Diabetes-classification

October 9, 2024

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
data = pd.read_csv("/home/pc13/Downloads/diabetes.csv")
data.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[3]:
```
data.isnull().any()
#used in Python with pandas to check for missing values in a DataFrame.
```

[3]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

[4]:
```
data.describe().T
#generate descriptive statistics for a DataFrame.
```

[4]:

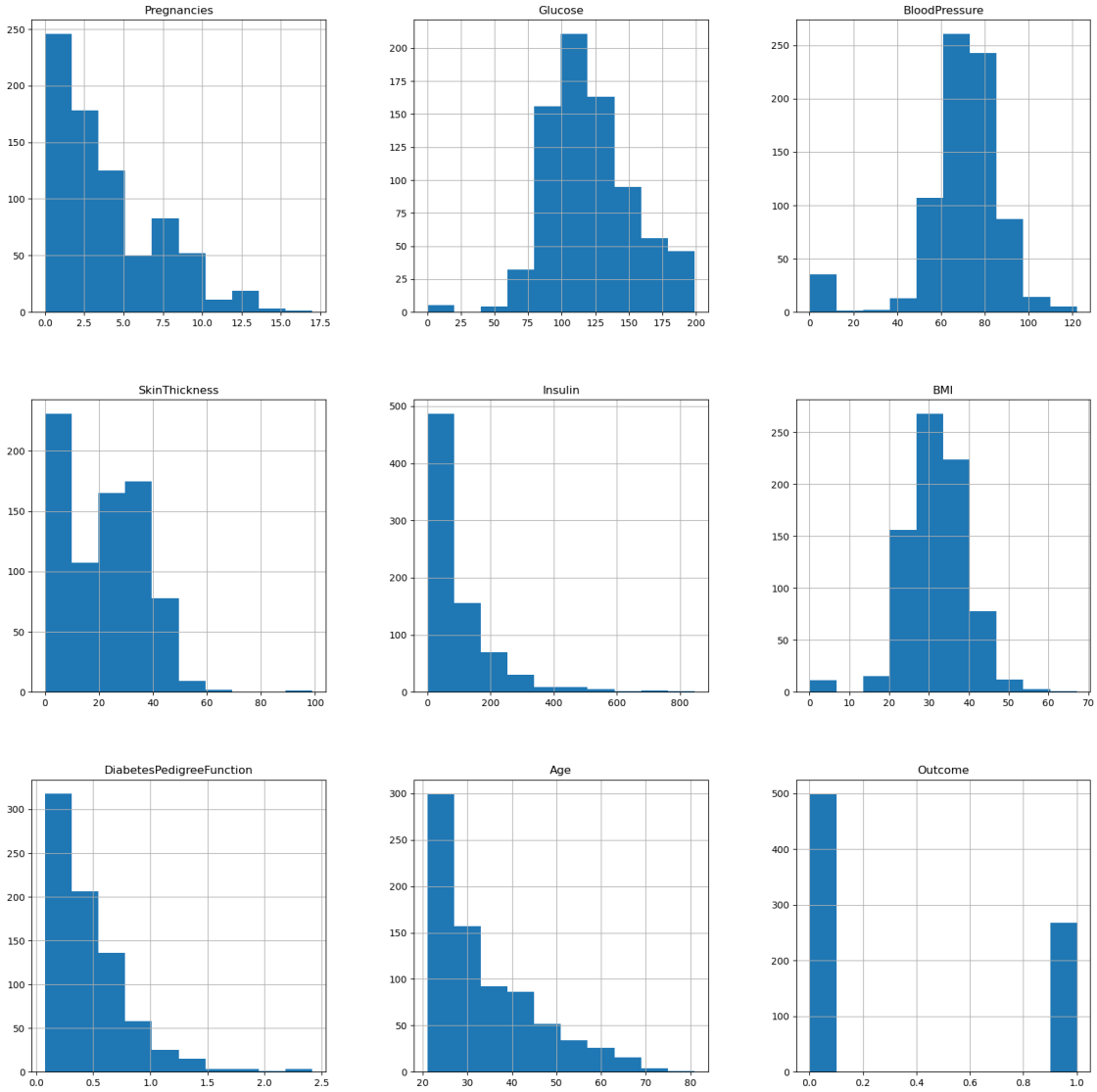
	count	mean	std	min	25%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000
BMI	768.0	31.992578	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

[5]:
```
data_copy = data.copy(deep = True)
data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =␣
 ↪data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].
 ↪replace(0,np.NaN)
data_copy.isnull().sum()
```

