```python
import pandas as pd
import os
from google.colab import drive, files
drive.mount('/content/drive')
uploaded = files.upload()

files = os.listdir()
print(files)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
<IPython.core.display.HTML object>
```

```
Saving diabetes.csv to diabetes (1).csv
['.config', 'diabetes.csv', 'diabetes (1).csv', 'drive',
'sample_data']
```

```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=0.25 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
>seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn)
(2023.3.post1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)

dia = 'diabetes.csv'

diabetes = pd.read_csv(dia, sep = ",")
print(diabetes)

     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]


diabetes.head()

   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

diabetes.columns.values

array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
       'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',
'Outcome'],
      dtype=object)

diabetes.describe()

       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
```

```
diabetes.nunique()
```

```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

Monotonih atributa ovdje nemamo, ali najbliže monotonom atributu je DiabetesPedigreeFunction koji ima 517 različitih vrijednosti.

```
diabetes.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
```

```
diabetes.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64

diabetes.describe()

       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
```

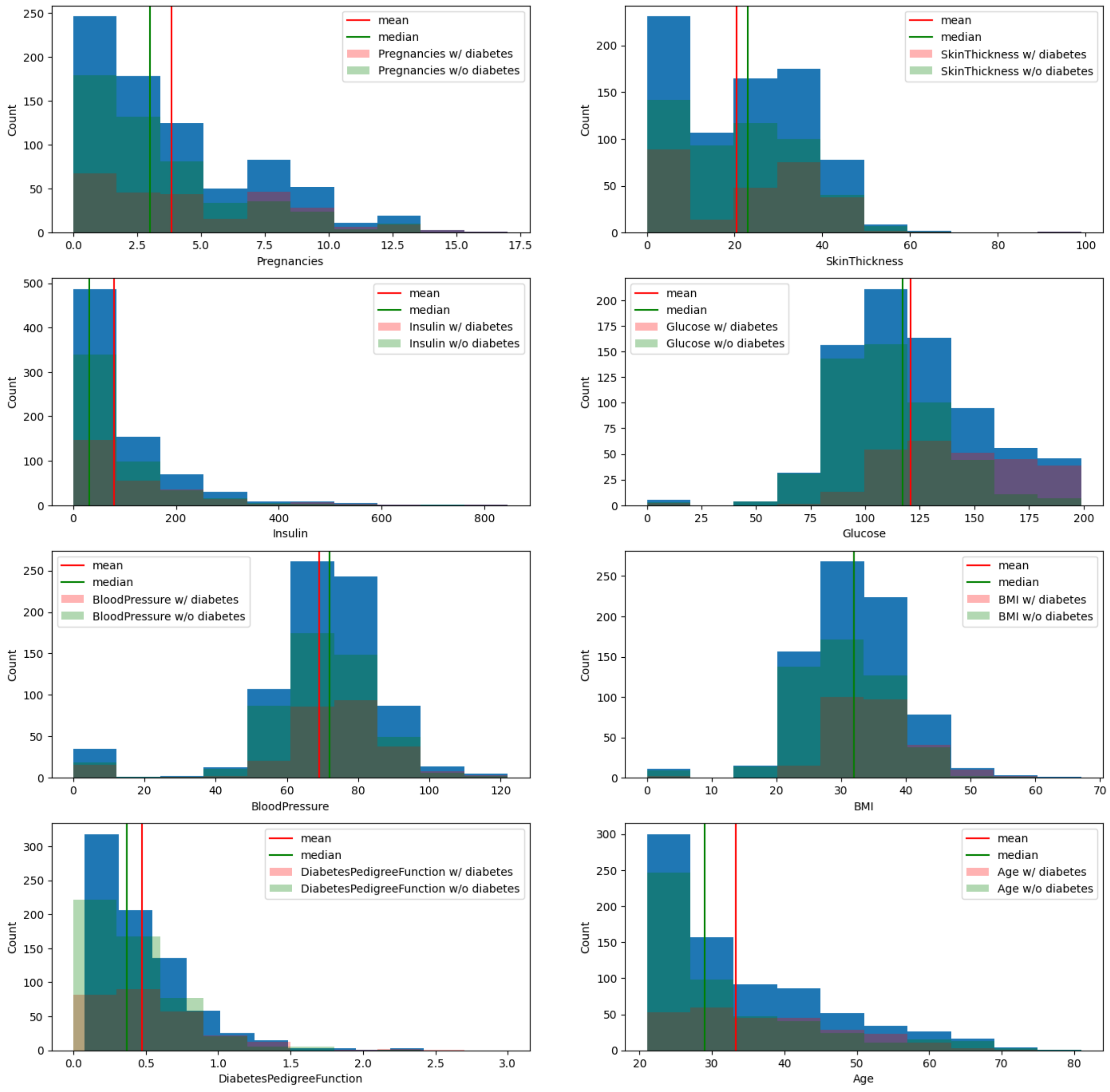
Stršeće značajke bi bile značajke Pregnancies, SkinThickness i Insulin. Također je neobično to što značajke Glucose, BloodPressure, SkinThickness, Insulin i BMI imaju 0 kao minimalnu vrijednost.

```
Atributi = ["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]
diabetes2 = diabetes[(diabetes[Atributi] != 0).all(axis=1)]

diabetes2.describe()
```

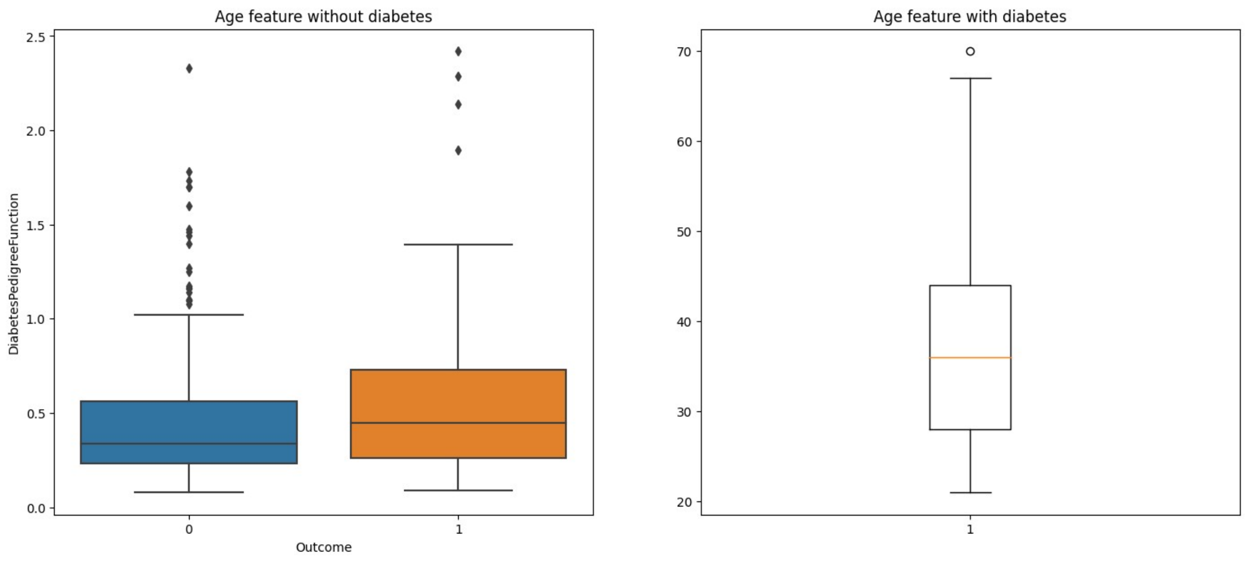
```
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   392.000000  392.000000     392.000000     392.000000
392.000000
mean      3.301020  122.627551      70.663265      29.145408
156.056122
std       3.211424   30.860781      12.496092      10.516424
118.841690
min       0.000000   56.000000      24.000000       7.000000
14.000000
25%       1.000000   99.000000      62.000000      21.000000
76.750000
50%       2.000000  119.000000      70.000000      29.000000
125.500000
75%       5.000000  143.000000      78.000000      37.000000
190.000000
max      17.000000  198.000000     110.000000      63.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  392.000000                392.000000  392.000000  392.000000
mean    33.086224                  0.523046   30.864796    0.331633
std      7.027659                  0.345488   10.200777    0.471401
min     18.200000                  0.085000   21.000000    0.000000
25%     28.400000                  0.269750   23.000000    0.000000
50%     33.200000                  0.449500   27.000000    0.000000
75%     37.100000                  0.687000   36.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

```python
import matplotlib.pyplot as plt
fi, ax = plt.subplots(4, 2)
fi.set_figheight(17)
fi.set_figwidth(17)
fi.add_subplot(4, 2, 1)
plt.hist(diabetes.Pregnancies, bins = 10)
plt.xlabel('Pregnancies')
plt.ylabel('Count')
plt.axvline(diabetes.Pregnancies.mean(), color = 'red', label =
'mean')
plt.axvline(diabetes.Pregnancies.median(), color = 'g', label =
'median')
plt.hist(diabetes[diabetes.Outcome == 1].Pregnancies, bins = 10, range
= (0, 17), label='Pregnancies w/ diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].Pregnancies, bins = 10, range
= (0, 17), label='Pregnancies w/o diabetes', color='green', alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 2)
plt.hist(diabetes.SkinThickness, bins = 10)
plt.xlabel('SkinThickness')
plt.ylabel('Count')
plt.axvline(diabetes.SkinThickness.mean(), color = 'red', label =
'mean')
plt.axvline(diabetes.SkinThickness.median(), color = 'g', label =
'median')
plt.hist(diabetes[diabetes.Outcome == 1].SkinThickness, bins = 10,
range = (0, 99), label='SkinThickness w/ diabetes', color='red',
alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].SkinThickness, bins = 10,
range = (0, 99), label='SkinThickness w/o diabetes', color='green',
alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 3)
plt.hist(diabetes.Insulin, bins = 10)
plt.xlabel('Insulin')
plt.ylabel('Count')
plt.axvline(diabetes.Insulin.mean(), color = 'red', label = 'mean')
plt.axvline(diabetes.Insulin.median(), color = 'g', label = 'median')
plt.hist(diabetes[diabetes.Outcome == 1].Insulin, bins = 10, range =
(0, 846), label='Insulin w/ diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].Insulin, bins = 10, range =
(0, 846), label='Insulin w/o diabetes', color='green', alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 4)
plt.hist(diabetes.Glucose, bins = 10)
plt.xlabel('Glucose')
plt.ylabel('Count')
plt.axvline(diabetes.Glucose.mean(), color = 'red', label = 'mean')
plt.axvline(diabetes.Glucose.median(), color = 'g', label = 'median')
plt.hist(diabetes[diabetes.Outcome == 1].Glucose, bins = 10, range =
(0, 199), label='Glucose w/ diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].Glucose, bins = 10, range =
(0, 199), label='Glucose w/o diabetes', color='green', alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 5)
plt.hist(diabetes.BloodPressure, bins = 10)
plt.xlabel('BloodPressure')
plt.ylabel('Count')
plt.axvline(diabetes.BloodPressure.mean(), color = 'red', label =
'mean')
plt.axvline(diabetes.BloodPressure.median(), color = 'g', label =
'median')
plt.hist(diabetes[diabetes.Outcome == 1].BloodPressure, bins = 10,
range = (0, 122), label='BloodPressure w/ diabetes', color='red',
alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].BloodPressure, bins = 10,
range = (0, 122), label='BloodPressure w/o diabetes', color='green',
alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 6)
```

