diabetesreports

March 27, 2023

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[6]:
```
data = pd.read_csv('C:/Users/beker/OneDrive/Masaüstü/programming/dataprocessing/
↪diabetes.csv')
```

[7]:
```
data
```

[7]:
```
     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
..           …      …              …              …       …   …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     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
..                        …  …      …
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

[8]:
```
data.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   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
```

[9]:
```
data.isna().any()
```

[9]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

[10]:
```
data.describe().T
```

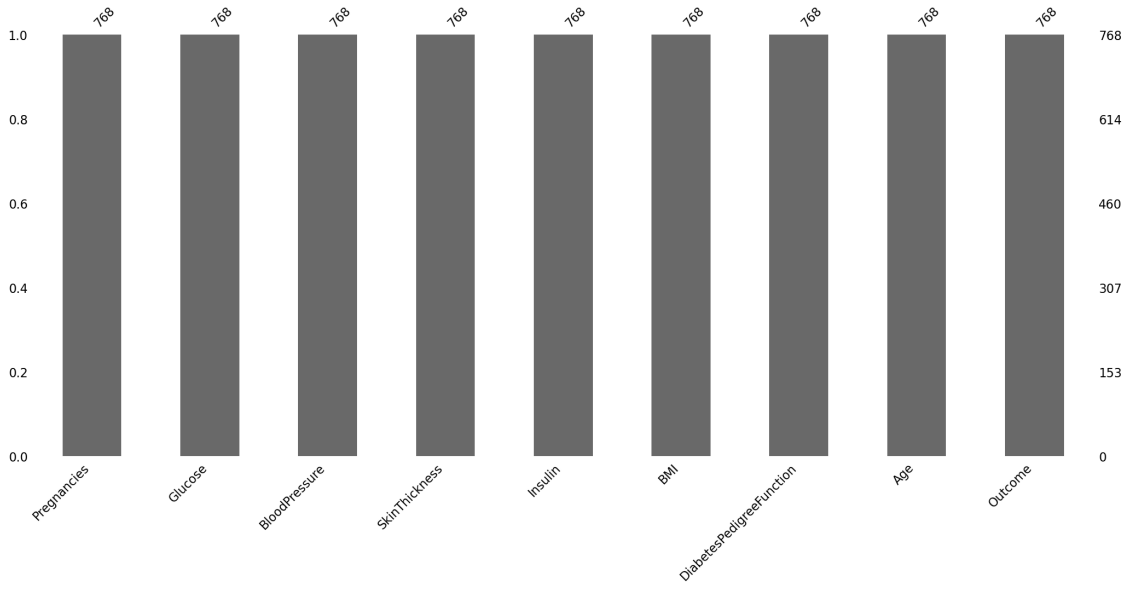
[10]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
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			

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

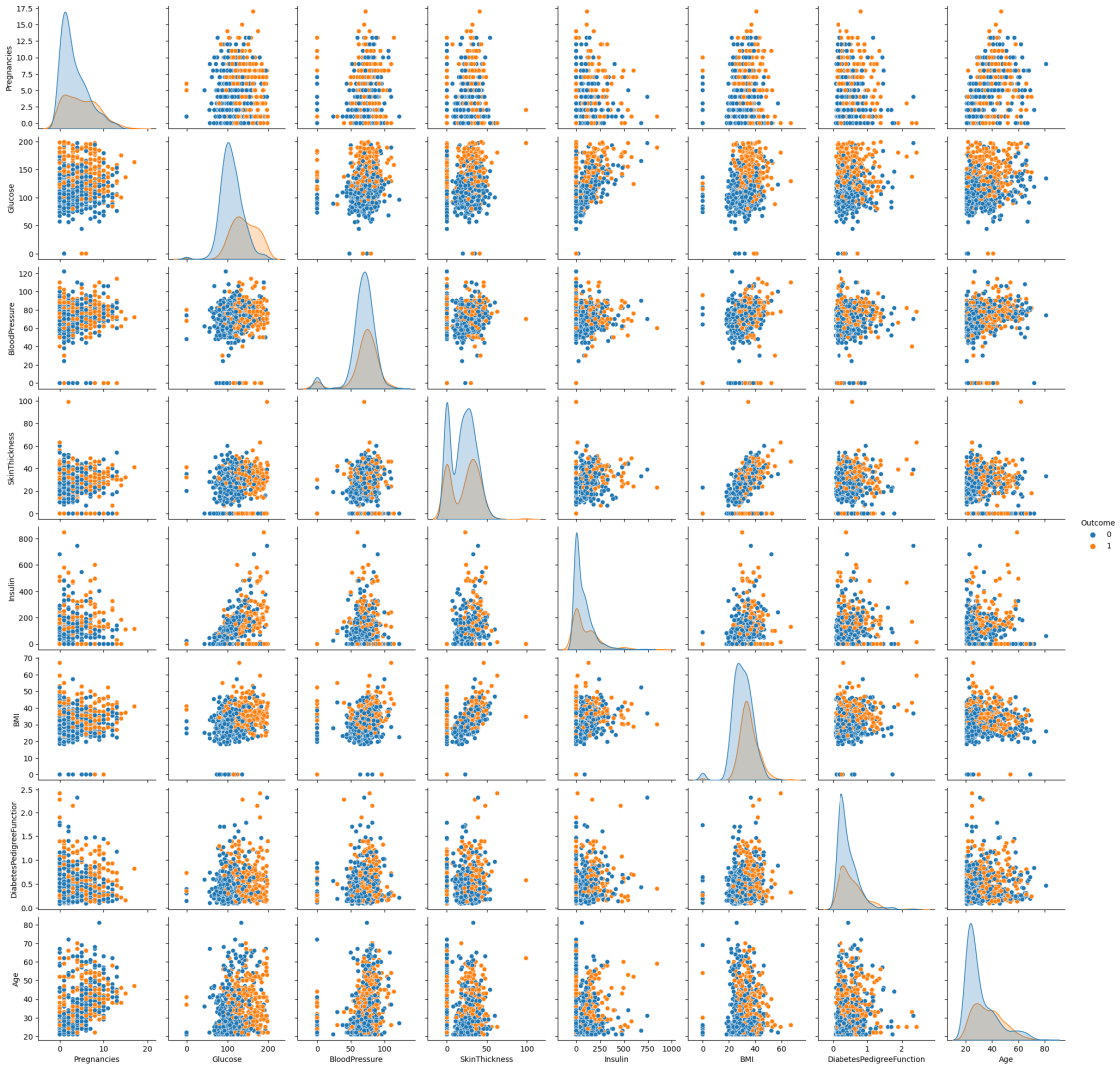
[11]:
```
import missingno as msno
msno.bar(data)
```

[11]: <AxesSubplot:>



[12]:
```
sns.pairplot(data,hue='Outcome')
```

[12]: <seaborn.axisgrid.PairGrid at 0x1bd270aa0a0>



[13]:
```
from sklearn.preprocessing import StandardScaler
```

[14]:
```
scaler = StandardScaler()
```

[15]:
```
scaler.fit(data.drop('Outcome',axis=1))
```

[15]:
```
StandardScaler()
```

[16]:
```
scaled_features = scaler.transform(data.drop('Outcome',axis=1))
```

[17]:
```
data_features = pd.DataFrame(scaled_features,columns=data.columns[:-1])
data_features.head()
```

[17]:
```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.639947  0.848324       0.149641       0.907270 -0.692891  0.204013
1    -0.844885 -1.123396      -0.160546       0.530902 -0.692891 -0.684422
2     1.233880  1.943724      -0.263941      -1.288212 -0.692891 -1.103255
3    -0.844885 -0.998208      -0.160546       0.154533  0.123302 -0.494043
4    -1.141852  0.504055      -1.504687       0.907270  0.765836  1.409746

