Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Кафедра информационных технологий и автоматизированных систем

ОТЧЁТ ПО САМОСТОЯТЕЛЬНОЙ РАБОТЕ № 4

по дисциплине «Технологии анализа данных DataMining»

Тема: «Использование библиотеки pandas»

Выполнил студент гр. АСУ8-24-1м

Пельц Данил Андреевич

(Фамилия И.О.)

(номер зачетной книжки)

Проверил доцент каф. ИТАС

(должность)

Городилов Алексей Юрьевич

(Фамилия И.О)

(оценка)

(дата, подпись)

Пермь 2025 г.

Цель работы: научиться применять библиотеки pandas, numpy, scipy и matplotlib для подготовки, анализа и визуализации данных.

Задачи:

1. Ознакомиться с основными функциями и средствами обработки данных в библиотеке pandas.
2. Изучить методы численных вычислений и работы с массивами в numpy.
3. Применить статистические и математические функции библиотеки scipy для анализа данных.
4. Построить графики и визуализировать результаты анализа с помощью matplotlib.

Решение задания 1:

Для начала импортируем библиотеки для работы с данными и визуализацией.

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

После чего загружаем данные из CSV-файла athlete_events.csv в DataFrame df.

```
df = pd.read_csv("athlete_events.csv")
```

Считаем, сколько значений в каждом столбце, чтобы понять полноту данных.

```
print("Сколько значений в каждом столбце:")
print(df.count())
```

Выводим общую информацию о DataFrame, включая типы данных и количество значений в столбцах.

```
print("\nИнфо про данные:")
print(df.info())
```

Далее ищем пропущенные значения в столбцах и определяем столбец с наибольшим числом пропусков.

```
nulls = df.isnull().sum()
print("\nПропуски в данных:")
print(nulls)
print("\nБольше всего пропусков тут:", nulls.idxmax())
```

Рассчитываю базовую статистику по числовым столбцам Age, Height и Weight.

```
print("\nСтатистика по Age, Height, Weight:")
print(df[["Age","Height","Weight"]].describe())
```

Находим самого молодого участника Олимпиады 1992 года и выводим его имя, возраст и вид спорта.

```
igra1992 = df[df["Year"]==1992]
minvozr = igra1992["Age"].min()
molodoi = igra1992[igra1992["Age"]==minvozr]
print("\nСамый молодой в 1992 году:")
print(molodoi[["Name","Age","Sport"]])
```

После чего выводим список всех видов спорта, представленных в датасете.

```
print("\nСписок всех видов спорта:")
print(df["Sport"].unique())
```

Вычисляем средний рост женщин-теннисисток на Олимпиаде 2000 года.

```
tennis2000 = df[(df["Year"]==2000) & (df["Sex"]=="F") & (df["Sport"]=="Tennis")]
print("\nСредний рост теннисисток в 2000:", tennis2000["Height"].mean())
```

Считаем количество золотых медалей, завоеванных сборной Китая в настольном теннисе на Олимпиаде 2008 года.

```
china2008 = df[(df["Year"]==2008) & (df["Sport"]=="Table Tennis") & (df["Team"]=="China") & (df["Medal"]=="Gold")]
print("\nЗолота у Китая в настольном теннисе в 2008:", len(china2008))
```

Сравниваем число уникальных видов спорта на летних Олимпиадах 1988 и 2004 годов и выводим разницу.

```
sport1988 = df[(df["Year"]==1988) & (df["Season"]=="Summer")]["Sport"].nunique()
sport2004 = df[(df["Year"]==2004) & (df["Season"]=="Summer")]["Sport"].nunique()
print("\nРазница видов спорта (2004 - 1988):", sport2004 - sport1988)
```

Для визуального представления строим гистограмму распределения возраста мужчин-керлингистов на Олимпиаде 2014 года.

```
curling2014 = df[(df["Year"]==2014) & (df["Sport"]=="Curling") & (df["Sex"]=="M")]
plt.hist(curling2014["Age"].dropna(), bins=10)
plt.title("Возраст мужчин-керлингистов 2014")
plt.xlabel("Возраст")
plt.ylabel("Количество")
plt.show()
```