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

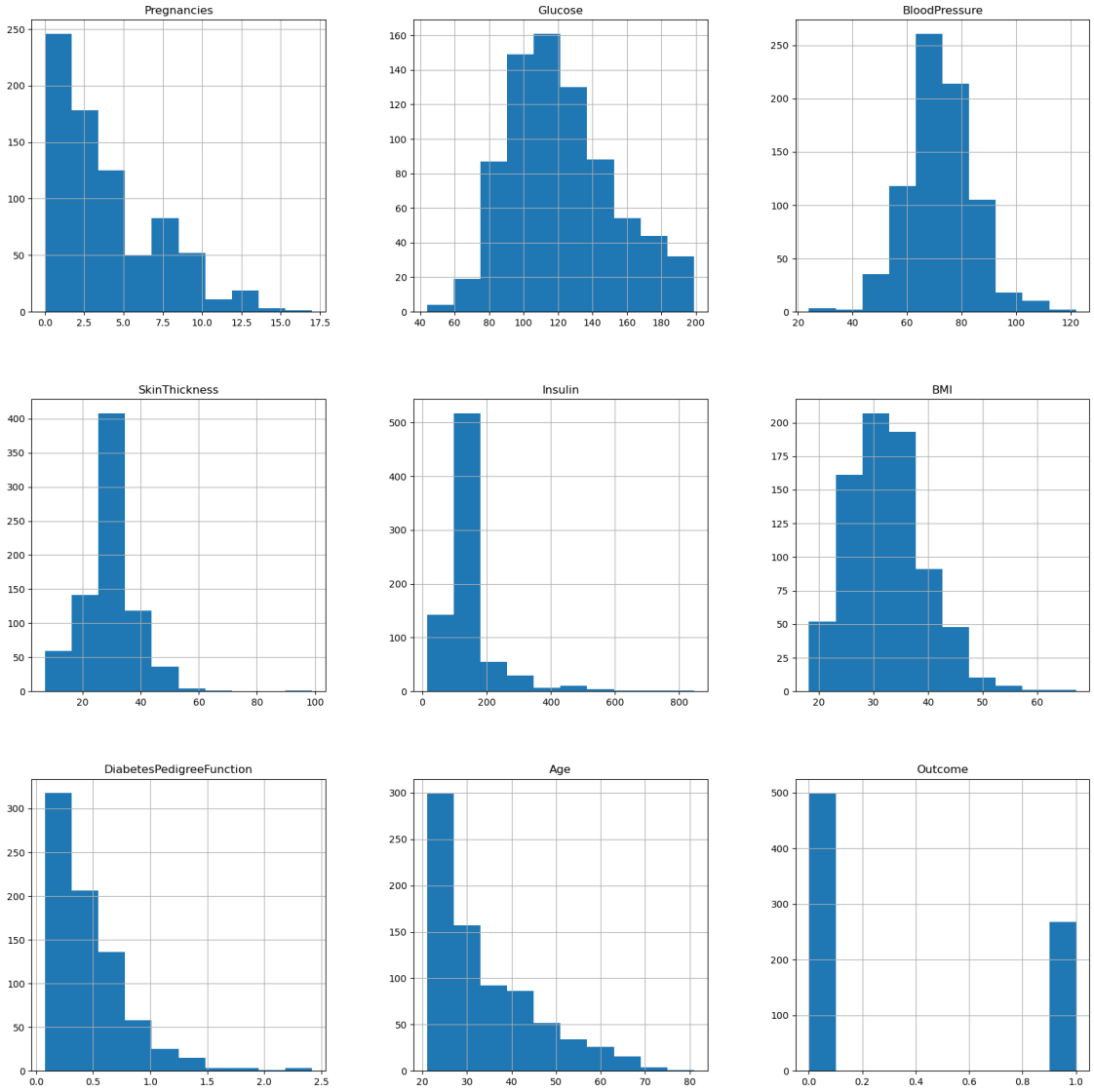
[6]:
```
#Create histograms for each numerical column in a pandas DataFrame.
p = data.hist(figsize = (20,20))
```



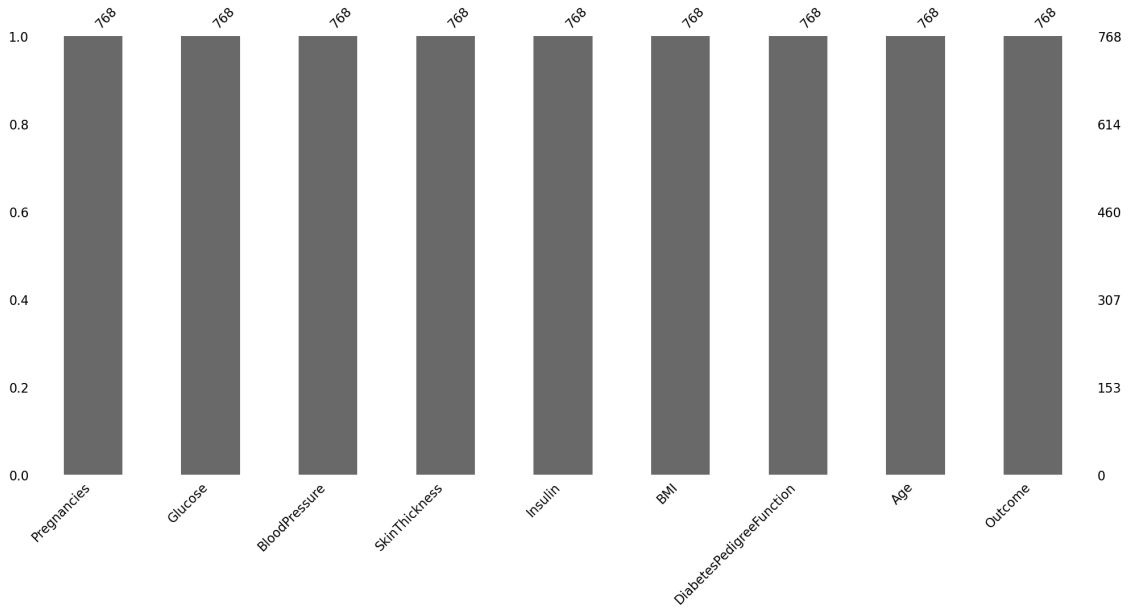
[7]:
```
# fill missing values in specific columns of a DataFrame (data_copy) using the␣
↪mean or median of those columns
```

