



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 № 6**

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

\_\_\_\_\_  
(подпись)

Ассистент *Тетерин Николай Николаевич*

\_\_\_\_\_  
(подпись)

Отчёт представлен «\_\_» ноября 2025 г.

Москва, 2025 г.

## **ЦЕЛЬ**

Ознакомиться с инструментами кластеризации данных на Python.

## ХОД РАБОТЫ

### Задание

1. Найти данные для кластеризации. Данные в группе не должны повторяться. Если признаки в данных имеют очень сильно разные масштабы, то необходимо данные предварительно нормализовать.
2. Провести кластеризацию данных с помощью алгоритма k-means. Использовать «правило локтя» и коэффициент силуэта для поиска оптимального количества кластеров.
3. Провести кластеризацию данных с помощью алгоритма иерархической кластеризации.
4. Провести кластеризацию данных с помощью алгоритма DBSCAN.
5. Визуализировать кластеризованные данные с помощью t-SNE или UMAP, если необходимо. Если данные трехмерные, то можно использовать трехмерный точечный график.

### Ход работы

В качестве данных был выбран датасет с результатами гонок Формулы 1 – <https://www.kaggle.com/datasets/ignale/formula-one-races-results-1950-2022>.

Код для объединения и преобразования начальных данных представлен на рисунках 1-2. Код для нормализации данных (Scale) и из вывода представлен на рисунке 3. Результат его работы – на рисунке 4.

```

RAW_DIR = Path(__file__).resolve().parent / 'data'
OUTPUT_PATH = RAW_DIR / 'f1_clustering.csv'

FEATURE_COLS = [
    'win_rate', 'podium_rate', 'avg_grid', 'avg_finish', 'best_finish', 'grid_vs_finish',
    'avg_championship_position_pct', 'title_rate'
]

def build_feature_table() -> pd.DataFrame: 1 usage new *
    results = pd.read_csv(RAW_DIR / 'results.csv')
    races = pd.read_csv(RAW_DIR / 'races.csv')
    drivers = pd.read_csv(RAW_DIR / 'drivers.csv')
    driver_standings = pd.read_csv(RAW_DIR / 'driver_standings.csv')
    constructor_standings = pd.read_csv(RAW_DIR / 'constructor_standings.csv')

    year_final_race = races.groupby('year')['raceId'].max().reset_index()
    year_final_race.columns = ['year', 'final_raceId']

    team_strength = (
        constructor_standings
        .merge(year_final_race, left_on='raceId', right_on='final_raceId')
        .groupby(['constructorId', 'year'])['position']
        .min()
        .reset_index()
    )
    team_strength.columns = ['constructorId', 'year', 'team_position']

    final_standings = (
        driver_standings
        .merge(year_final_race, left_on='raceId', right_on='final_raceId')[['driverId', 'year', 'position', 'points']]
    )

    drivers_per_season = (
        final_standings
        .groupby('year')['driverId']
        .nunique()
        .reset_index()
    )
    drivers_per_season.columns = ['year', 'total_drivers']

    final_standings = final_standings.merge(drivers_per_season, on='year', how='left')

    final_standings['championship_position_pct'] = (
        (final_standings['total_drivers'] - final_standings['position']) /
        (final_standings['total_drivers'] - 1) * 100
    ).clip(0, 100)

    avg_championship = (
        final_standings
        .groupby('driverId')['championship_position_pct']
        .mean()
        .reset_index()
    )
    avg_championship.columns = ['driverId', 'avg_championship_position_pct']

```

Рисунок 1 – Код для предобработки данных (1/2)

```

def build_feature_table() -> pd.DataFrame: 1 usage new *
    career_seasons = (
        final_standings
        .groupby('driverId')['year']
        .nunique()
        .reset_index()
    )
    career_seasons.columns = ['driverId', 'career_seasons']

    titles = (
        final_standings[final_standings['position'] == 1]
        .groupby('driverId')
        .size()
        .reset_index(name='total_titles')
    )

    merged = (
        results
        .merge(races[['raceId', 'year']], on='raceId', how='left')
        .merge(drivers[['driverId', 'driverRef', 'nationality']], on='driverId', how='left')
        .merge(team_strength, on=['constructorId', 'year'], how='left')
    )
    merged = merged[merged['positionOrder'] > 0]
    merged['team_position'] = merged['team_position'].fillna(merged['team_position'].median())

    features = (
        merged.groupby(['driverId', 'driverRef', 'nationality'])
        .agg(
            total_races=('raceId', 'nunique'),
            total_wins=('positionOrder', lambda x: (x == 1).sum()),
            total_podiums=('positionOrder', lambda x: (x <= 3).sum()),
            avg_grid=('grid', 'mean'),
            avg_finish=('positionOrder', 'mean'),
            best_finish=('positionOrder', 'min'),
            avg_team_position=('team_position', 'mean'),
        )
        .reset_index()
    )

    features = features.merge(avg_championship, on='driverId', how='left')
    features = features.merge(titles, on='driverId', how='left')
    features = features.merge(career_seasons, on='driverId', how='left')

    features['avg_championship_position_pct'] = features['avg_championship_position_pct'].fillna(0)
    features['total_titles'] = features['total_titles'].fillna(0).astype(int)
    features['career_seasons'] = features['career_seasons'].fillna(1).astype(int)
    features['win_rate'] = features['total_wins'] / features['total_races'] * 100
    features['podium_rate'] = features['total_podiums'] / features['total_races'] * 100
    features['grid_vs_finish'] = features['avg_finish'] - features['avg_grid']
    features['performance_vs_team'] = features['avg_team_position'] - features['avg_finish']
    features['title_rate'] = features['total_titles'] / features['career_seasons'] * 100

    FEATURE_COLS.append('performance_vs_team')
    FEATURE_COLS.append('avg_team_position')

    return features[features['total_races'] >= 10]

```

Рисунок 2 – Код для предобработки данных (2/2)

```

def scale_features(df: pd.DataFrame) -> pd.DataFrame: 1 usage new *
    filled = df.copy()
    for col in FEATURE_COLS:
        if col in filled.columns:
            filled[col] = filled[col].fillna(filled[col].median())

    scaler = StandardScaler()
    cols_to_scale = [c for c in FEATURE_COLS if c in filled.columns]
    scaled = scaler.fit_transform(filled[cols_to_scale])

    scaled_df = pd.DataFrame(
        scaled,
        columns=[f'{col}_scaled' for col in cols_to_scale],
        index=filled.index,
    )
    return pd.concat([filled, scaled_df], axis=1)

def main() -> None: 1 usage new *
    feature_df = build_feature_table()
    dataset = scale_features(feature_df)
    dataset.to_csv(OUTPUT_PATH, index=False)
    print(f'Сохранено {len(dataset)} гонщиков в {OUTPUT_PATH}')

    print("\nПример данных (топ-10 по title_rate):")
    preview_cols = ['driverRef', 'total_races', 'total_wins', 'total_titles', 'career_seasons', 'title_rate',
                    'win_rate', 'avg_championship_position_pct']
    print(dataset.nlargest(n=10, columns='title_rate')[preview_cols].to_string(index=False))

if __name__ == '__main__':
    main()

```

Рисунок 3 – Код для нормализации и вывода данных

Run 1\_data\_preprocessing x

C:\Users\Remsely\dev\mirea\python\big-data\.venv\Scripts\python.exe C:\Users\Remsely\dev\mirea\python\big-data\practice-6-clust  
Сохранено 374 гонщиков в C:\Users\Remsely\dev\mirea\python\big-data\practice-6-clustering\data\fl\_clustering.csv

Пример данных (топ-10 по title\_rate):

driverRef	total_races	total_wins	total_titles	career_seasons	title_rate	win_rate	avg_championship_position_pct
fangio	51	24	5	8	62.500000	47.058824	97.837393
max_verstappen	209	63	4	10	40.000000	30.143541	87.398771
hamilton	356	105	7	18	38.888889	29.494382	92.106051
michael_schumacher	308	91	7	19	36.842105	29.545455	84.262978
stewart	100	27	3	9	33.333333	27.000000	93.836442
ascari	32	13	2	6	33.333333	40.625000	85.160643
prost	202	51	4	13	30.769231	25.247525	93.330797
senna	162	41	3	11	27.272727	25.308642	86.760294
vettel	300	53	4	16	25.000000	17.666667	79.747867
lauda	174	25	3	13	23.076923	14.367816	78.051508

Process finished with exit code 0

Рисунок 4 – Результат выполнения предобработки данных

Код для выполнения задания 2 представлен на рисунках 5-6. Результат его работы – на рисунках 7-9. Построенные диаграммы – на рисунке 10.

