МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

КУРСОВА РОБОТА

з дисципліни “Бази даних”

спеціальність 121 – Програмна інженерія

на тему: _____
(назва теми)

Студент
групи КП-02 **Чекурда А. В.** _____
(ПІБ) (підпис)

Викладач
к.т.н, доцент кафедри
СПіСКС **Петрашенко А.В.** _____
(підпис)

Захищено з оцінкою _____

Київ – 2021

Анотація

При виконанні курсової роботи був реалізований такий функціонал:

1. CRUD операції сутностей
2. Генерація звіту з графіком
3. Реплікація даних
4. Модуль аутентифікації
5. Аналіз даних та прогноз цін

Під час запитів були застосовані засоби підвищення швидкодії. Отже, результатом цієї роботи стала програма з консольним інтерфейсом, орієнтована на взаємодію з базою даних PostgreSQL 13.

Зміст

Вступ

Даний проєкт є прототипом великого магазину, у якому є безліч товарів різних категорій, керування додатком здійснюється лише адміністраторами, тому модуль ідентифікації та реєстрації дає змогу лише так званим модераторам вносити корективи до бази даних товарів. Звичайно ж, оскільки нині ми можемо спостерігати розростання торгових центрів, інших магазинів, то такий проєкт є дуже актуальним і кожна компанія або має свій додаток або прагне його реалізації. Тому загальною постановкою завдання було забезпечення безпечного та правильного управління даними, а також додаткових функцій, які доповнюють цей проєкт. Серед них була реалізована генерація звіту, яка значно покращує ведення документообігу компанії, аналіз даних для прогнозування цін товарів певної категорії та бренду для вдалого перебігу купівлі та продажу товарів, визначення цін, які будуть задовольняти користувачів. В подальших пунктах буде розібрано якраз функціональність проєкту та більш конкретно описано, як здійснювалась розробка та реалізація тої чи іншої функції. Коротка характеризація призначення: допоміжна

програма для додатків з товарами для спрощення керування даними, тобто певний інструментарій, яким будуть користуватись адміністратори магазинів, в даному випадку додаток не має конкретних типів товарів, на кшталт мобільних телефонів чи подібного, там містяться товари будь-якого типу.

Аналіз інструментарію для виконання курсової роботи

В першу чергу слід порівняти інструментарії для виконання машинного навчання, а саме за джерелами з інтернету було знайдено два можливих варіанти такої реалізації: Shogun та ML.Net. В чому полягає їх різниця, а в чому ж вони схожі? В першу чергу саме з умови моєї поставленої задачі, ці алгоритми повинні включати тип навчання - регресію, необхідну для виконання прогнозування цін, що власне вони обоє вміють чудово виконувати. Проте, є суттєва відмінність, ML.Net навіть з назви натякає, що він створений саме під C# з урахуванням особливостей написання коду на цій мові, а Shogun є універсальним фреймворком, який можна використовувати на Python, Java, C++, C# в тому числі, однак саме оформлення є узагальненим для спрощення розуміння розробниками особливостей інших мов, які вони не вчили наприклад. Але, оскільки більшу кількість свого часу я програмую на C#, то і особливості цієї мови мене не лякали, тому використано=им у цьому випадку був ML.Net. Щодо СУБД, то за умовою можливо було вибрати лише або MySql або PostgreSQL, і оскільки певне знайомство з MySql у мене вже було, я пішов від супротивного і вирішив опанувати PostgreSQL, допоміжним додатком для неї став pgAdmin4, в якому можна було безпосередньо проводити операції над БД. Фреймворк для взаємодії з PostgreSQL на C# є одним без альтернатив, це Npgsql, за допомогою нього можна створювати конект з базою даних та писати sql запити до неї. Для генерації графіків була зосереджена увага на ScottPlot через те, що в мене вже була присутня практики роботи з цим фреймворком та наявність великої кількості типів графіків для їх реалізації.

Аналіз функціонування засобів реплікації

Тестування реплікації проводилось при запуску самого проєкту, спочатку вводились деякі команди до терміналу(їх суть полягає в створенні резервної бази даних, в яку будуть зберігатись дані), а потім запускався додаток, в консолі необхідно було ввести команду, яка візуально впливає на дані(додавання, редагування нового елементу), і потім порівняти дві бази даних, резервну в якості сервера, на якому зберігається ця інформація та оригінальну, з якою йде робота. Якщо дані співпадають в обох базах даних, то реплікація працює відмінно. Самий тип реалізованої реплікації - логічна. Це означає, що в нас є підписник на основну базу даних - резервна база даних, яка

отримує доступ до даних публікації(оригінальна база даних), а потім копіює їх та дає можливість краще слідкувати за безпекою даних.