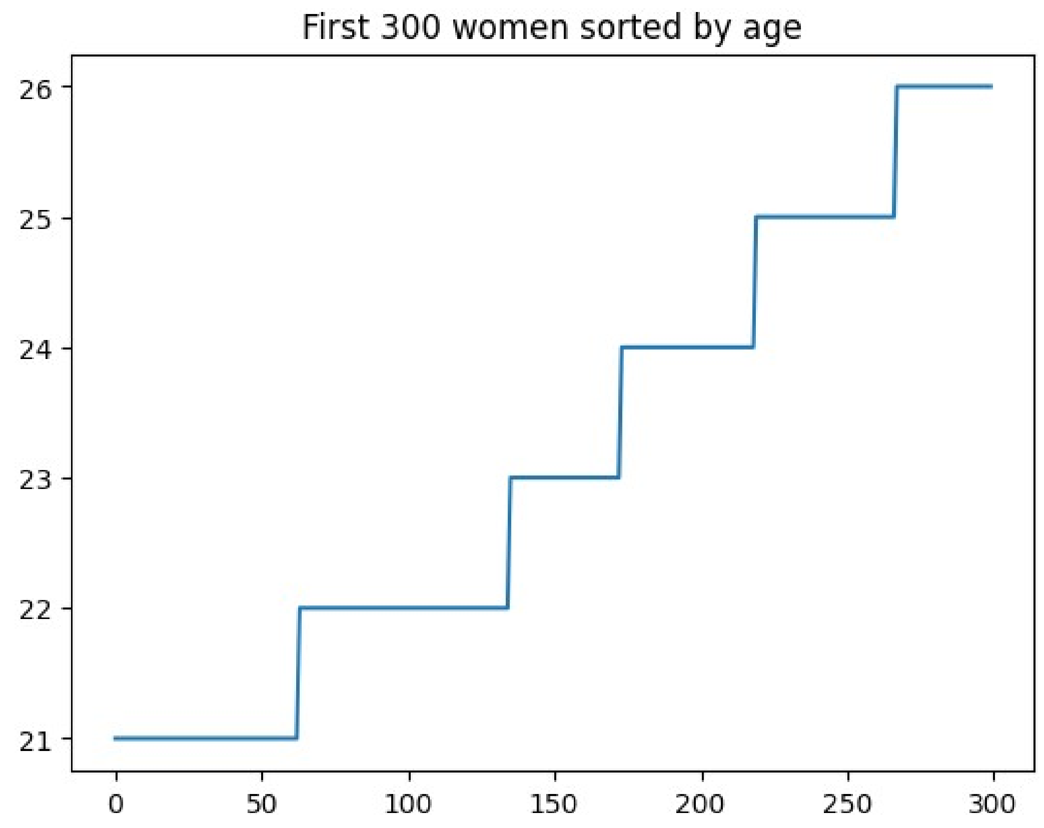
```python
plt.hist(diabetes.BMI, bins = 10)
plt.xlabel('BMI')
plt.ylabel('Count')
plt.axvline(diabetes.BMI.mean(), color = 'red', label = 'mean')
plt.axvline(diabetes.BMI.median(), color = 'g', label = 'median')
plt.hist(diabetes[diabetes.Outcome == 1].BMI, bins = 10, range = (0, 
67), label='BMI w/ diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].BMI, bins = 10, range = (0, 
67), label='BMI w/o diabetes', color='green', alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 7)
plt.hist(diabetes.DiabetesPedigreeFunction  , bins = 10)
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('Count')
plt.axvline(diabetes.DiabetesPedigreeFunction.mean(), color = 'red', 
label = 'mean')
plt.axvline(diabetes.DiabetesPedigreeFunction.median(), color = 'g', 
label = 'median')
plt.hist(diabetes[diabetes.Outcome == 1].DiabetesPedigreeFunction, 
bins = 10, range = (0, 3), label='DiabetesPedigreeFunction w/ 
diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].DiabetesPedigreeFunction, 
bins = 10, range = (0, 3), label='DiabetesPedigreeFunction w/o 
diabetes', color='green', alpha=0.3)
plt.legend()
fi.add_subplot(4, 2, 8)
plt.hist(diabetes.Age, bins = 10)
plt.xlabel('Age')
plt.ylabel('Count')
plt.axvline(diabetes.Age.mean(), color = 'red', label = 'mean')
plt.axvline(diabetes.Age.median(), color = 'g', label = 'median')
plt.hist(diabetes[diabetes.Outcome == 1].Age, bins = 10, range = (21, 
81), label='Age w/ diabetes', color='red', alpha=0.3)
plt.hist(diabetes[diabetes.Outcome == 0].Age, bins = 10, range = (21, 
81), label='Age w/o diabetes', color='green', alpha=0.3)
plt.legend()
for a in ax.flatten():
    a.set_xticks([])
    a.set_yticks([])
    a.set_xticklabels([])
    a.set_yticklabels([])
    a.spines['top'].set_visible(False)
    a.spines['right'].set_visible(False)
    a.spines['bottom'].set_visible(False)
    a.spines['left'].set_visible(False)
```



