In [1]:
```
import pandas as pd
```

In [25]:
```
df = pd.read_csv('C:/Users/stask/Analitics_Karpov/Module4/taxi_peru.csv', sep=';')
df.head()
```

Out[25]:

	journey_id	user_id	driver_id	
0	23a1406fc6a11d866e3c82f22eed4d4c	0e9af5bbf1edfe591b54ecdfd7e91e26	583949a89a9ee17d19e3ca4f137b6b4c	b
1	dd2af4715d0dc16eded53afc0e243577	a553c46e3a22fb9c326aeb3d72b3334e	NaN	
2	dd91e131888064bf7df3ce08f3d4b4ad	a553c46e3a22fb9c326aeb3d72b3334e	NaN	
3	dd2af4715d0dc16eded53afc0e2466d0	a553c46e3a22fb9c326aeb3d72b3334e	NaN	
4	85b7eabcf5d84e42dc7629b7d27781af	56772d544fdfa589a020a1ff894a86f7	d665fb9f75ef5d9cd0fd89479380ba78	0a

In [3]:
```
# 1) The variable df contains a dataframe.
# Your task is to put a string in the df_shape variable with information
# about how many lines and columns it has in the following form:
# 'df has y rows and x columns'. where y is the number of rows and x is the number of co
print(f'df has {df.shape[0]} rows and {df.shape[1]} columns')
```

```
df has 23111 rows and 19 columns
```

In [4]:
```
# 2) Put a series in the na_number variable,
# which tells you for each column how many cells contain missing values.
# For example, for this dataframe: task2_1.png (in the folder)
# The answer would be: task2_2.png (in folder)

# Create an empty dictionary to store column names and their respective missing value co
col_missing_count = dict()

# Iterate over each column in the DataFrame
for col in df.columns:
    # Count the number of missing values in the current column and assign it to the dict
    col_missing_count[col] = df[col].isnull().sum()

# Convert the dictionary to a DataFrame
dict_dataframe = pd.DataFrame.from_dict(col_missing_count, orient='index', columns=['mis

# Print the resulting DataFrame
dict_dataframe
```

Out[4]:

	missing_values_count
journey_id	0
user_id	0
driver_id	3385
taxi_id	3385
icon	0
start_type	0
start_at	0
start_lat	0
start_lon	0
end_at	276
end_lat	0
end_lon	0
end_state	12
driver_start_lat	3490
driver_start_lon	3490
arrived_at	5395
source	123
driver_score	15461
rider_score	7721

In [5]:

```
# it can be done much easier:
df.isna().sum().reset_index().rename(columns={'index':'column', 0:'missing_values_count'
```

Out[5]:

	column	missing_values_count
0	journey_id	0
1	user_id	0
2	driver_id	3385
3	taxi_id	3385
4	icon	0
5	start_type	0
6	start_at	0
7	start_lat	0
8	start_lon	0
9	end_at	276
10	end_lat	0
11	end_lon	0
12	end_state	12
13	driver_start_lat	3490
14	driver_start_lon	3490
15	arrived_at	5395
16	source	123
17	driver_score	15461
18	rider_score	7721

In [6]:
```
# 3) Save to the variable df_types the data types of each df dataphrame column.
df_types = df.dtypes
```

In [7]:
```
# To delete a column in pandas DataFrame, you can use the drop() method.

# new_df = df.drop('column_name', axis=1)
```

In [8]:
```
# Remove duplicates based on all columns

# df.drop_duplicates(inplace=True)


# Remove duplicates based on specific columns

# df.drop_duplicates(subset=['column1', 'column2'], inplace=True)
```

```
7) A short break to pure python. The list of numbers contains numbers. Add positive numbers from numbers to the
positive_numbers list via a loop. If you get a 0, this loop has to be terminated.
For example,
numbers = [1, -2, 3, 0, -3, 2]
Then positive_numbers should be:
positive_numbers = [1, 3]
```

In [9]:
```
list_num = [2, -2, 3, 1, -3, 0]
list_positive = list()
for num in list_num:
    if num == 0:
        break
    if num > 0:
```

```
        list_positive.append(num)
print(list_positive)
```

```
[2, 3, 1]
```

8) Let's continue the analysis of the trip data. In the previous lesson you examined the distributions of estimates for drivers and customers. Now let's look at the columns over time!
Save the data in the folder (taxi_peru.csv) to the variable taxi, separator - ;. Then bring the columns start_at, end_at, arrived_at to date format using pd.to_datetime(). Filter the data and leave observations with order status "asap" and "reserved" (start_type) .