Оптимальное кол-во кластеров по результатам замеров – 3, но для наших целей определения лучших гонщиков лучше возьмем 4 (уступает несильно).

```

DATA_PATH = Path(__file__).resolve().parent / 'data' / 'f1_clustering.csv'
SCALED_COLS = [
    'win_rate_scaled', 'podium_rate_scaled', 'avg_grid_scaled', 'avg_finish_scaled', 'best_finish_scaled',
    'grid_vs_finish_scaled', 'avg_championship_position_pct_scaled', 'title_rate_scaled', 'performance_vs_team_scaled',
    'avg_team_position_scaled'
]
K_RANGE = range(2, 11)
BEST_K = 4

def main() -> None:
    """usage new"""
    data = pd.read_csv(DATA_PATH)
    available_cols = [c for c in SCALED_COLS if c in data.columns]
    features = data[available_cols]

    inertias, silhouettes = [], []
    for k in K_RANGE:
        model = KMeans(n_clusters=k, random_state=42, n_init='auto')
        labels = model.fit_predict(features)
        inertias.append(model.inertia_)
        silhouettes.append(silhouette_score(features, labels))
        print(f'k={k}: силуэт={silhouettes[-1]:.3f}')

    model = KMeans(n_clusters=BEST_K, random_state=42, n_init='auto')
    data['cluster'] = model.fit_predict(features)

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))
    axes[0].plot(list(K_RANGE), inertias, marker='o')
    axes[0].set_xlabel('k')
    axes[0].set_ylabel('Inertia')
    axes[0].set_title('Правило локтя')

    axes[1].plot(list(K_RANGE), silhouettes, marker='o', color='green')
    axes[1].set_xlabel('k')
    axes[1].set_ylabel('Silhouette')
    axes[1].set_title('Коэффициент силуэта')
    plt.tight_layout()
    plt.savefig('visualization/kmeans_elbow.png')
    plt.show()

    print("\n" + "=" * 70)
    print("ОБЩАЯ СТАТИСТИКА")
    print("=" * 70)
    print(f"Всего гонщиков в выборке: {len(data)}")
    print(f"Количество кластеров: {BEST_K}")
    print(f"Коэффициент силуэта: {silhouettes[BEST_K - 2]:.3f}")
    print(f"\nРаспределение по кластерам:")
    print(data['cluster'].value_counts().sort_index().to_string())

    print("\n" + "=" * 70)
    print("СРЕДНИЕ ХАРАКТЕРИСТИКИ ПО КЛАСТЕРАМ")
    print("=" * 70)

    analysis_cols = ['total_races', 'total_wins', 'total_titles', 'title_rate', 'win_rate',
                     'podium_rate', 'avg_championship_position_pct', 'avg_finish', 'avg_team_position']
    cluster_stats = data.groupby('cluster')[analysis_cols].mean().round(2)
    print(cluster_stats.to_string())

```

Рисунок 5 – Код для кластеризации при помощи k-means (1/2)

```

for cluster_id in sorted(data['cluster'].unique()):
    cluster_data = data[data['cluster'] == cluster_id].copy()

    print(f"\n{'=' * 70}")
    print(f"КЛАСТЕР {cluster_id} – Гонщики ({len(cluster_data)} всего)")
    print(f"{'=' * 70}")

    cluster_sorted = cluster_data.sort_values('total_wins', ascending=False)

    if len(cluster_sorted) <= 50:
        print(cluster_sorted[display_cols].to_string(index=False))
    else:
        print(f"Топ-50 по победам:")
        print(cluster_sorted.head(50)[display_cols].to_string(index=False))

if __name__ == '__main__':
    main()

```

Рисунок 6 – Код для кластеризации при помощи k-means (2/2)

Run 2\_kmeans

```

C:\Users\Remsely\dev\mirea\python\big-data\.venv\Scripts\python.exe C:\Users\Remsely\dev\mirea\python\big-data\practice-6-clustering\2_kmeans.py
k=2: силуэт=0.346
k=3: силуэт=0.384
k=4: силуэт=0.349
k=5: силуэт=0.287
k=6: силуэт=0.295
k=7: силуэт=0.297
k=8: силуэт=0.302
k=9: силуэт=0.286
k=10: силуэт=0.251

=====
ОБЩАЯ СТАТИСТИКА
=====
Всего гонщиков в выборке: 374
Количество кластеров: 4
Коэффициент силуэта: 0.349

Распределение по кластерам:
cluster
0      19
1     139
2     204
3      12

=====
СРЕДНИЕ ХАРАКТЕРИСТИКИ ПО КЛАСТЕРАМ
=====

```

cluster	total_races	total_wins	total_titles	title_rate	win_rate	podium_rate	avg_championship_position_pct	avg_finish	avg_team_position
0	26.37	0.00	0.00	0.00	0.00	0.00	17.47	26.95	14.10
1	99.34	4.22	0.22	1.85	3.33	14.58	68.20	11.05	4.97
2	43.63	0.04	0.00	0.00	0.11	0.86	32.47	16.06	8.98
3	166.67	43.58	3.75	32.49	27.99	50.19	88.96	7.07	3.47

Рисунок 7 – Статистика по кластеризации при помощи k-means

Run 2\_kmeans x

```

=====
КЛАСТЕР 3 – Гонщики (12 всего)
=====
driverRef  total_races  total_wins  total_titles  title_rate  win_rate  avg_championship_position_pct  avg_team_position
-----
hamilton      356         105         7      38.888889  29.494382          92.106051          2.283708
michael_schumacher  308         91         7      36.842105  29.545455          84.262978          2.285714
max_verstappen  209         63         4      40.000000  30.143541          87.398771          2.751196
vettel        300         53         4      25.000000  17.666667          79.747867          3.096667
prost         202         51         4      30.769231  25.247525          93.330797          2.282178
senna         162         41         3      27.272727  25.308642          86.760294          2.444444
stewart       100         27         3      33.333333  27.000000          93.836442          2.850000
lauda         174         25         3      23.076923  14.367816          78.051508          3.132184
clark         72          25         2      22.222222  34.722222          92.156842          2.547945
fangio        51          24         5      62.500000  47.058824          97.837393          5.965517
ascari        32          13         2      33.333333  40.625000          85.160643          6.000000
farina        34           5         1      16.666667  14.705882          96.907317          6.000000
=====
Process finished with exit code 0

```

Рисунок 8 – Кластер «лучших» в истории Формулы-1 согласно нашей модели

Run 2\_kmeans x

```

=====
КЛАСТЕР 1 – Гонщики (139 всего)
=====
Топ-50 по победам:
driverRef  total_races  total_wins  total_titles  title_rate  win_rate  avg_championship_position_pct  avg_team_position
-----
alonso      404         32         2      9.523810  7.920792          72.056604          5.027228
mansell     192         31         1      6.666667  16.145833          75.310817          3.541667
rosberg     206         23         1      9.090909  11.165049          73.865079          4.000000
piquet      207         23         3      21.428571  11.111111          84.546729          3.917874
damon_hill  122         22         1      12.500000  18.032787          72.225644          3.688525
raikkonen   352         21         1      5.263158  5.965909          75.093989          3.815341
hakkinen    165         20         2      18.181818  12.121212          82.697903          3.648485
moss        67          16         0      0.000000  23.880597          86.107590          4.000000
button      309         15         1      5.555556  4.854369          61.806881          4.773463
hill        179         14         2      11.111111  7.821229          78.104005          4.983240
emerson_fittipaldi  149         14         2      18.181818  9.395973          82.118424          5.332215
jack_brabham  128         14         3      18.750000  10.937500          76.707262          3.310078
coulthard   247         13         0      0.000000  5.263158          75.151216          3.842105
mario_andretti  129         12         1      7.142857  9.302326          64.747151          4.713178
jones       117         12         1      10.000000  10.256410          67.226832          6.042735
reutemann   146         12         0      0.000000  8.219178          85.061808          3.479452
massa       271         11         0      0.000000  4.059041          70.309347          3.601476
barrichello  326         11         0      0.000000  3.374233          68.838972          4.898773
villeneuve  165         11         1      9.090909  6.666667          57.301189          5.309091
bottas      247         10         0      0.000000  4.048583          67.893676          4.295547
peterson    123         10         0      0.000000  8.130081          82.091175          4.455285
hunt        93          10         1      14.285714  10.752688          79.707146          5.387097
scheckter   113         10         1      11.111111  8.849558          76.066406          4.283186
berger      210         10         0      0.000000  4.761905          80.227261          3.476190
webber      217         9          0      0.000000  4.147465          66.987439          4.387097
ickx        121         8          0      0.000000  6.611570          73.482353          5.347107
hulme       112         8          1      10.000000  7.142857          89.121190          3.169643
leclerc     149         8          0      0.000000  5.369128          75.247039          3.597315
ricciardo   257         8          0      0.000000  3.112840          56.529325          5.136187
montoya     95          7          0      0.000000  7.368421          84.442056          2.663158
arnoux      164         7          0      0.000000  4.268293          65.715755          6.939024
laffite     180         6          0      0.000000  3.333333          75.782697          5.722222
ralf_schumacher  180         6          0      0.000000  3.333333          71.347756          4.027778
brooks      39          6          0      0.000000  15.384615          82.117900          3.560976
perez       283         6          0      0.000000  2.120141          69.686420          4.339223
rindt       62          6          1      14.285714  9.677419          75.952316          4.016129
surtees     112         6          1      7.692308  5.357143          79.111683          4.553571
gilles_villeneuve  68          6          0      0.000000  8.823529          72.807473          4.000000
patrese     257         6          0      0.000000  2.334630          72.507642          6.089494
watson      154         5          0      0.000000  3.246753          71.867328          5.818182
alboreto    215         5          0      0.000000  2.325581          61.946180          7.400000
keke_rosberg  128         5          1      11.111111  3.906250          68.365783          7.148438
regazzoni   138         5          0      0.000000  3.623188          79.857947          4.811594
norris      128         4          0      0.000000  3.125000          71.546001          3.476562
sainz       208         4          0      0.000000  1.923077          67.075208          4.120192
gurney      87          4          0      0.000000  4.597701          78.120139          5.241379
irvine      147         4          0      0.000000  2.721088          64.360173          4.775510
mclaren     103         4          0      0.000000  3.883495          83.755190          4.417476
herbert     165         3          0      0.000000  1.818182          51.651041          6.212121
boutsen     164         3          0      0.000000  1.829268          60.881927          7.426829
=====