   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
```

[18]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =␣
 ↪train_test_split(scaled_features,data['Outcome'],test_size=0.33,random_state␣
 ↪= 42)
```

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

[ ]:
```
KNN = KNeighborsClassifier(n_neighbors=1)
```

[ ]:
```
KNN.fit(X_train,y_train)
```

[ ]:
```
pred = KNN.predict(X_test)
```

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

[ ]:
```
print(confusion_matrix(y_test,pred))
```

[ ]:
```
print(classification_report(y_test,pred))
```

[ ]:
```
error = []

for i in range(1,50):

    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(X_train,y_train)
    pred_i = KNN.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

[ ]:
```
knn = KNeighborsClassifier(n_neighbors=15)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)
```

```
print('WITH K=16')
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

[ ]:
```
plt.figure(figsize=(10,8))
plt.plot(range(1,50),error,color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=5)
plt.title('Error and K Value')
plt.xlabel('K')
plt.ylabel('Error')
```

[ ]:
```
neighbors = np.arange(1, 100)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
k_list = list(range(3,20))
cv_scores = []

for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=20, scoring='accuracy')
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

plt.figure(figsize=(10,3))
plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE);
```

```
plt.show();
```

[ ]:
```
knn = KNeighborsClassifier()
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 51))
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy',␣
 ↪return_train_score=True,verbose=1)
grid_search=grid.fit(X_train, y_train)
print(grid_search.best_params_)
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,confusion_matrix,␣
 ↪ConfusionMatrixDisplay
from sklearn.model_selection import␣
 ↪train_test_split,cross_val_score,GridSearchCV
import sklearn.metrics as mt
for seed in range(100):
    X_train, X_test, y_train, y_test =␣
 ↪train_test_split(scaled_features,data['Outcome'],test_size=0.33,random_state␣
 ↪= seed)
    KNN = KNeighborsClassifier(n_neighbors=15)
    KNN.fit(X_train,y_train)
    pred = KNN.predict(X_test)
    print("random state : ",seed)
    print(confusion_matrix(y_test,pred))
    print(classification_report(y_test,pred))
```

[ ]:
```

```