DIAAAAA

October 20, 2024

[13]:
```
from mlxtend.plotting import plot_decision_regions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

[14]:
```
diabetes_data = pd.read_csv('diabetes.csv')

#Print the first 5 rows of the dataframe.
diabetes_data.head()
```

[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

[15]:
```
diabetes_data.info(verbose=True)
```

```
<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   DiabetesPedigreeFunction  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
```

[16]:
```
diabetes_data.describe()
```

[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479
std	3.369578	31.972618	19.355807	15.952218	115.244002
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000
75%	6.000000	140.250000	80.000000	32.000000	127.250000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

[17]:
```
diabetes_data.describe().T
```

[17]:

	count	mean	std	min	25%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000
BMI	768.0	31.992578	7.884160	0.000	27.30000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375
Age	768.0	33.240885	11.760232	21.000	24.00000
Outcome	768.0	0.348958	0.476951	0.000	0.00000

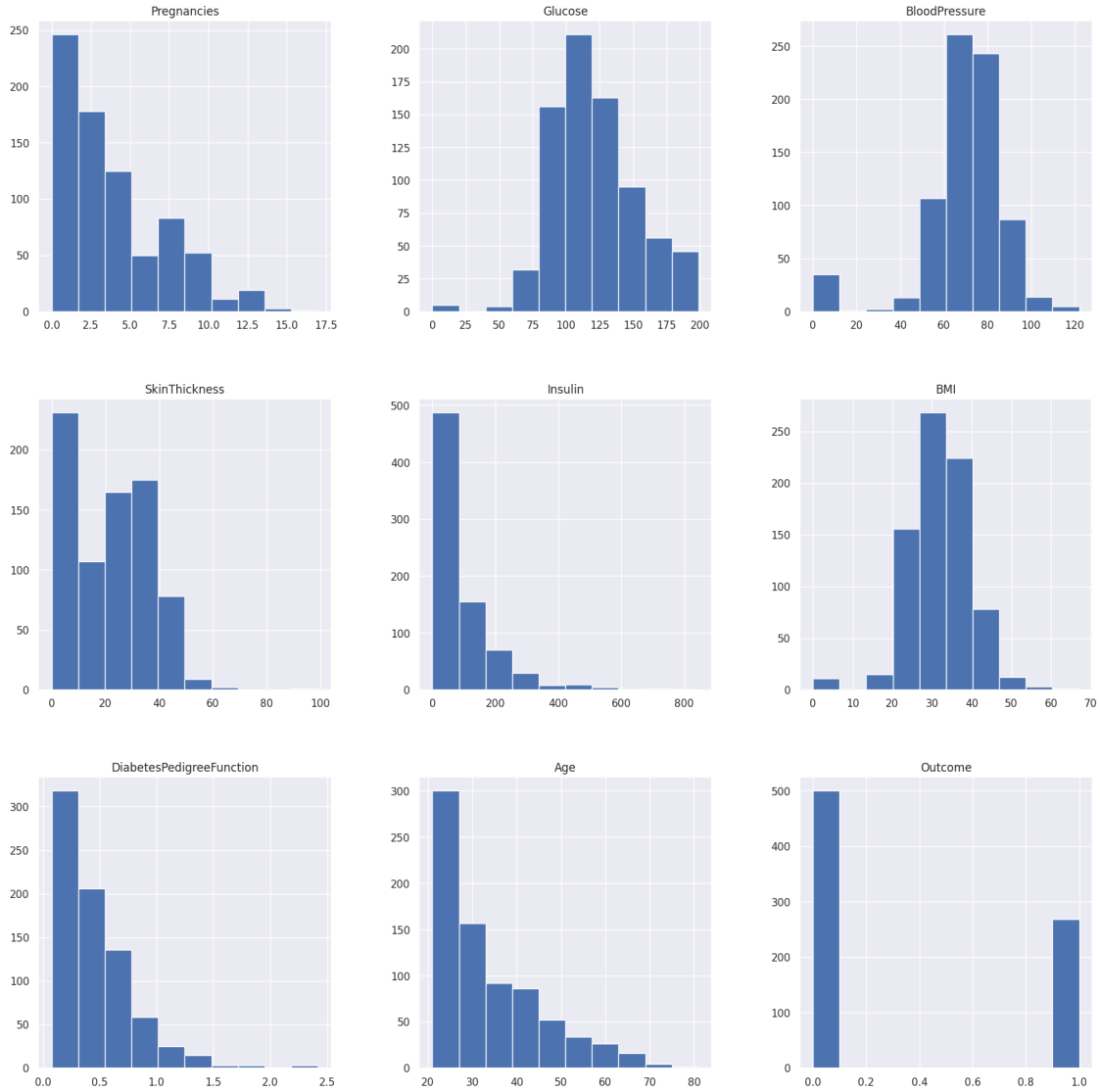
	50%	75%	max
Pregnancies	3.0000	6.00000	17.00

```
Glucose                    117.0000  140.25000  199.00
BloodPressure               72.0000   80.00000  122.00
SkinThickness               23.0000   32.00000   99.00
Insulin                     30.5000  127.25000  846.00
BMI                         32.0000   36.60000   67.10
DiabetesPedigreeFunction     0.3725    0.62625    2.42
Age                         29.0000   41.00000   81.00
Outcome                      0.0000    1.00000    1.00
```

[18]:
```
diabetes_data_copy = diabetes_data.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]␣
↪=␣
↪diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].
↪replace(0,np.NaN)

