Лабораторная 4. Метод k-ближайших соседей

Импорт библиотек

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import random
import math
```

1. Выбор датасетов: Студенты с **нечетным** порядковым номером в группе должны использовать датасет про **диабет**.

```
data = pd.read_csv('data/diabetes.csv')
data
```

```
     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


     Pedigree  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]
```

```
data.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    Pedigree         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
```

```
data.info()
```

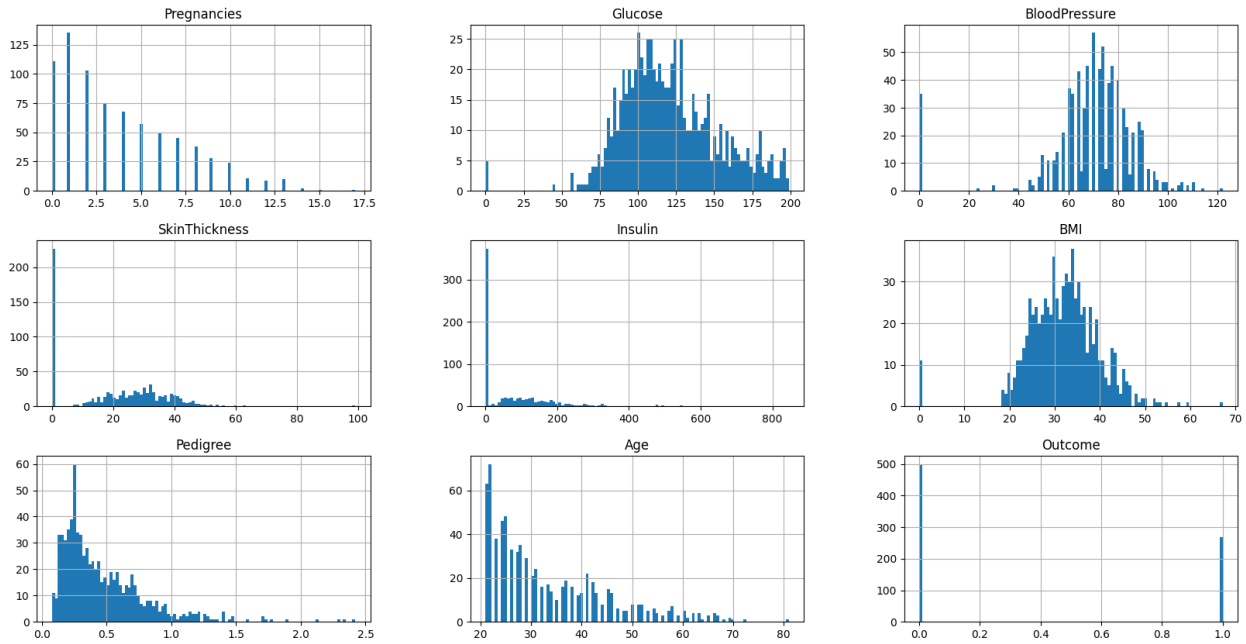
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Pregnancies    768 non-null    int64
```

```
 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   Pedigree        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
```

Визуализация

```
data.hist(bins=100, figsize=(20, 10))
```

```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
       [<Axes: title={'center': 'Pedigree'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>]], dtype=object)
```




```
data.isnull().sum()
```

```
Pregnancies      0
Glucose          0
```

```
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

Все значения не являются пустыми, нужно исключить нулевые значения

Outcome - категориальный признак.

--> Не все признаки обладают нормальным распределением, подлежат нормализации. Исключим записи с нулевыми значениями.

```
# data = data[~data[['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI']].isin([0]).any(axis=1)]
```

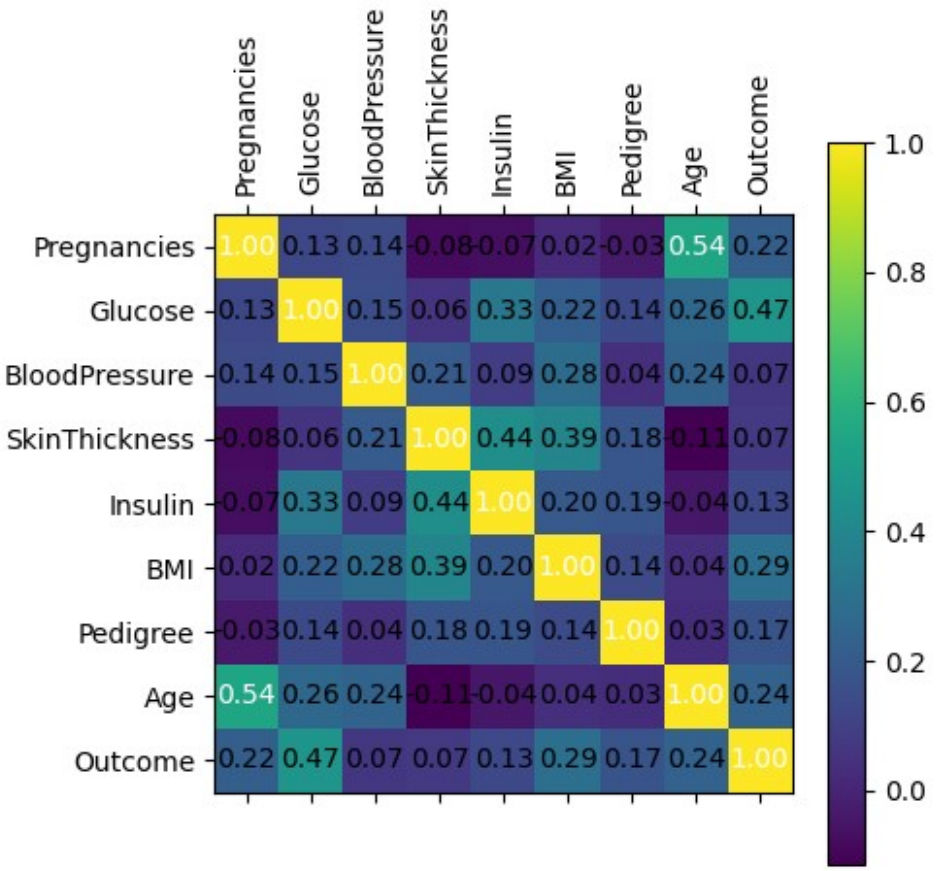
Избавимся от выбросов.

--> Колонка возраста не будет иметь выбросов, так как не имеет погрешности измерения.

```
# columns_to_cut_low = ['Glucose', 'BloodPressure']
# columns_to_cut_high = ['SkinThickness', 'Insulin', 'BMI',
'Pedigree']

# low_percentiles = data[columns_to_cut_low].quantile(0.05)
# high_percentiles = data[columns_to_cut_high].quantile(0.95)

# low = data[columns_to_cut_low] < low_percentiles
# high = data[columns_to_cut_high] > high_percentiles

# combined_mask = low.any(axis=1) | high.any(axis=1)

# data = data[~combined_mask]
```

Построим матрицу корреляции.

```
corr = data.corr()
plt.matshow(corr)
plt.yticks(range(len(data.columns)), data.columns)
plt.xticks(range(len(data.columns)), data.columns, rotation=90)
plt.colorbar()
for (i, j), val in np.ndenumerate(corr):
    plt.text(j, i, f'{val:.2f}', ha='center', va='center',
color='white' if abs(val) > 0.5 else 'black')
plt.show()
```



Признаки `Age` и `Pregnancies` имеют высокую корреляцию.

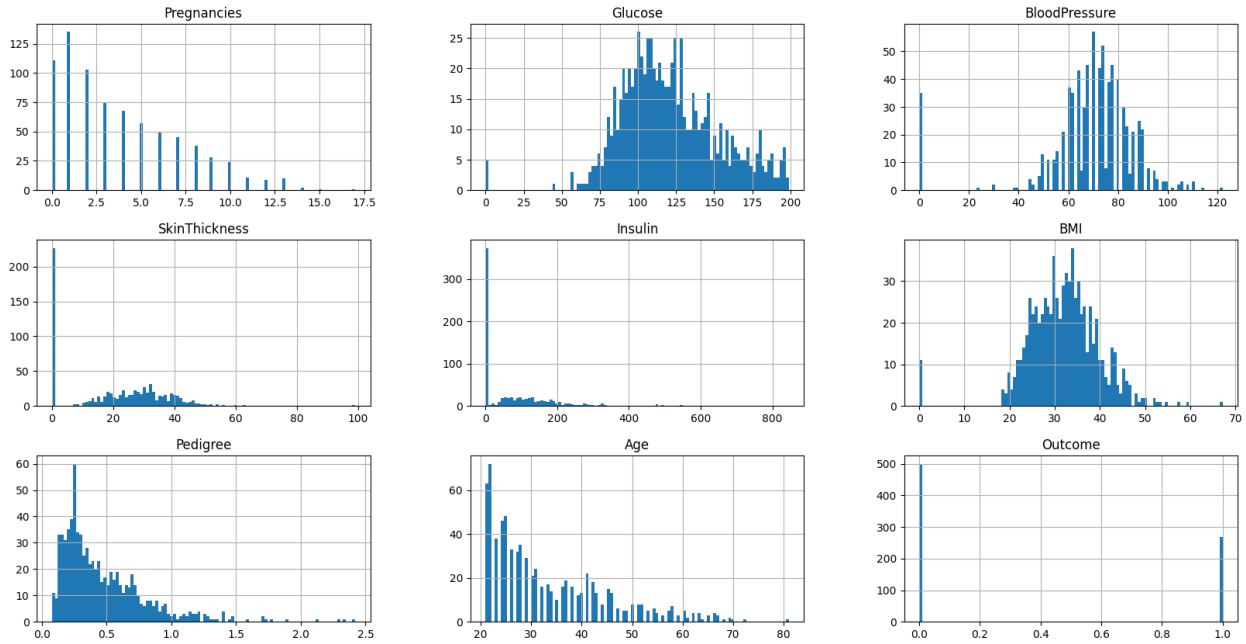
--> Использование матрицы корреляции позволяет оценить степень взаимосвязи между признаками в выборке. Если два или более признака сильно коррелируют друг с другом, это может привести к мультиколлинеарности, что может искажать результаты анализа и делать модель менее точной и надежной. Уберем второй признак.

```
# data.drop('Pregnancies', axis=1, inplace=True)
```

Снова посмотрим на данные

```
data.hist(bins=100, figsize=(20, 10))
```

```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
       [<Axes: title={'center': 'Pedigree'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>]], dtype=object)
```



Нормализация

Заметим, что записей `Outcome` с отрицательным результатом больше остальных.

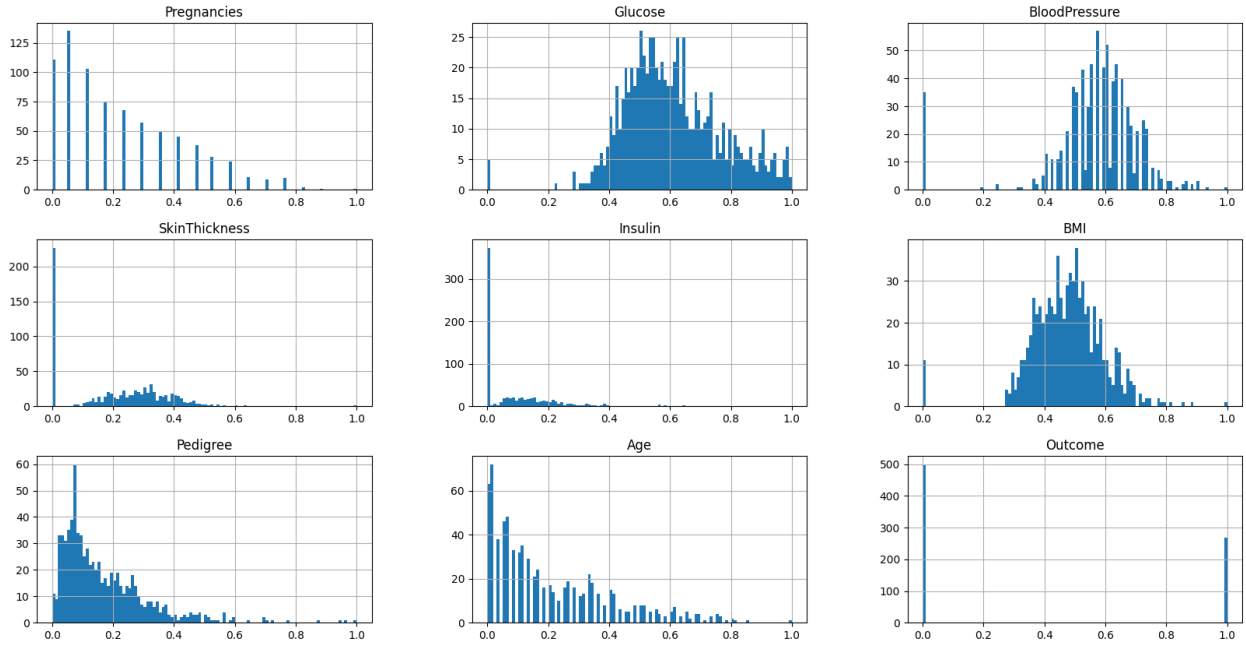
--> Балансирование датасета до равенства исходов категориального признака может быть необходимо в случае, если в исходных данных присутствует значительный дисбаланс между классами этого признака. Например, если у нас есть классификационная задача, и один класс встречается гораздо чаще других, модель может быть склонна к предсказыванию этого класса, игнорируя менее часто встречающиеся классы. Сбалансируем датасет до равенства исходов `Outcome` путем случайной выборки строк из основного класса.

```
# desired_samples = data['Outcome'].value_counts().min()
# data = data.groupby('Outcome').apply(lambda x:
x.sample(desired_samples)).reset_index(drop=True)
```

Используем min-max нормализацию.

