In [1]:
```
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [2]:
```
from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"
```

Перед тобой датасет некой интернет-компании. Загрузи его в Python или R, ознакомься с данными и ответь на приведенные ниже вопросы. Для ответа на некоторые из них понадобится произвести расчеты, ход которых опиши в файле с кодом.

- Какой рекламный канал принёс больше всего дохода за всё время?
- Как изменился средний чек транзакции после введения коронавирусных ограничений?
- Как он изменился для пользователей, покупающих с промокодом и без? (параметр «promo_activated»)
- Можно ли с уверенностью в 95% сказать, что CR (коэффициент конверсии в транзакцию) в выходные дни отличается от CR в будние дни?
- Вам необходимо спрогнозировать объем дохода, полученного с пользователей, приведенных на сайт контекстной рекламой (medium = cpc) на полгода вперед. Опишите, как бы вы подошли к этой задаче и какие дополнительные данные вам понадобятся?
- Если ты нашел что-то еще интересное в данных, то тоже пиши ;)

Считаем датасет из файла

In [3]:
```
df = pd.read_csv('summer_camp_data.csv', parse_dates=['date'])
df.head()
```

Out[3]:

	date	source	medium	delivery_available	device_type	promo_activated	filter_used	pageview
0	2020-05-11	google	organic	Не определено	Десктоп	no	no	312
1	2020-05-11	yandex	cpc	Не определено	Мобайл	yes	no	330
2	2020-05-11	google	cpc	Не определено	Мобайл	no	no	297
3	2020-05-11	google	cpc	Не определено	Десктоп	no	no	187