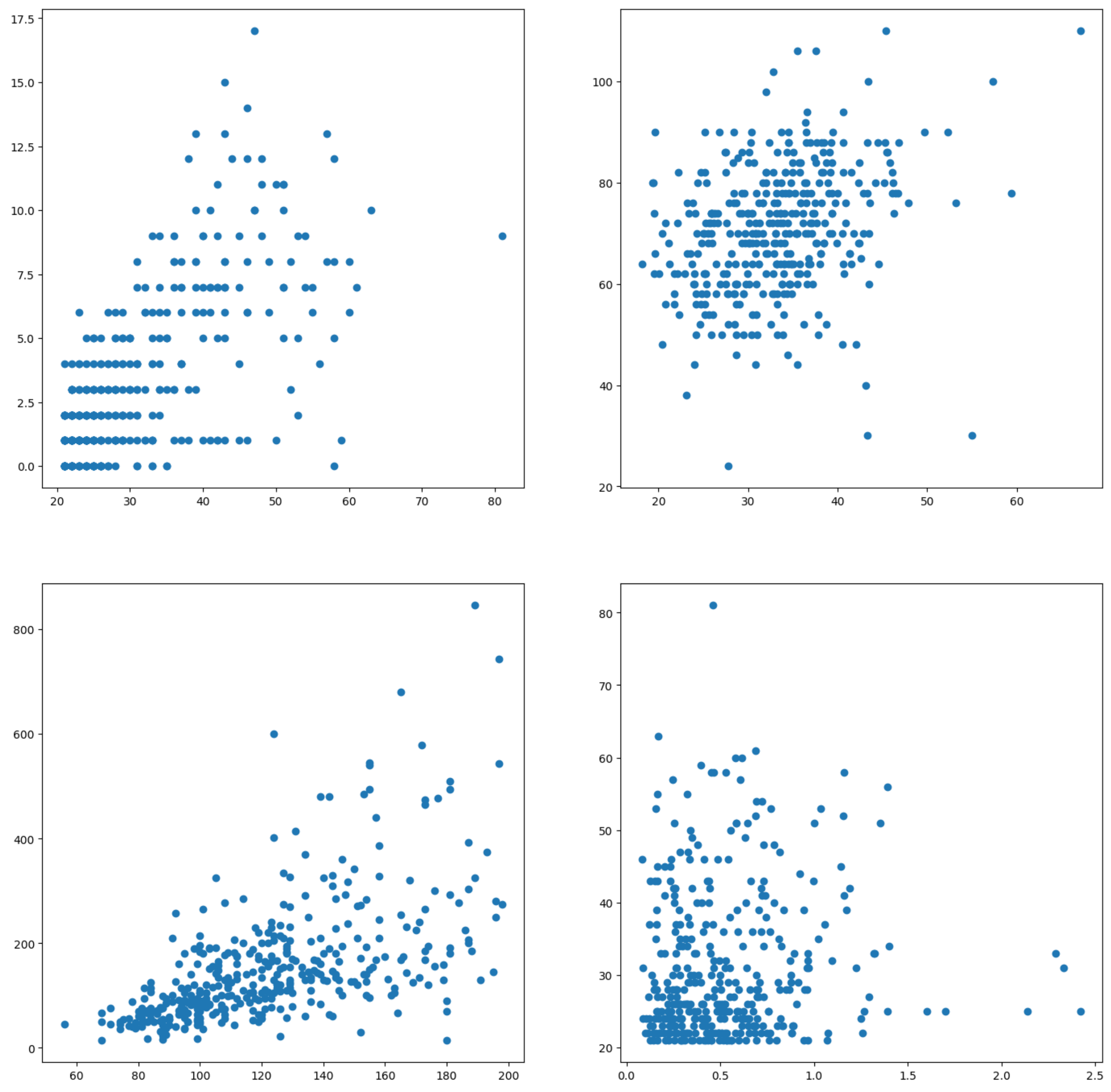
```
fi, ax = plt.subplots(1, 2)
fi.set_figheight(7)
fi.set_figwidth(17)
fi.add_subplot(1, 2, 1)
sns.boxplot(x = diabetes["Outcome"], y =
diabetes["DiabetesPedigreeFunction"])
plt.title("Age feature without diabetes")
fi.add_subplot(1, 2, 2)
plt.boxplot(diabetes[diabetes.Outcome == 1].Age)
plt.title("Age feature with diabetes")
for a in ax.flatten():
    a.set_xticks([])
    a.set_yticks([])
    a.set_xticklabels([])
    a.set_yticklabels([])
    a.spines['top'].set_visible(False)
    a.spines['right'].set_visible(False)
    a.spines['bottom'].set_visible(False)
    a.spines['left'].set_visible(False)
plt.show()
```



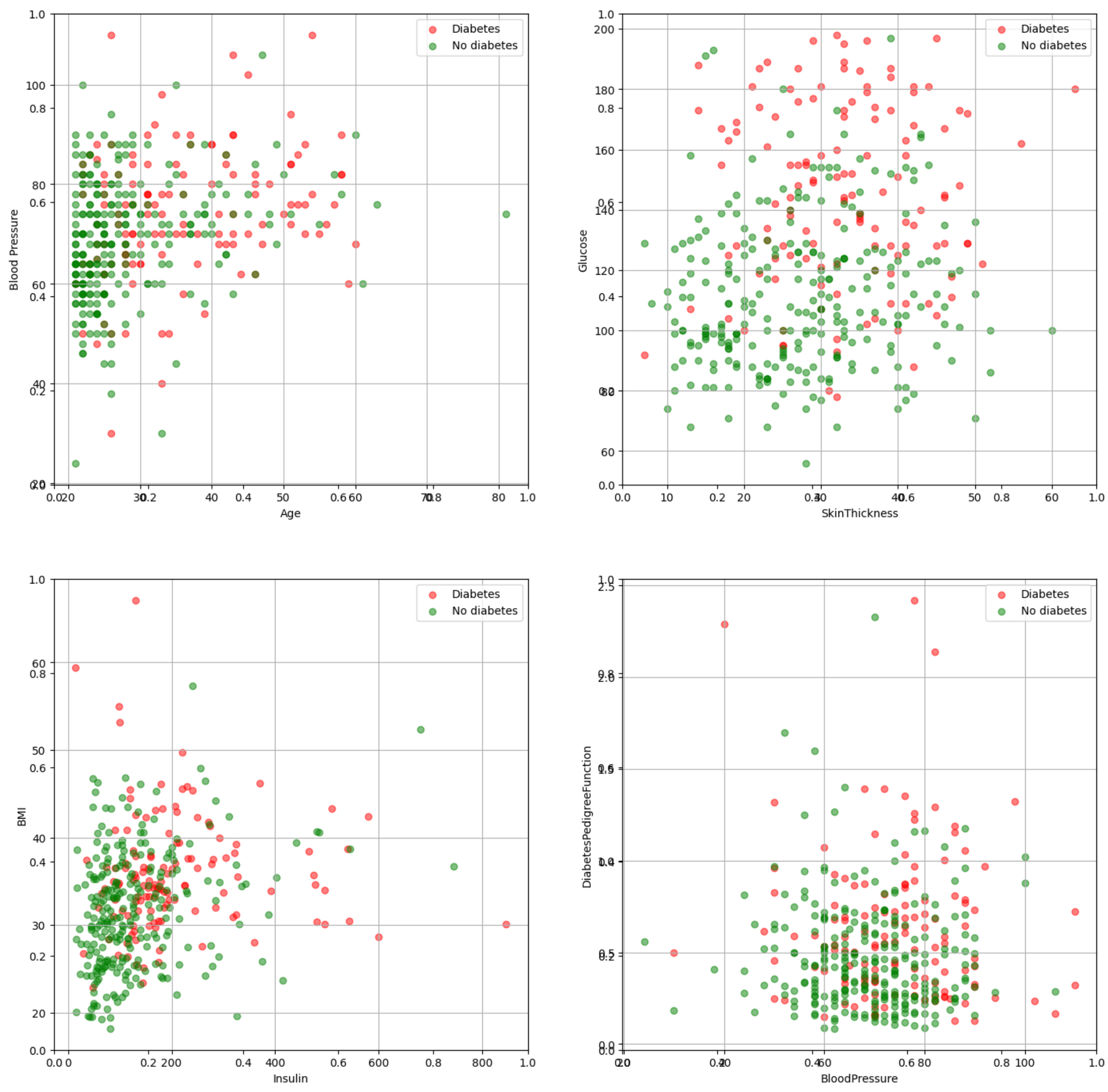
```
tmp = diabetes.copy()
sorted_age = tmp.sort_values(by=['Age']).Age.values[:300]
person = [x for x in range(300)]
plt.plot(person, sorted_age)
plt.title("First 300 women sorted by age")
plt.show()
```



