



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

**Институт информационных технологий (ИИТ)
Кафедра прикладной математики (ПМ)**

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ
по дисциплине «Технологии и инструментарий анализа больших данных»

Практическая работа № 2

Студент группы *ИКБО-01-22 Прокопчук Роман Олегович*

(подпись)

Ассистент *Тетерин Николай Николаевич*

(подпись)

Отчёт представлен «__» сентября 2025 г.

Москва, 2025 г.

ЦЕЛЬ

Ознакомиться с различными библиотеками визуализации данных (matplotlib, plotly, TSNE, UMAP) и особенностями работы с ними в среде программирования Python.

ХОД РАБОТЫ

Задание

1. Найти и выгрузить многомерные данные (с большим количеством признаков – столбцов) с использованием библиотеки pandas. В отчёте описать найденные данные.
2. Вывести информацию о данных при помощи методов `.info()`, `.head()`. Проверить данные на наличие пустых значений. В случае их наличия удалить данные строки или интерполировать пропущенные значения. При необходимости дополнительно предобработать данные для дальнейшей работы с ними.
3. Построить столбчатую диаграмму (`.bar`) с использованием модуля `graph_objs` из библиотеки Plotly со следующими параметрами:
 - 3.1. По оси X указать дату или название, по оси Y указать количественный показатель.
 - 3.2. Сделать так, чтобы столбец принимал цвет в зависимости от значения показателя (`marker=dict(color=признак, coloraxis="coloraxis")`).
 - 3.3. Сделать так, чтобы границы каждого столбца были выделены чёрной линией с толщиной равной 2.
 - 3.4. Отобразить заголовок диаграммы, разместив его по центру сверху, с 20 размером текста.
 - 3.5. Добавить подписи для осей X и Y с размером текста, равным 16. Для оси абсцисс развернуть метки так, чтобы они читались под углом, равным 315.
 - 3.6. Размер текста меток осей сделать равным 14.
 - 3.7. Расположить график во всю ширину рабочей области и присвоить высоту, равную 700 пикселей.
 - 3.8. Добавить сетку на график, сделать её цвет 'ivory' и толщину равную 2. (Можно сделать это при настройке осей с помощью `gridwidth=2, gridcolor='ivory'`)
 - 3.9. Убрать лишние отступы по краям.

4. Построить круговую диаграмму (`go.Pie`), используя данные и стиль оформления из предыдущего графика. Сделать так, чтобы границы каждой доли были выделены чёрной линией с толщиной, равной 2 и категории круговой диаграммы были читаемы (к примеру, объединить часть объектов)

5. Построить линейные графики, взять один из параметров и определить зависимость между другими несколькими (от 2 до 5) показателями с использованием библиотеки `matplotlib`. Сделать вывод.

5.1. Сделать график с линиями и маркерами, цвет линии `'crimson'`, цвет точек `'white'`, цвет границ точек `'black'`, толщина границ точек равна 2.

5.2. Добавить сетку на график, сделать её цвет `'mystyrose'` и толщину равную 2. (Можно сделать это при настройке осей с помощью `linewidth=2, color='mystyrose'`).

6. Выполнить визуализацию многомерных данных, используя `t-SNE`. Необходимо использовать набор данных `MNIST` или `fashion MNIST` (можно использовать и другие готовые наборы данных, где можно наблюдать разделение объектов по кластерам). Рассмотреть результаты визуализации для разных значений перплексии.

7. Выполнить визуализацию многомерных данных, используя `UMAP` с различными параметрами `n_neighbors` и `min_dist`. Рассчитать время работы алгоритма с помощью библиотеки `time` и сравнить его с временем работы `t-SNE`.

8. На основе проделанной работы составить отчёт с описанием и скриншотами полученных результатов, сделать выводы о выбранных данных на основе полученных графиков, сравнить библиотеки. Начиная с 6 пункта отчёт дополнительно должен содержать результаты визуализации для разных значений параметров и выводы.

В качестве данных для практической был выбран датасет с данным о пит стопах Формулы-1 с 1950 по 2024 года. Ссылка -

<https://www.kaggle.com/datasets/akashrane2609/formula-1-pit-stop-dataset?resource=download>.

На рисунке 1 представлен скрипт для загрузки и предобработки данных.

```
import pandas as pd

csv_file = "Formula1_Pitstop_Data_1950-2024_all_rounds.csv"

df = pd.read_csv(csv_file)

print("=== Информация о таблице ===")
print(df.info())
print("\n=== Первые 5 строк ===")
print(df.head())

print("\n=== Количество пропусков в каждом столбце ===")
print(df.isna().sum())

df_clean = df.dropna()

df_clean.reset_index(drop=True, inplace=True)

df_clean.to_csv(path_or_buf="Formula1_Pitstop_Data_clean.csv", index=False)
```

Рисунок 1 – Загрузка и преобразование данных

На рисунке 2 представлен код для создания столбчатой диаграммы. Созданная диаграмма представлена на рисунке 3.

```

import pandas as pd
import plotly.graph_objs as go
import ast

df = pd.read_csv("Formula1_Pitstop_Data_clean.csv")

Windsurf: Refactor | Explain | Docstring | ✕
def count_stops(text): 1 usage new *
    if pd.isna(text) or text.strip() == "":
        return 0
    try:
        return len(ast.literal_eval(text))
    except Exception:
        return 0

df["PitStopCount"] = df["PitStops"].apply(count_stops)

counts = df.groupby("Season")["PitStopCount"].sum().reset_index(name="pit_count")

trace = go.Bar(
    x=counts["Season"],
    y=counts["pit_count"],
    marker=dict(
        color=counts["pit_count"],
        coloraxis="coloraxis",
        line=dict(color="black", width=2) # чёрные границы
    )
)

layout = go.Layout(
    title=dict(
        text="Общее количество ПИТ-СТОПОВ по сезонам",
        x=0.5,
        font=dict(size=20)
    ),
    xaxis=dict(
        title=dict(text="Сезон", font=dict(size=16)),
        tickangle=315,
        tickfont=dict(size=14),
        showgrid=True,
        gridwidth=2,
        gridcolor="ivory"
    ),
    yaxis=dict(
        title=dict(text="Число ПИТ-СТОПОВ", font=dict(size=16)),
        tickfont=dict(size=14),
        showgrid=True,
        gridwidth=2,
        gridcolor="ivory"
    ),
    coloraxis=dict(colorscale="Viridis"),
    height=700,
    margin=dict(l=40, r=40, t=80, b=80)
)

fig = go.Figure(data=[trace], layout=layout)
fig.write_html(*args: "bar_plot.html", include_plotlyjs="cdn")