Анализируем зимние Олимпийские игры 2006 года: считаем количество медалей и средний возраст спортсменов по странам.

```
zim06 = df[(df["Year"]==2006) & (df["Season"]=="Winter")]
gr = zim06.groupby("NOC").agg({"Medal":"count", "Age":"mean"})
gr = gr[gr["Medal"]>0]
print("\nСтраны с медалями и средним возрастом 2006:")
print(gr)
```

В конце строим сводную таблицу с количеством медалей разных типов для каждой страны на зимних Олимпийских играх 2006 года.

```
pivot = pd.pivot_table(zim06, index="NOC", columns="Medal", values="ID", aggfunc="count", fill_value=0)
print("\nСводная таблица по медалям 2006:")
print(pivot)
```

Решение задания 2:

Для начала был произведен поиск файла данных telecom_churn.csv в нескольких возможных директориях: текущей, родительской, в папках data и /mnt/data. Если файл не обнаружен, выполняется поиск похожих CSV-файлов с ключевыми словами «telecom» или «churn» в названии. В случае неудачи выводится информативное сообщение с рекомендациями о расположении файла. Если файл найден, выводятся имена столбцов

первых пяти строк для понимания структуры данных перед загрузкой полного набора. Такой подход обеспечивает гибкую и надежную загрузку данных без необходимости жестко фиксировать путь к файлу.

```
import os, glob
import pandas as pd
import numpy as np

# Ищем файл telecom_churn.csv в нескольких местах
fn = 'telecom_churn.csv'
cands = [fn, os.path.join('/mnt/data', fn), os.path.join('.', fn), os.path.join('data', fn), os.path.join('..', fn)]
found = None
for p in cands:
    if os.path.exists(p):
        found = p
        break
if not found:
    # ищем похожие csv файлы
    lst = glob.glob('*.csv') + glob.glob('data/*.csv') + glob.glob('/mnt/data/*.csv')
    for f in lst:
        if 'telecom' in os.path.basename(f).lower() or 'churn' in os.path.basename(f).lower():
            found = f
            break

if not found:
    print("Файл 'telecom_churn.csv' не найден в окружении.")
    print("Положите файл рядом со скриптом или в /mnt/data/ и назовите 'telecom_churn.csv',\nили укажите точный путь и перезапустите.")
    print("Текущая директория:", os.getcwd())
    print("Содержимое текущей директории:", os.listdir('.'))
else:
    print("Найден файл:", found)
    # Загружаем только нужные столбцы (но сначала посмотрим шапку, чтобы адаптироваться к именам)
    preview = pd.read_csv(found, nrows=5)
    print("\nПревью столбцов (первые 5 строк) ->", list(preview.columns))
```