4	2020-05-11	yandex	organic	Не определено	Десктоп	no	no	215

In [4]:
```
#  Проверяем формат значений в колонках, иногда числовые значения записываются в виде строки.
df.dtypes
```

Out[4]:
```
date                  datetime64[ns]
source                        object
medium                        object
delivery_available            object
device_type                   object
promo_activated               object
filter_used                   object
pageviews                      int64
visits                         int64
productClick                   int64
addToCart                      int64
checkout                       int64
transactions                   int64
revenue                      float64
dtype: object
```

In [5]:
```
#  проверяем, есть ли пропуски в датасете
df.isna().any().any()
```

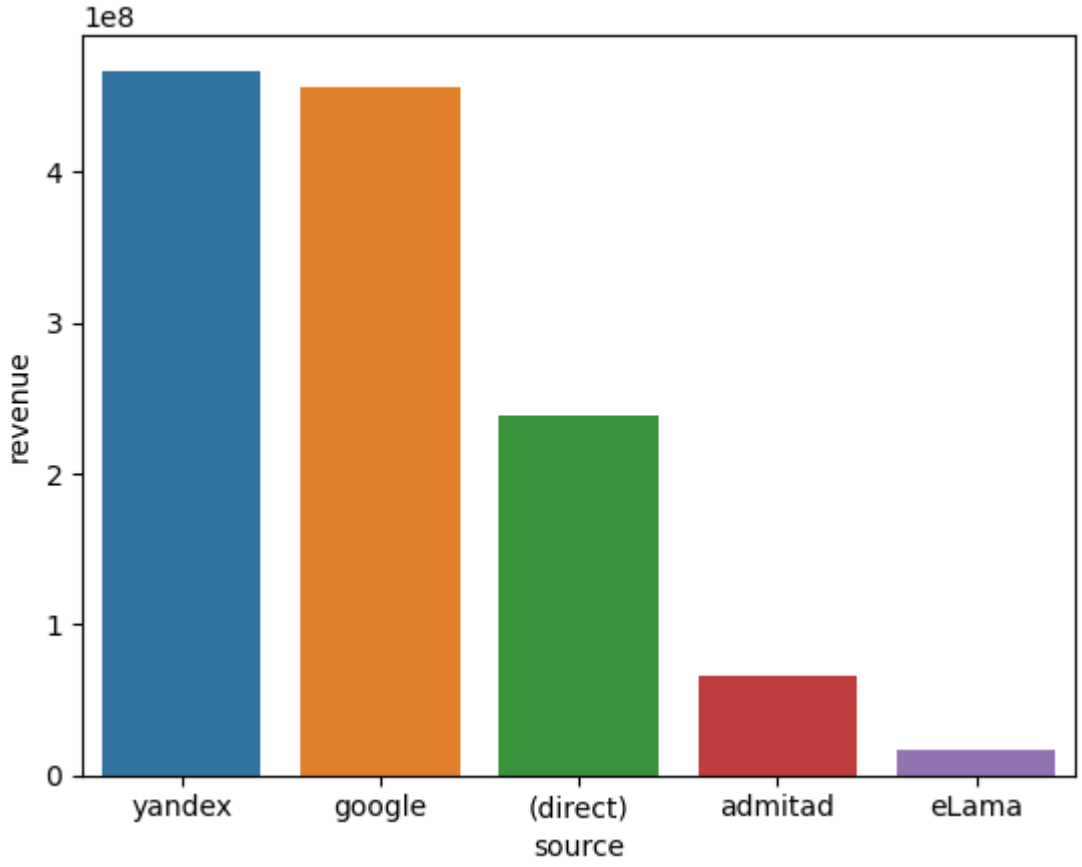
Out[5]:
```
False
```

a. Какой рекламный канал принёс больше всего дохода за всё время?

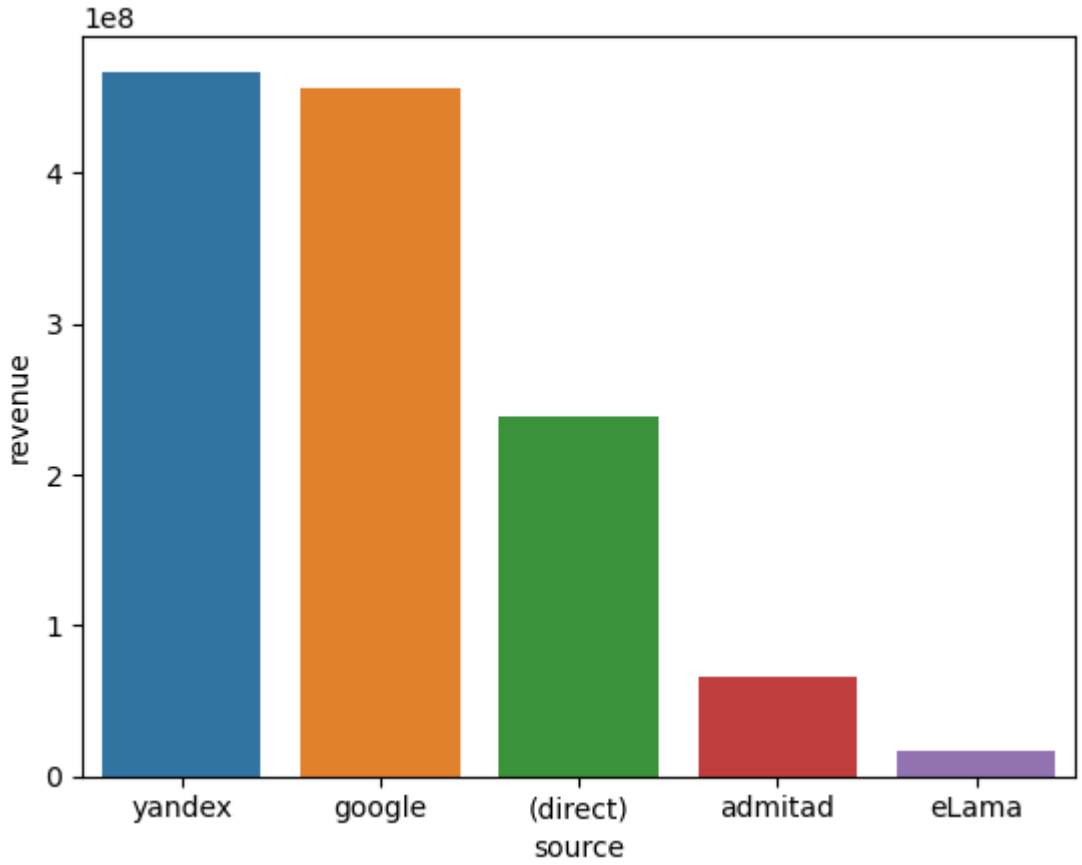
In [6]:
```
# Произведем группировку датасета по источникам пользователей и сагрегируем revenue
revenue_by_source = df.groupby('source', as_index=False).agg({'revenue': 'sum'}).sort_values('revenue', ascending=False).head(5)
```

In [7]:
```
#  Произведем группировку помимо источника
revenues = df.groupby(['source', 'medium'], as_index=False).agg({'revenue':
'sum'}).sort_values('revenue', ascending=False).head(5)
```

In [8]:
```
sns.barplot(data = revenue_by_source, x=revenue_by_source.source,