```

Рисунок 2 — Код для создания столбчатой диаграммы

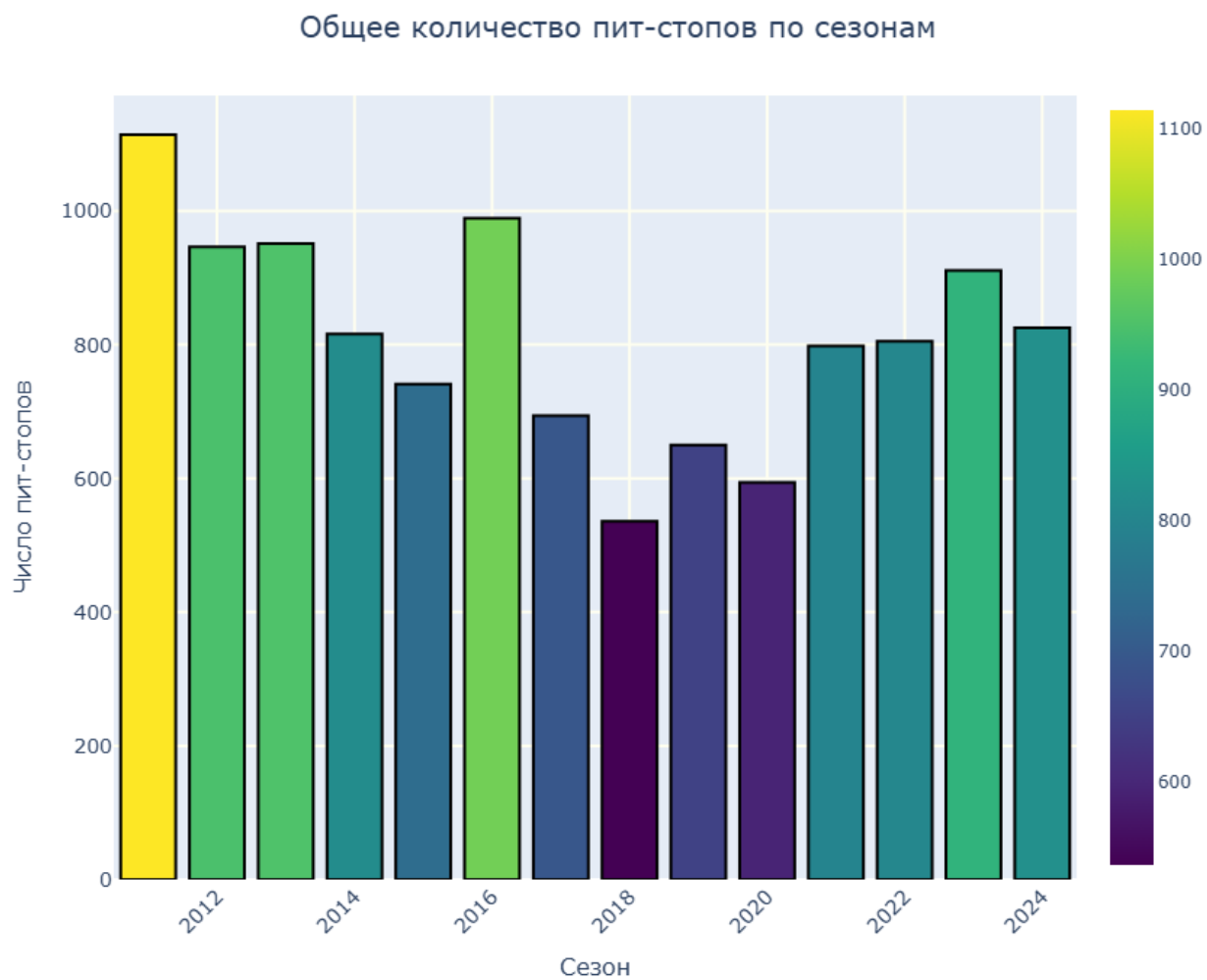


Рисунок 3 – Созданная столбчатая диаграмма

Код для создания круговой диаграммы представлен на рисунке 4.
Созданная диаграмма представлена на рисунке 5.

```

import pandas as pd
import plotly.graph_objs as go
import ast

df = pd.read_csv("Formula1_Pitstop_Data_clean.csv")

Windsurf: Refactor | Explain | Docstring | ✕
def count_stops(text): 1 usage new *
    if pd.isna(text) or text.strip() == "":
        return 0
    try:
        return len(ast.literal_eval(text))
    except Exception:
        return 0

df["PitStopCount"] = df["PitStops"].apply(count_stops)

counts = df.groupby("Season")["PitStopCount"].sum().reset_index(name="pit_count")

fig = go.Figure(
    go.Pie(
        labels=counts["Season"].astype(str),
        values=counts["pit_count"],
        marker=dict(line=dict(color="black", width=2))
    )
)

fig.update_layout(
    title=dict(
        text="Доля пит-стопов по всем сезонам",
        x=0.5,
        font=dict(size=20)
    ),
    height=700,
    margin=dict(l=40, r=40, t=80, b=40)
)

fig.write_html(*args: "pie_plot.html", include_plotlyjs="cdn")

