

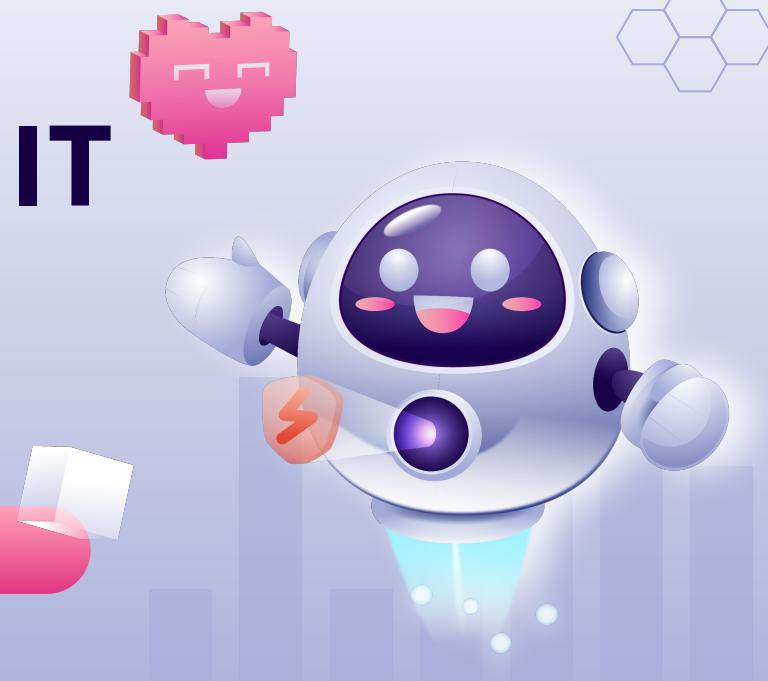


Make your business more efficient

increase
revenue

Cleaning and Analyse Data of IT online school

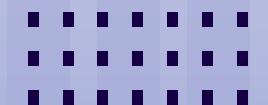
Medvedieva Svitlana





Cleaning Data

Be confident in your opportunities



Data structure

Table Deals

```
<class 'pandas.core.frame.DataFrame'>
Index: 21593 entries, 0 to 21592
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Id               21593 non-null   Int64  
 1   Deal Owner Name 21564 non-null   object  
 2   Closing Date     14645 non-null   object  
 3   Quality          19340 non-null   object  
 4   Stage            21593 non-null   object  
 5   Lost Reason      16124 non-null   object  
 6   Page              21593 non-null   object  
 7   Campaign          16067 non-null   object  
 8   SLA               15533 non-null   object  
 9   Content           14147 non-null   object  
 10  Term              12454 non-null   object  
 11  Source             21593 non-null   object  
 12  Payment Type     496 non-null    object  
 13  Product            3592 non-null   object  
 14  Education Type   3299 non-null   object  
 15  Created Time      21593 non-null   object  
 16  Course duration   3587 non-null   float64 
 17  Months of study   840 non-null    float64 
 18  Initial Amount Paid 4165 non-null   object  
 19  Offer Total Amount 4185 non-null   object  
 20  Contact Name      21532 non-null   Int64  
 21  City               2511 non-null   object  
 22  Level of Deutsch  1251 non-null   object  
dtypes: Int64(2), float64(2), object(19)
memory usage: 4.0+ MB
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13970 entries, 0 to 13969
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Id               13970 non-null   Int64  
 1   Deal Owner Name 13970 non-null   object  
 2   Closing Date     13970 non-null   datetime64[ns] 
 3   Stage            13970 non-null   category 
 4   Lost Reason      13970 non-null   category 
 5   Campaign          13970 non-null   object  
 6   SLA               13970 non-null   float64  
 7   Source             13970 non-null   object  
 8   Payment Type     13970 non-null   object  
 9   Product            13970 non-null   category 
 10  Education Type   13970 non-null   category 
 11  Created Time      13970 non-null   datetime64[ns] 
 12  Course duration   13970 non-null   int64  
 13  Months of study_deals 13970 non-null   float64 
 14  Initial Amount Paid 3211 non-null   float64 
 15  Offer Total Amount 3246 non-null   float64 
 16  Contact Name      13970 non-null   Int64  
 17  City               13970 non-null   object  
 18  Level of Deutsch  13970 non-null   category 
 19  Quality2           13970 non-null   category  
dtypes: Int64(2), category(6), datetime64[ns](2), float64(4), int64(1), object(5)
```

Cleaning strategy. Table ‘Deals’



Удаление полных дубликатов

Перевела столбцы с временем в формат времени, выбрав только год, месяц и день.
Closing day заполнила значениями **2025** года



Удаление дубликатов **Contact Name**

SLA привела сначала в формат строки, затем в число, означающее часы. Пустые значения заполнила медианой



Quality убрала буквы. Пустые значения = **No**



Деньги привела к одной валюте - евро.



Level of Deutsh оставила только ’A2’, ’B1’, ’B2’, ’C2’, ’A1’, ’C1’



Убрала все ячейки, ведущие на **test**, без оплат и без РК

Удаление дубликатов и пустых строк

```
1 filtered_rows = deals[deals['Id'].isnull()]
2 filtered_rows
```

Id	Deal Owner Name	Closing Date	Quality	Stage	Lost Reason	Page	Campaign	SLA	Content	...	Product	Education Type	Created Time	Course duration	Months of study	Initial Amount Paid	Offer Total Amount	Contact Name	City	Level of Deutsch
21593	<NA>	Nan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	<NA>	NaN	NaN
21594	<NA>	Nan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	#REF!	NaN	NaN	NaN	NaN	NaN	<NA>	NaN	NaN

```
1 deals['Contact Name'] = deals['Contact Name'].fillna(0)
2 deals['Contact Name']
```

```
1 deals['Contact Name'].duplicated()
2 duplicates = deals[deals['Contact Name'].duplicated(keep=False)]
```

6039 rows × 23 columns

```
1 deals_sorted = deals.sort_values(
2     by=['Offer Total Amount', 'Initial Amount Paid'],
3     ascending=[False, False]
4 )
5 deals_cleaned = deals_sorted.drop_duplicates(subset='Contact Name', keep='first')
6 deals_cleaned = deals_cleaned.reset_index(drop=True)
7 deals_cleaned
```

Отсортировала по финансовым колонкам и оставила первую строку из каждой группы

Временные столбцы

```
[ ] 1 deals['Created Time'] = pd.to_datetime(deals['Created Time'].str.split().str[0], format='%d.%m.%Y')
2 deals['Created Time'] #перевела столбец в дата тип и убрала время
```

```
1 deals['Closing Date'] = pd.to_datetime(deals['Closing Date'].str.split().str[0], format='%d.%m.%Y')
2 deals['Closing Date']
```

*Closing Date с пустыми значениями заполнила несуществующей датой:
01.01.2025 *

```
[ ] 1 deals['Closing Date'] = deals['Closing Date'].fillna('01.01.2025')
2 deals['Closing Date']
```

Привела к формату дата без времени, в столбце Closing Date пустые
значения заполнила несуществующей датой 01.01.2025



Столбец SLA

```

1 deals['SLA'] = deals['SLA'].astype(str)

1 deals['SLA'].unique()

array(['nan', '00:26:43', '01:00:04', ..., '71 days, 0:46:22',
       '56 days, 19:01:59', '4 days, 22:47:14'], dtype=object)

1 deals['SLA'] = deals['SLA'].replace({"null": np.nan})
2 deals['SLA'] = pd.to_timedelta(deals['SLA'])
3 deals['SLA']

```

Выявляем выбросы

```

1 Q1 = deals['SLA'].quantile(0.25)
2 Q3 = deals['SLA'].quantile(0.75)
3 IQR = Q3 - Q1
4 lower_bound = Q1 - 1.5 * IQR
5 upper_bound = Q3 + 1.5 * IQR
6 filtered_deals = deals[(deals['SLA'] >= lower_bound) & (deals['SLA'] <= upper_bound)]
7 print(filtered_deals['SLA'].describe())

count    14121.000000
mean      7.638147
std       7.710334
min      0.000833
25%     1.037778
50%     4.279722
75%    13.695278
max     37.274167
Name: SLA, dtype: float64

1 deals['SLA'] = deals['SLA'].fillna(deals['SLA'].median())

```

```

1 deals['SLA'] = deals['SLA'].dt.total_seconds() / 3600

1 print(deals['SLA'].describe())

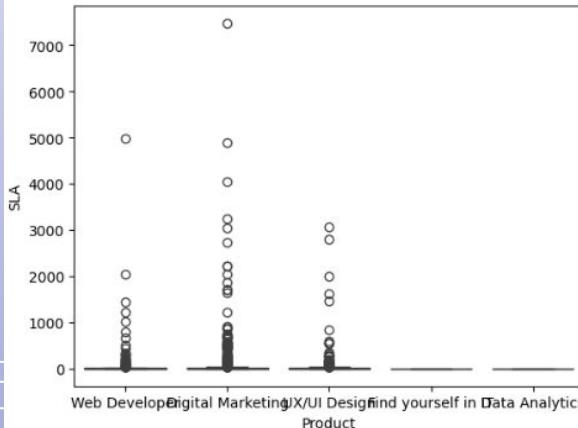
count    15533.000000
mean      32.173951
std       204.792399
min      0.000833
25%     1.216667
50%     5.526111
75%    15.643889
max     7474.573333
Name: SLA, dtype: float64

```

```

1 sns.boxplot(x='Product', y='SLA', data=deals)
2 plt.show()

```



Высокая дисперсия SLA в Digital Marketing:
Данное направление характеризуется наибольшим разбросом значений SLA.

Столбец Quality

QUALITY - убрала буквы. Заполнила пустые NO - неопределенное качество.
Разделила по дефису строку, удалила старый и ненужный столбец с буквой.
Столбец с категорией F - преобразовала и назвала SPECIAL

```
1 deals['Quality'] = deals['Quality'].replace('F', 'F - Special')
2 deals['Quality'] = deals['Quality'].fillna('No')
3 deals['Quality'] = deals['Quality'].astype(str)
4 deals['Quality1'] = deals['Quality'].str.split(r'\s*-|\s*').str[0].str.strip()

1 deals['Quality2'] = deals['Quality'].str.split(r'\s*-|\s*').str[1].fillna('No').str.strip()
2 deals.drop('Quality', axis=1, inplace=True)
3 len(deals)
4 deals.drop('Quality1', axis=1, inplace=True)
```

Столбец Stage и Lost

*ЗАМЕНИЛА ПРОПУЩЕННЫЕ ЗНАЧЕНИЯ В СТОЛБЦЕ *LOST REASON* ЗНАЧЕНИЕМ
"NO REASONS" *

```
1 deals['Lost Reason'] = deals['Lost Reason'].fillna('No reason')
2 deals['Lost Reason']
```

ЗАМЕНА НЕКОРРЕКТНОГО LOST , ГДЕ ПРОШЛА ОПЛАТА НА PAYMENT DONE

```
1 deals.loc[deals['Initial Amount Paid'] > 0, 'Stage'] = 'Payment Done'
2 deals.loc[deals['Initial Amount Paid'] > 0, 'Lost Reason'] = 'No reason'
```

Очистка столбца с уровнем

нем

```
1 deals['Level of Deutsch'] = deals['Level of Deutsch'].str.upper()
2

1 pattern = r"^(A1|A2|B1|B1|B2|B2|C1|C2)$"
2 deals['Level of Deutsch'] = deals['Level of Deutsch'].str.extract(pattern, expand=False)

1 deals['Level of Deutsch'].unique()

array([nan, 'A2', 'B1', 'B1', 'B2', 'C2', 'B2', 'A1', 'C1'], dtype=object)

1 replace_levels = {'B1': 'B1', 'B2': 'B2'}
2 deals['Level of Deutsch'] = deals['Level of Deutsch'].replace(replace_levels)
3 deals['Level of Deutsch'].unique()
4

array([nan, 'A2', 'B1', 'B2', 'C2', 'A1', 'C1'], dtype=object)

1 deals['Level of Deutsch'] = deals['Level of Deutsch'].fillna('Unknown')
2 deals['Level of Deutsch'].unique()

array(['Unknown', 'A2', 'B1', 'B2', 'C2', 'A1', 'C1'], dtype=object)
```



Initial Amount

```
1 deals['Initial Amount Paid'].unique()

array([nan, '0', '1000', '€ 3.500,00', '500', '100', '4500', '300', '200',
       '2000', '11000', '4000', '3000', '3500', '11500', '1200', '1500',
       '1', '5000', '600', '700', '350', '9', '400', '450'], dtype=object)

1 deals['Initial Amount Paid'] = np.where(deals['Initial Amount Paid'] == '€ 3.500,00', 3500, deals['Initial Amount Paid'])
2 deals['Initial Amount Paid'] = deals['Initial Amount Paid'].fillna('0').astype(str)
3 deals['Initial Amount Paid'] = deals['Initial Amount Paid'].str.replace(r'^\d', '', regex=True) #пытаюсь убрать все-все лишнее с 10й попытки, в данном случае все символы кроме цифр

1 deals['Initial Amount Paid'] = pd.to_numeric(deals['Initial Amount Paid'], errors='coerce')

1 deals['Initial Amount Paid'].unique()

array([    0,   1000,   3500,     500,    100,   4500,    300,    200,   2000,
       11000,   4000,   3000,  11500,   1200,   1500,      1,   5000,   600,
       700,    350,      9,   400,   450])

1 deals['Offer Total Amount'].unique()

array([nan, '2000', '9000', '11000', '3500', '4500', '€ 2.900,00', '6500',
       '4000', '3000', '10000', '2500', '5000', '11500', '1', '1000',
       '1200', '0', '1500', '€ 11398,00', '11111', '6000'], dtype=object)

1 target = [1, 9]
2 filtered_deals = deals[deals['Initial Amount Paid'].isin(target)]
```

Offer Total Amount

```
1 deals['Offer Total Amount'] = np.where(deals['Offer Total Amount'] == '€ 2.900,00', 2900, deals['Offer Total Amount'])
2 deals['Offer Total Amount'] = np.where(deals['Offer Total Amount'] == '€ 11398,00', 11398, deals['Offer Total Amount'])
3 deals['Offer Total Amount'] = deals['Offer Total Amount'].fillna('0').astype(str)
4 deals['Offer Total Amount'] = deals['Offer Total Amount'].str.replace(r'[^\d]', '', regex=True)
```

```
1 deals['Offer Total Amount'] = pd.to_numeric(deals['Offer Total Amount'], errors='coerce')
2
```

```
1 deals['Offer Total Amount'].unique()

array([    0,  2000,  9000, 11000,  3500,  4500,  2900,  6500,  4000,
       3000, 10000,  2500,  5000, 11500,      1,  1000,  1200,  1500,
      11398, 11111,  6000])
```

```
1 deals['Offer Total Amount'] = np.where(
2   (deals['Offer Total Amount'] > 0) &
3   (deals['Offer Total Amount'] != 2900) &
4   (deals['Offer Total Amount'] != 11398),
5   (deals['Offer Total Amount'] * 1.04).astype(int),
6   deals['Offer Total Amount']
7 )
```

```
1 target2 = [1]
2 filtered_deals2 = deals[deals['Offer Total Amount'].isin(target)]
3 filtered_deals2
```

Cleaning strategy. Table ‘Deals’



Удаление строк, где нет РК и
1ый платеж=0 и нет **Offers**,
платежей и нет РК



Заменила ячейки с оплатами,
если > **Offer**



Заменила время



Заполнила ячейки, где не
указны м-цы обучения по
условиям



Почистила города



Удалила строки, с 90%ой
схожестью



Заменила значения
PAYMENT TYPE согласно
условиям



Некорректный **Lost** заменила
на **Payment Done**



Заменила **Course Duration**, согласно условиям

Чистка по комбинациям

Где отсутствуют платежи и нет РК - удаляю

```
1 nopayment_no_campaign = deals[(deals['Campaign'] == 'No campaign') & (deals['Initial Amount Paid'] == 0) & (deals['Offer Total Amount'] == 0)]
2 nopayment_no_campaign
```

```
1 filtered_rows_to_delete = (deals['Campaign'] == 'No campaign') & (deals['Initial Amount Paid'] == 0) & (deals['Offer Total Amount'] == 0)
2 deals_cleaned = deals[~filtered_rows_to_delete].reset_index(drop=True)
3 deals_cleaned
4 deals = deals_cleaned
5
6
```

```
1 nopayment_no_campaign2 = deals[(deals['Campaign'] == 'No campaign') & (deals['Initial Amount Paid'] == 0)]
2 nopayment_no_campaign
```

```
1 filtered_rows_to_delete = (deals['Campaign'] == 'No campaign') & (deals['Initial Amount Paid'] == 0)
2 deals_cleaned = deals[~filtered_rows_to_delete].reset_index(drop=True)
3 deals_cleaned
4 deals = deals_cleaned
5 deals
```

Ошибки в заполнении полей

▼ ПРОВЕРЯЮ ПЕРЕПУТАННОЕ ЗАПОЛНЕНИЕ ОПЛАТ

меняю местами оффер и первую оплату там где оффер не равен 0

```
1 payment_mist = deals[(deals['Offer Total Amount']) < (deals['Initial Amount Paid'])]
2 payment_mist
```

```
[ ] 1 condition = (deals['Offer Total Amount'] < deals['Initial Amount Paid']) & (deals['Offer Total Amount'] != 0)
2 deals.loc[condition, ['Offer Total Amount', 'Initial Amount Paid']] = deals.loc[condition, ['Initial Amount Paid', 'Offer Total Amount']].values
3 deals
```

Столбец City

- ✗ CITY. Обнаружила схожие строки Проверяю сколько таких и удаляю дубли

```
[ ] 1 duplicate_rows = deals[deals.duplicated(subset=['City', 'Payment Type', 'Term', 'Campaign', 'Stage', 'Content', 'Deal Owner Name', 'Quality2', 'Stage', 'Lost Reason', 'Created Time', 'Closing Date', 'SLA', 'Page', 'Initial Amount Paid', 'Offer Total Amount' ], keep=False)]
2
3 duplicate_rows
```

- ✗ ЧИСТКА НАЗВАНИЯ ГОРОДА. Убираю полный адрес и пояснения в скобках.