Аналіз функціонування засобів резервування/відновлення бази даних

Резервування та відновлення в PostgreSQL можна здійснити двома методами: через косноль командами, або за допомогою вбудованих функцій pgAdmin4. З pgAdmin4 все просто: слід натиснути правою кнопкою на базу даних, яку хочеш зарезервувати, потім натиснути на команду backup, яка з'явиться після кліку, після цього вказати шлях до файлу, де будуть зберігатись резервована копія і резервування готове. Точно так само працює і відновлення, тільки вже необхідно вказати шлях до файлу, де розташована копія. Щодо консолі, то в ній є влаштована команда pg_dump, за допомогою якої і здійснюється резервування, а для відновлення використовується psql. Ось приклади команд на моїй базі даних:

```
pg_dump -U postgres -d courseWorkdb -f
C:/Database_Coursework/consoleApp/data/postgre.dump
psql -U postgres -d restored_db -f
C:/Database_Coursework/consoleApp/data/postgre.dump
```

Аналіз результатів підвищення швидкодії виконання запитів

Запити виконуються досить тривалий час напряму, оскільки необхідно створювати кожен раз нове з'єднання з базою даних та витягувати різні дані, що і призводить до втрати швидкодії. Шляхи її оптимізації, які були використані в моїй курсовій роботі - це опрацювання деяких запитів через контекст, тобто з лише одним з'єднання та через списки сутностей, що оптимізовувало деякі запити. А також використання індексів для швидшого опрацювання вже прямих запитів до бази даних. В моєму випадку серед індексів були використані BTree та Gin. BTree є основним інструментарієм для підвищення пошуку числових даних, а Gin оптимізовує пошук по рядку. Загалом, структура BTree - це дерево, яку має дуже багато листків, однак малу висоту, за рахунок цього і виконується швидкий пошук. Gin за структурою схожий на BTree, однак всередині кожного листка зберігається лексема з бази

даних, яка співставляється з індексом сутності з BTree, в якій вона з’являється. Прикладами ілюстрації служить виведений час у моїй програмі після пошуку з індексами та без, на якому видно, що з індексами програма набагато швидша. В додатку наведений графік розподілу часу на виконання пошуку однієї сутності з індексами та без них.

Опис результатів аналізу предметної галузі

Реалізація мого алгоритму полягала в генерації певних псевдовипадкових даних на основі товару, який ми мали аналізувати, а потім надання методу машинного навчання можливості знайти закономірності в тих псевдовипадкових даних та на їх основі спрогнозувати ціну товару, якщо коротко говорити. Якщо ж вдаватись в подробиці, то було створено три допоміжних класи для реалізації алгоритму, один служив перебудованою сутністю товару для її аналізу, інший вміщував основний критерій, за яким буде відбуватись аналіз, останній вже реалізовував всю логіку, а саме містив метод Train, за допомогою якого здійснювалось тренування машини на псевдовипадкових даних, Evaluate для виведення результату та підрахування потенційної помилки. Основним методом був TestSinglePrediction, в якому очевидно з назви здійснювалось безпосереднє прогнозування ціни. Виведення цієї функції наведено в додатку.

Висновки

Отже, почнемо з аутентифікації: користувач при запуску додатку має можливість вибрати лише дві команди registrate та authenticate. За допомогою них користувач відповідно або створює нового модератора даного магазину або входить під вже існуючим ім’ям. Далі після цього йому доступний такий перелік команд:

1. update i(c,b,m) id - модернізація певної сутності за ідентифікатором
2. delete i(c,b,m) id - видалення певної сутності за ідентифікатором
3. get i(c,b,m) id - витяг певної сутності з бд
4. search items(categories, brands, mods) - пошук сутностей за підрядком чи іншими полями, зауваженням є при введенні просто search відбувається одночасний пошук всіх сутностей
5. predict id - прогнозування ціни конкретного товари за ідентифікатором
6. generate plot - генерація графіку порівняння швидкодій з індексами та без них

7. generate docx - генерація звіту з інформацією про товари, а саме виведення найдорожчого та найдешевшого товару та графік розподілу цін на товари.
8. exit or “” - вихід з програми під цим користувачем

Слід також звернути увагу, що генерація всіх даних і псевдовипадкових, і з датасетів та навіть простим додаванням винесена в окрему програму generation, тому функцій додавання в основній частині немає, вони присутні в додатковій.