```

Рисунок 9 – Кластер «отличных гонщиков, но не легенд» по нашей модели

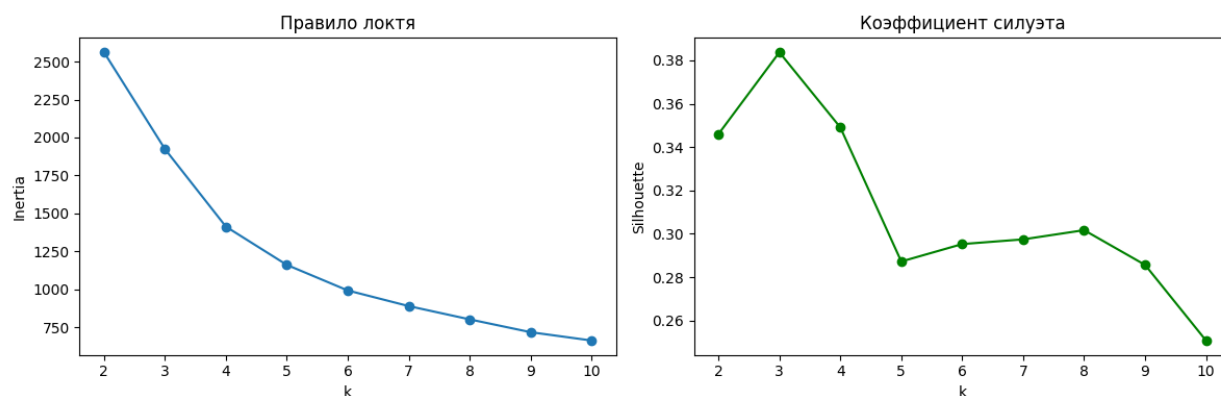


Рисунок 10 – Построенные графики

#### Выводы:

- Кластеризация выделила небольшую элитную группу – кластер 3 (12 гонщиков) с лучшими показателями, чем у остальных. В этот кластер вошли: Льюис Хэмилтон, Майкл Шумахер, Макс Ферстаппен, Себастьян Феттель, Ален Прост, Айртон Сenna, Джеки Стюарт, Ники Лауда, Джим Кларк, Хуан-Мануэль Фанхио, Альберто Ассари и Джузеппе Фарина. Высокие значения числа побед, титулов и стабильности выступлений у этих пилотов подтверждают их статус как наиболее успешных и доминирующих фигур в истории Формулы-1.

- Оптимальным числом кластеров является  $k = 3$ . При этом значении наблюдается максимальный коэффициент силуэта, что говорит о наилучшем разделении и компактности кластеров. Одновременно на графике правила локтя видно, что после  $k = 3-4$  снижение ошибки становится гораздо менее выраженным, то есть добавление новых кластеров почти не улучшает качество разбиения. Для наших целей правильной было взять 4 кластера.

Код для выполнения задания 3 представлен на рисунках 11-12. Результат его работы – на рисунках 13-15. Построенные диаграммы – на рисунках 16-17.

```

DATA_PATH = Path(__file__).resolve().parent / 'data' / 'f1_clustering.csv'
SCALED_COLS = [
    'win_rate_scaled', 'podium_rate_scaled', 'avg_grid_scaled', 'avg_finish_scaled', 'best_finish_scaled',
    'grid_vs_finish_scaled', 'avg_championship_position_pct_scaled', 'title_rate_scaled', 'performance_vs_team_scaled',
    'avg_team_position_scaled'
]
N_CLUSTERS = 4

def main() -> None:
    """usage: new"""
    data = pd.read_csv(DATA_PATH)
    available_cols = [c for c in SCALED_COLS if c in data.columns]
    features = data[available_cols]

    linkage_matrix = linkage(features, method='ward')
    plt.figure(figsize=(12, 5))
    dendrogram(linkage_matrix, truncate_mode='lastp', p=40)
    plt.title('Дендрограмма (метод Ward)')
    plt.xlabel('Объекты')
    plt.ylabel('Расстояние')
    plt.tight_layout()
    plt.savefig('visualization/hierarchical_dendrogram.png')
    plt.show()

    model = AgglomerativeClustering(n_clusters=N_CLUSTERS, linkage='ward')
    data['cluster'] = model.fit_predict(features)

    sil_score = silhouette_score(features, data['cluster'])

    print("\n" + "=" * 70)
    print("ОБЩАЯ СТАТИСТИКА (Иерархическая кластеризация)")
    print("=" * 70)
    print(f"Всего гонщиков в выборке: {len(data)}")
    print(f"Количество кластеров: {N_CLUSTERS}")
    print(f"Метод связи: Ward")
    print(f"Коэффициент силуэта: {sil_score:.3f}")
    print(f"Распределение по кластерам:")
    print(data['cluster'].value_counts().sort_index().to_string())

    print("\n" + "=" * 70)
    print("СРЕДНИЕ ХАРАКТЕРИСТИКИ ПО КЛАСТЕРАМ")
    print("=" * 70)

    analysis_cols = ['total_races', 'total_wins', 'total_titles', 'title_rate', 'win_rate',
                     'podium_rate', 'avg_championship_position_pct', 'avg_finish', 'avg_team_position']
    cluster_stats = data.groupby('cluster')[analysis_cols].mean().round(2)
    print(cluster_stats.to_string())

```

Рисунок 11 – Код для иерархической кластеризации данных (1/2)

```

display_cols = ['driverRef', 'total_races', 'total_wins', 'total_titles',
                'title_rate', 'win_rate', 'avg_championship_position_pct', 'avg_team_position']

for cluster_id in sorted(data['cluster'].unique()):
    cluster_data = data[data['cluster'] == cluster_id].copy()

    print(f"\n{'=' * 70}")
    print(f"КЛАСТЕР {cluster_id} – Гонщики ({len(cluster_data)} всего)")
    print(f"{'=' * 70}")

    cluster_sorted = cluster_data.sort_values('total_wins', ascending=False)

    if len(cluster_sorted) <= 50:
        print(cluster_sorted[display_cols].to_string(index=False))
    else:
        print(f"Топ-50 по победам:")
        print(cluster_sorted.head(50)[display_cols].to_string(index=False))

plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    data['avg_championship_position_pct'],
    data['win_rate'],
    c=data['cluster'],
    cmap='tab10',
    alpha=0.7
)
plt.xlabel('Средняя позиция в чемпионате (%)')
plt.ylabel('Процент побед (%)')
plt.title('Иерархическая кластеризация (Ward)')
plt.colorbar(scatter, label='Кластер')
plt.tight_layout()
plt.savefig('visualization/hierarchical_clusters.png')
plt.show()

if __name__ == '__main__':
    main()