столкнулась с тем, что есть клиент, который зарегистрировался на 2х курсах. И оплата прошла. все одинаково. Предположила, что он обучается на обоих курсах

```
[ ] 1 address_del = deals['City'].replace({'Karl-Liebknecht str. 24, Hildburghausen, Thüringen':'Thüringen','Halle (Saale)':'Halle', 'Vor Ebersbach 1, 77761 Schiltach': 'Schiltach',
2           'Poland , Gdansk , Al. Grunwaldzka 7, ap. 1a': 'Gdańsk'})
3 deals['City'] = address_del
```

Payment Type

МЕНЯЮ ЗНАЧЕНИЯ В СТОЛБЦЕ PAYMENT TYPE С УКАЗАННЫМИ НИЖЕ УСЛОВИЯМИ

```
1 deals['Payment Type'] = np.where(  
2     (deals['Payment Type'] == 'Reservation'), deals['Payment Type'],  
3     np.where(  
4         (deals['Offer Total Amount'] == 0) & (deals['Initial Amount Paid'] == 0), 'No Payments',  
5         np.where(  
6             (deals['Initial Amount Paid'] == 0), 'No Payments',  
7             np.where(  
8                 ((deals['Offer Total Amount'] >= deals['Initial Amount Paid']) &  
9                  (deals['Offer Total Amount'] - deals['Initial Amount Paid'] > 200)) |  
10                ((deals['Offer Total Amount'] < deals['Initial Amount Paid']) &  
11                  (deals['Initial Amount Paid'] - deals['Offer Total Amount'] > 200)),  
12                 'Recurring Payments',  
13                 'One Payment'  
14             )  
15         )  
16     )  
17 )
```

Если нет начальной суммы платежа, то "No Payment",

Если Initial больше 200 и либо общая Offer больше Initial, либо наоборот, то "Recurring"

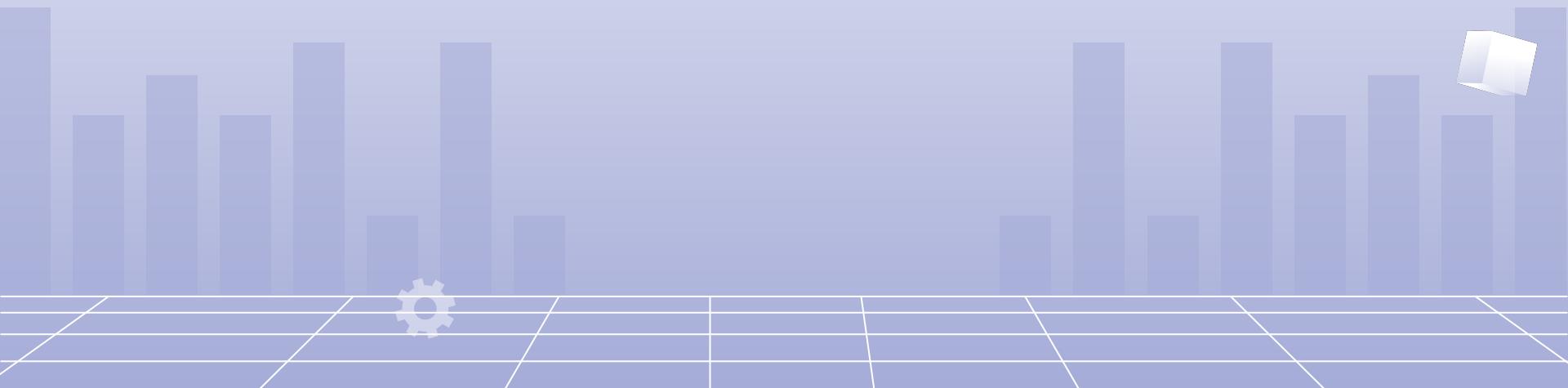
В остальных случаях = One Payment



Course Duration

```
1 deals.loc[(deals['Course duration'] == 0) & (deals['Initial Amount Paid'] == 500), 'Course duration'] = 6
2 deals.loc[(deals['Course duration'] == 0) & (deals['Initial Amount Paid'] == 1000), 'Course duration'] = 6
3 deals.loc[(deals['Course duration'] == 0) & (deals['Initial Amount Paid'] == 2000), 'Course duration'] = 11
4 deals.loc[(deals['Course duration'] == 1), 'Course duration'] = 11
```

```
1 deals['Course duration'] = pd.to_numeric(deals['Course duration'], errors='coerce') # была замечена ячейка с менеджером Ulysses Adams вместо кол-венного значения. заменено на 11
2 deals['Course duration'] = deals['Course duration'].replace({'Ulysses Adams': '11'})
3 deals['Course duration'].astype(int)
4 quest = deals.loc[deals['Course duration'] == 'Ulysses Adams'].index
5
```



Ошибки в данных

Где дельта отрицательная, создаем маску и меняем местами даты (я предположила, что это ошибочное заполнение)

```
[ ] 1 check_time = deals['Closing Date'] - deals['Created Time']
2 negative_check_time = deals[check_time < pd.Timedelta(0)]
3 negative_check_time
```

>Show hidden output

```
[ ] 1 mask_check = check_time < pd.Timedelta(0)
2 deals.loc[mask_check, ['Closing Date', 'Created Time']] = deals.loc[mask_check, ['Created Time', 'Closing Date']]
```

УДАЛИЛА ДВЕ СТРОКИ ВЫБРОСЫ С ОТСУТСТВИЕМ ИНФОРМАЦИИ, ОПЛАТЫ,

в 2022 году и те, что были созданы в январе 24 и закрыты в декабре

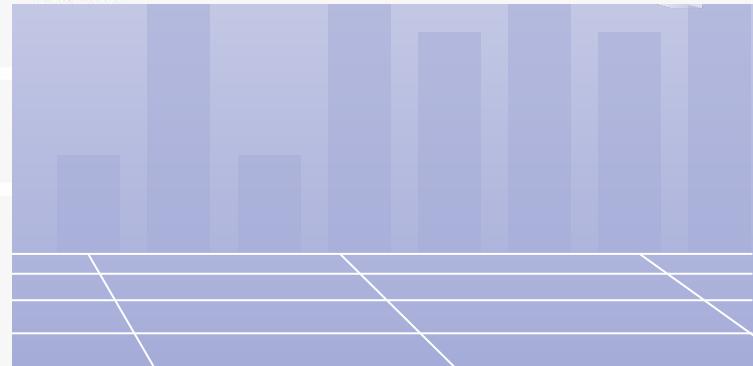
24

```
[ ] 1 mistakes = deals[deals['Closing Date'] < deals['Created Time']]
2 deals.loc[mistakes.index, ['Closing Date', 'Created Time']] = deals.loc[mistakes.index, ['Created Time', 'Closing Date']].values
3
```

```
[ ] 1 early_deals = deals[deals['Created Time'].dt.year == 2022]
2 early_deals
3 deals = deals.drop(early_deals.index)
4
```

```
[ ] 1 filtered_deals = deals[
2     (deals['Created Time'].dt.month == 1) &
3     (deals['Created Time'].dt.year == 2024) &
4     (deals['Closing Date'].dt.month == 12) &
5     (deals['Closing Date'].dt.year == 2024)
6 ]
7 filtered_deals
8 deals = deals.drop(filtered_deals.index)
```

```
[ ] 1 mistake = deals['Closing Date'] < deals['Created Time']
2 mistake.unique()
3 mistakes = deals[deals['Closing Date'] < deals['Created Time']]
4 mistakes
```



Заполнение пустых значений в Month of

ЗАПОЛНЯЮ ПУСТЫЕ ЯЧЕЙКИ, ГДЕ НЕ УКАЗАНЫ МЕСЯЦЫ ОБУЧЕНИЯ.

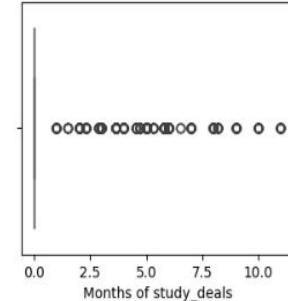
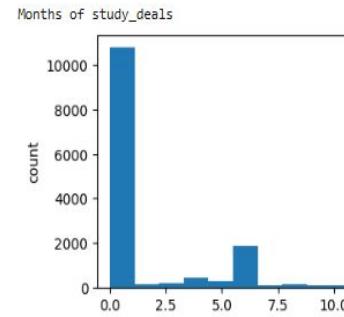
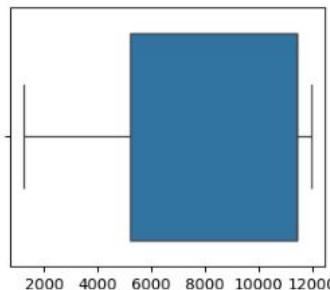
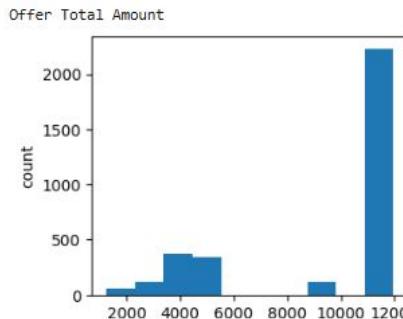
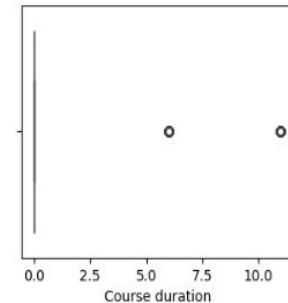
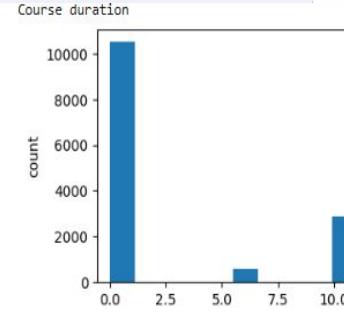
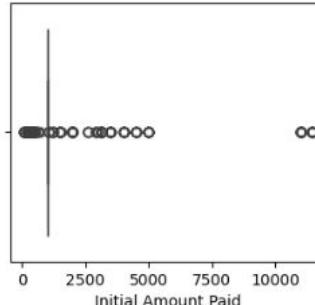
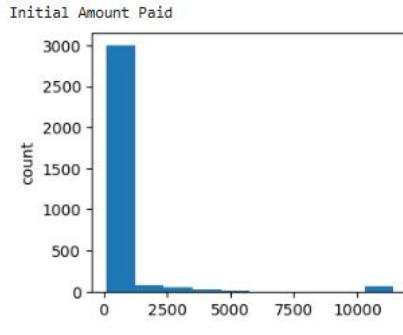
ГРУППИРУЮ Initial and Offer И СЧИТАЮ СРЕДНЕЕ В ИХ ГРУППАХ. Использую функцию cut, которая создает категории по группам

```
[1] 1 filtered_deals = deals[(deals['Initial Amount Paid'] > 0) & (deals['Offer Total Amount'] > 0)]
2 bins_initial = [0, 1000, 2000, 5000, 8000, 12000]
3 labels_initial = ['1-1000', '1001-2000', '2001-5000', '5001-8000', '8001-11440' ]
4
5 bins_offer = [0, 3000, 4000, 5000, 10000, 12000]
6 labels_offer = ['1-3000', '3001-4000', '4001-5000', '5001-10000', '10001-12000']
7
8 filtered_deals['Initial Group'] = pd.cut(filtered_deals['Initial Amount Paid'], bins=bins_initial, labels=labels_initial)
9 filtered_deals['Offer Group'] = pd.cut(filtered_deals['Offer Total Amount'], bins=bins_offer, labels=labels_offer)
10 filtered_deals.head()

1 group_means = filtered_deals[filtered_deals['Months of study'] != 0].groupby(['Initial Group', 'Offer Group'])['Months of study'].mean().reset_index()
2
3 deals['Initial Group'] = pd.cut(deals['Initial Amount Paid'], bins=bins_initial, labels=labels_initial)
4 deals['Offer Group'] = pd.cut(deals['Offer Total Amount'], bins=bins_offer, labels=labels_offer)
5
6 deals = deals.merge(group_means, on=['Initial Group', 'Offer Group'], how='left', suffixes=('_deals', '_mean'))
7 deals.loc[
8     (deals['Months of study_deals'] == 0) & (deals['Initial Amount Paid'] > 0) & (deals['Offer Total Amount'] > 0), 'Months of study_deals'] = deals.loc[
9     (deals['Months of study_deals'] == 0) & (deals['Initial Amount Paid'] > 0) & (deals['Offer Total Amount'] > 0), 'Months of study_mean']
```

Изучение данных. Одномерный

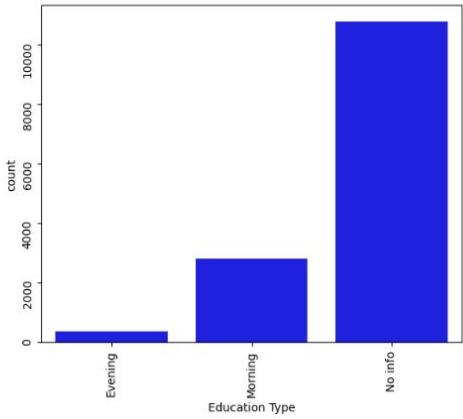
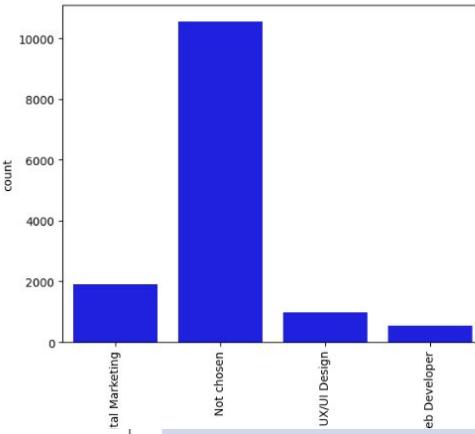
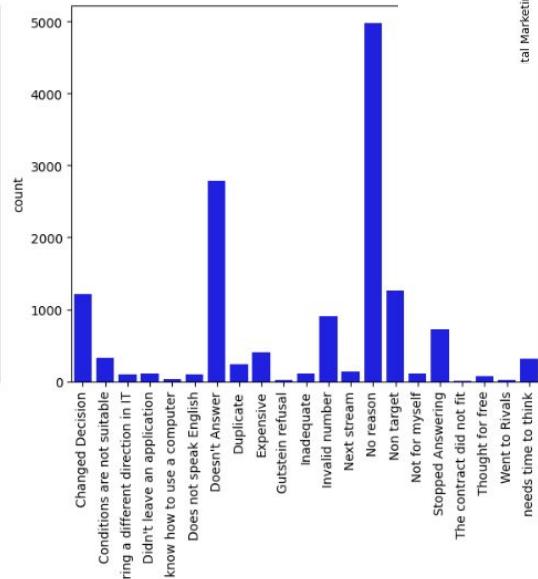
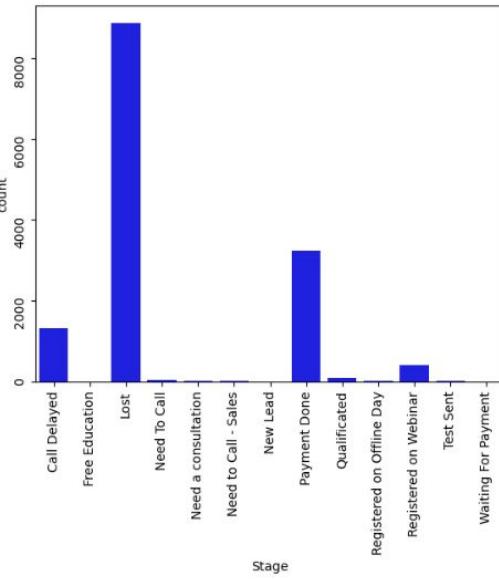
а



В процессе обнаружила ошибки в Course Duration - исправила.



