НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

КУРСОВА РОБОТА

з дисципліни «Аналіз даних в інформаційних системах»

на тему: «Прогнозування наявності серцево-судинного захворювання у людини на основі медичних показників. Методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM»

Студента 2 курсу групи ІП-11
Спеціальності: 121
«Інженерія програмного забезпечення»
Панченка Сергія Віталійовича

«ПРИЙНЯВ» з оцінкою

_____

доц. Ліхоузова Т.А. / доц. Олійник Ю.О.

_____

Підпис Дата

Київ - 2023 рік

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна Аналіз даних в інформаційно-управляючих системах

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 2 Група ІП-11 Семестр 4

ЗАВДАННЯ

на курсову роботу студента

Панченка Сергія Віталійовича

1.Тема роботи Прогнозування наявності серцево-судинного захворювання у людини на основі медичних показників. Методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM.

2.Строк здачі студентом закінченої роботи 15.06.2022

3. Вхідні дані до роботи методичні вказівки до курсової роботи, обрані дані з сайту https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

4.Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Вступ, постановка задачі, аналіз предметної області, робота з даними, інтелектуальний аналіз, висновки, перелік посилань, додаток А.

5.Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6.Дата видачі завдання 16.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	16.03.2023	
2.	Визначення зовнішніх джерел даних	16.03.2023	
3.	Пошук та вивчення літератури з питань курсової роботи	16.03.2023	
4.	Обробка та аналіз даних	16.03.2023	
5.	Обґрунтування методів інтелектуального аналізу даних	16.03.2023	
6.	Застосування та порівняння ефективності методів інтелектуального аналізу даних	16.03.2023	
7.	Підготовка пояснювальної записки	16.03.2023	
8.	Здача курсової роботи на перевірку	16.03.2023	
9.	Захист курсової роботи	16.03.2023	

Студент _____ Панченко С. В.

(підпис) (прізвище, ім’я, по батькові)

Керівник _____ доц. Ліхоузова Т.А

(підпис) (прізвище, ім’я, по батькові)

Керівник _____ доц. Олійник Ю.О.

(підпис) (прізвище, ім’я, по батькові)

"26" червня 2023 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 38 сторінок, 32 рисунки, 11 посилань.

Об’єкт дослідження: інтелектуальний аналіз даних.

Предмет дослідження: створення програмного забезпечення, що проводить аналіз даних з подальшим прогнозуванням та графічним відображенням результатів.

Мета роботи: пошук, аналіз та обробка даних, реалізація ПЗ, що використовує отримані дані для подальшого аналізу та прогнозування результату.

Дана курсова робота включає в себе: постановку задачі, аналіз предметної області, роботу з даними, аналіз обраних методів для прогнозування та їх порівняння.

ДАТАСЕТ, ПРОГНОЗУВАННЯ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, K-NEAREST NEIGHBORS, DECISION TREE CLASSIFIER.

ЗМІСТ

ВСТУП

Зараз все більше стає запитаною медична діагностика різних захворювань, що є дуже важливим аспектом для лікарів та здоров’я пацієнтів. Аналізуючи захворювання, особливо хочеться виділити цукровий діабет, адже з розвитком рівня життя цукровий діабет все частіше зустрічається в повсякденні. Саме через це швидкість та точність діагностування цукрового діабету – завдання, що потребує швидких та точних результатів.

У рамках даної курсової роботи були проаналізовані дані лікарень з різними важливими характеристиками для визначення цукрового діабету та на основі отриманих даних було використано декілька методів для прогнозування даного захворювання.

Дана курсова робота буде розроблена з використанням технологій та бібліотек Python 3[1], Pandas[2], Seaborn[3], Matplotlib[4], Sklearn[5], NumPy[6].

1 ПОСТАНОВКА ЗАДАЧІ

Виконання даної курсової роботи потребує виконання декількох задач: аналізу предметної області; роботи з датасетом: завантаження, дослідження його структури та виправлення наявних помилок; вибір методів для прогнозування та обґрунтування даного вибору; аналіз отриманих результатів кожного з методів та порівняння отриманих результатів ефективності.

Створення застосунку, що поділяє отримані дані на тренувальні та тестові для перевірки декількох методів, у подальшому порівняння ефективності методів. Для прогнозування буде використано методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM. Для кожного методу проаналізувати результати та в кінці порівняти результати цих методів. Обрати найоптимальніший метод для прогнозування хвороби.

Виконане дослідження можна буде використовувати для прогнозування можливої серцево-судинної хвороби.

Вхідними даними будуть вік, зріст, вага, стать, систолічний кров’яний тиск, діастолічний артеріальний тиск, кількість холестеролу в крові, кількість глюкози в крові, чи курить пацієнт, чи вживає алкоголь, чи займається фізичною активністю.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Серцево-судинні захворювання (ССЗ) є одними з найбільш поширених причин смерті в світі. ССЗ можуть включати такі захворювання, як ішемічна хвороба серця (ІХС), цереброваскулярні захворювання (ЦВЗ), артеріальна гіпертензія (АГ), хронічна обструктивна хвороба легень (ХОЛЛ), діабет та інші.

Статистика ССЗ може відрізнятися залежно від країни та регіону, а також від факторів ризику, таких як вік, стать, стиль життя і генетика. Однак загалом ССЗ становлять значну частку смертності в світі.

За даними Всесвітньої організації охорони здоров'я (ВООЗ), в 2019 році ССЗ стали причиною смерті для більше ніж 17 мільйонів людей, що становить більше 30% від загальної кількості смертей в світі. Від ІХС та ЦВЗ померли понад 13 мільйонів людей, що становить більше 80% від усіх смертей від ССЗ. Більшість смертей від ССЗ відбуваються в країнах з низьким і середнім рівнем доходів.

Науковці вважають, що багато випадків ССЗ можна запобігти шляхом зменшення факторів ризику, таких як неправильне харчування, недостатній рівень фізичної активності, куріння та надмірне вживання алкоголю. Зменшення факторів ризику та поліпшення доступу до належної медичної допомоги може допомогти знизити рівень ССЗ та покращити здоров'я людей.

У програмному забезпеченні буде реалізовано наступну функціональність, що включає в себе:

- завантаження та дослідження структури датасету;
- використання декількох моделей прогнозування даних;
- прогнозування за характеристиками наявність серцево-судинного захворювання у людини;
- відображення отриманих результатів та їх аналіз;
- порівняння використаних методів.

3 РОБОТА З ДАНИМИ

3.1 Опис обраних даних

Імпортуємо пакет pandas, завантажимо дані в датафрейм, виводимо інформацію про нього. Для вирішення поставленої задачі було обрано "Cardiovascular Disease" датасет. Він складається з 70 000 пацієнтів. Даний датасет містить в собі такі колонки, як-от: "age" - вік, "height" - зріст, "weight" - вага, "gender" - стать, "ap_hi" - систолічний кров'яний тиск, "ap_lo" - діастолічний артеріальний тиск, "cholesterol" - холестерол, "gluc" - глюкоза, "smoke" - чи курить пацієнт, "alco" - чи вживає алкоголь, "active" - фізична активність, "cardio" - присутність захворювання. Додатково імпортуємо модулі matplotlib.pyplot та seaborn для відображення графіків.

In [46]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/cardio_train.csv', sep=';')
df
```

Out[46]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	17474	1	156	56.0	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
69995	19240	2	168	76.0	120	80	1	1	1	0	1	0
69996	22601	1	158	126.0	140	90	2	2	0	0	1	1
69997	19066	2	183	105.0	180	90	3	1	0	1	0	1
69998	22431	1	163	72.0	135	80	1	2	0	0	0	1
69999	20540	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 12 columns

Рисунок 3.1 - Завантажений датафрейм

Видалимо колонку id за допомогою методу pandas.Dataframe.drop, передавши в нього список з єдиним елементом id та параметром за замовчуванням inplace=True, щоб зміна відбулася в самому датафреймі, а не видало новий.

```
In [47]: df.isna().any()

Out[47]: age            False
         gender         False
         height         False
         weight         False
         ap_hi          False
         ap_lo          False
         cholesterol    False
         gluc           False
         smoke          False
         alco           False
         active         False
         cardio         False
         dtype: bool
```

Рисунок 3.3 - Колонки, де всі значення ініціалізовані

3.2 Вибір ознак для аналізу

Побудуємо матрицю кореляції та визначимо, наскільки кожен фактор впливає на якість води. Для цього імпортуємо модуль seaborn та застосуємо функцію heatmap, де передаємо йому вище зазначену матрицю, використавши pandas.Dataframe.corr метод датафрейму.

