

ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

специалист

должность, уч. степень, звание

подпись, дата

В. В. Боженко

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

ИССЛЕДОВАТЕЛЬСКИЙ АНАЛИЗ ДАННЫХ

По курсу: ВВЕДЕНИЕ В АНАЛИЗ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4117

подпись, дата

Д. С. Николаев

инициалы, фамилия

Санкт-Петербург 2023

Цель работы

Изучение связи между признаками двумерного набора данных, визуализация данных.

Индивидуальное задание

Вариант 2.

Задание 1 : Использовать seaborn. По группировке - CATEGORY и количество поездок каждого типа (по цели маршрута - PURPOSE) отфильтровать данные группировки по количеству поездок больше 2 и построить диаграмму.

Задание 2: Использовать pandas и plot. По сводной таблице (pivot_table) - отобразить среднее количество пройденных миль по каждой цели поездки (PURPOSE). Оставить только маркеры в виде ★ зеленого цвета размеров 18.

Задание 3: Использовать matplotlib. Построить круговую диаграмму, которая отображает процент по каждой цели поездки. Уберите из диаграммы количество поездок меньше 5.

Ход работы

Ссылка на [gitHub](#).

Проведена предварительная обработка данных как в 1 лабораторной работе. Загружен датасет через библиотеку Python - pandas. Используем ';' для разделение данных. Код расположен в листинге 1.

Листинг 1 - Код загрузка датасета.

```
import pandas
df = pandas.read_csv("drivers2.csv", sep = ',')
```

Затем было выведено на экран названия столбцов с помощью `df.columns`. Результат показан на рисунке 1.

```
Index(['START_DATE*', 'END_DATE*', 'CATEGORY*', 'START*', 'STOP*', 'MILES*',
      'PURPOSE*', 'time', 'speed', 'price'],
      dtype='object')
```

Рисунок 1 — Вывод названия столбцов

Проблема заключается в том, что названия не имеют единого формата. Для этого столбцы были переименованы в едином формате. Код продемонстрирован на листинге 2.

Листинг 2 - Код изменения названий столбцов.

```
df.columns = ['START_DATE', 'END_DATE', 'CATEGORY',  
'START', 'STOP', 'MILES', 'PURPOSE_ROUTE', 'TIME', 'SPEED',  
'PRICE']  
df.columns
```

Результат переименования столбцов продемонстрирован на рисунке 2.

```
Index(['START_DATE', 'END_DATE', 'CATEGORY', 'START', 'STOP', 'MILES',  
      'PURPOSE_ROUTE', 'TIME', 'SPEED', 'PRICE'],  
      dtype='object')
```

Рисунок 2 — Результат переименования столбцов

Также была проведена проверка на наличие пропусков, используя функцию `df.info()`. Результат данной функции представлен на рисунке 3.

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1099 entries, 0 to 1098  
Data columns (total 10 columns):  
#   Column                Non-Null Count  Dtype  
---  ---  
0   START_DATE            1099 non-null   object  
1   END_DATE              1099 non-null   object  
2   CATEGORY              1099 non-null   object  
3   START                 1099 non-null   object  
4   STOP                  1099 non-null   object  
5   MILES                 1099 non-null   float64  
6   PURPOSE_ROUTE         598 non-null    object  
7   TIME                  1099 non-null   float64  
8   SPEED                 1099 non-null   float64  
9   PRICE                 1099 non-null   float64  
dtypes: float64(4), object(6)  
memory usage: 86.0+ KB
```

Рисунок 3 — Результат функции info

По данным видно, что пропуски есть только в столбце PURPOSE_ROUTE. С помощью метода `fillna()` пустые строки были заменены на 'Unknown'. Код представлен на листинге 3.

Листинг 3 - Код замены пустых строк.