[8]:
```
data_copy['Glucose'] = data_copy['Glucose'].fillna(data_copy['Glucose'].mean())
data_copy['BloodPressure'] = data_copy['BloodPressure'].
↪fillna(data_copy['BloodPressure'].mean())
data_copy['SkinThickness'] = data_copy['SkinThickness'].
↪fillna(data_copy['SkinThickness'].median())
data_copy['Insulin'] = data_copy['Insulin'].fillna(data_copy['Insulin'].
↪median())
data_copy['BMI'] = data_copy['BMI'].fillna(data_copy['BMI'].median())
```

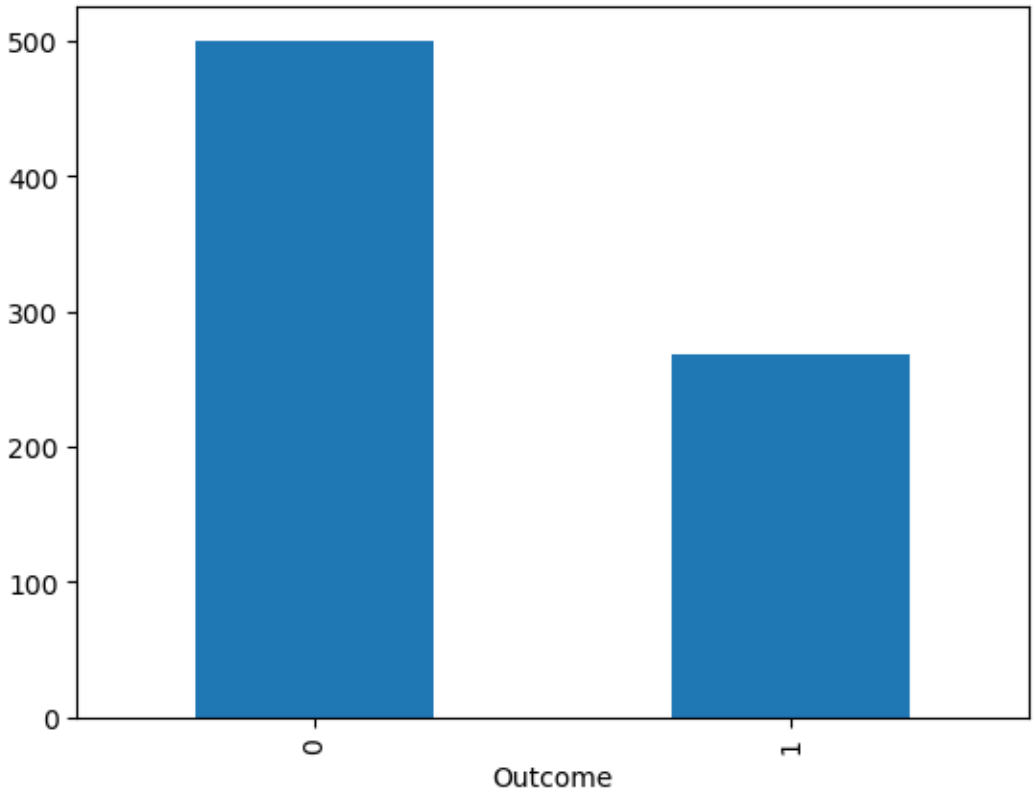
[9]:
```
p = data_copy.hist(figsize = (20,20))
```