```
In [13]: import seaborn as sns
         fig, axis = plt.subplots(figsize=(10, 6))
         axis.set_title('Кореляція між факторами')
         sns.heatmap(df.corr(), ax=axis, annot=True)

Out[13]: <AxesSubplot: title={'center': 'Кореляція між факторами'}>
```

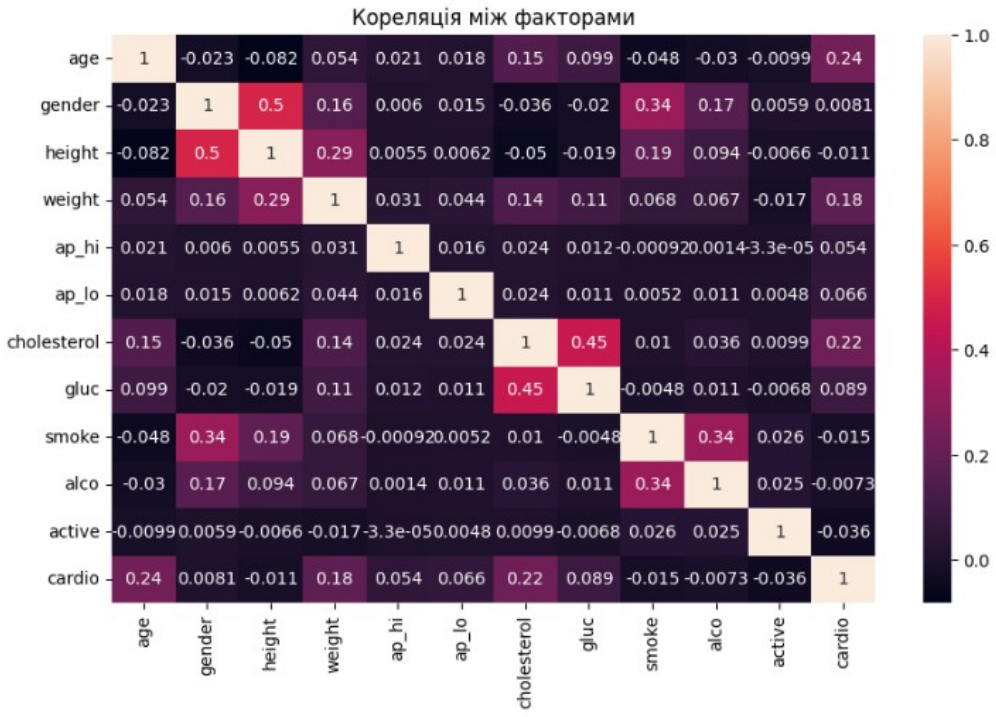

Рисунок 3.4 - Матриця кореляцій

Бачимо, що найбільше на виникнення серцево-судинних захворювань впливає вік, вага, холестерол. Доволі цікавим фактами є кореляція між віком та холестеролом, статтю та курінням, статтю та алкоголем, холестеролом та глюкозою. Залежність між статтю та вагою і висотою доволі очевидна, що пояснюється звичайною різницею у фізичних показниках між чоловіком та жінокю. Зобразимо статистику факторів та виникненням захворювання, розділену за статтю. За допомогою функції seaborn.FacetGrid згрупуємо значення та за допомогою seaborn.histplot зобразимо їх у вигляді гістограми. Побудуємо статистику за віком.

```
In [48]: def plot_stat(factor: str):
             g = sns.FacetGrid(df[['gender', 'cardio'] + [factor]], col='gender', hue='cardio')
             g.map(sns.histplot, factor)
             g.add_legend()
         plot_stat('age')
```

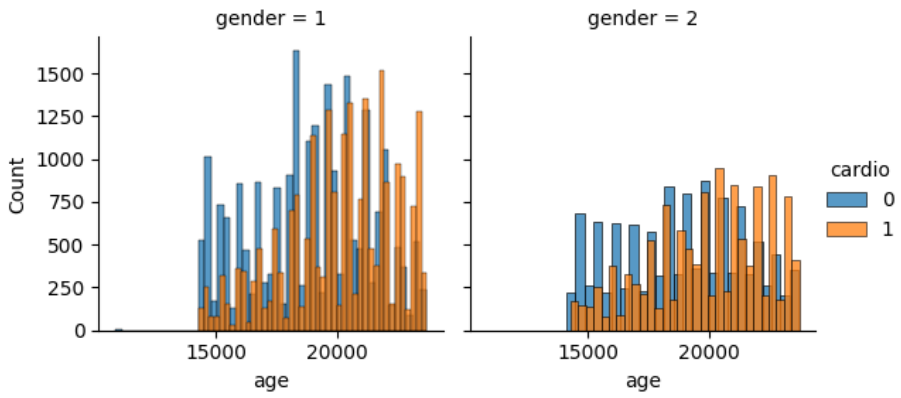

Рисунок 3.5 - Статистика захворювання від віку між чоловіками та жінками

Зобразимо статистику за вагою.

```
In [15]: plot_stat('weight')
```

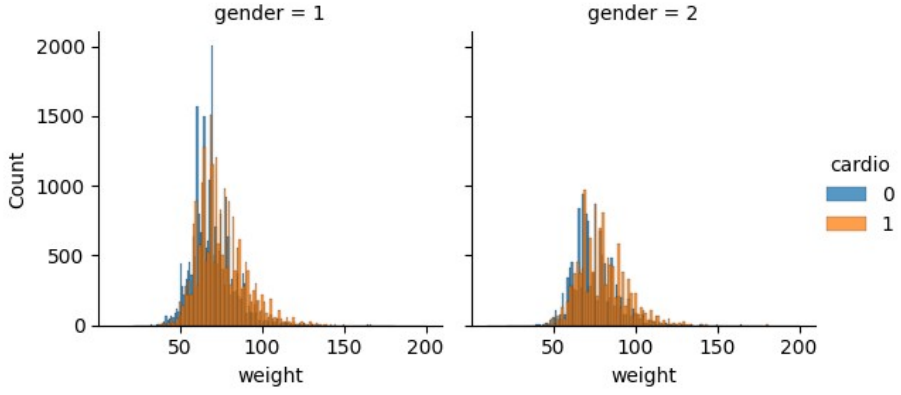

Рисунок 3.6 - Статистика захворювання від ваги між чоловіками та жінками

Зобразимо статистику за холестеролом.

```
In [16]: plot_stat('cholesterol')
```

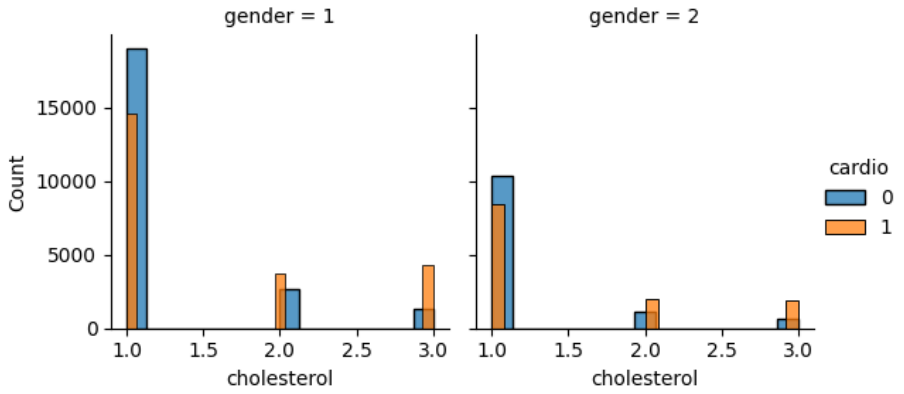

Рисунок 3.7 - Статистика захворювання від холестеролу між чоловіками та жінками

3.3 Поділ даних

Для збільшення якості моделі залишимо лише чотири колонки: age, weight, cholesterol, cardio.

```
In [49]: df = df[['age', 'weight', 'cholesterol', 'cardio']]
         df
```

Out[49]:

	age	weight	cholesterol	cardio
0	18393	62.0	1	0
1	20228	85.0	3	1
2	18857	64.0	3	1
3	17623	82.0	1	1
4	17474	56.0	1	0
...	...	...	...	...
69995	19240	76.0	1	0
69996	22601	126.0	2	1
69997	19066	105.0	3	1
69998	22431	72.0	1	1
69999	20540	72.0	2	0

70000 rows × 4 columns

Рисунок 3.8 - Відфільтрований датафрейм

Ділимо дані на тренувальні та тестові для подальшох роботи. Імпортуємо

модуль sklearn.model_selection та застосуємо функцію train_test_split. Розділимо набір даних на 80% навчальних та 20% тестових.