## showing the count of Nans
print(diabetes_data_copy.isnull().sum())
```

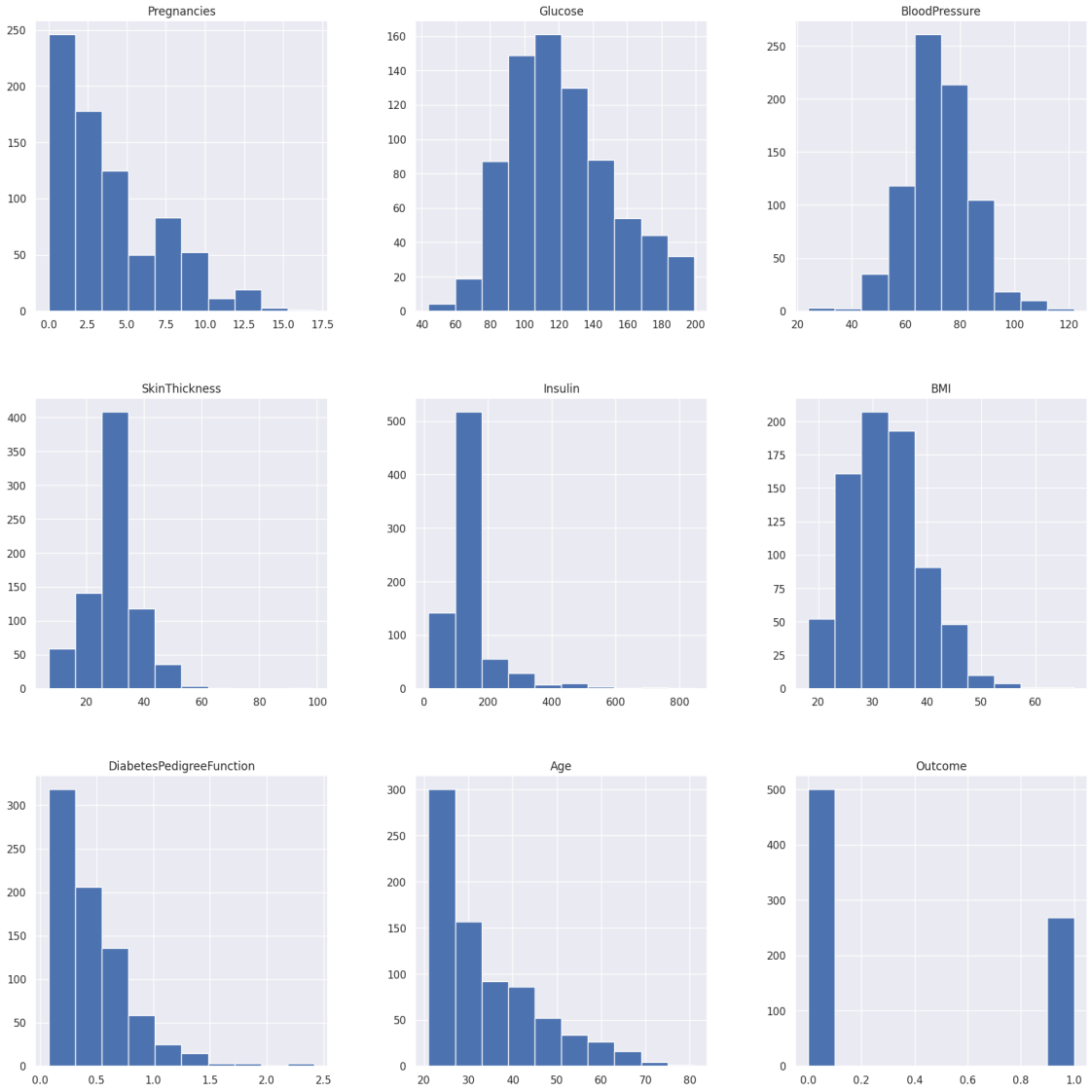
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[19]:
```
p = diabetes_data.hist(figsize = (20,20))
```



[20]:
```
diabetes_data_copy['Glucose'].fillna(diabetes_data_copy['Glucose'].mean(),␣
↪inplace = True)
diabetes_data_copy['BloodPressure'].fillna(diabetes_data_copy['BloodPressure'].
↪mean(), inplace = True)