y=revenue_by_source.revenue);
```



In [9]:
```
sns.barplot(data = revenue_by_source, x=revenue_by_source.source,
y=revenue_by_source.revenue);
```



Как видим, больше всего дохода принес канал в Яндексе, однако если смотреть отдельно бесплатные и платные источники, то выигрывает google с бесплатным источником organic, т.е. через запросы пользователей.

b. Как изменился средний чек транзакции после введения коронавирусных ограничений?Как он изменился для пользователей, покупающих с промокодом и без? (параметр «promo_activated»)

Сначала создадим колокну стоимости чека, который будем считать как доход, деленный на количество транзакций (revenue / transaction)

In [10]:
```
df['check_cost'] = round(df.revenue / df.transactions, 2)
```

In [11]:
```
df.check_cost.fillna(0, inplace=True)
```

По официальным данным, коронавирусные ограничения были введены 16 марта 2020 года. Разделим пользователей на группы с помощью фильтрации по дате

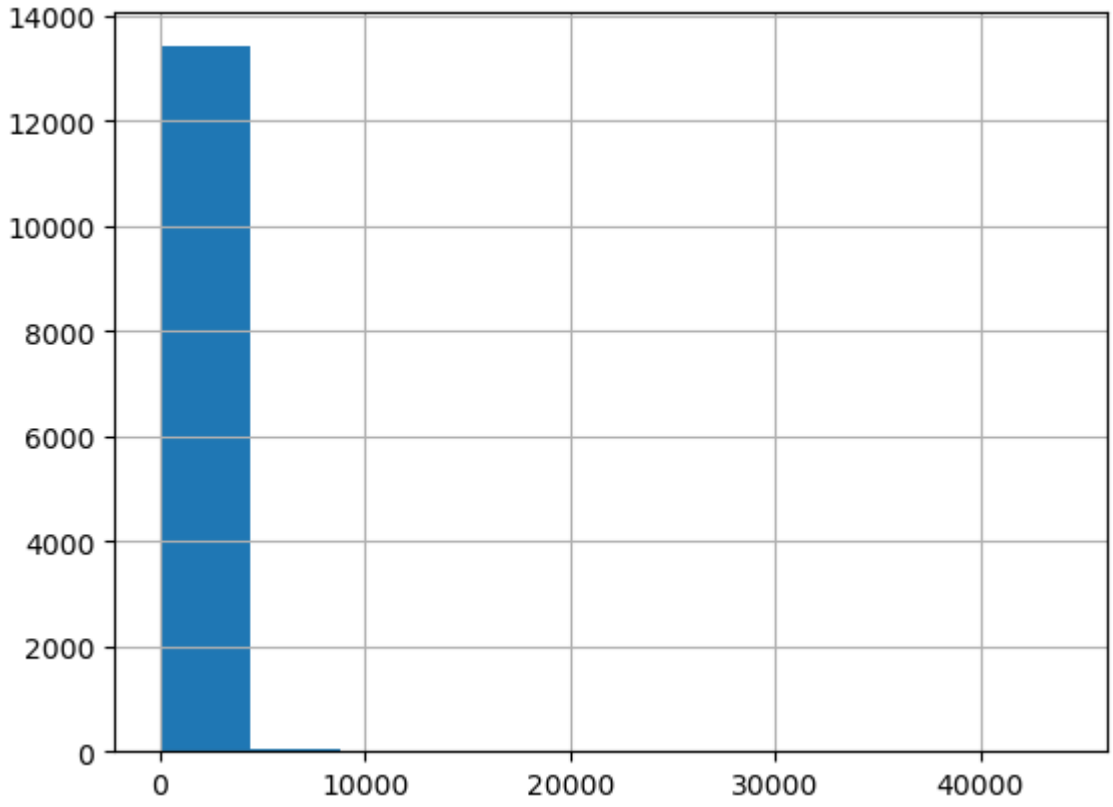
In [12]:
```
#  Убеждаемся, что минимальная дата соответствует дате ввода ограничений
df[df['date'] >= dt.datetime(2020, 3, 16)].date.min()
```

Out[12]:
```
Timestamp('2020-03-16 00:00:00')
```

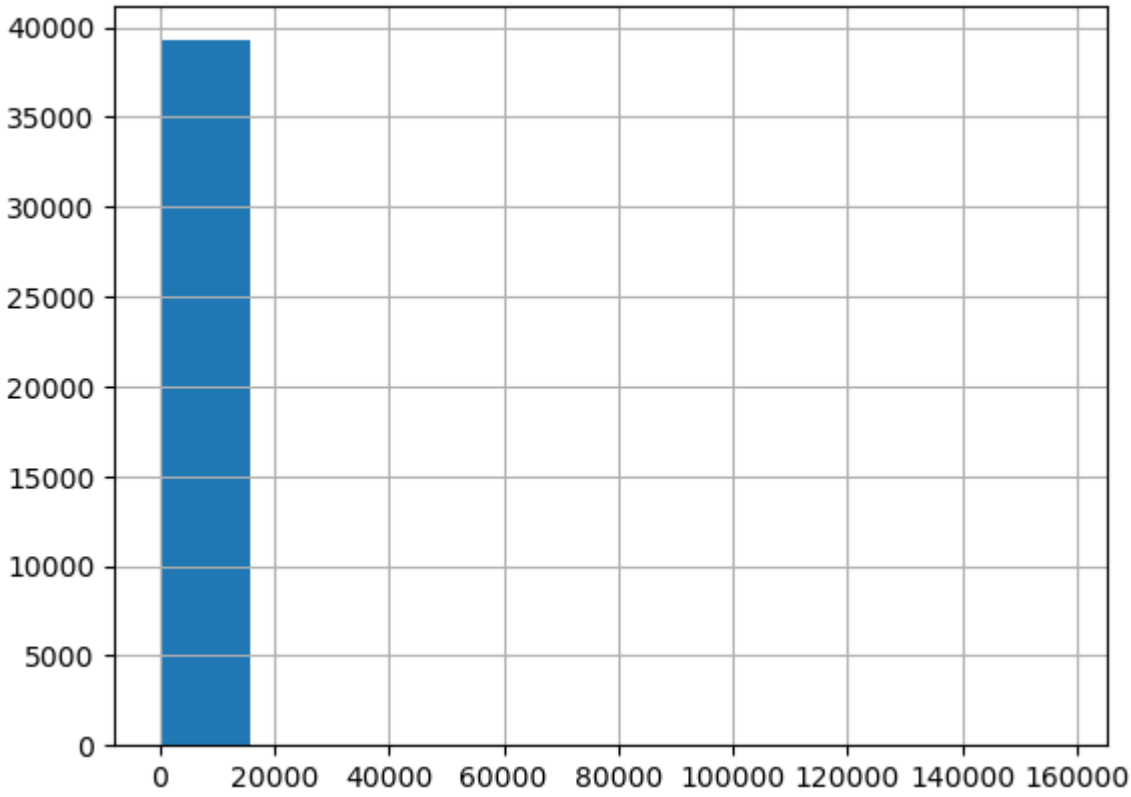
In [13]:
```
#  Тогда получаем два датафрейма
df_before_covid = df[df['date'] < dt.datetime(2020, 3, 16)]
df_after_covid = df[df['date'] >= dt.datetime(2020, 3, 16)]
```

Посмотрим на распределение величины среднего чека

In [14]:
```
df_before_covid.check_cost.hist();