```
In [50]: from sklearn.model_selection import train_test_split
         x = df.iloc[:, :-1]
         y = df.iloc[:, -1]
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, train_size=0.8, r
```

Рисунок 3.9 - Поділ інформації на інформацію та результат

Зберігатимемо результати тестування моделей у списку results

```
In [ ]: results = []
```

Рисунок 3.10 - Список результатів

4 ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ

4.1 Обґрунтування вибору методів інтелектуального аналізу даних

Було обрано чотири методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM.

K-Nearest Neighbors (KNN) — це простий, але ефективний алгоритм машинного навчання, який використовується для завдань класифікації та регресії. Я обрав, щоб порівняти наскільки він швидший за інші алгоритми, оскільки він не робить жодних припущень щодо розподілу даних.

До того ж однією з сильних сторін алгоритму KNN є те, що він легко інтерпретується. Прогнози можна пояснити, просто подивившись на K найближчих сусідів даної точки даних. Крім того, KNN може обробляти нелінійні межі рішень і може використовуватися як для бінарних, так і для багатокласових задач класифікації. Тому він чудово підійде для прогнозування хвороб, де треба враховувати багато факторів.

Однак, якщо датасети великі, то KNN може бути дорогим з обчислювальної точки зору. Крім того, продуктивність KNN значною мірою залежить від значення K, яке потрібно ретельно вибирати для досягнення оптимальних результатів.

Logistic Regression – це статистичний метод, який використовується для прогнозування ймовірності події. Оскільки її залежна змінна є двійковою, то це дуже підходить для виявлення хвороби: вона або є, або її нема.

Logistic Regression має кілька переваг перед іншими алгоритмами класифікації. По-перше, її легко реалізувати та інтерпретувати. По-друге, вона може обробляти як категоричні, так і безперервні незалежні змінні. Тобто можна обробляти як дискретні дані, які лекго визначити: вага, зріст - так і виявлення в організмі сполук, які складно визначити точно, і де робляться припущення про інтервали: рівень глюкози, рівень холестиролу, вітамінів в організмі тощо. По-третє, він може надати ймовірність події, що станеться, що корисно під час медичної діагностики.

Random Forest — це популярний алгоритм машинного навчання, який використовується для завдань класифікації та регресії. Це метод ансамблю, який поєднує передбачення кількох дерев рішень для отримання більш точних

результатів.

Я його обрав, бо є стійким до шуму. До того ж цікаво, як він буде передбачувати наявність хвороби в пацієнта. Варто зауважити, що Random Forest може бути дорогим з точки зору обчислень, особливо при роботі з великими наборами даних і багатьма деревами. Також може бути важко інтерпретувати результати випадкового лісу, оскільки процес прийняття рішення розподіляється між кількома деревами.

SVM - це популярний і потужний клас алгоритмів керованого навчання, які використовуються для завдань класифікації та регресії. SVM особливо добре підходять для проблем, де існують складні межі між класами. У медичній сфері це часто використовується для прогнузування пацієту декількох хвороб, де окреме захворювання - це клас, де варто мати чіткі межі для поставлення правильного діагнозу.

Основна ідея SVM — знайти гіперплощину, яка найкраще розділяє дані на різні класи. Гіперплощина — це лінійна межа рішення, яка розділяє два класи в просторі ознак.

Однією з сильних сторін SVM є те, що вони нечутливі до викидів, оскільки на гіперплощину впливають лише найближчі до неї точки. Це робить SVM дуже стійкими до зашумлених даних, де зазвичай пацієнтів дуже багато, а тому ця особливість є важливою.

Однак SVM мають деякі обмеження. Навчання їх може бути дорогим з обчислювальної точки зору, особливо з великими наборами даних або просторами функцій великої розмірності. Крім того, вибір правильної функції ядра може бути складним, а продуктивність SVM залежить від вибору гіперпараметрів.

4.2 Аналіз отриманих результатів для методу K-Nearest Neighbors

Для виконання роботи методу KNN імпортуємо sklearn.neighbors.KNeighborsClassifier та sklearn.model_selection.GridSearchCV.

```
In [19]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import GridSearchCV
```

Рисунок 4.1 - Імпортування модулів

Визначимо, які варіанти параметрів найкраще вирішують дану задачу.

```
In [20]: classificator = KNeighborsClassifier()
         params = {'n_neighbors': range(1, 60)}
         grid_search = GridSearchCV(classificator, params, cv=10, verbose=1)
         grid_search.fit(x_train, y_train)
         knn = grid_search.best_estimator_
         knn

         Fitting 10 folds for each of 59 candidates, totalling 590 fits
Out[20]: ▼ KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=58)
```

Рисунок 4.2 - Визначення найкращого параметра

Натренуємо модель з найкращим параметром.

```
In [21]: knn.fit(x_train, y_train)
Out[21]: ▼ KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=58)
```

Рисунок 4.3 - Тренування моделі K-Nearest Neighbors

Визначимо точність моделі на тренувальних та тестових даних

```
In [51]: train_score = round(knn.score(x_train, y_train), 5)
         test_score = round(knn.score(x_test, y_test), 5)
         results.append({'method': 'knn', 'score': train_score, 'type': 'train'})
         results.append({'method': 'knn', 'score': test_score, 'type': 'test'})
         print(f'Train accuracy: {train_score}')
         print(f'Test accuracy: {test_score}')

         Train accuracy: 0.63252
         Test accuracy: 0.60807
```

Рисунок 4.4 - Точність моделі K-Nearest Neighbors

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей. Для цього застосуємо sklearn.metrics.plot_confusion_matrix.

```
In [23]: from sklearn.metrics import confusion_matrix
         def conf_mat(model, x_test, y_test):
             y_predicted = model.predict(x_test)
             cm = confusion_matrix(y_test, y_predicted)
             plt.figure(figsize = (8,5))
             sns.heatmap(cm, annot=True, fmt=".1f")
             plt.xlabel('Predicted')
```

Рисунок 4.5 - Імпортування модуля

Матриця має два рядки та дві колонки: перший ряд і перша колонка - це істинно позитивні значення, тобто людина здорова і модель не визначила захворювання; першмй ряд і друга колонка - хибно позитивні, тобто людина здорова, а модель сказала, що є захворювання; другий ряд і перша колонка - хибно негативні, тобто людина хвора, а модель сказала, що здорова; другий ряд і друга колонка - істинно негативні, тобто людина хвора і модель сказала, що хвора.

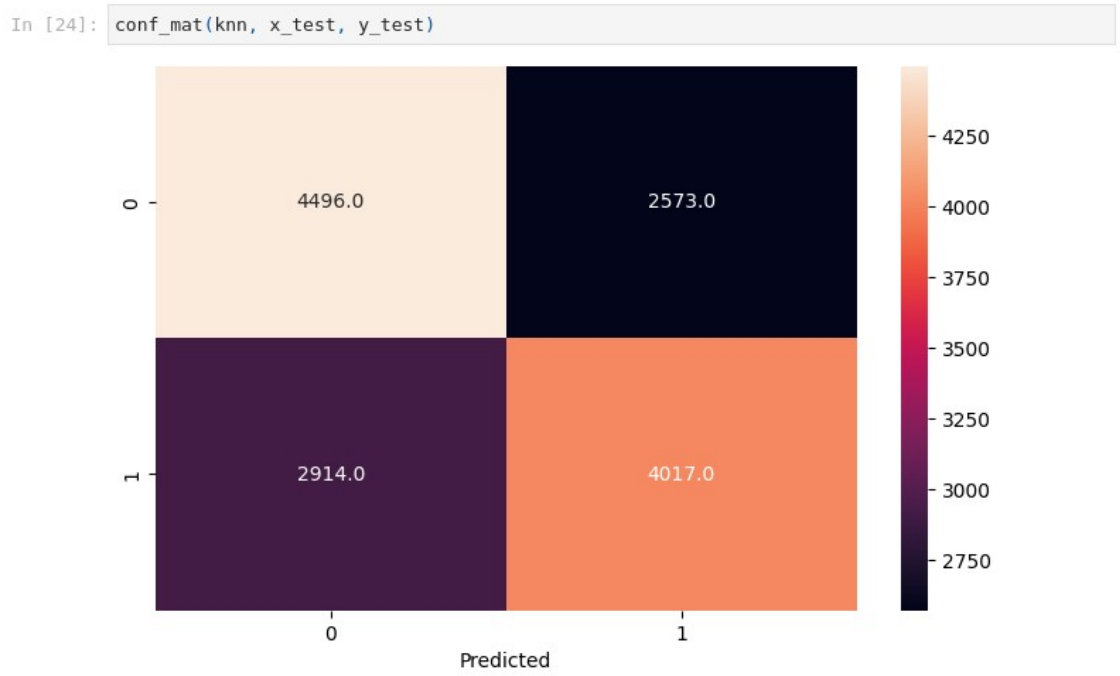

Рисунок 4.6 - Матриця невідповідностей для K-Nearest Neighbors

Побудуємо графік ROC(Receiver Operating Characteristic), що є графіком істинно позитивної відносної частоти проти хибно позитивної частоти. Це показує компроміс між чутливістю та специфічністю. Для цього імпортуємо sklearn.metrics.roc_curve та sklearn.metrics.roc_auc_score. До того ж визначимо AUC(Area Under the ROC Curve), що є мірою того, наскільки добре модель може розрізняти позитивні та негативні рузультати. Він коливається від 0 до 1, де 1 є найкращим класифікатором, а 0,5 – випадковим класифікатором. AUC корисний під час порівняння продуктивності різних класифікаторів на одному наборі даних, бо даж єдине число, яке підсумовує загальну продуктивність.