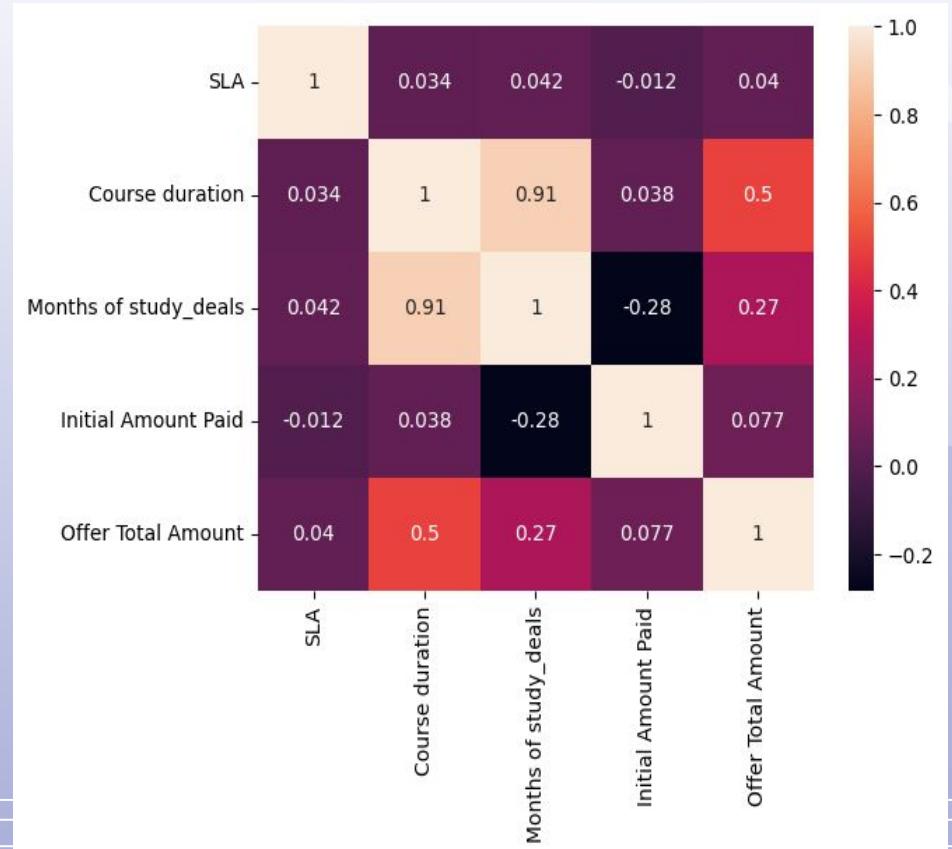
Категориальный анализ





Многомерный анализ

- Высокая положительная корреляция между "Course duration" (длительностью курса) и "Months of study_deals" (количеством месяцев обучения)
- Слабая положительная корреляция между "Course duration" и "Offer Total Amount"
- Слабая отрицательная корреляция между "Months of study_deals" и "Initial Amount Paid"





Структура данных

Таблица Spend

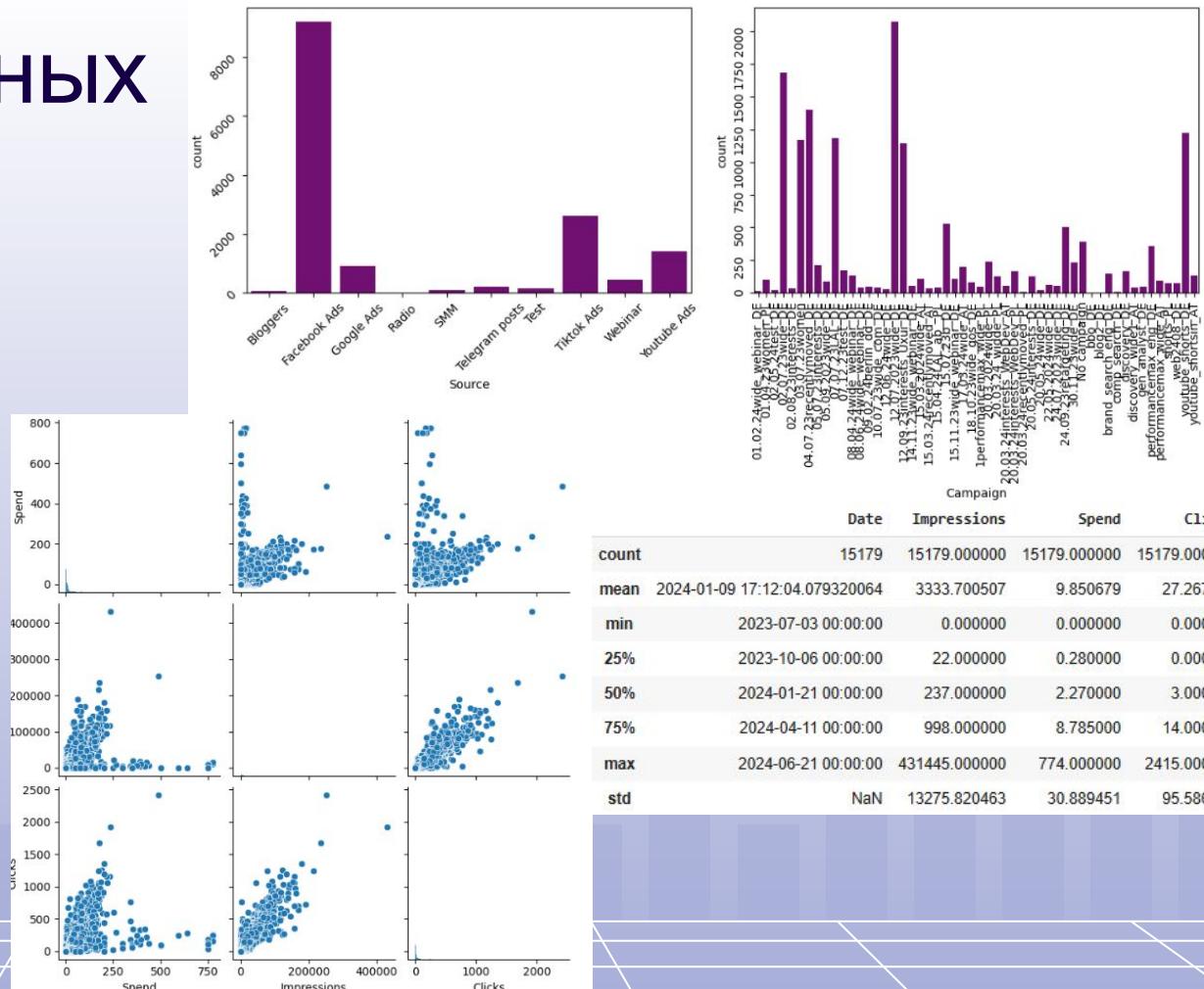
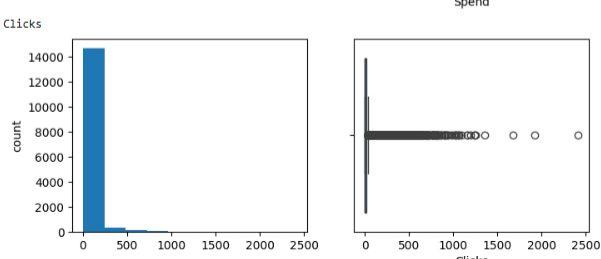
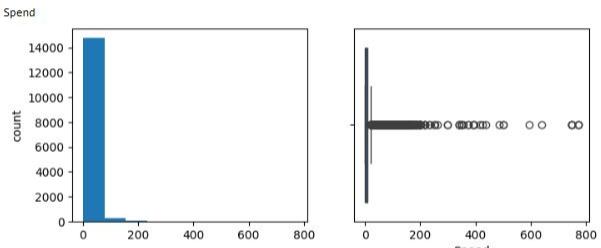
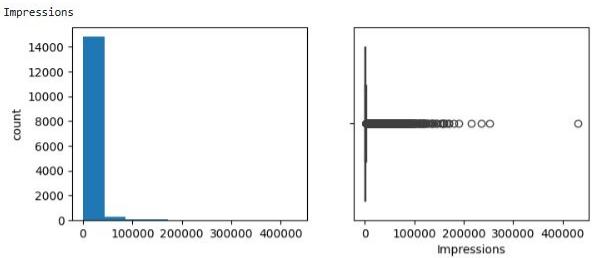
```
1 spent.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20779 entries, 0 to 20778
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date         20779 non-null   datetime64[ns]
 1   Source        20779 non-null   object  
 2   Campaign      14785 non-null   object  
 3   Impressions   20779 non-null   int64   
 4   Spend         20779 non-null   float64 
 5   Clicks        20779 non-null   int64   
 6   AdGroup       13951 non-null   object  
 7   Ad            13951 non-null   object  
dtypes: datetime64[ns](1), float64(1), int64(2), object(4)
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 15179 entries, 0 to 20778
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date         15179 non-null   datetime64[ns]
 1   Source        15179 non-null   category
 2   Campaign      15179 non-null   category
 3   Impressions   15179 non-null   int64   
 4   Spend         15179 non-null   float64 
 5   Clicks        15179 non-null   int64  
dtypes: category(2), datetime64[ns](1), float64(1), int64(2)
```

Изучение данных



Структура данных

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95874 entries, 0 to 95873
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Id               95874 non-null   Int64  
 1   Call Start Time  95874 non-null   object  
 2   Call Owner Name  95874 non-null   object  
 3   CONTACTID        91941 non-null   Int64  
 4   Call Type         95874 non-null   object  
 5   Call Duration (in seconds) 95791 non-null   float64 
 6   Call Status       95874 non-null   object  
 7   Dialled Number   0 non-null      float64 
 8   Outgoing Call Status 86875 non-null   object  
 9   Scheduled in CRM 86875 non-null   float64 
 10  Tag              0 non-null      float64 
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18548 entries, 0 to 18547
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Id               18548 non-null   Int64  
 1   Contact Owner Name 18548 non-null   object  
 2   Created Time     18548 non-null   object  
 3   Modified Time    18548 non-null   object  
dtypes: Int64(1), object(3)
memory usage: 597.9+ KB
```



Таблица Calls, Contacts

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95874 entries, 0 to 95873
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Id               95874 non-null   Int64  
 1   Call Start Time  95874 non-null   datetime64[ns]
 2   Call Owner Name  95874 non-null   category 
 3   CONTACTID        95874 non-null   Int64  
 4   Call Type         95874 non-null   category 
 5   Call Duration (in seconds) 95874 non-null   float64 
 6   Call Status       95874 non-null   category 
 7   Outgoing Call Status 95874 non-null   category 
 8   Scheduled in CRM 95874 non-null   category 
dtypes: Int64(2), category(5), datetime64[ns](1), float64(1)
```



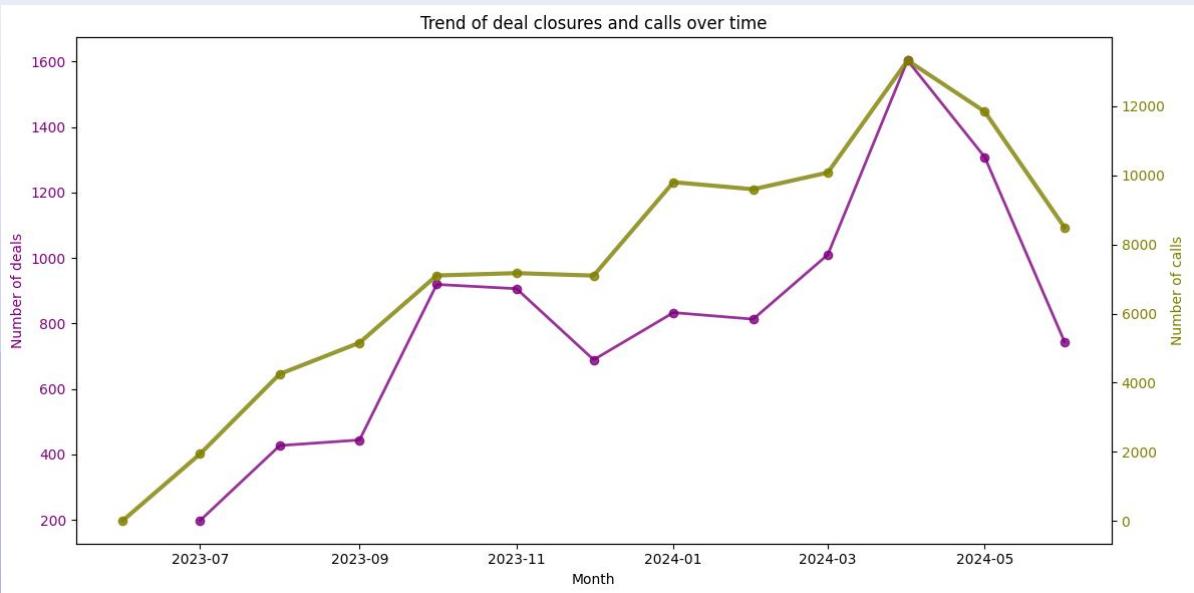
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18548 entries, 0 to 18547
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Id               18548 non-null   Int64  
 1   Contact Owner Name 18548 non-null   object  
 2   Created Time     18548 non-null   datetime64[ns]
 3   Modified Time    18548 non-null   datetime64[ns]
dtypes: Int64(1), datetime64[ns](2), object(1)
memory usage: 597.9+ KB
```

Анализ данных!

Here will be many useful information and graphics...



Анализ временных рядов



```
deals_filtered = deals[deals['Closing Date'] !=  
pd.to_datetime('2025-01-01')]  
deals_by_month =  
deals_filtered.groupby(deals_filtered['Closing  
Date'].dt.to_period('M')).size()
```

```
calls_by_month = call.groupby(call['Call Start  
Time'].dt.to_period('M')).size()
```

```
deals_by_month.index = deals_by_month.index.to_timestamp()  
calls_by_month.index = calls_by_month.index.to_timestamp()
```

```
fig, ax1 = plt.subplots(figsize=(12, 6))
```

```
ax1.plot(deals_by_month.index, deals_by_month.values,  
alpha=0.8, color='purple', lw=2, marker='o')  
ax1.set_xlabel('Month')  
ax1.set_ylabel('Number of deals', color='purple')  
ax1.tick_params(axis='y', labelcolor='purple')
```

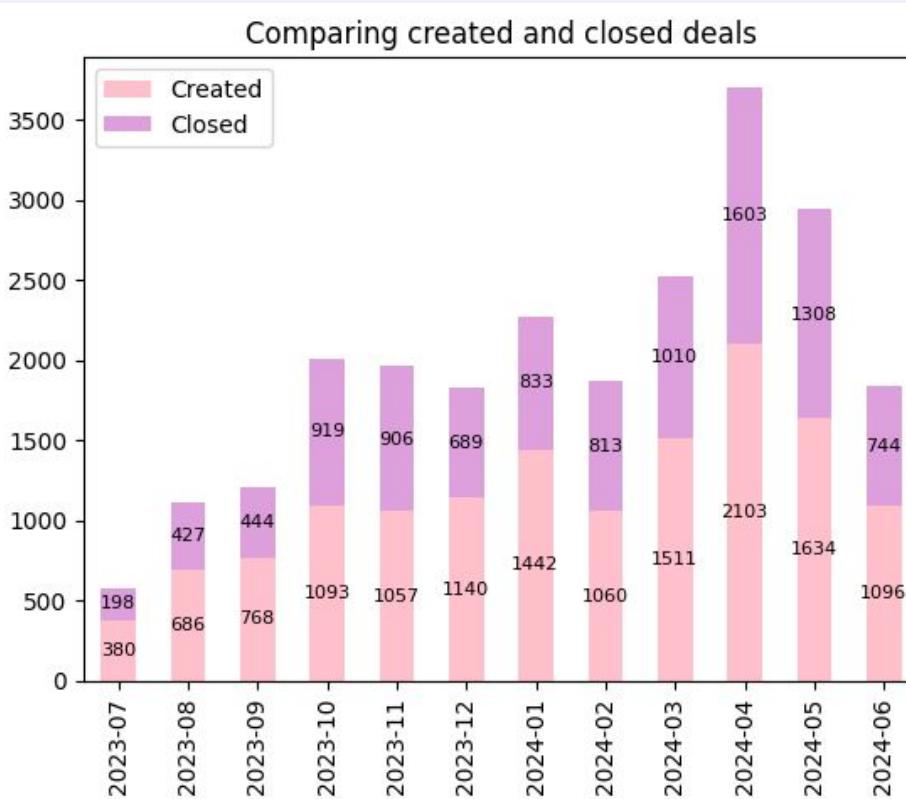
```
ax2 = ax1.twinx()  
ax2.plot(calls_by_month.index, calls_by_month.values,  
alpha=0.8, color='olive', lw=3, marker='o')  
ax2.set_ylabel('Number of calls', color='olive')  
ax2.tick_params(axis='y', labelcolor='olive')
```

```
plt.title('Trend of deal closures and calls over time')
```

```
plt.xticks(rotation=0)  
plt.tight_layout()  
plt.show()
```



Анализ временных рядов



```
deals['Created Month'] = deals['Created Time'].dt.to_period('M')
created_by_month = deals.groupby('Created Month').size()

deals_filtered = deals[deals['Closing Date'] != pd.to_datetime('2025-01-01')]
deals_filtered['Deal Month'] = deals_filtered['Closing Date'].dt.to_period('M')
closed_by_month = deals_filtered.groupby('Deal Month').size()

summary = pd.DataFrame({
    'Created': created_by_month,
    'Closed': closed_by_month}).fillna(0).astype(int)
summary = summary[(summary['Created'] != 0)]
summary['Closed Percentage'] = (summary['Closed'] / summary['Created']) * 100
summary

plt.figure(figsize=(10, 8))
summary[['Created', 'Closed']].plot(kind='bar', stacked=True, color=['pink', 'plum'])