```



In [15]:
```
df_after_covid.check_cost.hist();
```



Распределения несимметричные, поэтому для ответа об изменении среднего чека транзакции будем сравнивать медианные значения, то есть насколько изменилась величина среднего чека. Если расчитывать средние арифметические двух групп, тогда будет дан ответ, насколько изменился средний по значению чек.

In [16]:
```
before_median = round(df_before_covid.check_cost.median(), 2)
after_median = round(df_after_covid.check_cost.median(), 2)
diff_median = round(100 * (after_median - before_median)/ before_median, 2)

before_mean = round(df_before_covid.check_cost.mean(), 2)
after_mean = round(df_after_covid.check_cost.mean(), 2)
diff_mean = round(100 * (after_mean - before_mean)/ before_mean, 2)
print(f'''Величина среднего чека до ковида {before_median}, во время ковида
{after_median}, относительная разница между величинами {diff_median}%
Значение чека транзакции в среднем до ковида {before_mean}, во время ковида
{after_mean}, относительная разница между величинами {diff_mean}%
''')
```

```
Величина среднего чека до ковида 1058.0, во время ковида 1006.07, относительная ра
зница между величинами -4.91%
Значение чека транзакции в среднем до ковида 967.41, во время ковида 862.75, относ
ительная разница между величинами -10.82%
```

Используя медианные значения, получаем, что средний чек транзакции после введения ковидных ограничений снизился на 4,91%