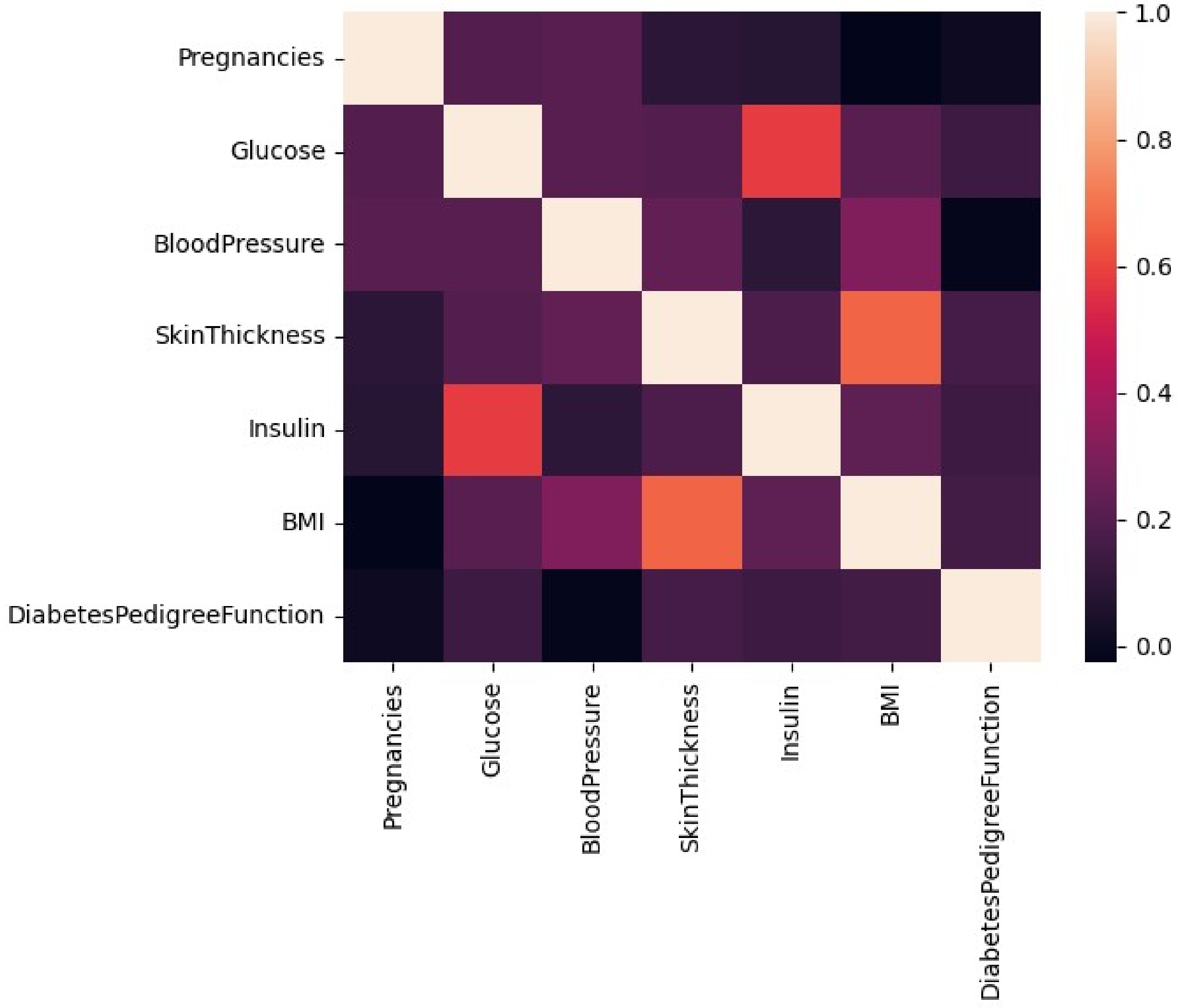
```
fi, ax = plt.subplots(2, 2)
fi.set_figheight(17)
fi.set_figwidth(17)
fi.add_subplot(2, 2, 1)
plt.scatter(diabetes2.Age, diabetes2.Pregnancies)
fi.add_subplot(2, 2, 2)
plt.scatter(diabetes2.BMI, diabetes2.BloodPressure)
fi.add_subplot(2, 2, 3)
plt.scatter(diabetes2.Glucose, diabetes2.Insulin)
fi.add_subplot(2, 2, 4)
plt.scatter(diabetes2.DiabetesPedigreeFunction, diabetes2.Age)
for a in ax.flatten():
    a.set_xticks([])
    a.set_yticks([])
    a.set_xticklabels([])
    a.set_yticklabels([])
    a.spines['top'].set_visible(False)
    a.spines['right'].set_visible(False)
    a.spines['bottom'].set_visible(False)
    a.spines['left'].set_visible(False)
plt.show()
```



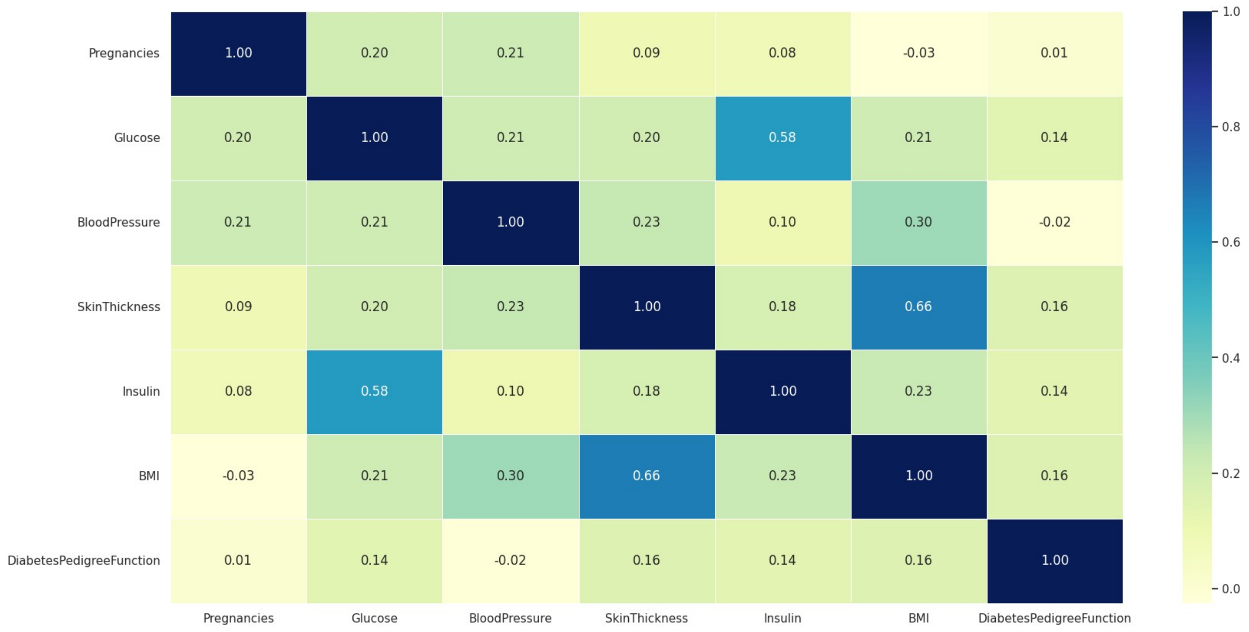
```
fi, ax = plt.subplots(2, 2)
fi.set_figheight(17)
fi.set_figwidth(17)
mask_survived = diabetes.Outcome == 1
fi.add_subplot(2, 2, 1)
plt.scatter(diabetes2.loc[mask_survived,'Age'],
diabetes2.loc[mask_survived,'BloodPressure'], alpha=0.5, c='r',
label='Diabetes')
plt.scatter(diabetes2.loc[~mask_survived,'Age'],
diabetes2.loc[~mask_survived,'BloodPressure'], alpha=0.5, c='g',
label='No diabetes')
plt.xlabel('Age')
plt.ylabel('Blood Pressure')
plt.grid()
plt.legend(loc='best')
fi.add_subplot(2, 2, 2)
plt.scatter(diabetes2.loc[mask_survived,'SkinThickness'],
diabetes2.loc[mask_survived,'Glucose'], alpha=0.5, c='r',
label='Diabetes')
plt.scatter(diabetes2.loc[~mask_survived,'SkinThickness'],
diabetes2.loc[~mask_survived,'Glucose'], alpha=0.5, c='g', label='No
diabetes')
plt.xlabel('SkinThickness')
plt.ylabel('Glucose')
plt.grid()
plt.legend(loc='best')
fi.add_subplot(2, 2, 3)
plt.scatter(diabetes2.loc[mask_survived,'Insulin'],
diabetes2.loc[mask_survived,'BMI'], alpha=0.5, c='r',
label='Diabetes')
plt.scatter(diabetes2.loc[~mask_survived,'Insulin'],
diabetes2.loc[~mask_survived,'BMI'], alpha=0.5, c='g', label='No
diabetes')
plt.xlabel('Insulin')
plt.ylabel('BMI')
plt.grid()
plt.legend(loc='best')
fi.add_subplot(2, 2, 4)
plt.scatter(diabetes2.loc[mask_survived,'BloodPressure'],
diabetes2.loc[mask_survived,'DiabetesPedigreeFunction'], alpha=0.5,
c='r', label='Diabetes')
plt.scatter(diabetes2.loc[~mask_survived,'BloodPressure'],
diabetes2.loc[~mask_survived,'DiabetesPedigreeFunction'], alpha=0.5,
c='g', label='No diabetes')
plt.xlabel('BloodPressure')
plt.ylabel('DiabetesPedigreeFunction')
plt.grid()
plt.legend(loc='best')
plt.show()
```