```
In [25]: from sklearn.metrics import roc_curve, roc_auc_score
def roc(model, x_test, y_test):
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = roc_curve(y_test,  y_pred_proba)
    auc = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="Area = "+str(auc)+')')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc=4)
    plt.show()
```

Рисунок 4.7 - Імпортування модуля та визначення функції roc

Побудуємо ROC для K-Nearest Neighbors.

```
In [26]: roc(knn, x_test, y_test)
```

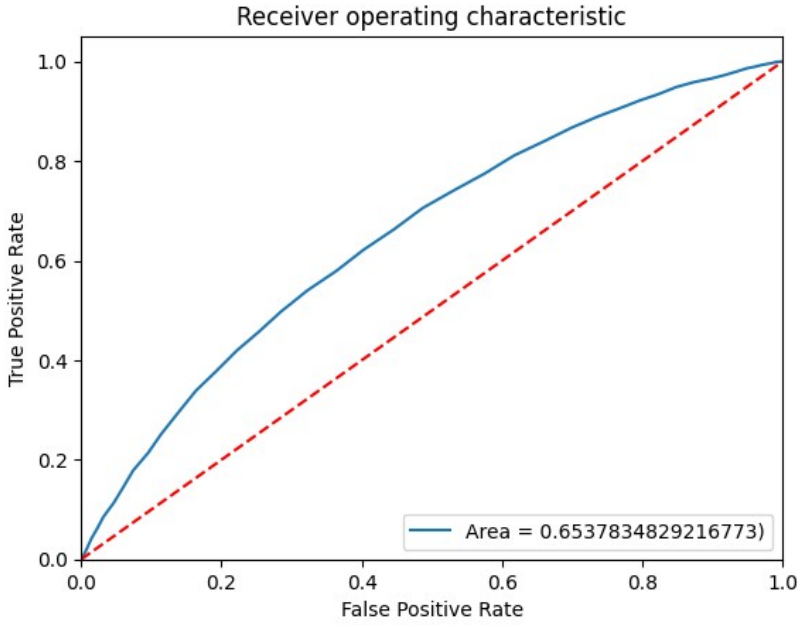

Рисунок 4.8 - Графік ROC для K-Nearest Neighbors

4.3 Аналіз отриманих результатів для методу Logistic Regression

Для виконання роботи методу KNN імпортуємо sklearn.linear_model.LogisticRegression; sklearn.pipeline.Pipeline для побудови пайплайну, щоб у зручному вигляді передавати параметри до декількох функцій; sklearn.preprocessing.StandardScaler для скейлингу даних до проміжку від 0 до 1. Визначимо найкращі параметри моделі, передавши в неї параметри регуляризації.

In [27]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
logisticRegr = LogisticRegression()
pipe = Pipeline(steps=[('sc', sc), ('logisticRegr', logisticRegr)])
c = np.logspace(-4, 4, 60)
penalty = ['l1', 'l2']
params = dict(logisticRegr__C=c, logisticRegr__penalty=penalty)
log_reg = GridSearchCV(pipe, params)
log_reg.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFai
ledWarning:
300 fits failed out of a total of 600.
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='rais
e'.

Below are more details about the failures:
--------------------------------------------------------------------------------
300 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", l
ine 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/pipeline.py", line 405, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1
162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 5
4, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarnin
g: One or more of the test scores are non-finite: [       nan 0.64080357        nan 0.6409
1071        nan 0.64094643
        nan 0.64083929        nan 0.641             nan 0.64110714
        nan 0.64114286        nan 0.64121429        nan 0.64133929
        nan 0.64139286        nan 0.64146429        nan 0.64142857
        nan 0.64139286        nan 0.64141071        nan 0.64139286
        nan 0.64135714        nan 0.64133929        nan 0.64133929
        nan 0.64133929        nan 0.64135714        nan 0.64135714
        nan 0.64133929        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714
        nan 0.64135714        nan 0.64135714        nan 0.64135714]
  warnings.warn(
```

Out[27]:

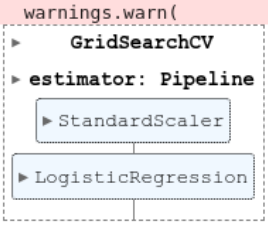

Рисунок 4.9 - Тренування моделі Logistic Regression

Визначимо точність моделі на тренувальних та тестових даних

```
In [52]: train_score = round(log_reg.score(x_train, y_train), 5)
         test_score = round(log_reg.score(x_test, y_test), 5)
         results.append({'method': 'logress', 'score': train_score, 'type': 'train'})
         results.append({'method': 'logress', 'score': test_score, 'type': 'test'})
         print(f'Train accuracy: {train_score}')
         print(f'Test accuracy: {test_score}')

         Train accuracy: 0.64145
         Test accuracy: 0.63729
```

Рисунок 4.10 - Точність моделі Logistic Regression

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [29]: conf_mat(log_reg, x_test, y_test)
```

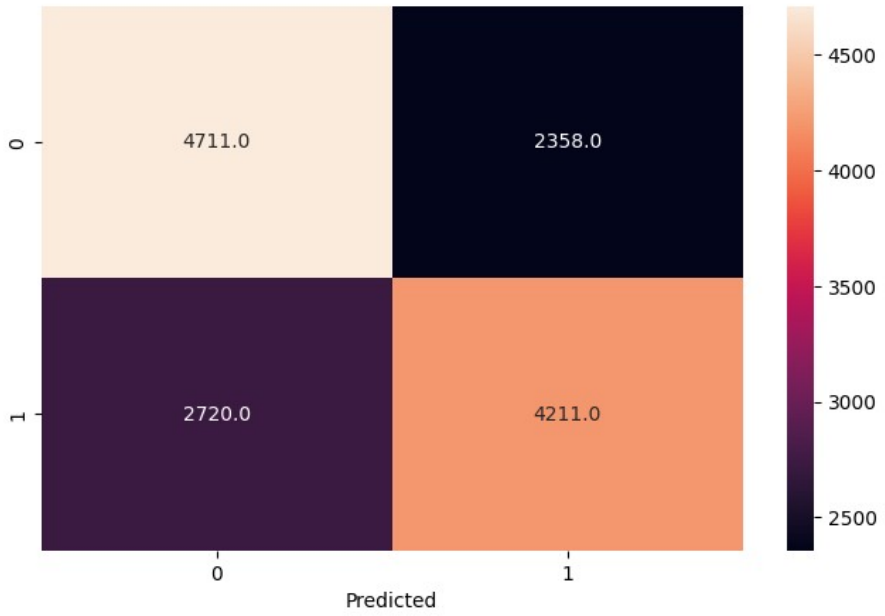

Рисунок 4.11 - Матриця невідповідностей для Logistic Regression

Побудуємо графік ROC для Logistic Regression.