diabetes_data_copy['SkinThickness'].fillna(diabetes_data_copy['SkinThickness'].
↪median(), inplace = True)
diabetes_data_copy['Insulin'].fillna(diabetes_data_copy['Insulin'].median(),␣
↪inplace = True)
diabetes_data_copy['BMI'].fillna(diabetes_data_copy['BMI'].median(), inplace =␣
↪True)
```

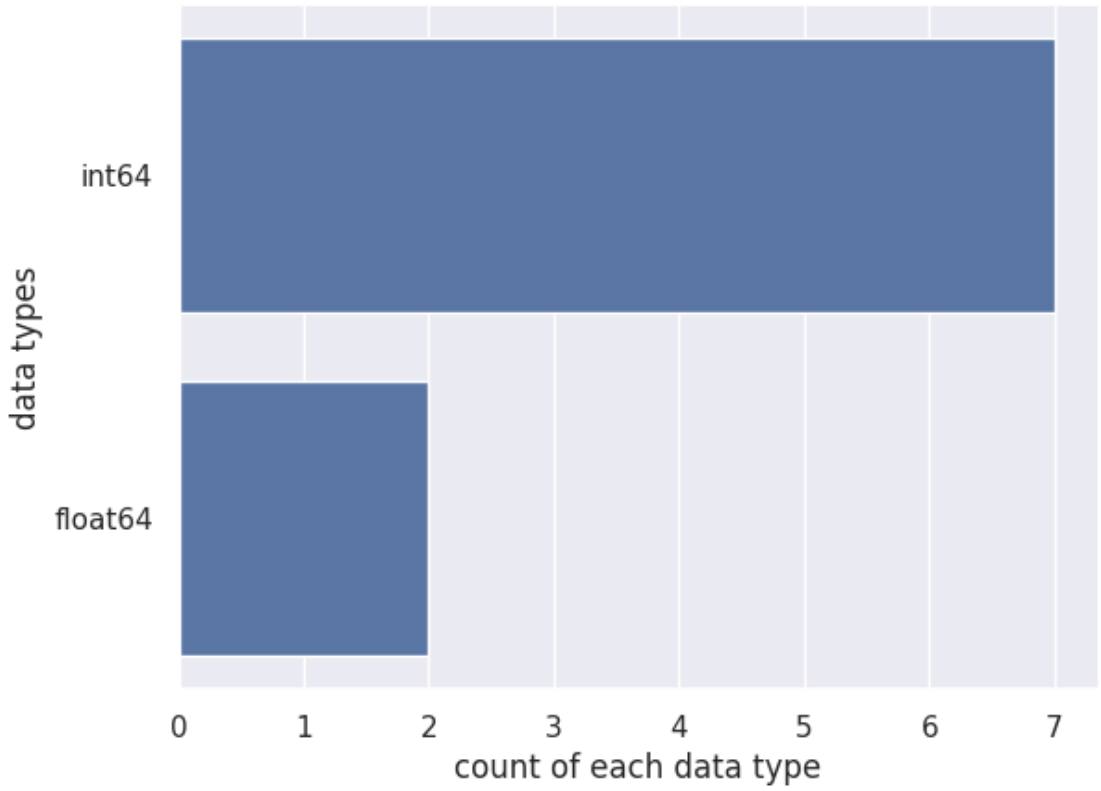
[29]:
```
p = diabetes_data_copy.hist(figsize = (20,20))
```



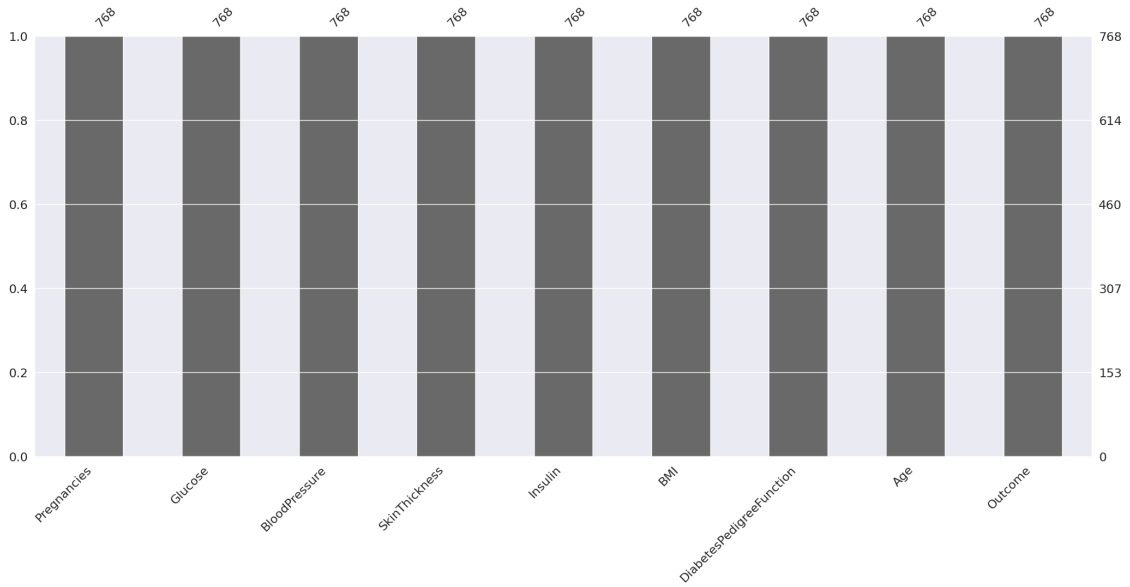
[28]:
```
diabetes_data.shape
```