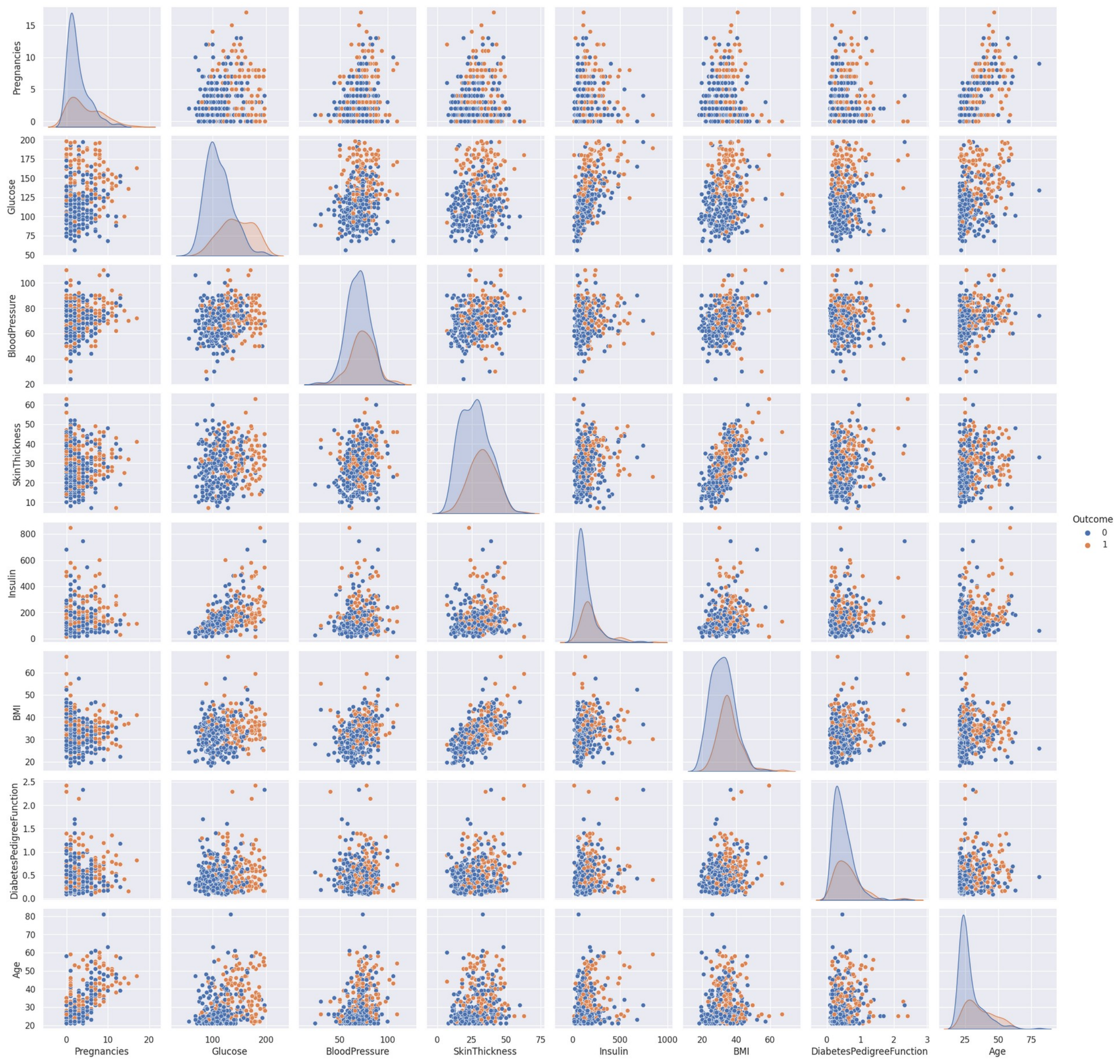
```python
import seaborn as sns
diabetes_cor = diabetes2.loc[:,diabetes2.columns.isin(['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction'])]
sns.heatmap(diabetes_cor.corr())
plt.show()
```



```
sns.set(rc={'figure.figsize':(20,10)})
sns.heatmap(diabetes_cor.corr(), annot=True, fmt=".2f", cmap="YlGnBu",
linewidths=0.5)
plt.show()
```



```
sns.set(rc={'figure.figsize':(25,15)})
sns.pairplot(diabetes2.loc[:,[ 'Pregnancies', 'Glucose',
'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age', 'Outcome']], hue="Outcome")
plt.show()
```

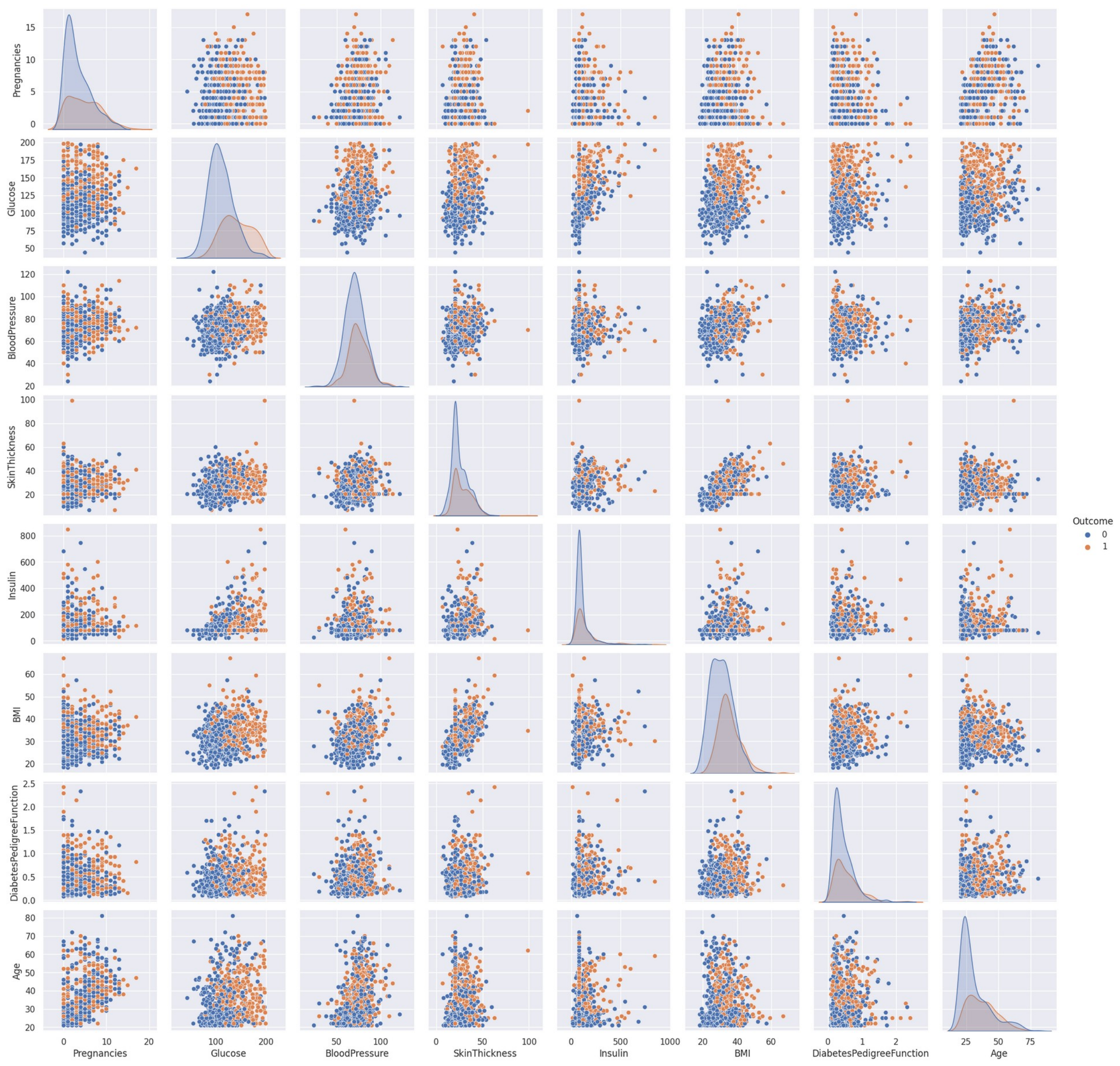


```
diabetes.isnull().values.any()

False

Atributi = ["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]
sred = diabetes[Atributi].mean()
diabetes[Atributi] = diabetes[Atributi].mask(diabetes[Atributi] == 0,
pd.NA)
diabetes[Atributi] = diabetes[Atributi].apply(lambda col:
col.fillna(sred[col.name]))
diabetes.describe()

       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  121.681605      72.254807      26.606479
118.660163
std       3.369578   30.436016      12.115932       9.631241
93.080358
min       0.000000   44.000000      24.000000       7.000000
14.000000
25%       1.000000   99.750000      64.000000      20.536458
79.799479
50%       3.000000  117.000000      72.000000      23.000000
79.799479
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    32.450805                  0.471876   33.240885    0.348958
std      6.875374                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  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

Atributi = ["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]
sred = diabetes[Atributi].median()
diabetes[Atributi] = diabetes[Atributi].mask(diabetes[Atributi] == 0,
pd.NA)
diabetes[Atributi] = diabetes[Atributi].apply(lambda col:
col.fillna(sred[col.name]))
diabetes.describe()

       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  121.681605      72.254807      26.606479
118.660163
std       3.369578   30.436016      12.115932       9.631241
93.080358
min       0.000000   44.000000      24.000000       7.000000
14.000000
25%       1.000000   99.750000      64.000000      20.536458
79.799479
50%       3.000000  117.000000      72.000000      23.000000
79.799479
```

```
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    32.450805                  0.471876   33.240885    0.348958
std      6.875374                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  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
```