In [10]:
```
taxi = pd.read_csv('C:/Users/stask/Analitics_Karpov/Module4/taxi_peru.csv', sep=';')
df.dtypes
```

Out[10]:
```
journey_id           object
user_id              object
driver_id            object
taxi_id              object
icon                 object
start_type           object
start_at             object
start_lat            object
start_lon            object
end_at               object
end_lat              object
end_lon              object
end_state            object
driver_start_lat     object
driver_start_lon     object
arrived_at           object
source               object
driver_score        float64
rider_score         float64
dtype: object
```

In [11]:
```
taxi = taxi.query('start_type in ["asap", "reserved"]')

taxi['start_at'] = pd.to_datetime(taxi['start_at'])
taxi['arrived_at'] = pd.to_datetime(taxi['arrived_at'])
taxi['end_at'] = pd.to_datetime(taxi['end_at'])
taxi[['start_at', 'arrived_at', 'end_at']].dtypes
# another way to do the same
# taxi[['arrived_at', 'start_at', 'end_at']].apply(pd.to_datetime)
taxi.shape[0]
```

Out[11]:
```
23091
```

9) ⭐ Task with an asterisk!⭐
Create a wait_time column, which will store the difference between the machine arrival time (arrived_at) and the ordering time (start_at) in minutes. You can use the .astype('timedelta64[m]') method to convert the result to minutes

In [12]:
```
taxi['wait_time'] = (taxi.arrived_at - taxi.start_at).astype('timedelta64[m]')
```

9.1) ⭐ Task with asterisk!⭐
Let's see which drivers were late for their orders by a certain time (start_type == 'reserved'). Wait_time > 0.0 is considered late.
Try to group filtered data by driver_id and answer with the id of driver who was late most of the time

In [13]:
```
taxi.query('start_type == "reserved" and wait_time > 0').driver_id.value_counts()
# if task is to get max in would be not effective to sort list in the beginning (finding
```

Out[13]:
```
406921adcca37705ef527b4246c0cfea    67
d665fb9f75ef5d9cd0fd89479380ba78    59
ec84a73745199ff840ecafcb924383ad    57
56f59b58bcbbd1cdabc3652e713134c2    51
c814db2127582cf95dea1f74f43127c2    45
                                    ..
fa5c3d1ad73379ba86b960210e63d537     1
b5821eaaa5d49fb2936ff5b0ffa34a35     1
761737b194876dd53761a03c958f7660     1
cd6fba80de11849ce566009f41dd27a0     1
bc5c1ae63a528f0371154594b3477211     1
Name: driver_id, Length: 105, dtype: int64
```

11) ⭐ Task with an asterisk!⭐
Add a month column to store the month in which the order was placed (start_at) as a number, and a weekday column to name the day of the week.

In [22]:
```
taxi['month_num'] = taxi.start_at.dt.month
taxi['day_name'] = taxi.start_at.dt.day_name()
taxi[['start_at', 'month_num', 'day_name']].head(5)
```

Out[22]:

	start_at	month_num	day_name
0	2010-11-16 16:44:00	11	Tuesday
1	2010-06-01 00:34:00	6	Tuesday
2	2010-05-31 05:01:00	5	Monday
3	2010-06-01 00:29:00	6	Tuesday
4	2010-09-11 23:55:00	9	Saturday

12) ⭐ Task with an asterisk!⭐
Plot the number of orders by month (month). Think about what the resulting picture might be related to.
Question: In which summer month were the least number of orders placed? Answer the question by writing the number of the month (6, 7 or 8).

