Computing Final Project Executive Report

Data Analysis on Breast Cancer Prediction

Submitted By –

Tallam Sravan

Date – 12th May 2023

University of Colorado Denver

Table of Contents –

Data Description:

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei. The mean, standard error and “worst” or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

Attribute Information:

Real-valued features are computed for three cell nucleus that are mean, se, and worst.

A) Radius (mean of distances from centre to points on the perimeter)

B) Texture (standard deviation of gray-scale values)

C) Perimeter

D) Area

E) Smoothness (local variation in radius lengths)

F) Compactness (perimeter^2/area- 1.0)

G) Concavity (severity of concave portions of the contour)

H) Concave points (number of concave portions of the contour)

I) Symmetry

J) Fractal dimension (“coastline approximation” – 1)

K Nearest Neighbour (KNN) is exceptionally useful and easy to understand. KNN is used in a wide range of activities and industries such as financial, medical care, political theory, and other several industries. Further, it is also used in credit scoring, credit ratings for customers and loan disbursement. Politically KNN is used postal ballots to classify potential voters in regard to voting and non-voting. KNN can be used for both classification and regression KNN calculation in light of the element comparability approach.

KNN is a non-parametric and lazy learning calculation or algorithm.

Non-parametric means there are no assumptions for hidden underlying information. In simple words, the model is determined within the dataset. This will be exceptionally useful and by when the majority of this present reality dataset doesn't follow numerical hypothetical suppositions. The lazy algorithm means it needn't bother with any preparation data for model creation. All training data is used in the testing phase meaning the training is faster than the testing.

How would you choose the number of neighbours in KNN?

Presently, you comprehend the KNN calculation working instrument. Right now, the inquiry emerges How to pick the ideal number of neighbours? Also, what are its consequences for the classifier? The quantity of neighbours (K) in KNN is a hyper boundary that you really want to pick at the hour of model structure. You can consider K a controlling variable for the expectation model.

Research has shown that no ideal number of neighbours suits general sort of informational indexes. Each dataset has its own necessities. On account of few neighbours, the commotion will impact the outcome, and an enormous number of neighbours make it computationally costly. Research has likewise shown that a limited quantity of neighbours is the most adaptable fit which will have low inclination yet high fluctuation and countless neighbours will have a smoother choice limit which means lower difference but higher predisposition.

For the most part, Information researchers pick an odd number on the off chance that the quantity of classes is even. You can likewise check by producing the model on various upsides of k and actually look at their exhibition. You can likewise attempt the Elbow strategy here.

Defining the Libraries-

The libraries which are used in our project are mentioned below.

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#%matplotlib inline
```

In [5]:
```
from IPython.display import display
#pd.options.display.max_columns = None
#pd.options.display.max_rows = None
```

Reading the data –

In [6]:
```
# reading data set
df_cancerdata = pd.read_csv("cancerdata.csv")
display(df_cancerdata)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	842302	M	14.204716	20.202641	122.80	1001.0	0.096113	0.089479	0.184766	0.096526
1	842517	M	18.483496	22.464181	132.90	1326.0	0.094443	0.063299	0.086927	0.021108
2	84300903	M	15.516300	18.271486	130.00	1203.0	0.075842	0.059376	0.132738	0.052357
3	84348301	M	15.540583	27.560108	77.58	386.1	0.091525	0.088022	0.194466	0.010895
4	84358402	M	14.005198	11.678064	135.10	1297.0	0.092790	0.010642	0.030518	0.037828
...	...	...	...	...	...	...	...	...	...	...
564	926424	M	9.477218	17.173224	142.00	1479.0	0.110550	0.037279	0.160806	0.047462
565	926682	M	11.657306	25.502865	131.20	1261.0	0.113881	0.108912	0.134949	0.039279
566	926954	M	20.102883	14.544119	108.30	858.1	0.100213	0.003465	0.072762	0.042914
567	927241	M	16.206551	19.616466	140.10	1265.0	0.112486	0.158756	0.096860	0.024182
568	92751	B	13.256191	25.087474	47.92	181.0	0.110289	0.044248	-0.072171	0.029954

569 rows × 32 columns

In [7]:
```
x=df_cancerdata['id']
```

In [8]:
```
df_cancerdata.info()
```

The dataset consists of 569 Observations with 32 Variables, there are no missing values in the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_wors         569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [9]:

```
import warnings
warnings.filterwarnings('ignore')
```

Removing Columns –

In the next step we remove the columns which are not required.

```
In [10]: # removing id and Unnamed columns
df_cancerdata = df_cancerdata.drop(['id'], axis = 1)
```

```
In [11]: df_cancerdata[['radius_mean','perimeter_mean']]
```

Out[11]:

	radius_mean	perimeter_mean
0	14.204716	122.80
1	18.483496	132.90
2	15.516300	130.00
3	15.540583	77.58
4	14.005198	135.10
...	...	...
564	9.477218	142.00
565	11.657306	131.20
566	20.102883	108.30
567	16.206551	140.10
568	13.256191	47.92

After removing the unwanted column (id) the variables are now down to 31.

Dimensions of the taken dataset –

```
In [12]: # To get dimensions of dataset
df_cancerdata.shape

Out[12]: (569, 31)
```

Diagnosis is the only variable that is object (string) rest all are float datatypes.