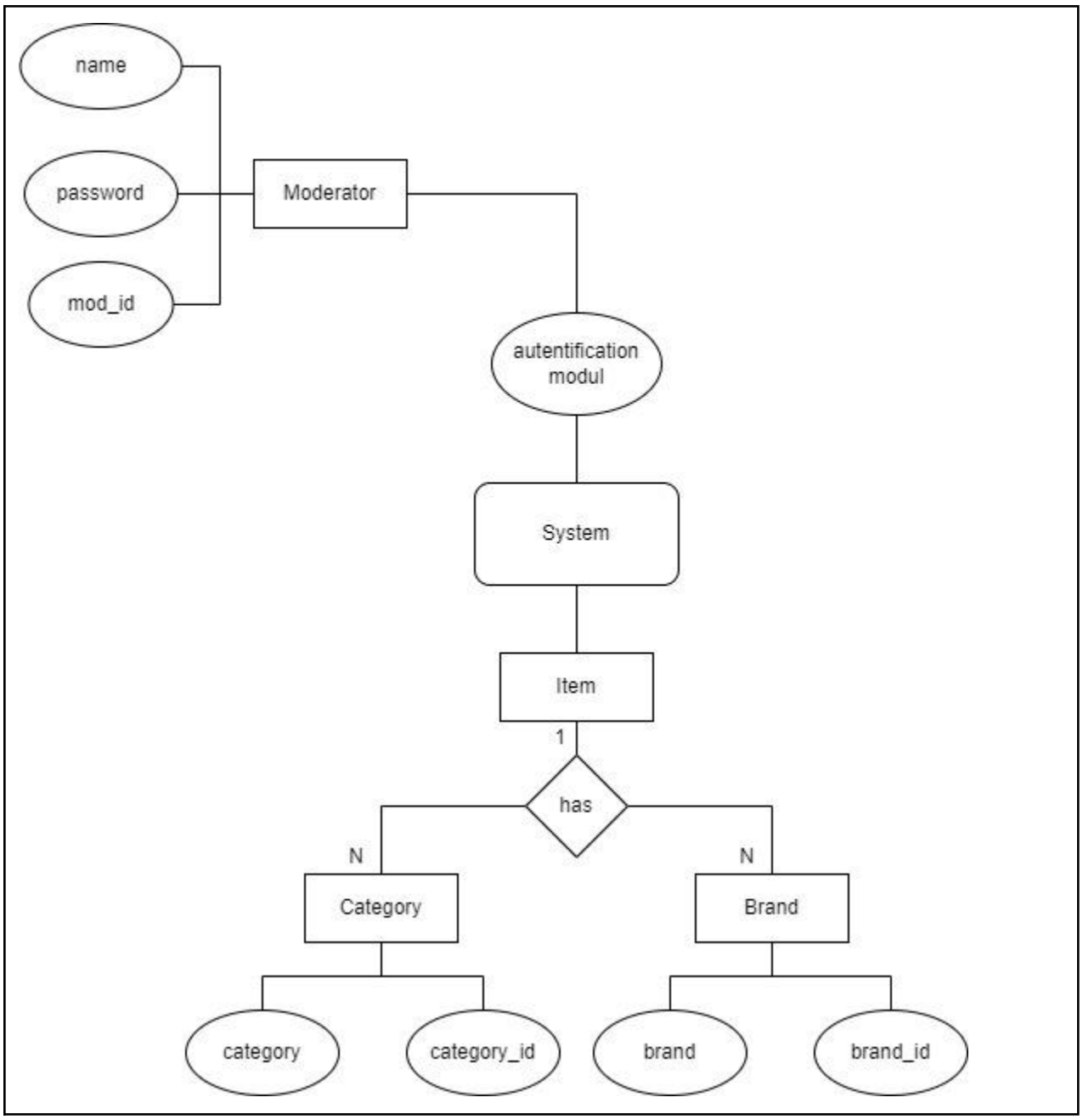
Отож, дана програма успішно реагує на некоректні дані, вміє здійснювати пошук сутностей, прогнозувати ціни на товари, генерувати по них звіт, а також робити різні маніпуляції з самими даними, такими як видалення, модернізація, та просто їх перегляд. Згідно з цим можна сказати, що мета цієї роботи успішно виконана, оскільки був створений чудовий додаток для взаємодії з базою даних, який точно знайде свій застосунок у будь-якому магазині. Щодо етапів розробки, то найбільш цікавим є прогноз цін, однак з ними виникали певні проблеми, оскільки сутності Category та Brand, присутні в класі Item, не давали тренувати машину, виходом з цієї ситуації стало створення нового класу, який виніс в себе основні поля Item, тому в прогнозуванні навіть присутній спеціальний метод переведення звичайного Item в Item для прогнозування. З рештою функціоналу проблем особливих не виникло. Також був під час проведення роботи модернізований клас контексту для здійснення реплікації даних.