```
In [30]: roc(log_reg, x_test, y_test)
```

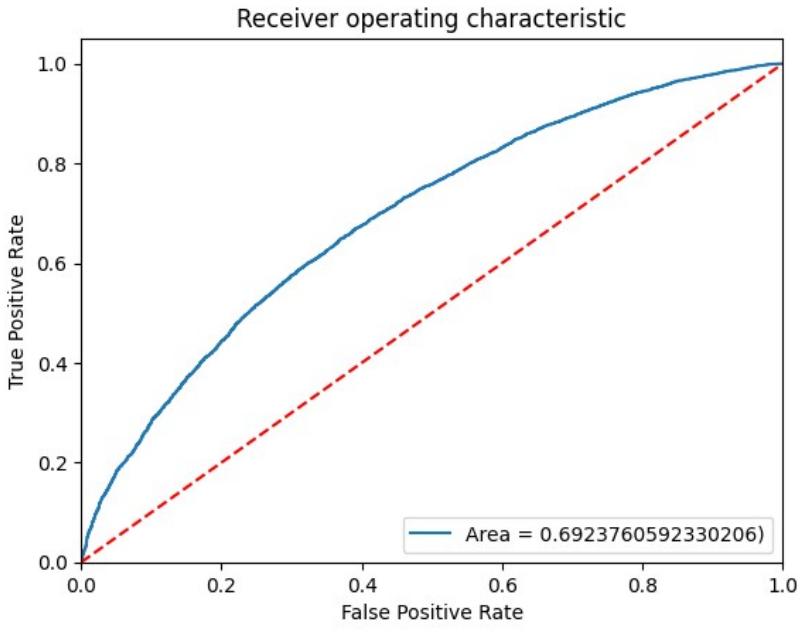

Рисунок 4.12 - Графік ROC для Logistic Regression

4.4 Аналіз отриманих результатів для методу Random Forest

Для виконання роботи методу KNN імпортуємо sklearn.ensemble.RandomForestClassifier. Визначимо найкращі параметри для моделі. У випадку Random Forest параметри включають кількість дерев рішень та кількість характеристик, які враховуються кожним деревом під час поділу вузла.використовуються для поділу кожного вузла, отриманого під час навчання. Імпортуємо sklearn.model_selection.RandomizedSearchCV.

```
In [31]: from sklearn.ensemble import RandomForestClassifier
         import numpy as np
         # Кількість дерев
         n_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, num = 60)]
         params = {'n_estimators': n_estimators}
         rf = RandomForestClassifier()
         rf_random = GridSearchCV(rf, param_grid=params, cv=3, n_jobs=5)
         rf_random.fit(x_train, y_train)

         /usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:700: Use
         rWarning: A worker stopped while some jobs were given to the executor. This can be caused
         by a too short worker timeout or by a memory leak.
           warnings.warn(

Out[31]: GridSearchCV
         estimator: RandomForestClassifier
         RandomForestClassifier
```

Рисунок 4.13 - Тренування моделі Random Forest

Визначимо точність моделі на тренувальних та тестових даних

```
In [53]: train_score = round(rf_random.score(x_train, y_train), 5)
         test_score = round(rf_random.score(x_test, y_test), 5)
         results.append({'method': 'rf', 'score': train_score, 'type': 'train'})
         results.append({'method': 'rf', 'score': test_score, 'type': 'test'})
         print(f'Train accuracy: {train_score}')
         print(f'Test accuracy: {test_score}')

         Train accuracy: 0.96564
         Test accuracy: 0.57243
```

Рисунок 4.14 - Точність моделі Random Forest

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [33]: conf_mat(rf_random, x_test, y_test)
```

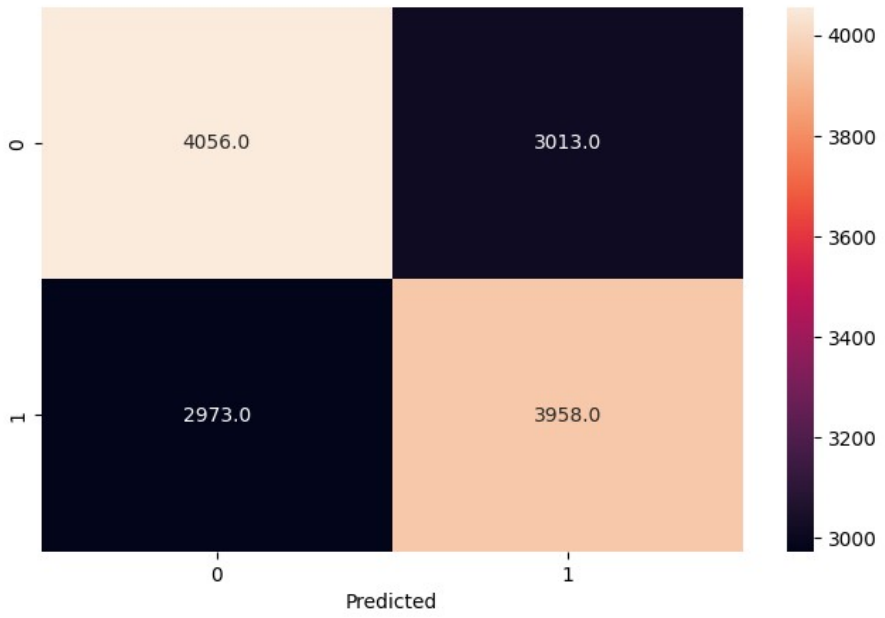

Рисунок 4.15 - Матриця невідповідностей для Random Forest

Побудуємо графік ROC для Logistic Regression.

```
In [34]: roc(rf_random, x_test, y_test)
```

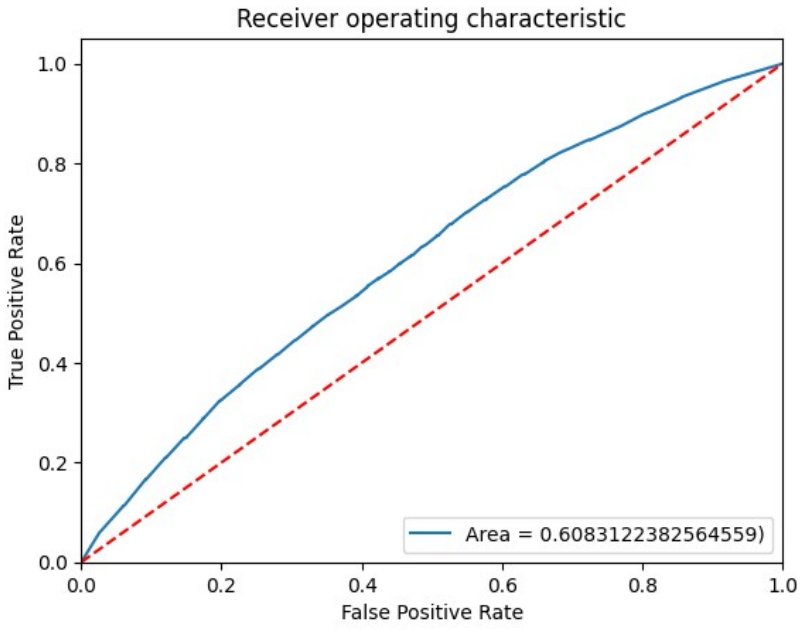

Рисунок 4.16 - Графік ROC для Random Forest

4.5 Аналіз отриманих результатів для методу Support Vector Machines

Для виконання роботи методу SVM імпортуємо sklearn.svm.SVC. Визначимо найкращі параметри для моделі. Оскільки складність SVM - це O(n_samples^2 * n_features), тобто для якщо факторів 3 і 70 000 зразків, то маємо 1.47e10 ітерацій, що надзвичайно багато. Тому оберемо 1000 випадковиз зразків і отримаємо загальну кільскість ітерацій в 3e6.

```
In [37]: from sklearn.svm import SVC
         import numpy as np
         df_svm = df.sample(n=500)
         svm_x = df_svm.iloc[:, :-1]
         svm_y = df_svm.iloc[:, -1]
         svm_x_train, svm_x_test, svm_y_train, svm_y_test = train_test_split(svm_x, svm_y, test_siz
         c = [1, 5]
         params = {'C': c, 'kernel': ['rbf', 'linear']}
         svc = SVC(gamma='auto', probability=True)
         svc_model = GridSearchCV(svc, param_grid=params, cv=3, n_jobs=5)
         svc_model.fit(svm_x_train, svm_y_train)
```

```
Out[37]: ▸ GridSearchCV
         ▸ estimator: SVC
             ▸ SVC
```

Рисунок 4.17 - Тренування моделі SVM

Визначимо точність моделі на тренувальних та тестових даних

```
In [54]: train_score = round(svc_model.score(x_train, y_train), 5)
         test_score = round(svc_model.score(x_test, y_test), 5)
         results.append({'method': 'svm', 'score': train_score, 'type': 'train'})
         results.append({'method': 'svm', 'score': test_score, 'type': 'test'})
         print(f'Train accuracy: {train_score}')
         print(f'Test accuracy: {test_score}')

         Train accuracy: 0.62605
         Test accuracy: 0.62371
```

Рисунок 4.18 - Точність моделі SVM

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

```
In [39]: conf_mat(svc_model, svm_x_test, svm_y_test)
```