[10]:
```
import missingno as msno
p = msno.bar(data)
```

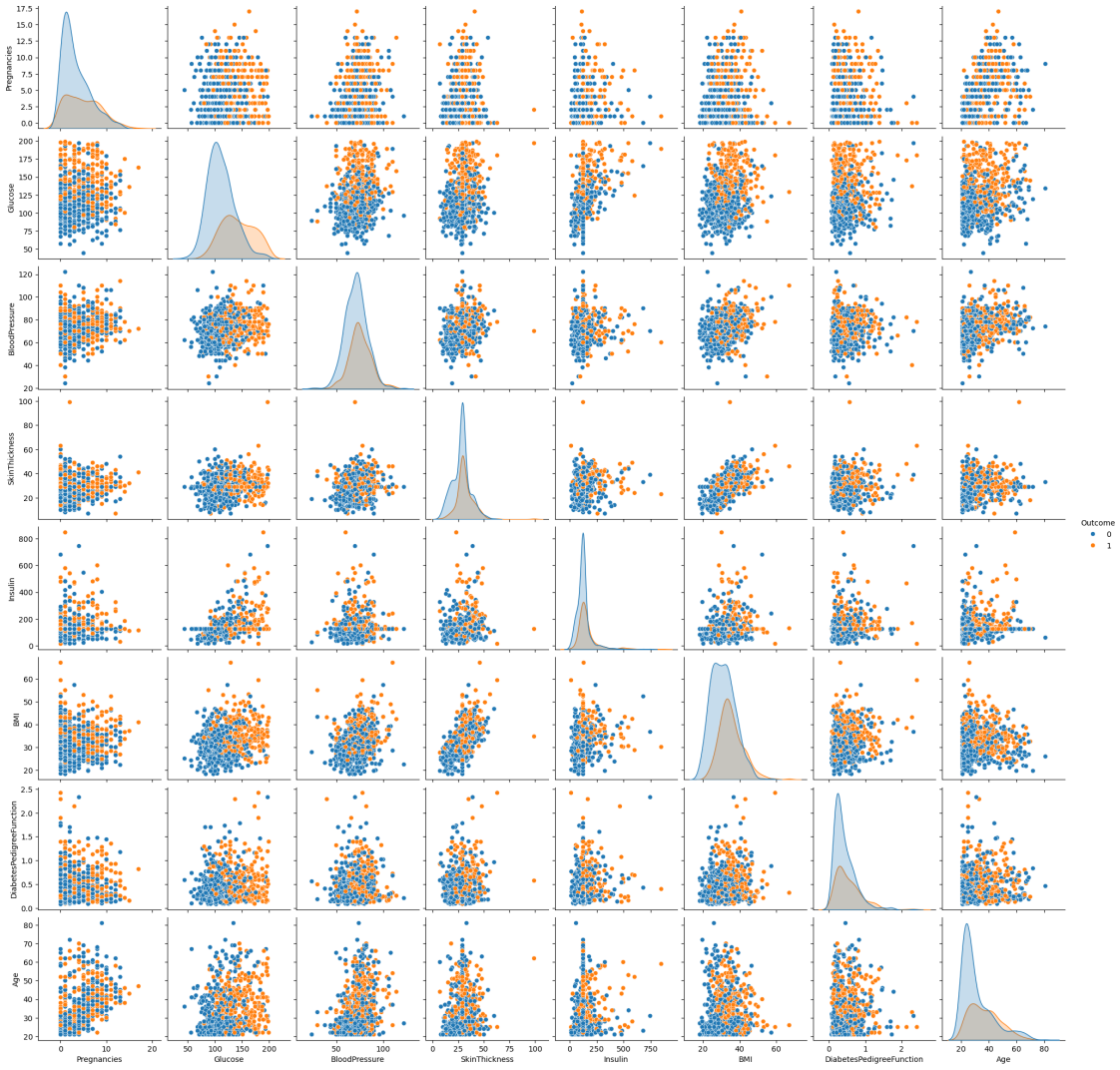


[11]:
```
p=data.Outcome.value_counts().plot(kind="bar")
```

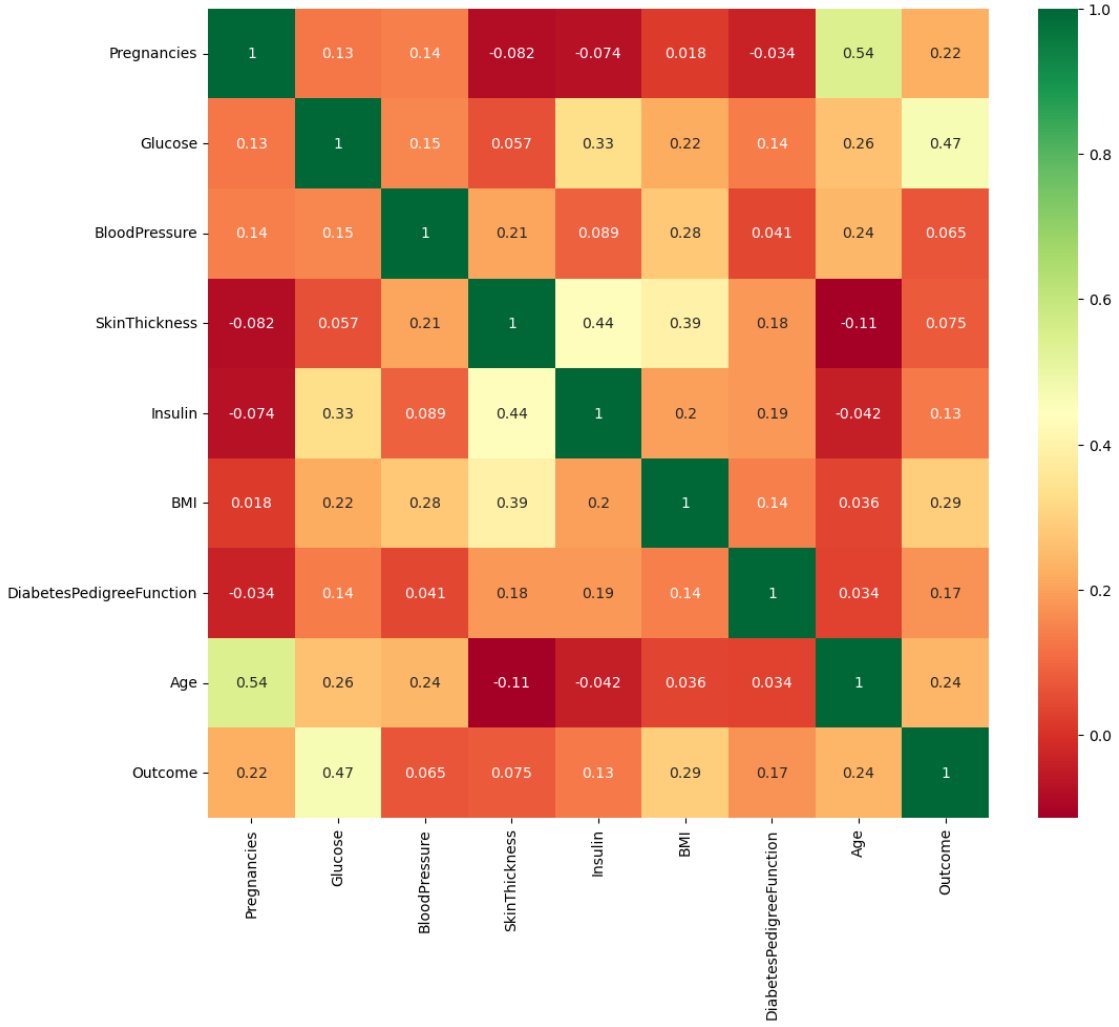


[12]:
```
#The above graph shows that the data is biased towards datapoints having␣
 ↪outcome value as 0 where it means that diabetes was not present actually.
#The number of non-diabetics is almost twice the number of diabetic patients
```