```

Рисунок 4 – Код для создания круговой диаграммы



Рисунок 5 – Созданная круговая диаграмма

Код для создания линейных графиков представлен на рисунке 6. Созданные графики – на рисунка 7 и 8.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("Formula1_Pitstop_Data_clean.csv")

race_stats = df.groupby(["Season", "Round"]).agg(
    AvgPitStopTime=("AvgPitStopTime", "mean"),
    TotalPitStops=("TotalPitStops", "sum"),
    Drivers=("Driver", "count"),
    Laps=("Laps", "max")
).reset_index()

race_stats["AvgPitStopsPerDriver"] = race_stats["TotalPitStops"] / race_stats["Drivers"]

stats = race_stats.groupby("Laps").agg(
    AvgPitStopTime=("AvgPitStopTime", "mean"),
    AvgPitStopsPerDriver=("AvgPitStopsPerDriver", "mean")
).reset_index()

# === График 1: среднее время пит-стопа ===
plt.figure(figsize=(12, 6))
plt.plot(
    *args: stats["Laps"], stats["AvgPitStopTime"],
    color="crimson", marker="o",
    markerfacecolor="white", markeredgewidth=2, linewidth=2, label="Среднее время пит-стопа (с)"
)
plt.grid(linewidth=2, color="mistyrose")
plt.title(label: "Зависимость времени пит-стопа от длины гонки", fontsize=16)
plt.xlabel(xlabel: "Количество кругов в гонке", fontsize=14)
plt.ylabel(ylabel: "Среднее время пит-стопа (с)", fontsize=14)
plt.legend(fontsize=12)
plt.tight_layout()
plt.savefig(*args: "line_plot_time.png", dpi=300)
plt.show()

# === График 2: среднее число пит-стопов на гонщика ===
plt.figure(figsize=(12, 6))
plt.plot(
    *args: stats["Laps"], stats["AvgPitStopsPerDriver"],
    color="crimson", marker="s",
    markerfacecolor="white", markeredgewidth=2, linewidth=2, label="Среднее число пит-стопов на гонщика"
)
plt.grid(linewidth=2, color="mistyrose")
plt.title(label: "Зависимость числа пит-стопов от длины гонки", fontsize=16)
plt.xlabel(xlabel: "Количество кругов в гонке", fontsize=14)
plt.ylabel(ylabel: "Среднее число пит-стопов", fontsize=14)
plt.legend(fontsize=12)
plt.tight_layout()
plt.savefig(*args: "line_plot_stops.png", dpi=300)
plt.show()
```

Рисунок 6 – Код для создания линейных графиков

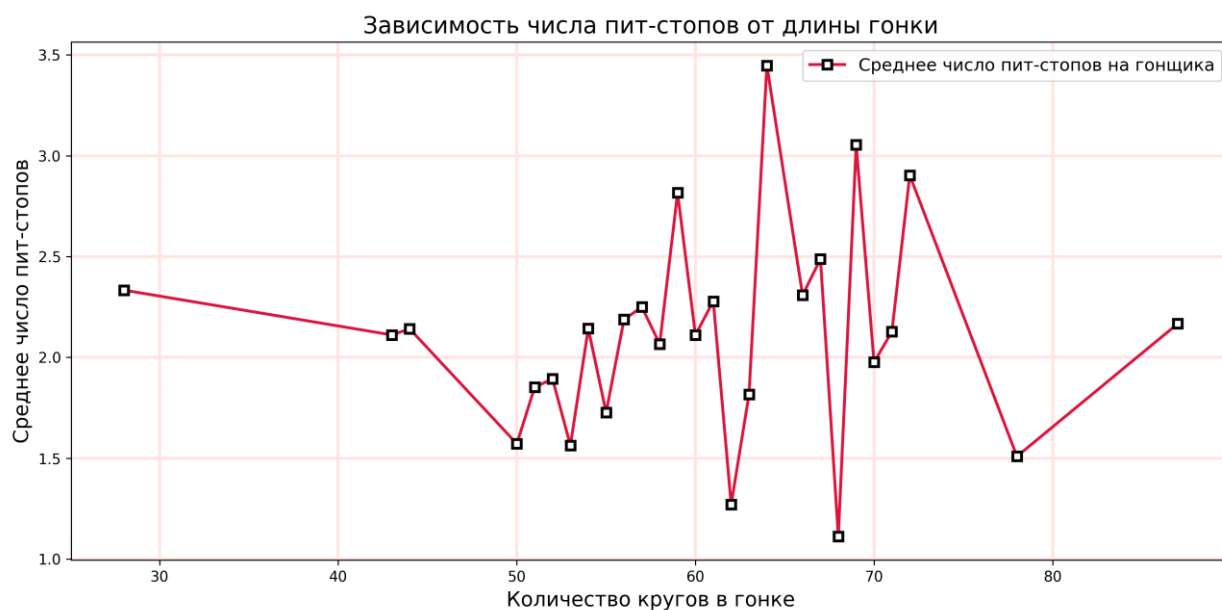


Рисунок 7 – Зависимость числа пит-стопов от кол-ва кругов в гонке

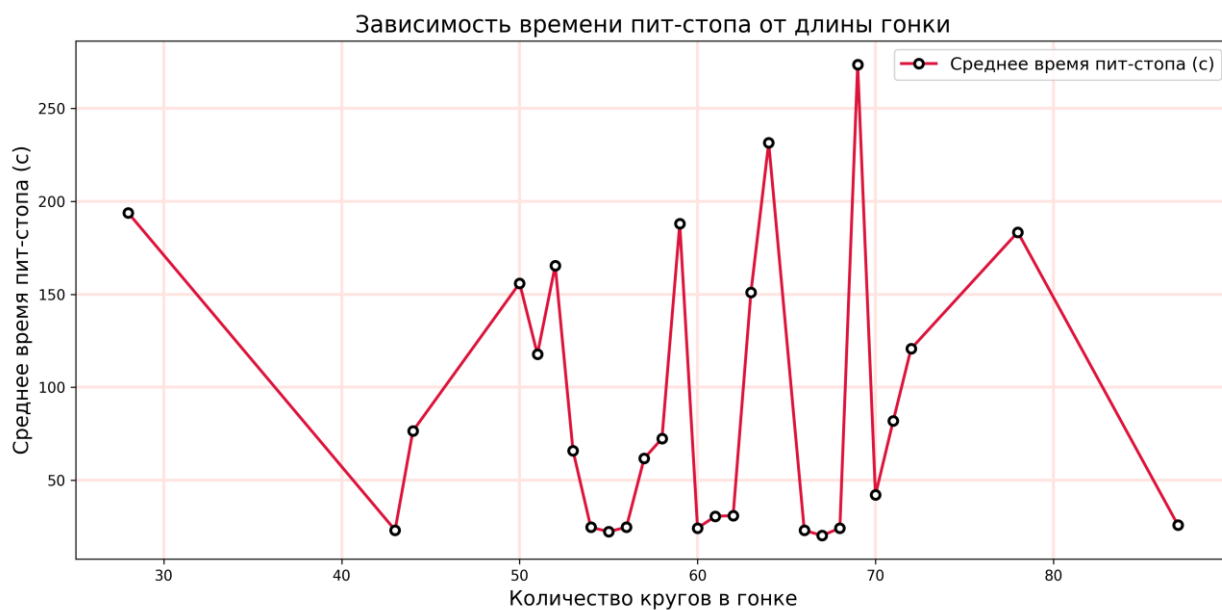


Рисунок 8 – Зависимость времени пит-стопов от кол-ва кругов в гонке

Из полученных графиков видно, что кол-во пит-стопов в гран-при формулы 1, а также их длительность никак не соотносятся с кол-вом кругов в гонке. Эти параметры сильно зависят от внешних факторов. В первую очередь это касается температуры трассы, возможных осадков во время гонок, кол-ва аварий.

На рисунках 9 и 10 представлен код для визуализации многомерных данных при помощи t-SNE и UMAP соответственно.