Література

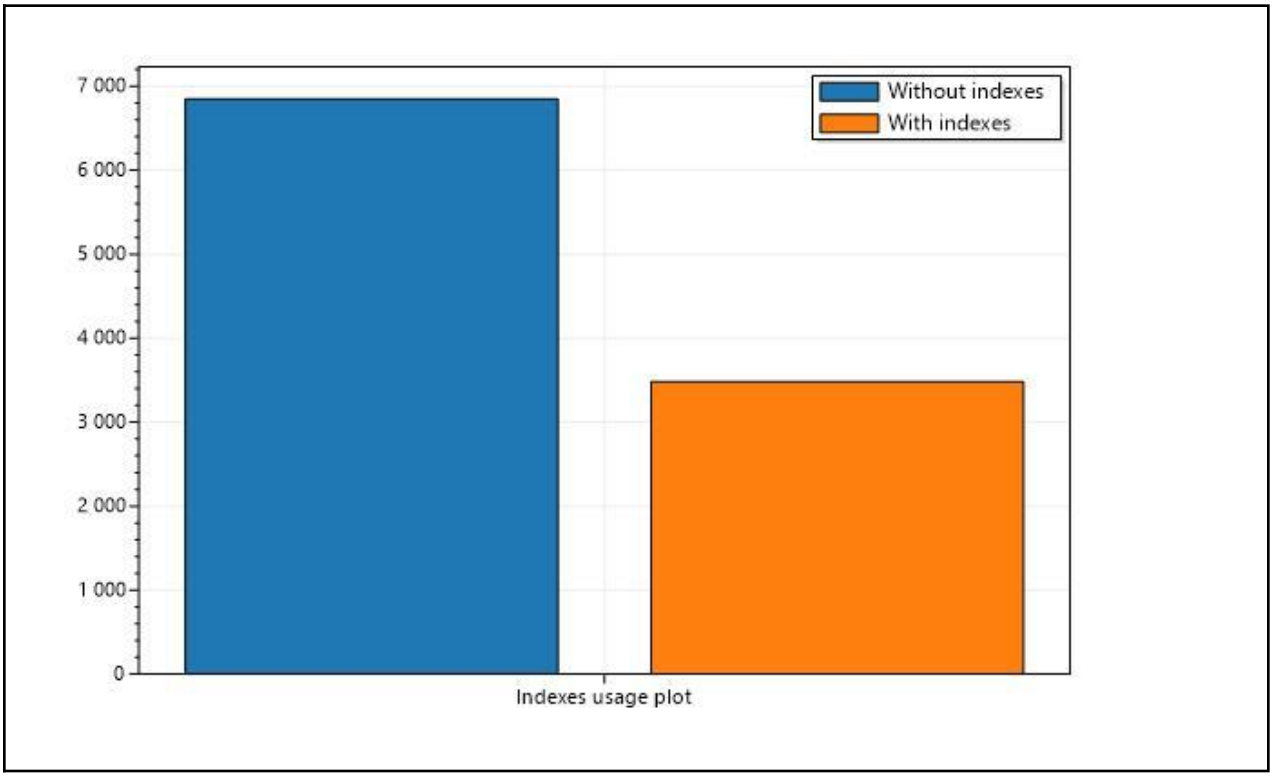
1. ML.Net documentation https://docs.microsoft.com/en-us/dotnet/machine-learning/
2. Shogun toolbox https://en.wikipedia.org/wiki/Shogun_(toolbox)
3. MySql meaning and using https://www.mysql.com/
4. PostgreSQL https://www.postgresql.org/
5. Npgsql https://www.npgsql.org/
6. ScottPlot https://scottplot.net/
7. PostgresPro - logical replication https://postgrespro.ru/docs/postgresql/10/logical-replication
8. Gin index https://ru.wikipedia.org/wiki/GIN
9. PostgresPro - text search indexes https://postgrespro.ru/docs/postgrespro/9.5/textsearch-indexes
10. Habr - indexes in PostgreSQL https://habr.com/ru/company/postgrespro/blog/330544/

Додатки

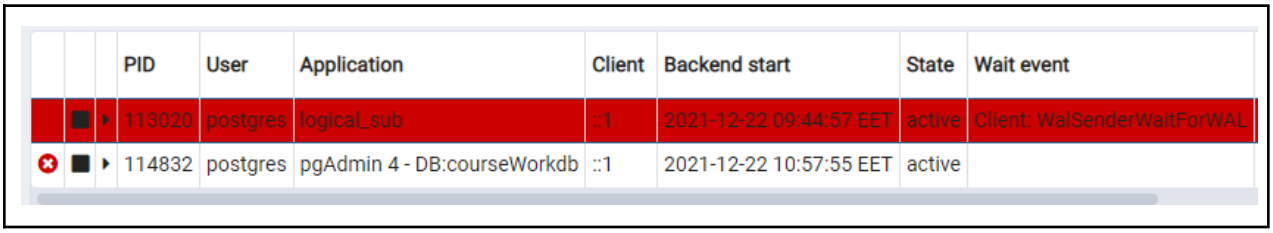
Структура роботи проєкту з базою даних:



Графік часу на запити пошуку з індексами та без них:

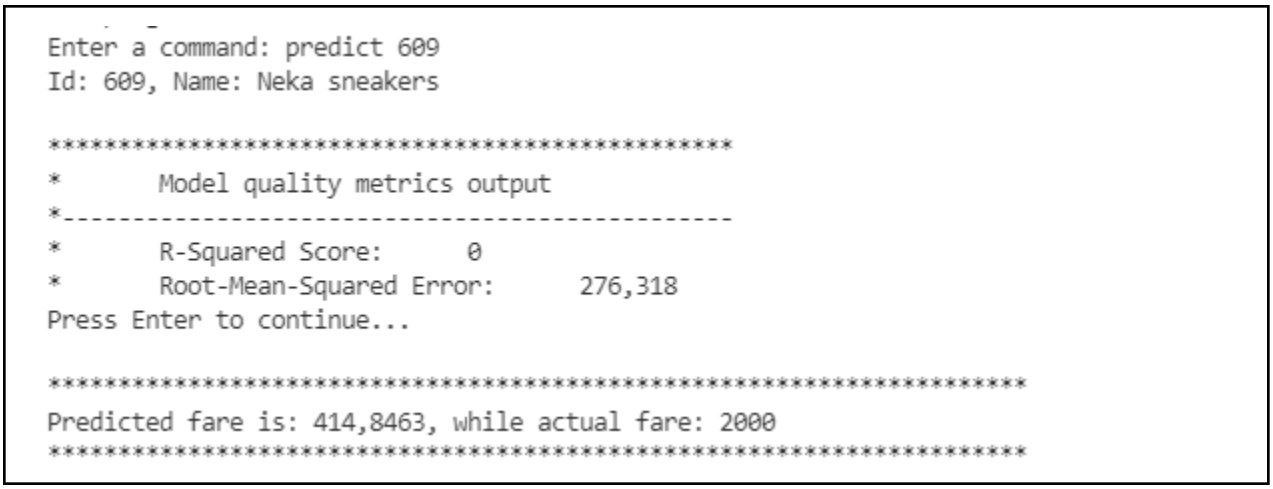


Виведення pgAdmin4 про реплікацію системи:



	PID	User	Application	Client	Backend start	State	Wait event
	113020	postgres	logical_sub	::1	2021-12-22 09:44:57 EET	active	Client: WalSenderWaitForWAL
	114832	postgres	pgAdmin 4 - DB:courseWorkdb	::1	2021-12-22 10:57:55 EET	active	

Виведення консолі про прогнозування цін та коефіцієнт можливої поилки при цій дії:

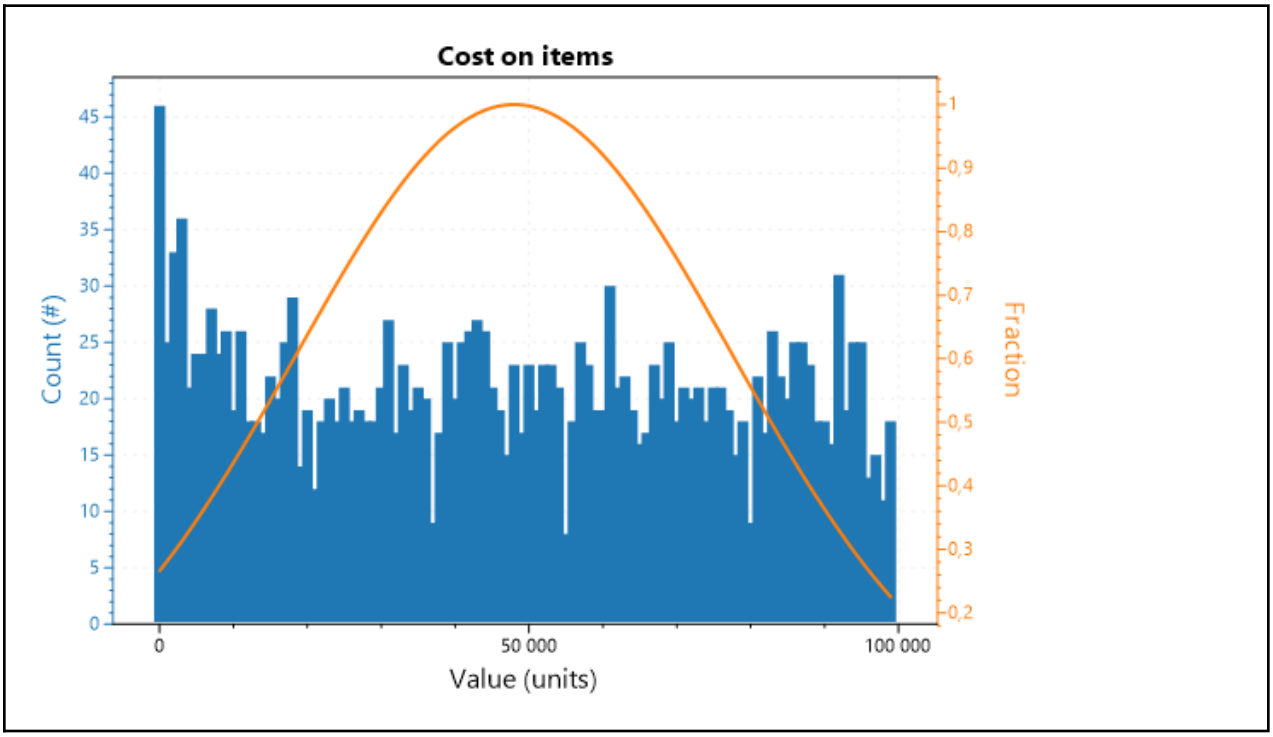


```
Enter a command: predict 609
Id: 609, Name: Neka sneakers

************************************************
*       Model quality metrics output
*------------------------------------------------
*       R-Squared Score:      0
*       Root-Mean-Squared Error:      276,318
Press Enter to continue...

*************************************************************************
Predicted fare is: 414,8463, while actual fare: 2000
*************************************************************************
```

Згенерований графік аналізу цін магазину:



Фрагменти програмного коду:
Модуль аутентифікації

```
using RepoCode;
namespace consoleApp
{
    public class Autentification
    {
        public ModRepository moderators;
        public Autentification(ModRepository moderators)
        {
            this.moderators = moderators;
        }
        public int GetRegistration(Moderator mod, string password)
        {
            long count = moderators.GetUniqueNamesCount(mod.name);
            if(count != 0)
            {
                return -1;
            }
            else
            {
                mod.password = password;
                int mod_id = (int)moderators.Insert(mod);
                return mod_id;
            }
        }
        public int GetAutentification(string name, string password)
        {
            long count = moderators.GetUniqueNamesCount(name);
```

```
            if(count == 1)
            {
                Moderator moderator = moderators.GetByName(name);
                bool check = VerifyHash(password, moderator.password);
                if(check)
                {
                    return moderator.mod_id;
                }
                else
                {
                    return -1;
                }
            }
            else
            {
                return -1;
            }
        }
        private bool VerifyHash(string input, string output)
        {
            return input == output;
        }
    }
}
```