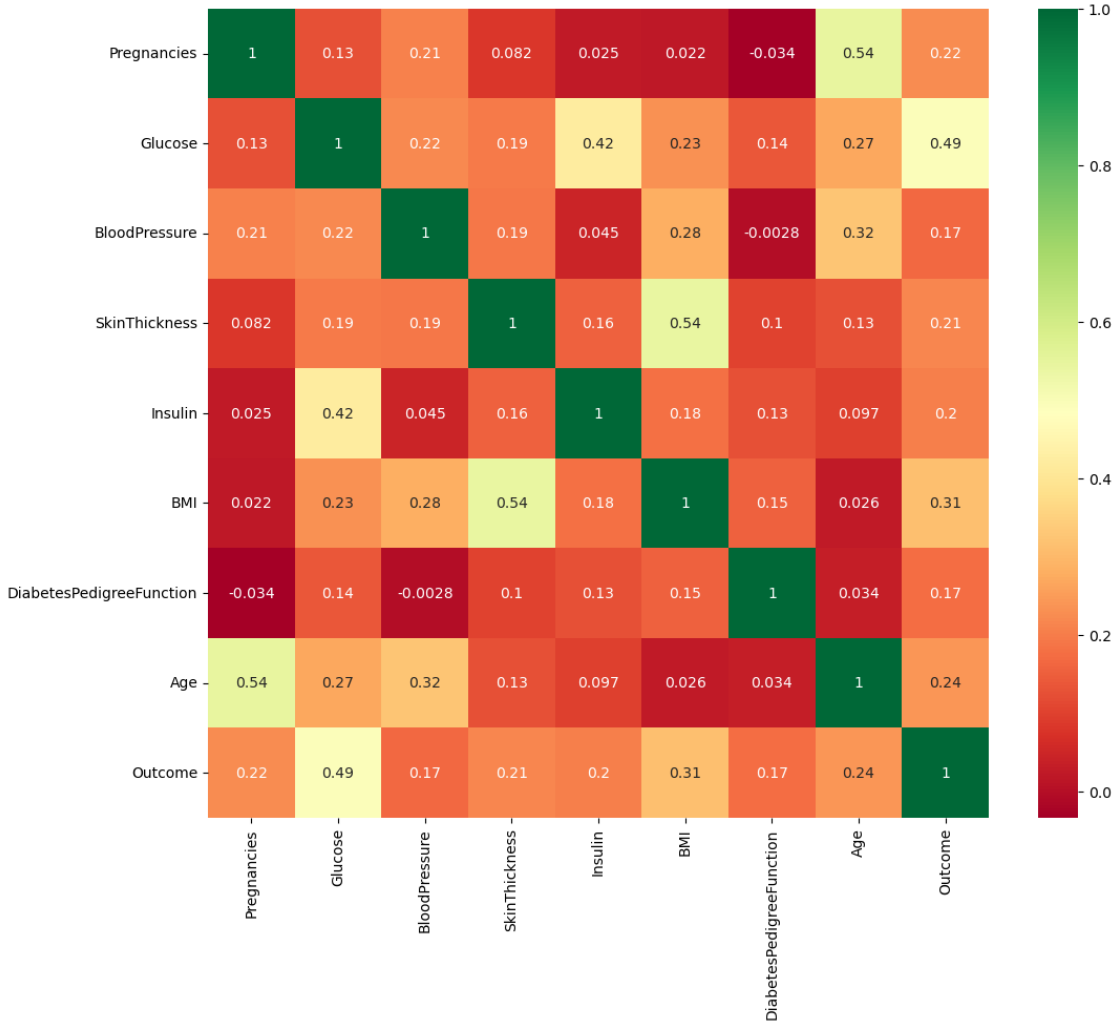
[15]:
```
import seaborn as sns
p=sns.pairplot(data_copy, hue = 'Outcome')
```



[16]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12␣
 ↪by 10.
p=sns.heatmap(data.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very␣
 ↪simple solution for heatmap
```



[17]:
```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12␣
↪by 10.
p=sns.heatmap(data_copy.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very␣
↪simple solution for heatmap
```



[19]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"], axis␣
 ↪=1),),columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin',
       'X.head()BMI', 'DiabetesPedigreeFunction', 'Age'])
```

[20]:
```
X.head()
```

[20]:

	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
```

[21]:
```
y =data_copy.Outcome
```

[22]:
```
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)
```

[23]:
```
from sklearn.neighbors import KNeighborsClassifier

train_scores = []
test_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))
```

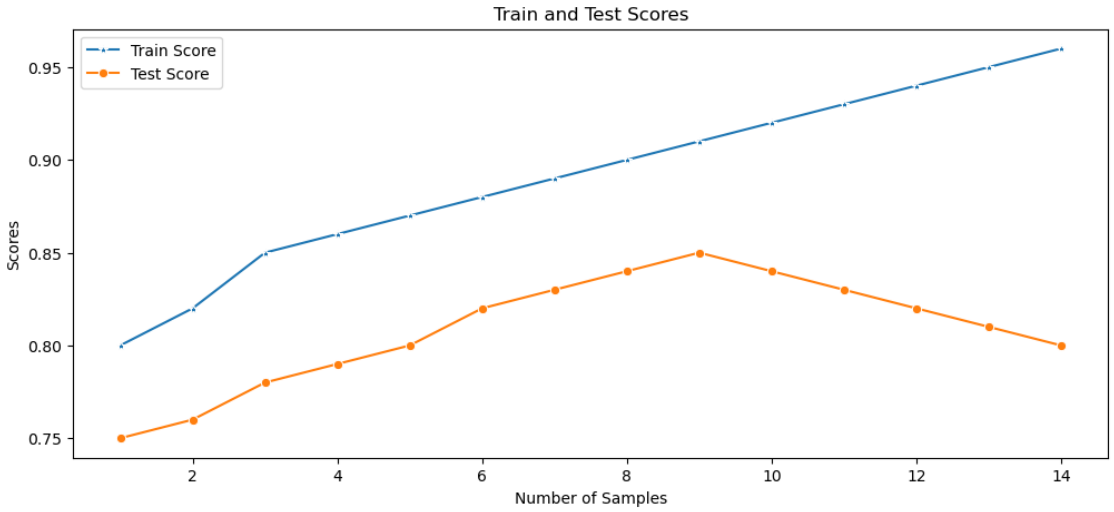
[24]:
```
max_test_score =max(test_scores)
```

[28]:
```
test_score_index = [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_score_index))))
```

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

[31]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Example data for train_scores and test_scores
train_scores = [0.8, 0.82, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93,␣
 ↪0.94, 0.95, 0.96]
test_scores = [0.75, 0.76, 0.78, 0.79, 0.8, 0.82, 0.83, 0.84, 0.85, 0.84, 0.83,␣
 ↪0.82, 0.81, 0.8]

# Create a figure with specified size
plt.figure(figsize=(12, 5))
```

```
# Use keyword arguments for x and y
p = sns.lineplot(x=range(1, 15), y=train_scores, marker='*', label='Train␣
↪Score')
p = sns.lineplot(x=range(1, 15), y=test_scores, marker='o', label='Test Score')

# Add titles and labels
plt.title('Train and Test Scores')
plt.xlabel('Number of Samples')
plt.ylabel('Scores')
plt.legend()

# Show the plot
plt.show()
```