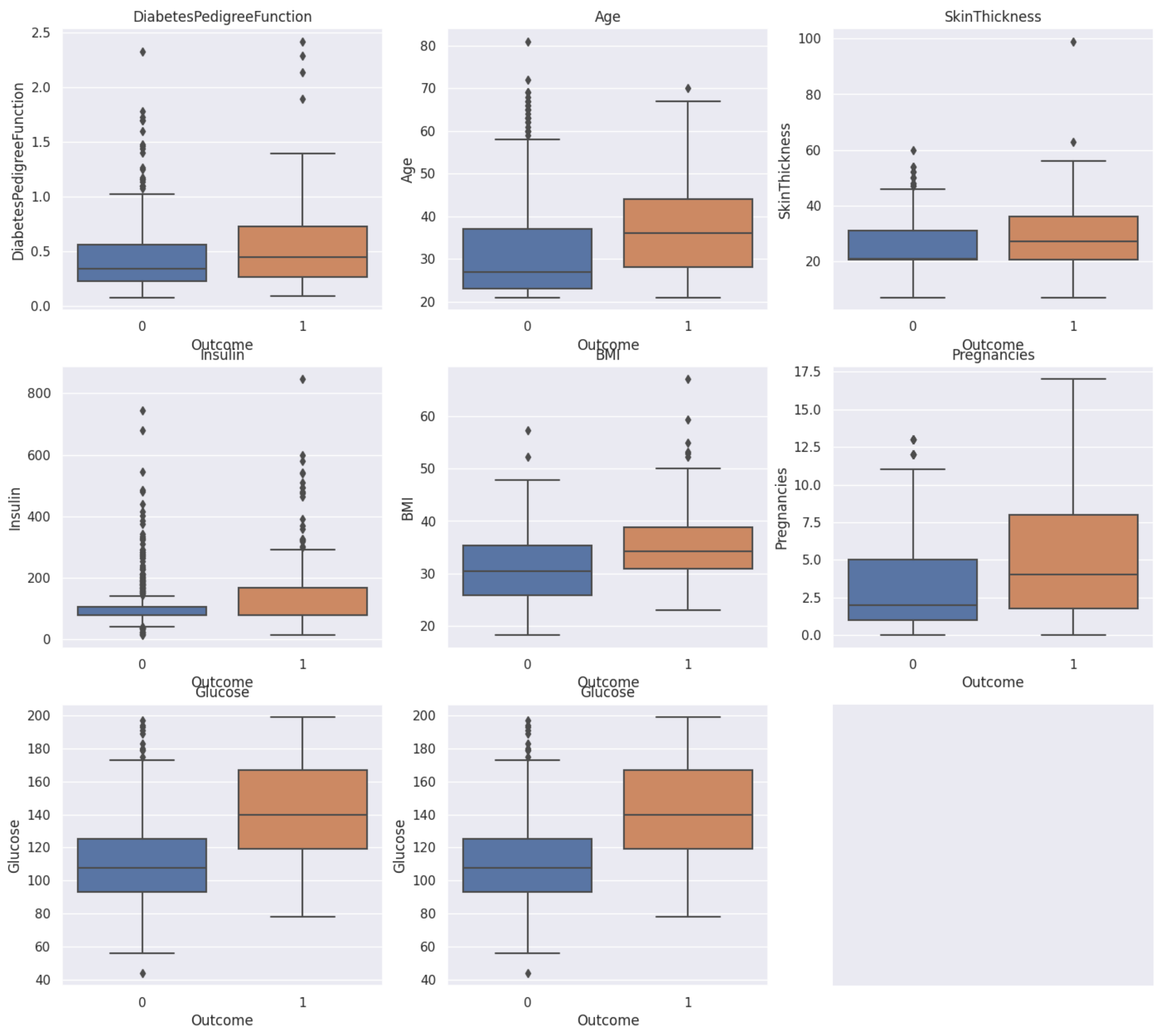
```python
import seaborn as sns
sns.pairplot(diabetes, hue="Outcome")
```

```
<seaborn.axisgrid.PairGrid at 0x7c95a0a69150>
```



```
fi, ax = plt.subplots(3, 3)
fi.set_figheight(15)
fi.set_figwidth(17)
fi.add_subplot(3, 3, 1)
sns.boxplot(x = diabetes["Outcome"], y =
diabetes["DiabetesPedigreeFunction"])
plt.title("DiabetesPedigreeFunction")
fi.add_subplot(3, 3, 2)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["Age"])
plt.title("Age")
fi.add_subplot(3, 3, 3)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["SkinThickness"])
plt.title("SkinThickness")
fi.add_subplot(3, 3, 4)
```

```
sns.boxplot(x = diabetes["Outcome"], y = diabetes["Insulin"])
plt.title("Insulin")
fi.add_subplot(3, 3, 5)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["BMI"])
plt.title("BMI")
fi.add_subplot(3, 3, 6)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["Pregnancies"])
plt.title("Pregnancies")
fi.add_subplot(3, 3, 7)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["Glucose"])
plt.title("Glucose")
fi.add_subplot(3, 3, 8)
sns.boxplot(x = diabetes["Outcome"], y = diabetes["Glucose"])
plt.title("Glucose")
for a in ax.flatten():
    a.set_xticks([])
    a.set_yticks([])
    a.set_xticklabels([])
    a.set_yticklabels([])
    a.spines['top'].set_visible(False)
    a.spines['right'].set_visible(False)
    a.spines['bottom'].set_visible(False)
    a.spines['left'].set_visible(False)
plt.show()
```



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, f1_score
X = diabetes.iloc[:,:-1]
y = diabetes.iloc[:,-1:]
X_train, X_test, y_train, y_test1 = train_test_split(X, y,
test_size=0.3)
clf = DecisionTreeClassifier()
model = clf.fit(X_train, y_train)
y_pred1 = model.predict(X_test)
print("Tocnost (decision tree): ", metrics.accuracy_score(y_test1,
y_pred1)) #Njihova tocnost je oko 76%
print("Precision (decision tree): ", metrics.precision_score(y_test1,
y_pred1)) #Njima 70%
```

```
print(metrics.confusion_matrix(y_test1, y_pred1))
tn, fp, fn, tp = metrics.confusion_matrix(y_test1, y_pred1).ravel()
Spec1 = tn / (tn+fp)
Acc1 = metrics.accuracy_score(y_test1, y_pred1)
Prec1 = metrics.precision_score(y_test1, y_pred1)
Rec1 = metrics.recall_score(y_test1, y_pred1)
F1_1 = metrics.f1_score(y_test1, y_pred1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print()
clf2 = DecisionTreeClassifier()
model2 = clf2.fit(X_train, y_train)
y_pred4 = model2.predict(X_test)
print("Tocnost (decision tree) (skalirano): ",
metrics.accuracy_score(y_test1, y_pred4))
print(metrics.confusion_matrix(y_test1, y_pred4))
tn, fp, fn, tp = metrics.confusion_matrix(y_test1, y_pred4).ravel()
Spec4 = tn / (tn+fp)
Acc4 = metrics.accuracy_score(y_test1, y_pred4)
Prec4 = metrics.precision_score(y_test1, y_pred4)
Rec4 = metrics.recall_score(y_test1, y_pred4)
F1_4 = metrics.f1_score(y_test1, y_pred4)
```

```
Tocnost (decision tree):  0.6926406926406926
Precision (decision tree):  0.5925925925925926
[[112  33]
 [ 38  48]]