Checking the Data types –

```
In [13]: # check datatypes
df_cancerdata.dtypes

Out[13]: diagnosis                   object
radius_mean                 float64
texture_mean                float64
perimeter_mean              float64
area_mean                   float64
smoothness_mean             float64
compactness_mean            float64
concavity_mean              float64
concave points_mean         float64
symmetry_mean               float64
fractal_dimension_mean      float64
radius_se                   float64
texture_se                  float64
perimeter_se                float64
area_se                     float64
smoothness_se               float64
compactness_se              float64
concavity_se                float64
concave points_se           float64
symmetry_se                 float64
fractal_dimension_se        float64
radius_worst                float64
texture_worst               float64
perimeter_worst             float64
area_worst                  float64
smoothness_worst            float64
compactness_wors            float64
concavity_worst             float64
concave points_worst        float64
symmetry_worst              float64
fractal_dimension_worst     float64
dtype: object
```

dtypes handles data type of variables. here we can see that all the values are numeric in our dataset

Checking the missing values –

In [14]:
```
# Check for null values
df_cancerdata.isnull().sum()
```

Out[14]:
```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_wors           0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In the above result we can see that null values are not present in the dataset

Summary Statistics –

Here we take a look at the summary of each attribute

In [15]:
```
# To get summary statistics of data
df_cancerdata.describe()
```

Out[15]:

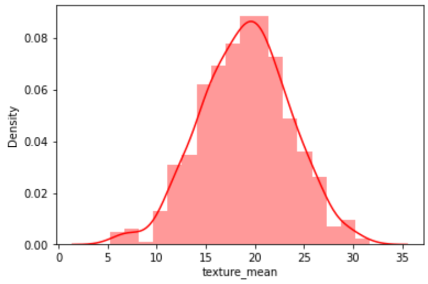
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.157073	19.025699	91.969033	654.889104	0.095887	0.106397	0.092388	0.048655	0.179979
std	3.435295	4.515361	24.298981	351.914129	0.013778	0.051773	0.082573	0.042227	0.026742
min	2.174108	5.225685	43.790000	143.500000	0.045915	-0.055431	-0.156186	-0.078615	0.103582
25%	11.762811	15.926529	75.170000	420.300000	0.085801	0.070217	0.037784	0.020214	0.162974
50%	14.142920	19.135095	86.240000	551.100000	0.096244	0.105569	0.093832	0.048666	0.179796
75%	16.403557	22.060029	104.100000	782.700000	0.105867	0.143065	0.148154	0.077010	0.198569
max	24.961616	31.692291	188.500000	2501.000000	0.144596	0.248430	0.335120	0.189527	0.261135

8 rows × 30 columns

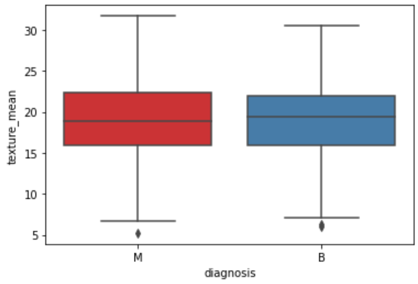
Summary of statistics pertaining to the DataFrame columns. This function gives the mean, std, minimum value, maximum value and IQR values and given summary about numeric columns

Below is the first data visualization we did using the seaborn package (library), we used the "texture mean" and plotted a distribution plot which is a combination of distribution and a histogram. A box-plot is followed then by the distribution plot with two variables diagnosis and texture mean. As we observe the plot is almost normally distributed.

```
In [16]:  sns.distplot(df_cancerdata["texture_mean"],color='red')
Out[16]: <AxesSubplot:xlabel='texture_mean', ylabel='Density'>
```

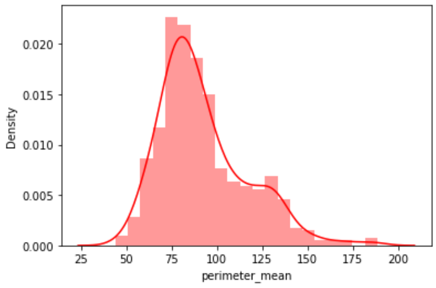



```
In [17]:  sns.boxplot(x="diagnosis",y="texture_mean",data=df_cancerdata,palette="Set1")
Out[17]: <AxesSubplot:xlabel='diagnosis', ylabel='texture_mean'>
```

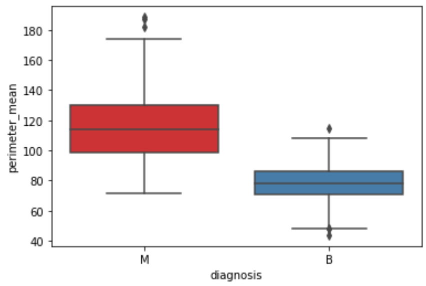


The third visualization is also a distribution plot with perimeter mean, it is then followed by a box-plot with two variables diagnosis and perimeter mean. As we can see the distribution plot is rightly skewed.