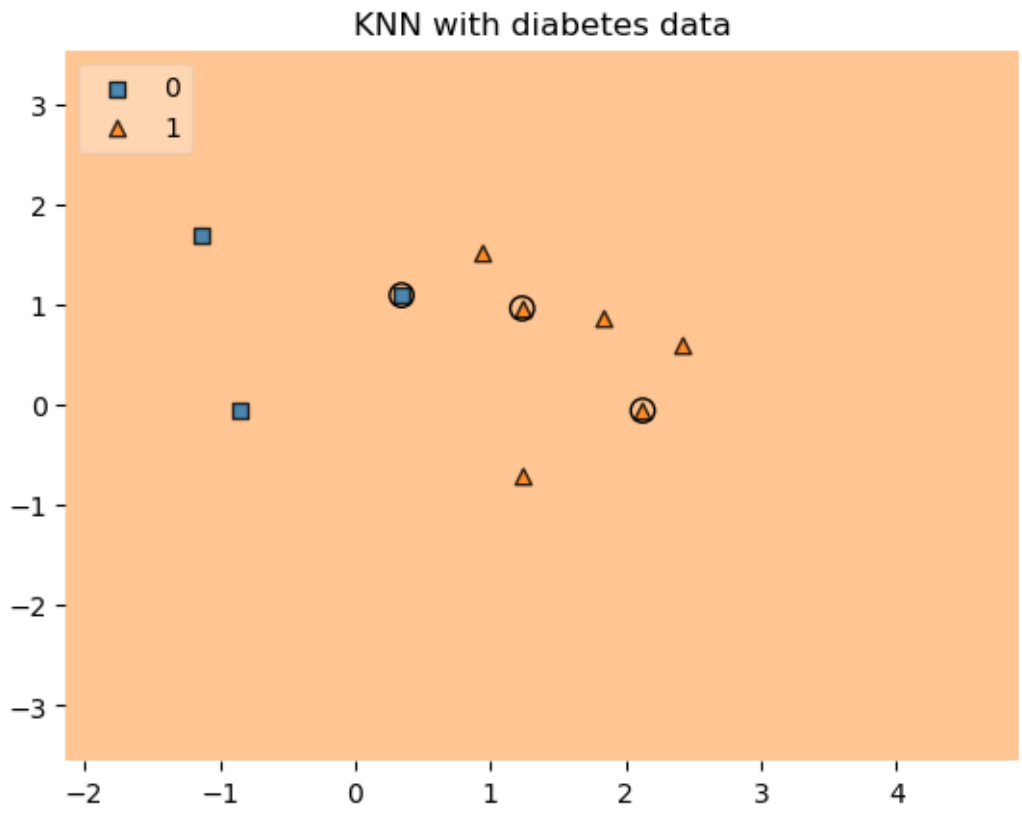
[32]:
```
# K=11
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

[32]: 0.765625

[36]:
```
from mlxtend.plotting import plot_decision_regions
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)
plt.title("KNN with diabetes data")
plt.show()
```

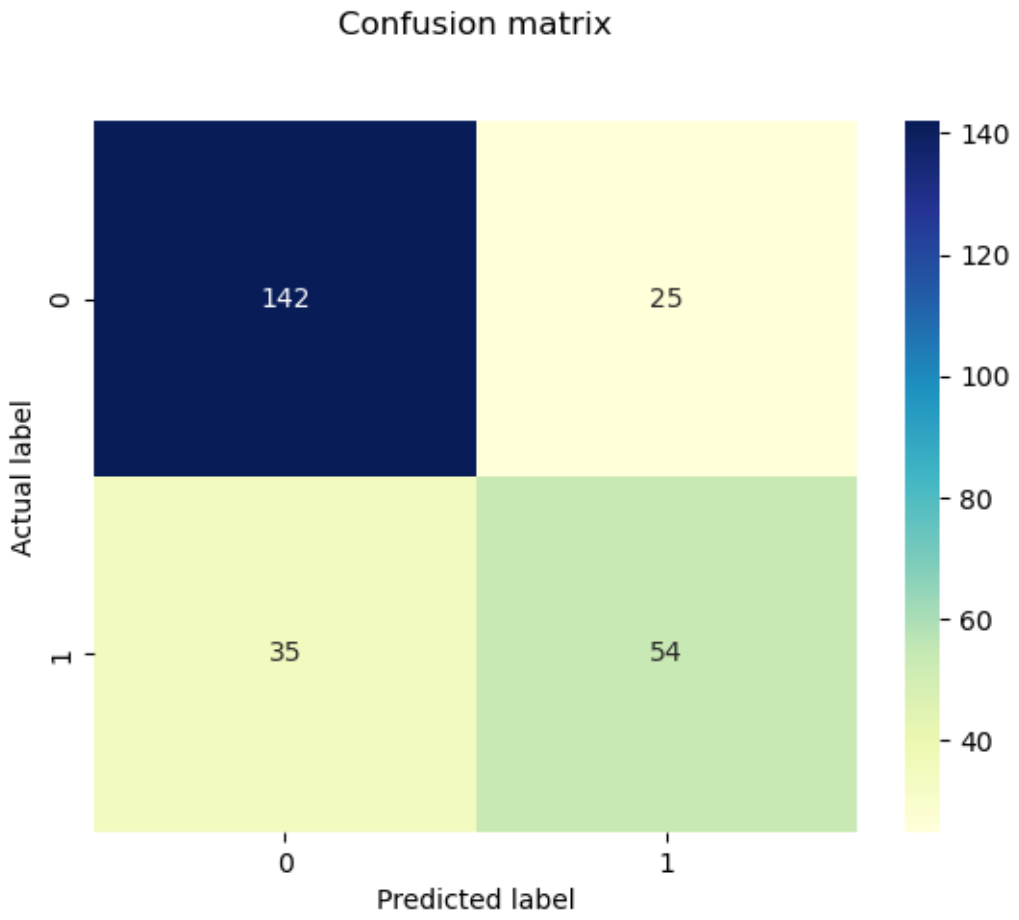
```
/home/pc13/miniconda3/lib/python3.12/site-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but KNeighborsClassifier was
fitted with feature names
  warnings.warn(
```



[37]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, fbeta_score
y_pred = knn.predict(X_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
```

[38]:
```
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')
```

[38]:
```
Text(0.5, 23.52222222222222, 'Predicted label')
```



[40]:
```
def model_evaluation(y_test, y_pred, model_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta = 2.0)

    results = pd.DataFrame([[model_name, acc, prec, rec, f1, f2]],
                      columns = ["Model", "Accuracy", "Precision", "Recall",
                                 "F1 SCore", "F2 Score"])
```

```
    results = results.sort_values(["Precision", "Recall", "F2 Score"],␣
 ↪ascending = False)
    return results



model_evaluation(y_test, y_pred, "KNN")# Alternate way
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
```

[40]:

	Model	Accuracy	Precision	Recall	F1 SCore	F2 Score
0	KNN	0.765625	0.683544	0.606742	0.642857	0.62069