```
def min_max_standardize_data(data, columns=[]):
    standardized_data = data.copy()
    for column in columns:
        # x' = (x - min(x)) / (max(x) - min(x))
        standardized_data[column] = (data[column] -
np.min(data[column])) / (np.max(data[column]) - np.min(data[column]))
    return standardized_data

data = min_max_standardize_data(data, data.columns)
data.hist(bins=100, figsize=(20, 10))
```

```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
       [<Axes: title={'center': 'Pedigree'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>]], dtype=object)
```



Метод K-ближайших соседей

Разделение данных на обучающий и тестовый наборы

```
X = data.drop('Outcome', axis=1)
Y = data['Outcome']

def train_test_split(X, Y, seed, test_percent=0.2):
  random.seed(seed)
  random.shuffle(list(range(len(X))))

  test_size = int(len(X) * test_percent)

  x_train = X[test_size:]
  x_test = X[:test_size]
  y_train = Y[test_size:]
  y_test = Y[:test_size]

  return x_train, x_test, y_train, y_test
```

Реализация KNN

```
class KNN:
  def __init__(self, k=3):
    self.k = k

  def fit(self, x_train, y_train):
    self.x_train = x_train
    self.y_train = y_train

  def predict(self, x_test):
    return np.array([self.predict_test(x) for x in x_test])

  def predict_test(self, x):
    dists = [np.sqrt(np.sum((x - x_train_idx)**2)) for x_train_idx in
self.x_train]

    k_idx = np.argsort(dists)[:self.k]
    k_nearest_labels = [self.y_train[i] for i in k_idx]

    most_common = np.bincount(k_nearest_labels).argmax()
    return most_common
```

Определение оценки модели

```
def accuracy_score(y_test, y_pred):
    correct_predictions = np.sum(y_test == y_pred)
    total_predictions = len(y_test)
    return correct_predictions / total_predictions

def error_matrix(pred_y, true_y, n):
    res = np.zeros((n, n))
    for pred, true in zip(pred_y, true_y):
        res[int(pred), int(true)] += 1
    return res

def show_matrix(ax, pred_y, true_y, n):
    res = error_matrix(pred_y, true_y, n)
    ax.matshow(res)
    ax.set_xlabel('True class')
    ax.set_ylabel('Predicted class')
    for (i, j), z in np.ndenumerate(res):
        ax.text(j, i, str(int(z)), ha='center', va='center')
```

Модель 1 (Модель с случайным набором признаков)

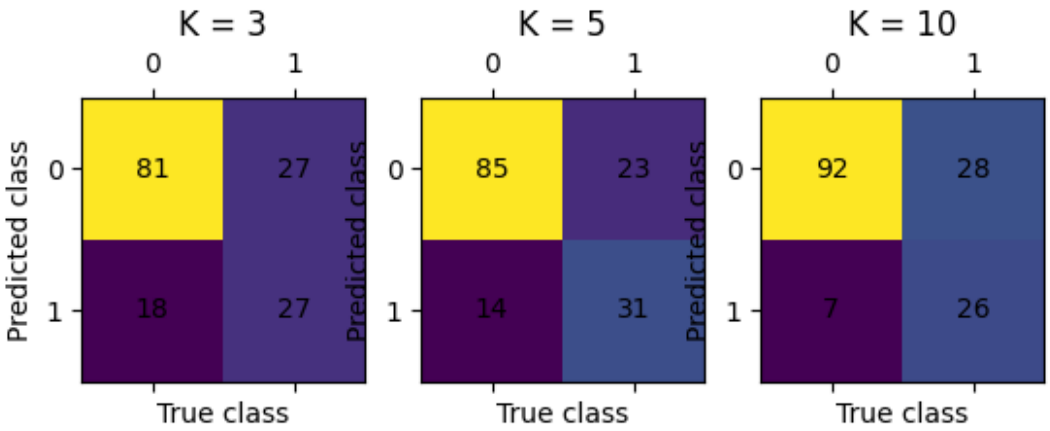
```
def random_features_knn(X, Y, k, subplot_i):
    selected_features = random.sample(list(X.columns),
random.randint(2, len(X.columns)))
    new_data = X[selected_features]
```

```
    display(new_data.head())

    x_train, x_test, y_train, y_test =
train_test_split(np.array(new_data), np.array(Y), 42, 0.2)
    print(f"Train size: {len(x_train)}")
    print(pd.DataFrame(y_train).value_counts())
    print()
    print(f"Test size: {len(x_test)}")
    print(pd.DataFrame(y_test).value_counts())

    knn = KNN(k=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)

    print("Оценка модели: ", accuracy_score(y_test, y_pred))

    ax = plt.subplot(1, 3, i)
    ax.set_title('K = %d' %k)
    show_matrix(ax, y_pred, y_test, 2)

for i, k in enumerate([3, 5, 10], 1):
    random_features_knn(X, Y, k, i)
```

```
        BMI  Pregnancies  Pedigree  BloodPressure       Age   Insulin
0  0.500745     0.352941  0.234415       0.590164  0.483333  0.000000
1  0.396423     0.058824  0.116567       0.540984  0.166667  0.000000
2  0.347243     0.470588  0.253629       0.524590  0.183333  0.000000
3  0.418778     0.058824  0.038002       0.540984  0.000000  0.111111
4  0.642325     0.000000  0.943638       0.327869  0.200000  0.198582

Train size: 615
0.0    401
1.0    214
dtype: int64

Test size: 153
0.0    99
1.0    54
dtype: int64
Оценка модели:  0.7058823529411765

   BloodPressure  SkinThickness   Glucose   Insulin       Age
Pedigree  \
0       0.590164       0.353535  0.743719  0.000000  0.483333
0.234415
1       0.540984       0.292929  0.427136  0.000000  0.166667
0.116567
2       0.524590       0.000000  0.919598  0.000000  0.183333
0.253629
3       0.540984       0.232323  0.447236  0.111111  0.000000
```

```
0.038002
4       0.327869       0.353535  0.688442  0.198582  0.200000
0.943638

        BMI
0  0.500745
1  0.396423
2  0.347243
3  0.418778
4  0.642325

Train size: 615
0.0    401
1.0    214
dtype: int64

Test size: 153
0.0    99
1.0    54
dtype: int64
Оценка модели:  0.7581699346405228

   BloodPressure  SkinThickness   Glucose   Insulin       Age
Pedigree  \
0       0.590164       0.353535  0.743719  0.000000  0.483333
0.234415
1       0.540984       0.292929  0.427136  0.000000  0.166667
0.116567
2       0.524590       0.000000  0.919598  0.000000  0.183333
0.253629
3       0.540984       0.232323  0.447236  0.111111  0.000000
0.038002
4       0.327869       0.353535  0.688442  0.198582  0.200000
0.943638

        BMI
0  0.500745
1  0.396423
2  0.347243
3  0.418778
4  0.642325

Train size: 615
0.0    401
1.0    214
dtype: int64

Test size: 153
0.0    99
1.0    54
```

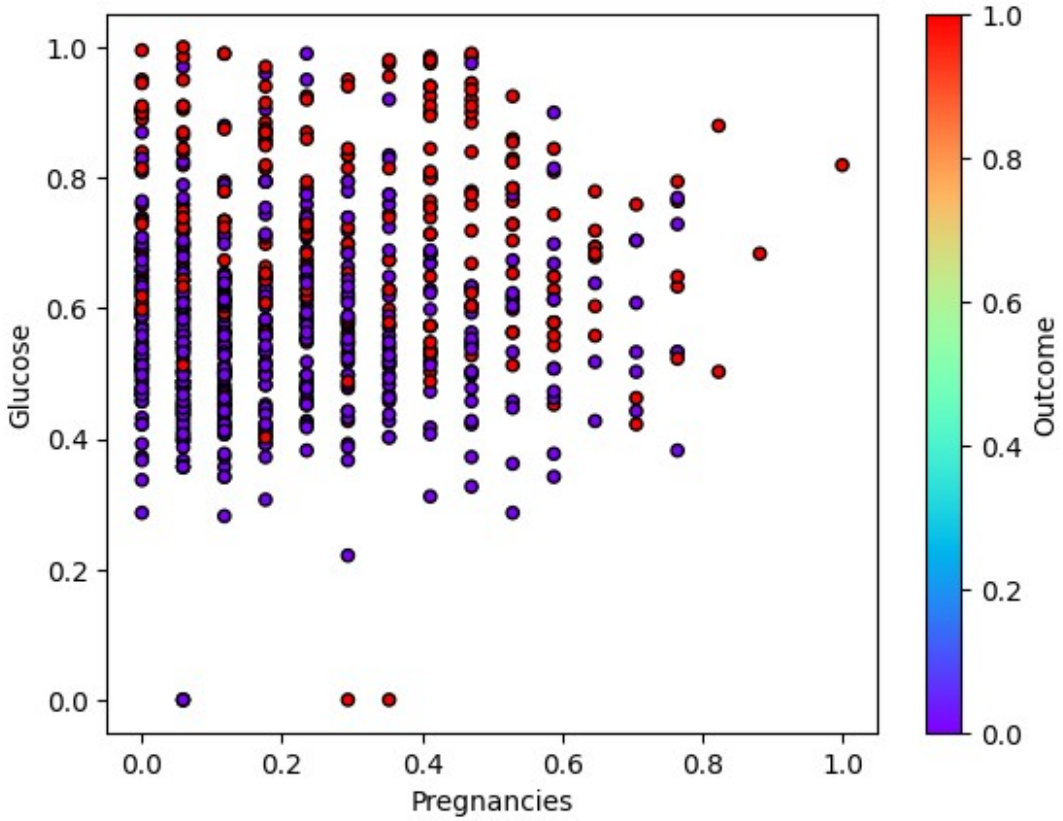
```
dtype: int64
Оценка модели:  0.7712418300653595
```

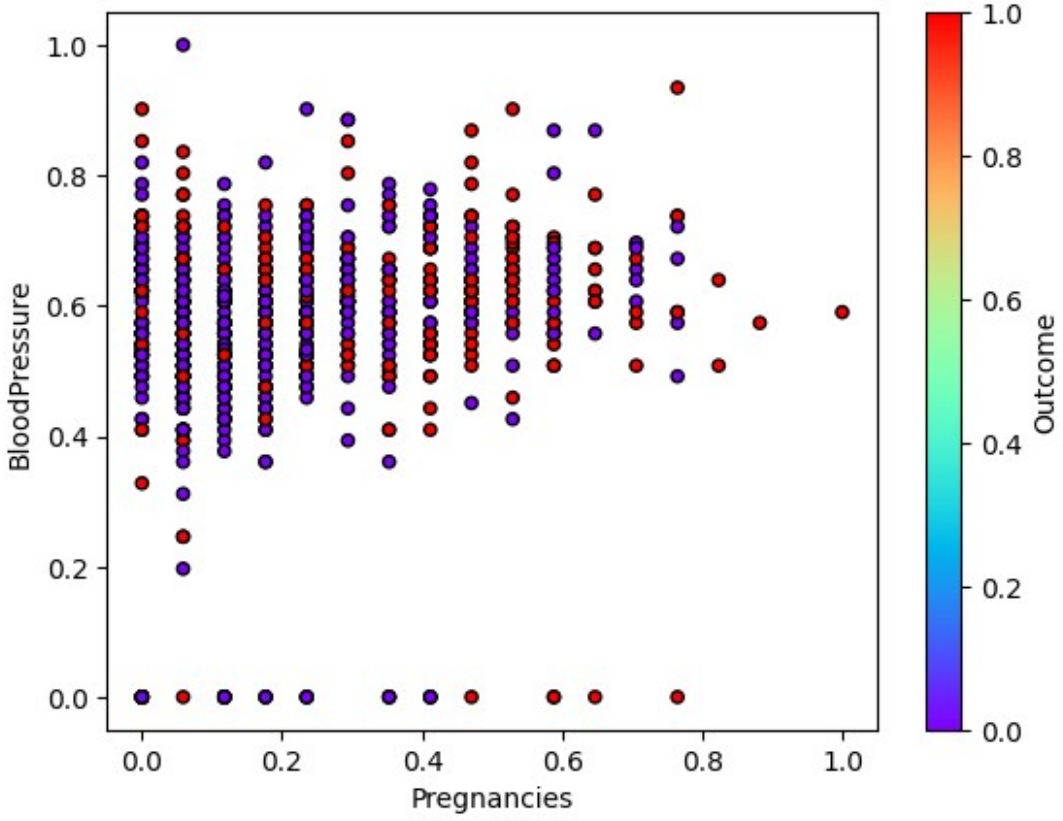


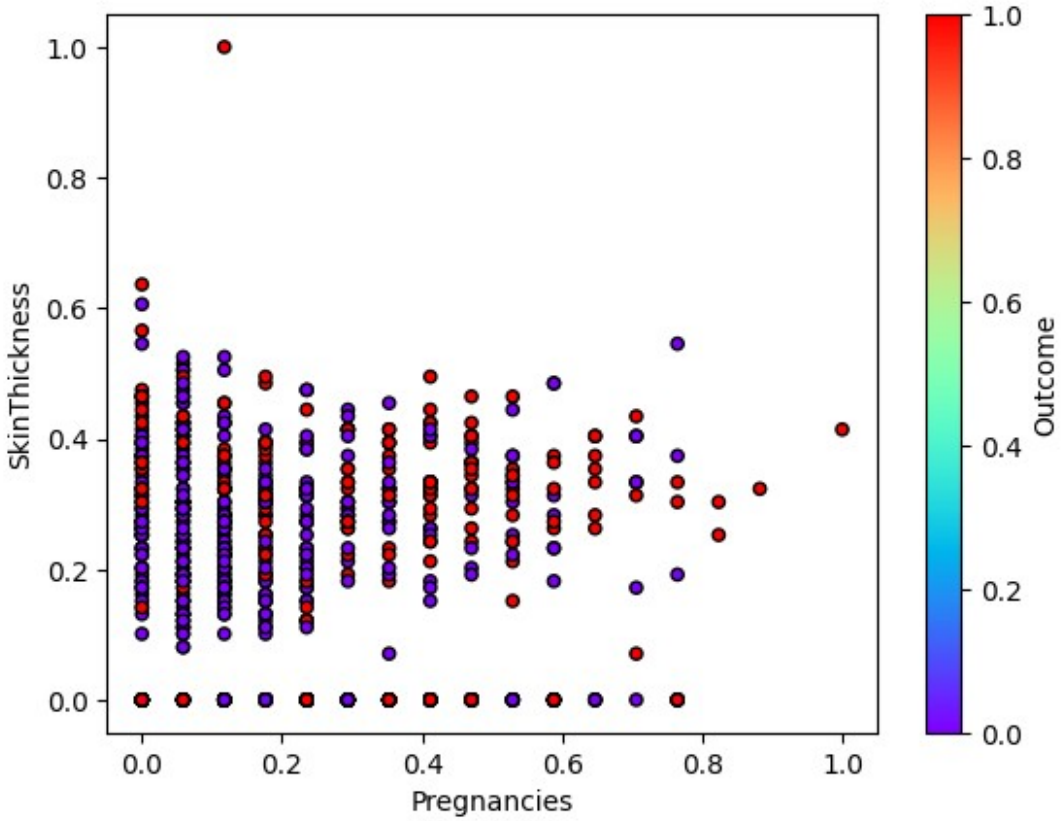
Модель 2 (Фиксированный набор признаков, который выбирается заранее)

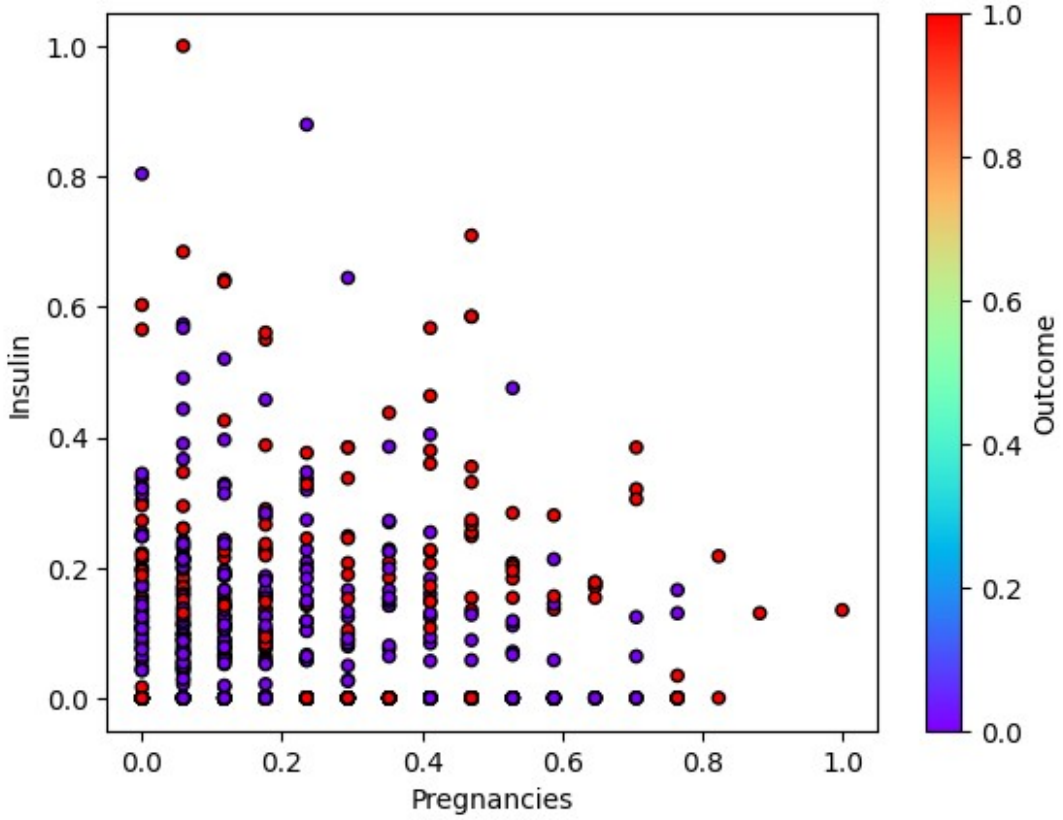
```
import itertools