```
df['PURPOSE_ROUTE'].fillna('Unknown', inplace=True)
df.info()
```

Результат работы функции на рисунке 4.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1099 entries, 0 to 1098
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   START_DATE      1099 non-null   object
1   END_DATE        1099 non-null   object
2   CATEGORY        1099 non-null   object
3   START           1099 non-null   object
4   STOP            1099 non-null   object
5   MILES           1099 non-null   float64
6   PURPOSE_ROUTE   1099 non-null   object
7   TIME            1099 non-null   float64
8   SPEED           1099 non-null   float64
9   PRICE           1099 non-null   float64
dtypes: float64(4), object(6)
memory usage: 86.0+ KB
```

Рисунок 4 — Результат заполнения пустых строк

Далее была проведена проверка на явные дубликаты, с помощью `duplicated()`, и на явные дубликаты, с помощью `unique()`. Ни там, ни там дубликатов не было обнаружено.

Для удобства у столбцов с датой тип данных был изменен на `datetime` (Листинг 4).

Листинг 4 - Код изменения типа данных.

```
df['START_DATE'] = pandas.to_datetime(df['START_DATE'])
df['END_DATE']   = pandas.to_datetime(df['END_DATE'])
df.dtypes
```

Результат работы функции на рисунке 5.

```

START_DATE      datetime64[ns]
END_DATE        datetime64[ns]
CATEGORY        object
START           object
STOP            object
MILES           float64
PURPOSE_ROUTE   object
TIME            float64
SPEED           float64
PRICE           float64
dtype: object

```

Рисунок 5 — Результат изменения типа данных

Далее была проделана работа по визуализации данных.

Для начала была подключена библиотека `matplotlib` - одна из наиболее популярных библиотек для визуализации данных. В данной случае достаточно будет импортировать модуль `pyplot`, который содержит все необходимые компоненты для построения графиков.

Была построена диаграмма рассеивания (`scatter`), которая помогает обнаружить взаимосвязи между данными (например, определить связь роста и веса, связь стоимости жилья от площади и т.д.). Код представлен на листинге 5.

Листинг 5 — Код создания диаграммы рассеивания.

```

import matplotlib.pyplot as plt
plt.scatter(df['TIME'], df['PRICE'], s=5, color='green')

```

Результат рисования графика на рисунке 6.

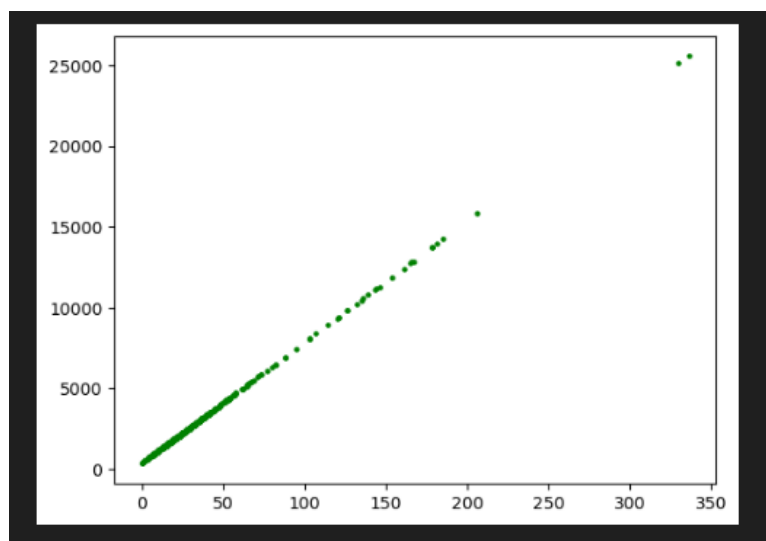


Рисунок 6 — Диаграмма рассеивания

Использовался столбец - время в пути (TIME). С помощью данного столбца легче будет строить графики, которые в дальнейшем можно анализировать. Так был построен график, который показывает зависимость времени поездки от ее цены. Из него видно, что зависимость линейная. Получается цена строится от времени в пути.

Для дальнейшего анализа была построена матрица диаграмм рассеивания - диаграммы для нескольких признаков набора данных, которая отражает попарные взаимосвязи величин.

Для построения нескольких попарных двумерных распределений в наборе данных можно воспользоваться библиотекой `seaborn`, которая содержит метод `pairplot()`. Данные графики были построены для места количества миль, времени, скорости и цены. Ниже представлен листинг 6.

Листинг 6 — Код создания матрица диаграмм рассеивания.

```
import seaborn as sns
sns.pairplot(df[['MILES', 'TIME', 'SPEED', 'PRICE']], hue = 'TIME')
```

Результат рисования графика на рисунке 7.