```

import time
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.datasets import fetch_openml
import umap

print("Загрузка данных Fashion MNIST...")
X, y = fetch_openml( name: 'Fashion-MNIST', version=1, return_X_y=True, as_frame=False, parser='liac-arff', c

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

print(f"Данные загружены. Используется полный набор данных из {X.shape[0]} объектов.")
print("Внимание: вычисления могут занять много времени!")

X_full = X / 255.0
y_full = y.astype(int)

# --- Визуализация с помощью t-SNE (4 запуска) ---
print("\n--- Запуск t-SNE ---")
perplexities = [5, 30, 50, 100]
tsne_total_start_time = time.time()

for perplexity in perplexities:
    print(f"Вычисление t-SNE с perplexией = {perplexity}...")
    start_time = time.time()

    tsne = TSNE(
        n_components=2,
        perplexity=perplexity,
        random_state=42,
    )

    X_tsne = tsne.fit_transform(X_full)
    end_time = time.time()
    print(f"t-SNE с perplexией = {perplexity} завершен за {end_time - start_time:.2f} секунд.")

    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_full, cmap=plt.get_cmap( name: "jet", lut: 10), s=1)
    plt.title(f't-SNE с perplexией = {perplexity} (на полных данных)')
    plt.xlabel('Компонента 1')
    plt.ylabel('Компонента 2')
    handles, _ = scatter.legend_elements(num=10)
    plt.legend( *args: handles, class_names, title="Классы")
    plt.show()

tsne_total_end_time = time.time()
tsne_duration = tsne_total_end_time - tsne_total_start_time
print(f"Общее время выполнения t-SNE: {tsne_duration:.2f} секунд")

```

Рисунок 9 – Загрузка данных и их визуализация при помощи t-SNE

```

print("\n--- Запуск UMAP ---")
umap_params = [
    {'n_neighbors': 5, 'min_dist': 0.1, 'desc': 'Локальная структура, плотные кластеры'},
    {'n_neighbors': 15, 'min_dist': 0.8, 'desc': 'Локальная структура, разреженные кластеры'},
    {'n_neighbors': 50, 'min_dist': 0.1, 'desc': 'Глобальная структура, плотные кластеры'},
    {'n_neighbors': 50, 'min_dist': 0.8, 'desc': 'Глобальная структура, разреженные кластеры'},
]

umap_total_start_time = time.time()

for params in umap_params:
    n_neighbors = params['n_neighbors']
    min_dist = params['min_dist']
    print(f"Вычисление UMAP с n_neighbors={n_neighbors} и min_dist={min_dist}...")
    start_time = time.time()

    reducer = umap.UMAP(
        n_neighbors=n_neighbors,
        min_dist=min_dist,
        n_components=2,
        random_state=42
    )

    X_umap = reducer.fit_transform(X_full)
    end_time = time.time()
    print(f"UMAP с n_neighbors={n_neighbors} и min_dist={min_dist} завершен за {end_time - start_time:.2f} секунд.")