x_train, x_test, y_train, y_test = train_test_split(X, Y, 42, 0.2)
columns = list(x_train.columns)
combinations = itertools.combinations(columns, 2)

for i_column_name, j_column_name in combinations:
    data.plot.scatter(x=i_column_name, y=j_column_name, c='Outcome',
colormap='rainbow', edgecolor='black');
```

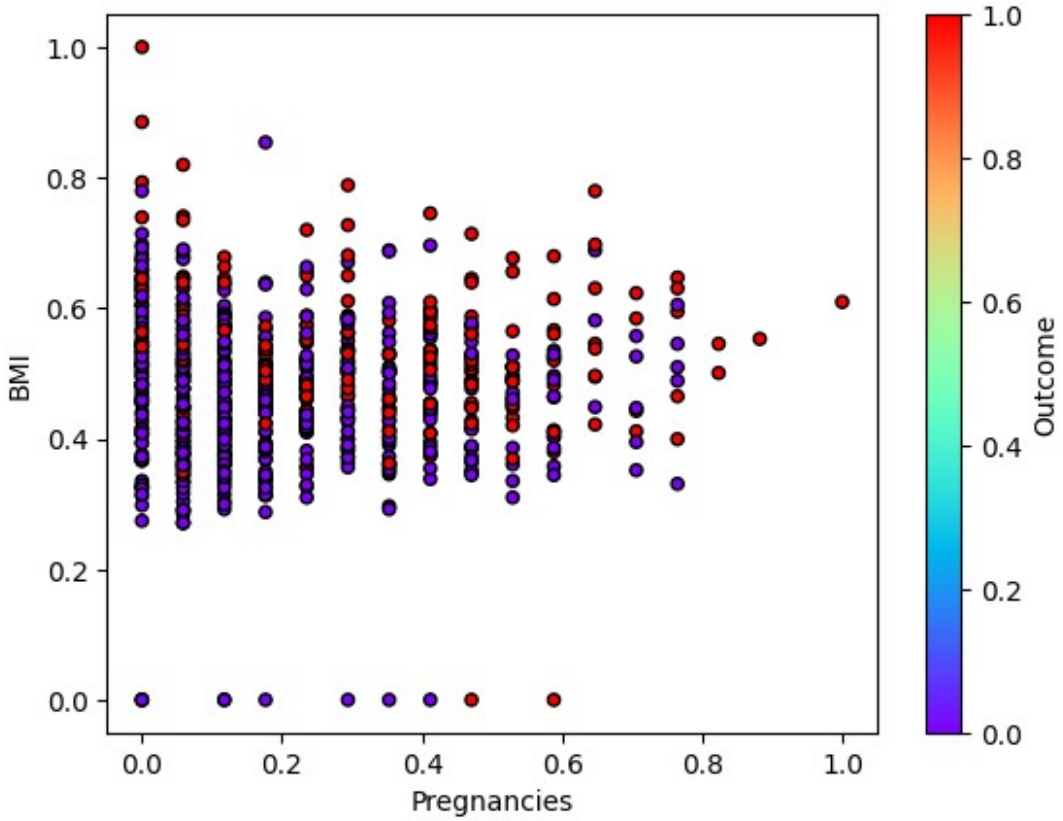
```
C:\Data\Programming\Python310\lib\site-packages\pandas\plotting\
_matplotlib\core.py:512: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using
`matplotlib.pyplot.close()`.
  fig = self.plt.figure(figsize=self.figsize)
```

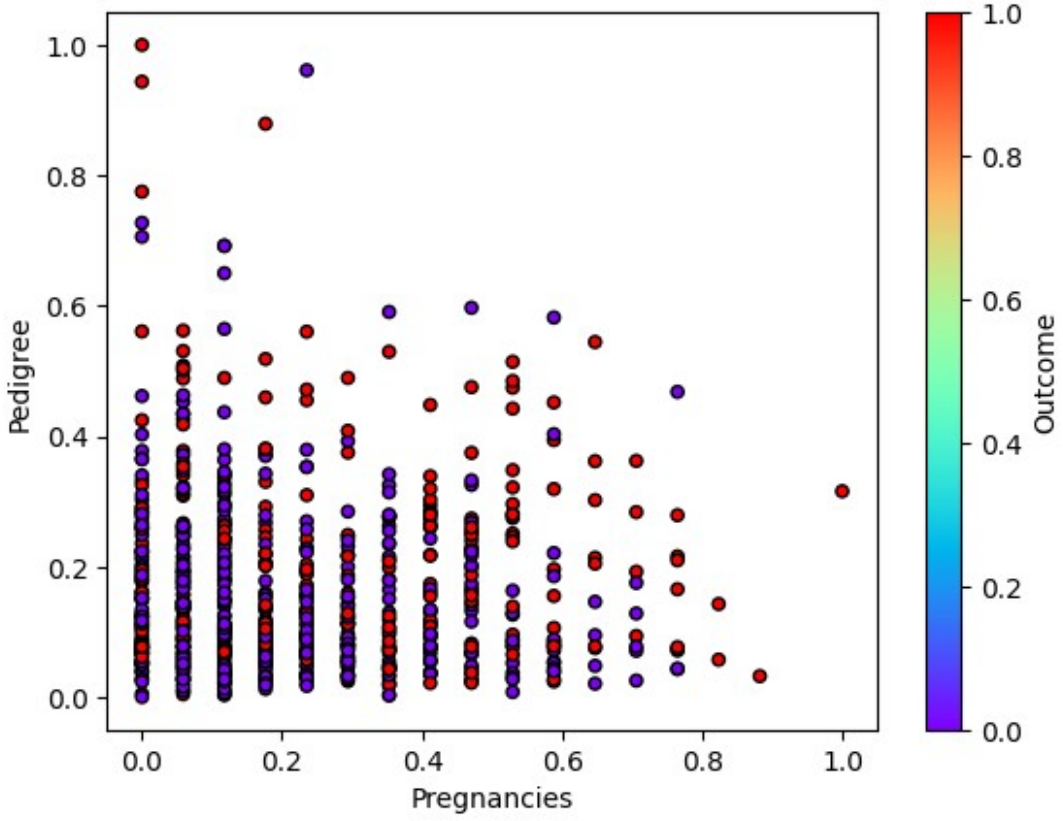

Glucose
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