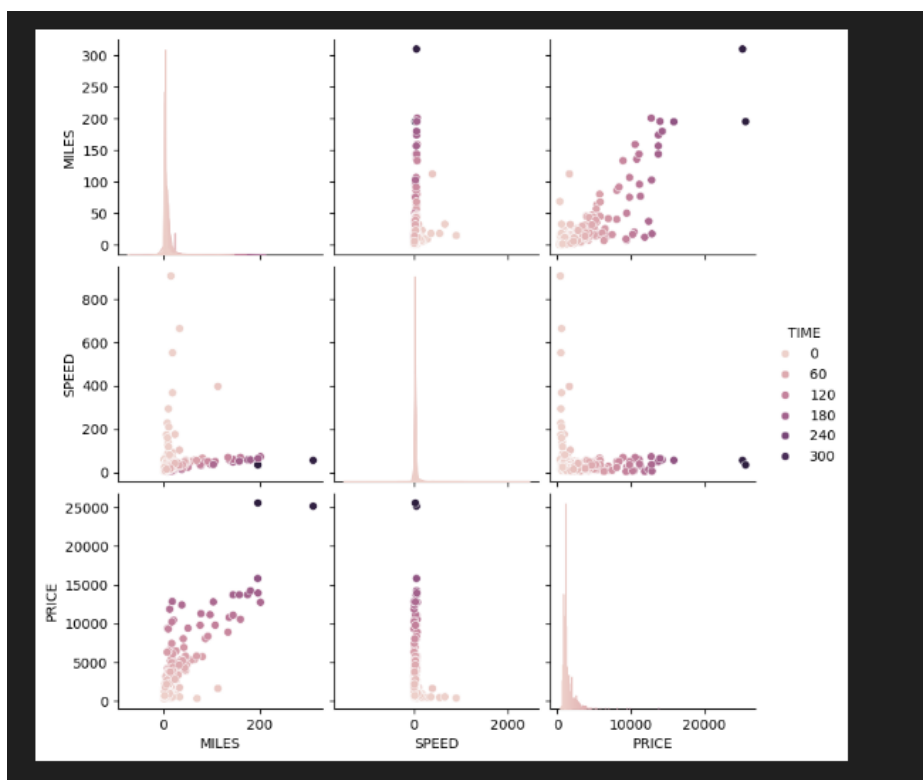


Рисунок 7 - Матрица диаграмм рассеивания

Данный график показывает зависимость скорости, пройденных миль, времени и цены от друг друга по времени.

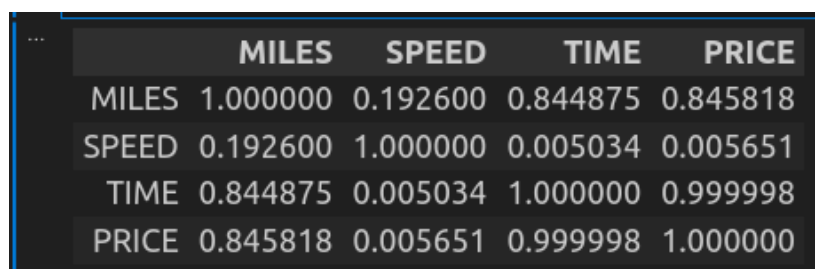
По графику можно сказать, что скорость никак не связана с другими данными, так как в целом скорость зависит только от ситуации на дороге (скоростной режим).

Также видно что количество миль также зависит от стоимости, но уже более хаотично. А так как время линейно зависимо от количества миль, то и время от стоимости зависит также как и время от количества миль.

Корреляция может быть измерена с помощью коэффициента корреляции, такого как коэффициент Пирсона. Коэффициент Пирсона находят с помощью метода `corr()`. Результат и код продемонстрированы на листинге 7, рисунке 8.