```
In [18]:  sns.distplot(df_cancerdata["perimeter_mean"],color="red")
Out[18]: <AxesSubplot:xlabel='perimeter_mean', ylabel='Density'>
```




```
In [19]:  sns.boxplot(x="diagnosis",y="perimeter_mean",data=df_cancerdata,palette="Set1")
Out[19]: <AxesSubplot:xlabel='diagnosis', ylabel='perimeter_mean'>
```



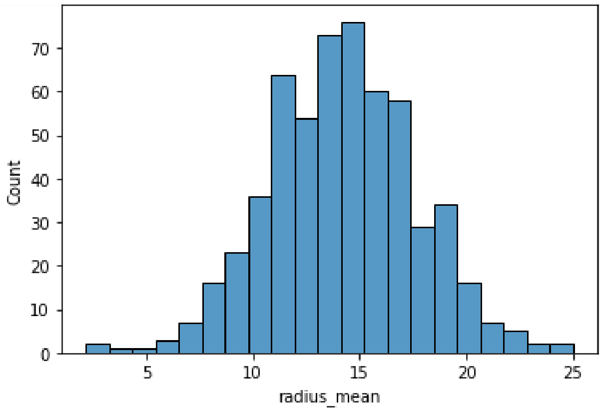
In [27]:
```
cancer_num= df_cancerdata.select_dtypes(include=np.number)
cancer_num.head()
```

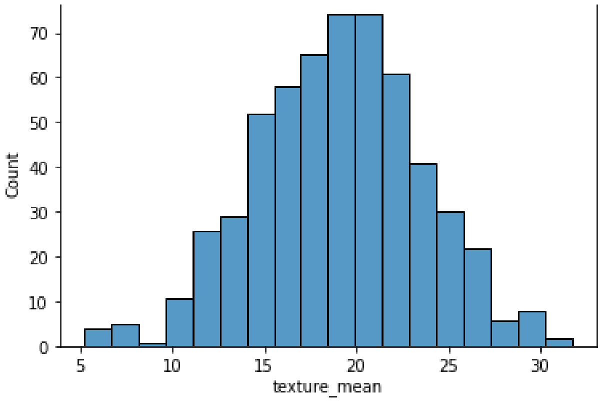
Out[27]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal
0	14.204716	20.202641	122.80	1001.0	0.096113	0.089479	0.184766	0.096526	0.150053	
1	18.483496	22.464181	132.90	1326.0	0.094443	0.063299	0.086927	0.021108	0.213798	
2	15.516300	18.271486	130.00	1203.0	0.075842	0.059376	0.132738	0.052357	0.222581	
3	15.540583	27.560108	77.58	386.1	0.091525	0.088022	0.194466	0.010895	0.179685	
4	14.005198	11.678064	135.10	1297.0	0.092790	0.010642	0.030518	0.037828	0.147534	

5 rows × 30 columns

In [28]:
```
from matplotlib.pyplot import figure
for i in cancer_num.columns:
    figure()
    sns.histplot(cancer_num[i])
```



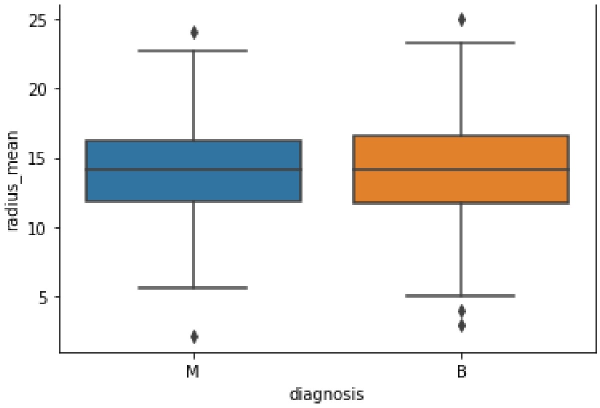


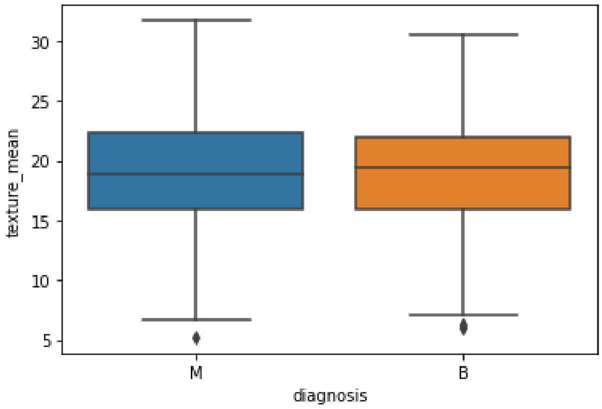
In [29]:
```
cancer_num.columns
```

Out[29]:
```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_wors',
       'concavity_worst', 'concave points_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

Next, we used the matplotlib.pyplot library to plot various boxplots for numerical variables on the y-axis with x-axis being a categorical variable (diagnosis).

In [30]:
```
from matplotlib.pyplot import figure
for i in cancer_num.columns:
    figure()
    sns.boxplot(y=cancer_num[i],x=df_cancerdata['diagnosis'])
```





Further, we did a confusion matrix to determine the correlation between all the variables.