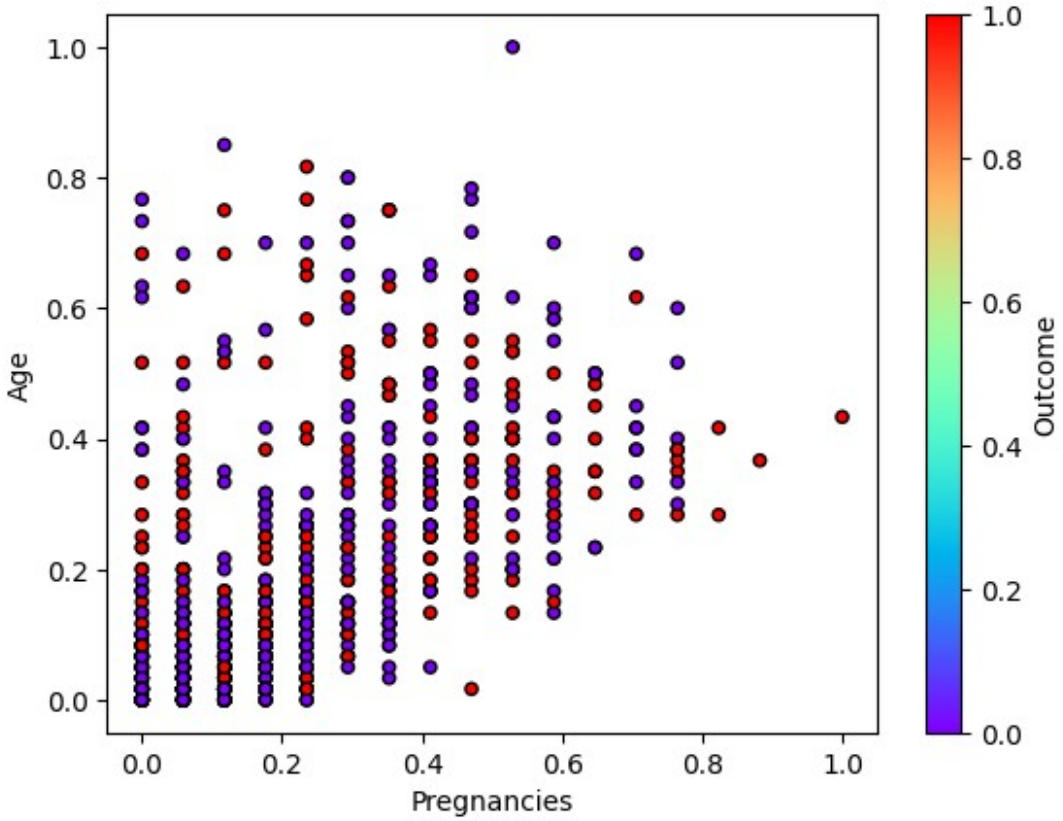

BloodPressure
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

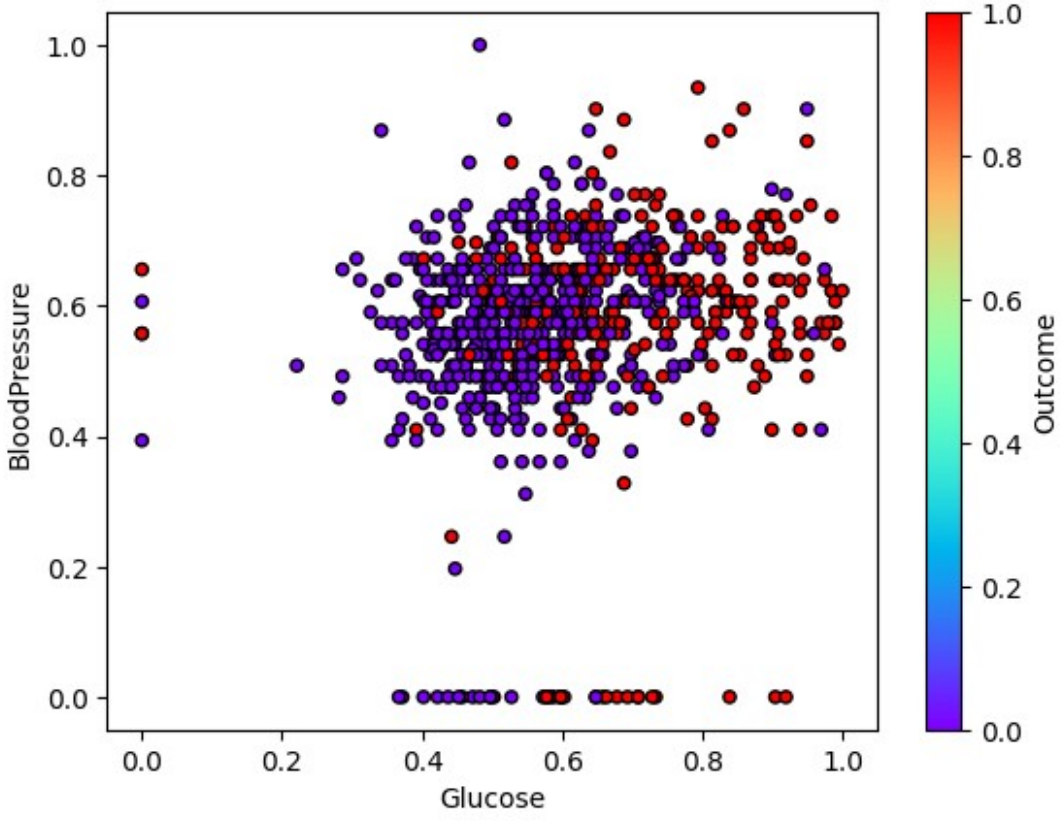

SkinThickness
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

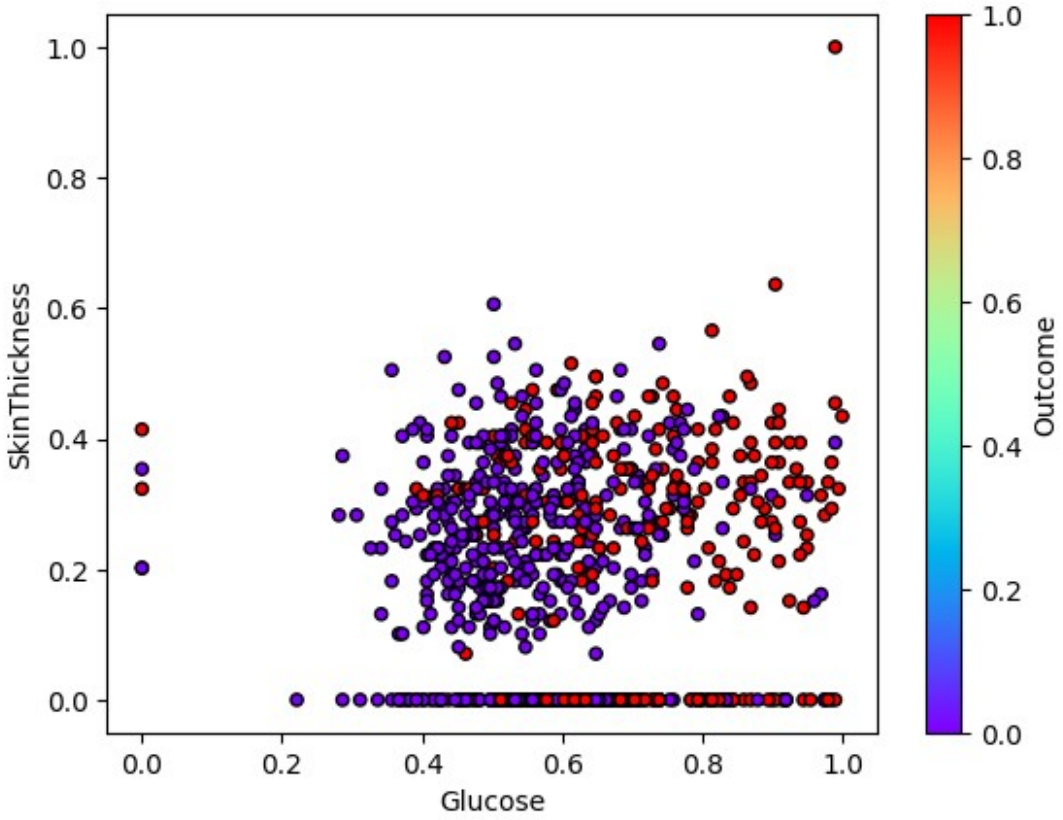

Insulin
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

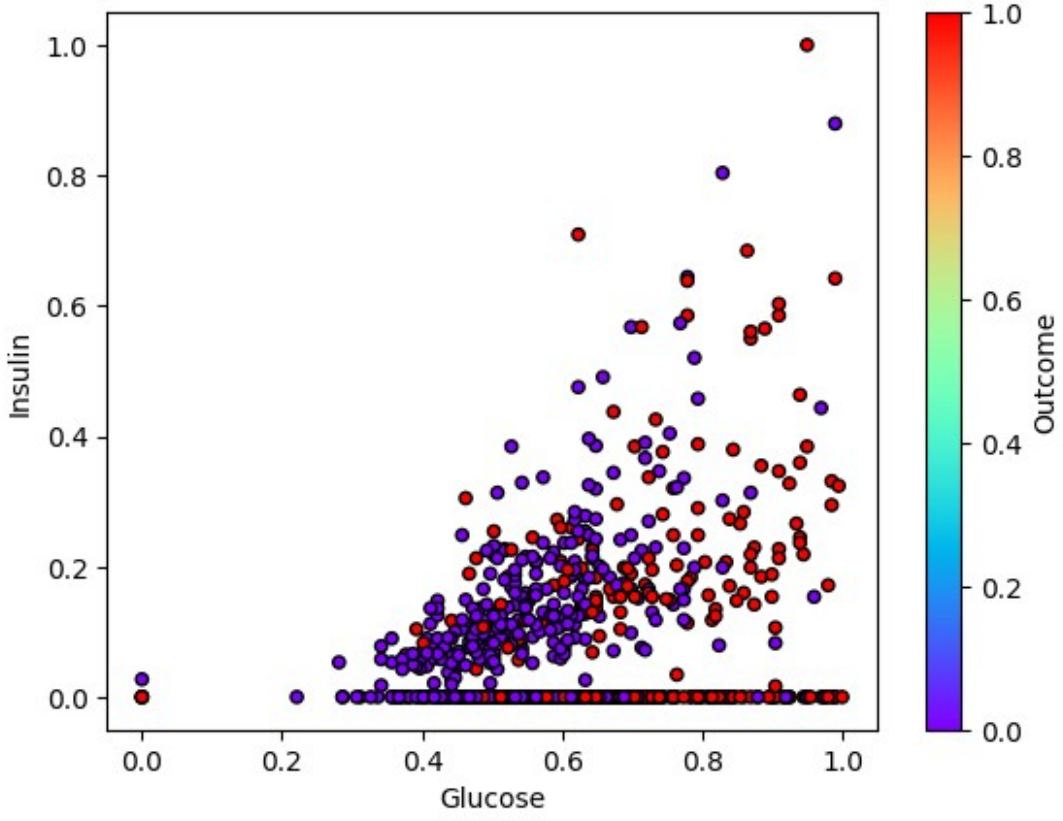

BMI
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

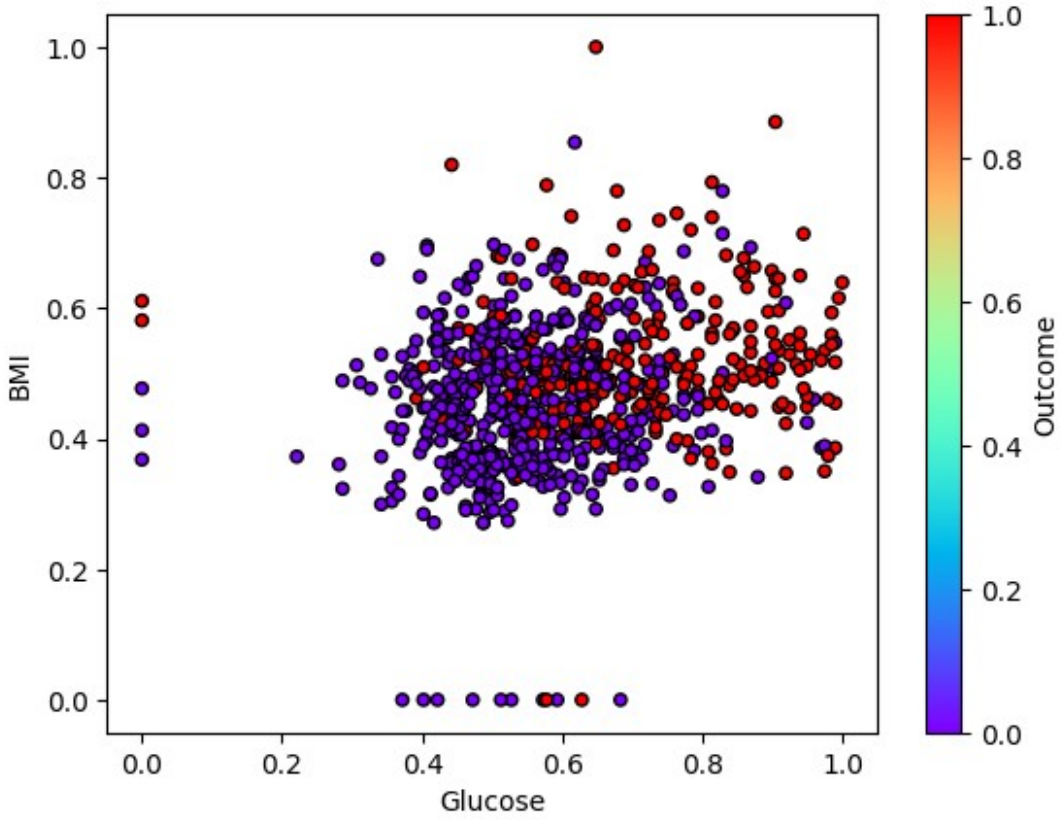

Pedigree
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

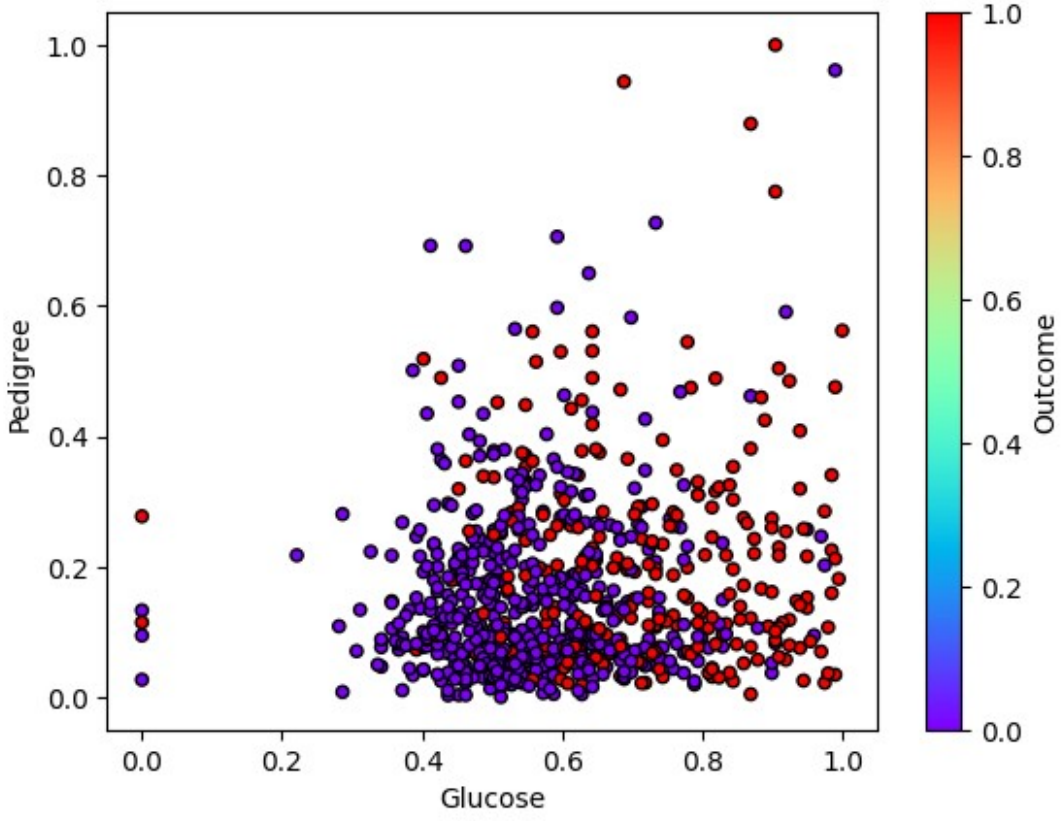

Age
Pregnancies
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

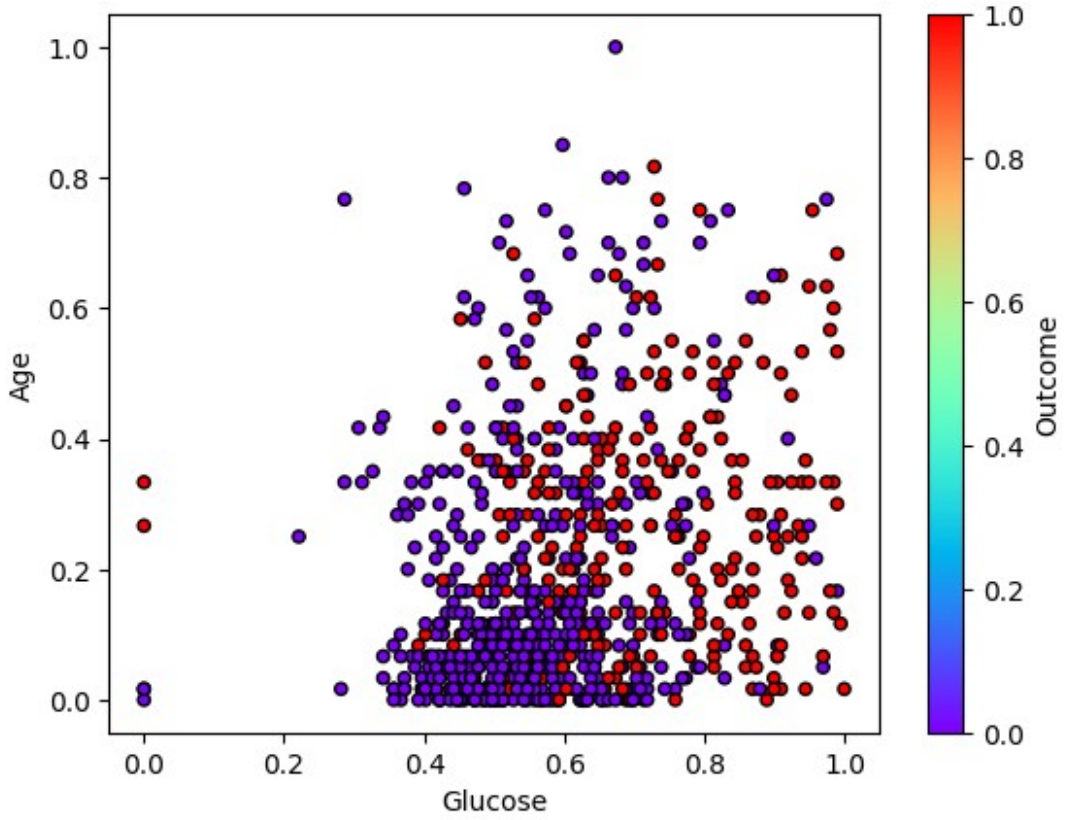

BloodPressure