Листинг 7 — Код нахождения корреляции.

```
df2 = df[['MILES', 'SPEED', 'TIME', 'PRICE']]
df2.corr()
```



	MILES	SPEED	TIME	PRICE
MILES	1.000000	0.192600	0.844875	0.845818
SPEED	0.192600	1.000000	0.005034	0.005651
TIME	0.844875	0.005034	1.000000	0.999998
PRICE	0.845818	0.005651	0.999998	1.000000

Рисунок 8 — Результат нахождения корреляции

По корреляции уже точно можно сказать, что время линейно зависит от цены. Также видно что время и цена зависят от количества миль почти линейно. А вот скорость почти не зависит от других столбцов.

Также можно вычислить коэффициент корреляции между двумя конкретными столбцами, используя `df2['TIME'].corr(df2['PRICE'])`.

Для расчета ковариации используется функция `cov()` библиотеки `numpy`. Код представлен на листинге 8.

Листинг 8 — Код нахождения ковариации.

```
import numpy as np
data = np.array([df['TIME'], df['MILES'], df['PRICE'],
                 df['SPEED']])
cov = np.cov(data, bias=True)
print(cov)
```

На рисунке 9 представлен результат листинга 8.

```
[ [7.69130946e+02 5.16293703e+02 5.77798309e+04 nan]
 [5.16293703e+02 4.85521301e+02 3.88291572e+04 nan]
 [5.77798309e+04 3.88291572e+04 4.34063841e+06 nan]
 [ nan nan nan nan]]
```

Рисунок 9 — Результат нахождения корреляции

Ковариация положительна, что значит, что при увеличении одного значения - увеличивается и другое и наоборот. Поля с speed имеют nan так как видимо нет разнообразия данных.

Также была построена тепловая карта корреляции ('heatmap'). Матрица оформляется в цветовой палитре, так яркие цвета - положительные коэффициенты, а темные - отрицательные. 'imshow' позволяет создавать двумерные картинки, используя цветовую карту 'summer'. Для наименования столбцов и строк использовались методы 'xticks' и 'yticks'. Код представлен на листинге 9.

Листинг 9 — Код построения тепловой карты.

```
df2 = df[['MILES', 'TIME', 'SPEED', 'PRICE']]
corr = df2.corr()
plt.imshow(corr, cmap='summer')
plt.colorbar()
plt.xticks(range(len(corr.columns)),
            corr.columns)
plt.yticks(range(len(corr.columns)),
            corr.columns)
```

На рисунке 10 представлен результат листинга 9.

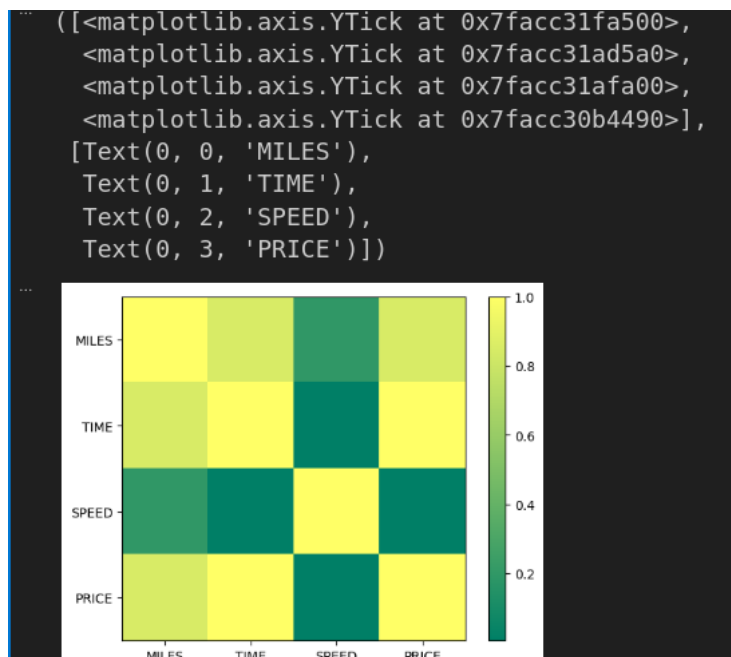


Рисунок 10 — Результат построения тепловой карты

По данному графику можно сказать, что количество миль, время и цена зависят друг от друга почти линейно (светлее), а вот скорость не зависит от других значений (темнее).