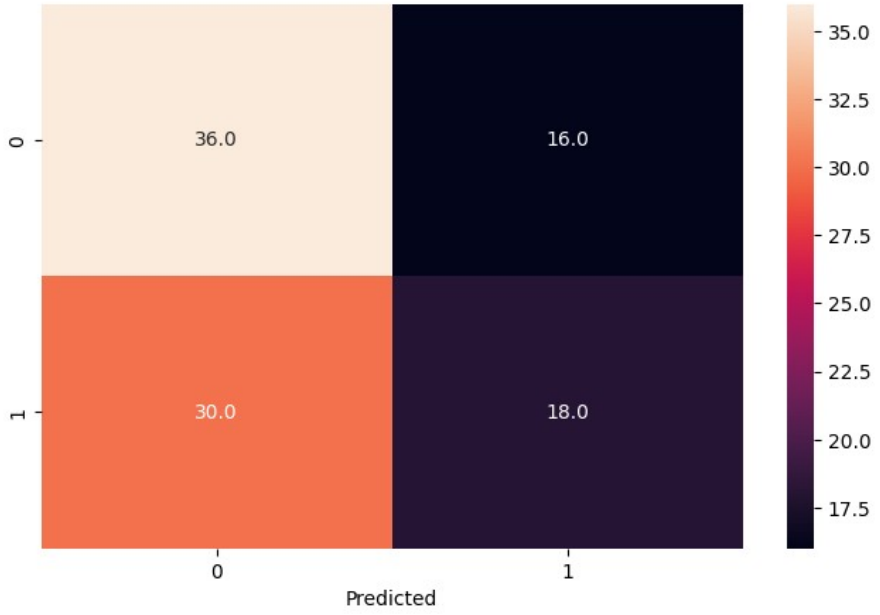

Рисунок 4.19 - Матриця невідповідностей для SVM

Побудуємо графік ROC для SVM.

```
In [40]: roc(svc_model, x_test, y_test)
```

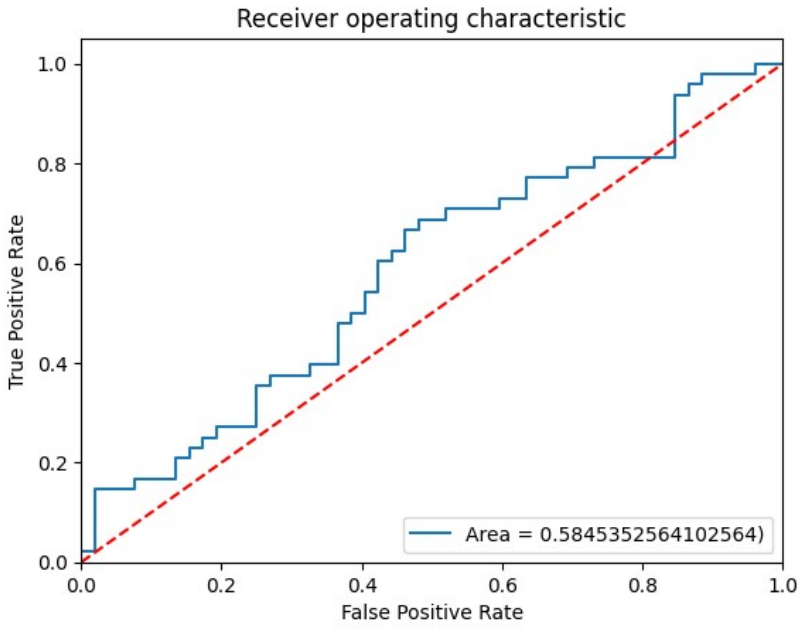

Рисунок 4.20 - Графік ROC для SVM

4.6 Порівняння отриманих результатів методів

Проаналізувавши окремо кожен із методів, проведемо порівнянняданих методів.

```
In [55]: df_score = pd.DataFrame(results, columns=['method','score','type'])
         df_score
```

Out[55]:

	method	score	type
0	knn	0.63252	train
1	knn	0.60807	test
2	logress	0.64145	train
3	logress	0.63729	test
4	rf	0.96564	train
5	rf	0.57243	test
6	svm	0.62605	train
7	svm	0.62371	test

Рисунок 4.21 - Датафрейм результатів

Для наочності побудуємо гістограму.

```
In [56]: sns.barplot(x='method', y='score', hue='type', data=df_score)

Out[56]: <AxesSubplot: xlabel='method', ylabel='score'>
```

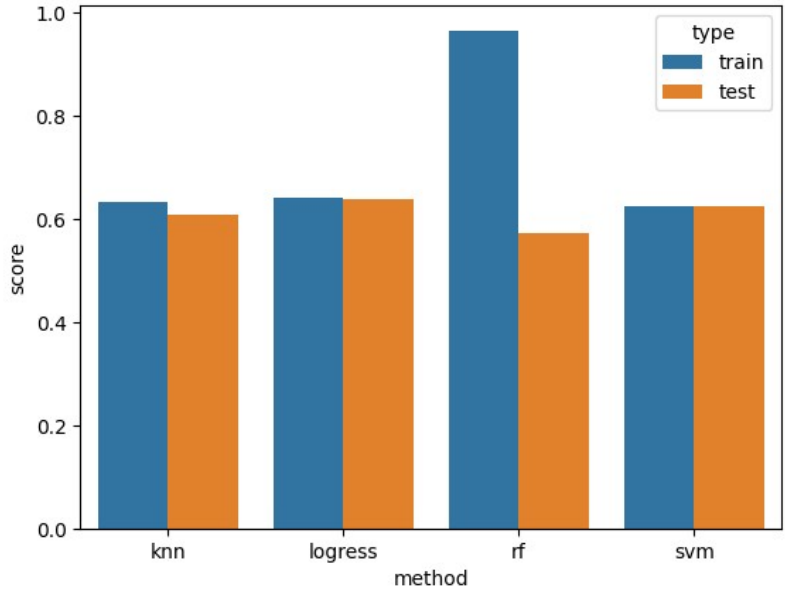

Рисунок 4.22 - Результати моделей

З огляду бачимо, що на тестових даних усі методи відпрацювали більш-менш однаково. Однак на тренувальних даних Random Forest показує себе найкраще.

ВИСНОВКИ

Аналіз результатів щодо серцево-судинних захворювань є критично важливим, оскільки він допомагає медичним працівникам краще зрозуміти фактори, які сприяють розвитку та прогресуванню серцево-судинних захворювань. Потім ці знання можна використати для розробки ефективніших стратегій профілактики та лікування.

Серцево-судинні захворювання, які включають такі стани, як ішемічна хвороба серця, інсульт і серцева недостатність, є основною причиною смерті в усьому світі. Вони спричинені поєднанням генетичних факторів, факторів навколишнього середовища та способу життя, що робить їх складними та важкими для вивчення.

Після аналізу даних встановлено, що метод Logistic Regression показав найкращі результати на тестових даних з точністю близько 63,73%. На тренувальних даних найкраще відпрацював Random Forest з точністю 96,56%, але на тестових даних його точність була найгіршою з результатом 57,24%. До того ж з матриць невідповідностей бачимо, що Logistic Regression виявив найбільше позитивно негативних результатів і найменше хибно негативних. Logistic Regression та SVM виявилися приблизно однаковими, проте через складність SVM було обрано набагато меншу вибірку ніж для Logistig Regression. Тому враховуючи те, що від діагностування хвороби залежить життя людини, я б обрав Logistic Regression.

Отже, висновок полягає в тому, що метод Logistic Regression є найбільш ефективним.

ПЕРЕЛІК ПОСИЛАНЬ

1. Документація мови програмування Python. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/

2. Бібліотека Pandas. [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/docs/

3. Бібліотека Seaborn. [Електронний ресурс] – Режим доступу до ресурсу: https://seaborn.pydata.org/introduction.html

4. Бібліотека Matplotlib. [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/stable/

5. Бібліотека Sklearn. [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/user_guide.html

6. Бібліотека NumPy. [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org

7. 190 - Finding the best model between Random Forest & SVM via hyperparameter tuning [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=f20fU6so580

8. Scikit-learn SVM Tutorial with Python (Support Vector Machines) [Електронний ресурс] – Режим доступу до ресурсу:

9. Scikit-learn SVM Tutorial with Python. [Електронний ресурс] – Режим доступу до ресурсу: https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python.