Tocnost (decision tree) (skalirano):  0.658008658008658
[[106  39]
 [ 40  46]]
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
X = diabetes.iloc[:,:-1]
y = diabetes.iloc[:,-1:]
X_train, X_test, y_train, y_test2 = train_test_split(X, y,
test_size=0.3)
model = clf.fit(X_train, y_train)
y_pred2 = model.predict(X_test)
print("Tocnost (random forest): ", metrics.accuracy_score(y_test2,
y_pred2)) #Njihova tocnost je oko 81%
print("Precision (random forest): ", metrics.precision_score(y_test2,
y_pred2)) #Njima 89%
print(metrics.confusion_matrix(y_test2, y_pred2))
tn, fp, fn, tp = metrics.confusion_matrix(y_test2, y_pred2).ravel()
Spec2 = tn / (tn+fp)
Acc2 = metrics.accuracy_score(y_test2, y_pred2)
Prec2 = metrics.precision_score(y_test2, y_pred2)
```

```
Rec2 = metrics.recall_score(y_test2, y_pred2)
F1_2 = metrics.f1_score(y_test2, y_pred2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print()
clf2 = RandomForestClassifier()
model2 = clf2.fit(X_train, y_train)
y_pred5 = model2.predict(X_test)
print("Tocnost (random forest) (skalirano): ",
metrics.accuracy_score(y_test2, y_pred5))
print(metrics.confusion_matrix(y_test2, y_pred5))
tn, fp, fn, tp = metrics.confusion_matrix(y_test2, y_pred5).ravel()
Spec5 = tn / (tn+fp)
Acc5 = metrics.accuracy_score(y_test2, y_pred5)
Prec5 = metrics.precision_score(y_test2, y_pred5)
Rec5 = metrics.recall_score(y_test2, y_pred5)
F1_5 = metrics.f1_score(y_test2, y_pred5)
```

```
<ipython-input-97-1e4957305201>:6: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
  model = clf.fit(X_train, y_train)
```

```
Tocnost (random forest):  0.70995670995671
Precision (random forest):  0.6521739130434783
[[119  24]
 [ 43  45]]
```

```
<ipython-input-97-1e4957305201>:22: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
  model2 = clf2.fit(X_train, y_train)
```

```
Tocnost (random forest) (skalirano):  0.70995670995671
[[118  25]
 [ 42  46]]
```

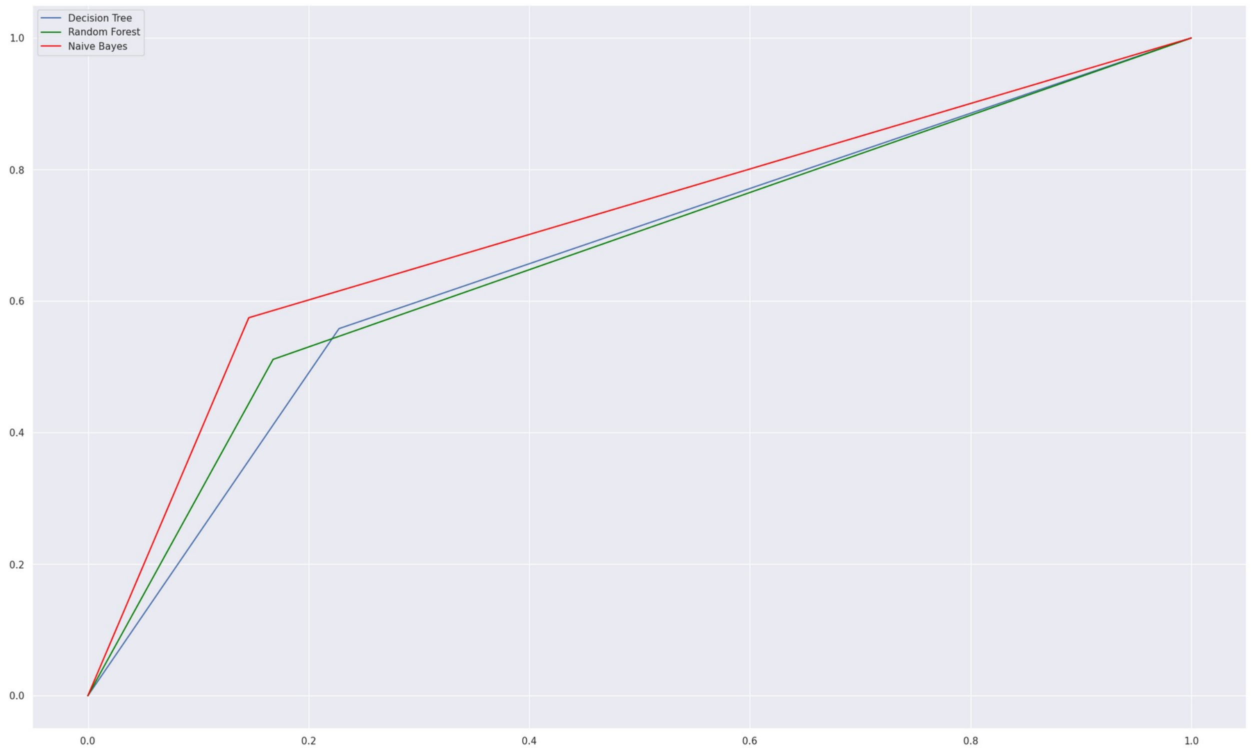
```
from sklearn.naive_bayes import GaussianNB
X = diabetes.iloc[:,:-1]
y = diabetes.iloc[:,-1:]
X_train, X_test, y_train, y_test3 = train_test_split(X, y,
test_size=0.3)
clf = GaussianNB()
model = clf.fit(X_train, y_train)
y_pred3 = model.predict(X_test)
print("Tocnost (naivni Bayes): ", metrics.accuracy_score(y_test3,
y_pred3)) #Njihova tocnost je oko 78%
print("Precision (naivni Bayes): ", metrics.precision_score(y_test3,
y_pred3)) #Njima 81%
print(metrics.confusion_matrix(y_test3, y_pred3))
tn, fp, fn, tp = metrics.confusion_matrix(y_test3, y_pred3).ravel()
Spec3 = tn / (tn+fp)
Acc3 = metrics.accuracy_score(y_test3, y_pred3)
Prec3 = metrics.precision_score(y_test3, y_pred3)
Rec3 = metrics.recall_score(y_test3, y_pred3)
F1_3 = metrics.f1_score(y_test3, y_pred3)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print()
clf2 = GaussianNB()
model2 = clf2.fit(X_train, y_train)
y_pred6 = model2.predict(X_test)
print("Tocnost (naivni Bayes) (skalirano): ",
metrics.accuracy_score(y_test3, y_pred6))
print(metrics.confusion_matrix(y_test3, y_pred6))
tn, fp, fn, tp = metrics.confusion_matrix(y_test3, y_pred6).ravel()
Spec6 = tn / (tn+fp)
Acc6 = metrics.accuracy_score(y_test3, y_pred6)
Prec6 = metrics.precision_score(y_test3, y_pred6)
Rec6 = metrics.recall_score(y_test3, y_pred6)
F1_6 = metrics.f1_score(y_test3, y_pred6)
```

```
Tocnost (naivni Bayes):  0.7489177489177489
Precision (naivni Bayes):  0.704225352112676
[[123  21]
 [ 37  50]]

Tocnost (naivni Bayes) (skalirano):  0.7489177489177489
[[123  21]
 [ 37  50]]

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:11
43: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

```
from sklearn.metrics import roc_auc_score, roc_curve

fpr1, tpr1, thresholds1 = roc_curve(y_test1, y_pred1)
plt.plot(fpr1, tpr1, label="Decision Tree")
fpr2, tpr2, thresholds2 = roc_curve(y_test2, y_pred2)
```