[28]: (768, 9)

[27]:
```
sns.countplot(y=diabetes_data.dtypes ,data=diabetes_data)
plt.xlabel("count of each data type")
plt.ylabel("data types")
plt.show()
```

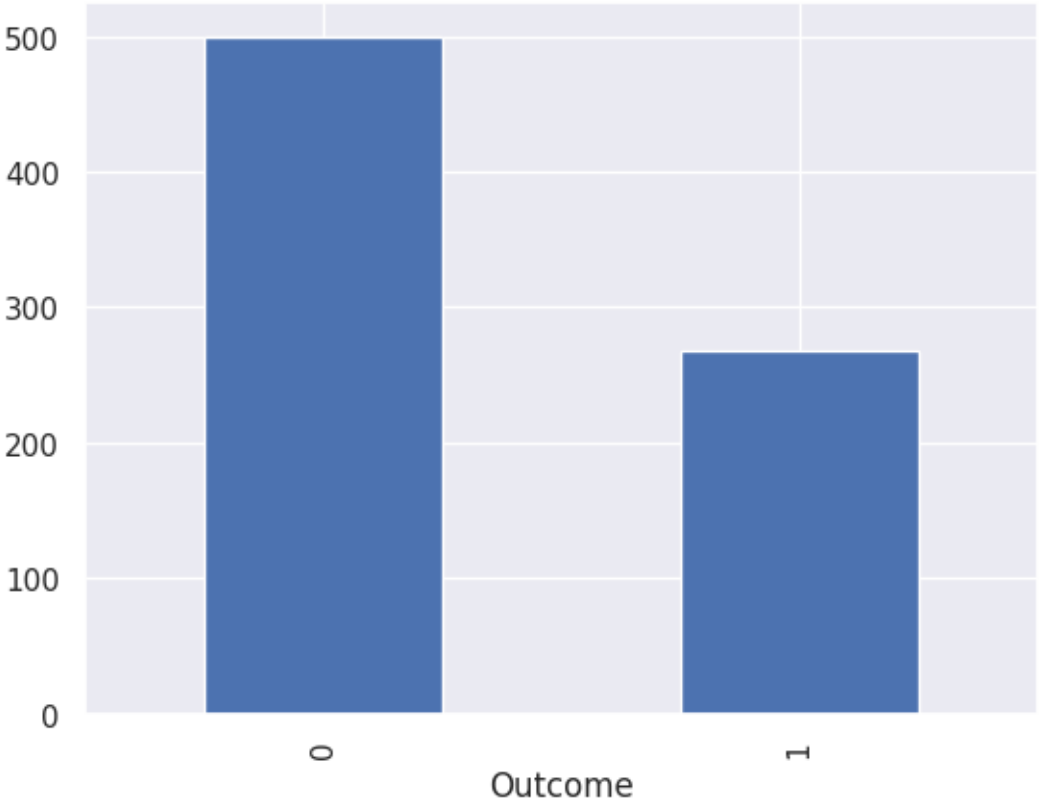


[30]:
```
import missingno as msno
p=msno.bar(diabetes_data)
```

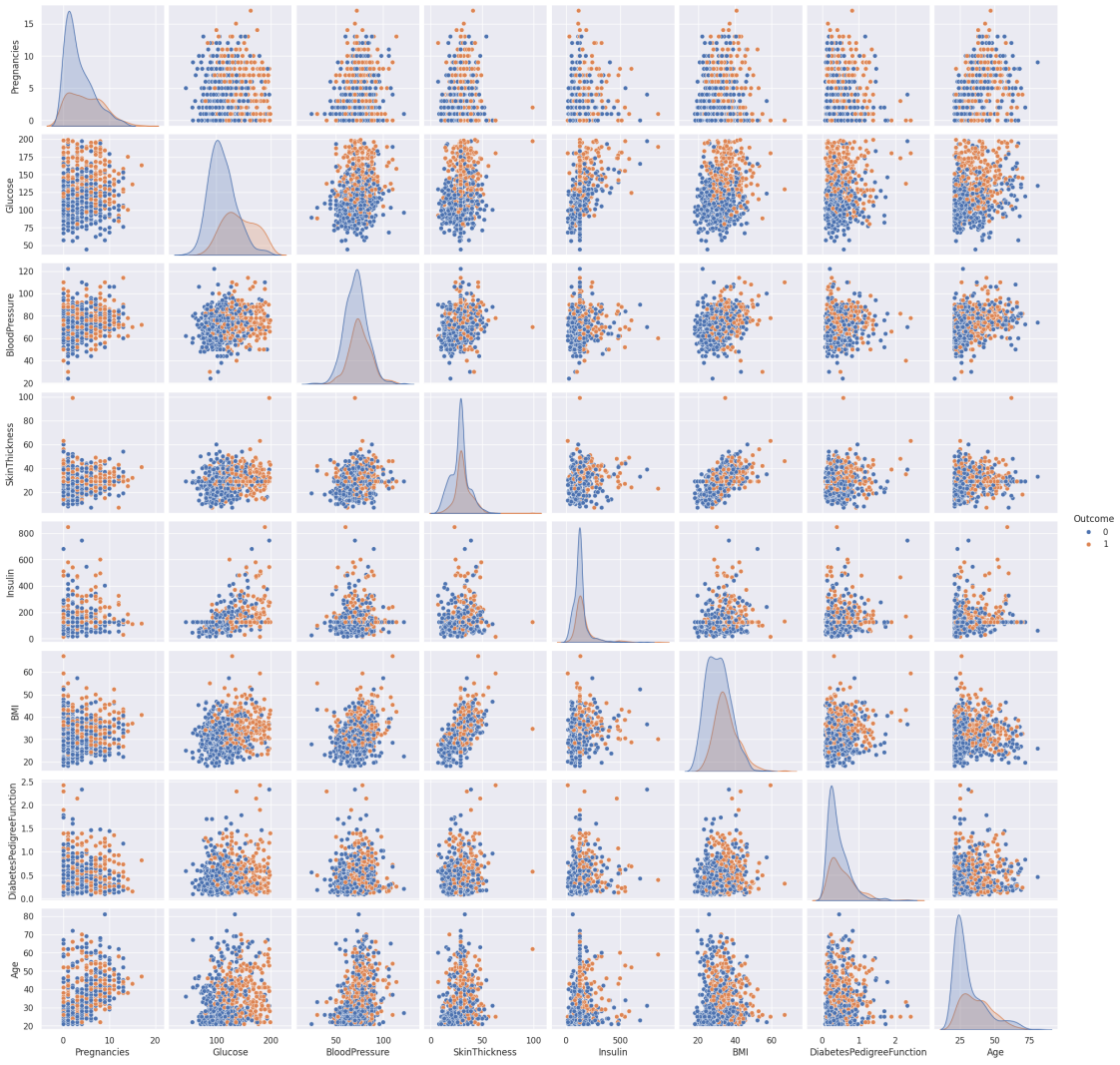


[31]:
```
color_wheel = {1: "#0392cf",
               2: "#7bc043"}
colors = diabetes_data["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_data.Outcome.value_counts())