После чего выполняется нормализация и стандартизация названий столбцов для удобства работы с данными: пробелы удаляются, а имена приводятся к нижнему регистру. Затем задается словарь с нужными столбцами и их возможными вариантами написания, чтобы обеспечить гибкий поиск. Функция find_col реализует поиск реального имени столбца в датасете, используя несколько методов сравнения (точное совпадение, без пробелов и подчеркиваний, вхождение подстроки), что позволяет корректно сопоставлять столбцы даже при различиях в форматировании и именах. После этого собирается словарь найденных соответствий для последующего использования в анализе.

```
# нормализуем имена столбцов (уберём пробелы в начале/конце, приведём к нижнему регистру)
colmap = {c: c.strip() for c in preview.columns}
df_full = pd.read_csv(found)
df_full.rename(columns=lambda x: x.strip(), inplace=True)

# Список столбцов, которые нам нужны (в возможных вариантах написания)
want_variants = {
    'State': ['state'],
    'Area code': ['area code','area_code','areacode','area'],
    'International plan': ['international plan','international_plan','international','intl plan','intl'],
    'Number vmail messages': ['number vmail messages','number_vmail_messages','vmail','number vmail'],
    'Total day minutes': ['total day minutes','total_day_minutes','day minutes','total_day_min'],
    'Total day calls': ['total day calls','total_day_calls','day calls'],
    'Total eve minutes': ['total eve minutes','total_eve_minutes','eve minutes'],
    'Total eve calls': ['total eve calls','total_eve_calls','eve calls'],
    'Total night minutes': ['total night minutes','total_night_minutes','night minutes'],
    'Total night calls': ['total night calls','total_night_calls','night calls'],
    'Customer service calls': ['customer service calls','customer_service_calls','custservcalls','customer service','customer calls'],
    'Churn': ['churn','churn?','churned','churn_flag','churn?']
}

# функция для поиска реального имени колонки по варианту
def find_col(df_cols, variants):
    dfc = [c.lower() for c in df_cols]
    for v in variants:
        v_low = v.lower()
        # полный матч
        for i,c in enumerate(dfc):
            if c == v_low:
                return df_cols[i]
        # убираем пробелы/подчёркивания
        for i,c in enumerate(dfc):
            if c.replace(' ', '').replace('_','') == v_low.replace(' ','').replace('_',''):
                return df_cols[i]
        # contains
        for i,c in enumerate(dfc):
            if v_low in c:
                return df_cols[i]
    return None

cols_found = {}
for nice, variants in want_variants.items():
    real = find_col(list(df_full.columns), variants)
    cols_found[nice] = real
print("\nНайденные соответствия столбцов:")
for k,v in cols_found.items():
    print(f"  {k} -> {v}")
```

На следующем этапе выполняется проверка наличия всех ключевых столбцов, необходимых для дальнейшего анализа, таких как «Churn», «Customer service calls», «International plan» и суммарные минуты и звонки по разным периодам дня. В случае отсутствия каких-либо из них выводится предупреждение, однако анализ продолжается с доступными данными. Затем создается новый DataFrame, содержащий только найденные нужные столбцы, которые переименовываются в удобочитаемые названия для упрощения работы с ними. Далее производится вывод общей информации о DataFrame, включая типы данных, размер и количество пропущенных значений с помощью метода info(), что позволяет оценить качество и полноту данных. После этого выводится распределение значений в столбце «Churn» и их процентное соотношение, что помогает понять соотношение активных и ушедших клиентов в выборке.

```
# Проверяем, что ключевые столбцы найдены (Churn и Customer service calls и International plan и суммарные минуты/звонки)
required = ['Churn','Customer service calls','International plan','Total day minutes','Total eve minutes','Total night minutes','Total day calls','Total eve calls','Total night calls']
miss = [r for r in required if cols_found.get(r) is None]
if miss:
    print("\nВнимание: не найдены некоторые ожидаемые столбцы:", miss)
    print("Попытка продолжить с теми столбцами, которые найдены. Если данных мало — загрузите ожидаемый файл.")
# Создаём DataFrame с нужными столбцами, если они есть
use_cols = [v for v in cols_found.values() if v is not None]
df = df_full[use_cols].copy()
# Переименуем колонки на удобные имена
rename_map = {v:k for k,v in cols_found.items() if v is not None}
df.rename(columns=rename_map, inplace=True)

# 1) Общая информация
print("\n1) Общая информация о датафрейме (info и пропуски):")
# info() печатает в STDOUT, чтобы было аккуратно — используем print с описанием
print(df.info())
print("\nСумма пропусков по столбцам:")
print(df.isna().sum())

# 2) value_counts по Churn и проценты
print("\n2) Сколько клиентов активны и сколько потеряно (value_counts):")
if 'Churn' in df.columns:
    vc = df['Churn'].value_counts(dropna=False)
    vc_pct = df['Churn'].value_counts(normalize=True, dropna=False) * 100
    print(vc)
    print("\nПроцентное распределение (в процентах):")
    print(vc_pct.round(2))
else:
    print("Столбец Churn отсутствует — нельзя посчитать распределение.")
```