In [31]:
```
#calculating correlation among numeric variable
corr_matrix = cancer_num.corr()

corr_matrix
```

Out[31]:

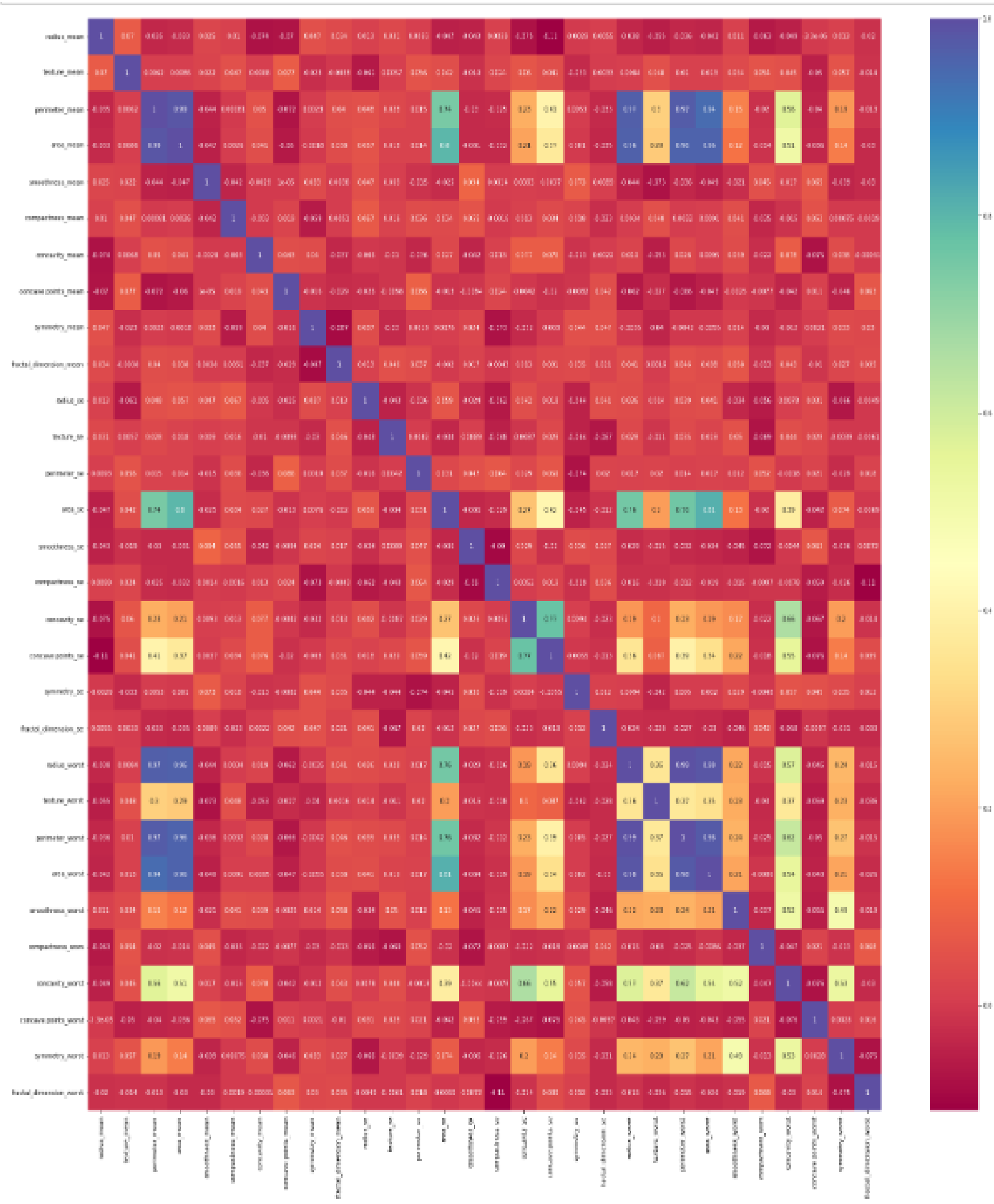
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	s
radius_mean	1.000000	0.070425	-0.035228	-0.032708	0.024861	0.010012	-0.074375	-0.070011	
texture_mean	0.070425	1.000000	0.006155	0.008631	0.021654	0.047145	0.006832	0.077424	
perimeter_mean	-0.035228	0.006155	1.000000	0.986507	-0.043853	0.000807	0.049568	-0.071642	
area_mean	-0.032708	0.008631	0.986507	1.000000	-0.047062	0.002561	0.040642	-0.060022	
smoothness_mean	0.024861	0.021654	-0.043853	-0.047062	1.000000	-0.041543	-0.002777	0.000010	
compactness_mean	0.010012	0.047145	0.000807	0.002561	-0.041543	1.000000	-0.002953	0.019169	
concavity_mean	-0.074375	0.006832	0.049568	0.040642	-0.002777	-0.002953	1.000000	0.042943	
concave points_mean	-0.070011	0.077424	-0.071642	-0.060022	0.000010	0.019169	0.042943	1.000000	
symmetry_mean	0.046623	-0.023344	0.002254	-0.001786	0.032645	-0.058799	0.039856	-0.015877	
fractal_dimension_mean	0.033805	-0.003806	0.040042	0.038391	0.003785	0.005069	-0.037378	-0.028798	
radius_se	0.012795	-0.061184	0.047773	0.057057	0.047041	0.067240	-0.004993	-0.024701	
texture_se	0.030862	0.005729	0.028094	0.017864	0.008970	0.016067	-0.009966	-0.009831	
perimeter_se	0.009318	0.056161	0.015025	0.014144	-0.015423	0.036341	-0.035997	0.085809	
area_se	-0.046940	0.042223	0.744983	0.800086	-0.025343	0.033558	0.027478	-0.012913	
smoothness_se	-0.042970	-0.018967	-0.029894	-0.030848	0.094440	0.054655	-0.041651	-0.009404	
compactness_se	0.009876	0.024364	-0.025082	-0.032259	0.001379	-0.001585	0.013409	0.024362	
concavity_se	-0.074536	0.059769	0.228082	0.207660	0.009347	0.013099	0.076777	-0.004215	
concave points_se	-0.106635	0.041280	0.407217	0.372320	0.003734	0.033779	0.075849	-0.019548	
symmetry_se	-0.002874	-0.032921	0.005295	0.001013	0.073480	0.018448	-0.012510	-0.008168	
fractal_dimension_se	0.005511	0.003319	-0.032627	-0.035046	0.008877	-0.022843	0.002185	0.042237	
radius_worst	-0.037750	0.008352	0.969476	0.962746	-0.044234	0.003426	0.019058	-0.062171	
texture_worst	-0.054781	0.048012	0.303038	0.287489	-0.073102	0.048005	-0.052707	-0.026923	
perimeter_worst	-0.035869	0.010269	0.970387	0.959120	-0.036288	0.003217	0.028040	-0.065773	
area_worst	-0.041812	0.012803	0.941550	0.959213	-0.048910	0.009143	0.009482	-0.047116	
smoothness_worst	0.011173	0.033872	0.150549	0.123523	-0.021490	0.040886	0.039363	-0.002481	
compactness_wors	-0.063024	0.053946	-0.020032	-0.013764	0.044842	-0.035002	-0.021552	-0.007732	
concavity_worst	-0.049089	0.044502	0.563879	0.512606	0.016772	-0.015305	0.077975	-0.041824	
concave points_worst	-0.000033	-0.049535	-0.039836	-0.036284	0.065017	0.052424	-0.074706	0.011258	
symmetry_worst	0.013231	0.056662	0.189115	0.143570	-0.038502	0.000749	0.037577	-0.046071	
fractal_dimension_worst	-0.019669	-0.013935	-0.012852	-0.029543	-0.029748	-0.001925	-0.000306	0.063022	