p=diabetes_data.Outcome.value_counts().plot(kind="bar")
```

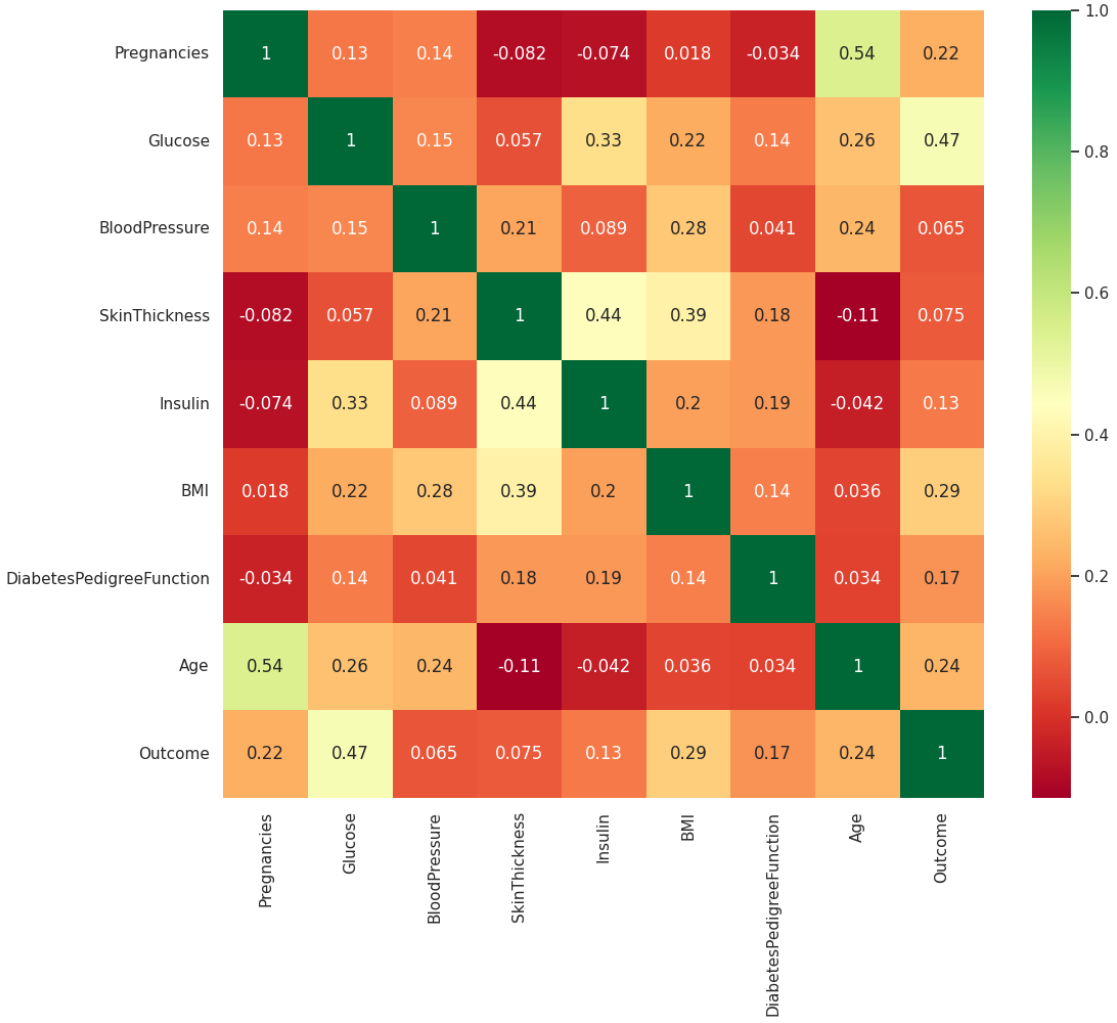
```
Outcome
0    500
1    268
Name: count, dtype: int64
```



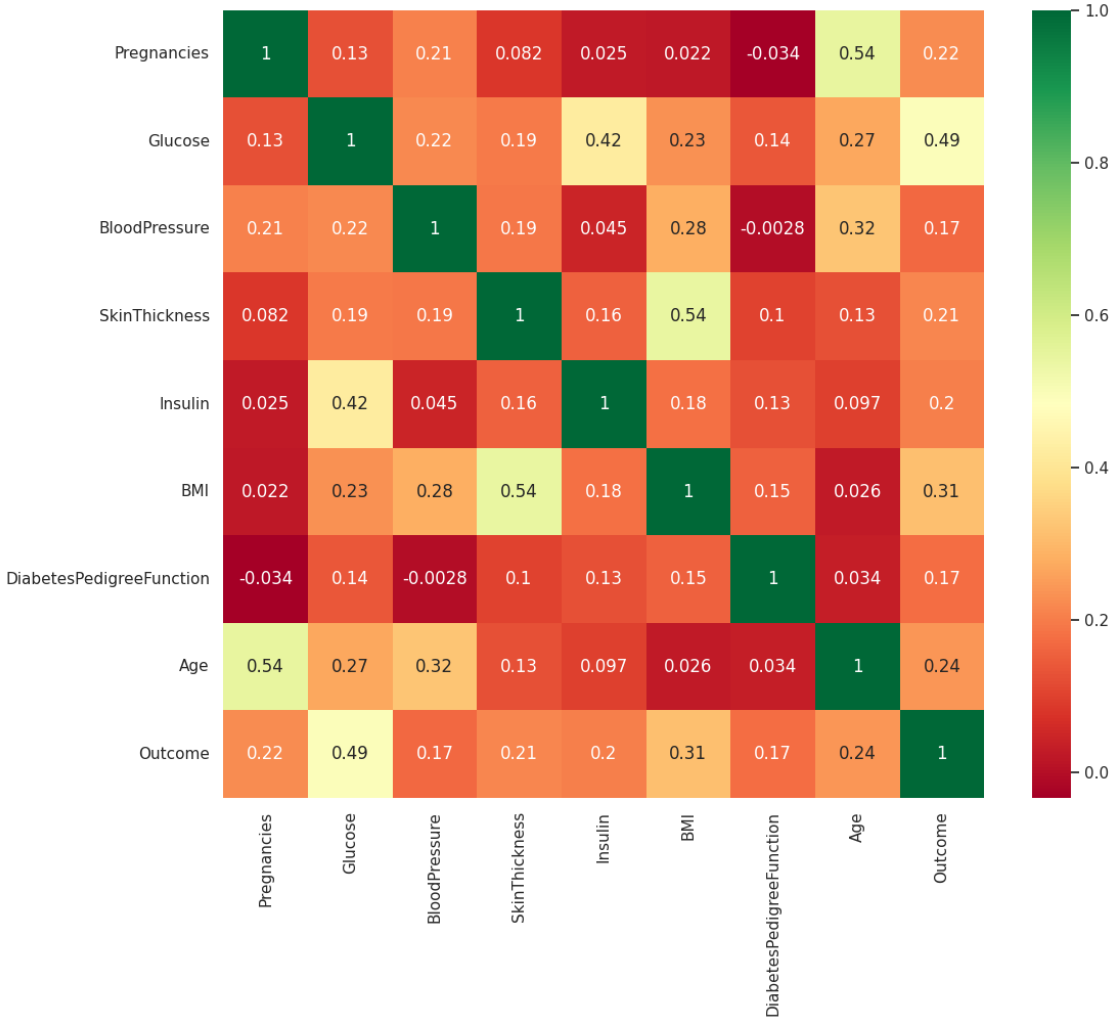
[32]:
```
p=sns.pairplot(diabetes_data_copy, hue = 'Outcome')
```



[33]:
```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12 by 10.
p=sns.heatmap(diabetes_data.corr(), annot=True,cmap ='RdYlGn')
```



[34]:
```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12 by 10.
p=sns.heatmap(diabetes_data_copy.corr(), annot=True,cmap ='RdYlGn')
```



[35]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_data_copy.drop(["Outcome"],axis =␣
↪1),),
       columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin',
      'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

[36]:
```
X.head()
```

[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	