    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y_full, cmap=plt.get_cmap(name="jet", lut=10), s=1)
    plt.title(f'UMAP: n_neighbors={n_neighbors}, min_dist={min_dist} ({params["desc"]}) (на полных данных)')
    plt.xlabel('Компонента 1')
    plt.ylabel('Компонента 2')
    handles, _ = scatter.legend_elements(num=10)
    plt.legend(*args: handles, class_names, title="Классы")
    plt.show()

umap_total_end_time = time.time()
umap_duration = umap_total_end_time - umap_total_start_time
print(f"Общее время выполнения UMAP: {umap_duration:.2f} секунд")

print("\n--- Итоги производительности ---")
print(f"Время выполнения t-SNE (4 запуска): {tsne_duration:.2f} секунд")
print(f"Время выполнения UMAP (4 запуска): {umap_duration:.2f} секунд")

if umap_duration > 0 and tsne_duration > 0:
    if umap_duration < tsne_duration:
        print(f"UMAP был быстрее t-SNE в {tsne_duration / umap_duration:.2f} раз.")
    else:
        print(f"t-SNE был быстрее UMAP в {umap_duration / tsne_duration:.2f} раз.")

```

Рисунок 10 – Визуализация данных при помощи UMAP и сравнение по времени выполнения

Визуализации, созданные при помощи t-SNE представлены на рисунках 11-14, при помощи UMAP – на рисунках 15-18.

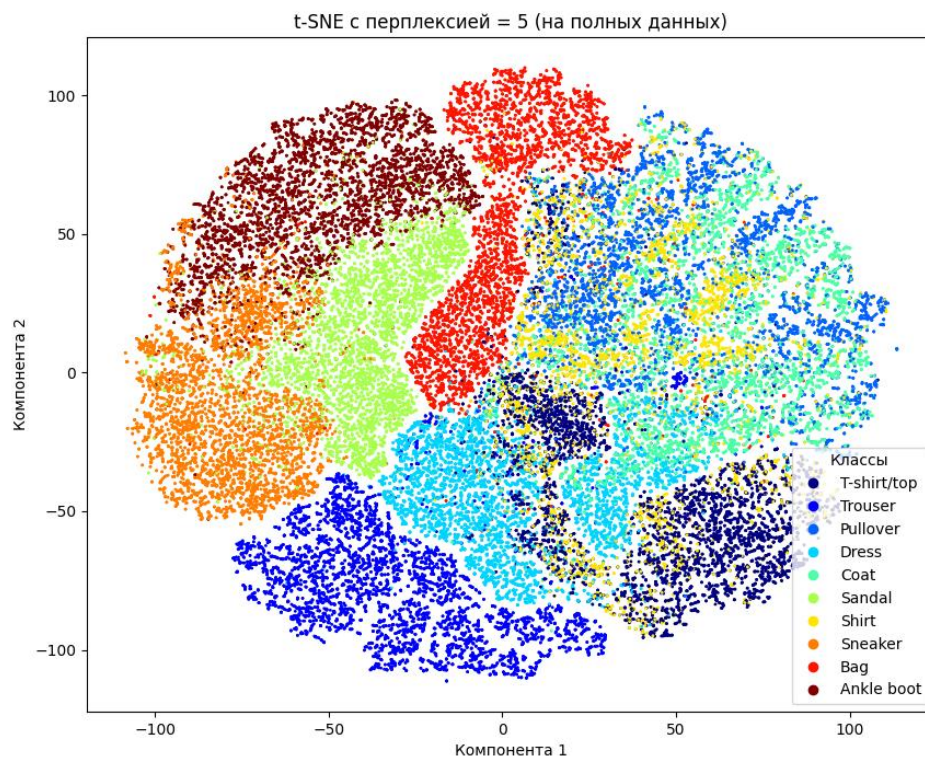


Рисунок 11 – Визуализация данных при помощи t-SNE (1/4)

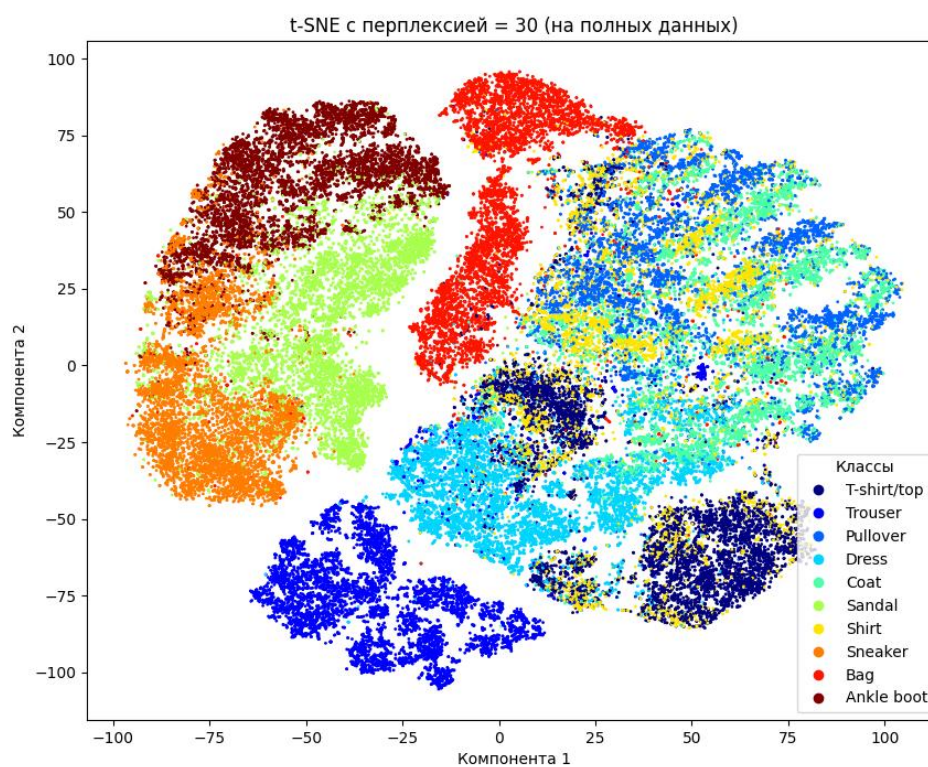


Рисунок 12 – Визуализация данных при помощи t-SNE (2/4)

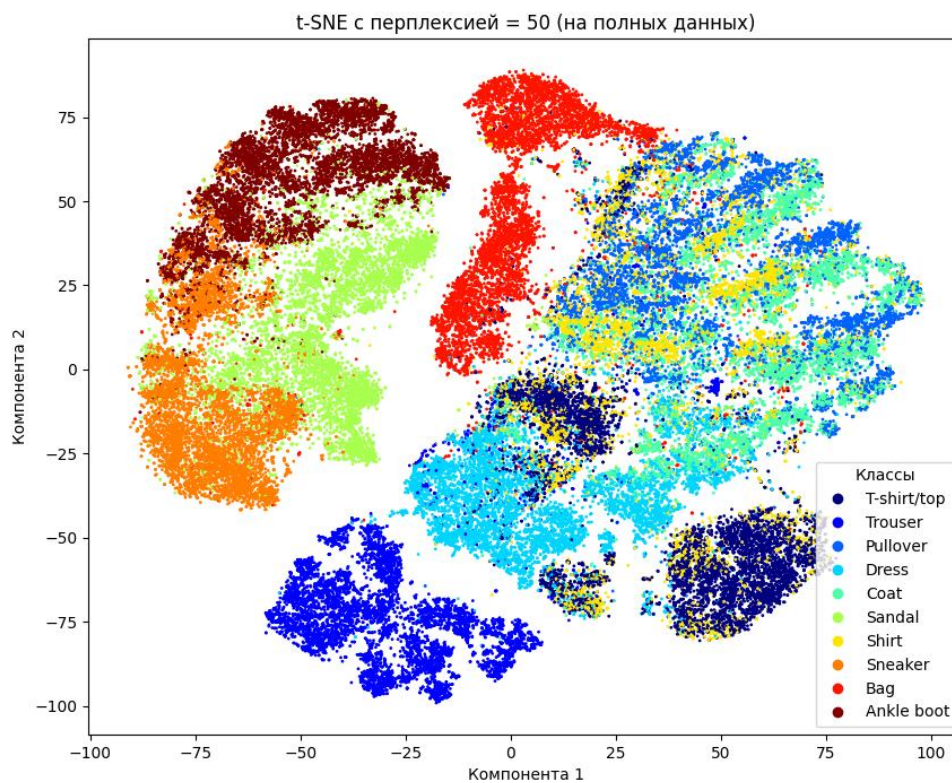


Рисунок 13 – Визуализация данных при помощи t-SNE (3/4)