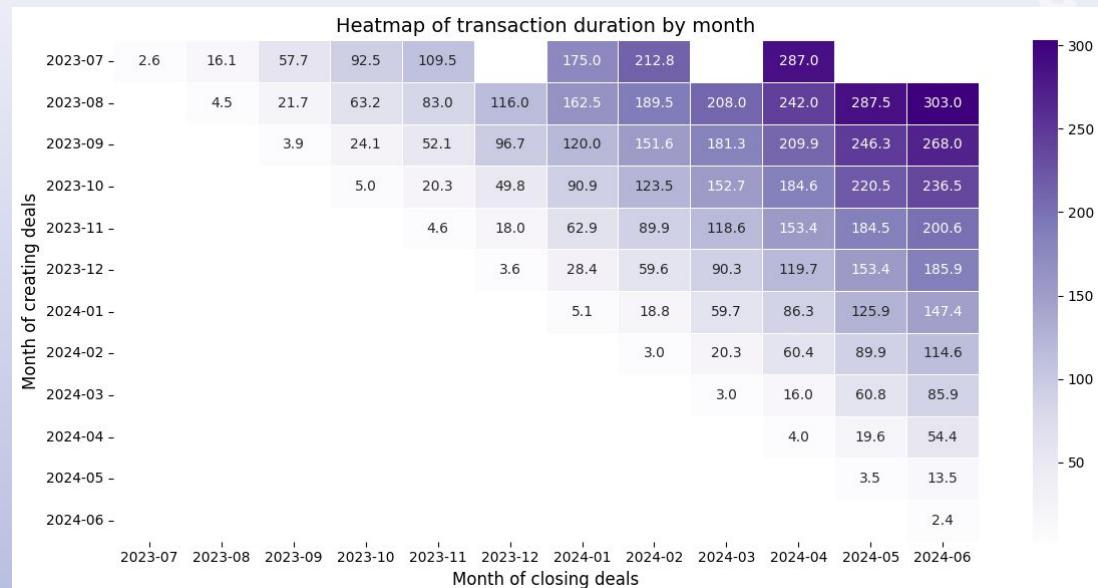
for container in plt.gca().containers:
    plt.bar_label(container, label_type='center', fontsize=8, color='black')

plt.title('Comparing created and closed deals')
plt.show()
```



Анализ временных рядов

```
deals_filtered['Duration']=(deals_filtered['Closing Date']-  
deals_filtered['Created Time']).dt.days  
duration_by_month = deals_filtered.groupby('Created  
Month')['Duration'].mean()  
  
heatmap_data = deals_filtered.groupby(['Created Month', 'Deal  
Month'])['Duration'].mean().unstack()  
  
plt.figure(figsize=(12, 6))  
sns.heatmap(heatmap_data, annot=True, cmap='Purples', fmt='.1f',  
linewidths=0.5)  
  
plt.title('Heatmap of transaction duration by month', fontsize=14)  
plt.xlabel('Month of closing deals', fontsize=12)  
plt.ylabel('Month of creating deals', fontsize=12)  
plt.tight_layout()  
  
plt.show()
```



```
call['Call Start Time']=pd.to_datetime(call['Call Start Time'])  
deals['Closing Date']=pd.to_datetime(deals['Closing Date'])
```

```
deals['Successful']=(deals['Initial Amount Paid']>0).astype(int)  
call['Call Month']=call['Call Start Time'].dt.to_period('M')  
calls_by_month=call.groupby('Call Month').size()
```

```
deals_filtered=deals[deals['Closing Date']!=  
pd.to_datetime('2025-01-01')]  
deals_filtered['Deal Month']=deals_filtered['Closing  
Date'].dt.to_period('M')  
deals_by_month=deals_filtered.groupby('Deal Month').agg(
```

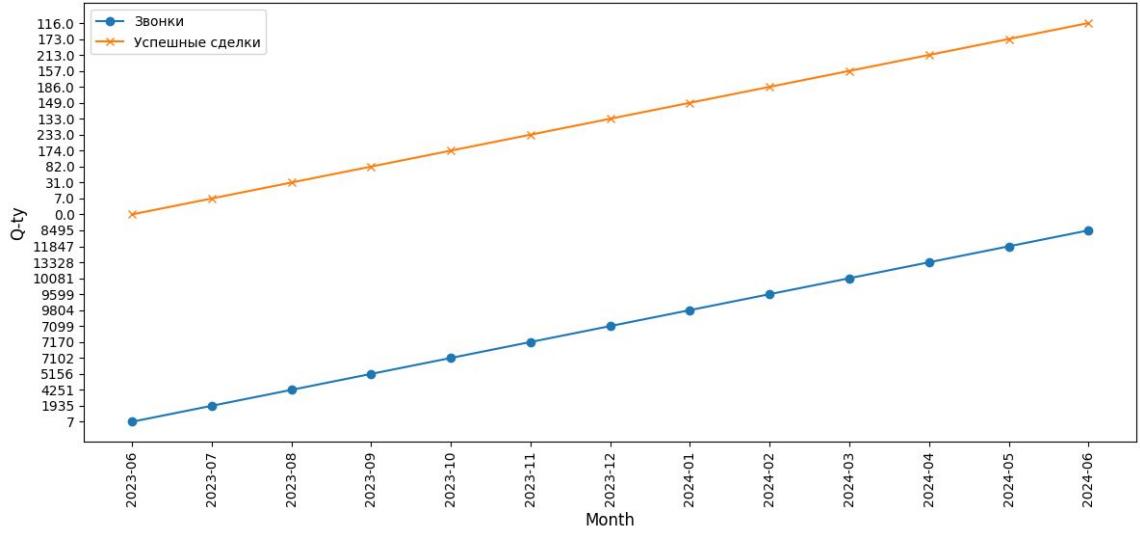
```
total_deals=('Deal Month', 'size'),  
successful_deals='Successful', sum),  
min_closing_date='Closing Date', min),  
max_closing_date='Closing Date', max))  
calls_by_month_times=call.groupby('Call Month').agg(  
min_call_time='Call Start Time', min),  
max_call_time='Call Start Time', max))
```

```
summary=pd.DataFrame({  
'Calls': calls_by_month,  
'Successful Deals': deals_by_month['successful_deals'],  
'Min Call Time': calls_by_month_times['min_call_time'],  
'Max Call Time': calls_by_month_times['max_call_time'],  
'Min Closing Date': deals_by_month['min_closing_date'],  
'Max Closing Date': deals_by_month['max_closing_date']  
}).fillna(0).astype(str)  
plt.figure(figsize=(12, 6))  
plt.plot(summary.index.astype(str), summary['Calls'], label='Звонки',  
marker='o')
```

```
plt.plot(summary.index.astype(str), summary['Successful Deals'],  
label='Успешные сделки', marker='x')  
plt.title('Dependence of calls and successful transactions by month',  
fontsize=16)
```



Dependence of calls and successful transactions by month



Между звонками и успешными сделками существует линейная зависимость. Это означает, что с каждым дополнительным звонком количество успешных сделок увеличивается примерно на одно и то же значение. Это говорит о стабильных процессах продаж

Выводы

- Есть положительная корреляция между числом звонков и закрытыми сделками. Это подчеркивает важность увеличения взаимодействия с клиентами для достижения более высоких продаж!!!
- Некоторые сделки закрываются в течение нескольких дней, другие делятся несколько месяцев.
- Влияние месяца создания:
Сделки, созданные в начале года, закрываются быстрее, чем сделки, созданные ближе к концу года. Возможно, это связано с изменением темпов работы сотрудников.
- В отдельных месяцах количество создаваемых сделок превышает число закрытых. Это может свидетельствовать: О накоплении необработанных сделок, недостаточной производительности.

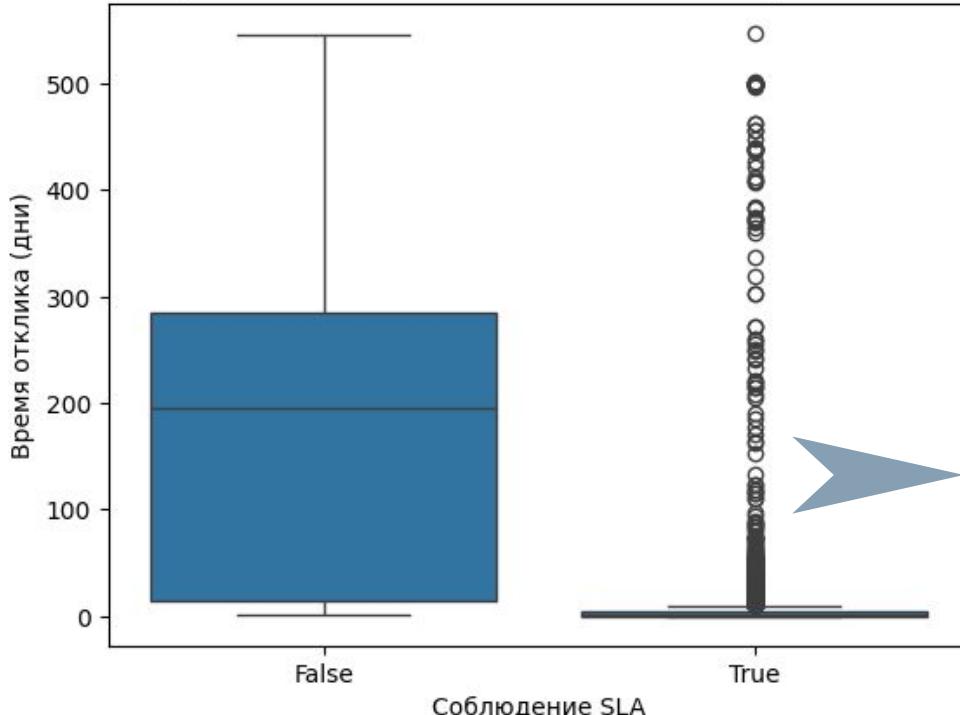




SLA и время закрытия сделки



Время отклика в зависимости от соблюдения SLA



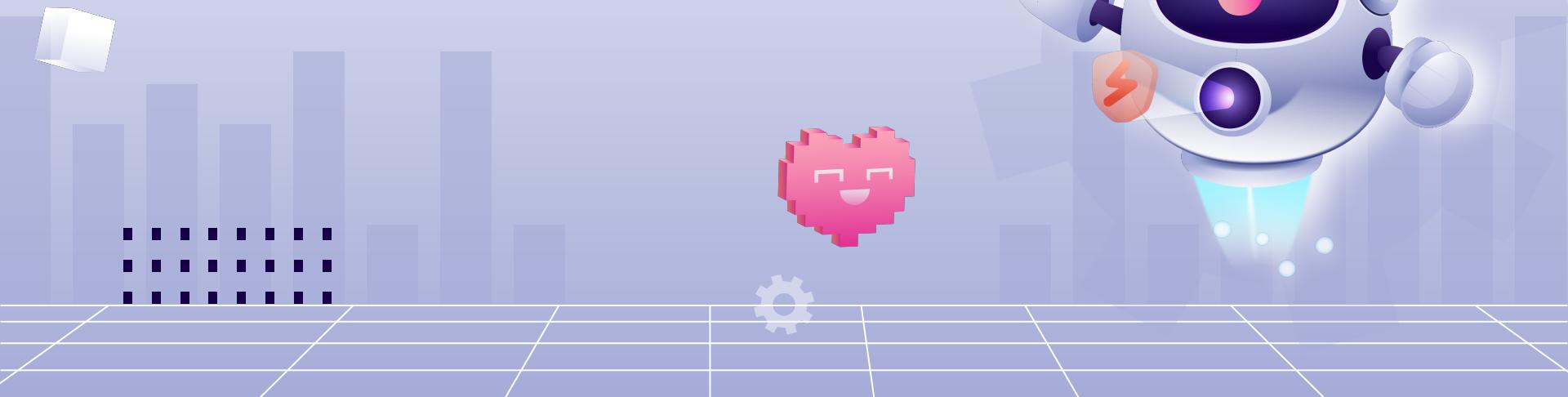
```
deals['Response Time'] = (deals['Closing Date'] - deals['Created Time']).dt.days
```

```
deals['Response Within SLA'] = deals['Response Time'] <= deals['SLA']
```

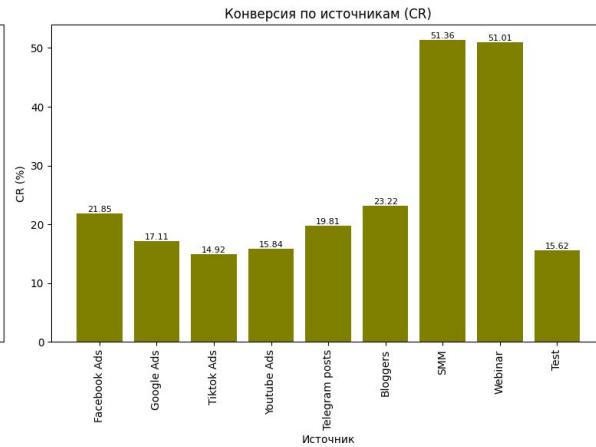
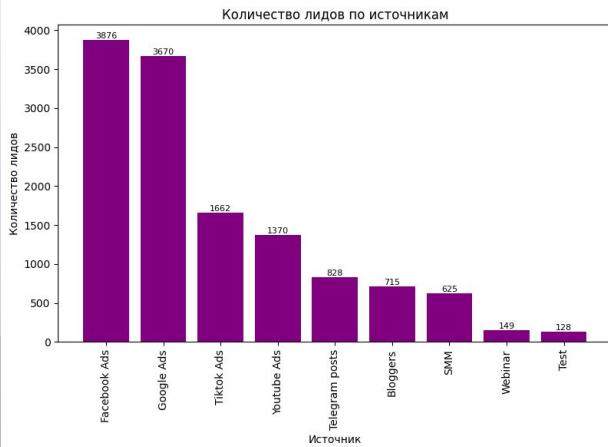
```
sns.boxplot(data=deals, x='Response Within SLA',  
y='Response Time')  
plt.title('Время отклика в зависимости от соблюдения SLA')  
plt.xlabel('Соблюдение SLA')  
plt.ylabel('Время отклика (дни)')  
plt.show()
```

Для случаев, когда SLA не соблюдалось, характерен большой разброс данных, а вот при соблюдении - время закрытия сделки меньше. График показывает, что при соблюдении SLA (значение "True") среднее время отклика значительно меньше, чем при его нарушении (значение "False"). Это свидетельствует о том, что соблюдение SLA положительно влияет на скорость обработки запросов.

Анализ РК



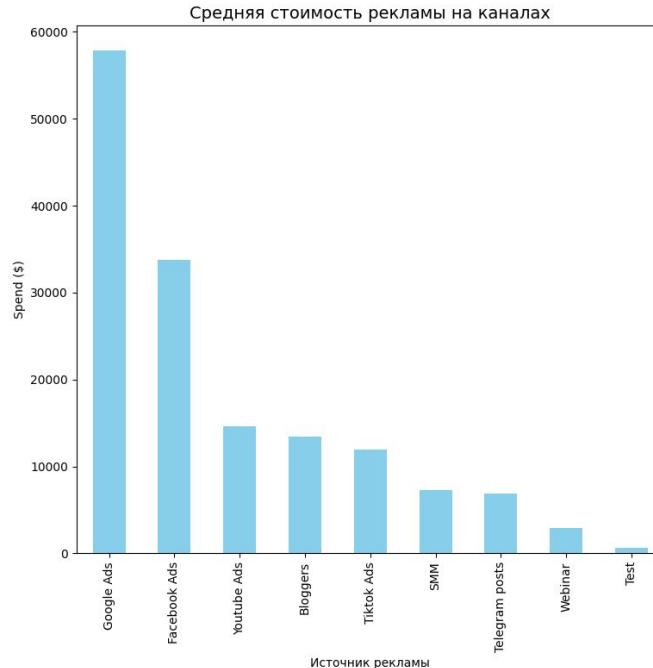
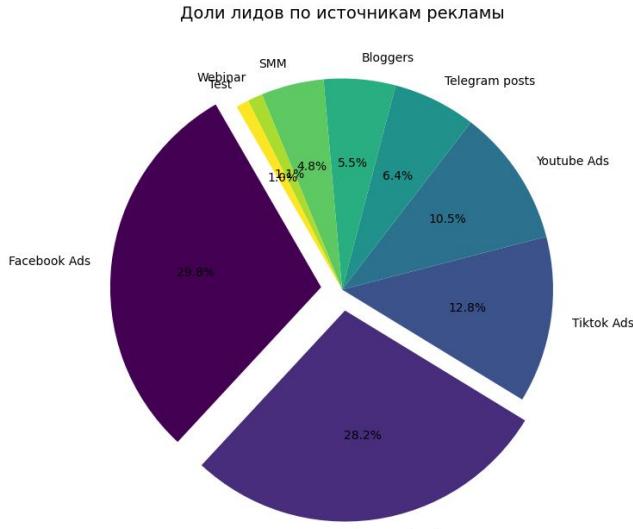
Анализ рекламных каналов



```
campaign_summary_sorted =  
campaign_summary.sort_values(by='Contact Name',  
ascending=False)  
  
fig, axes = plt.subplots(ncols=2, figsize=(16, 6))  
  
axes[0].bar(campaign_summary_sorted['Source'],  
campaign_summary_sorted['Contact Name'], color='purple')  
axes[0].set_title('Количество лидов по источникам')  
axes[0].set_xlabel('Источник')  
axes[0].set_ylabel('Количество лидов')  
axes[0].tick_params(axis='x', rotation=90)  
for container in axes[0].containers:  
    axes[0].bar_label(container,  
labels=campaign_summary_sorted['Contact Name'],  
fontsize=8, color='black')  
  
axes[1].bar(campaign_summary_sorted['Source'],  
campaign_summary_sorted['CR'], color='olive')  
axes[1].set_title('Конверсия по источникум (CR)')  
axes[1].set_xlabel('Источник')  
axes[1].set_ylabel('CR (%)')  
axes[1].tick_params(axis='x', rotation=90)  
for container in axes[1].containers:  
    axes[1].bar_label(container,  
labels=campaign_summary_sorted['CR'], fontsize=8,  
color='black')  
  
plt.tight_layout()  
plt.show()
```



Анализ рекламных каналов



```
fig, axes = plt.subplots(1, 2, figsize=(16, 8))
```

```
explode =(0.1, 0.1, 0, 0, 0, 0, 0, 0, 0)
campaign_summary_sorted.set_index('Source')['Cont
act Name'].plot(
    kind='pie',
    autopct='%.1f%%',
    startangle=120,
    cmap='viridis',
    explode=explode,
    ax=axes[0] )
```