```
4    -1.141852  0.503458      -2.680669       0.670643  0.316566  1.549303

   DiabetesPedigreeFunction       Age
0                  0.468492  1.425995
1                 -0.365061 -0.190672
2                  0.604397 -0.105584
3                 -0.920763 -1.041549
4                  5.484909 -0.020496
```

[37]:
```
y = diabetes_data_copy.Outcome
```

[38]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/
 ↪3,random_state=42, stratify=y)
```

[39]:
```
from sklearn.neighbors import KNeighborsClassifier


test_scores = []
train_scores = []

for i in range(1,15):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))
```

[40]:
```
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v ==␣
 ↪max_train_score]
print('Max train score {} % and k = {}'.
 ↪format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))
```

```
Max train score 100.0 % and k = [1]
```

[41]:
```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.
 ↪format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

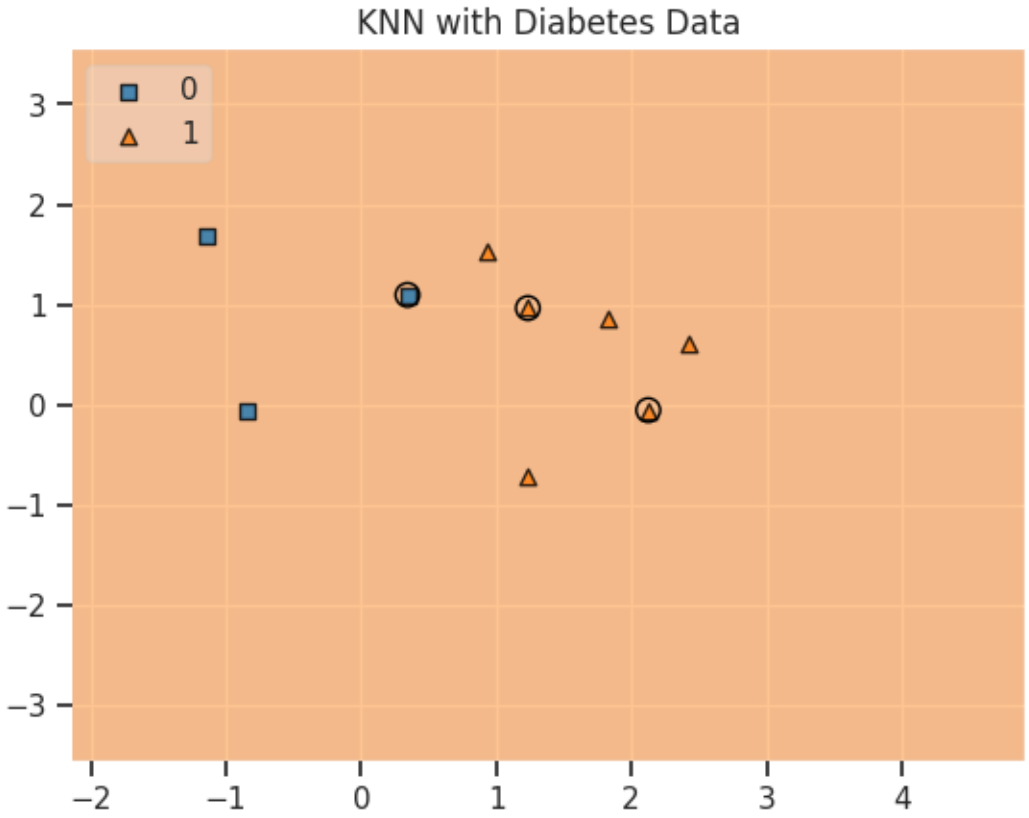
```
Max test score 76.5625 % and k = [11]
```

[42]:
```
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
```

```
knn.score(X_test,y_test)
```

[42]: 0.765625

[43]:
```
value = 20000
width = 20000
plot_decision_regions(X.values, y.values, clf=knn, legend=2,
                      filler_feature_values={2: value, 3: value, 4: value, 5:␣
 ↪value, 6: value, 7: value},
                      filler_feature_ranges={2: width, 3: width, 4: width, 5:␣
 ↪width, 6: width, 7: width},
                      X_highlight=X_test.values)

# Adding axes annotations
#plt.xlabel('sepal length [cm]')
#plt.ylabel('petal length [cm]')
plt.title('KNN with Diabetes Data')
plt.show()
```

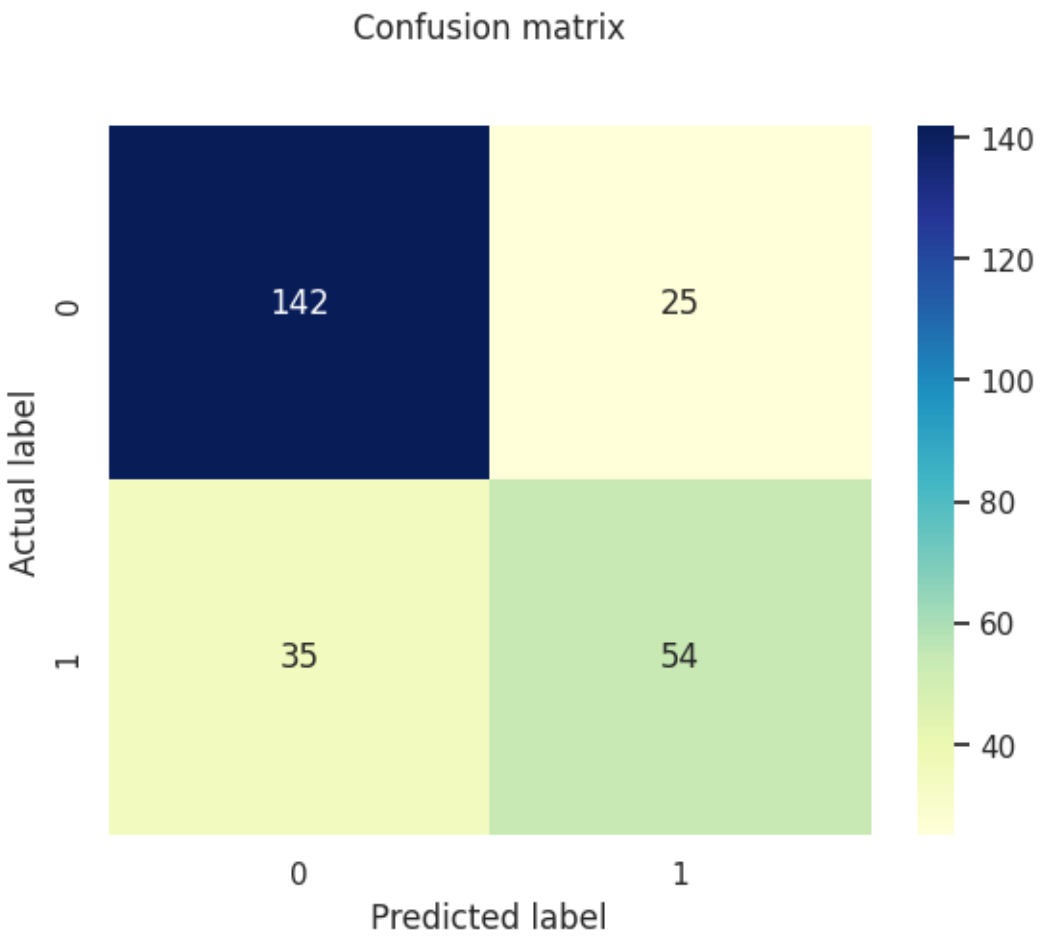


[44]:
```
from sklearn.metrics import confusion_matrix
#let us get the predictions using the classifier we had fit above
y_pred = knn.predict(X_test)
confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],␣
 ↪margins=True)
```

[44]:
```
Predicted    0   1  All
True
0          142  25  167
1           35  54   89
All        177  79  256
```

[45]:
```
y_pred = knn.predict(X_test)
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[45]:
```
Text(0.5, 19.049999999999997, 'Predicted label')
```