Далее добавляется новый столбец со средней продолжительностью одного звонка, рассчитываемой как отношение суммарного времени звонков за день, вечер и ночь к их общему количеству. Для устойчивости к отсутствующим данным столбцы с минутами и звонками, которые отсутствуют, заполняются нулями, чтобы избежать ошибок. Также предусмотрена обработка деления на ноль: когда количество звонков равно нулю, средняя длительность устанавливается в 0, что предотвращает появление бесконечностей или NaN и позволяет корректно выполнять дальнейший анализ. Затем выводится топ-10 клиентов с наибольшей средней длительностью звонка для выявления наиболее активных. Далее, при наличии столбца «Churn», происходит группировка данных для оценки средней длительности звонка в зависимости от статуса клиента (отток или нет), что помогает выявить закономерности в поведении пользователей.

```
# 3) Добавим столбец средняя продолжительность одного звонка:
# Суммарная длительность всех звонков = total_day_minutes + total_eve_minutes + total_night_minutes
# Суммарное кол-во звонков = total_day_calls + total_eve_calls + total_night_calls
# Берём названия столбцов, которые у нас есть (если нет — заполняем нулями)
for col in ['Total day minutes','Total eve minutes','Total night minutes','Total day calls','Total eve calls','Total night calls']:
    if col not in df.columns:
        df[col] = 0.0  # если нет — 0, чтобы не падать

df['total_minutes_all'] = df['Total day minutes'] + df['Total eve minutes'] + df['Total night minutes']
df['total_calls_all'] = df['Total day calls'] + df['Total eve calls'] + df['Total night calls']

# избегаем деления на ноль
df['avg_call_dur'] = df['total_minutes_all'] / df['total_calls_all']
# если calls == 0 -> inf или NaN, заменим на 0
df.loc[df['total_calls_all']==0, 'avg_call_dur'] = 0.0

print("\n3) Топ-10 клиентов по средней продолжительности одного звонка (по убыванию):")
top10 = df.sort_values('avg_call_dur', ascending=False).head(10)
print(top10[['avg_call_dur','total_minutes_all','total_calls_all']].to_string(index=False))

# 4) Группировка по Churn и средняя длительность одного звонка
if 'Churn' in df.columns:
    grp = df.groupby('Churn')['avg_call_dur'].mean().reset_index().rename(columns={'avg_call_dur':'mean_avg_call_dur'})
    print("\n4) Средняя продолжительность одного звонка по группам Churn:")
    print(grp.to_string(index=False))
else:
    print("\n4) Нельзя сгруппировать по Churn — столбец отсутствует.")
```

После чего выполняем группировку данных по признаку "Churn" для вычисления среднего количества звонков в службу поддержки у клиентов, которые ушли или остались, а также строим таблицу сопряженности, показывающую связь между числом звонков и уходом клиентов. В случае, если данные о «Churn» и «Customer service calls» присутствуют, он выводит среднее значение и таблицу с пропорциями, а также идентифицирует случаи, когда процент оттока превышает 40% по определённому количеству звонков, что помогает выявить повышенный риск ухода. Если необходимые столбцы отсутствуют, выводится сообщение об отсутствии информации для анализа.

```
# 5) Группировка по Churn и среднее количество звонков в службу поддержки
if 'Churn' in df.columns and 'Customer service calls' in df.columns:
    grp2 = df.groupby('Churn')['Customer service calls'].mean().reset_index().rename(columns={'Customer service calls':'mean_custserv_calls'})
    print("\n5) Среднее количество звонков в службу поддержки по Churn:")
    print(grp2.to_string(index=False))
else:
    print("\n5) Нужные столбцы отсутствуют для этой операции.")

# 6) Таблица сопряженности (crosstab) между Churn и Customer service calls
if 'Churn' in df.columns and 'Customer service calls' in df.columns:
    ct = pd.crosstab(df['Customer service calls'], df['Churn'], margins=False)
    print("\n6) Таблица сопряженности (Customer service calls x Churn):")
    print(ct.to_string())
    # вычислим процент оттока в каждой строке (по числу клиентов с данным числом звонков)
    ct_pct = ct.div(ct.sum(axis=1), axis=0).fillna(0)
    # если названия колонок содержают булевы значения, приведём их к строке 'True'/'False' для доступа
    print("\nПроцент оттока в разрезе количества звонков (в строках):")
    print((ct_pct * 100).round(2).to_string())
    # где процент оттока > 40%?
    # найдем значения customer service calls, где churn True процент > 40%
    if True in ct_pct.columns or 'True' in ct_pct.columns or 1 in ct_pct.columns:
        # определить колонки, обозначающие отток: ищем колонку, которая соответствует True/1/'True'
        churn_cols = [c for c in ct_pct.columns if str(c).lower() in ['true','1','t','y','yes'] or isinstance(c, bool) and c is True]
        if not churn_cols:
            # возможно столбец имеет метку 'True' в строковом виде
            churn_cols = [c for c in ct_pct.columns if str(c).lower()=='true']
        if churn_cols:
            churn_col = churn_cols[0]
            high = (ct_pct[churn_col] > 0.4)
            high_idx = list(ct_pct.index[high])
            print("\nЗначения 'Customer service calls', при которых процент оттока > 40%:", high_idx)
        else:
            # если не нашли явного столбца True — попробуем выбрать любой столбец с меткой, равной True при приведении
            print("\nНе удалось явно определить столбец, обозначающий отток (True).")
    else:
        print("\nНе найден столбец, соответствующий значению True у Churn в crosstab.")
else:
    print("\n6) Нужные столбцы (Churn и Customer service calls) отсутствуют для crosstab.")
```