Задание 1:

Для начала была проведена группировка данных функцией `groupby`, а с помощью `size` считает кол-во вхождений. Далее создается таблица с фильтрацией количества вхождений больше двух. Функция `barplot` строит столбчатую диаграмму, где по оси X отображаются различные цели маршрута (PURPOSE_ROUTE), по оси Y отображается количество поездок, а цвет столбцов указывает на категорию (CATEGORY). Код представлен на листинге 10, а результат на рисунке 11.

Листинг 10 — Код задания 1.

```
grouped_data = df.groupby(['CATEGORY',
                           'PURPOSE_ROUTE']).size().reset_index(name='count')
filtered_data = grouped_data[grouped_data['count'] > 2]

plt.subplots(figsize=(20, 6))

sns.barplot(x='PURPOSE_ROUTE',
            y='count',
            hue='CATEGORY',
            data=filtered_data
)
plt.title('Диаграмма количества поездок по целям и категориям')
plt.xlabel(xlabel = 'Цель поездки')
plt.ylabel(ylabel = 'Количество поездок')
```

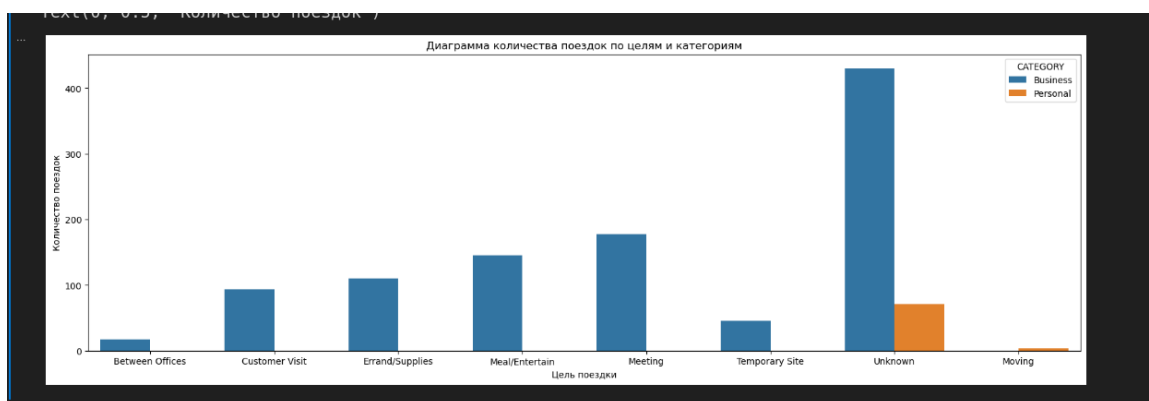


Рисунок 11 — Результат задания 1

Этот график позволяет наглядно сравнить количество поездок для различных целей маршрута в разных категориях. Визуализация данных в виде столбчатой диаграммы делает анализ более понятным и удобным для интерпретации.

Задание 2:

Сначала создаем сводную таблицу функцией `pivot_table`, с средним значением `MILES`, которые группируются по `PURPOSE_ROUTE`. Затем строим график по условию, звездочку рисуем символом - '*', а размер указываем 180. Сетку же добавляем при помощи `grid()`. Код представлен на листинге 11, а результат на рисунке 12.

Листинг 11 — Код задания 2.

```
pivot_table = pandas.pivot_table(df, values='MILES',
    index='PURPOSE_ROUTE', aggfunc='mean')
pivot_table.plot(
    style='*',
    figsize=(20, 6),
    grid = True,
    markersize = 18,
    xlabel='Цель поездки',
    ylabel='Кол-во миль',
    title='График среднего количество пройденных миль по
        каждой цели поездки',
)
```