```

Рисунок 12 – Код для иерархической кластеризации данных (2/2)

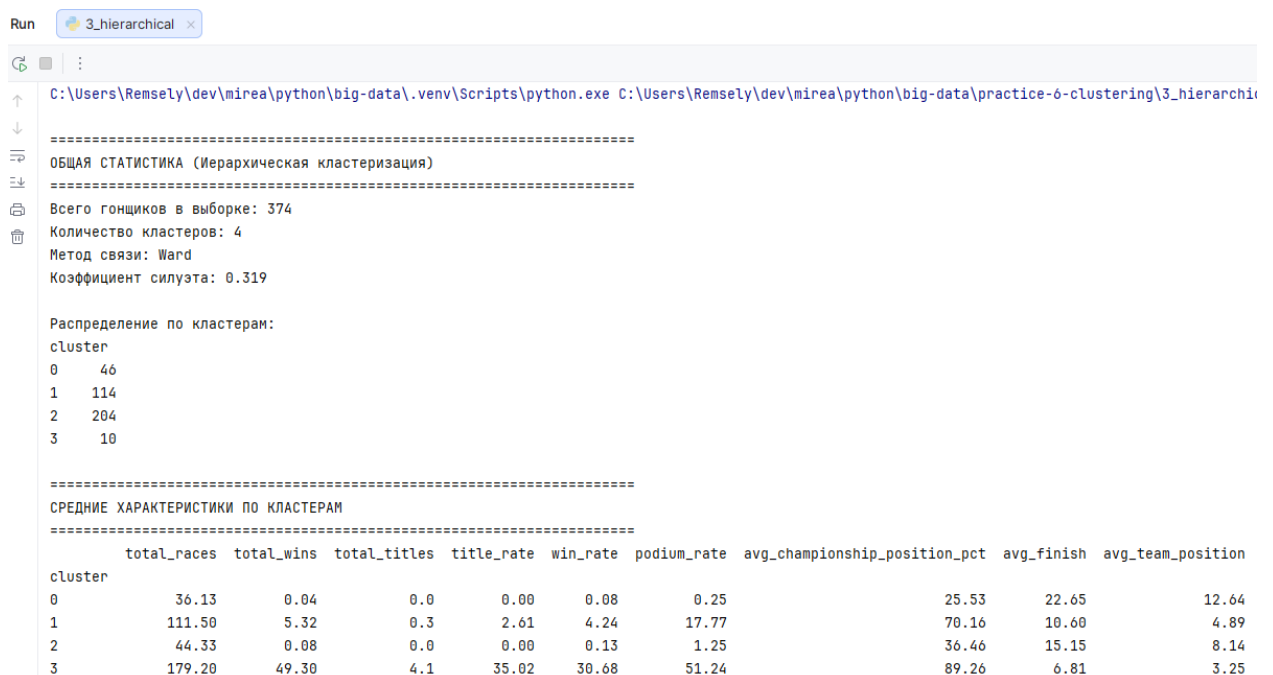


Рисунок 13 – Статистика по иерархической кластеризации

Run 3\_hierarchical

```

=====
КЛАСТЕР 3 – Гонщики (10 всего)
=====

```

	driverRef	total_races	total_wins	total_titles	title_rate	win_rate	avg_championship_position_pct	avg_team_position
	hamilton	356	105	7	38.888889	29.494382	92.106051	2.283708
	michael_schumacher	308	91	7	36.842105	29.545455	84.262978	2.285714
	max_verstappen	209	63	4	40.000000	30.143541	87.398771	2.751196
	vettel	300	53	4	25.000000	17.666667	79.747867	3.096667
	prost	202	51	4	30.769231	25.247525	93.330797	2.282178
	senna	162	41	3	27.272727	25.308642	86.760294	2.444444
	stewart	100	27	3	33.333333	27.000000	93.836442	2.850000
	clark	72	25	2	22.222222	34.722222	92.156842	2.547945
	fangio	51	24	5	62.500000	47.058824	97.837393	5.965517
	ascari	32	13	2	33.333333	40.625000	85.160643	6.000000

Рисунок 14 – Кластер «лучших» в истории Формулы-1 по иерархической кластеризации

Run 3\_hierarchical

```

=====
КЛАСТЕР 1 – Гонщики (114 всего)
=====

```

Топ-50 по победам:

	driverRef	total_races	total_wins	total_titles	title_rate	win_rate	avg_championship_position_pct	avg_team_position
	alonso	404	32	2	9.523810	7.920792	72.056604	5.027228
	mansell	192	31	1	6.666667	16.145833	75.310817	3.541667
	lauda	174	25	3	23.076923	14.367816	78.051508	3.132184
	piquet	207	23	3	21.428571	11.111111	84.546729	3.917874
	rosberg	206	23	1	9.090909	11.165049	73.865079	4.000000
	damon_hill	122	22	1	12.500000	18.032787	72.225644	3.688525
	raikkonen	352	21	1	5.263158	5.965909	75.093989	3.815341
	hakkinen	165	20	2	18.181818	12.121212	82.697903	3.648485
	moss	67	16	0	0.000000	23.880597	86.107590	4.000000
	button	309	15	1	5.555556	4.854369	61.806881	4.773463
	hill	179	14	2	11.111111	7.821229	78.104005	4.983240
	jack_brabham	128	14	3	18.750000	10.937500	76.707262	3.310078
	emerson_fittipaldi	149	14	2	18.181818	9.395973	82.118424	5.832215
	coulthard	247	13	0	0.000000	5.263158	75.151216	3.842105
	reutemann	146	12	0	0.000000	8.219178	85.061808	3.479452
	jones	117	12	1	10.000000	10.256410	67.226832	6.042735
	mario_andretti	129	12	1	7.142857	9.302326	64.747151	4.713178
	villeneuve	165	11	1	9.090909	6.666667	57.301189	5.309091
	massa	271	11	0	0.000000	4.059041	70.309347	3.601476
	barrichello	326	11	0	0.000000	3.374233	68.838972	4.898773
	berger	210	10	0	0.000000	4.761905	80.227261	3.476190
	peterson	123	10	0	0.000000	8.130081	82.091175	4.455285
	bottas	247	10	0	0.000000	4.048583	67.893676	4.295547
	scheckter	113	10	1	11.111111	8.849558	76.066406	4.283186
	hunt	93	10	1	14.285714	10.752688	79.707146	5.387097
	webber	217	9	0	0.000000	4.147465	66.987439	4.387097
	ickx	121	8	0	0.000000	6.611570	73.482353	5.347107
	hulme	112	8	1	10.000000	7.142857	89.121190	3.169643
	leclerc	149	8	0	0.000000	5.369128	75.247039	3.597315
	ricciardo	257	8	0	0.000000	3.112840	56.529325	5.136187
	arnoux	164	7	0	0.000000	4.268293	65.715755	6.939024
	montoya	95	7	0	0.000000	7.368421	84.442056	2.663158
	patrese	257	6	0	0.000000	2.334630	72.507642	6.089494
	ralf_schumacher	180	6	0	0.000000	3.333333	71.347756	4.027778
	rindt	62	6	1	14.285714	9.677419	75.952316	4.016129
	gilles_villeneuve	68	6	0	0.000000	8.823529	72.807473	4.000000
	laffite	180	6	0	0.000000	3.333333	75.782697	5.722222
	brooks	39	6	0	0.000000	15.384615	82.117900	3.560976
	surtees	112	6	1	7.692308	5.357143	79.111683	4.553571
	perez	283	6	0	0.000000	2.120141	69.686420	4.339223
	farina	34	5	1	16.666667	14.705882	96.907317	6.000000
	watson	154	5	0	0.000000	3.246753	71.867328	5.818182
	keke_rosberg	128	5	1	11.111111	3.906250	68.365783	7.148438
	regazzoni	138	5	0	0.000000	3.623188	79.857947	4.811594
	norris	128	4	0	0.000000	3.125000	71.546001	3.476562
	sainz	208	4	0	0.000000	1.923077	67.075208	4.120192

Рисунок 15 – Кластер «отличных гонщиков, но не легенд» по иерархической кластеризации