Для ответа на вторую часть вопроса разделим датафреймы по признаку 'promo_activated' и определим медианные значения групп

In [17]:
```
# медианные значения чеков до ковида
before_medians = df_before_covid.groupby('promo_activated', as_index=False) \
                                [['check_cost']].median().round(2) \
                                .rename(columns={'check_cost':
'checks_before'})
```

In [18]:
```
# медианные значения чеков после ввода ковидных ограничений
after_medians = df_after_covid.groupby('promo_activated', as_index=False) \
                              [['check_cost']].median().round(2) \
                              .rename(columns={'check_cost': 'checks_after'})
```

In [19]:
```
#  объединение датафреймов
medians_promo = before_medians.merge(after_medians, on='promo_activated')
```

In [20]:
```
# Создаем колонку с относительным изменением медианных значений и добавляем %
medians_promo['difference'] = round(100 *(medians_promo.checks_after -
medians_promo.checks_before) / medians_promo.checks_before , 2)

medians_promo['difference'] = medians_promo['difference'].apply(lambda x:
str(x) + '%')
```

In [21]:
```
medians_promo
```

Out[21]:

	promo_activated	checks_before	checks_after	difference
0	no	983.88	919.25	-6.57%
1	yes	1121.55	1078.08	-3.88%

Как видим, средний чек пользователей без промокода упал на 6.57% средний чек с промокодом также снизился на 3.87%

c. Можно ли с уверенностью в 95% сказать, что CR (коэффициент конверсии в транзакцию) в выходные дни отличается от CR в будние дни?