Below is the correlation heatmap, and as we observe perimeter mean and radius worst, perimeter worst and radius worst, perimeter worst and area worst have the highest correction with 0.99, 0.98 and 0.98. Area se and concave se has the least correlation with 0.26.

In [32]:

```
#plot correlation matrix
plt.figure(figsize=(30,30))

sns.heatmap(corr_matrix,cmap='Spectral',annot=True);plt.show()
```



Next, we did standardization for our dataset using min-max scaler and we imported that from sklearn.preprocessing. Standardization is used to ensure that we have zero mean and unit standard deviation for easy model building.

In [34]:
```
# apply the min-max scaling to our numeric variables
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()

# Scaling down the numeric variables
#df_housingdata_numcols = pd.DataFrame(min_max.fit_transform(df_numeric_features.iloc[:,0:38]),columns = df_numeric_features.

df_cancerdata_inp=pd.DataFrame(min_max.fit_transform(df_cancerdata.iloc[:,1:31]),columns=df_cancerdata.iloc[:,1:31].columns.t
```

In [35]:
```
df_cancerdata_inp.head()
```

Out[35]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal
0	0.527948	0.565881	0.545989	0.363733	0.508690	0.476895	0.693971	0.653166	0.294959	
1	0.715716	0.651330	0.615783	0.501591	0.491769	0.390740	0.494830	0.371905	0.699552	
2	0.585505	0.492916	0.595743	0.449417	0.303272	0.377829	0.588074	0.488442	0.755294	
3	0.586570	0.843872	0.233501	0.102906	0.462201	0.472101	0.713713	0.333815	0.483032	
4	0.519192	0.243793	0.630986	0.489290	0.475016	0.217447	0.380016	0.434261	0.278969	

5 rows × 30 columns

In [36]:
```
df_cancerdata.iloc[:,1:31].columns
```

Out[36]:
```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_wors',
       'concavity_worst', 'concave points_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

After standardization the columns are now reduced to 30.

In [37]:
```
df_cancerdata_inp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   radius_mean              569 non-null    float64
 1   texture_mean             569 non-null    float64
 2   perimeter_mean           569 non-null    float64
 3   area_mean                569 non-null    float64
 4   smoothness_mean          569 non-null    float64
 5   compactness_mean         569 non-null    float64
 6   concavity_mean           569 non-null    float64
 7   concave points_mean      569 non-null    float64
 8   symmetry_mean            569 non-null    float64
 9   fractal_dimension_mean   569 non-null    float64
 10  radius_se                569 non-null    float64
 11  texture_se               569 non-null    float64
 12  perimeter_se             569 non-null    float64
 13  area_se                  569 non-null    float64
 14  smoothness_se            569 non-null    float64
 15  compactness_se           569 non-null    float64
 16  concavity_se             569 non-null    float64
 17  concave points_se        569 non-null    float64
 18  symmetry_se              569 non-null    float64
 19  fractal_dimension_se     569 non-null    float64
 20  radius_worst             569 non-null    float64
 21  texture_worst            569 non-null    float64
 22  perimeter_worst          569 non-null    float64
 23  area_worst               569 non-null    float64
 24  smoothness_worst         569 non-null    float64
 25  compactness_wors         569 non-null    float64
 26  concavity_worst          569 non-null    float64
 27  concave points_worst     569 non-null    float64
 28  symmetry_worst           569 non-null    float64
 29  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [38]:
```
df_cancerdata_inp.iloc[:,1:31]
```