Следующий этап – анализ связи между оттоком клиентов (Churn) и наличием у них международного плана (International plan) с помощью таблицы сопряженности, позволяющей визуализировать распределение клиентов по этим категориям и вычислить процент оттока внутри групп. После чего формируется простая эвристическая прогностическая модель, которая предсказывает отток, если клиент либо имеет более 3 звонков в службу поддержки, либо подключен к международному плану. Для этого международный план нормализуется в булев тип по стандартным значениям "yes", "true" и т.п., а количество звонков приводится к числовому виду.

```
# 7) Связь Churn и International plan
if 'Churn' in df.columns and 'International plan' in df.columns:
    ct2 = pd.crosstab(df['International plan'], df['Churn'], margins=False)
    print("\n7) Таблица сопряженности (International plan x Churn):")
    print(ct2.to_string())
    ct2_pct = ct2.div(ct2.sum(axis=1), axis=0).fillna(0)
    print("\nПроцент оттока внутри групп International plan (в процентах):")
    print((ct2_pct * 100).round(2).to_string())
else:
    print("\n7) Нужные столбцы для анализа International plan отсутствуют.")

# 8) Простая прогностическая метрика на основе Customer service calls и International plan
# Предсказываем CHURN=True, если Customer service calls >= 3 OR International plan == 'yes' (варианты Yes/YES/true/True учитываются)
print("\n8) Построим простой прогноз (школьный эвристический):")
def is_yes(x):
    if pd.isna(x): return False
    s = str(x).strip().lower()
    return s in ['yes','y','true','t','1','on']

# нормализуем International plan в булевую колонку
if 'International plan' in df.columns:
    df['int_plan_bool'] = df['International plan'].apply(is_yes)
else:
    df['int_plan_bool'] = False

# гарантируем что Customer service calls числовой
if 'Customer service calls' in df.columns:
    df['Customer service calls'] = pd.to_numeric(df['Customer service calls'], errors='coerce').fillna(0).astype(int)
else:
    df['Customer service calls'] = 0

# предсказание по правилу
df['pred_churn'] = ((df['Customer service calls'] >= 3) | (df['int_plan_bool'] == True))
```