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

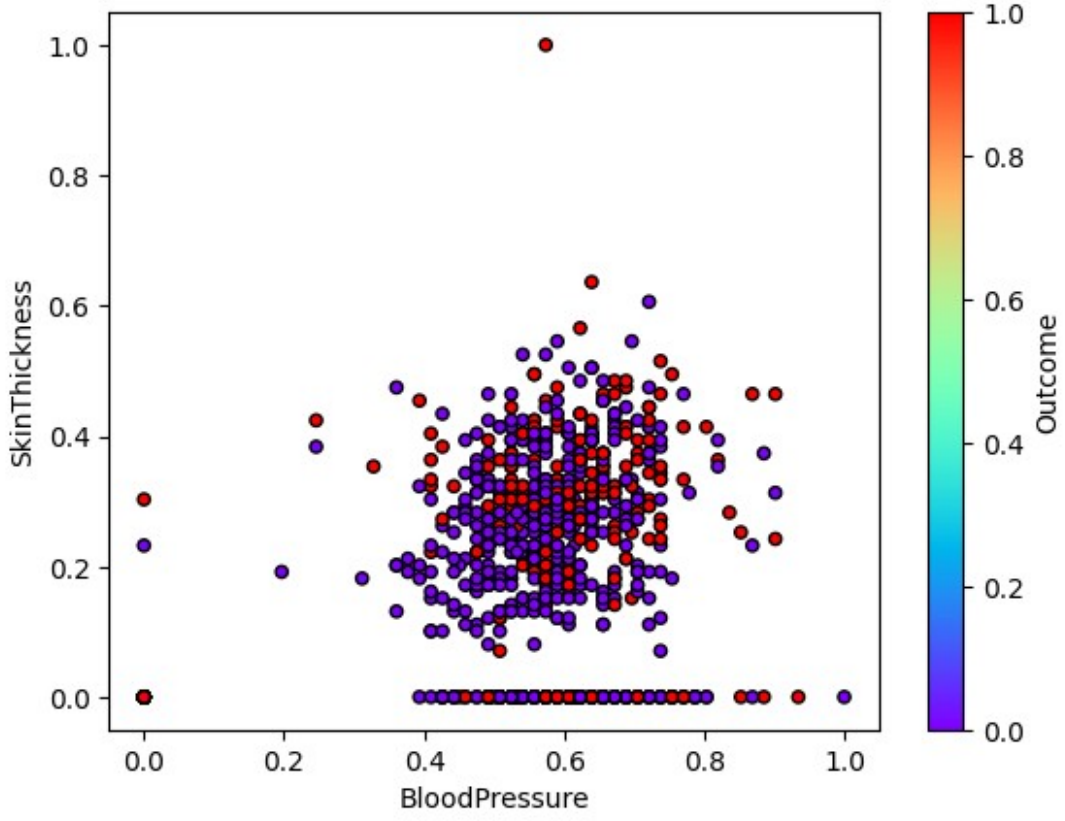

SkinThickness
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

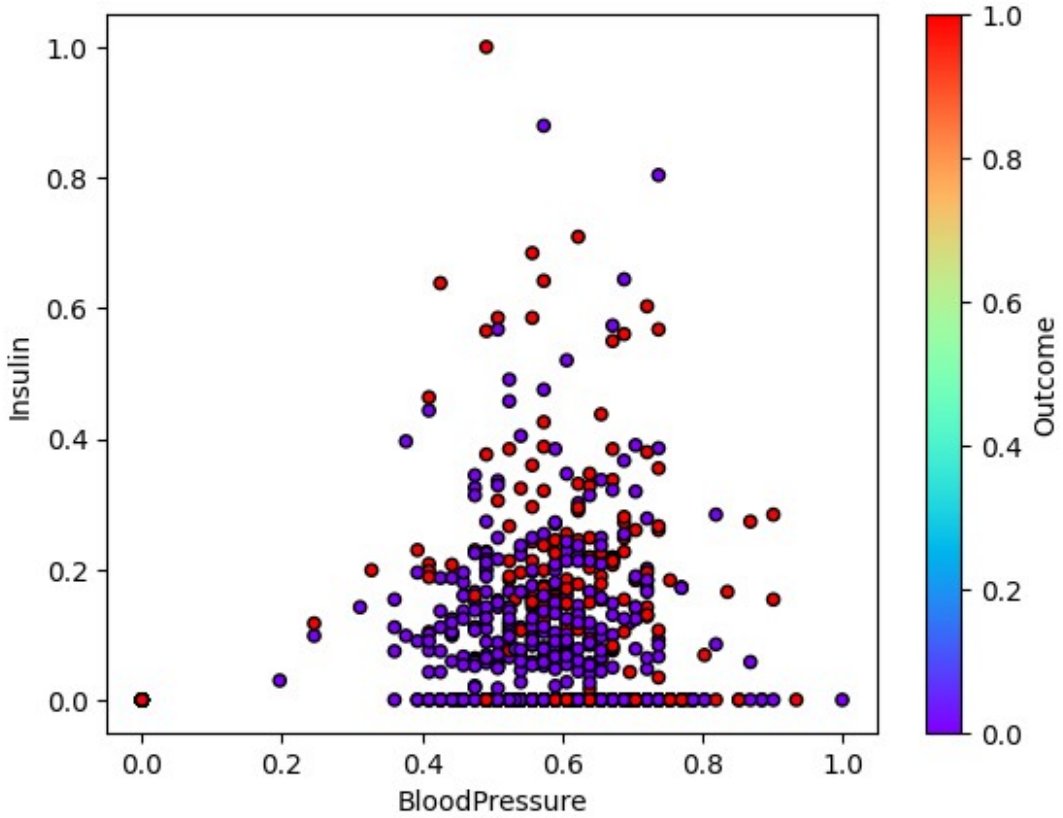

Insulin
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

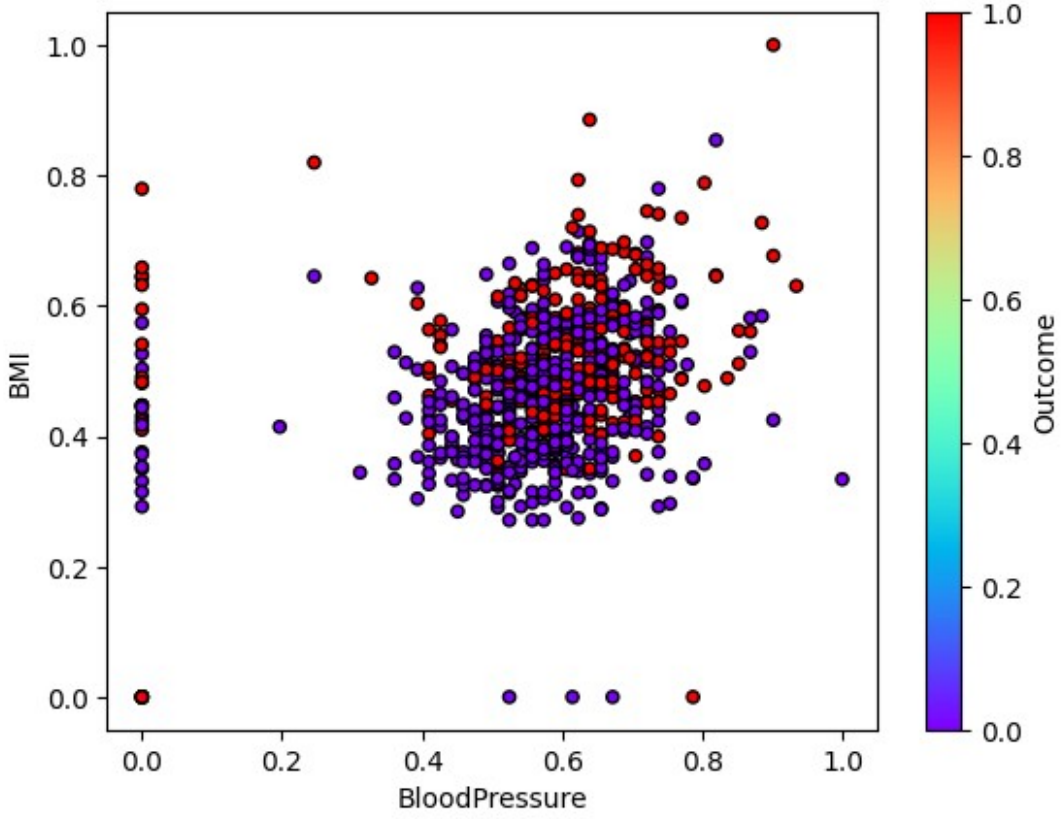

BMI
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

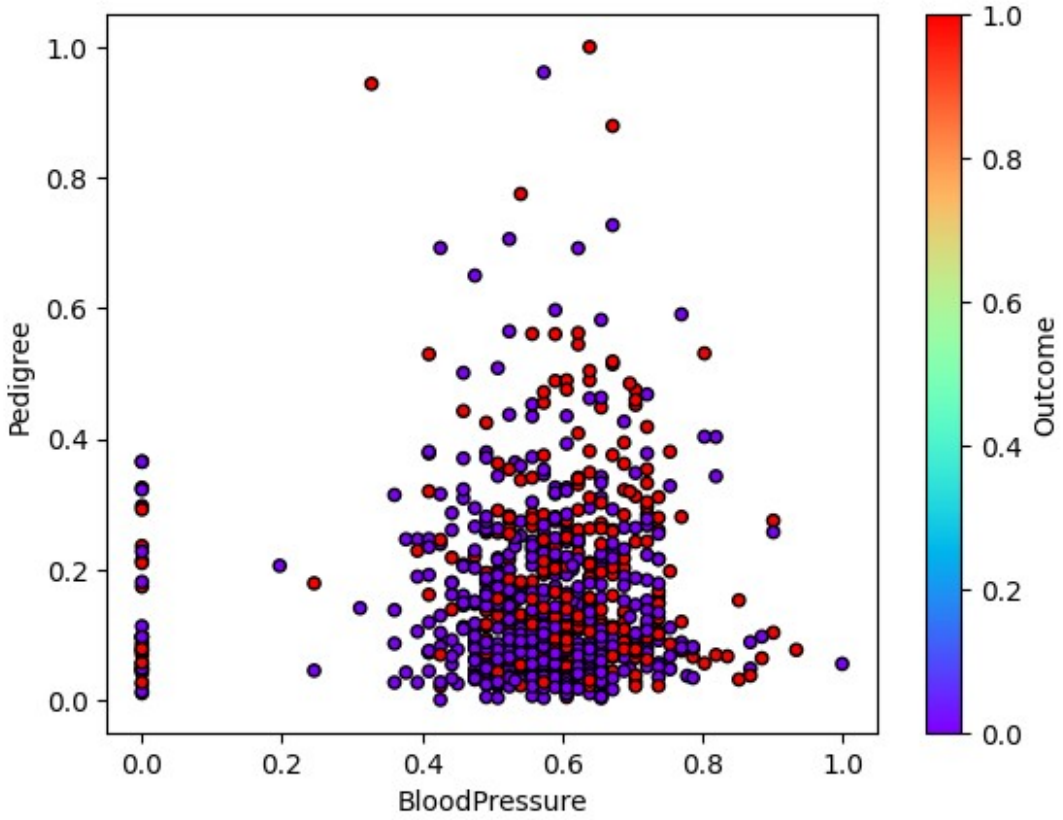

Pedigree
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

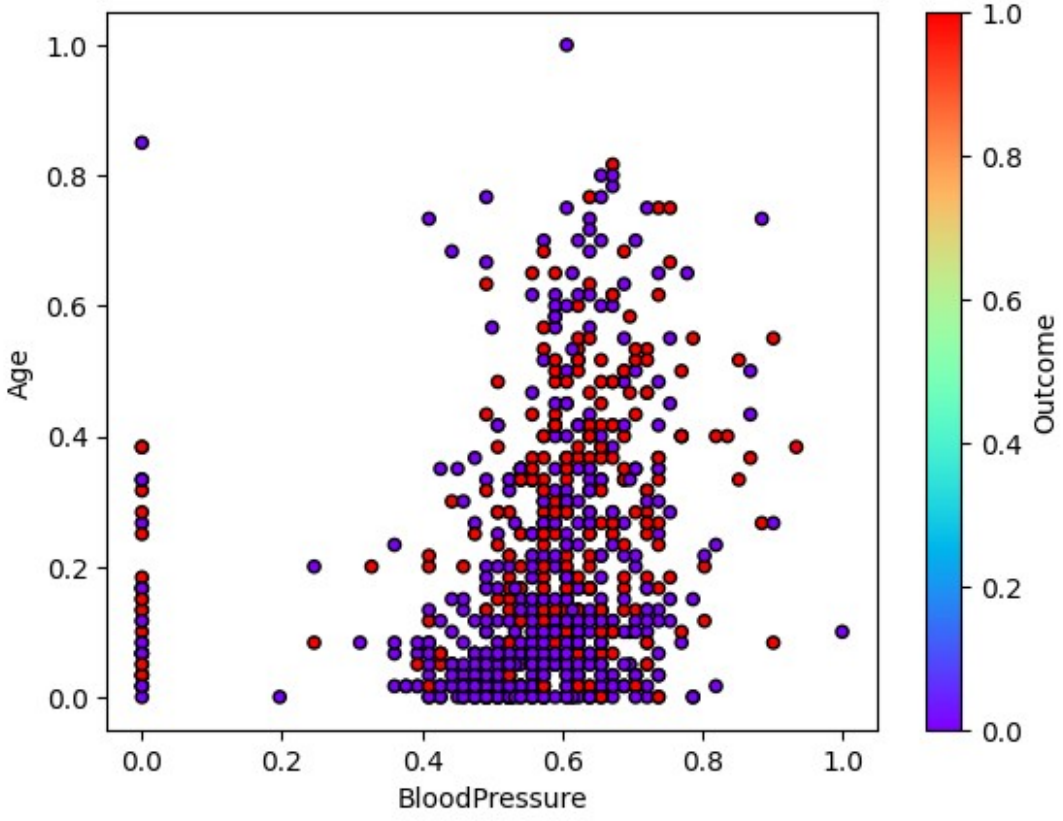

Age
Glucose
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

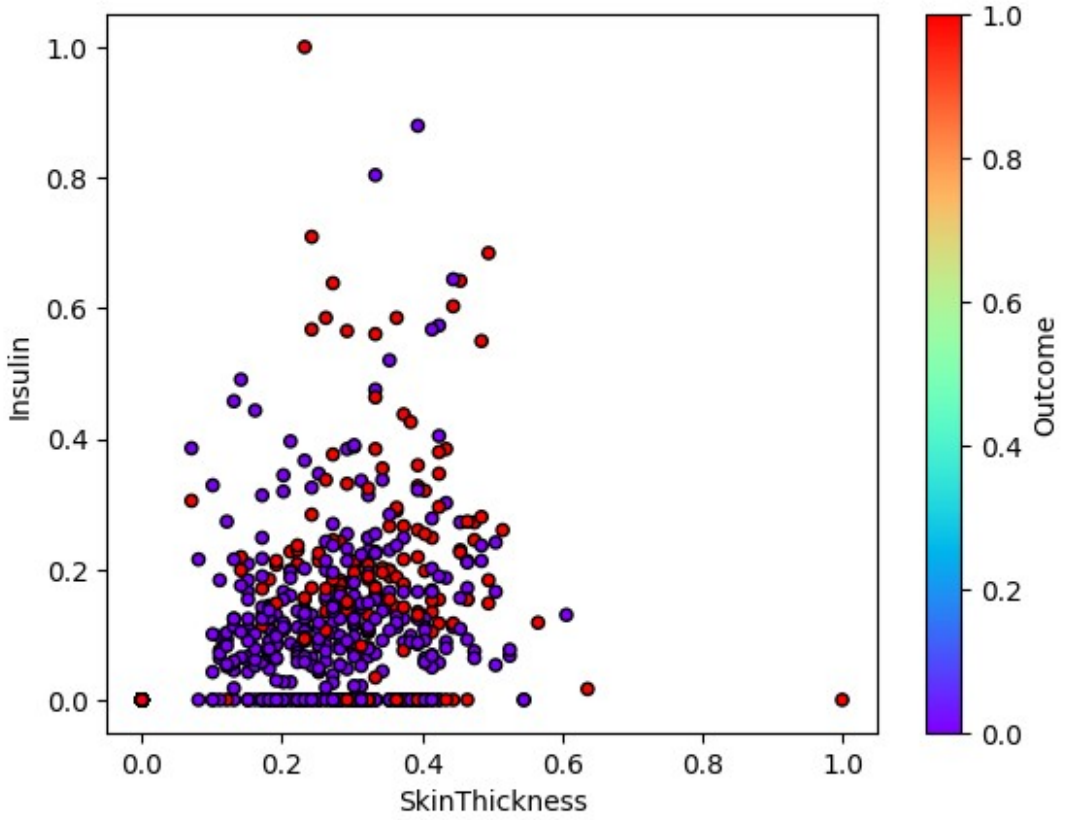

SkinThickness
BloodPressure
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

