Факультет программной инженерии и компьютерной техники

Системы искусственного интеллекта

Лабораторная работа №5

Метод k-ближайших соседей

Преподаватель: Кугаевских Александр Владимирович
Выполнил: Ле Чонг Дат
Группа: P33302

2023 г.

Лабораторная 5. **Метод k-ближайших соседей**

Задание

- Выбор датасета:
  - Нечетный номер в группе - Датасет про диабет
- Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и масштабирование.
- Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили), постройте 3d-визуализацию признаков.
- Реализуйте метод k-ближайших соседей ****без использования сторонних библиотек, кроме NumPy и Pandas.
- Постройте две модели k-NN с различными наборами признаков:
  - Модель 1: Признаки случайно отбираются .
  - Модель 2: Фиксированный набор признаков, который выбирается заранее.
- Для каждой модели проведите оценку на тестовом наборе данных при разных значениях k. Выберите несколько различных значений k, например, k=3, k=5, k=10, и т. д. Постройте матрицу ошибок.

Step 1: Installing Libraries

This step is usually done once. If you already have these libraries installed, you can skip this step.

In [1]:
```
# Install necessary libraries
# !pip install pandas numpy sklearn
```

Step 2: Importing Libraries and define helper functions

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

In [3]:
```
def print_bold(text):
    BOLD = '\033[1m'
    END = '\033[0m'
    print('-' * 100 + f"\n{BOLD}{text}{END}")
```

In [4]:
```python
def fill_missing_with_mean(column):
    mean_val = column.mean()
    return column.fillna(mean_val)
```

In [5]:
```python
def min_max_scaler(column):
    # Calculate the minimum and maximum values of the column
    min_val = column.min()
    max_val = column.max()

    # Apply the Min-Max scaling formula
    scaled_column = (column - min_val) / (max_val - min_val)

    return scaled_column
```

Step 3: Performing tasks

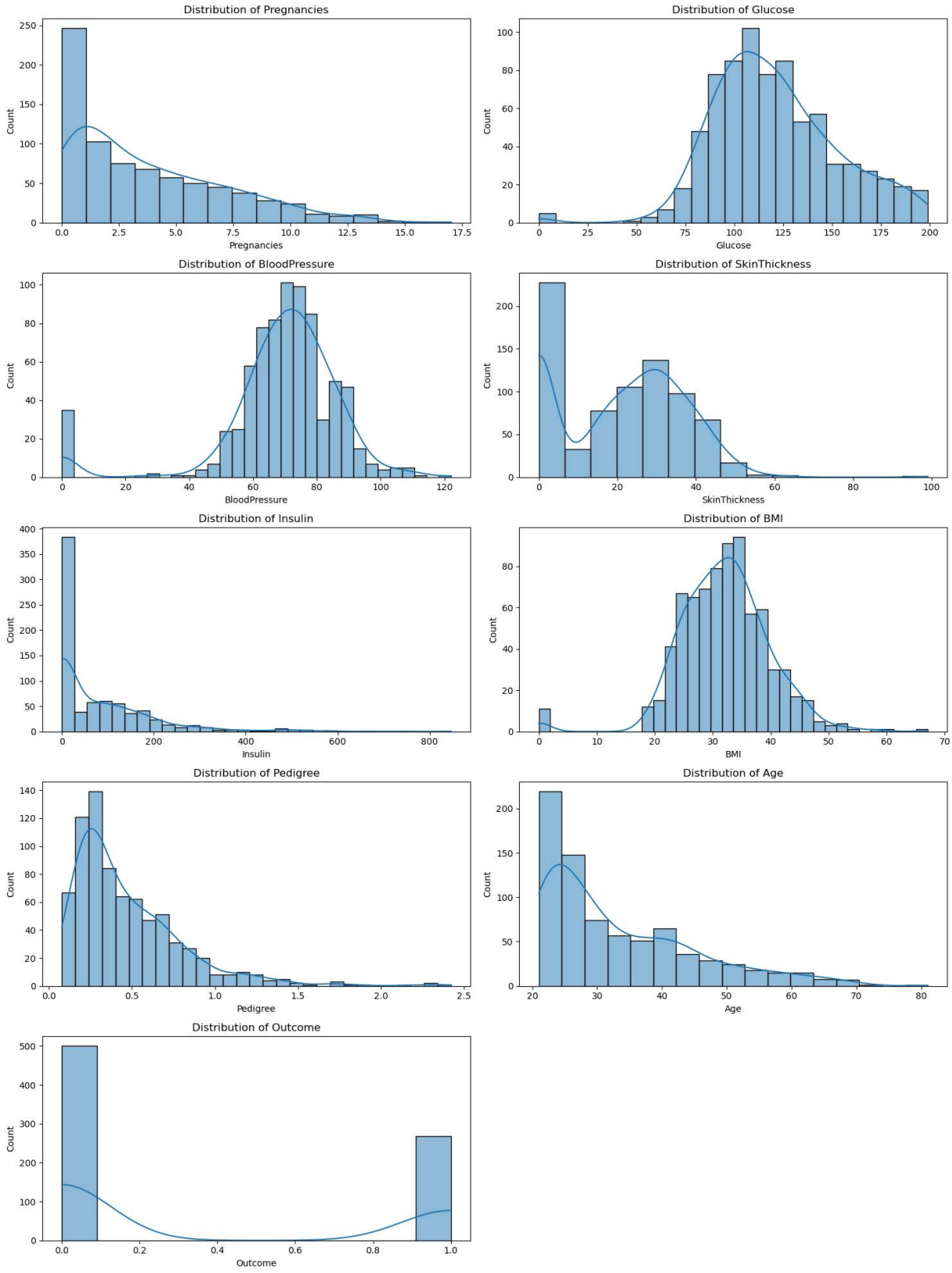
Task 1: Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и масштабирование.