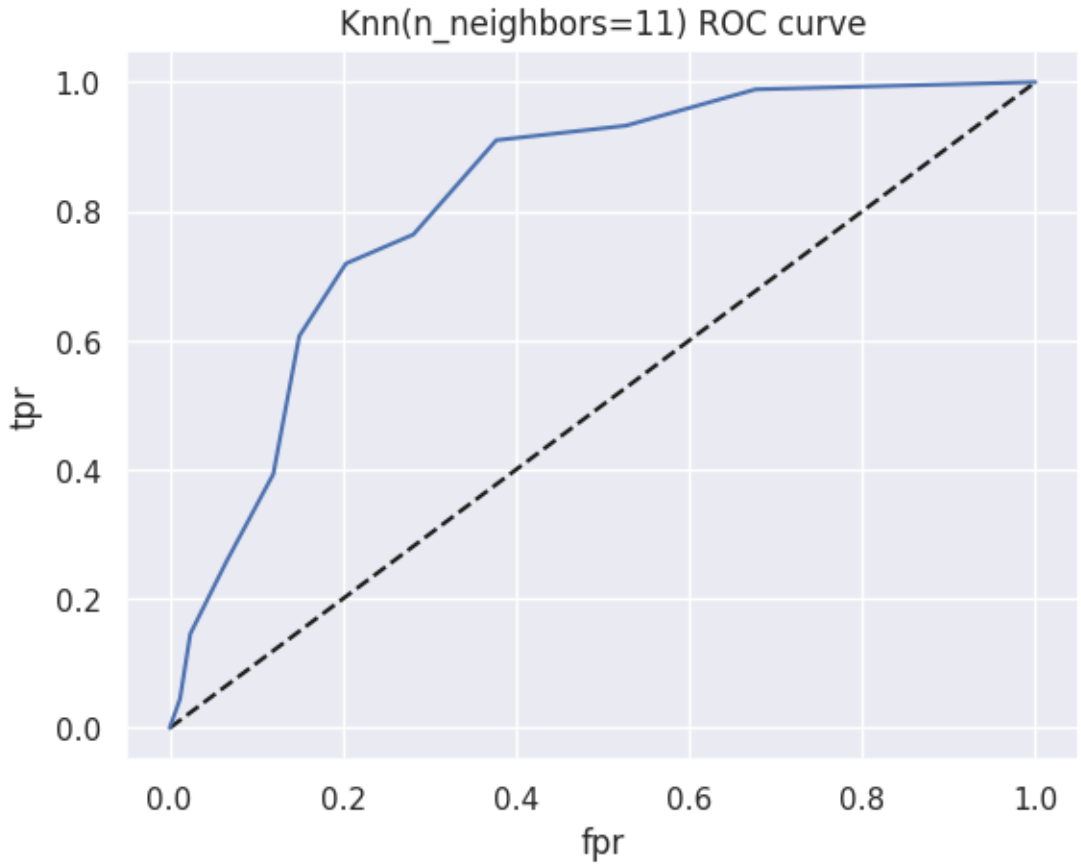


[46]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       167
           1       0.68      0.61      0.64        89

    accuracy                           0.77       256
   macro avg       0.74      0.73      0.73       256
weighted avg       0.76      0.77      0.76       256
```

[47]:
```
from sklearn.metrics import roc_curve
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

[48]:
```
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.show()
```



[49]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

[49]: 0.8193500639171096

[ ]:
```
from sklearn.model_selection import GridSearchCV
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)
```

```
print("Best Score:" + str(knn_cv.best_score_))
print("Best Parameters: " + str(knn_cv.best_params_))
```