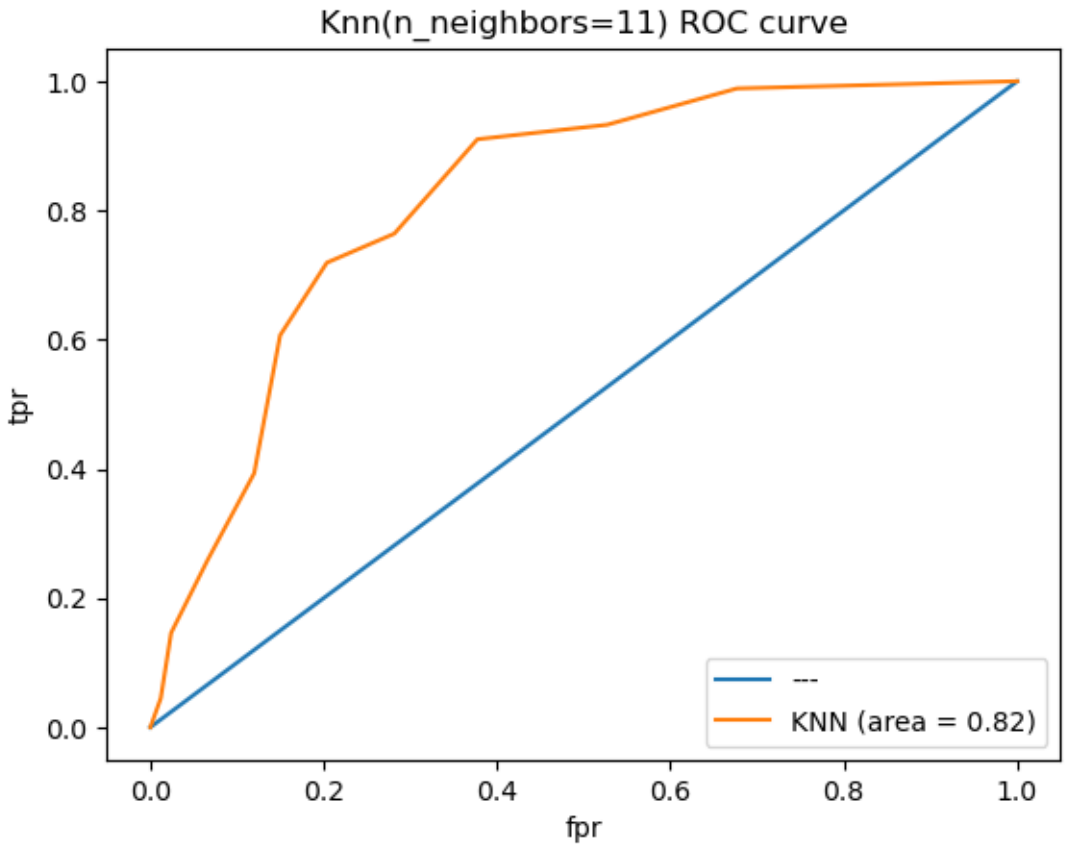
[43]:
```
# Alternate way
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              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
```

[44]:
```
from sklearn.metrics import auc, roc_auc_score, roc_curve

y_pred_proba = knn.predict_proba(X_test)[:,-1]
fpr, tpr, threshold = roc_curve(y_test, y_pred_proba)
```

[45]:
```
classifier_roc_auc = roc_auc_score(y_test, y_pred_proba)
plt.plot([0,1],[0,1], label = "---")

plt.plot(fpr, tpr, label ='KNN (area = %0.2f)' % classifier_roc_auc)
```

```
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title('Knn(n_neighbors=11) ROC curve')
plt.legend(loc="lower right", fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")
plt.show()
```



[46]:
```
#Hyper parameters tuning using GridSearchCV

from sklearn.model_selection import GridSearchCV
parameters_grid = {"n_neighbors": np.arange(0,50)}
knn= KNeighborsClassifier()
knn_GSV = GridSearchCV(knn, param_grid=parameters_grid, cv = 5)
knn_GSV.fit(X, y)
```

```
/home/pc13/miniconda3/lib/python3.12/site-
packages/sklearn/model_selection/_validation.py:540: FitFailedWarning:
5 fits failed out of a total of 250.
The score on these train-test partitions for these parameters will be set to
```

```
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/home/pc13/miniconda3/lib/python3.12/site-
packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/pc13/miniconda3/lib/python3.12/site-packages/sklearn/base.py",
line 1466, in wrapper
    estimator._validate_params()
  File "/home/pc13/miniconda3/lib/python3.12/site-packages/sklearn/base.py",
line 666, in _validate_params
    validate_parameter_constraints(
  File "/home/pc13/miniconda3/lib/python3.12/site-
packages/sklearn/utils/_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'n_neighbors'
parameter of KNeighborsClassifier must be an int in the range [1, inf) or None.
Got 0 instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/pc13/miniconda3/lib/python3.12/site-
packages/sklearn/model_selection/_search.py:1102: UserWarning: One or more of
the test scores are non-finite: [       nan 0.68759019 0.71362363 0.73312962
0.7369663  0.73441134
 0.73700025 0.74609965 0.74870554 0.75395977 0.74743231 0.76436635
 0.75787285 0.76699771 0.75524998 0.76306765 0.76828792 0.76698073
 0.77086835 0.76698073 0.76438333 0.76696376 0.76307614 0.76566505
 0.76176895 0.77218403 0.76566505 0.76958662 0.77088532 0.76568203
 0.76045327 0.76306765 0.76305916 0.76047874 0.7696036  0.76437484
 0.76046176 0.76047874 0.76046176 0.75526696 0.76176895 0.76438333
 0.75655717 0.75916306 0.75785587 0.75525847 0.75264409 0.75135387
 0.75265258 0.75136236]
  warnings.warn(
```

[46]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 0,  1,  2,  3,  4,  5,  6,  7,
 8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

[49]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Define the parameter grid
param_grid = {'n_neighbors': np.array(range(1, 50))}  # Change to range for␣
 ↪simplicity

# Create the GridSearchCV object
grid_search = GridSearchCV(cv=5, estimator=KNeighborsClassifier(),␣
 ↪param_grid=param_grid)

# Example data (X_train, y_train) should be defined here
# grid_search.fit(X_train, y_train)  # Uncomment this line when you have your␣
 ↪data

# Now you can proceed with fitting the grid search
```

[50]:
```
print("Best Params" ,knn_GSV.best_params_)
print("Best score" ,knn_GSV.best_score_)
```

```
Best Params {'n_neighbors': 25}
Best score 0.7721840251252015
```

[ ]:
```

```