In [6]:
```python
print_bold("Load the dataset")
df = pd.read_csv('diabetes.csv')

print_bold("Display the size of the dataset")
print(df.shape)

print_bold("Display basic statistics")
stats = df.describe()
print(stats)
```

```
--------------------------------------------------------------------------------
---------------
Load the dataset
--------------------------------------------------------------------------------
---------------
Display the size of the dataset
(768, 9)
--------------------------------------------------------------------------------
---------------
Display basic statistics
       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
```

In [7]:

```python
# Define the number of columns for visualization
num_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Set the number of columns in each row of plots
num_plots_per_row = 2

# Calculate the number of rows needed for subplots
num_rows = (len(num_cols) + 1) // num_plots_per_row

plt.figure(figsize=(15, 4 * num_rows))

# Loop through all numerical columns
for i, col in enumerate(num_cols, 1):
    # Distribution plot
    plt.subplot(num_rows, num_plots_per_row, i)
    sns.histplot(df[col], kde=True)
    plt.title('Distribution of ' + col)

plt.tight_layout()
plt.show()
```



In [8]:
```
# Apply the function to each numerical column
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
df[numerical_cols] = df[numerical_cols].apply(fill_missing_with_mean)

# Print the first few rows to check the result
print(df.head())
```

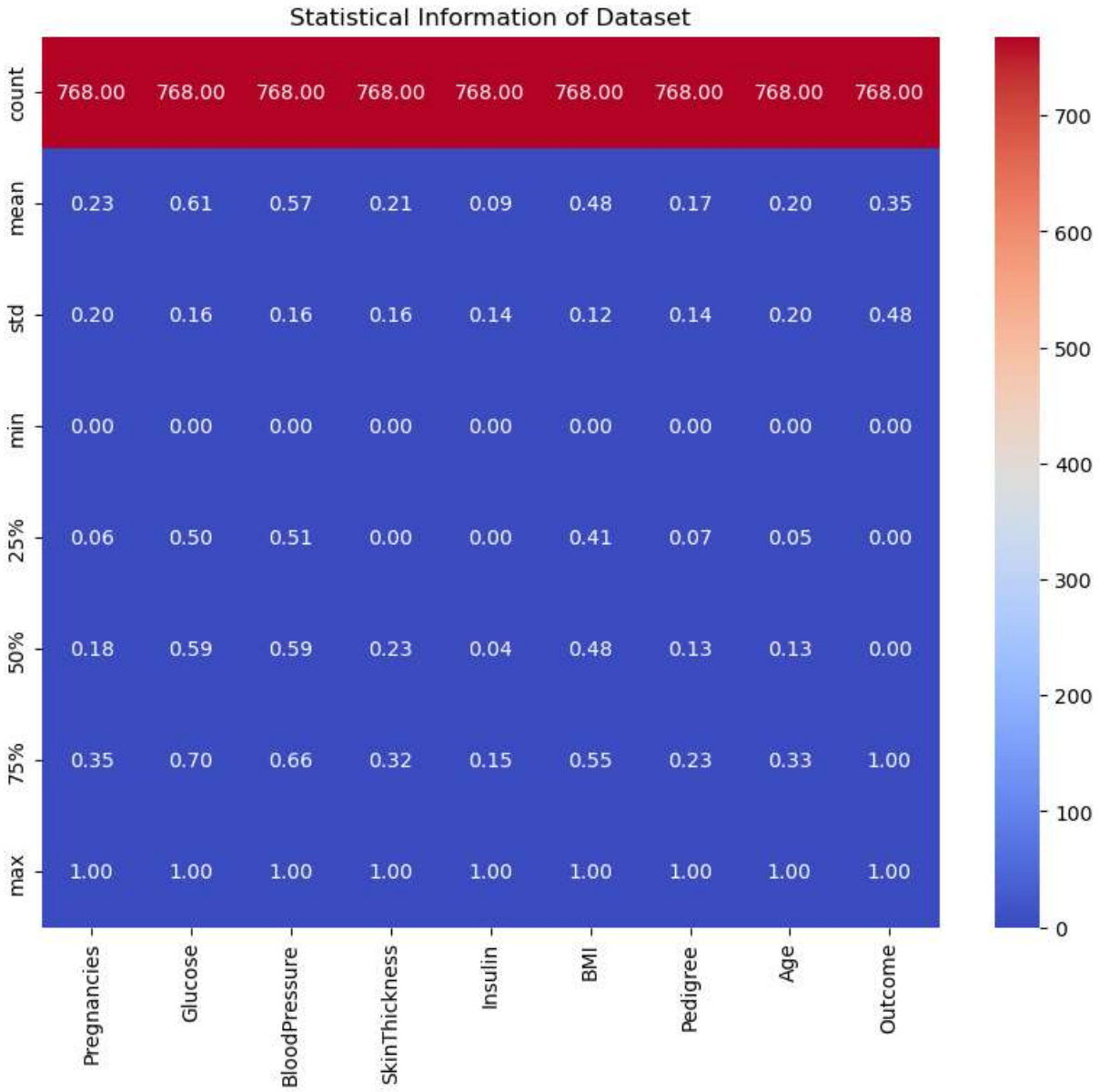
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1   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

   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
```

In [9]:
```python
# Exclude the 'Outcome' column and apply Min-Max scaling to the rest of the columns
features = df.columns.drop('Outcome')
df[features] = df[features].apply(min_max_scaler)

# Print the first few rows of the scaled dataframe to check the result
print(df.head())
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000  0.500745
1     0.058824  0.427136       0.540984       0.292929  0.000000  0.396423
2     0.470588  0.919598       0.524590       0.000000  0.000000  0.347243
3     0.058824  0.447236       0.540984       0.232323  0.111111  0.418778
4     0.000000  0.688442       0.327869       0.353535  0.198582  0.642325