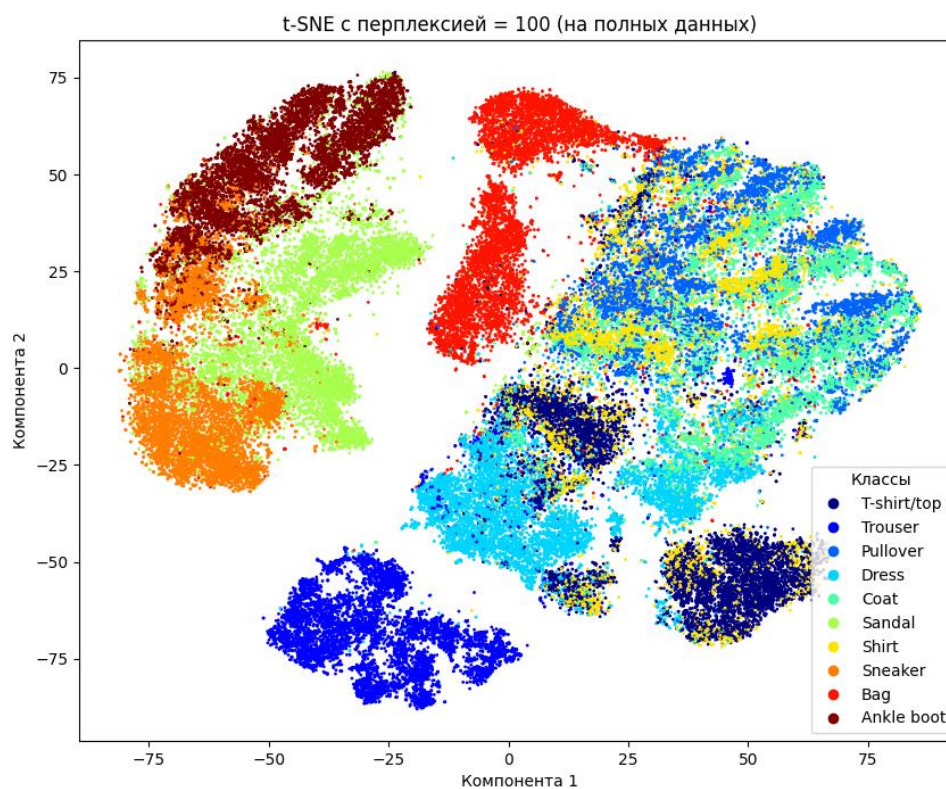


Рисунок 14 – Визуализация данных при помощи t-SNE (4/4)

UMAP: $n_neighbors=5$, $min_dist=0.1$ (Локальная структура, плотные кластеры) (на полных данных)

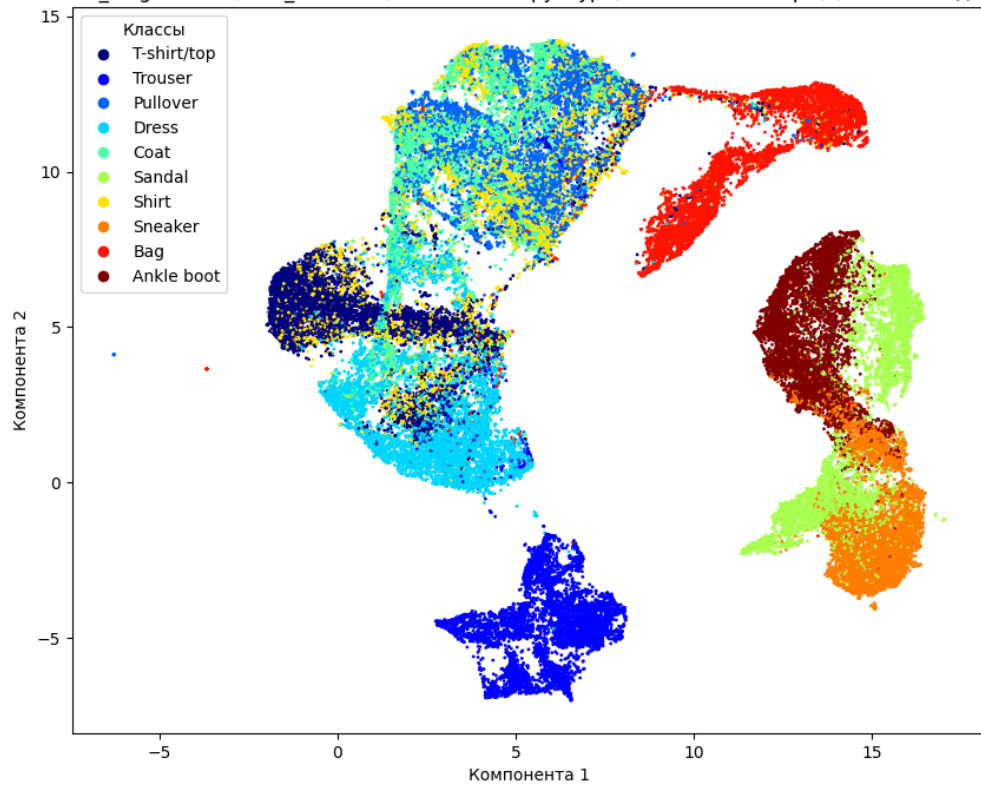


Рисунок 15 – Визуализация данных при помощи UMAP (1/4)

UMAP: $n_neighbors=15$, $min_dist=0.8$ (Локальная структура, разреженные кластеры) (на полных данных)

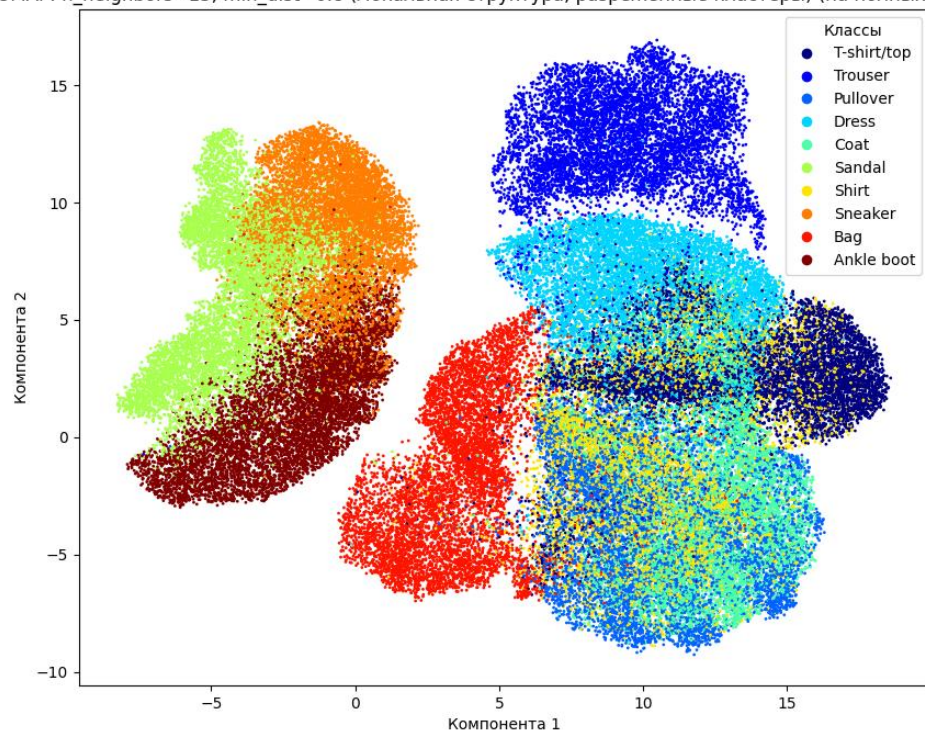


Рисунок 16 – Визуализация данных при помощи UMAP (2/4)

UMAP: n_neighbors=50, min_dist=0.1 (Глобальная структура, плотные кластеры) (на полных данных)

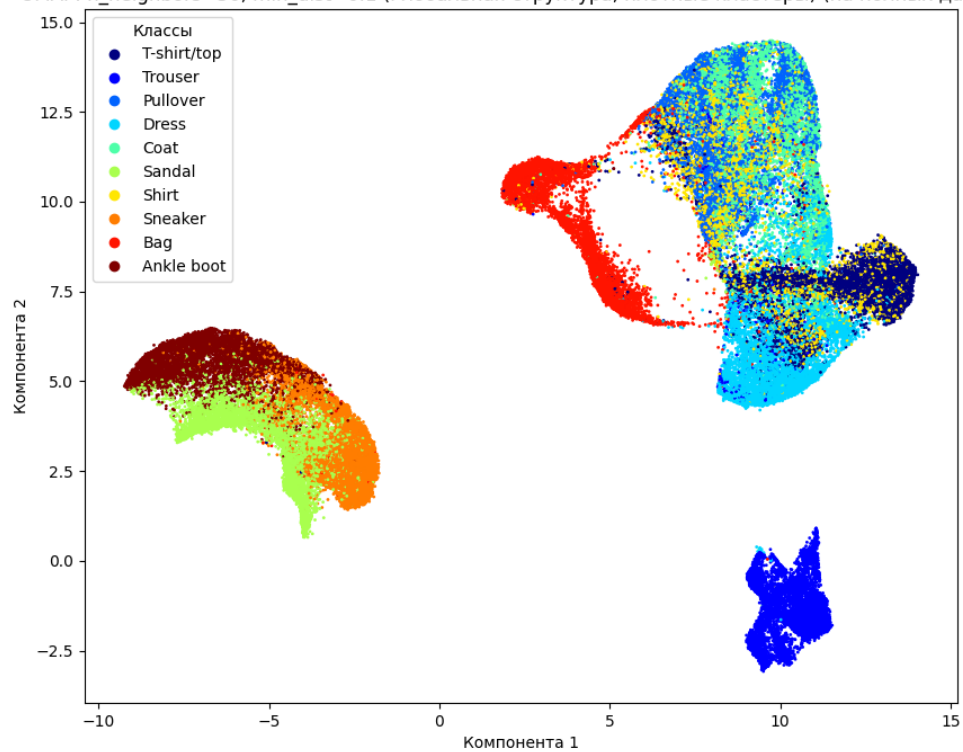


Рисунок 17 – Визуализация данных при помощи UMAP (3/4)