Коээфициент конверсии в ключевое действие расчитывается как отношение числа пользователей, выполнивших его, к числу пользователей с прошлого шага

Для ответа на этот вопрос необходимо ввести колонку CR, сгруппировать данные CR по дням недели, рассмотреть распределения величины CR в группах, подобрать стат критерий для сравнения величины.

Также можем считать конверсию относительно предыдущего шага, а можем относительно числа просмотров, сделаем оба варианта

In [22]:
```
# Добавим колонку с названием дня недели
df['day_week'] = df.date.dt.day_name()
```

In [23]:
```
df.head()
```

Out[23]:

	date	source	medium	delivery_available	device_type	promo_activated	filter_used	pageview
0	2020-05-11	google	organic	Не определено	Десктоп	no	no	312
1	2020-05-11	yandex	cpc	Не определено	Мобайл	yes	no	330
2	2020-05-11	google	cpc	Не определено	Мобайл	no	no	297
3	2020-05-11	google	cpc	Не определено	Десктоп	no	no	187
4	2020-05-11	yandex	organic	Не определено	Десктоп	no	no	215

In [24]:
```
# Создаем колонки CR относительно числа checkout и pageviews
df['CR_prev'] =  round(100 * df.transactions / df.checkout, 2)

df['CR_full'] =  round(100 * df.transactions / df.pageviews, 2)
```

При расчете conversion rate мы столкнемся с делением на 0, так как есть строки, где не было просмотров или перехода в корзину, при формировании данных эти строки уберем, при расчете CR_full брали число просмотров, поэтому убираем строки с 0 значением pageviews и посмотрим на распределение CR_full

In [25]:
```
df.query('pageviews > 0').CR_full.describe()
```

Out[25]:
```
count    51100.000000
mean         5.595580
std         10.087777
min          0.000000
25%          0.000000
50%          4.350000
75%          7.140000
max        300.000000
Name: CR_full, dtype: float64
```

Получили CR > 100! Посмотрим на эти строки

In [26]:
```
df.query('pageviews > 0').query('CR_full > 100')
```

Out[26]:

	date	source	medium	delivery_available	device_type	promo_activated	filter_used	pag
17608	2020-03-26	(direct)	(none)	Не определено	Не определено	no	no	
17610	2020-09-23	(direct)	(none)	Не определено	Не определено	no	no	
17619	2020-09-03	(direct)	(none)	Не определено	Не определено	no	no	
17627	2020-08-11	(direct)	(none)	Не определено	Не определено	no	no	
17631	2020-02-22	(direct)	(none)	Не определено	Не определено	no	no	
...	...	...	...	...	...	...	...	
24460	2020-04-01	eLama	cpc	Не определено	Не определено	no	no	
24642	2020-02-22	other	organic	Не определено	Не определено	no	no	
25232	2020-07-10	eLama	cpc	Не определено	Не определено	no	no	
25614	2020-01-29	opmcpa	cpa	Не определено	Не определено	no	no	
26585	2020-05-24	cityads	cpa	Не определено	Не определено	no	no	

107 rows × 18 columns

Обратим внимание, что в полученном датафрейме число просмотров страницы меньше числа посещений, что может говорить о потере части данных в логах, либо о переходе в корзину быстрее, чем прогружается страница. Причин может быть много, стоит обратиться к тех отделу для выяснения причины. Однако необходимо проанализировать текущий датафрейм, поэтому отфильтруем данные

In [27]:
```
df.query('pageviews < visits').shape[0]
```

Out[27]:
```
3167