   Pedigree       Age  Outcome
0  0.234415  0.483333        1
1  0.116567  0.166667        0
2  0.253629  0.183333        1
3  0.038002  0.000000        0
4  0.943638  0.200000        1
```

Task 2: Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили), постройте 3d-визуализацию признаков.

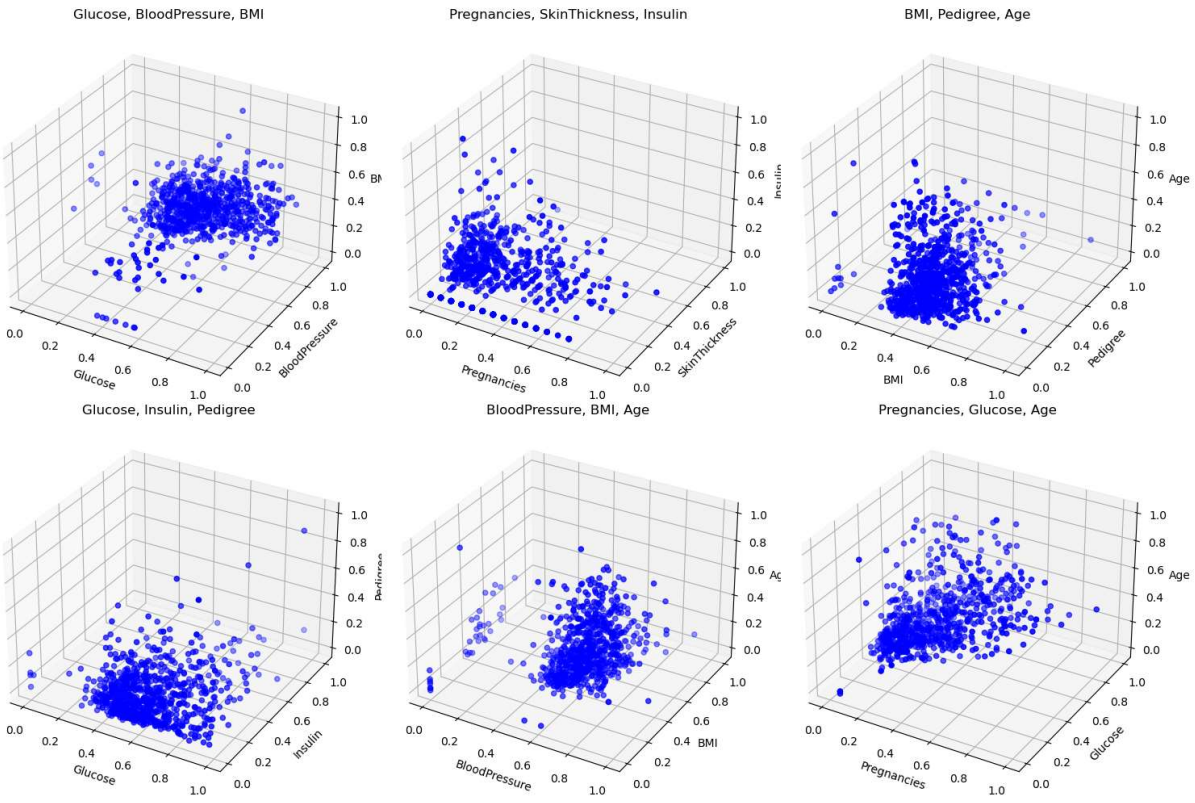
In [10]:
```python
# Получение статистической информации
stat_info = df.describe()

# Визуализация статистической информации
plt.figure(figsize=(10, 8))
sns.heatmap(stat_info, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Statistical Information of Dataset")
plt.show()
```



In [11]:

```python
from mpl_toolkits.mplot3d import Axes3D

# Define combinations of three features for the plots
feature_combinations = [
    ['Glucose', 'BloodPressure', 'BMI'],
    ['Pregnancies', 'SkinThickness', 'Insulin'],
    ['BMI', 'Pedigree', 'Age'],
    ['Glucose', 'Insulin', 'Pedigree'],
    ['BloodPressure', 'BMI', 'Age'],
    ['Pregnancies', 'Glucose', 'Age']
]