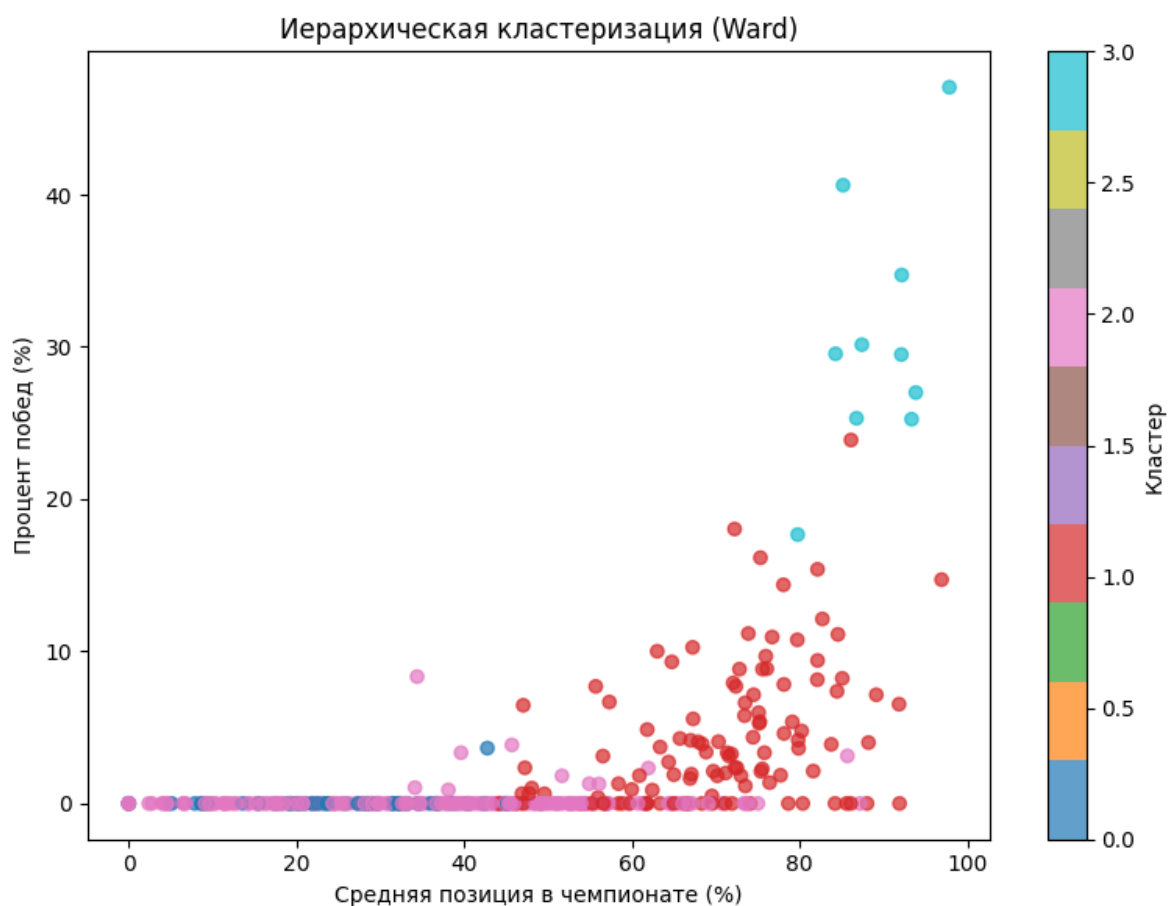


Рисунок 16 – Визуализация кластеризации относительно процента побед и относительной средней позиции в чемпионате

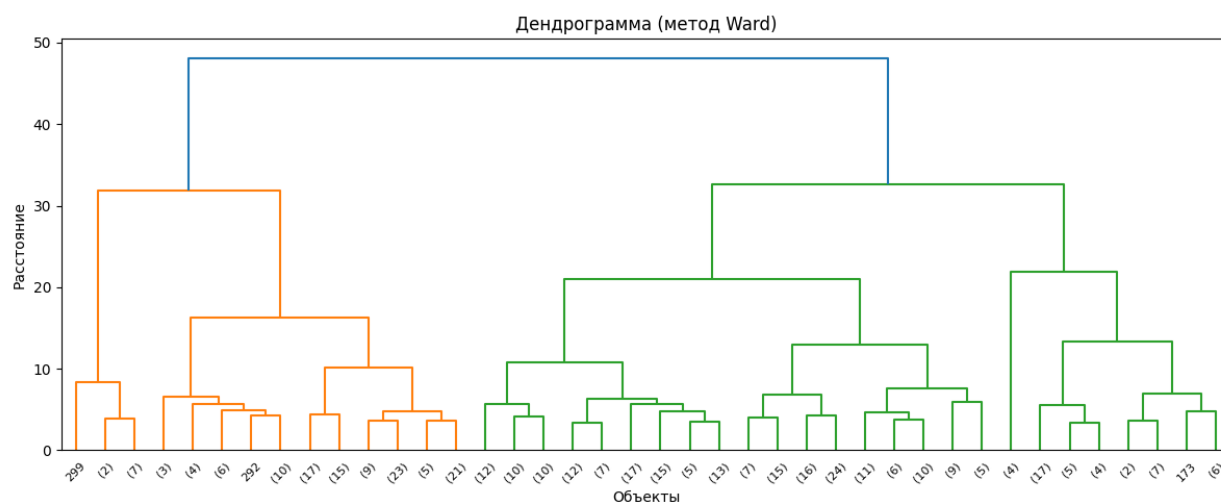


Рисунок 17 – Дендрограмма

### Выводы:

- Кластеризация выделила элитную группу – кластер 3 (10 гонщиков) с исключительными показателями. В этот кластер вошли: Льюис Хэмилтон, Майкл Шумахер, Макс Ферстаппен, Себастьян Феттель, Ален Прост, Айртон Сenna, Джеки Стюарт, Джим Кларк, Хуан-Мануэль Фанхио и Альберто

Аскари. Эти пилоты демонстрируют выдающиеся значения, что подтверждает их статус как величайших гонщиков в истории Формулы-1.

- Коэффициент силуэта составляет 0.319, что указывает на умеренное качество разделения кластеров. Дендрограмма демонстрирует чёткое отделение элитного кластера на высоком уровне расстояния (~48), что подтверждает значительное отличие лучших гонщиков от остальных.

- Оба алгоритма выделили практически идентичное ядро элитных пилотов. 10 из 10 гонщиков иерархического кластера 3 полностью совпадают с K-Means.

- K-Means дополнительно включил Ники Лауду и Джузеппе Фарину в элитный кластер. При иерархической кластеризации Лауда попал в кластер 1 (опытные гонщики), а Фарина – в кластер 3. Это объясняется особенностями метода Ward, который более чувствителен к межкластерным расстояниям.

- K-Means показал более высокое качество разделения (силуэт 0.349 против 0.319), однако разница незначительна (~9%), что говорит о сопоставимости результатов обоих методов.

Код для выполнения задания 4 представлен на рисунках 18-19. Результат его работы – на рисунках 20-22. Построенные диаграммы – на рисунках 23-24.

```

DATA_PATH = Path(__file__).resolve().parent / 'data' / 'f1_clustering.csv'
SCALED_COLS = [
    'win_rate_scaled', 'podium_rate_scaled', 'avg_grid_scaled', 'avg_finish_scaled', 'best_finish_scaled',
    'grid_vs_finish_scaled', 'avg_championship_position_pct_scaled', 'title_rate_scaled', 'performance_vs_team_scaled',
    'avg_team_position_scaled'
]
EPS = 1.5
MIN_SAMPLES = 5

def plot_k_distance(features, k=5): 1 usage new *
    neighbors = NearestNeighbors(n_neighbors=k)
    neighbors.fit(features)
    distances, _ = neighbors.kneighbors(features)
    distances = np.sort(distances[:, k - 1])

    plt.figure(figsize=(8, 4))
    plt.plot(distances)
    plt.xlabel('Точки (отсортированные)')
    plt.ylabel(f'{k}-расстояние')
    plt.title(f'График k-расстояний (k={k}) для подбора eps')
    plt.grid(visible=True, alpha=0.3)
    plt.tight_layout()
    plt.savefig('visualization/dbscan_k_distance.png')
    plt.show()

def main() -> None: 1 usage new *
    data = pd.read_csv(DATA_PATH)
    available_cols = [c for c in SCALED_COLS if c in data.columns]
    features = data[available_cols]

    plot_k_distance(features, k=MIN_SAMPLES)

    model = DBSCAN(eps=EPS, min_samples=MIN_SAMPLES)
    data['cluster'] = model.fit_predict(features)

    n_clusters = len(set(data['cluster'])) - (1 if -1 in data['cluster'].values else 0)
    n_noise = (data['cluster'] == -1).sum()

    print("\n" + "=" * 70)
    print("ОБЩАЯ СТАТИСТИКА (DBSCAN)")
    print("=" * 70)
    print(f"Всего гонщиков в выборке: {len(data)}")
    print(f"Параметры: eps={EPS}, min_samples={MIN_SAMPLES}")
    print(f"Количество кластеров: {n_clusters}")
    print(f"Количество шума (выбросов): {n_noise}")

    non_noise_mask = data['cluster'] != -1
    if n_clusters > 1 and non_noise_mask.sum() > 0:
        sil_score = silhouette_score(features[non_noise_mask], data.loc[non_noise_mask, 'cluster'])
        print(f"Коэффициент силуэта (без шума): {sil_score:.3f}")
    else:
        print("Силуэт не вычисляется: недостаточно кластеров.")