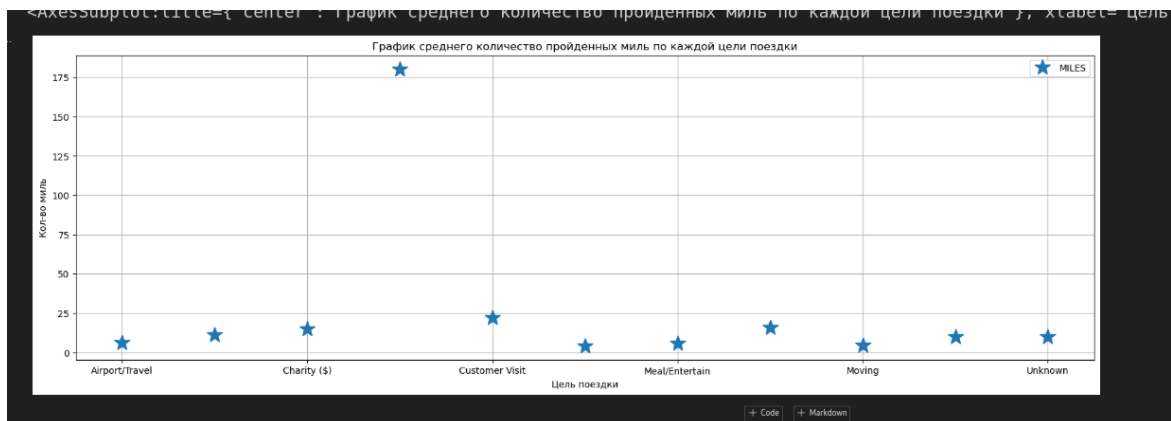


Рисунок 12 — Результат задания 2

Этот график позволяет визуализировать и сравнить средние расстояния, пройденные для различных целей поездки. Зеленые звезды на графике представляют каждую уникальную цель поездки, и их вертикальное положение отражает среднее количество миль для каждой цели. Такой график позволяет быстро определить различия в пройденных расстояниях между разными целями поездки.

Задание 3:

Создаем сводную таблицу и фильтруем ее значения аналогично предыдущим заданиям. Далее строим круговую диаграмму, у которой указываем формат отображения процентов. Код представлен на листинге 12, а результат на рисунке 13.

```

pivot_table = pandas.pivot_table(df, values='MILES',
    index='PURPOSE_ROUTE', aggfunc='count')
filtered_pivot_table = pivot_table[pivot_table['MILES']
    >= 5]
plt.figure(figsize=(8, 8))
plt.pie(
    filtered_pivot_table['MILES'],
    labels=filtered_pivot_table.index,
    autopct='%1.1f%%'
)
plt.legend(title='PURPOSE_ROUTE', loc='lower right')
plt.title('Распределение целей поездки')