Out[38]:

	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_
0	0.565881	0.545989	0.363733	0.508690	0.476895	0.693971	0.653166	0.294959	0.3
1	0.651330	0.615783	0.501591	0.491769	0.390740	0.494830	0.371905	0.699552	0.7
2	0.492916	0.595743	0.449417	0.303272	0.377829	0.588074	0.488442	0.755294	0.3
3	0.843872	0.233501	0.102906	0.462201	0.472101	0.713713	0.333815	0.483032	0.5
4	0.243793	0.630986	0.489290	0.475016	0.217447	0.380016	0.434261	0.278969	0.6
...	...	...	...	...	...	...	...	...	
564	0.451419	0.678668	0.566490	0.654992	0.305106	0.645202	0.470187	0.343029	0.4
565	0.766142	0.604036	0.474019	0.688749	0.540850	0.592573	0.439670	0.664148	0.6
566	0.352083	0.445788	0.303118	0.550239	0.193828	0.465998	0.453227	0.682039	0.4
567	0.543734	0.665538	0.475716	0.674606	0.704884	0.515048	0.383369	0.302933	0.5
568	0.750447	0.028540	0.015907	0.652342	0.328043	0.171003	0.404896	0.338923	0.4

569 rows × 29 columns

In [39]:
```
df_cancerdata_inp.describe()
```

Out[39]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fr
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	0.525857	0.521412	0.332935	0.216920	0.506398	0.532573	0.505944	0.474637	0.484899	
std	0.150753	0.170606	0.167915	0.149274	0.139618	0.170385	0.168069	0.157481	0.169732	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.420788	0.404315	0.216847	0.117413	0.404198	0.413507	0.394804	0.368569	0.376968	
50%	0.525236	0.525546	0.293345	0.172895	0.510015	0.529847	0.508883	0.474678	0.483738	
75%	0.624441	0.636060	0.416765	0.271135	0.607539	0.653246	0.619451	0.580385	0.602891	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 30 columns

In [40]:
```
df_cancerdata_inp['diagnosis']=df_cancerdata['diagnosis']

df_cancerdata=df_cancerdata_inp

df_cancerdata.head()
```

Out[40]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal
0	0.527948	0.565881	0.545989	0.363733	0.508690	0.476895	0.693971	0.653166	0.294959	
1	0.715716	0.651330	0.615783	0.501591	0.491769	0.390740	0.494830	0.371905	0.699552	
2	0.585505	0.492916	0.595743	0.449417	0.303272	0.377829	0.588074	0.488442	0.755294	
3	0.586570	0.843872	0.233501	0.102906	0.462201	0.472101	0.713713	0.333815	0.483032	
4	0.519192	0.243793	0.630986	0.489290	0.475016	0.217447	0.380016	0.434261	0.278969	

5 rows × 31 columns

In [41]:
```
df_cancerdata.describe(include='all')
```

Out[41]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	0.525857	0.521412	0.332935	0.216920	0.506398	0.532573	0.505944	0.474637	0.484899
std	0.150753	0.170606	0.167915	0.149274	0.139618	0.170385	0.168069	0.157481	0.169732
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.420788	0.404315	0.216847	0.117413	0.404198	0.413507	0.394804	0.368569	0.376968
50%	0.525236	0.525546	0.293345	0.172895	0.510015	0.529847	0.508883	0.474678	0.483738
75%	0.624441	0.636060	0.416765	0.271135	0.607539	0.653246	0.619451	0.580385	0.602891
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

11 rows × 31 columns

Convert diagnosis value of M and B to a numerical value

M is Malignant which is represented by 0 and B is Benign which is represented by 1.

In [42]:
```
# Change M & B
def diagnosis_value(diagnosis):
    if diagnosis == 'M':
        return 1
    else:
        return 0

df_cancerdata['diagnosis'] = df_cancerdata['diagnosis'].apply(diagnosis_value)
```