Приклад сутності(Item)

```
namespace RepoCode
{
    public class Item
    {
        public int item_id;
        public string name;
        public double cost;
        public int brand_id;
        public int category_id;
        public int createYear;
        public virtual Category category { get; set; }
        public virtual Brand brand { get; set; }
        public Item(string name, double cost, int brand_id, int category_id, int createYear)
        {
            this.name = name;
            this.cost = cost;
            this.brand_id = brand_id;
            this.category_id = category_id;
            this.createYear = createYear;
        }
        public Item()
        {
            this.name = "Beta item";
            this.cost = 1000;
            this.brand_id = 173;
            this.category_id = 171;
            this.createYear = 2018;
```

```
        }
        public override string ToString()
        {
            return $"Id: {this.item_id}, Name: {this.name}";
        }
    }
}
```

Приклад репозиторію сутності(конкретно ItemRepository)

```
using System;
using static System.Console;
using Npgsql;

namespace RepoCode
{
    public class ItemRepository
    {
        private NpgsqlConnection connection;
        private courseWorkdbContext context;
        public ItemRepository(string connString, courseWorkdbContext context)
        {
            this.connection = new NpgsqlConnection(connString);
            this.context = context;
        }
        public Item GetById(int id)
        {
            Item item = context.items.Find(id);
            if (item == null)
            {
                throw new NullReferenceException("Cannot find an object with such id.");
            }
            else
                return item;
        }
        public bool DeleteById(int id)
        {
            Item item = context.items.Find(id);
            if (item == null)
            {
                return false;
            }
            else
            {
                context.items.Remove(item);
                context.SaveChanges();
                return true;
            }
        }
        public bool DeleteAllByCtgId(int id)
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
```

```
            command.CommandText = @"DELETE FROM items WHERE category_id = @ctg_id";
            command.Parameters.AddWithValue("ctg_id", id);
            int nChanged = (int)command.ExecuteScalar();
            connection.Close();
            return nChanged == 1;
        }
        public bool DeleteAllByBrandId(int id)
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"DELETE FROM items WHERE brand_id = @brand_id";
            command.Parameters.AddWithValue("brand_id", id);
            int nChanged = (int)command.ExecuteScalar();
            connection.Close();
            return nChanged == 1;
        }
        public object Insert(Item item)
        {
            context.items.Add(item);
            context.SaveChanges();
            return item.item_id;
        }
        public bool Update(Item item)
        {
            try
            {
                Item itemToUpdate = context.items.Find(item.item_id);
                if (itemToUpdate == null)
                    return false;
                else
                    itemToUpdate.name = item.name;
                    itemToUpdate.cost = item.cost;
                    itemToUpdate.category_id = item.category_id;
                    itemToUpdate.brand_id = item.brand_id;
                    itemToUpdate.createYear = item.createYear;
                context.SaveChanges();
                return true;
            }
            catch(Exception ex)
            {
                WriteLine(ex.Message);
                return false;
            }
        }
        public Item GetMinValue()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"SELECT MIN(cost) FROM items";
            double minValue = (double)command.ExecuteScalar();
            NpgsqlCommand command1 = connection.CreateCommand();
            command1.CommandText = @"SELECT * FROM items WHERE cost <= @minValue";
            command1.Parameters.AddWithValue("minValue", minValue);
            NpgsqlDataReader reader = command1.ExecuteReader();
            Item item = new Item();
```

```
            if(reader.Read())
            {
                item.item_id = reader.GetInt32(0);
                item.name = reader.GetString(1);
                item.cost = reader.GetDouble(2);
                item.brand_id = reader.GetInt32(3);
                item.category_id = reader.GetInt32(4);
                item.createYear = reader.GetInt32(5);
            }
            connection.Close();
            return item;
        }
        public Item GetMaxValue()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"SELECT MAX(cost) FROM items";
            double maxValue = (double)command.ExecuteScalar();
            NpgsqlCommand command1 = connection.CreateCommand();
            command1.CommandText = @"SELECT * FROM items WHERE cost >= @maxValue";
            command1.Parameters.AddWithValue("maxValue", maxValue);
            NpgsqlDataReader reader = command1.ExecuteReader();
            Item item = new Item();
            if(reader.Read())
            {
                item.item_id = reader.GetInt32(0);
                item.name = reader.GetString(1);
                item.cost = reader.GetDouble(2);
                item.brand_id = reader.GetInt32(3);
                item.category_id = reader.GetInt32(4);
                item.createYear = reader.GetInt32(5);
            }
            connection.Close();
            return item;
        }
        public double[] GetItemCosts()
        {
            double[] values = new double[GetCount()];
            List<Item> items = GetAll();
            Item[] array = new Item[items.Count];
            items.CopyTo(array);
            for(int i = 0; i < array.Length; i++)
            {
                values[i] = array[i].cost;
            }
            return values;
        }
        public long GetCount()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"SELECT COUNT(*) FROM items";
            long count = (long)command.ExecuteScalar();
            connection.Close();
            return count;
```

```
        }
        public List<Item> GetAllSearch(string value, int[] measures)
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"SELECT * FROM items WHERE name LIKE '%' || @value || '%'
                or cost >= @a AND cost <= @b";
            command.Parameters.AddWithValue("value", value);
            command.Parameters.AddWithValue("a", measures[0]);
            command.Parameters.AddWithValue("b", measures[1]);
            NpgsqlDataReader reader = command.ExecuteReader();
            List<Item> list = new List<Item>();
            while(reader.Read())
            {
                Item item = new Item();
                item.item_id = reader.GetInt32(0);
                item.name = reader.GetString(1);
                item.cost = reader.GetDouble(2);
                item.brand_id = reader.GetInt32(3);
                item.category_id = reader.GetInt32(4);
                item.createYear = reader.GetInt32(5);
                list.Add(item);
            }
            connection.Close();
            return list;
        }
        public List<Item> GetAll()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"SELECT * FROM items";
            NpgsqlDataReader reader = command.ExecuteReader();
            List<Item> list = new List<Item>();
            while(reader.Read())
            {
                Item item = new Item();
                item.item_id = reader.GetInt32(0);
                item.name = reader.GetString(1);
                item.cost = reader.GetDouble(2);
                item.brand_id = reader.GetInt32(3);
                item.category_id = reader.GetInt32(4);
                item.createYear = reader.GetInt32(5);
                list.Add(item);
            }
            connection.Close();
            return list;
        }
        public void AddingIndexes()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"
                CREATE INDEX if not exists items_cost_idx ON items (cost);
                CREATE INDEX if not exists items_name_idx ON items using GIN (name);
            ";
```

```
            int nChanged = command.ExecuteNonQuery();
            connection.Close();
        }
        public void DroppingIndexes()
        {
            connection.Open();
            NpgsqlCommand command = connection.CreateCommand();
            command.CommandText = @"
                DROP INDEX if exists items_cost_idx;
                DROP INDEX if exists items_name_idx;
            ";
            int nChanged = command.ExecuteNonQuery();
            connection.Close();
        }
    }
}
```