# Number of rows and columns for the subplot grid
n_rows = 2
n_cols = 3

# Create a figure for the subplots
fig = plt.figure(figsize=(15, 10))
```

```
# Create 3D plots for each combination in the grid
for i, features in enumerate(feature_combinations):
    ax = fig.add_subplot(n_rows, n_cols, i + 1, projection='3d')
    ax.scatter(df[features[0]], df[features[1]], df[features[2]], c='blue', marker=
    ax.set_xlabel(features[0])
    ax.set_ylabel(features[1])
    ax.set_zlabel(features[2])
    ax.set_title(f"{features[0]}, {features[1]}, {features[2]}")

# Adjust layout
plt.tight_layout()
plt.show()
```



Task 3: Реализуйте метод k-ближайших соседей без использования сторонних библиотек, кроме NumPy и Pandas.

In [12]:
```
def euclidean_distance(row1, row2):
    # Calculating the Euclidean distance between two rows
    return np.sqrt(np.sum((row1 - row2) ** 2))

def k_nearest_neighbors(train, test_row, k):
    # Finding the k nearest neighbors
    distances = []
    for train_row in train:
        dist = euclidean_distance(test_row[:-1], train_row[:-1])  # Ignore the labe
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])

    neighbors = []
    for i in range(k):
        neighbors.append(distances[i][0])
```

```
    # Aggregating the labels of neighbors
    labels = [neighbor[-1] for neighbor in neighbors]

    # Majority vote for the most common label
    prediction = max(set(labels), key=labels.count)
    return prediction
```

Task 4:

- Постройте две модели k-NN с различными наборами признаков:
  - Модель 1: Признаки случайно отбираются .
  - Модель 2: Фиксированный набор признаков, который выбирается заранее.

In [13]:
```
# Splitting the data into training and testing sets
train_df = df.sample(frac=0.8, random_state=1)  # 80% of data for training
test_df = df.drop(train_df.index)  # Remaining 20% for testing

# Model 1: Random Feature Selection
# Randomly select 3 features (excluding the last one, which is assumed to be the la
random_features = np.random.choice(df.columns[:-1], size=3, replace=False)
print_bold("Random features:")
print(random_features)
# Create training and testing sets with these features and the 'Outcome' label
train_random = train_df[random_features].join(train_df['Outcome'])
test_random = test_df[random_features].join(test_df['Outcome'])

# Model 2: Fixed Feature Selection
# Selecting specific features based on domain knowledge or prior analysis
# Glucose: Typically, higher glucose levels are a significant indicator of diabetes

# BMI (Body Mass Index): BMI is a key factor because it is often correlated with th

# Age: Age is another important factor. The risk of developing type 2 diabetes incr
fixed_features = ['Glucose', 'BMI', 'Age']
print_bold("Fixed features:")
print(fixed_features)
# Create training and testing sets with these fixed features and the 'Outcome' labe
train_fixed = train_df[fixed_features].join(train_df['Outcome'])
test_fixed = test_df[fixed_features].join(test_df['Outcome'])

# Function to test a model's accuracy
def test_model(train_set, test_set, k):
    correct = 0
    for test_row in test_set.values:
        # Predict the class of each test row
        prediction = k_nearest_neighbors(train_set.values, test_row, k)
        # Check if the prediction matches the actual label
        if prediction == test_row[-1]:  # Assuming the class label is the last colu
            correct += 1
    # Calculate accuracy as the proportion of correct predictions
    accuracy = correct / len(test_set)
    return accuracy

# Testing both models with k=3 neighbors
```

```
k = 3
accuracy_random = test_model(train_random, test_random, k)
accuracy_fixed = test_model(train_fixed, test_fixed, k)