```

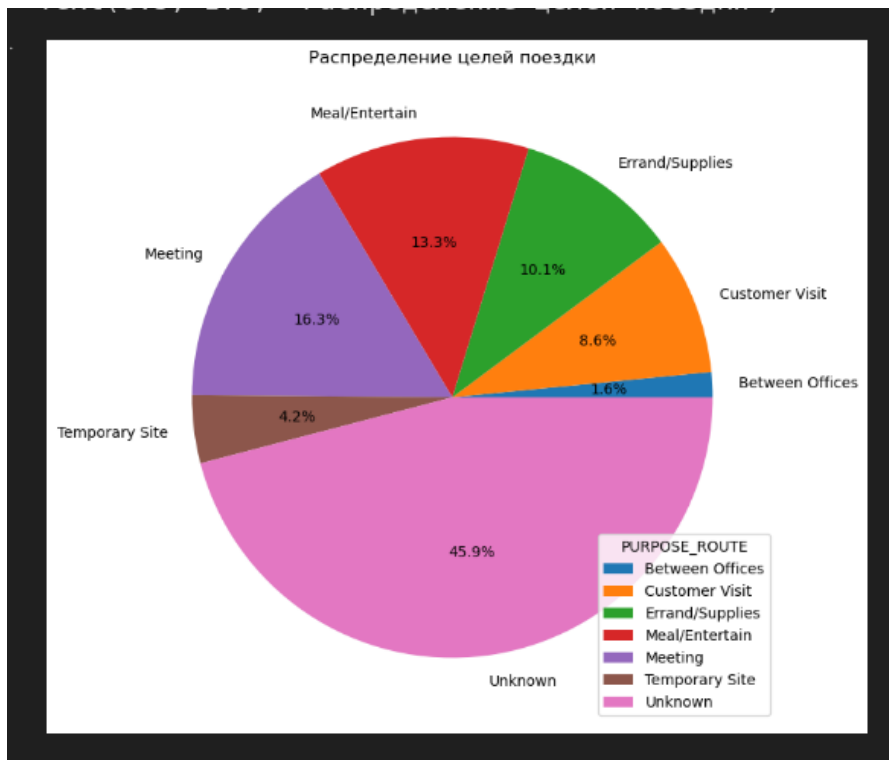


Рисунок 13 — Результат задания 3

Этот график позволяет визуализировать процентное распределение целей поездки для категорий, где количество поездок составляет 5 и более. Круговая диаграмма наглядно показывает, какие цели поездки являются более распространенными среди данных с учетом заданного условия (поездки ≥ 5).

Далее были выполнены дополнительные задания.

Дополнительное задание 1 - Из цели поездки выбрать топ 3 по количеству и построить boxplot по цели и времени.

Для начала была создана таблица `df_top_3`, в которой находятся 3 самых встречаемых цели поездки. С помощью функции `value_counts` ведется подсчет количества упоминаний, с помощью `value_counts(3)`, берутся 3 самых высоких значений. Далее строится таблица `filtered_df`, которая содержит только данные с целями поездки из `df_top_3`, с этим помогает функция `isin`, которая берет только вхождения. Далее строится `boxplot` по цели поездки и времени. Код представлен на листинге 13, а результат на рисунке 14.

Листинг 13 — Код дополнительного задания 1.

```
df_top_3 =
    df['PURPOSE_ROUTE'].value_counts().nlargest(3).index
filtered_df = df[df['PURPOSE_ROUTE'].isin(df_top_3)]
plt.figure(figsize=(12, 6))
sns.boxplot(x='PURPOSE_ROUTE', y='TIME',
            data=filtered_df, order=df_top_3)
plt.xlabel('Цель поездки')
plt.ylabel('Время в пути')
plt.title('Boxplot времени в пути для топ 3 целей
поездки')
plt.xticks(rotation=45)
plt.grid(True)
```

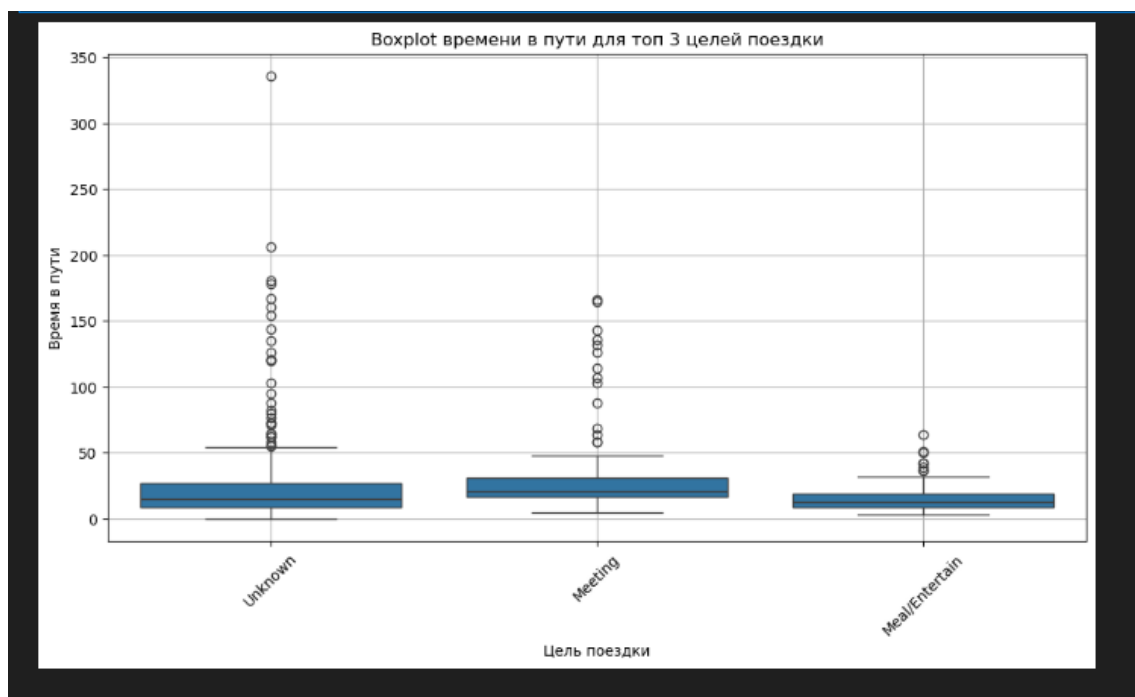


Рисунок 14 — Результат дополнительного задания 1

Анализируя этот график, можно сделать выводы о том, как различные цели поездки влияют на время в пути и наличие выбросов, что может указывать на необычные или