Модуль генерації звіту:

```
using System;
using System.Xml;
using System.Xml.Linq;
using System.IO;
using System.IO.Compression;
using System.Collections.Generic;
using RepoCode;
using System.Drawing;
namespace consoleApp
{
    public class GatherInfo
    {
        private ItemRepository items;
        public GatherInfo(ItemRepository items)
        {
            this.items = items;
        }
        public double GetAverage()
        {
            double[] values = items.GetItemCosts();
            double sum = 0;
            for(int i = 0; i < values.Length; i++)
            {
                sum += values[i];
            }
            double average = sum / values.Length;
            return average;
        }
        public void Main()
        {
            ClearAndDeleteFolder("./data/output/");
            string zipPath = @"./data/report.docx";
            string extractPath = @"./data/output";
            ZipFile.ExtractToDirectory(zipPath, extractPath);
            XElement root = XElement.Load("./data/output/word/document.xml");
```

```
            Dictionary<string, string> dict = new Dictionary<string, string>
            {
                {"{count}", $"{items.GetCount()}"},
                {"{average}", $"{GetAverage()}"},
                {"{name}", $"{items.GetMaxValue().name}"},
                {"{cost}", $"{items.GetMaxValue().cost}"},
                {"{createYear}", $"{items.GetMaxValue().createYear}"},
                {"{name1}", $"{items.GetMinValue().name}"},
                {"{cost1}", $"{items.GetMinValue().cost}"},
                {"{createYear1}", $"{items.GetMinValue().createYear}"}
            };
            Bitmap bmp = new Bitmap("./data/image.png");
            bmp.Save("./data/output/word/media/image1.png");
            FindAndReplace(root, dict);
            root.Save("./data/output/word/document.xml", SaveOptions.DisableFormatting);
            DateTime now = DateTime.Now;
            ZipFile.CreateFromDirectory("./data/output", $"./data/report{now.Minute}.docx");
        }
        public void FindAndReplace(XElement node, Dictionary<string, string> dict)
        {
            if (node.FirstNode != null
                && node.FirstNode.NodeType == XmlNodeType.Text)
            {
                string replacement;
                if (dict.TryGetValue(node.Value, out replacement))
                {
                    node.Value = replacement;
                }
            }
            foreach (XElement el in node.Elements())
            {
                FindAndReplace(el, dict);
            }
        }
        public void ClearAndDeleteFolder(string folderPath)
        {
            DirectoryInfo dirInfo = new DirectoryInfo(folderPath);
            foreach(FileInfo file in dirInfo.GetFiles())
            {
                file.Delete();
            }
            dirInfo.Delete(true);
        }
    }
}
```