```

Рисунок 18 – Код для выполнения кластеризации при помощи DBSCAN (1/2)

```

def main() -> None: 1 usage new *

print(f"\nРаспределение по кластерам (-1 = шум):")
print(data['cluster'].value_counts().sort_index().to_string())
print("\n" + "=" * 70)
print("СРЕДНИЕ ХАРАКТЕРИСТИКИ ПО КЛАСТЕРАМ")
print("=" * 70)

analysis_cols = ['total_races', 'total_wins', 'total_titles', 'title_rate', 'win_rate',
                  'podium_rate', 'avg_championship_position_pct', 'avg_finish', 'avg_team_position']
cluster_stats = data.groupby('cluster')[analysis_cols].mean().round(2)
print(cluster_stats.to_string())

display_cols = ['driverRef', 'total_races', 'total_wins', 'total_titles',
                'title_rate', 'win_rate', 'avg_championship_position_pct', 'avg_team_position']

for cluster_id in sorted(data['cluster'].unique()):
    cluster_data = data[data['cluster'] == cluster_id].copy()

    cluster_name = "ШУМ (выбросы)" if cluster_id == -1 else f"КЛАСТЕР {cluster_id}"

    print(f"\n{'=' * 70}")
    print(f"{cluster_name} – Гонщики ({len(cluster_data)} всего)")
    print("=" * 70)

    cluster_sorted = cluster_data.sort_values('total_wins', ascending=False)

    if len(cluster_sorted) <= 50:
        print(cluster_sorted[display_cols].to_string(index=False))
    else:
        print(f"Топ-50 по победам:")
        print(cluster_sorted.head(50)[display_cols].to_string(index=False))

plt.figure(figsize=(8, 6))
plt.scatter(
    data['avg_championship_position_pct'],
    data['win_rate'],
    c=data['cluster'],
    cmap='Spectral',
    alpha=0.7
)
plt.xlabel('Средняя позиция в чемпионате (%)')
plt.ylabel('Процент побед (%)')
plt.title(f'DBSCAN (eps={EPS}, min_samples={MIN_SAMPLES})\nШум (серый) = {n_noise} точек')
plt.colorbar(label='Кластер (-1 = шум)')
plt.tight_layout()
plt.savefig('visualization/dbscan_clusters.png')
plt.show()

if __name__ == '__main__':
    main()

```

Рисунок 19 – Код для выполнения кластеризации при помощи DBSCAN (2/2)

```
Run 4_dbscan x
C:\Users\Rensely\dev\mirea\python\big-data\.venv\Scripts\python.exe C:\Users\Rensely\dev\mirea\python\big-data\practice-6-clustering\4_dbscan.py

=====
ОБЩАЯ СТАТИСТИКА (DBSCAN)
=====
Всего гонщиков в выборке: 374
Параметры: eps=1.5, min_samples=5
Количество кластеров: 2
Количество шума (выбросов): 26
Коэффициент силуэта (без шума): 0.649

Распределение по кластерам (-1 = шум):
cluster
-1      26
 0     343
 1       5

=====
СРЕДНИЕ ХАРАКТЕРИСТИКИ ПО КЛАСТЕРАМ
=====
total_races total_wins total_titles title_rate win_rate podium_rate avg_championship_position_pct avg_finish avg_team_position
cluster
-1          41.04         7.04         0.69         7.74         8.78         18.76          42.99         19.46          9.54
 0          66.96         1.74         0.09         0.78         1.31          6.00          46.46         14.20          7.50
 1         235.00        67.40         5.00        35.97        28.29        51.23          90.19          6.49          2.49
```

Рисунок 20 – Статистика по кластеризации DBSCAN

```
Run 4_dbscan x
=====
КЛАСТЕР 1 – Гонщики (5 всего)
=====
driverRef total_races total_wins total_titles title_rate win_rate avg_championship_position_pct avg_team_position
hamilton      356         105          7  38.888889 29.494382          92.106051          2.283708
michael_schumacher  308          91          7  36.842105 29.545455          84.262978          2.285714
max_verstappen   209          63          4  40.000000 30.143541          87.398771          2.751196
prost           202          51          4  30.769231 25.247525          93.330797          2.282178
stewart         100          27          3  33.333333 27.000000          93.836442          2.850000
```

Рисунок 21 – Кластер «лучших» в истории Формулы-1 согласно DBSCAN

Run 4\_dbscan x

ШУМ (выбросы) – Гонщики (26 всего)

driverRef	total_races	total_wins	total_titles	title_rate	win_rate	avg_championship_position_pct	avg_team_position
vettel	300	53	4	25.000000	17.666667	79.747867	3.096667
senna	162	41	3	27.272727	25.308642	86.760294	2.444444
clark	72	25	2	22.222222	34.722222	92.156842	2.547945
fangio	51	24	5	62.500000	47.058824	97.837393	5.965517
moss	67	16	0	0.000000	23.880597	86.107590	4.000000
ascari	32	13	2	33.333333	40.625000	85.160643	6.000000
farina	34	5	1	16.666667	14.705882	96.907317	6.000000
hawthorn	46	3	1	14.285714	6.521739	91.835176	5.166667
gonzalez	26	2	0	0.000000	7.692308	72.381132	5.896552
rathmann	10	1	0	0.000000	10.000000	63.004637	6.000000
weidler	10	0	0	0.000000	0.000000	8.695652	13.000000
langes	14	0	0	0.000000	0.000000	5.128205	17.000000
foitek	22	0	0	0.000000	0.000000	28.511706	16.227273
schneider	32	0	0	0.000000	0.000000	19.830652	17.906250
chaves	13	0	0	0.000000	0.000000	0.000000	19.000000
bailey	20	0	0	0.000000	0.000000	36.785714	8.200000
chiesa	10	0	0	0.000000	0.000000	0.000000	14.000000
poele	29	0	0	0.000000	0.000000	19.314516	14.896552
walker	11	0	0	0.000000	0.000000	20.134395	1.818182
charlton	13	0	0	0.000000	0.000000	16.019123	4.384615
ashley	11	0	0	0.000000	0.000000	22.550073	12.818182
gabbiani	17	0	0	0.000000	0.000000	15.467836	15.647059
larrauri	21	0	0	0.000000	0.000000	7.888199	17.714286
villota	14	0	0	0.000000	0.000000	13.583898	8.428571
raphanel	17	0	0	0.000000	0.000000	8.695652	15.823529
seidel	13	0	0	0.000000	0.000000	43.222030	4.076923