аномальные значения времени в пути для определенных целей. Из графика видно, что неизвестный тип поездки имеет большее время в сравнении с остальными.

Дополнительное задание 2 - Построить любой hexbin график.

Для примера построен hexbin-график для цены и времени в пути. Количество плит - 10, палитра - красная.. Код представлен на листинге 14, а результат на рисунке 15.

Листинг 14 — Код дополнительного задания 2.

```
plt.figure(figsize=(8, 6))
plt.hexbin(df['TIME'], df['PRICE'], gridsize=10,
           cmap='Reds')
plt.colorbar(label='Количество точек')
plt.xlabel('Время в пути')
plt.ylabel('Количество миль')
plt.title('Hexbin-график для времени в пути и количества миль')
```

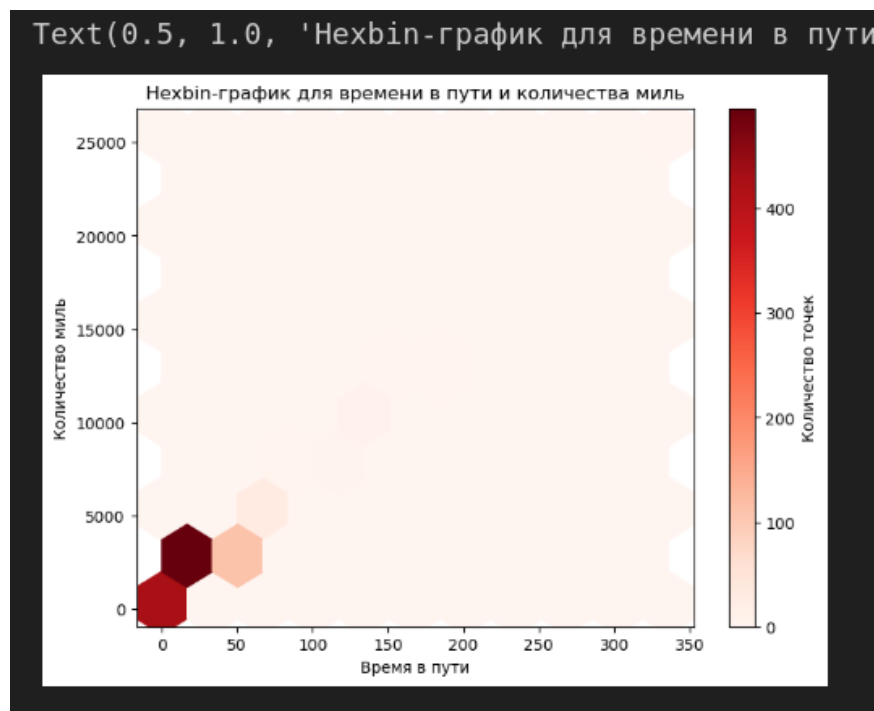


Рисунок 15 — Результат дополнительного задания 2

График получился такой маленький, так как большинство поездок по этим параметрам сосредоточено в левом нижнем углу. Следовательно можно сделать вывод, что больше 75% поездок совершены до 5000 миль и до 50 минут.

Вывод

В ходе исследования данных были освоены различные библиотеки Python для визуализации и анализа данных. Овладение навыками создания графиков, включая столбчатые, круговые и точечные диаграммы, а также умение работать с аспектами графического представления данных, такими как добавление аннотаций, легенд и цветовых схем, позволяет создавать информативные визуализации данных и проводить анализ данных, выявляя закономерности и взаимосвязи в больших объемах информации. Эти навыки будут полезны для более эффективного и наглядного представления данных в будущем, делая их доступными и понятными для аудитории.