In [38]:
```
taxi.query('month_num in (6,7,8)') \
    .groupby('month_num') \
    .agg({'driver_id':'count'}) \
    .rename(columns={'driver_id':'orders_count'}) \
    .idxmax()
```

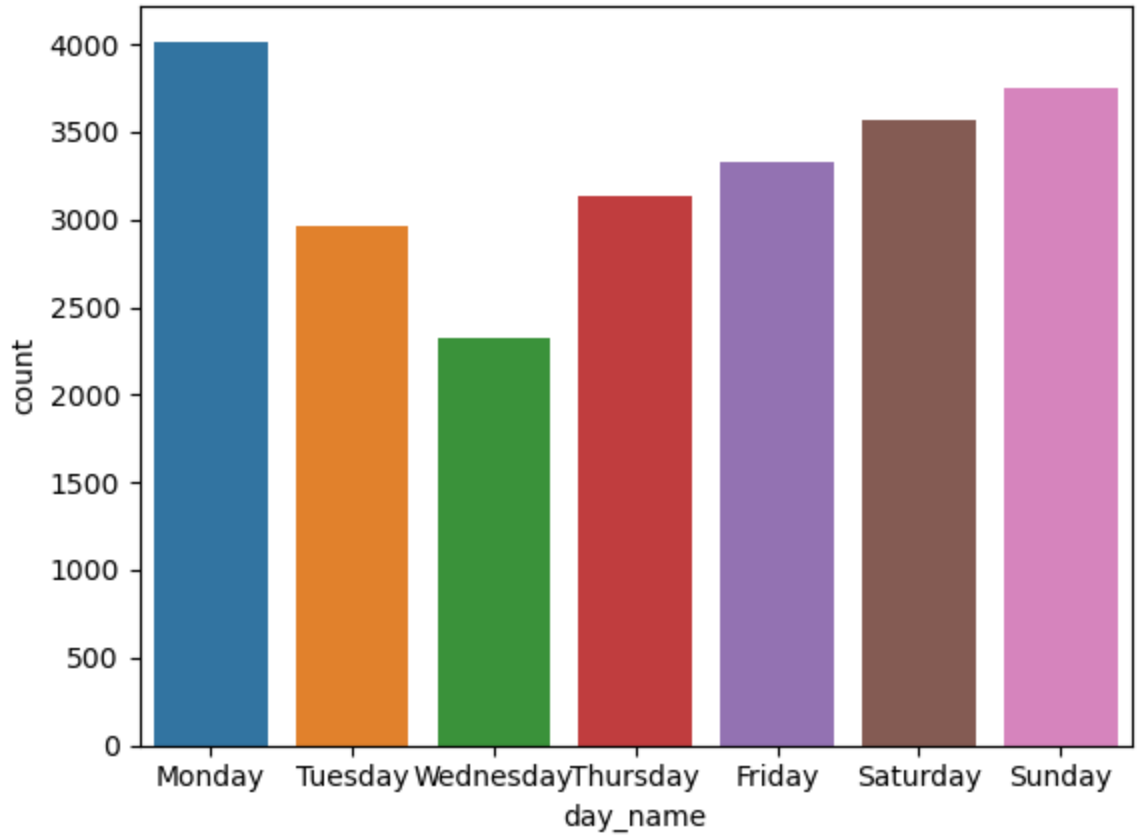
Out[38]:
```
orders_count    8
dtype: int64
```

13) ⭐ A task with an asterisk!
Now the graph by weekday. You can specify the order of the columns in the chart using the argument order, to which you have to pass the list of names in the desired order. For example:
sns.countplot(data['column'], order=['One', 'Two', 'Three'])
Question: on which day of the week did the total number of orders come in the least?

In [77]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(taxi.day_name, order=['Monday','Tuesday', 'Wednesday', 'Thursday', 'Friday
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

Out[77]:
```
<AxesSubplot:xlabel='day_name', ylabel='count'>
```



14) ☆ Task with an asterisk!
We can also look at some simple metrics showing the number of unique users in a given period:
DAU (daily active users) - number of unique users per day
WAU (weekly active users) - number of unique users per week
MAU (monthly active users) - number of unique users per month
Active users are those who have used the application and placed at least one order within selected time frame.
Plot the MAU and select the correct statements.
May come in handy:
groupby - grouping
nunique - number of unique values
plot.line - line chart

In [79]:
```
MAU = taxi.groupby('month_num', as_index=False) \
    .agg({'user_id': pd.Series.nunique}) \
    .rename(columns={'user_id': 'unique_users_count'})
plt.figure(figsize=(10,4))
sns.lineplot(data=MAU, x='month_num', y='unique_users_count', markers=True, marker='D')
plt.xlabel('Month')
plt.ylabel('Count of Unique Users')
plt.title('Distribution of Unique Users per Month')
plt.grid(True)
plt.xticks(range(1, 13))
plt.yticks(range(50, 650, 50))
sns.despine()
plt.show()
```