In [43]:
```
df_cancerdata.head(50)
```

Out[43]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	0.527948	0.565881	0.545989	0.363733	0.508690	0.476895	0.693971	0.653166	0.294959
1	0.715716	0.651330	0.615783	0.501591	0.491769	0.390740	0.494830	0.371905	0.699552
2	0.585505	0.492916	0.595743	0.449417	0.303272	0.377829	0.588074	0.488442	0.755294
3	0.586570	0.843872	0.233501	0.102906	0.462201	0.472101	0.713713	0.333815	0.483032
4	0.519192	0.243793	0.630986	0.489290	0.475016	0.217447	0.380016	0.434261	0.278969
5	0.522655	0.798713	0.267984	0.141506	0.719655	0.146535	0.410847	0.563557	0.784489
6	0.571590	0.706064	0.523875	0.380276	0.398886	0.626996	0.427816	0.294042	0.480780
7	0.586659	0.405768	0.320710	0.184263	0.612105	0.685787	0.811967	0.178938	0.638476
8	0.566706	0.374056	0.302052	0.159618	0.506316	0.335553	0.127329	0.256735	0.644776
9	0.569469	0.494724	0.277659	0.140997	0.654041	0.224150	0.375009	0.288319	0.579971
10	0.878983	0.653811	0.407090	0.277540	0.507958	0.432815	0.655944	0.290728	0.164954
11	0.512611	0.444024	0.413309	0.270414	0.628560	0.748897	0.602269	0.288996	0.554106
12	0.495430	0.608108	0.612328	0.415483	0.570176	0.465092	0.561435	0.469060	0.416325
13	0.531942	0.679141	0.414000	0.271135	0.535199	0.729738	0.538642	0.728881	0.539190
14	0.436008	0.704743	0.344206	0.184433	0.212626	0.570463	0.610210	0.553830	0.275358
15	0.506631	0.497272	0.365835	0.218579	0.489130	0.626220	0.689978	0.221403	0.571514
16	0.150760	0.444043	0.352083	0.229480	0.578570	0.607795	0.177211	0.594895	0.241159
17	0.538478	0.684226	0.444406	0.277964	0.498296	0.375494	0.418185	0.541998	0.182928
18	0.531057	0.372989	0.595743	0.473595	0.520839	0.380559	0.306620	0.418053	0.467527
19	0.697308	0.454559	0.301776	0.179343	0.492680	0.753163	0.300198	0.409215	0.585597
20	0.641585	0.543426	0.289130	0.159703	0.494980	0.416731	0.595805	0.419210	0.283631
21	0.821539	0.638234	0.114367	0.055313	0.585851	0.571326	0.511215	0.464459	0.428932
22	0.461778	0.696266	0.405708	0.237922	0.680645	0.783947	0.271852	0.364355	0.496845
23	0.539819	0.486157	0.645498	0.534677	0.253047	0.726848	0.125234	0.627945	0.027557
24	0.453097	0.473585	0.457536	0.322842	0.402705	0.841202	0.440544	0.663101	0.476801
25	0.571485	0.663097	0.498998	0.326278	0.394470	0.425810	0.617891	0.158500	0.818403
26	0.416726	0.173173	0.370534	0.212641	0.264205	0.597107	0.443558	0.270488	0.341874
27	0.274680	0.000000	0.541151	0.403181	0.335722	0.533039	0.729788	0.249844	0.437913
28	0.358609	0.857780	0.405017	0.249799	0.575438	0.297884	0.387216	0.090641	0.392165
29	0.958356	0.547226	0.492088	0.344263	0.632098	0.430046	0.613545	0.457937	0.703225
30	0.658706	0.264611	0.559809	0.400636	0.340799	0.908479	0.581080	0.439220	0.579951
31	0.617035	0.490790	0.235920	0.126023	0.691201	0.326300	0.759892	0.564923	0.333870

32	0.154744	0.525357	0.476885	0.320594	0.495827	0.536690	0.780399	0.391806	0.537381
33	0.665740	0.477528	0.581231	0.432025	0.392679	0.407051	0.266024	0.303527	0.515817
34	0.527211	0.336355	0.436805	0.281527	0.631140	0.428375	0.500995	0.522344	0.759436
35	0.661673	0.300154	0.458227	0.307953	0.362875	0.526954	0.329260	0.227828	0.423082
36	0.771508	0.367137	0.344413	0.207635	0.435108	0.696685	0.580226	0.500545	0.684596
37	0.489444	0.224413	0.268261	0.161315	0.399542	0.476626	0.489650	0.611967	0.376863
38	0.303945	0.404315	0.357612	0.235546	0.527577	0.703708	0.408887	0.235805	0.377017
39	0.389455	0.255141	0.308272	0.176331	0.794864	0.420744	0.656437	0.616858	0.452739
40	0.252044	0.477114	0.292931	0.177943	0.754433	0.683256	0.425051	0.479860	0.592348
41	0.601865	0.596545	0.194251	0.096543	0.507534	0.365280	0.581753	0.686446	0.480441
42	0.575510	0.489303	0.583996	0.407423	0.689980	0.296334	0.592376	0.561980	0.278468
43	0.552053	0.593954	0.300809	0.170392	0.320523	0.449883	0.364079	0.527779	0.370585
44	0.351122	0.521356	0.287679	0.164581	0.447490	0.548927	0.484525	0.493720	0.540800
45	0.392167	0.397096	0.552208	0.395546	0.458012	0.487397	0.693132	0.400913	0.425086
46	0.525236	0.492053	0.054730	0.024772	0.477504	0.528133	0.925876	0.434130	0.332262
47	0.411762	0.618585	0.291549	0.165896	0.514186	0.695569	0.693542	0.482577	0.182492
48	0.529693	0.535918	0.236680	0.129714	0.501088	0.739524	0.722783	0.613147	0.583120
49	0.464768	0.895224	0.297975	0.177094	0.471975	0.535042	0.442209	0.769250	0.742342

Splitting the data: training and testing

We, split the data in training and testing for model prediction and building, we split the data in 70:30 ratio. 70% to training and 30% to testing.