Модуль генерації зображення

```
using RepoCode;
using ScottPlot;
namespace consoleApp
{
    public class ImageGenerator
    {
```

```
        private string filePath;
        private ItemRepository items;
        public ImageGenerator(string filePath, ItemRepository items)
        {
            this.filePath = filePath;
            this.items = items;
        }
        public void GenerateImage()
        {
            var plt = new ScottPlot.Plot(600, 400);
            double[] values = items.GetItemCosts();
            Item minItem = items.GetMinValue();
            Item maxItem = items.GetMaxValue();
            var hist = new ScottPlot.Statistics.Histogram(values, min: minItem.cost, max: maxItem.cost);

            // plot the bins as a bar graph (on the primary Y axis)
            var bar = plt.AddBar(hist.counts, hist.bins);
            bar.BarWidth = hist.binSize * 1.5; // oversize to reduce render artifacts
            bar.BorderLineWidth = 0;
            bar.YAxisIndex = 0;
            plt.YAxis.Label("Count (#)");
            plt.YAxis.Color(bar.FillColor);

            // plot the mean curve as a scatter plot (on the secondary Y axis)
            var sp = plt.AddScatter(hist.bins, hist.countsFracCurve);
            sp.MarkerSize = 0;
            sp.LineWidth = 2;
            sp.YAxisIndex = 1;
            plt.YAxis2.Label("Fraction");
            plt.YAxis2.Color(sp.Color);
            plt.YAxis2.Ticks(true);

            // decorate the plot
            plt.XAxis2.Label("Cost on items", bold: true);
            plt.XAxis.Label("Value (units)");
            plt.SetAxisLimits(yMin: 0);
            plt.Grid(lineStyle: LineStyle.Dot);

            plt.SaveFig(filePath);
        }
        public void GenerateGraphic(double num1, double num2)
        {
            var plt = new ScottPlot.Plot(600, 400);

            // generate random data to plot
            int groupCount = 1;
            Random rand = new(0);
            double[] values1 = new double[]{num1};
            double[] values2 = new double[]{num2};
            double[] errors1 = new double[]{0};
            double[] errors2 = new double[]{0};

            // group all data together
            string[] groupNames = {"Indexes usage plot"};
            string[] seriesNames = { "Without indexes", "With indexes"};
```

```
            double[][] valuesBySeries = { values1, values2};
            double[][] errorsBySeries = { errors1, errors2};

            // add the grouped bar plots and show a legend
            plt.AddBarGroups(groupNames, seriesNames, valuesBySeries, errorsBySeries);
            plt.Legend(location: Alignment.UpperRight);

            // adjust axis limits so there is no padding below the bar graph
            plt.SetAxisLimits(yMin: 0);

            plt.SaveFig(this.filePath);
        }
    }
}
```

Основний модуль прогнозування цін

```
using Microsoft.ML;
using RepoCode;
namespace PredictionLib
{
    public class CostPrediction
    {
        public ItemForPrediction ItemToItemForPrediction(Item item, CategoryRepository ctgs, BrandRepository
brands)
        {
            Category category = ctgs.GetById(item.category_id);
            Brand brand = brands.GetById(item.brand_id);
            ItemForPrediction predict_item = new ItemForPrediction();
            predict_item.item_id = item.item_id;
            predict_item.name = item.name;
            predict_item.cost = (float)item.cost;
            predict_item.brand = brand.brand;
            predict_item.category = category.category;
            predict_item.createYear = item.createYear;
            return predict_item;
        }
        public ITransformer Train(MLContext mlContext, string dataPath)
        {
            // The IDataView object holds the training dataset
            IDataView dataView = mlContext.Data.LoadFromTextFile<ItemForPrediction>(dataPath, hasHeader: true,
separatorChar: ',');

            var pipeline = mlContext.Transforms.CopyColumns(outputColumnName: "Label", inputColumnName: "cost")
            .Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName: "ItemIdEncoded",
inputColumnName: "item_id"))
                .Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName: "NameEncoded",
inputColumnName: "name"))
                .Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName: "BrandEncoded",
inputColumnName: "brand"))
                .Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName: "CategoryEncoded",
inputColumnName: "category"))
                .Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName: "CreateYearEncoded",
inputColumnName: "createYear"))
```

```
                .Append(mlContext.Transforms.Concatenate("Features", "BrandEncoded", "CategoryEncoded",
"CreateYearEncoded"))
                .Append(mlContext.Regression.Trainers.FastTree());

            //Create the model
            var model = pipeline.Fit(dataView);

            //Return the trained model
            return model;
        }
        public void Evaluate(MLContext mlContext, ITransformer model, string _testDataPath)
        {
            IDataView dataView = mlContext.Data.LoadFromTextFile<ItemForPrediction>(_testDataPath, hasHeader:
true, separatorChar: ',');
            var predictions = model.Transform(dataView);
            var metrics = mlContext.Regression.Evaluate(predictions, "Label", "Score");

            Console.WriteLine();
            Console.WriteLine($"*************************************************");
            Console.WriteLine($"*       Model quality metrics output         ");
            Console.WriteLine($"*------------------------------------------------");

            Console.WriteLine($"*       R-Squared Score:      {metrics.RSquared:0.###}");

            Console.WriteLine($"*       Root-Mean-Squared Error:      {metrics.RootMeanSquaredError:#.###}");
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        public void TestSinglePrediction(MLContext mlContext, ITransformer model, ItemForPrediction item)
        {
            var predictionFunction = mlContext.Model.CreatePredictionEngine<ItemForPrediction,
ItemCostPrediction>(model);

            //Create a single TaxiTrip object to be used for predictin
            //Make a prediction
            var prediction = predictionFunction.Predict(item);

            Console.WriteLine($"**********************************************************************");
            Console.WriteLine($"Predicted fare is: {prediction.cost:0.####}, while actual fare: {item.cost}");
            Console.WriteLine($"**********************************************************************");
        }
    }
}
```