Ну и в конце реализуется преобразование реальных меток оттока (Churn) к булевому типу для унифицирования значения для последующего сравнения с прогнозом. Затем вычисляются компоненты матрицы ошибок (confusion matrix): истинно-положительные (TP), истинно-отрицательные (TN), ложноположительные (FP) и ложноотрицательные (FN) случаи, по которым рассчитываются ошибки первого и второго рода (false positive rate и false negative rate). Код выводит сводную таблицу с сопоставлением реальных и предсказанных значений, а также примеры ложных срабатываний.

```python
# реальные метки — приведём к булевому виду (True/False)
def to_bool_churn(x):
    if pd.isna(x): return False
    if isinstance(x, bool): return x
    s = str(x).strip().lower()
    if s in ['true','t','yes','y','1','churn','true.']:
        return True
    if s in ['false','f','no','n','0','active','false.']:
        return False
    # если это число: 1 -> True, 0 -> False
    try:
        v = float(s)
        return v != 0.0
    except:
        return False

df['real_churn_bool'] = df['Churn'].apply(to_bool_churn) if 'Churn' in df.columns else False

# confusion matrix components
TP = ((df['pred_churn']==True) & (df['real_churn_bool']==True)).sum()   # true positives
TN = ((df['pred_churn']==False) & (df['real_churn_bool']==False)).sum()
FP = ((df['pred_churn']==True) & (df['real_churn_bool']==False)).sum()  # false positives (Type I)
FN = ((df['pred_churn']==False) & (df['real_churn_bool']==True)).sum()  # false negatives (Type II)
total = len(df)
print(f"Всего записей: {total}. TP={TP}, TN={TN}, FP={FP}, FN={FN}")

# Ошибка первого рода (false positive rate) обычно = FP / (FP + TN) — доля отрицательных, ошибочно отмеченных как положительные
# Ошибка второго рода (false negative rate) = FN / (FN + TP) — доля положительных, ошибочно промаркированных как отрицательные
fpr = FP / (FP + TN) if (FP + TN) > 0 else np.nan
fnr = FN / (FN + TP) if (FN + TP) > 0 else np.nan
print("\nПроцент ошибок:")
print(f"  Ошибка первого рода (FP rate) = {fpr*100:.2f}% (FP / (FP+TN))")
print(f"  Ошибка второго рода (FN rate) = {fnr*100:.2f}% (FN / (FN+TP))")

# Также покажем простую таблицу сравнения предикта и реального
conf = pd.crosstab(df['real_churn_bool'], df['pred_churn'])
print("\nТаблица сопряженности (реально x предсказано):")
print(conf.to_string())

# Выведем примеры ошибок (несколько строк)
print("\nНесколько примеров ложноположительных (предсказали churn, но на самом деле нет):")
print(df[(df['pred_churn']==True)&(df['real_churn_bool']==False)].head(5)[['Customer service calls','International plan','pred_churn','real_churn_bool','avg_call_dur']].to_string(index=False))

print("\nНесколько примеров ложноотрицательных (предсказали не churn, но на самом деле churn):")
print(df[(df['pred_churn']==False)&(df['real_churn_bool']==True)].head(5)[['Customer service calls','International plan','pred_churn','real_churn_bool','avg_call_dur']].to_string(index=False))

# Отобразим df небольшой выборкой пользователю
try:
    from caas_jupyter_tools import display_dataframe_to_user
    display_dataframe_to_user("Telecom churn sample with predictions", df.head(200))
except Exception:
    pass

print("\nАнализ завершён.")
```

Решение задания 3:

Дня начала загружаем страницу Википедии с перечнем чемпионов Формулы 1 по годам, используя requests и BeautifulSoup для получения и парсинга HTML кода. После получения HTML передаем его в pandas для извлечения всех таблиц, присутствующих на странице, и выводим количество найденных таблиц. Это подготовительный этап для дальнейшей работы с таблицей чемпионов — последующего выбора нужной таблицы и очистки данных.

```python
import pandas as pd
import requests
from bs4 import BeautifulSoup
import re

# --- 1. Загружаем страницу ---
url = "https://en.wikipedia.org/wiki/List_of_Formula_One_World_Drivers%27_Champions"
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)
response.raise_for_status()

# --- 2. Парсим HTML через BeautifulSoup ---
soup = BeautifulSoup(response.text, "html.parser")
html = str(soup)

# --- 3. Читаем таблицы через pandas ---
tables = pd.read_html(html)
print(f"Найдено таблиц: {len(tables)}")
```