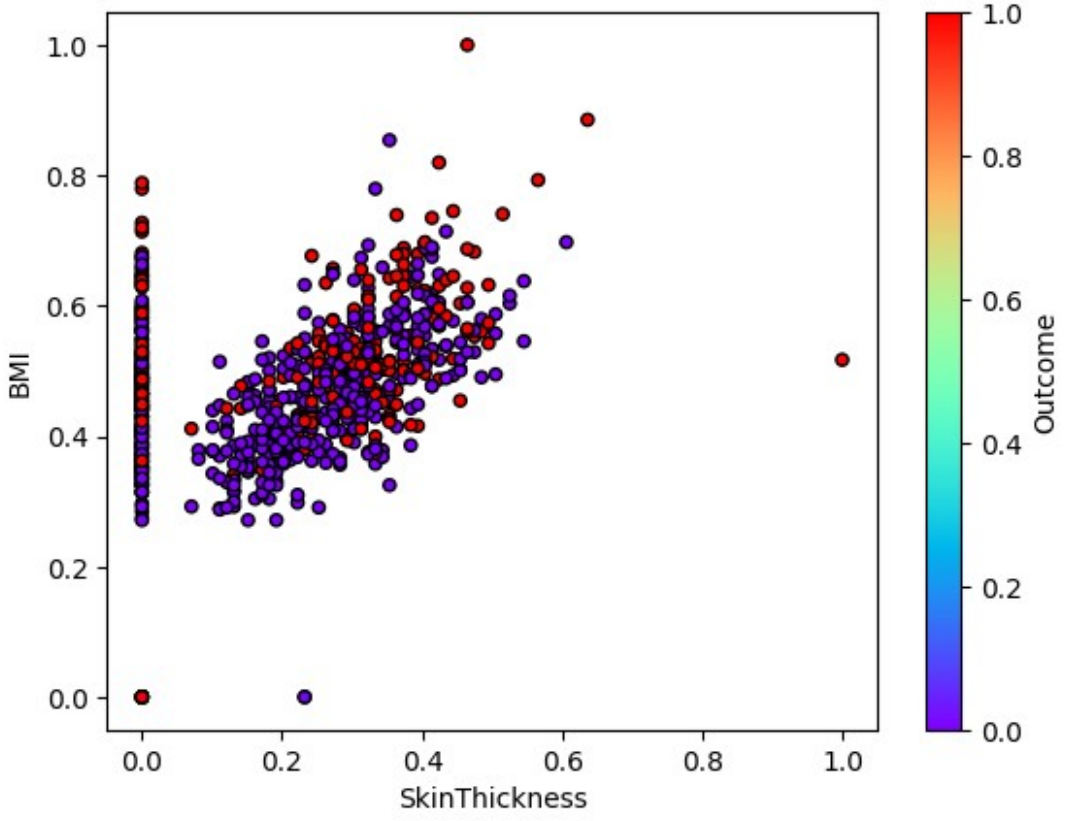

Insulin
BloodPressure
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

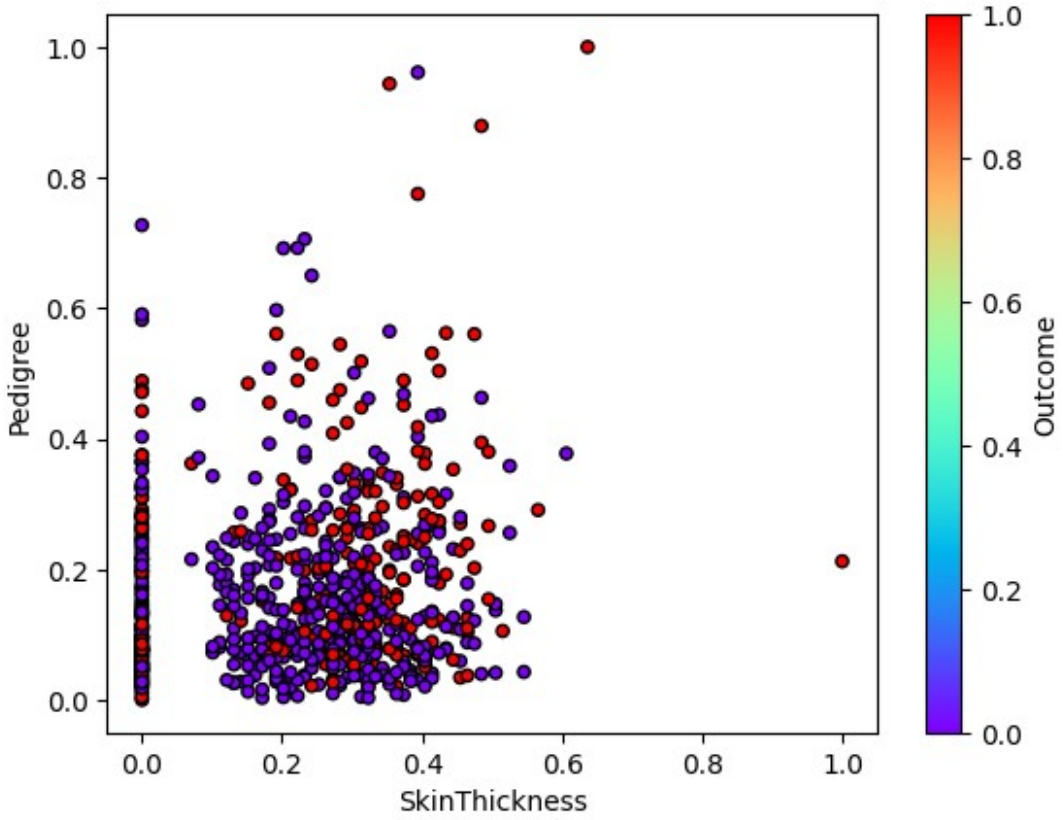

BMI
BloodPressure
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

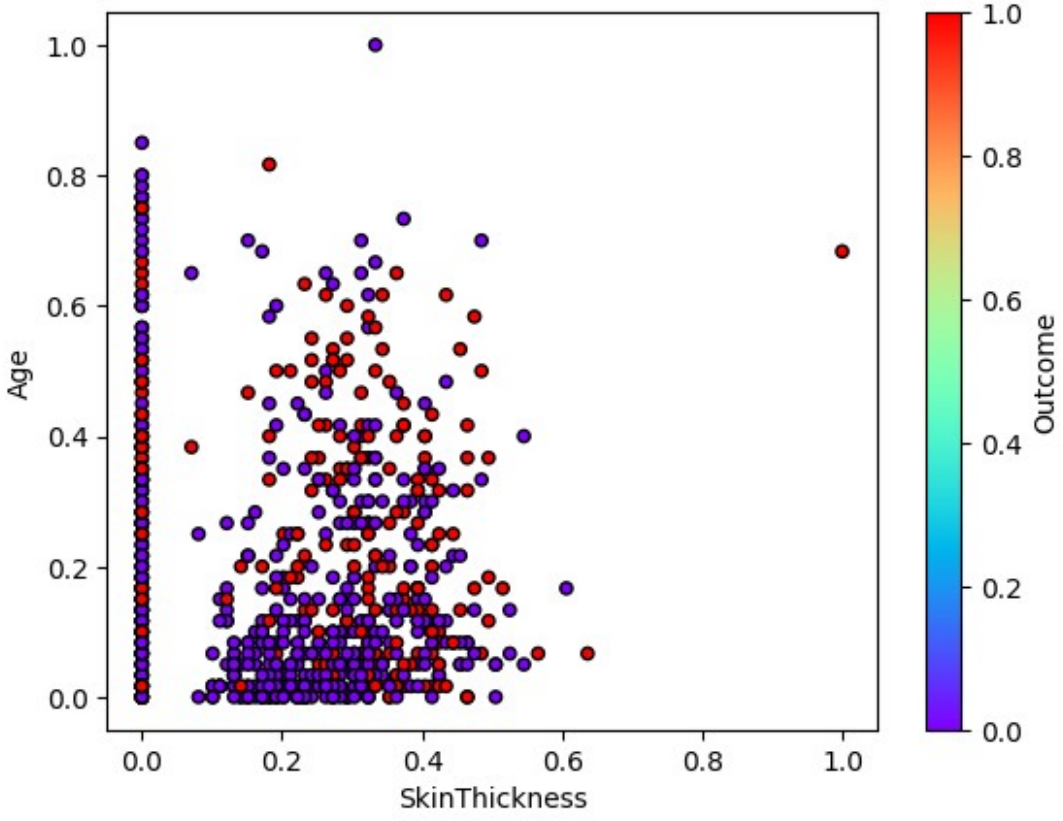

Pedigree
BloodPressure
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

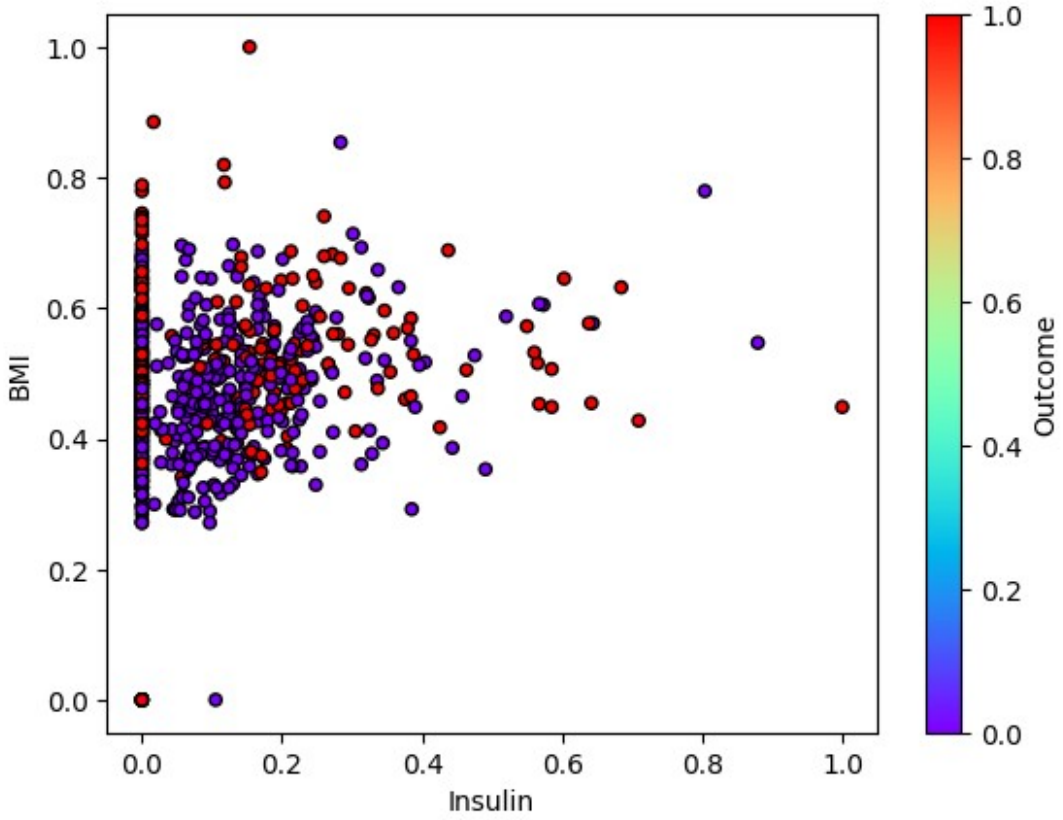

Age
BloodPressure
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

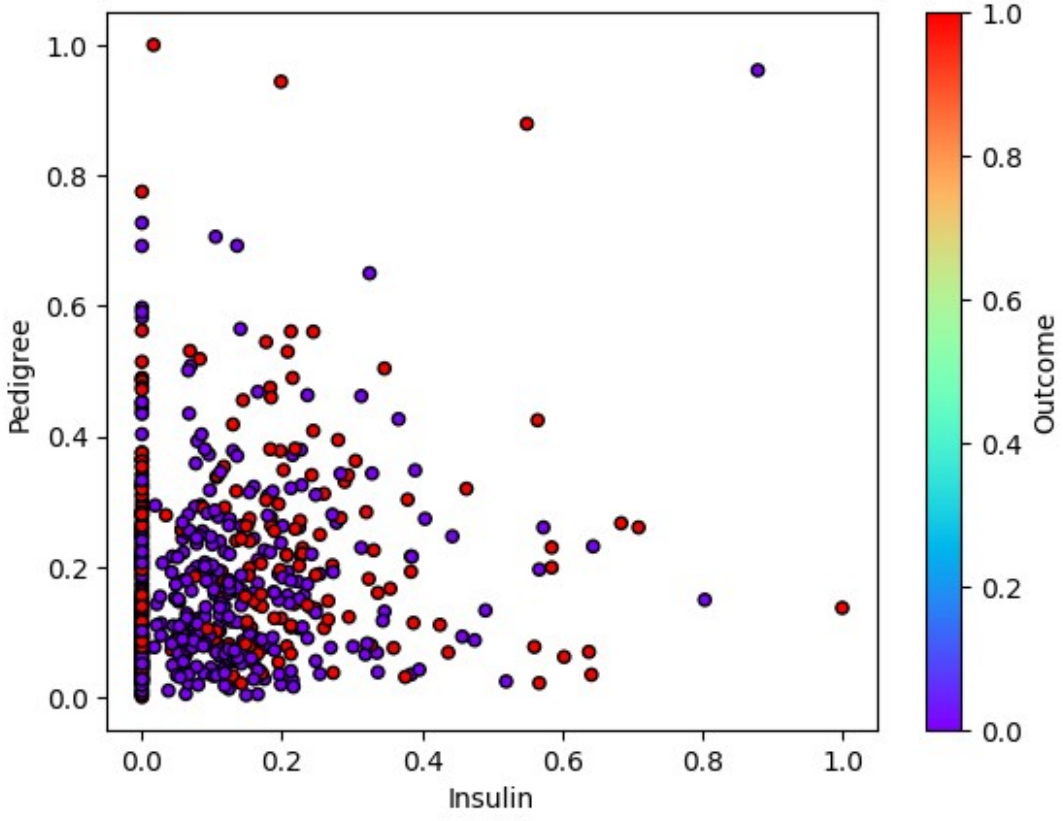

Insulin
SkinThickness
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

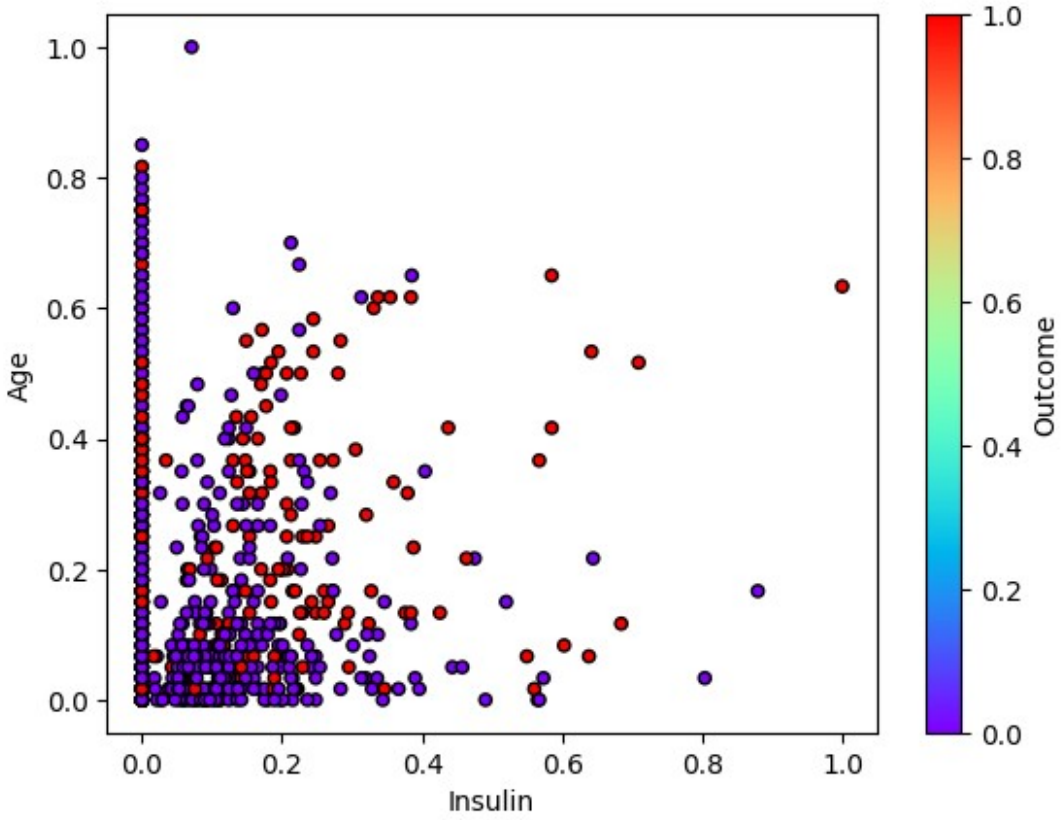

BMI
SkinThickness
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