In [44]:
```
# Train-Test-Split
from sklearn.model_selection import train_test_split
X = df_cancerdata.drop(['diagnosis'], axis=1)
Y = df_cancerdata ['diagnosis']
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size = 0.33,random_state = 42)
```

KNN implementation

Now we are fitting KNN algorithm on training data, predicting labels for dataset and printing the accuracy of the model for different values of K. As per our understanding the Accuracy is high when we select the K-value as 11 with 95.744 %.

In [45]:
```
# Create-KNN-model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
for K in range(25):
    K_value = K+1
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='distance', algorithm='auto')
    neigh.fit(x_train, y_train)
    y_pred = neigh.predict(x_test)
    print ("Accuracy is ", accuracy_score(y_test,y_pred)*100,"% for K-Value:",K_value)
```

```
Accuracy is  87.2340425531915 % for K-Value: 1
Accuracy is  87.2340425531915 % for K-Value: 2
Accuracy is  92.5531914893617 % for K-Value: 3
Accuracy is  92.5531914893617 % for K-Value: 4
Accuracy is  95.2127659574468 % for K-Value: 5
Accuracy is  94.14893617021278 % for K-Value: 6
Accuracy is  94.68085106382979 % for K-Value: 7
Accuracy is  95.2127659574468 % for K-Value: 8
Accuracy is  94.68085106382979 % for K-Value: 9
Accuracy is  95.74468085106383 % for K-Value: 10
Accuracy is  95.74468085106383 % for K-Value: 11
Accuracy is  95.2127659574468 % for K-Value: 12
Accuracy is  94.68085106382979 % for K-Value: 13
Accuracy is  94.14893617021278 % for K-Value: 14
Accuracy is  93.61702127659575 % for K-Value: 15
Accuracy is  94.14893617021278 % for K-Value: 16
Accuracy is  94.14893617021278 % for K-Value: 17
Accuracy is  94.68085106382979 % for K-Value: 18
Accuracy is  93.61702127659575 % for K-Value: 19
Accuracy is  93.61702127659575 % for K-Value: 20
Accuracy is  93.61702127659575 % for K-Value: 21
Accuracy is  94.14893617021278 % for K-Value: 22
Accuracy is  94.14893617021278 % for K-Value: 23
Accuracy is  94.14893617021278 % for K-Value: 24
Accuracy is  94.14893617021278 % for K-Value: 25
```

In [46]:
```
?KNeighborsClassifier
```

It shows that we are getting highest accuracy 97.87 on k=11

In [48]:
```
neigh.fit(x_train, y_train)
```
Out[48]:
```
KNeighborsClassifier(n_neighbors=11, weights='distance')
```

In [49]:
```
y_pred = neigh.predict(x_test)

y_pred
```
Out[49]:
```
array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0], dtype=int64)
```

In [50]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```
```
Accuracy: 0.9574468085106383
```

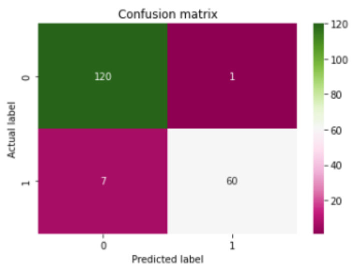
In [51]:
```
#import confusion_matrix
from sklearn.metrics import confusion_matrix
#let us get the predictions using the classifier we had fit above

confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=[' Predicted'], margins=True)
```
Out[51]:

Predicted	0	1	All
Actual			
0	120	1	121
1	7	60	67
All	127	61	188

Based on our results we conclude that only one person will die with breast cancer.

In [52]:
```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="PiYG" ,fmt='g')#PiYG  #YlGnBu
plt.title('Confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')#YlGnBu
```
Out[52]:
```
Text(0.5, 15.0, 'Predicted label')
```



In [53]:
```
?sns.heatmap
```

In [54]:
```
?sns.heatmap
```

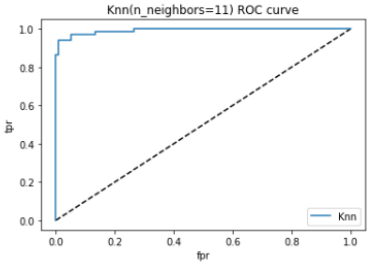
In [55]:
```
#import classification_report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```
```
              precision    recall  f1-score   support

           0       0.94      0.99      0.97       121
           1       0.98      0.90      0.94        67

    accuracy                           0.96       188
   macro avg       0.96      0.94      0.95       188
weighted avg       0.96      0.96      0.96       188
```

In [58]:
```
from sklearn.metrics import roc_curve
y_pred_proba = neigh.predict_proba(x_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [59]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=11) ROC curve')
plt.legend()
plt.show()
```



- We got the precision (positive predicted value) value of 94% for 0 and 98% for 1.
- Recall (sensitivity) value of 99% for 0 and 90% for 1.
- The black line (dash line) represents the 50% accuracy, the farther blue line represents the accuracy we predicted and it is close to 97.77%.
- We further, predict that out of 188 cases we were able to predict 180 correctly and conclude that the results for breast cancer is true (Malignant and Benign).