После чего происходит проверка, является ли индекс столбцов MultiIndex, и если да, то он преобразуется в плоский список строк, объединяя уровни через разделитель " / ". Это упрощает дальнейшую работу с таблицей, так как устраняет вложенность в названиях колонок. Затем определяется функция для поиска столбцов, которые содержат заданное

ключевое слово (например, "Season", "Driver", "Age" и т.д.), и с её помощью находятся нужные названия столбцов для последующего анализа.

```
# --- 4. Находим таблицу с сезонами ---
df = None
for i, t in enumerate(tables):
    if any("Season" in str(c) for c in t.columns) and any("Driver" in str(c) for c in t.columns):
        print(f"✅ Найдена таблица под номером {i}")
        df = t
        break

# --- 5. Преобразуем MultiIndex в плоский список ---
if isinstance(df.columns, pd.MultiIndex):
    df.columns = [f"{a}".strip() + (" / " + f"{b}".strip() if b else "") for a, b in df.columns]

# --- 6. Находим нужные колонки ---
def find_col_like(df, keyword):
    for c in df.columns:
        if keyword.lower() in c.lower():
            return c
    return None
col_season = find_col_like(df, "Season")
col_driver = find_col_like(df, "Driver")
col_age = find_col_like(df, "Age")
col_wins = find_col_like(df, "Wins")
col_chassis = find_col_like(df, "Chassis")
col_engine = find_col_like(df, "Engine")
col_rounds = find_col_like(df, "round")
```

Следующий этап задания – новый столбец "Constructor", который объединяет название шасси и двигателя через " / " для формирования полного название команды-конструктора. После этого числовые столбцы, такие как возраст гонщика и количество побед, преобразуются из строковых значений в числовой тип с помощью pandas. Это необходимо для корректного математического анализа и обработки данных.

```
# --- 7. Конструктор ---
if col_chassis and col_engine:
    df["Constructor"] = df[col_chassis] + " / " + df[col_engine]
col_constructor = "Constructor"

# --- 8. Преобразуем числовые данные ---
df[col_age] = pd.to_numeric(df[col_age], errors="coerce")
df[col_wins] = pd.to_numeric(df[col_wins], errors="coerce")
```

Далее на фото решаются три задачи. Во-первых, вычисляются и выводятся средний, минимальный и максимальный возраст чемпионов Формулы 1, используя числовой столбец с возрастом. Во-вторых, определяется топ конструкторов (команд), которые набрали наибольшее количество чемпионских титулов, группируя по объединенному названию шасси и двигателя. В-третьих, выводится топ гонщиков по числу завоеванных титулов. Получился базовый аналитический блок для получения ключевых статистик по возрасту чемпионов и их командам.

```
# --- 9. Вопрос 1 ---
print("\n---- Вопрос 1: Возраст чемпионов ----")
print("Средний:", df[col_age].mean())
print("Минимальный:", df[col_age].min())
print("Максимальный:", df[col_age].max())

# --- 10. Вопрос 2 ---
print("\n---- Вопрос 2: Топ конструкторов ----")
print(df.groupby(col_constructor).size().sort_values(ascending=False).head(10))

# --- 11. Вопрос 3 ---
print("\n---- Вопрос 3: Топ гонщиков ----")
print(df.groupby(col_driver).size().sort_values(ascending=False).head(10))
```