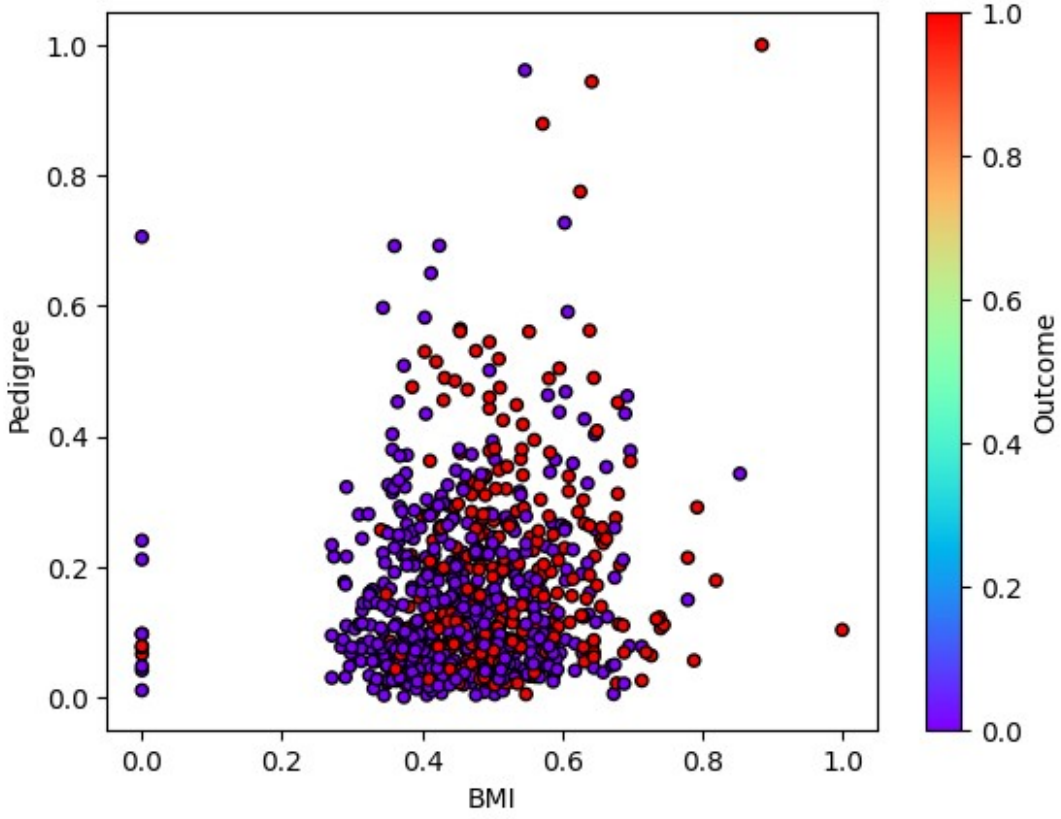

Pedigree
SkinThickness
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

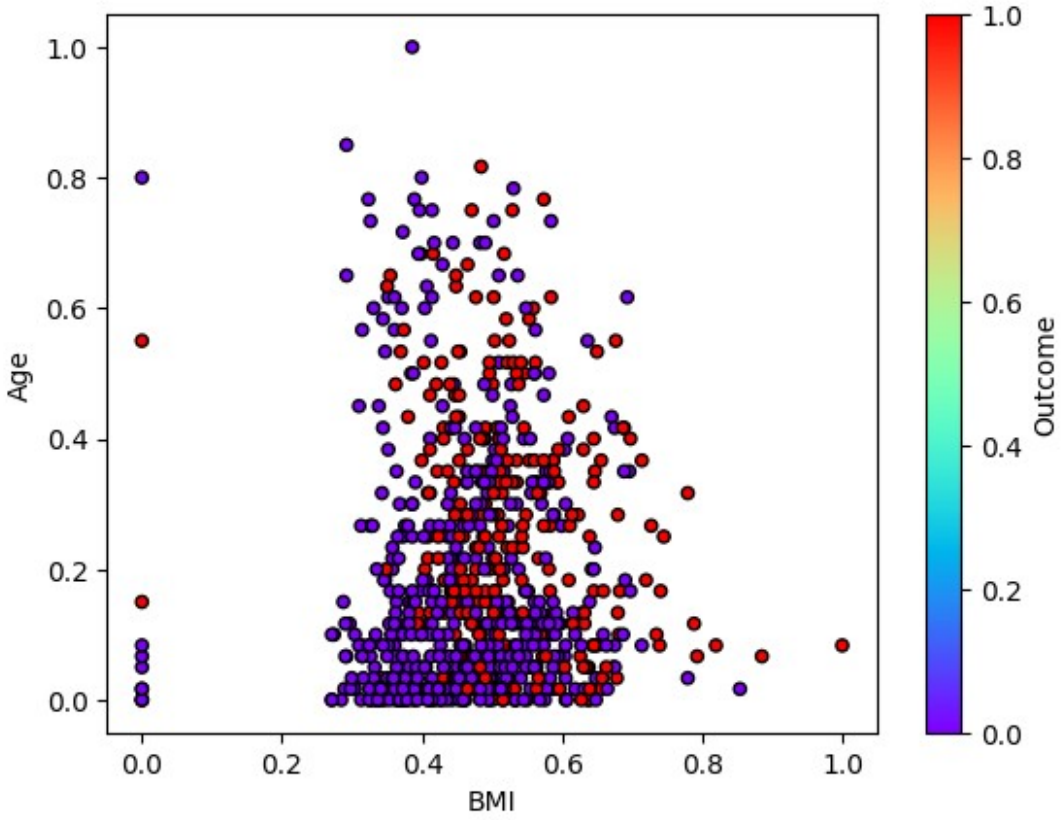

Age
SkinThickness
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

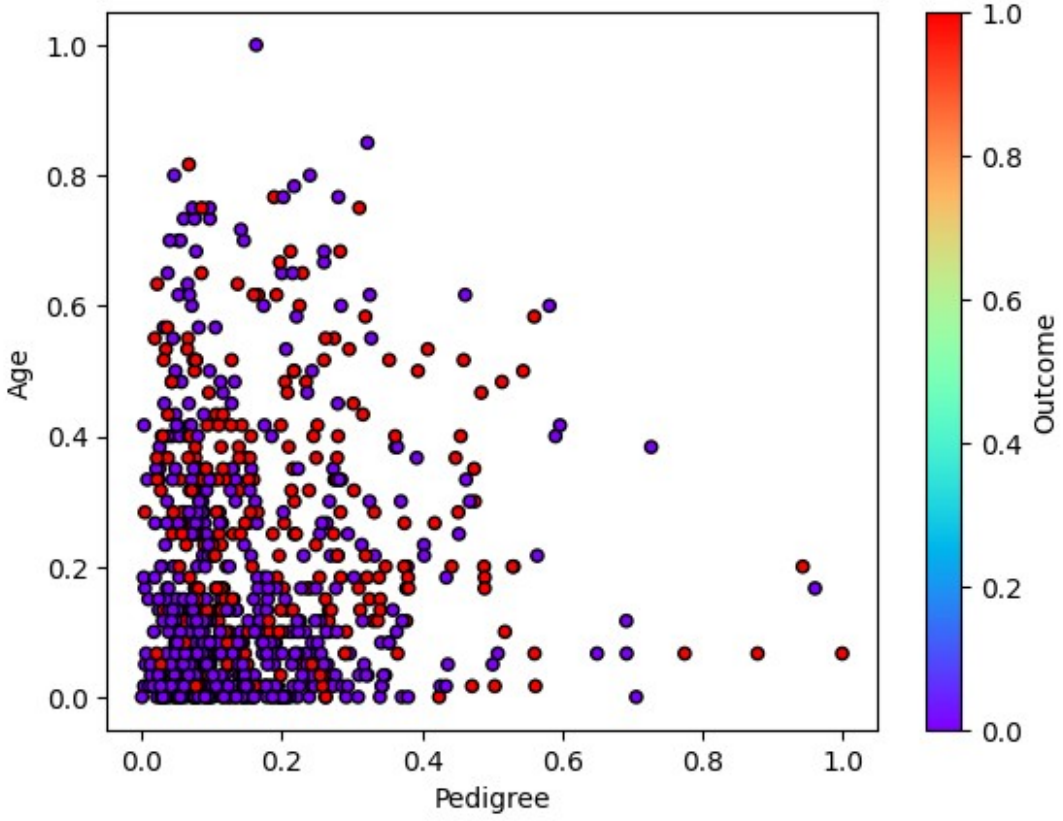

BMI
Insulin
Outcome
0.0
0.2
0.4
0.6
0.8
1.0


Pedigree
Insulin
Outcome
0.0
0.2
0.4
0.6
0.8
1.0


Age
Insulin
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

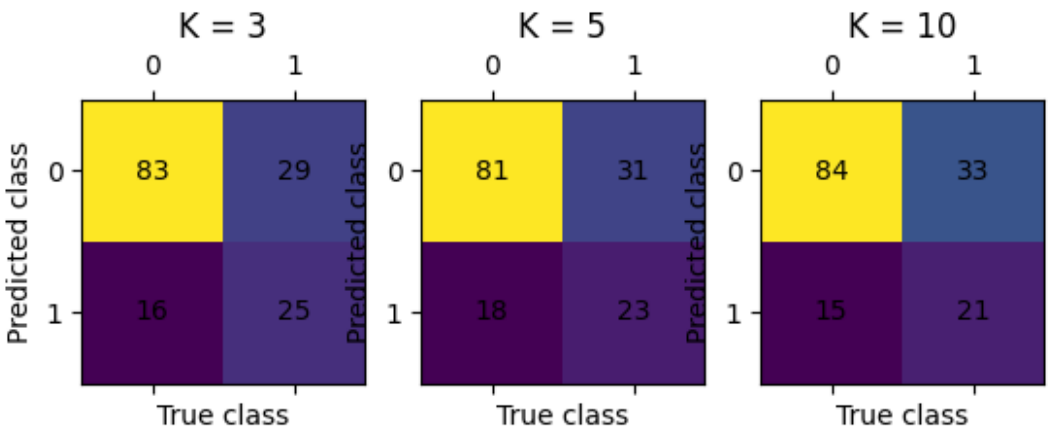

Pedigree
BMI
Outcome
0.0
0.2
0.4
0.6
0.8
1.0


Age
BMI
Outcome
0.0
0.2
0.4
0.6
0.8
1.0


Age
Pedigree
Outcome
0.0
0.2
0.4
0.6
0.8
1.0

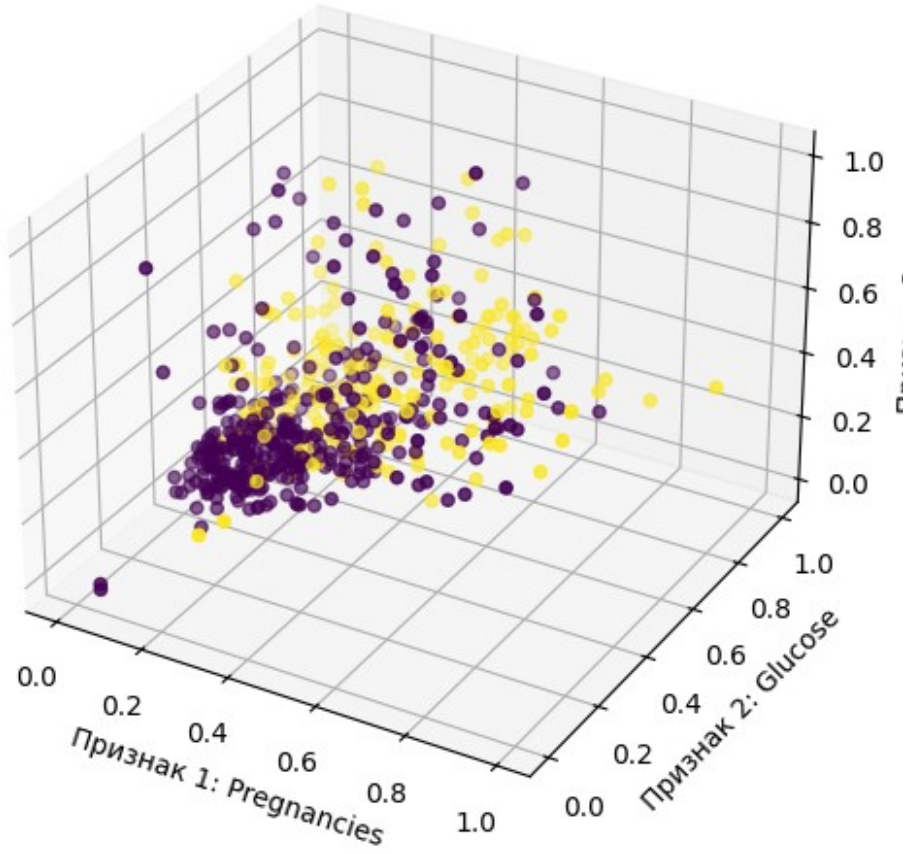
```python
from itertools import combinations

