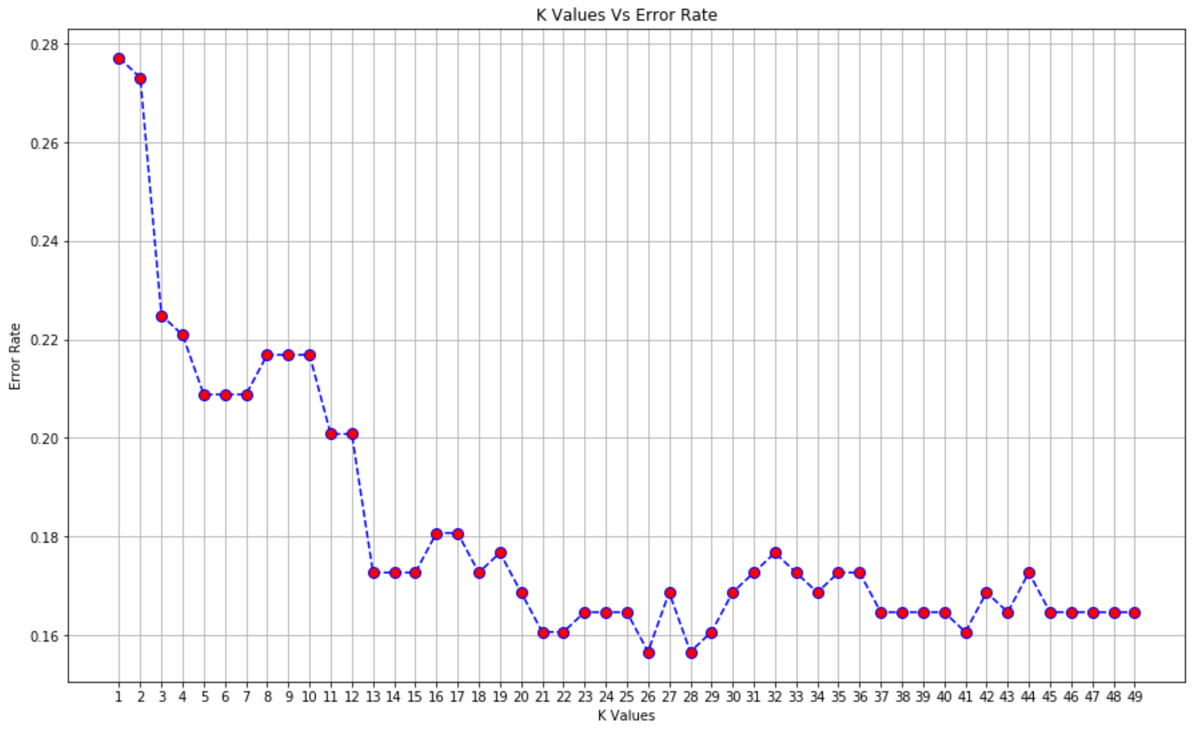
In [105]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,50), error_rate, color='blue', linestyle='--', marker='o', ma
rkerfacecolor='red', markersize=8)
plt.title('K Values Vs Error Rate')
plt.xlabel('K Values')
plt.ylabel('Error Rate')
plt.grid()
```



If you wish to see all the values on X axis, refer the below figure.

In [106]:
```
plt.figure(figsize=(15,9))
plt.plot(range(1,50), error_rate, color='blue', linestyle='--', marker='o', ma
rkerfacecolor='red', markersize=8)
plt.title('K Values Vs Error Rate')
plt.xlabel('K Values')
plt.ylabel('Error Rate')
plt.grid()
plt.xticks(range(1,50))
```

```
Out[106]: ([<matplotlib.axis.XTick at 0x120b9d30>,
            <matplotlib.axis.XTick at 0x122c4d30>,
            <matplotlib.axis.XTick at 0x13469a58>,
            <matplotlib.axis.XTick at 0x13486438>,
            <matplotlib.axis.XTick at 0x13486a90>,
            <matplotlib.axis.XTick at 0x1348b128>,
            <matplotlib.axis.XTick at 0x1348b780>,
            <matplotlib.axis.XTick at 0x1348bdd8>,
            <matplotlib.axis.XTick at 0x12350470>,
            <matplotlib.axis.XTick at 0x12350ac8>,
            <matplotlib.axis.XTick at 0x12357160>,
            <matplotlib.axis.XTick at 0x123577b8>,
            <matplotlib.axis.XTick at 0x12357e10>,
            <matplotlib.axis.XTick at 0x1235f4a8>,
            <matplotlib.axis.XTick at 0x1235fb00>,
            <matplotlib.axis.XTick at 0x12365198>,
            <matplotlib.axis.XTick at 0x123657f0>,
            <matplotlib.axis.XTick at 0x12365e48>,
            <matplotlib.axis.XTick at 0x1236b4e0>,
            <matplotlib.axis.XTick at 0x1236bb38>,
            <matplotlib.axis.XTick at 0x123721d0>,
            <matplotlib.axis.XTick at 0x12372828>,
            <matplotlib.axis.XTick at 0x12372e80>,
            <matplotlib.axis.XTick at 0x12377518>,
            <matplotlib.axis.XTick at 0x12377b70>,
            <matplotlib.axis.XTick at 0x1237f208>,
            <matplotlib.axis.XTick at 0x1237f860>,
            <matplotlib.axis.XTick at 0x1237feb8>,
            <matplotlib.axis.XTick at 0x12385550>,
            <matplotlib.axis.XTick at 0x12385ba8>,
            <matplotlib.axis.XTick at 0x1238b240>,
            <matplotlib.axis.XTick at 0x1238b898>,
            <matplotlib.axis.XTick at 0x1238bef0>,
            <matplotlib.axis.XTick at 0x12392588>,
            <matplotlib.axis.XTick at 0x12392be0>,
            <matplotlib.axis.XTick at 0x12398278>,
            <matplotlib.axis.XTick at 0x123988d0>,
            <matplotlib.axis.XTick at 0x12398f28>,
            <matplotlib.axis.XTick at 0x123a05c0>,
            <matplotlib.axis.XTick at 0x123a0c18>,
            <matplotlib.axis.XTick at 0x123a52b0>,
            <matplotlib.axis.XTick at 0x123a5908>,
            <matplotlib.axis.XTick at 0x123a5f60>,
            <matplotlib.axis.XTick at 0x123ac5f8>,
            <matplotlib.axis.XTick at 0x123acc50>,
            <matplotlib.axis.XTick at 0x123b22e8>,
            <matplotlib.axis.XTick at 0x123b2940>,
            <matplotlib.axis.XTick at 0x123b2f98>,
            <matplotlib.axis.XTick at 0x123b9630>],
           <a list of 49 Text xticklabel objects>)
```



From the above graph it is evident that the error rate is lowest for K range in 25 to 28. We'll start with K = 28

In [110]:
```
knn = KNeighborsClassifier(n_neighbors=28)
knn.fit(X_train, y_train)
```

Out[110]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=28, p=2,
           weights='uniform')
```

In [111]:
```
pred = knn.predict(X_test)
```

In [112]:
```
print(confusion_matrix(y_test, pred))
print('\n')
print(classification_report(y_test, pred))
```

```
[[111  22]
 [ 17  99]]


             precision    recall  f1-score   support

          0       0.87      0.83      0.85       133
          1       0.82      0.85      0.84       116

avg / total       0.84      0.84      0.84       249
```

Hence from with K=28, our model predicts with 84% precision