```
axes[0].set_title('Доли лидов по источникам
рекламы', fontsize=14)
axes[0].set_ylabel("")
```

```
average_cost =
campaign_summary_sorted.groupby('Source')['Spend'
].mean().sort_values(ascending=False)
average_cost.plot(
    kind='bar',
    color='skyblue',
    ax=axes[1])
```

```
axes[1].set_title('Средняя стоимость рекламы на
каналах', fontsize=14)
axes[1].set_xlabel('Источник рекламы')
axes[1].set_ylabel('Spend ($)')
axes[1].tick_params(axis='x', rotation=90)
```

```
plt.tight_layout()
plt.show()
```



Анализ рекламных каналов

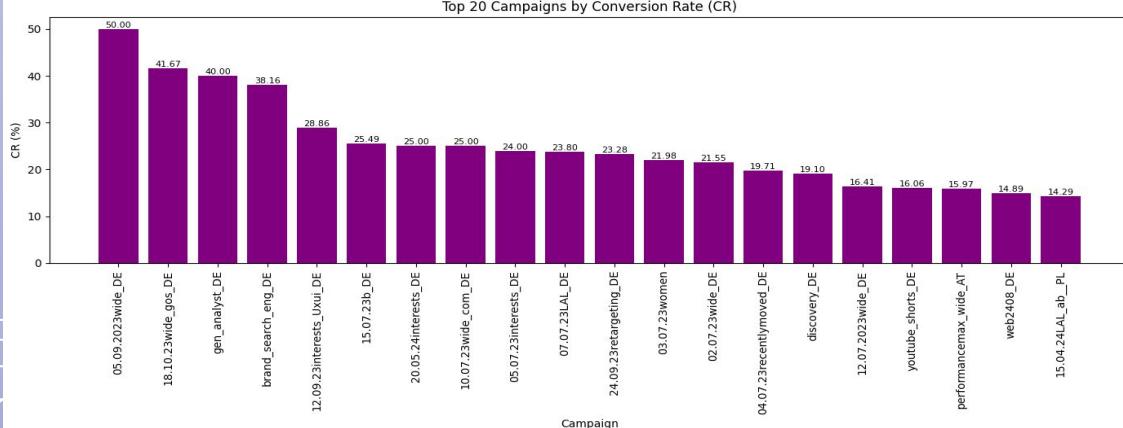
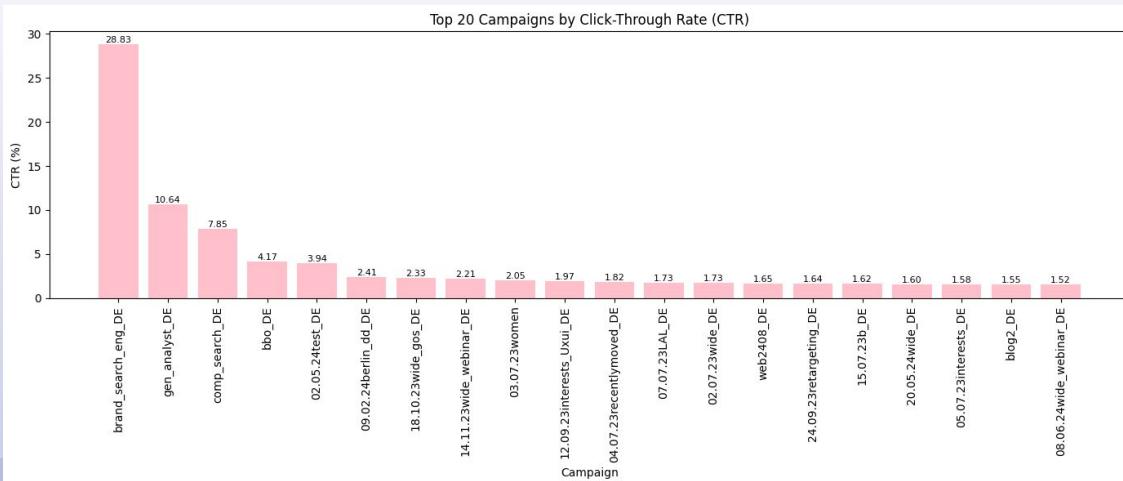


```
correlation_matrix = campaign_summary[['Spend', 'CR',  
'CTR', 'CPC', 'CPL']].corr()
```

```
plt.figure(figsize=(8, 6))  
sns.heatmap(correlation_matrix, annot=True,  
cmap='Purples', fmt='.2f', linewidths=0.5)  
plt.title('Корреляция между показателями рекламных  
кампаний', fontsize=14)  
plt.tight_layout()  
plt.show()
```

- Чем выше кликабельность (CTR), тем выше стоимость за клик (CPC)
- CR и CTR имеют умеренную положительную корреляцию (0.74): Это говорит о том, что более привлекательные объявления – визуально и призыв к действию корректен (высокий CTR) могут приводить к более высокой конверсии (CR)
- Spend и CR имеют слабую положительную корреляцию (0.21): Увеличение затрат на рекламу не всегда приводит к значительному повышению конверсии

ТОП 20 Кампаний



```
plt.figure(figsize=(14, 6))
bars_ctr = plt.bar(top_20_ctr['Campaign'], top_20_ctr['CTR'],
color='pink')
```

```
plt.title('Top 20 Campaigns by Click-Through Rate (CTR)')
plt.xlabel('Campaign')
plt.ylabel('CTR (%)')
plt.xticks(rotation=90)
plt.tight_layout()
```

```
for container in plt.gca().containers:
```

```
    plt.bar_label(container, label_type='edge', fontsize=8, color='black',
fmt=".2f")
```

```
plt.show()
```

```
filter = campaign_summary_by_campaign_sorted_cr['Campaign'] != 'No campaign'
```

```
campaign_summary_by_campaign_sorted_cr =
campaign_summary_by_campaign_sorted_cr[filter]
```

```
top_20_cr = campaign_summary_by_campaign_sorted_cr.head(20)
```

```
plt.figure(figsize=(14, 6))
```

```
bars_cr = plt.bar(top_20_cr['Campaign'], top_20_cr['CR'],
color='purple')
```

```
plt.title('Top 20 Campaigns by Conversion Rate (CR)')
plt.xlabel('Campaign')
plt.ylabel('CR (%)')
plt.xticks(rotation=90)
plt.tight_layout()
```

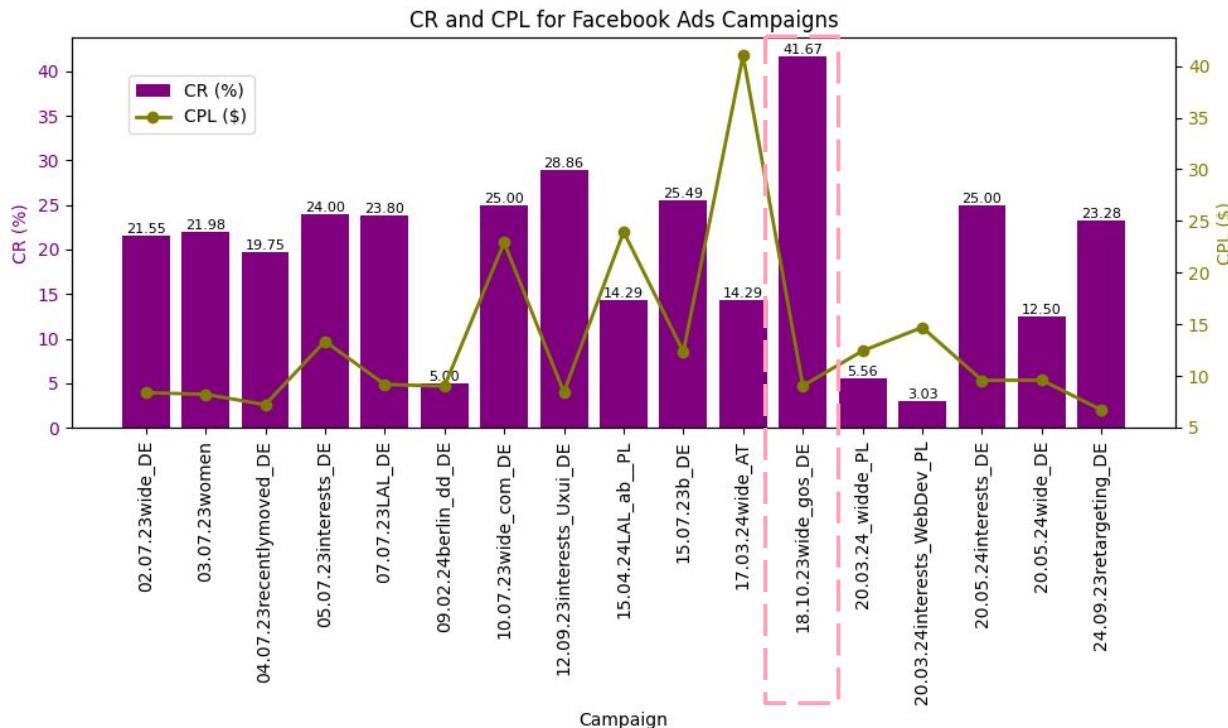
```
for container in plt.gca().containers:
```

```
    plt.bar_label(container, label_type='edge', fontsize=8, color='black',
fmt=".2f")
```

```
plt.show()
```



Анализ кампаний в Facebook



```

facebook_data =
cam_source_summary[cam_source_summary['Source']
]== 'Facebook Ads']
fb_data_filtered =
facebook_data[(facebook_data['CTR'] > 0) &
(facebook_data['CR'] > 0) & (facebook_data['CPL'] > 0)]

fig, ax1 = plt.subplots(figsize=(10, 6))

bars_cr = ax1.bar(fb_data_filtered['Campaign'],
fb_data_filtered['CR'], color='purple', label='CR (%)')
ax1.set_xlabel('Campaign')
ax1.set_ylabel('CR (%)', color='purple')
ax1.tick_params(axis='y', labelcolor='purple')
ax1.bar_label(bars_cr, label_type='edge', fontsize=8,
color='black', fmt=".2f")
ax1.tick_params(axis='x', rotation=90)

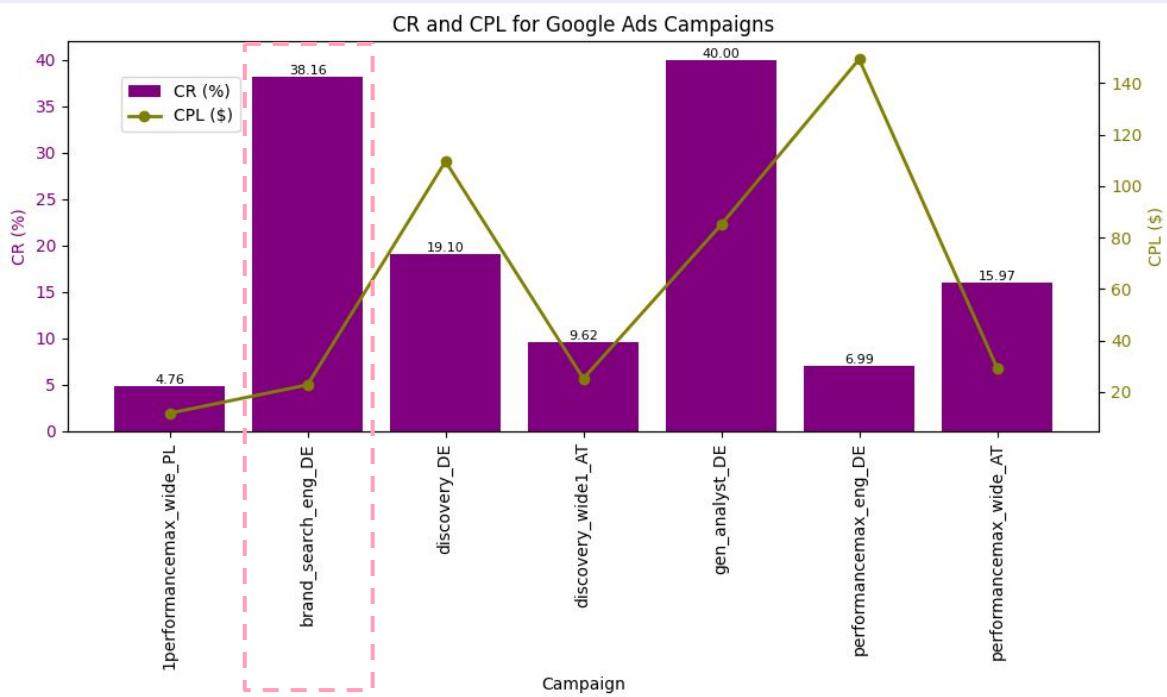
ax2 = ax1.twinx()
ax2.plot(fb_data_filtered['Campaign'],
fb_data_filtered['CPL'], color='olive', marker='o',
label='CPL ($)', linewidth=2)
ax2.set_ylabel('CPL ($)', color='olive')
ax2.tick_params(axis='y', labelcolor='olive')

fig.legend(loc='upper left', bbox_to_anchor=(0.1, 0.9),
fontsize=10)
plt.title('CR and CPL for Facebook Ads Campaigns')
plt.tight_layout()
plt.show()

```



Анализ кампаний PPC



```
google_data =
cam_source_summary[cam_source_summary['Source']
]== 'Google Ads']
g_data_filtered = google_data[(google_data['CTR'] > 0)
& (google_data['CR'] > 0) & (google_data['CPL'] > 0)]
```

```
fig, ax1 = plt.subplots(figsize=(10, 6))
```

```
bars_cr = ax1.bar(g_data_filtered['Campaign'],
g_data_filtered['CR'], color='purple', label='CR (%)')
ax1.set_xlabel('Campaign')
ax1.set_ylabel('CR (%)', color='purple')
ax1.tick_params(axis='y', labelcolor='purple')
ax1.bar_label(bars_cr, label_type='edge', fontsize=8,
color='black', fmt=".2f")
ax1.tick_params(axis='x', rotation=90)
```

```
ax2 = ax1.twinx()
ax2.plot(g_data_filtered['Campaign'],
g_data_filtered['CPL'], color='olive', marker='o',
label='CPL ($)', linewidth=2)
ax2.set_ylabel('CPL ($)', color='olive')
ax2.tick_params(axis='y', labelcolor='olive')
```

```
fig.legend(loc='upper left', bbox_to_anchor=(0.1, 0.9),
fontsize=10)
plt.title('CR and CPL for Google Ads Campaigns')
plt.tight_layout()
plt.show()
```

Выводы

- Наиболее успешные Рекламные каналы: Facebook, Google Ads, при этом самые затратные.
- Отсутствие прямой корреляции между CTR и CR. Сильно различаются кампании по результатам. Причины различий связаны с такими факторами, как целевая аудитория, креатив, посадочная страница, ставки
- Самые эффективные РК - кампания в фейсбуке по широкой немецкой аудитории от 18.10.2023 и brand_search_eng PPC, где наблюдается самая высокая конверсия и низкий CPL





География

I'll try to investigate correlation between Level of Deutsch and City





```
order=['A1', 'A2', 'B1', 'B2', 'C1', 'C2']
```

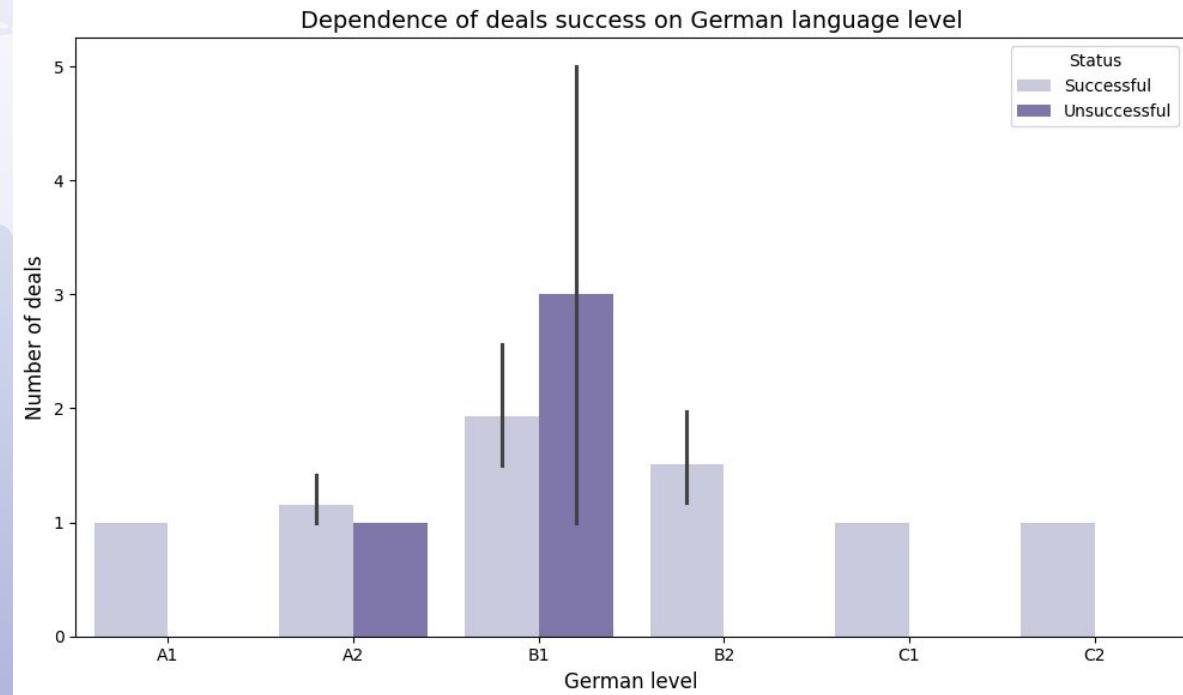
```
language_success['Level of Deutsch']=  
pd.Categorical(language_success['Level of Deutsch'],  
categories=order, ordered=True)
```