def fix_features_knn(X, Y, k, i):
    x_train, x_test, y_train, y_test = train_test_split(X, Y, 42, 0.2)
    best_accuracy = 0
    best_feature_subset = []
    best_matrix=[]
    print(len(combinations(X.columns, 3)))
    for subset in combinations(X.columns, 3):
        selected_features = list(subset)

        X_train_subset = np.array(x_train[selected_features].values)
        X_test_subset = np.array(x_test[selected_features].values)

        knn = KNN(k=k)
        knn.fit(X_train_subset, np.array(y_train))
        y_pred = knn.predict(X_test_subset)

        accuracy = accuracy_score(np.array(y_test), y_pred)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_feature_subset = selected_features
            best_matrix=error_matrix(np.array(y_test), y_pred, 2)

    print(f"Лучший набор признаков: {best_feature_subset}")
    print(f"Точность: {best_accuracy}")

    ax = plt.subplot(1, 3, i)
    ax.set_title('K = %d' %k)
    show_matrix(ax, y_pred, y_test, 2)

for i, k in enumerate([3, 5, 10], 1):
    print("K =", k)
    fix_features_knn(X, Y, k, i)
```

```
K = 3
Лучший набор признаков: ['Pregnancies', 'Glucose', 'Age']
Точность: 0.7647058823529411
K = 5
Лучший набор признаков: ['Pregnancies', 'BMI', 'Age']
Точность: 0.7712418300653595
K = 10
Лучший набор признаков: ['Pregnancies', 'Glucose', 'BMI']
Точность: 0.7843137254901961
```

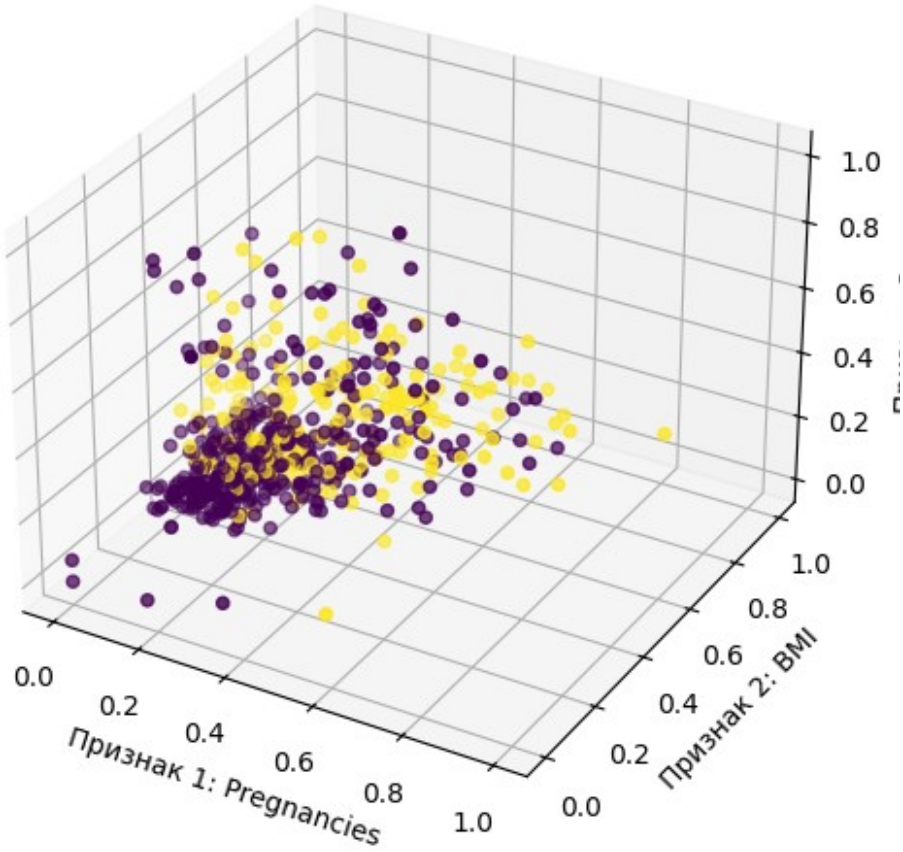



```
from mpl_toolkits.mplot3d import Axes3D

def plot_3d(X, y, features_indices):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection="3d")

    ax.scatter([x[0] for x in X], [x[1] for x in X], [x[2] for x in
X], c=y, cmap="viridis")
    ax.set_xlabel("Признак 1: " + features_indices[0])
    ax.set_ylabel("Признак 2: " + features_indices[1])
    ax.set_zlabel("Признак 3: " + features_indices[2])
    plt.show()

fixed_features_list = {
    3: ['Pregnancies', 'Glucose', 'Age'],
    5: ['Pregnancies', 'BMI', 'Age'],
    10: ['Pregnancies', 'Glucose', 'BMI']
}
for k in [3, 5, 10]:
    fixed_features = fixed_features_list[k]
    print(f"K = {k}, Fixed features: {fixed_features}")
    x_train, x_test, y_train, y_test = train_test_split(X, Y, 42, 0.2)
    x_train_subset = np.array(x_train[fixed_features].values)
    plot_3d(x_train_subset, np.array(y_train), fixed_features)
```

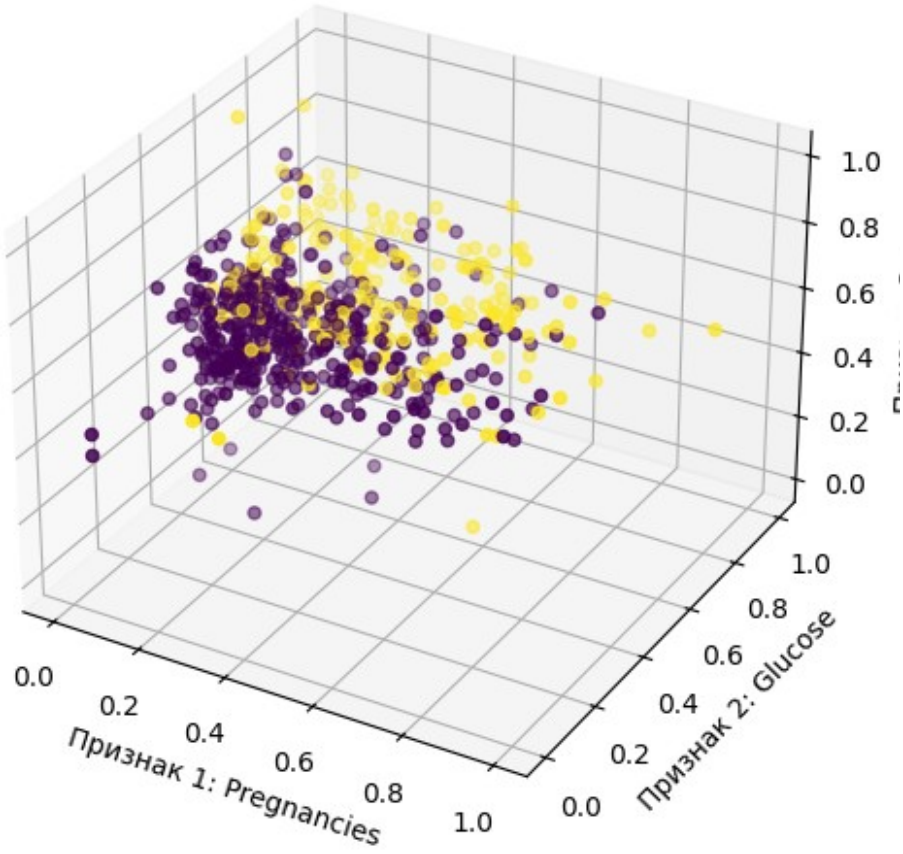
```
K = 3, Fixed features: ['Pregnancies', 'Glucose', 'Age']
```




```
K = 5, Fixed features: ['Pregnancies', 'BMI', 'Age']
```




```
K = 10, Fixed features: ['Pregnancies', 'Glucose', 'BMI']
```


0.0
0.2
0.4
0.6
0.8
1.0
Признак 1: Pregnancies
0.0
0.2
0.4
0.6
0.8
1.0
Признак 2: Glucose
0.0
0.2
0.4
0.6
0.8
1.0