Рисунок 22 – Выбросы согласно DBSCAN

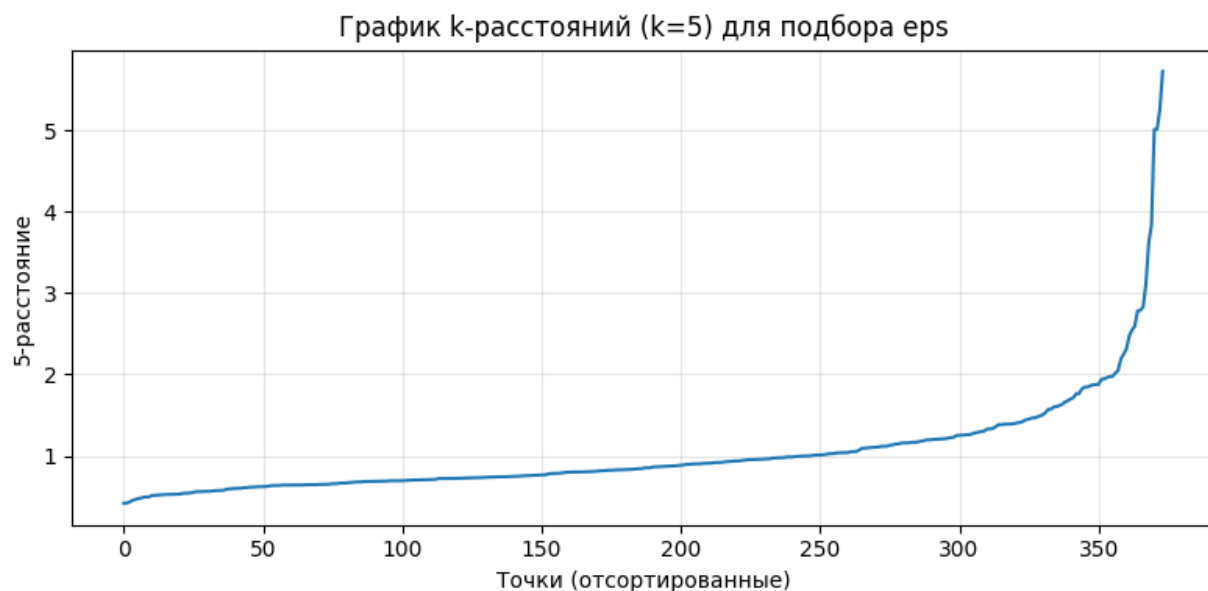


Рисунок 23 – График к-расстояний

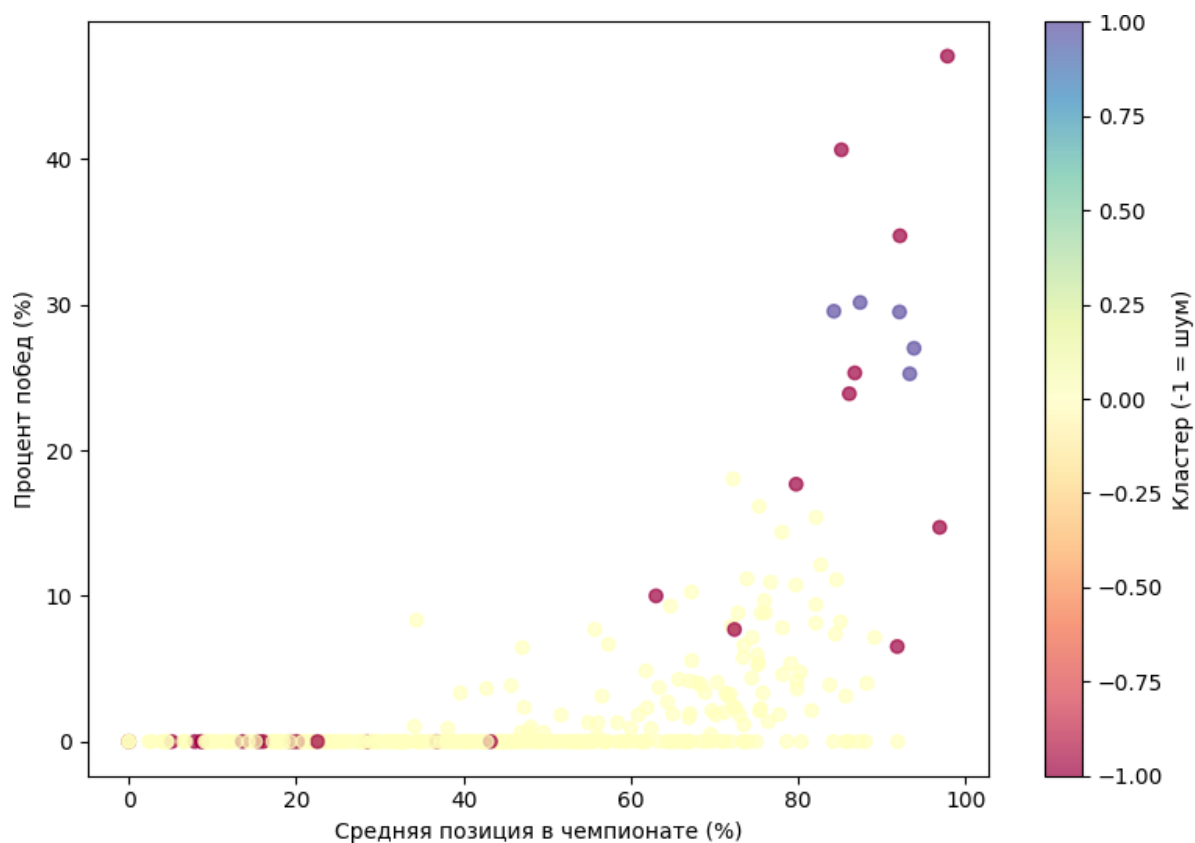


Рисунок 24 – Визуализация кластеризации DBSCAN

Выводы:

- DBSCAN выделил крайне компактную элитную группу — кластер 1 (всего 5 гонщиков) с абсолютно исключительными показателями. В этот кластер вошли только: Льюис Хэмилтон, Майкл Шумахер, Макс Ферстаппен, Ален Прост и Джеки Стюарт. Эти пилоты демонстрируют выдающиеся значения: в среднем 67.4 победы, 5 титулов чемпиона мира, win\_rate 28.29% и avg\_championship\_position\_pct 90.19%. Это «суперэлита» — гонщики, которые по совокупности признаков образуют плотную группу в пространстве данных.
- Кластер 0 (343 гонщика) — основная масса всех пилотов Формулы-1, объединённая в один большой кластер. Средние показатели: 67 гонок, 1.74 победы, win\_rate 1.31%. Сюда попали как опытные успешные гонщики (Алонсо, Мэнселл, Лауда, Пике), так и рядовые участники. DBSCAN не смог разделить эту группу на подкатегории из-за высокой плотности точек.
- Шум (выбросы) — 26 гонщиков — особая категория, которую DBSCAN классифицировал как аномалии. Интересно, что в шум попали две принципиально разные группы:

- Легенды: Феттель, Сенна, Кларк, Фанхио, Мосс, Аскари, Фарина, Хоторн — гонщики с выдающимися показателями, но нетипичным профилем (меньше гонок, выше win\_rate).

- Аутсайдеры: Weidler, Langes, Foitek, Schneider, Chaves — пилоты с очень низкими показателями и нетипичными характеристиками.

- График k-расстояний показывает резкий излом в районе  $\text{eps} \approx 1.5\text{--}2.0$ , что подтверждает корректность выбора параметра  $\text{eps}=1.5$ . Точки правее излома — это выбросы и элитные гонщики.

- DBSCAN показал себя наименее подходящим методом для данной задачи. Алгоритм, ориентированный на поиск областей высокой плотности, не смог адекватно разделить гонщиков по уровню успешности. Основная масса пилотов объединена в один кластер, а выдающиеся гонщики (Сенна, Фанхио, Кларк) ошибочно классифицированы как «шум» наравне с аутсайдерами. Для задач сегментации спортсменов по карьерным показателям методы K-Means и иерархическая кластеризация демонстрируют значительно более интерпретируемые и практически полезные результаты.

Код для выполнения задания 5 представлен на рисунках 25-26. Результат его работы — на рисунке 27. Построенные графики — на рисунках 28-29.

```

DATA_PATH = Path(__file__).resolve().parent / 'data' / 'f1_clustering.csv'
SCALED_COLS = [
    'win_rate_scaled', 'podium_rate_scaled', 'avg_grid_scaled', 'avg_finish_scaled', 'best_finish_scaled',
    'grid_vs_finish_scaled', 'avg_championship_position_pct_scaled', 'title_rate_scaled', 'performance_vs_team_scaled',
    'avg_team_position_scaled'
]
BEST_K = 4

def main() -> None: 1 usage new *
    data = pd.read_csv(DATA_PATH)
    available_cols = [c for c in SCALED_COLS if c in data.columns]
    features = data[available_cols]

    kmeans = KMeans(n_clusters=BEST_K, random_state=42, n_init='auto')
    data['cluster'] = kmeans.fit_predict(features)

    print("Вычисление t-SNE 2D...")
    tsne_2d = TSNE(
        n_components=2,
        random_state=42,
        init='pca',
        learning_rate='auto',
        perplexity=30
    )
    embedding_2d = tsne_2d.fit_transform(features)
    data['tsne_2d_1'] = embedding_2d[:, 0]
    data['tsne_2d_2'] = embedding_2d[:, 1]

    print("Вычисление t-SNE 3D...")
    tsne_3d = TSNE(
        n_components=3,
        random_state=42,
        init='pca',
        learning_rate='auto',
        perplexity=30
    )
    embedding_3d = tsne_3d.fit_transform(features)
    data['tsne_3d_1'] = embedding_3d[:, 0]
    data['tsne_3d_2'] = embedding_3d[:, 1]
    data['tsne_3d_3'] = embedding_3d[:, 2]

    print("\n" + "=" * 70)
    print("t-SNE ВИЗУАЛИЗАЦИЯ")
    print("=" * 70)
    print(f"Всего гонщиков: {len(data)}")
    print(f"Исходная размерность: {len(available_cols)}")
    print(f"Количество кластеров (K-Means): {BEST_K}")
    print(f"Perplexity: 30")