```
plt.figure(figsize=(10, 6))  
sns.barplot(data=language_success, x='Level of  
Deutsch', y='Deal Count', hue='Status', palette='Purples')
```

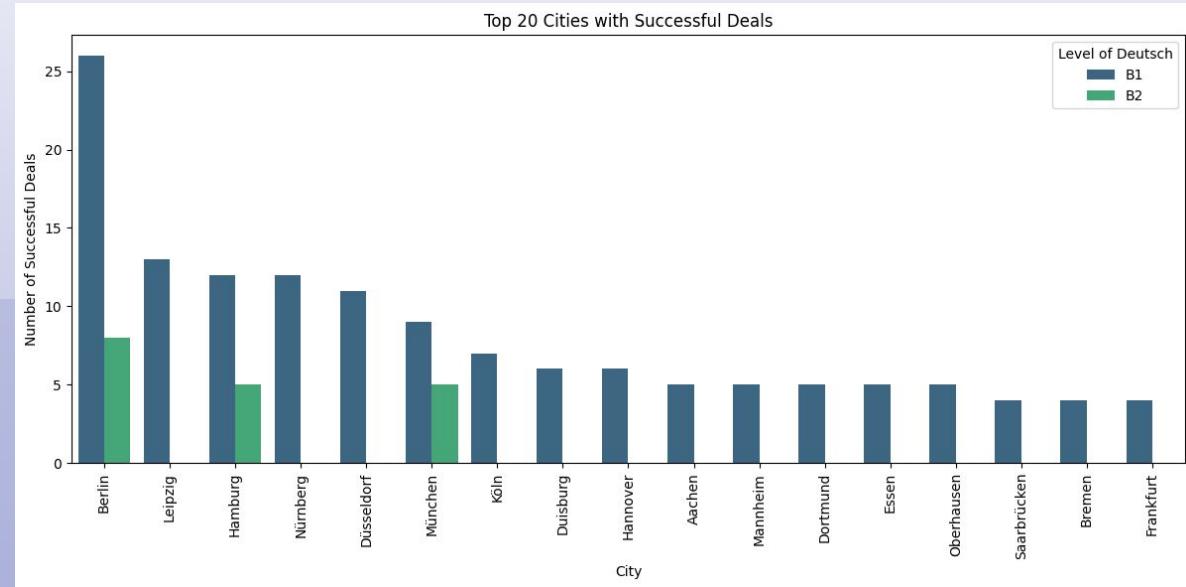
```
plt.title('Dependence of deals success on German  
language level', fontsize=14)  
plt.xlabel('German level', fontsize=12)  
plt.ylabel('Number of deals', fontsize=12)  
plt.xticks(rotation=0)  
plt.tight_layout()  
plt.show()
```

Наиболее высокий уровень успешности сделок наблюдается у людей с уровнем владения немецким языком B1. Это может говорить о том, что именно с как минимум с этим уровнем и наплывом украинцев, которые как раз заканчивают языковые курсы и находятся в поисках работы

Зависимость успешных сделок от языка владения



ТОП 20 городов с успешными сделками и языковые знания в них



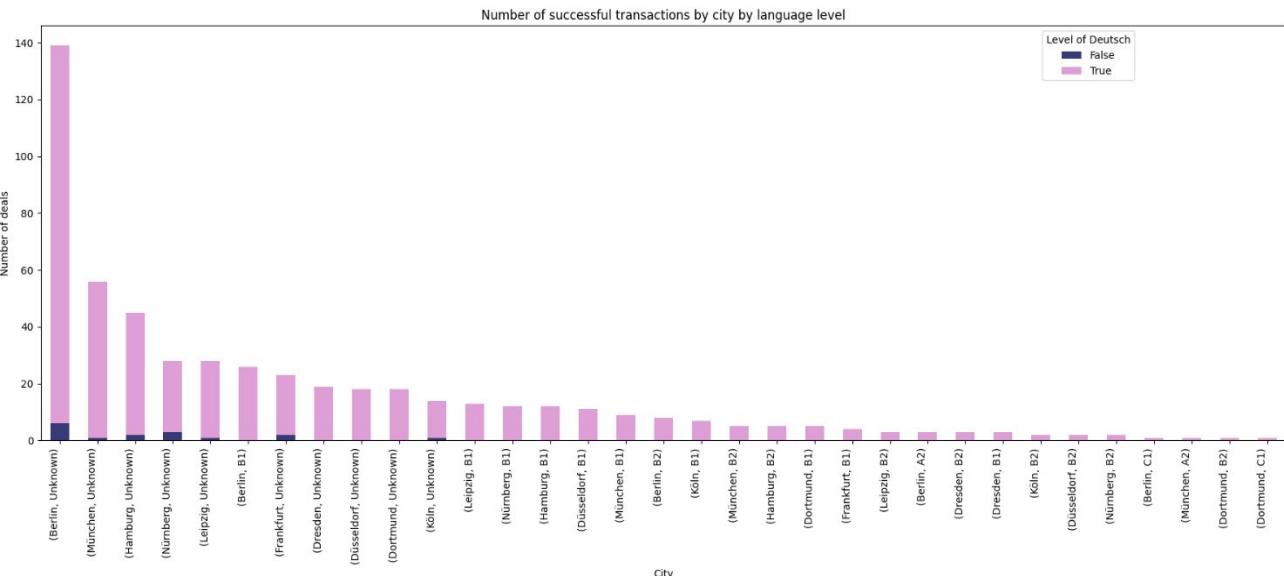
```
successful_deals =  
language_success[(language_success['Status'] ==  
'Successful') & (language_success['City'] != 'Unknown') &  
(language_success['Level of Deutsch'] != 'Unknown')]  
successful_deals =  
successful_deals.sort_values(by='Deal Count',  
ascending=False)  
top_successful_deals = successful_deals.head(20)
```

```
plt.figure(figsize=(12, 6))  
sns.barplot(x='City', y='Deal Count',  
data=top_successful_deals, palette='viridis', hue='Level  
of Deutsch')
```

```
plt.title('Top 20 Cities with Successful Deals')  
plt.xlabel('City')  
plt.ylabel('Number of Successful Deals')  
plt.xticks(rotation=90)  
plt.tight_layout()  
plt.show()
```



Динамика успешных сделок



```
df_filtered = deals[(deals['City'] != 'Unknown')]
```

```
top_cities = df_filtered[df_filtered['Is Successful'] == True].groupby('City').size()
top_cities_sorted =
top_cities.sort_values(ascending=False).head(10).index
```

```
df_top_cities =
df_filtered[df_filtered['City'].isin(top_cities_sorted)]
```

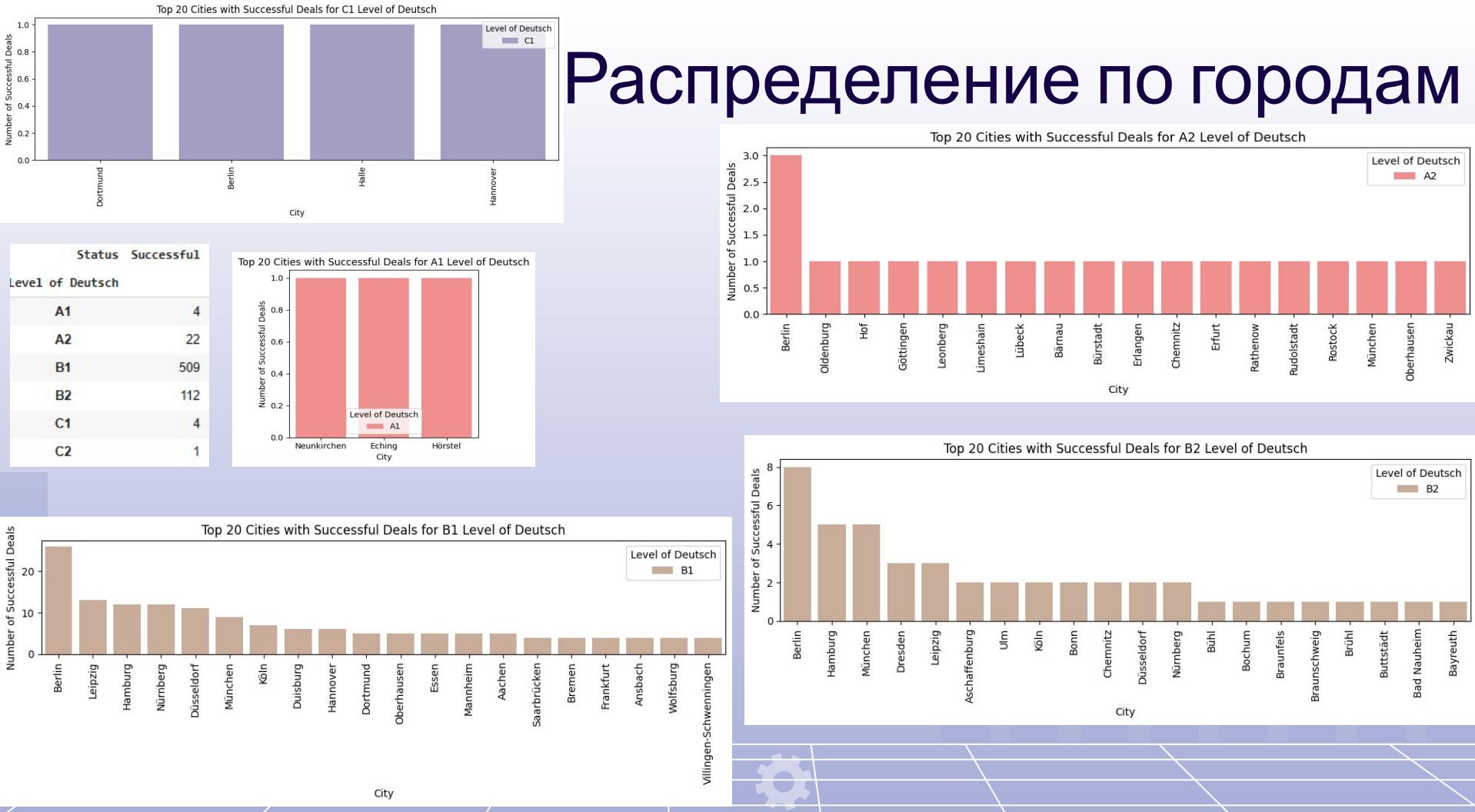
```
grouped = df_top_cities.groupby(['City', 'Level of Deutsch', 'Is Successful']).size().unstack(fill_value=0)
grouped =
grouped.loc[grouped.sum(axis=1).sort_values(ascending=False).index]
```

```
fig, ax = plt.subplots(figsize=(18, 8))
```

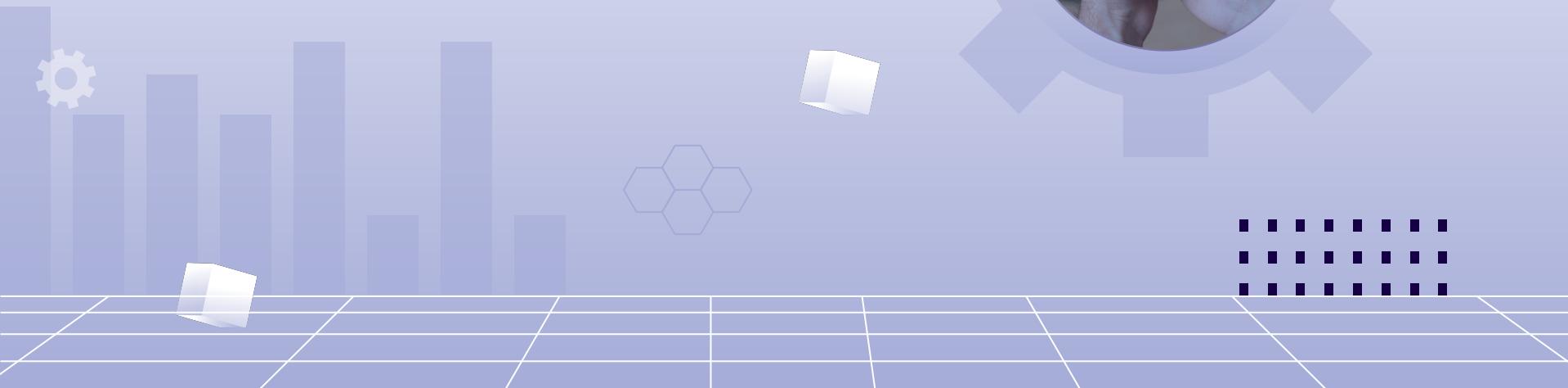
```
grouped.plot(kind='bar', stacked=True, ax=ax,
colormap='tab20b')
```

```
plt.title('Number of successful transactions by city by language level')
plt.xlabel('City')
plt.ylabel('Number of deals')
plt.xticks(rotation=90, ha='right')
plt.legend(title='Level of Deutsch',
bbox_to_anchor=(0.8, 1), loc='upper left')
```

```
plt.tight_layout()
plt.show()
```



Анализ эффективности отдела продаж



Сравнение работы менеджеров



Deal Owner Name	total_deals	successful_deals	total_sales	total_offer_amount	conversion_rate
0 Alice Johnson	15	0	0.0	0.0	0.000000
1 Amy Green	25	0	0.0	0.0	0.000000
2 Ben Hall	998	241	236840.0	2101220.0	0.241483
3 Bob Brown	137	2	950.0	7800.0	0.014599
4 Cara Iverson	809	103	686800.0	950080.0	0.127318
5 Charlie Davis	1856	424	429370.0	3896648.0	0.228448
6 Diana Evans	595	53	50450.0	585000.0	0.089076
7 Eva Kent	351	62	65100.0	576340.0	0.176638
8 George King	66	4	2900.0	33280.0	0.060606
9 Ian Miller	302	30	29670.0	277020.0	0.099338
10 Jane Smith	615	140	137350.0	1496040.0	0.227642
11 John Doe	1	1	100.0	4160.0	1.000000
12 Julia Nelson	1444	385	377620.0	3704141.0	0.266620
13 Kevin Parker	350	72	68250.0	722980.0	0.205714
14 Mason Roberts	176	20	18300.0	219440.0	0.113636
15 Nina Scott	761	210	201400.0	1992520.0	0.275953
16 Oliver Taylor	153	152	151650.0	1715480.0	0.993464
17 Paula Underwood	1277	325	324410.0	2949540.0	0.254503
18 Quincy Vincent	1339	199	218160.0	1832700.0	0.148618
19 Rachel White	256	4	14000.0	46280.0	0.015625
20 Sam Young	32	0	0.0	0.0	0.000000
21 Ulysses Adams	1475	541	521170.0	5138160.0	0.366780
22 Unspecified	20	0	0.0	0.0	0.000000
23 Victor Barnes	877	243	276144.0	2290320.0	0.277081
24 Wendy Clark	1	0	0.0	0.0	0.000000
25 Xander Dean	3	0	0.0	0.0	0.000000
26 Yara Edwards	35	0	0.0	0.0	0.000000
27 Zachary Foster	1	0	0.0	0.0	0.000000

```
deals['Is Successful'] = deals['Status'] == 'Successful'

owner_campaigns = deals.groupby(['Deal Owner Name']).agg(
    total_deals=('Status', 'count'),
    successful_deals=('Is Successful', 'sum'),
    total_sales=('Initial Amount Paid', 'sum'),
    total_offer_amount=('Offer Total Amount', 'sum')
).reset_index()