```

Отфильтруем строки, в которых CR > 100

In [28]:
```
CR_full_df = df.query('pageviews > 0').query('CR_full <= 100')
```

In [29]:
```
round(100 * (df.shape[0] - df.query('pageviews > 0').query('CR_full <=
100').shape[0]) / df.shape[0], 2)
```

Out[29]:
```
3.28
```

Фильтрация отсеяла 3.28% данных, но это необходимо для исключения влияния выбросов и неточных данных на выводы

Теперь группируем данные по будним дням и выходным, для этого создадим колонку фильтрации

In [30]:
```
#  список для сравнения данных"
WEEKENDS = ['Saturday', 'Sunday']

# функция для будних и выходных дней
def weekends(row):
    if row in WEEKENDS:
        return 'weekend'
    return 'workday'
```

In [31]:
```
# применяем функцию для создания новой колонки
CR_full_df['weekend'] = CR_full_df.day_week.apply(weekends)
#  Фильтруем датафрейм по дням недели
CR_workday = CR_full_df[CR_full_df['weekend'] == 'workday'].CR_full
CR_weekend = CR_full_df[CR_full_df['weekend'] == 'weekend'].CR_full
```

Теперь необходимо выбрать статистический критерий на основе распределения данных. Проверим на нормальность распределения с помощью критерия Шапиро

In [32]:
```
from scipy.stats import shapiro
```

In [33]:
```
# Формируем выборки из датафреймов, т.к. на больших объемах данных тест
Шапиро становится менее точным
```

```
shapiro(CR_weekend.sample(1000, random_state=17))
shapiro(CR_workday.sample(1000, random_state=17))
```

Out[33]:
```
ShapiroResult(statistic=0.6265932321548462, pvalue=5.5281224417614033e-42)
```

Out[33]:
```
ShapiroResult(statistic=0.5024590492248535, pvalue=0.0)
```

Нулевая гипотеза критерия говорит о том, что рассматриваемые распределения являются нормальными. Получили p-уровень значимости < 0.05 в обоих случаях, поэтому принимается альтернативная гипотеза, следовательно, распределения не являются нормальными. В данном случае можем привести данные к нормальному виду с помощью логарифмирования шкалы, либо провести непараметрические тесты.

Проведем непараметрический тест Манна-Уитни, нулевая гипотеза которого говорит о том, что в рассматриваемых выборках отсутствуют различия

In [34]:
```
from scipy.stats import mannwhitneyu
```

In [35]:
```
mannwhitneyu(CR_weekend, CR_workday)
```

Out[35]:
```
MannwhitneyuResult(statistic=283139546.0, pvalue=6.278453494676855e-19)
```

p-value < 0.05, поэтому принимается альтернативная гипотеза, которая говорит о различиях в CR в будние дни и в выходные. Данный тест отвечает на поставленный вопрос, а именно, мы с 95% уверенностью можем говорить о различиях в CR в будни и выходные. Для более глубокого анализа можем использовать bootstrap, чтобы конкретизировать, по каким параметрам есть различия в выборках

теперь посмотрим на конверсию из корзины в транзакцию

In [36]:
```
df.query('checkout > 0').shape[0]
```

Out[36]:
```
42289
```

In [37]:
```
df.query('checkout > 0').CR_prev.describe()
```

Out[37]:
```
count    42289.000000
mean        14.011075
std         10.019568
min          0.000000
25%          6.990000
50%         13.640000
75%         19.750000
max        300.000000
Name: CR_prev, dtype: float64
```

И снова есть значения CR > 100!