Следующий этап – решение задачи выявления чемпионов с менее чем 30% побед в сезоне. Из столбца со сведениями о количестве гонок в сезоне методом регулярных выражений извлекается общее число гонок, затем вычисляется процент побед каждого чемпиона относительно числа гонок, а записи с неизвестным числом гонок удаляются для точности анализа. Далее фильтруются и выводятся гонщики, чей процент побед меньше 30%. Это позволяет выявить чемпионов, выигравших меньше трети гонок в своем выигрышном сезоне, что является показателем нестандартного пути к титулу.

```
print("\n---- Вопрос 4: Чемпионы с менее чем 30% побед в сезоне ----")
# Используем колонку 'Clinched[17] / Clinched[17]' для определения количества гонок в сезоне
col_title_clinched = 'Clinched[17] / Clinched[17]'
def extract_race_info(text):
    if pd.isna(text):
        return None
    text = str(text)
    # Ищем паттерны типа "10 of 17", "race 8 out of 16" и т.д.
    matches = re.findall(r'(\d+)\s*(?:of|out of|/)\s*(\d+)', text)
    if matches:
        return int(matches[0][1])  # возвращаем общее количество гонок
    return None
# Извлекаем информацию о гонках
df['Total_Races'] = df[col_title_clinched].apply(extract_race_info)
if df['Total_Races'].isna().any():
    # Удаляем строки, где не удалось определить количество гонок
    df_clean = df.dropna(subset=['Total_Races']).copy()
else:
    df_clean = df.copy()
# Вычисляем процент побед
df_clean['Win_Percentage'] = (df_clean[col_wins] / df_clean['Total_Races']) * 100
# Находим гонщиков с менее чем 30% побед
low_win_champions = df_clean[df_clean['Win_Percentage'] < 30]
if not low_win_champions.empty:
    print("Гонщики, ставшие чемпионами с менее чем 30% побед в сезоне:")
    for _, row in low_win_champions.iterrows():
        print(f"{row[col_driver]} ({row[col_season]}): {int(row[col_wins])}/{int(row['Total_Races'])} побед ({row['Win_Percentage']:.1f}%)")
else:
    print("Не найдено гонщиков с менее чем 30% побед")
```

Последняя часть кода – определение максимального перерыва между последовательными чемпионствами гонщиков, которые становились чемпионами как минимум дважды. Для этого данные очищаются от некорректных значений сезонов, затем для каждого гонщика формируется список годов его титулов, после чего вычисляются интервалы между соседними чемпионствами. Максимальный промежуток запоминается с соответствующим именем гонщика и периодом. Такой подход позволяет выявить самого "устойчивого" или "возвращающегося" чемпиона в истории Формулы 1.

```python
print("\n---- Вопрос 5: Максимальный перерыв между последовательными чемпионствами ----")
# Очищаем данные от строк с некорректными значениями Season
df_clean_season = df[pd.to_numeric(df[col_season], errors='coerce').notna()].copy()
df_clean_season[col_season] = pd.to_numeric(df_clean_season[col_season])
# Группируем по гонщикам и собираем года их чемпионств
champions_by_driver = df_clean_season.groupby(col_driver)[col_season].apply(list).reset_index()
# Оставляем только тех, у кого 2 или более титулов
multiple_champions = champions_by_driver[champions_by_driver[col_season].apply(len) >= 2]
max_break = 0
max_break_driver = ""
max_break_period = ""
for _, row in multiple_champions.iterrows():
    driver = row[col_driver]
    years = sorted(row[col_season])  # сортируем года по возрастанию
    # Вычисляем перерывы между последовательными чемпионствами
    for i in range(1, len(years)):
        break_years = years[i] - years[i-1] - 1  # перерыв в годах между титулами
        if break_years > max_break:
            max_break = break_years
            max_break_driver = driver
            max_break_period = f"{years[i-1]}-{years[i]}"
print(f"Максимальный перерыв между последовательными чемпионствами: {max_break} лет")
print(f"Гонщик: {max_break_driver}")
print(f"Период: {max_break_period}")
```

Вывод:

В ходе выполнения лабораторной работы были изучены и практически применены основные методы и инструменты библиотек pandas, numpy, scipy и matplotlib для анализа и визуализации данных. Полученные результаты анализа данных позволили наглядно продемонстрировать использование статистических функций, обработку массивов и построение графиков.

Выполненные задачи позволили понять важность комплексного подхода к работе с данными — от предобработки и описательной статистики до визуальной интерпретации результатов. Результаты работы могут быть полезны при дальнейшем анализе больших объемов информации и построении более сложных моделей.

Также выявлены возможности для улучшения работы, включая расширение спектра применяемых методов и углубление анализа. Это может стать основой для последующих исследований и практических применений в области анализа данных.