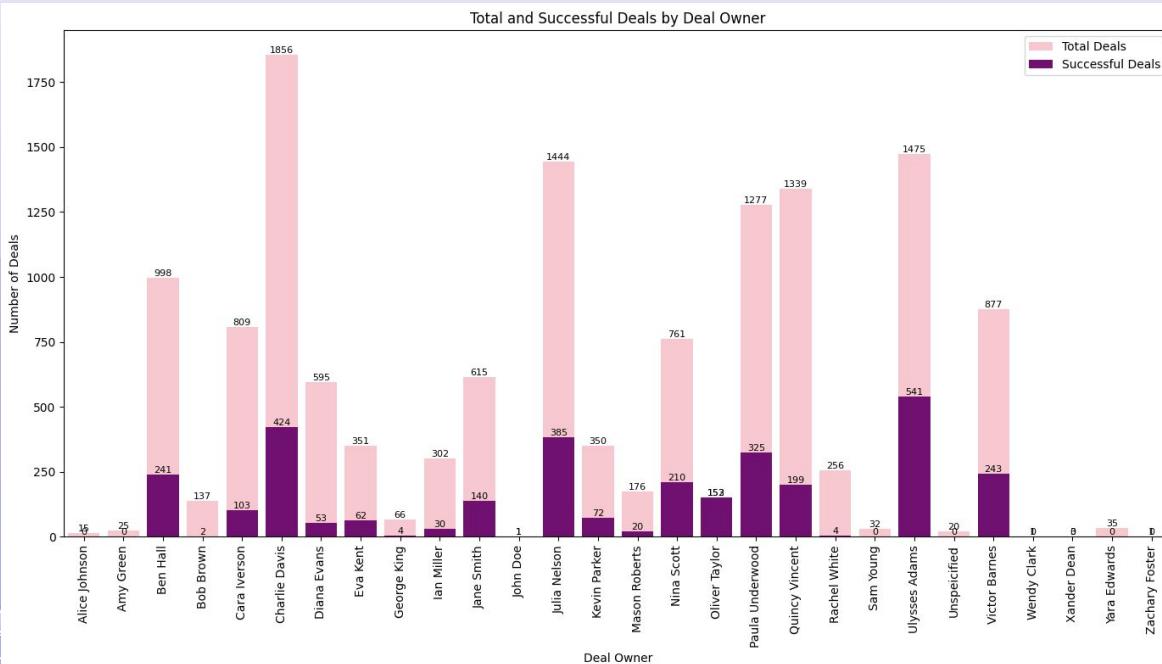
owner_campaigns['conversion_rate'] =
    owner_campaigns['successful_deals'] /
    owner_campaigns['total_deals']

owner_campaigns.head(100)
```





Сравнение менеджеров по кол-ву успешности заключения сделок



```
df_filtered = deals[(deals['City'] != 'Unknown')]
```

```
top_cities = df_filtered[df_filtered['Is Successful'] == True].groupby('City').size()
```

```
top_cities_sorted =
```

```
top_cities.sort_values(ascending=False).head(10).index
```

```
df_top_cities =
```

```
df_filtered[df_filtered['City'].isin(top_cities_sorted)]
```

```
grouped = df_top_cities.groupby(['City', 'Level of Deutsch', 'Is Successful']).size().unstack(fill_value=0)
```

```
grouped =
```

```
grouped.loc[grouped.sum(axis=1).sort_values(ascending=False).index]
```

```
fig, ax = plt.subplots(figsize=(18, 8))
```

```
grouped.plot(kind='bar', stacked=True, ax=ax, colormap='tab20b')
```

```
plt.title('Number of successful and unsuccessful transactions by city by language level')
```

```
plt.xlabel('City')
```

```
plt.ylabel('Number of deals')
```

```
plt.xticks(rotation=90, ha='right')
```

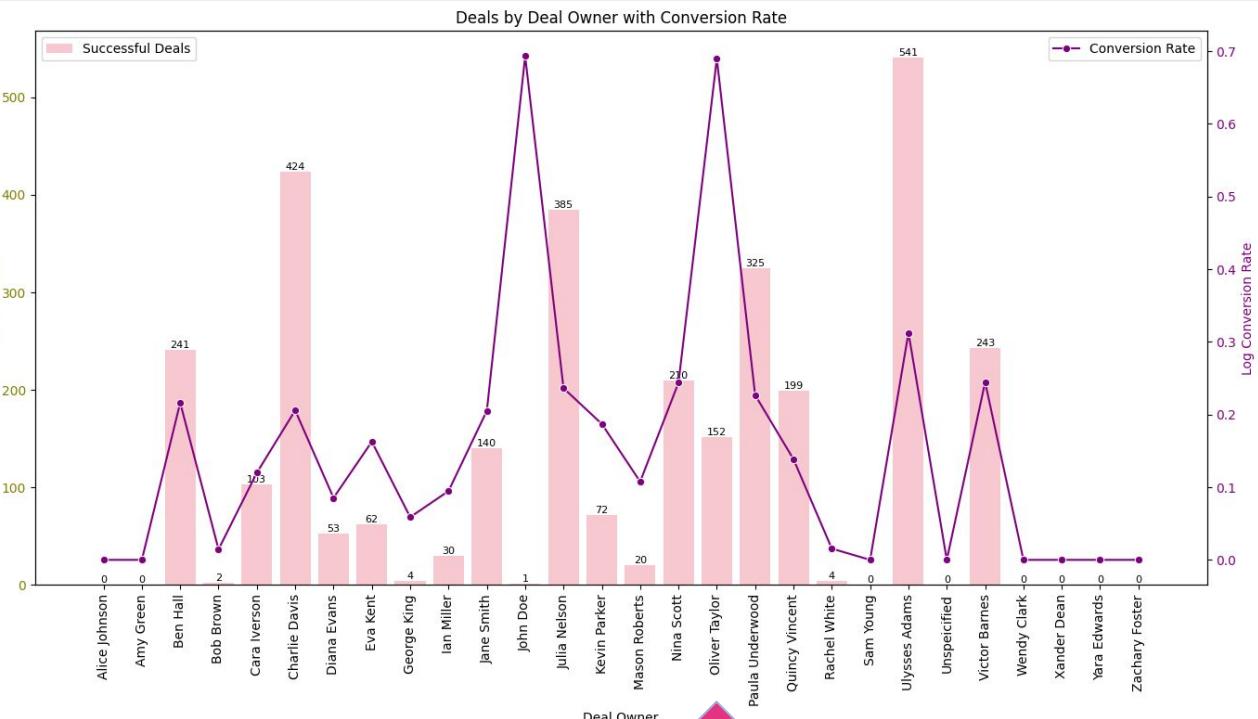
```
plt.legend(title='Level of Deutsch',
```

```
bbox_to_anchor=(0.8, 1), loc='upper left')
```

```
plt.tight_layout()
```

```
plt.show()
```

Конверсия по менеджерам



```
owner_campaigns['log_conversion_rate']=  
np.log(owner_campaigns['conversion_rate']+1)  
fig, ax1 = plt.subplots(figsize=(14, 8))  
sns.barplot(x='Deal Owner Name',  
y='successful_deals', data=owner_campaigns,  
color='pink', label='Successful Deals')  
ax2 = ax1.twinx()  
sns.lineplot(x='Deal Owner Name',  
y='log_conversion_rate', data=owner_campaigns,  
marker='o', color='purple', label='Conversion Rate')
```

```
for container in ax1.containers:  
    ax1.bar_label(container, label_type='edge',  
    fontsize=8, color='black')
```

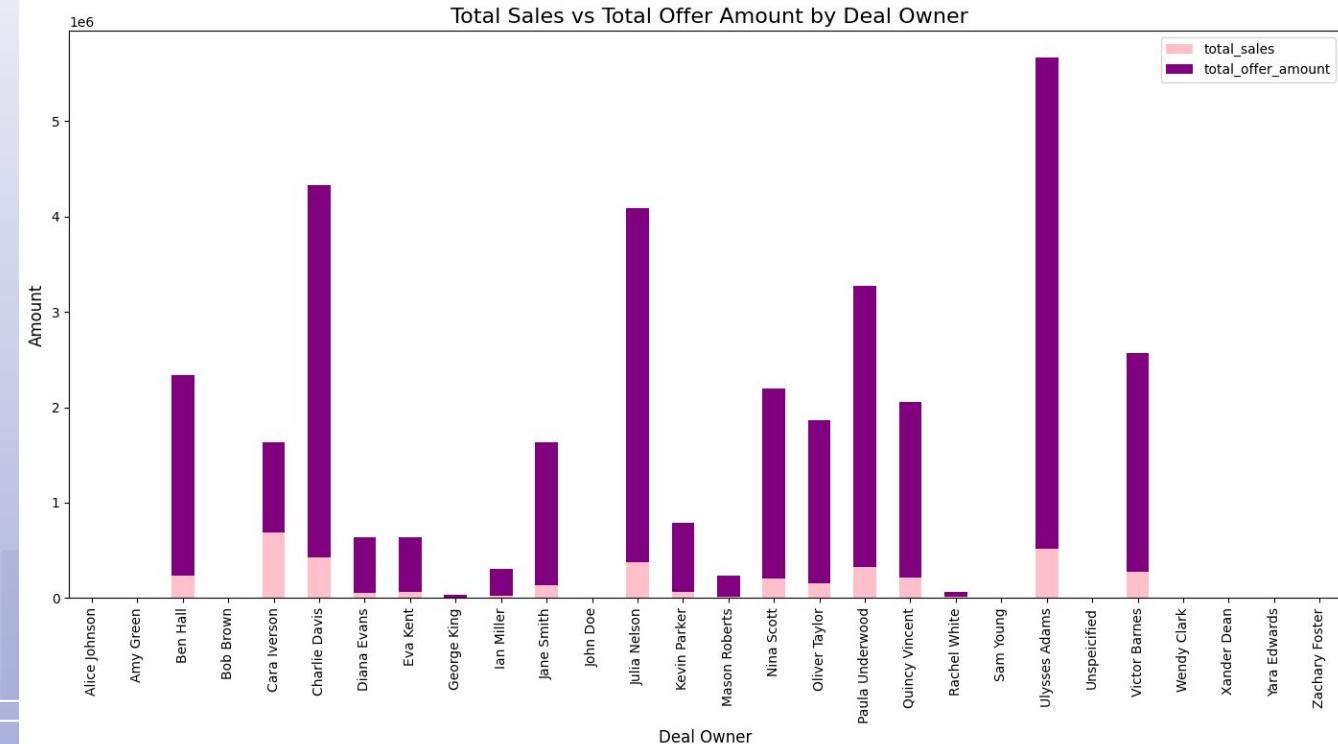
```
ax2.set_ylabel('Log Conversion Rate', color='purple')  
ax2.tick_params(axis='y', labelcolor='purple')  
ax1.set_title('Deals by Deal Owner with Conversion  
Rate')  
ax1.set_xlabel('Deal Owner')  
ax1.set_ylabel('Number of Deals')  
ax1.tick_params(axis='x', rotation=90)  
ax1.set_ylabel('Number of Deals', color='olive')  
ax1.tick_params(axis='y', labelcolor='olive')
```

```
ax1.legend(loc='upper left')  
ax2.legend(loc='upper right')
```



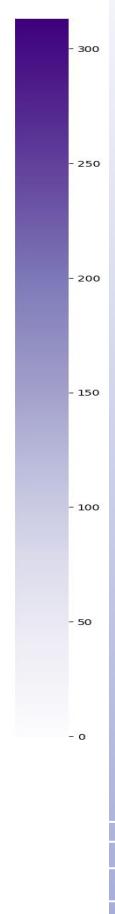
Продажи против предложения

Deal Owner Name	total_sales	total_offer_amount
Cara Iverson	686800.0	950080.0
Ulysses Adams	521170.0	5138160.0
Charlie Davis	429370.0	3896648.0
Julia Nelson	377620.0	3704141.0
Paula Underwood	324410.0	2949540.0
Victor Barnes	276144.0	2290320.0
Ben Hall	236840.0	2101220.0
Quincy Vincent	218160.0	1832700.0
Nina Scott	201400.0	1992520.0
Oliver Taylor	151650.0	1715480.0
Jane Smith	137350.0	1496040.0
Kevin Parker	68250.0	722980.0
Eva Kent	65100.0	576340.0
Diana Evans	50450.0	585000.0
Ian Miller	29670.0	277020.0
Mason Roberts	18300.0	219440.0
Rachel White	14000.0	46280.0
George King	2900.0	33280.0
Bob Brown	950.0	7800.0
John Doe	100.0	4160.0



Отказы

Deal Owner Name



```
filtered_deals = deals[deals['Lost Reason'] != 'No reason']
```

```
pivot_table = filtered_deals.pivot_table(
```

```
    index='Deal Owner Name',
```

```
    columns='Lost Reason',
```

```
    values='Contact Name',
```

```
    aggfunc='count',
```

```
    fill_value=0
```

```
)
```

```
pivot_table = pivot_table.round(0).astype(int)
```

```
plt.figure(figsize=(16, 16))
```

```
sns.heatmap(pivot_table, cmap='Purples',
```

```
annot=True, fmt='d')
```

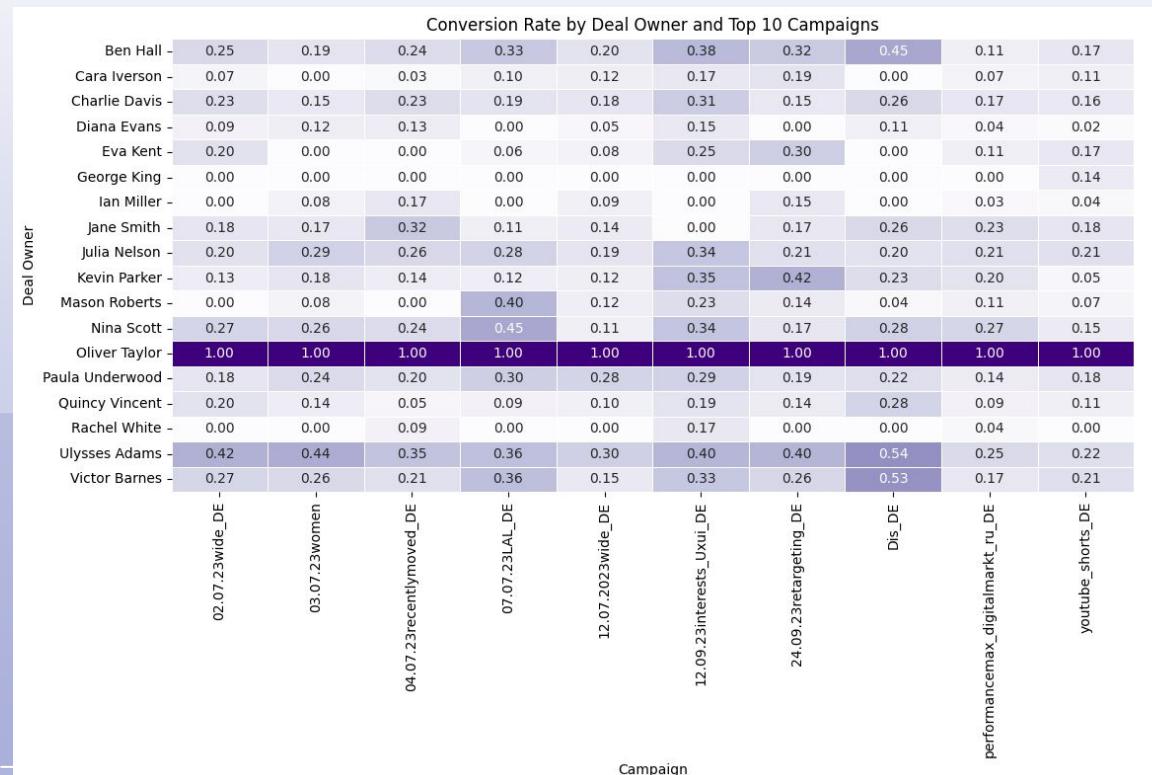
```
plt.title('Lost Reasons by Manager', fontsize=16)
```

```
plt.xlabel('Lost Reason', fontsize=14)
```

```
plt.ylabel('Deal Owner Name', fontsize=14)
```

```
plt.show()
```

Конверсия в работе менеджеров по РК



```

owner_campaigns = deals[deals['Campaign'] != 'No campaign'].groupby(['Deal Owner Name', 'Campaign']).agg(
    total_deals=('Status', 'count'),
    successful_deals=('Is Successful', 'sum'),
    total_sales=('Initial Amount Paid', 'sum'),
    total_offer_amount='Offer Total Amount', 'sum')
).reset_index()
owner_campaigns['conversion_rate'] = owner_campaigns['successful_deals'] / owner_campaigns['total_deals']

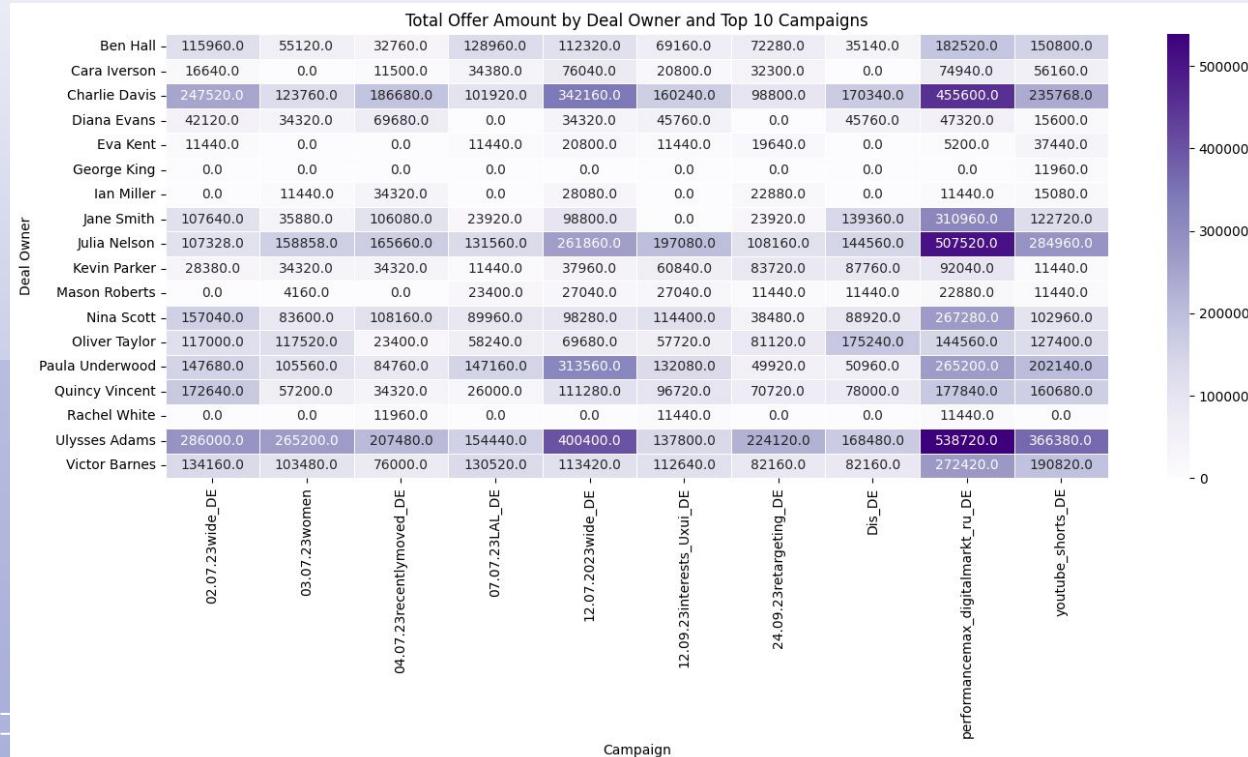
top_campaigns =
owner_campaigns.groupby('Campaign')['successful_deals'].sum().sort_values(ascending=False).head(10).index

filtered_owner_campaigns =
owner_campaigns[owner_campaigns['Campaign'].isin(top_campaigns)]
conversion_heatmap_data =
filtered_owner_campaigns.pivot_table(
    values='conversion_rate',
    index='Deal Owner Name',
    columns='Campaign',
    aggfunc='mean',
    fill_value=0
)
conversion_heatmap_data = conversion_heatmap_data.loc[
    (conversion_heatmap_data.sum(axis=1) > 0),
    (conversion_heatmap_data.sum(axis=0) > 0)
]

```



Предложения менеджеров