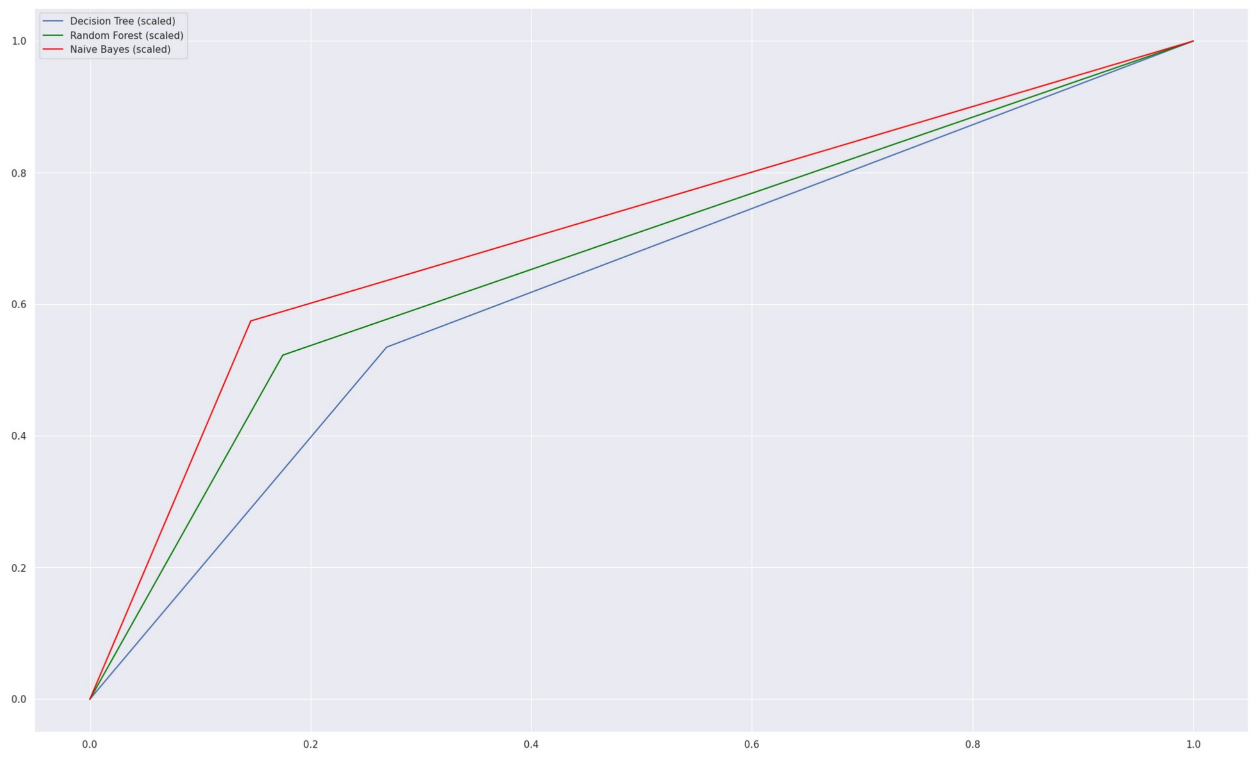
```
plt.plot(fpr2, tpr2, label="Random Forest", color = "green")
fpr3, tpr3, thresholds3 = roc_curve(y_test3, y_pred3)
plt.plot(fpr3, tpr3, label="Naive Bayes", color = "red")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c959e847f10>
```



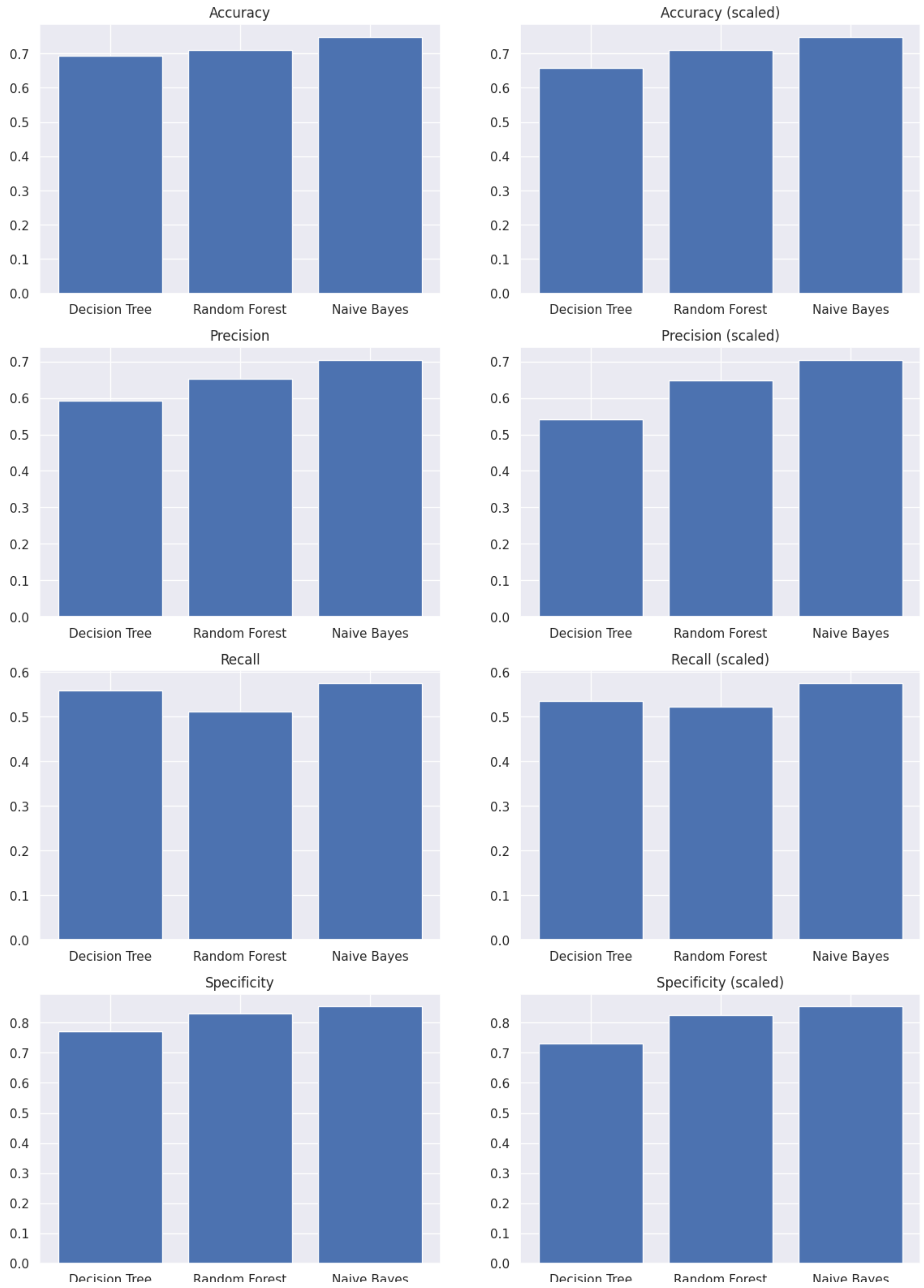
```
fpr4, tpr4, thresholds4 = roc_curve(y_test1, y_pred4)
plt.plot(fpr4, tpr4, label="Decision Tree (scaled)")
fpr5, tpr5, thresholds5 = roc_curve(y_test2, y_pred5)
plt.plot(fpr5, tpr5, label="Random Forest (scaled)", color = "green")
fpr6, tpr6, thresholds6 = roc_curve(y_test3, y_pred6)
plt.plot(fpr6, tpr6, label="Naive Bayes (scaled)", color = "red")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c959e8a5480>
```



```
fi, ax = plt.subplots(5, 2)
fi.set_figheight(25)
fi.set_figwidth(14)
fi.add_subplot(5, 2, 1)
plt.title("Accuracy")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Acc1,
Acc2, Acc3])
fi.add_subplot(5, 2, 2)
plt.title("Accuracy (scaled)")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Acc4,
Acc5, Acc6])
fi.add_subplot(5, 2, 3)
plt.title("Precision")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Prec1,
Prec2, Prec3])
fi.add_subplot(5, 2, 4)
plt.title("Precision (scaled)")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Prec4,
Prec5, Prec6])
fi.add_subplot(5, 2, 5)
plt.title("Recall")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Rec1,
Rec2, Rec3])
fi.add_subplot(5, 2, 6)
plt.title("Recall (scaled)")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Rec4,
Rec5, Rec6])
```

```
fi.add_subplot(5, 2, 7)
plt.title("Specificity")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Spec1,
Spec2, Spec3])
fi.add_subplot(5, 2, 8)
plt.title("Specificity (scaled)")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [Spec4,
Spec5, Spec6])
fi.add_subplot(5, 2, 9)
plt.title("F1 score")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [F1_1,
F1_2, F1_3])
fi.add_subplot(5, 2, 10)
plt.title("F1 score (scaled)")
plt.bar(["Decision Tree", "Random Forest", "Naive Bayes"], [F1_4,
F1_5, F1_6])

for a in ax.flatten():
    a.set_xticks([])
    a.set_yticks([])
    a.set_xticklabels([])
    a.set_yticklabels([])
    a.spines['top'].set_visible(False)
    a.spines['right'].set_visible(False)
    a.spines['bottom'].set_visible(False)
    a.spines['left'].set_visible(False)
plt.show()
```


Accuracy
Accuracy (scaled)
Precision
Precision (scaled)
Recall
Recall (scaled)
Specificity
Specificity (scaled)
Decision Tree
Random Forest
Naive Bayes
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8