UMAP: n_neighbors=50, min_dist=0.8 (Глобальная структура, разреженные кластеры) (на полных данных)

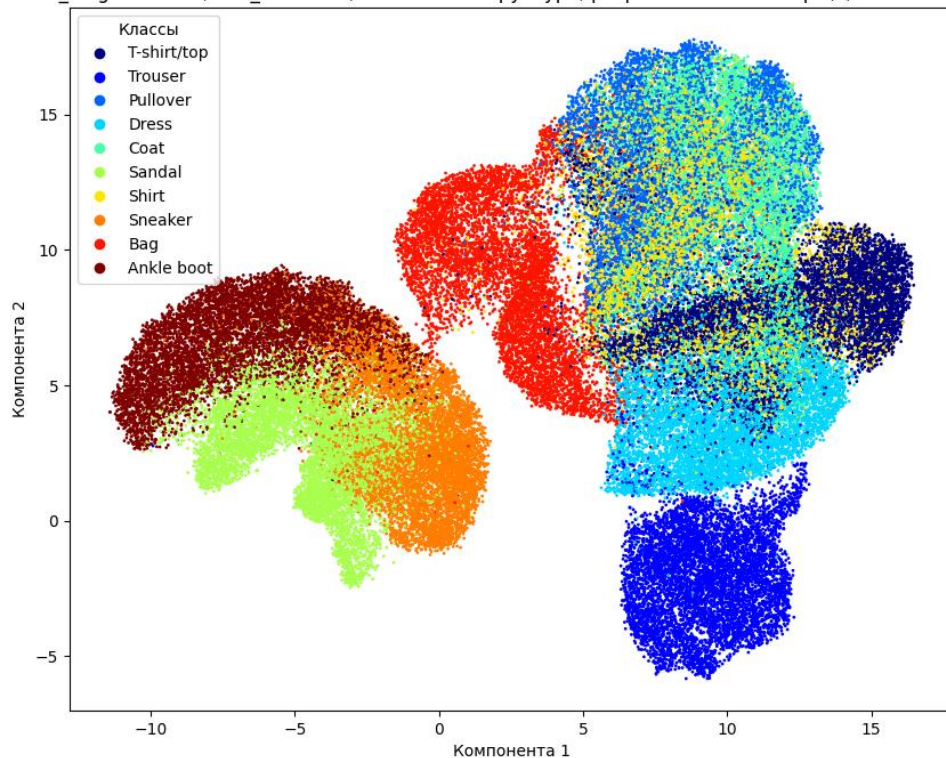
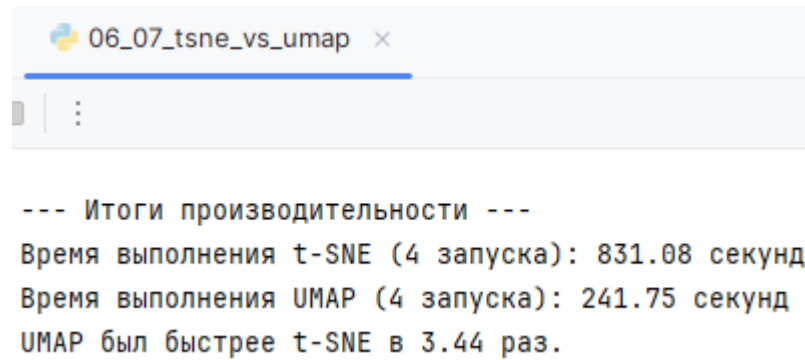


Рисунок 18 – Визуализация данных при помощи UMAP (4/4)

Сравнение двух способов по времени работы представлено на рисунке 19.



```
06_07_tsne_vs_umap x
|
:

--- Итоги производительности ---
Время выполнения t-SNE (4 запуска): 831.08 секунд
Время выполнения UMAP (4 запуска): 241.75 секунд
UMAP был быстрее t-SNE в 3.44 раз.
```

Рисунок 19 – Сравнение времени работы t-SNE и UMAP

ВЫВОД

В ходе выполненной практической работы проведено ознакомление с различными библиотеками визуализации данных (matplotlib, plotly, TSNE, UMAP) и особенностями работы с ними в среде программирования Python.