```

plt.figure(figsize=(14, 8))
sns.heatmap(conversion_heatmap_data, annot=True,
cmap='Purples', fmt=',.2f', cbar=True, linewidths=0.5)
plt.title('Heatmap of Conversion Rate by Deal Owner and Top 10 Campaigns')
plt.xlabel('Campaign')
plt.ylabel('Deal Owner')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

```

```

offer_amount_heatmap_data =
filtered_owner_campaigns.pivot_table(
    values='total_offer_amount',
    index='Deal Owner Name',
    columns='Campaign',
    aggfunc='sum',
    fill_value=0)
offer_amount_heatmap_data =
offer_amount_heatmap_data.loc[
    (offer_amount_heatmap_data.sum(axis=1)>0),
    (offer_amount_heatmap_data.sum(axis=0)>0)]

```

```

plt.figure(figsize=(14, 8))
sns.heatmap(offer_amount_heatmap_data, annot=True,
cmap='Purples', fmt=',.1f', cbar=True, linewidths=0.5)
plt.title('Heatmap of Total Offer Amount by Deal Owner and Top 10 Campaigns')
plt.xlabel('Campaign')

```

ВЫВОДЫ

- Некоторые сотрудники, такие как Victor Barnes, добиваются отличных результатов: они успешно закрывают сделки и приносят компании хороший доход.
- В то же время у других, например, George King или Ian Miller, показатели оставляют желать лучшего. Это может говорить о нехватке опыта, сложностях с клиентами или недостатке мотивации.
- Есть и те, кто активно работает, например, Paula Underwood. Она показывает высокие результаты как в предложениях, так и в завершённых сделках.

Для менеджеров с низкими показателями нужно понять, где возникают трудности – это могут быть навыки переговоров, подход к клиентам или отсутствие уверенности.
Организация тренингов и менторской поддержки могла бы помочь им подтянуть результаты.

- кто-то закрывает меньше сделок, но с высокой конверсией (например, Oliver Taylor), а кто-то берет количеством, но теряет в качестве.
- Прямая связь между количеством сделок и успехом не всегда очевидна. Это значит, что важно не только число звонков или встреч, но и их результативность.

Помочь менеджерам с низкой конверсией за счёт персонального обучения. Возможно, они упускают клиентов на этапе переговоров или плохо доносят ценность продукта. Поделиться успешными стратегиями лидеров команды: какие фразы работают, как лучше отвечать на возражения.

ВЫВОДЫ

Тепловая карта показала, что менеджеры теряют сделки по разным причинам. Вот самые частые:

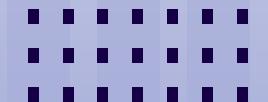
- Клиент уходит к конкурентам.
- Неподходящее предложение (например, дорого или контракт не устраивает).
- Проблемы с коммуникацией — кто-то перестал отвечать, а кто-то решил, что услуга должна быть бесплатной.

Необходимо: Научить менеджеров работать с возражениями, обучить эффективным методам переговоров.

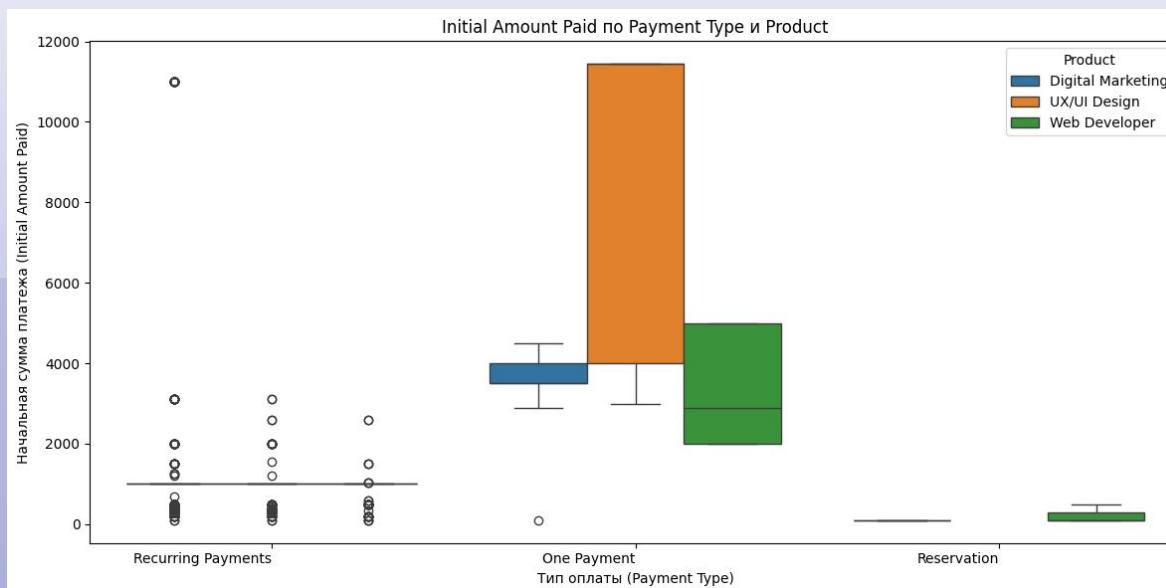
Что стоит сделать:

1. **Определить сильных лидеров** и использовать их опыт как пример для остальных.
2. **Провести обучение для слабых менеджеров.** Это могут быть тренинги по продажам, улучшение навыков переговоров или работа над личной мотивацией.
3. **Оптимизировать маркетинговые кампании.** Некоторые из них привлекают больше клиентов, чем другие, — важно понять, почему.
4. **Внедрить регулярную оценку работы.** Это поможет своевременно выявлять проблемы и поддерживать команду.

Анализ платежей



ЗАВИСИМОСТЬ УСПЕШНОСТИ ОТ ТИПА ПЛАТЕЖА



```
filtered_deals = deals[  
    (deals['Initial Amount Paid'] > 0) &  
    (deals['Product'] != 'Not chosen')  
]
```

```
plt.figure(figsize=(12, 6))  
sns.boxplot(  
    data=filtered_deals,  
    x='Payment Type',  
    y='Initial Amount Paid',  
    hue='Product'  
)
```

```
plt.title('Initial Amount Paid по Payment Type и Product')  
plt.xlabel('Тип оплаты (Payment Type)')  
plt.ylabel('Начальная сумма платежа (Initial Amount Paid)')  
plt.xticks(rotation=0, ha='right')  
plt.tight_layout()  
plt.show()
```



Зависимость суммы первого платежа и типа платежа и влияние на качество сделки

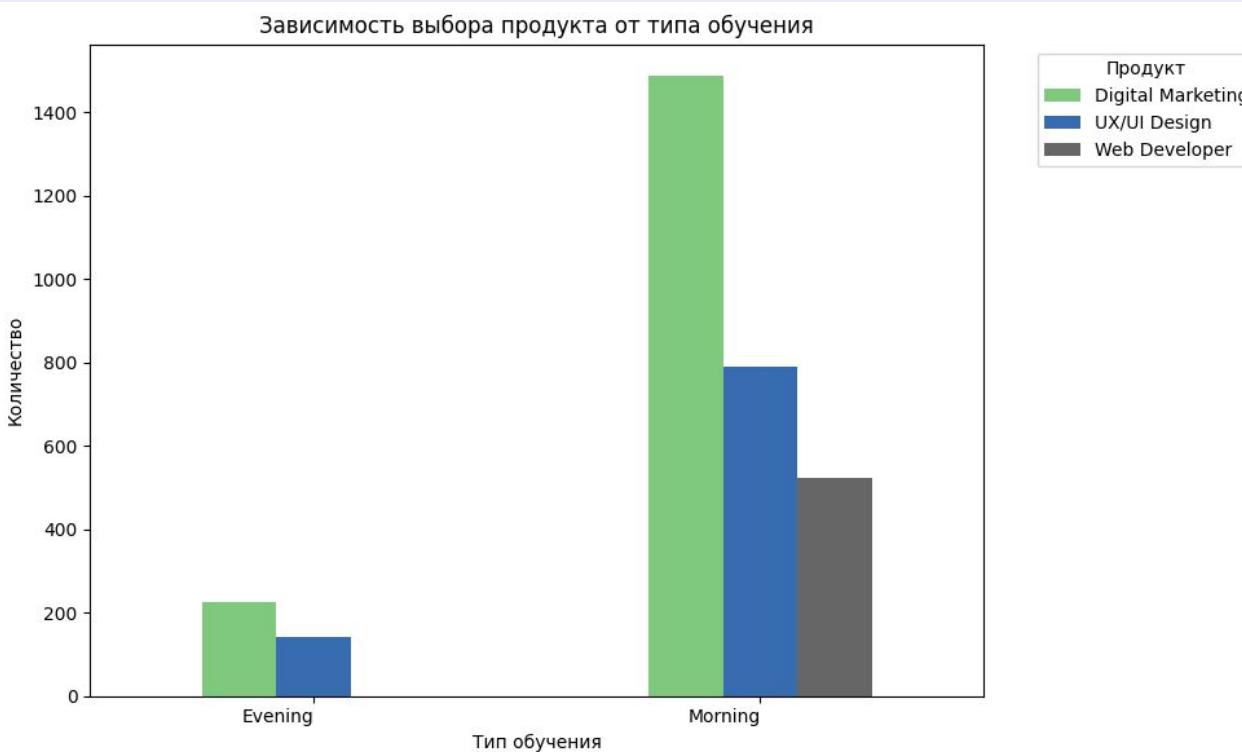


```
plt.figure(figsize=(12, 6))
sns.boxplot(
    data=filtered_deals,
    x='Payment Type',
    y='Initial Amount Paid',
    hue='Quality2'
)

plt.title('Initial Amount Paid по Payment Type и Quality2')
plt.xlabel('Тип оплаты (Payment Type)')
plt.ylabel('Начальная сумма платежа (Initial Amount Paid)')
plt.xticks(rotation=0, ha='right')
plt.tight_layout()
plt.show()
```



Зависимость суммы первого платежа и типа платежа и влияние на качество



```
filtered_deals = deals[(deals['Product'] != 'Not chosen') & (deals['Education Type'] != 'No info')]

grouped = filtered_deals.groupby(['Education Type', 'Product']).size().unstack(fill_value=0)

fig, ax = plt.subplots(figsize=(10, 6))

grouped.plot(kind='bar', stacked=False, ax=ax, colormap='Accent')

plt.title('Зависимость выбора продукта от типа обучения')
plt.xlabel('Тип обучения')
plt.ylabel('Количество')
plt.xticks(rotation=0, ha='right')
plt.legend(title='Продукт',
bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



ВЫВОДЫ

Для типа оплаты One Payment наблюдается наибольший разброс в начальной сумме платежа. Это говорит о том, что стоимость услуг по этому типу оплаты может сильно варьироваться

Самая высокая средняя начальная сумма платежа наблюдается для разовых платежей по продукту "Digital Marketing" по причине длительности курса и более высокой стоимости курса.

Клиенты, выбирающие разовые платежи и более высокое качество, в среднем платят больше. Recurring Payments например имеют наименьшую медиану и размах, что указывает на более низкие начальные платежи по сравнению с другими типами оплаты. Большое количество выбросов свидетельствует о значительном разнообразии платежей в этом сегменте.





ЮНИТ ЭКОНОМИКА

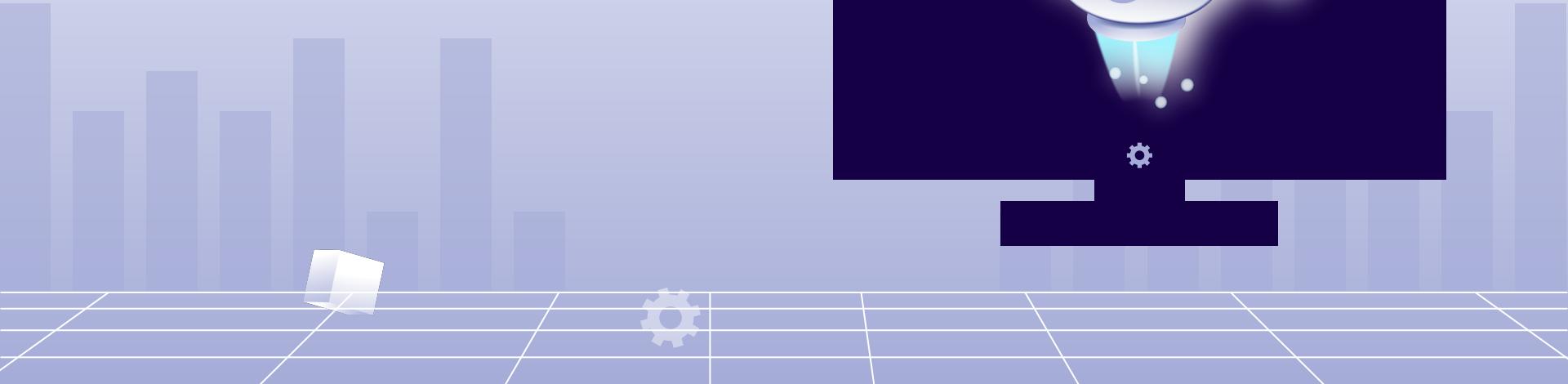
	UA	B	C1	AOV	T (кол-во плат)	REV (сумма)	CPA лида	APC(T/B)	ARPU (REV/T)	CLTV (lifetime)	CM (м.приб)
All	13 970,00	3 211	22,98%	438,87	17 224,00	7 559 096,88	11,46	5,4	438,9	2 354,1	7 399 000,7
DM		1776	12,71%	403,16	10224	4121908	11,46	5,76	403,16	2 320,9	3 961 811,6
UX/UI Design		902	6,46%	465,15	4990	2321099	11,46	5,53	465,15	2 573,3	2 161 002,3
Web Developer		513	3,67%	499,83	1956	977667	11,46	3,81	499,83	1 905,8	817 571,3



Дерево метрик



Гипотеза



Разработка чат-бота для сайта

Гипотеза: Добавление чат-бота для мгновенного общения с клиентами на сайте поможет оперативно отвечать на вопросы пользователей и направлять их к следующему шагу, что увеличит конверсию на 3% в течение месяца.

6%

Как правило многие пользователи уходят с сайта, если не получают быстрых ответов на свои вопросы. Чат-бот станет первым помощником, который будет доступен 24/7, мгновенно предоставляя информацию и помогая пользователям сделать выбор в пользу покупки или записи на урок. Бот ведет по воронке продаж, в нем есть выбор получить консультацию сразу, и есть получить большинство ответов на свои вопросы, ведя клиента по воронке продаж и предлагая конкретный курс под его интересы и скачать бесплатный урок. Мало того, это снизит нагрузку на менеджеров и оставит им в задачах уже упрощенную коммуникацию.



Проведение А/В тестирования

Нулевая гипотеза: Добавление чат-бота не влияет на конверсию, и она остаётся на текущем уровне.

Альтернативная гипотеза : Добавление чат-бота увеличивает конверсию на 3% в течение месяца.

Метод: А/В тестирование

- **Контрольная группа (A):** Пользователи, которые используют текущий сайт без чат-бота.
- **Экспериментальная группа (B):** Пользователи, которые взаимодействуют с сайтом с чат-ботом.

Сравнение метрик: Измерить конверсию в обеих группах.

Гипотеза будет считаться подтверждённой, если конверсия в группе с чат-ботом будет не менее чем на 3% выше, чем в контрольной группе. Текущая конверсия $p=0.2298$. Желаемая конверсия = 0.2366

Расчет требуемой выборки: $N=16*p*(1-p)/x^2 = (16 * 0.2298 * (1 - 0.2298)) / 0.03^2 = 3166$ для каждой группы.

Предполагаемое время проведения времени тестирования - 14 дней (исходя из среднего времени проведения тестирований). Более точное значение выбрать не предоставляет возможности ввиду отсутствия данных о трафике



Thanks!

Do you have any questions?

smedvedeva2412@gmail.com