10. Hyperparameter Tuning the Random Forest in Python. [Електронний ресурс] — режим доступу до ресурсу: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

11. Optimization of hyper parameters for logistic regression in Python. [Електронний ресурс] — режим доступу до ресурсу: https://www.projectpro.io/recipes/optimize-hyper-parameters-of-logistic-regression-model-using-grid-search-in-python

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду прогнозування наявності серцево-судинних захворювань у людини на основі медичних показників. Методи K-Nearest Neighbors, Logistic Regression, Random Forest, SVM

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-11 2 курсу

Панченка С. В.

```
#!/usr/bin/env python
# coding: utf-8

# # 3 Робота з даними

# ## 3.1 Опис обраних даних

# ### Імпортуємо пакет pandas, завантажимо дані в датафрейм, виводимо інформацію про нього. Для вирішення поставленої задачі було обрано "Cardiovascular Disease" датасет. Він складається з 70 000 пацієнтів. Даний датасет містить в собі такі колонки, як-от: "age" - вік, "height" - зріст, "weight" - вага, "gender" - стать, "ap_hi" - систолічний кров’яний тиск, "ap_lo" - діастолічний артеріальний тиск, "cholesterol" - холестерол, "gluc" - глюкоза, "smoke" - чи курить пацієнт, "alco" - чи вживає алкоголь, "active" - фізична активність, "cardio" - присутність захворювання. Додатково імпортуємо модулі matplotlib.pyplot та seaborn для відображення графіків.

# In[46]:


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/cardio_train.csv', sep=';')
df


# *Завантажений датафрейм*

# ### Видалимо колонку id за допомогою методу pandas.Dataframe.drop, передавши в нього список з єдиним елементом id та параметром за замовчуванням inplace=True, щоб зміна відбулася в самому датафреймі, а не видало новий.

# In[ ]:


df.drop(['id'], axis=1, inplace=True)


# *Видалення колонки "id"*

# In[47]:


df.isna().any()


# *Колонки, де всі значення ініціалізовані*

# ## 3.2 Вибір ознак для аналізу

# ### Побудуємо матрицю кореляції та визначимо, наскільки кожен фактор впливає на якість води. Для цього імпортуємо модуль seaborn та застосуємо функцію heatmap, де передаємо йому вище зазначену матрицю, використавши pandas.Dataframe.corr метод датафрейму.

# In[13]:


import seaborn as sns
fig, axis = plt.subplots(figsize=(10, 6))
axis.set_title('Кореляція між факторами')
sns.heatmap(df.corr(), ax=axis, annot=True)


# *Матриця кореляцій*

# ### Бачимо, що найбільше на виникнення серцево-судинних захворювань впливає вік, вага, холестерол. Доволі цікавим фактами є кореляція між віком та холестеролом, статтю
```

та курінням, статтю та алкоголем, холестеролом та глюкозою. Залежність між статтю та вагою і висотою доволі очевидна, що пояснюється звичайною різницею у фізичних показниках між чоловіком та жінокю. Зобразимо статистику факторів та виникненням захворювання, розділену за статтю. За допомогою функції seaborn.FacetGrid згрупуємо значення та за допомогою seaborn.histplot зобразимо їх у вигляді гістограми. Побудуємо статистику за віком.

```
# In[48]:


def plot_stat(factor: str):
    g = sns.FacetGrid(df[['gender', 'cardio'] + [factor]], col='gender', hue='cardio')
    g.map(sns.histplot, factor)
    g.add_legend()
plot_stat('age')


# *Статистика захворювання від віку між чоловіками та жінками*

# ### Зобразимо статистику за вагою.

# In[15]:


plot_stat('weight')


# *Статистика захворювання від ваги між чоловіками та жінками*

# ### Зобразимо статистику за холестеролом.

# In[16]:


plot_stat('cholesterol')


# *Статистика захворювання від холестеролу між чоловіками та жінками*

# ## 3.3 Поділ даних

# ### Для збільшення якості моделі залишимо лише чотири колонки: age, weight, cholesterol, cardio.

# In[49]:


df = df[['age', 'weight', 'cholesterol', 'cardio']]
df


# *Відфільтрований датафрейм*

# ### Ділимо дані на тренувальні та тестові для подальшох роботи. Імпортуємо модуль sklearn.model_selection та застосуємо функцію train_test_split. Розділимо набір даних на 80% навчальних та 20% тестових.

# In[50]:


from sklearn.model_selection import train_test_split
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
train_size=0.8, random_state=0)


# *Поділ інформації на інформацію та результат*
```

```
# ### Зберігатимемо результати тестування моделей у списку results

# In[ ]:


results = []


# *Список результатів*

# # 4 Інтелектуальний аналіз даних

# ## 4.1 Обґрунтування вибору методів інтелектуального аналізу даних

# ### Було обрано чотири методи K-Nearest Neighbors, Logistic Regression, Random
Forest, SVM.

# ### K-Nearest Neighbors (KNN) — це простий, але ефективний алгоритм машинного
навчання, який використовується для завдань класифікації та регресії. Я обрав, щоб
порівняти наскільки він швидший за інші алгоритми, оскільки він не робить жодних
припущень щодо розподілу даних.

# ### До того ж однією з сильних сторін алгоритму KNN є те, що він легко
інтерпретується. Прогнози можна пояснити, просто подивившись на K найближчих сусідів
даної точки даних. Крім того, KNN може обробляти нелінійні межі рішень і може
використовуватися як для бінарних, так і для багатокласових задач класифікації. Тому
він чудово підійде для прогнозування хвороб, де треба враховувати багато факторів.

# ### Однак, якщо датасети великі, то KNN може бути дорогим з обчислювальної точки
зору. Крім того, продуктивність KNN значною мірою залежить від значення K, яке потрібно
ретельно вибирати для досягнення оптимальних результатів.

# ### Logistic Regression – це статистичний метод, який використовується для
прогнозування ймовірності події. Оскільки її залежна змінна є двійковою, то це дуже
підходить для виявлення хвороби: вона або є, або її нема.

# ### Logistic Regression має кілька переваг перед іншими алгоритмами класифікації. По-
перше, її легко реалізувати та інтерпретувати. По-друге, вона може обробляти як
категоричні, так і безперервні незалежні змінні. Тобто можна обробляти як дискретні
дані, які лекго визначити: вага, зріст - так і виявлення в організмі сполук, які
складно визначити точно, і де робляться припущення про інтервали: рівень глюкози,
рівень холестиролу, вітамінів в організмі тощо. По-третє, він може надати ймовірність
події, що станеться, що корисно під час медичної діагностики.

# ### Random Forest — це популярний алгоритм машинного навчання, який використовується
для завдань класифікації та регресії. Це метод ансамблю, який поєднує передбачення
кількох дерев рішень для отримання більш точних результатів.

# ### Я його обрав, бо є стійким до шуму. До того ж цікаво, як він буде передбачувати
наявність хвороби в пацієнта. Варто зауважити, що Random Forest може бути дорогим з
точки зору обчислень, особливо при роботі з великими наборами даних і багатьма
деревами. Також може бути важко інтерпретувати результати випадкового лісу, оскільки
процес прийняття рішення розподіляється між кількома деревами.

# ### SVM - це популярний і потужний клас алгоритмів керованого навчання, які
використовуються для завдань класифікації та регресії. SVM особливо добре підходять для
проблем, де існують складні межі між класами. У медичній сфері це часто
використовується для прогнузування пацієту декількох хвороб, де окреме захворювання -
це клас, де варто мати чіткі межі для поставлення правильного діагнозу.

# ### Основна ідея SVM — знайти гіперплощину, яка найкраще розділяє дані на різні
класи. Гіперплощина — це лінійна межа рішення, яка розділяє два класи в просторі ознак.

# ### Однією з сильних сторін SVM є те, що вони нечутливі до викидів, оскільки на
гіперплощину впливають лише найближчі до неї точки. Це робить SVM дуже стійкими до
зашумлених даних, де зазвичай пацієнтів дуже багато, а тому ця особливість є важливою.
```

```
# ### Однак SVM мають деякі обмеження. Навчання їх може бути дорогим з обчислювальної точки зору, особливо з великими наборами даних або просторами функцій великої розмірності. Крім того, вибір правильної функції ядра може бути складним, а продуктивність SVM залежить від вибору гіперпараметрів.

# ## 4.2 Аналіз отриманих результатів для методу K-Nearest Neighbors

# ### Для виконання роботи методу KNN імпортуємо sklearn.neighbors.KNeighborsClassifier та sklearn.model_selection.GridSearchCV.

# In[19]:


from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV


# *Імпортування модулів*

# ### Визначимо, які варіанти параметрів найкраще вирішують дану задачу.

# In[20]:


classificator = KNeighborsClassifier()
params = {'n_neighbors': range(1, 60)}
grid_search = GridSearchCV(classificator, params, cv=10, verbose=1)
grid_search.fit(x_train, y_train)
knn = grid_search.best_estimator_
knn


# *Визначення найкращого параметра*

# ### Натренуємо модель з найкращим параметром.

# In[21]:


knn.fit(x_train, y_train)


# *Тренування моделі K-Nearest Neighbors*

# ### Визначимо точність моделі на тренувальних та тестових даних

# In[51]:


train_score = round(knn.score(x_train, y_train), 5)
test_score = round(knn.score(x_test, y_test), 5)
results.append({'method': 'knn', 'score': train_score, 'type': 'train'})
results.append({'method': 'knn', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')


# *Точність моделі K-Nearest Neighbors*

# ### Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей. Для цього застосуємо sklearn.metrics.plot_confusion_matrix.

# In[23]:


from sklearn.metrics import confusion_matrix
def conf_mat(model, x_test, y_test):
    y_predicted = model.predict(x_test)
```

```
    cm = confusion_matrix(y_test, y_predicted)
    plt.figure(figsize = (8,5))
    sns.heatmap(cm, annot=True, fmt=".1f")
    plt.xlabel('Predicted')


# *Імпортування модуля*

# ### Матриця має два рядки та дві колонки: перший ряд і перша колонка - це істинно позитивні значення, тобто людина здорова і модель не визначила захворювання; першмй ряд і друга колонка - хибно позитивні, тобто людина здорова, а модель сказала, що є захворювання; другий ряд і перша колонка - хибно негативні, тобто людина хвора, а модель сказала, що здорова; другий ряд і друга колонка - істинно негативні, тобто людина хвора і модель сказала, що хвора.

# In[24]:


conf_mat(knn, x_test, y_test)


# *Матриця невідповідностей для K-Nearest Neighbors*

# ### Побудуємо графік ROC( Receiver Operating Characteristic ), що є графіком істинно позитивної відносної частоти проти хибно позитивної частоти. Це показує компроміс між чутливістю та специфічністю. Для цього імпортуємо sklearn.metrics.roc_curve та sklearn.metrics.roc_auc_score. До того ж визначимо AUC( Area Under the ROC Curve ), що є мірою того, наскільки добре модель може розрізняти позитивні та негативні рузультати. Він коливається від 0 до 1, де 1 є найкращим класифікатором, а 0,5 – випадковим класифікатором. AUC корисний під час порівняння продуктивності різних класифікаторів на одному наборі даних, бо даж єдине число, яке підсумовує загальну продуктивність.

# In[25]:


from sklearn.metrics import roc_curve, roc_auc_score
def roc(model, x_test, y_test):
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = roc_curve(y_test,  y_pred_proba)
    auc = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="Area = "+str(auc)+')')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc=4)
    plt.show()


# *Імпортування модуля та визначення функції roc*

# ### Побудуємо ROC для K-Nearest Neighbors.

# In[26]:


roc(knn, x_test, y_test)


# *Графік ROC для K-Nearest Neighbors*

# ## 4.3 Аналіз отриманих результатів для методу Logistic Regression

# ### Для виконання роботи методу KNN імпортуємо sklearn.linear_model.LogisticRegression; sklearn.pipeline.Pipeline для побудови пайплайну, щоб у зручному вигляді передавати параметри до декількох функцій;
```