# Print the accuracy of both models
print(f"Accuracy of Model 1 (Random Features): {accuracy_random}")
print(f"Accuracy of Model 2 (Fixed Features): {accuracy_fixed}")
```

```
--------------------------------------------------------------------------------
----------------
Random features:
['Age' 'Insulin' 'Glucose']
--------------------------------------------------------------------------------
----------------
Fixed features:
['Glucose', 'BMI', 'Age']
Accuracy of Model 1 (Random Features): 0.6688311688311688
Accuracy of Model 2 (Fixed Features): 0.6948051948051948
```

Task 5: Для каждой модели проведите оценку на тестовом наборе данных при разных значениях k. Выберите несколько различных значений k, например, k=3, k=5, k=10, и т. д. Постройте матрицу ошибок.

In [14]:
```
def confusion_matrix(true, pred):
    # Calculate confusion matrix components
    true = np.array(true)
    pred = np.array(pred)
    TP = sum((true == 1) & (pred == 1))  # True Positives
    TN = sum((true == 0) & (pred == 0))  # True Negatives
    FP = sum((true == 0) & (pred == 1))  # False Positives
    FN = sum((true == 1) & (pred == 0))  # False Negatives

    # Create a DataFrame for the confusion matrix
    conf_matrix_df = pd.DataFrame(
        [[TP, FP],
         [FN, TN]],
        columns=['Predicted Positive', 'Predicted Negative'],
        index=['Actual Positive', 'Actual Negative']
    )
    return conf_matrix_df

def evaluate_model(train_set, test_set, k_values):
    # Evaluate the model for different values of k and return confusion matrices
    true_labels = test_set[:, -1]
    confusion_matrices = []

    for k in k_values:
        # Make predictions for each test row
        predictions = [k_nearest_neighbors(train_set, test_row, k) for test_row in
        # Calculate and store the confusion matrix for this k value
        confusion_matrices.append(confusion_matrix(true_labels, predictions))

    return confusion_matrices

# Define different values of k
```

```python
k_values = [3, 5, 10, 20]

# Evaluate Model 1 (Random Features)
confusion_matrices_random = evaluate_model(train_random.values, test_random.values,

# Evaluate Model 2 (Fixed Features)
confusion_matrices_fixed = evaluate_model(train_fixed.values, test_fixed.values, k_

# Print confusion matrices for each model and k value
for i, k in enumerate(k_values):
    print_bold("k = " + str(k))
    print(f"Model 1 (Random Features) accuracy {test_model(train_random, test_rando
    print('\n')
    print(f"Model 2 (Fixed Features) accuracy {test_model(train_fixed, test_fixed,
```

---
k = 3
Model 1 (Random Features) accuracy 0.6688311688311688

	Predicted Positive	Predicted Negative
Actual Positive	28	22
Actual Negative	29	75

Model 2 (Fixed Features) accuracy 0.6948051948051948

	Predicted Positive	Predicted Negative
Actual Positive	30	20
Actual Negative	27	77

---
k = 5
Model 1 (Random Features) accuracy 0.7142857142857143

	Predicted Positive	Predicted Negative
Actual Positive	35	22
Actual Negative	22	75

Model 2 (Fixed Features) accuracy 0.7077922077922078

	Predicted Positive	Predicted Negative
Actual Positive	31	19
Actual Negative	26	78

---
k = 10
Model 1 (Random Features) accuracy 0.6818181818181818

	Predicted Positive	Predicted Negative
Actual Positive	26	18
Actual Negative	31	79

Model 2 (Fixed Features) accuracy 0.7337662337662337

	Predicted Positive	Predicted Negative
Actual Positive	27	11
Actual Negative	30	86

---
k = 20
Model 1 (Random Features) accuracy 0.6948051948051948

	Predicted Positive	Predicted Negative
Actual Positive	28	18
Actual Negative	29	79

Model 2 (Fixed Features) accuracy 0.7402597402597403

	Predicted Positive	Predicted Negative
Actual Positive	32	15
Actual Negative	25	82