```

Рисунок 25 – Код для визуализации при помощи t-SNE (1/2)

```

plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    data['tsne_2d_1'],
    data['tsne_2d_2'],
    c=data['cluster'],
    cmap='tab10',
    alpha=0.7
)
plt.xlabel('t-SNE измерение 1')
plt.ylabel('t-SNE измерение 2')
plt.title('t-SNE 2D: раскраска по кластерам K-Means')
plt.colorbar(scatter, label='Кластер')
plt.tight_layout()
plt.savefig('visualization/tsne_2d.png')
plt.show()

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

scatter_3d = ax.scatter(
    data['tsne_3d_1'],
    data['tsne_3d_2'],
    data['tsne_3d_3'],
    c=data['cluster'],
    cmap='tab10',
    alpha=0.7
)
ax.set_xlabel('t-SNE 1')
ax.set_ylabel('t-SNE 2')
ax.set_zlabel('t-SNE 3')
ax.set_title('t-SNE 3D: раскраска по кластерам K-Means')
fig.colorbar(scatter_3d, shrink=0.5, label='Кластер')

plt.tight_layout()
plt.savefig('visualization/tsne_3d.png')
plt.show()

if __name__ == '__main__':
    main()

```

Рисунок 26 – Код для визуализации при помощи t-SNE (2/2)

Run

5\_tsne\_visualization

↑

↓

≡

≡

🖨

🗑

C:\Users\Remsely\dev\mirea\python\big-data\.venv\Scripts\python.exe C:\
Вычисление t-SNE 2D...
Вычисление t-SNE 3D...

=====
t-SNE ВИЗУАЛИЗАЦИЯ
=====
Всего гонщиков: 374
Исходная размерность: 10
Количество кластеров (K-Means): 4
Perplexity: 30

Process finished with exit code 0

Рисунок 27 – Вывод кода для визуализации

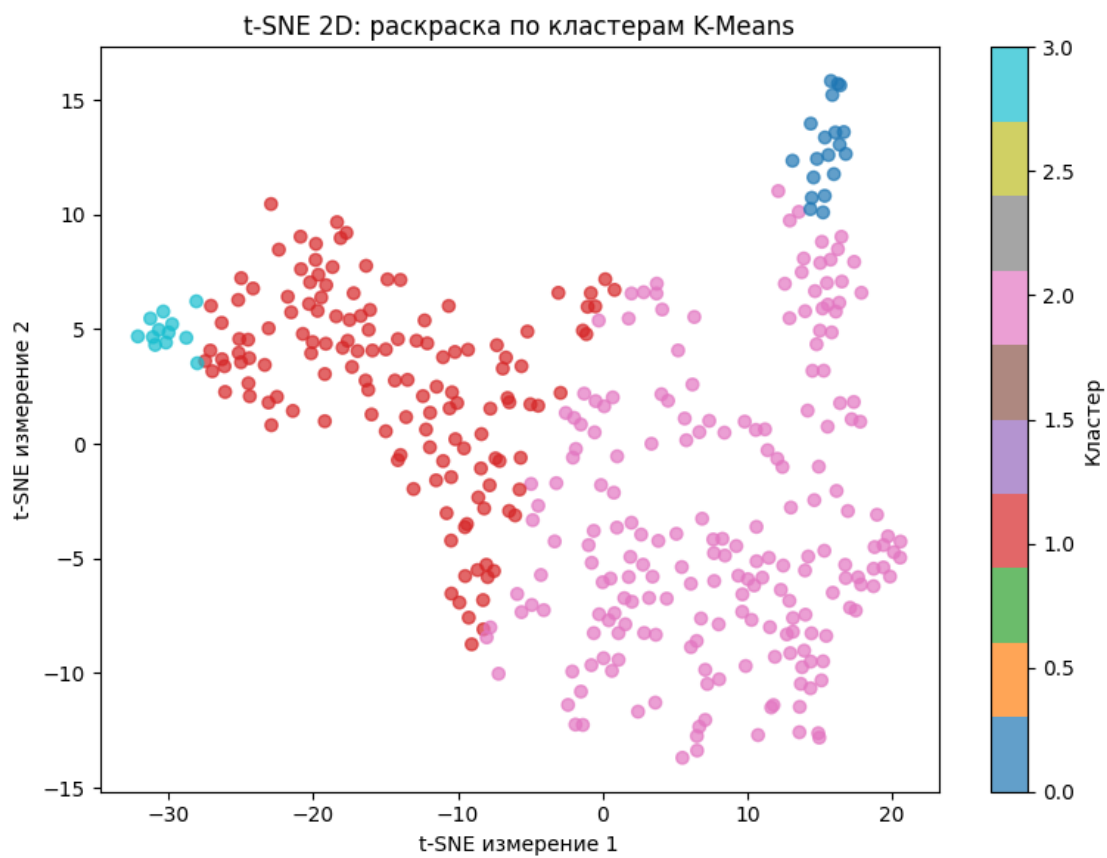


Рисунок 28 – 2D визуализация данных

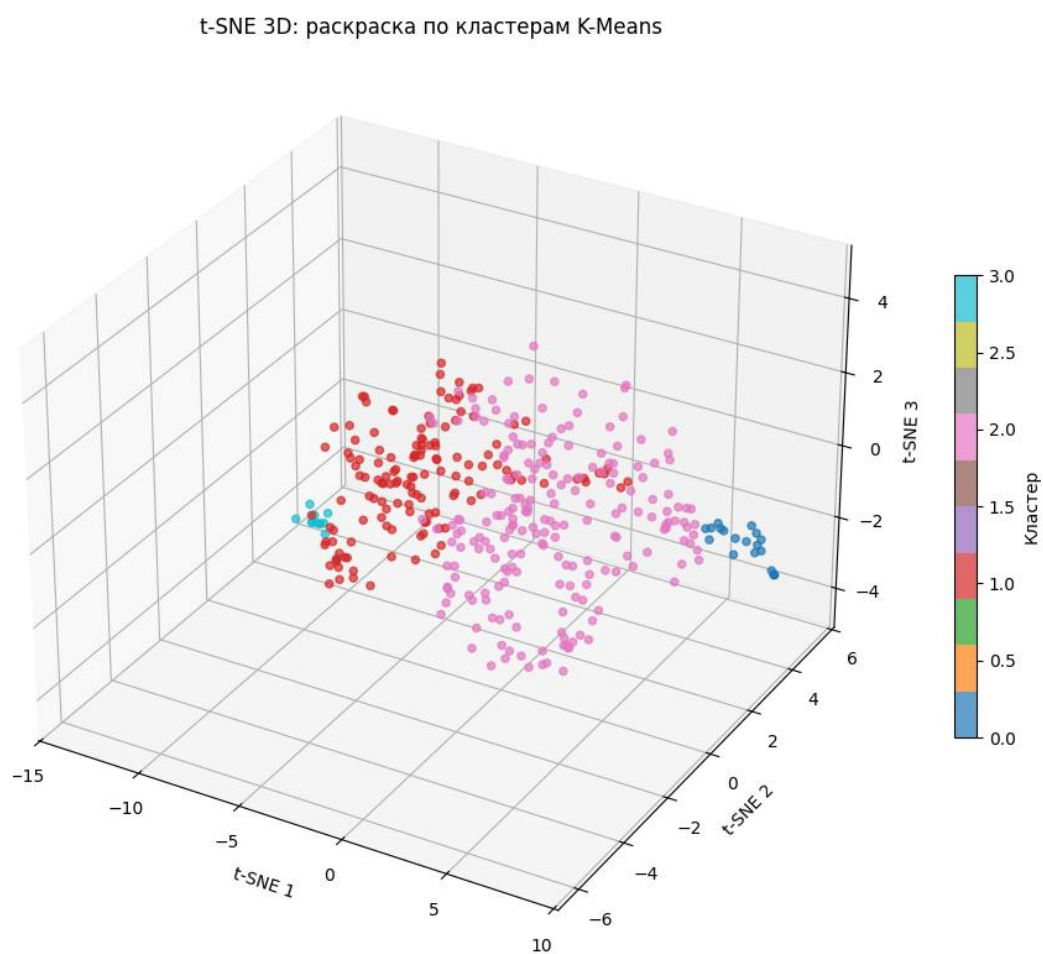


Рисунок 24 – 3D визуализация данных

## **ВЫВОД**

В ходе выполненной практической работы проведено ознакомление с инструментами кластеризации данных на Python.