sklearn.preprocessing.StandardScaler для скейлингу даних до проміжку від 0 до 1. Визначимо найкращі параметри моделі, передавши в неї параметри регуляризації.

In[27]:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
logisticRegr = LogisticRegression()
pipe = Pipeline(steps=[('sc', sc), ('logisticRegr', logisticRegr)])
c = np.logspace(-4, 4, 60)
penalty = ['l1', 'l2']
params = dict(logisticRegr__C=c, logisticRegr__penalty=penalty)
log_reg = GridSearchCV(pipe, params)
log_reg.fit(x_train, y_train)
```

Тренування моделі Logistic Regression

Визначимо точність моделі на тренувальних та тестових даних

In[52]:

```
train_score = round(log_reg.score(x_train, y_train), 5)
test_score = round(log_reg.score(x_test, y_test), 5)
results.append({'method': 'logress', 'score': train_score, 'type': 'train'})
results.append({'method': 'logress', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')
```

Точність моделі Logistic Regression

Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

In[29]:

```
conf_mat(log_reg, x_test, y_test)
```

Матриця невідповідностей для Logistic Regression

Побудуємо графік ROC для Logistic Regression.

In[30]:

```
roc(log_reg, x_test, y_test)
```

Графік ROC для Logistic Regression

4.4 Аналіз отриманих результатів для методу Random Forest

Для виконання роботи методу KNN імпортуємо sklearn.ensemble.RandomForestClassifier. Визначимо найкращі параметри для моделі. У випадку Random Forest параметри включають кількість дерев рішень та кількість характеристик, які враховуються кожним деревом під час поділу вузла.використовуються для поділу кожного вузла, отриманого під час навчання. Імпортуємо sklearn.model_selection.RandomizedSearchCV.

In[31]:

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
# Кількість дерев
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, num = 60)]
params = {'n_estimators': n_estimators}
rf = RandomForestClassifier()
rf_random = GridSearchCV(rf, param_grid=params, cv=3, n_jobs=5)
rf_random.fit(x_train, y_train)


# *Тренування моделі Random Forest*

# ### Визначимо точність моделі на тренувальних та тестових даних

# In[53]:


train_score = round(rf_random.score(x_train, y_train), 5)
test_score = round(rf_random.score(x_test, y_test), 5)
results.append({'method': 'rf', 'score': train_score, 'type': 'train'})
results.append({'method': 'rf', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')


# *Точність моделі Random Forest*

# ### Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

# In[33]:


conf_mat(rf_random, x_test, y_test)


# *Матриця невідповідностей для Random Forest*

# ### Побудуємо графік ROC для Logistic Regression.

# In[34]:


roc(rf_random, x_test, y_test)


# *Графік ROC для Random Forest*

# ## 4.5 Аналіз отриманих результатів для методу Support Vector Machines

# ### Для виконання роботи методу SVM імпортуємо sklearn.svm.SVC. Визначимо найкращі
параметри для моделі. Оскільки складність SVM - це O(n_samples^2 * n_features), тобто
для якщо факторів 3 і 70 000 зразків, то маємо 1.47e10 ітерацій, що надзвичайно багато.
Тому оберемо 1000 випадковиз зразків і отримаємо загальну кільскість ітерацій в 3e6.

# In[37]:


from sklearn.svm import SVC
import numpy as np
df_svm = df.sample(n=500)
svm_x = df_svm.iloc[:, :-1]
svm_y = df_svm.iloc[:, -1]
svm_x_train, svm_x_test, svm_y_train, svm_y_test = train_test_split(svm_x, svm_y,
test_size=0.2, train_size=0.8, random_state=0)
c = [1, 5]
params = {'C': c, 'kernel': ['rbf', 'linear']}
svc = SVC(gamma='auto', probability=True)
```

```
svc_model = GridSearchCV(svc, param_grid=params, cv=3, n_jobs=5)
svc_model.fit(svm_x_train, svm_y_train)


# *Тренування моделі SVM*

# ### Визначимо точність моделі на тренувальних та тестових даних

# In[54]:


train_score = round(svc_model.score(x_train, y_train), 5)
test_score = round(svc_model.score(x_test, y_test), 5)
results.append({'method': 'svm', 'score': train_score, 'type': 'train'})
results.append({'method': 'svm', 'score': test_score, 'type': 'test'})
print(f'Train accuracy: {train_score}')
print(f'Test accuracy: {test_score}')


# *Точність моделі SVM*

# ### Визначимо продуктивність роботи моделі на прикладі матриці невідповідностей.

# In[39]:


conf_mat(svc_model, svm_x_test, svm_y_test)


# *Матриця невідповідностей для SVM*

# ### Побудуємо графік ROC для SVM.

# In[40]:


roc(svc_model, x_test, y_test)


# *Графік ROC для SVM*

# ## 4.6 Порівняння отриманих результатів методів

# ### Проаналізувавши окремо кожен із методів, проведемо порівнянняданих методів.

# In[55]:


df_score = pd.DataFrame(results, columns=['method','score','type'])
df_score


# *Датафрейм результатів*

# ### Для наочності побудуємо гістограму.

# In[56]:


sns.barplot(x='method', y='score', hue='type', data=df_score)


# *Результати моделей*

# ### З огляду бачимо, що на тестових даних усі методи відпрацювали більш-менш
однаково. Однак на тренувальних даних Random Forest показує себе найкраще.
```