In [38]:
```
df.query('checkout > 0').query('CR_prev > 100')
```

Out[38]:

	date	source	medium	delivery_available	device_type	promo_activated	filter_used	p
18484	2020-02-18	cityads	cpa	Не определено	Десктоп	no	no	
28138	2020-03-04	cityads	cpa	Не определено	Десктоп	no	no	
44992	2020-04-06	cityads	cpa	Не определено	Десктоп	no	no	
50806	2020-06-04	newsletter	email	Не определено	Десктоп	yes	no	

In [39]:
```
#  уберем 4 строки, которые дают CR > 100
CR_prev_df = df.query('checkout > 0').query('CR_prev <= 100')
```

In [40]:
```
CR_prev_df['weekend'] = CR_prev_df.day_week.apply(weekends)
#  Фильтруем датафрейм по дням недели
CR_prev_workday = CR_prev_df[CR_prev_df['weekend'] == 'workday'].CR_prev
CR_prev_weekend = CR_prev_df[CR_prev_df['weekend'] == 'weekend'].CR_prev
```

Также проверим с помощью критерия Шапиро распределения

In [41]:
```
shapiro(CR_prev_workday.sample(1000, random_state=17))
shapiro(CR_prev_weekend.sample(1000, random_state=17))
```

Out[41]:
```
ShapiroResult(statistic=0.9539801478385925, pvalue=3.686288709939695e-17)
```

Out[41]:
```
ShapiroResult(statistic=0.9542859196662903, pvalue=4.238087992556264e-17)
```

In [42]:
```
mannwhitneyu(CR_prev_workday, CR_prev_weekend)
```

Out[42]:
```
MannwhitneyuResult(statistic=182606638.0, pvalue=4.910094018013949e-05)
```

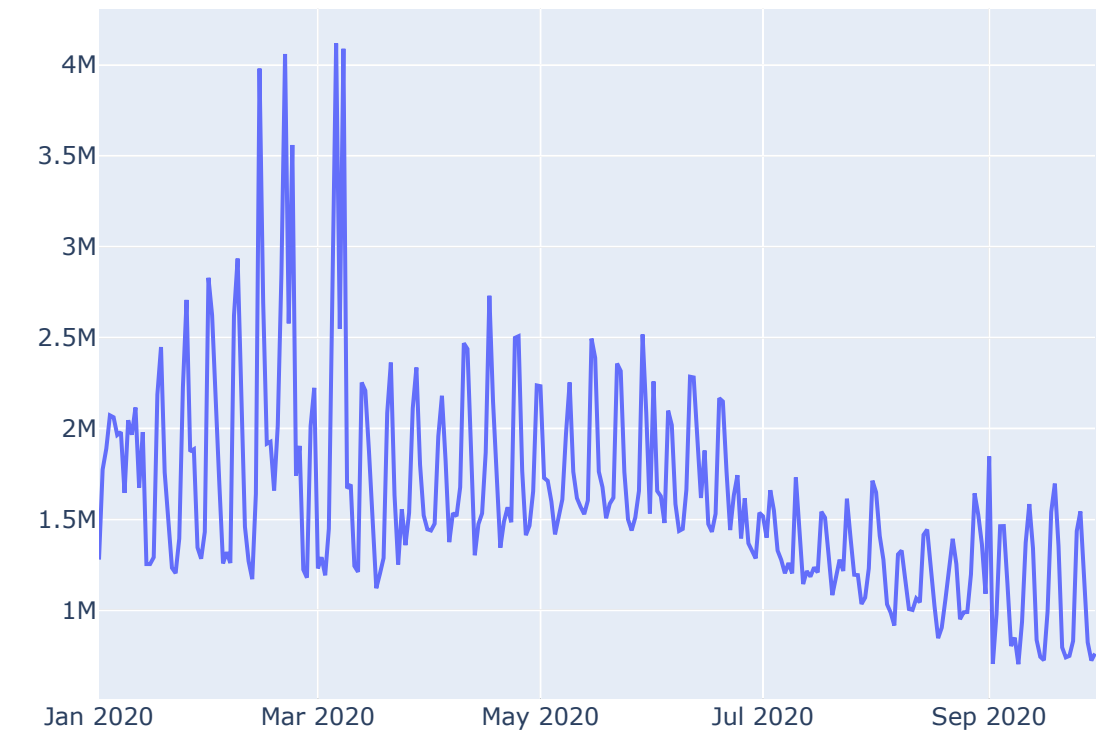
d. Вам необходимо спрогнозировать объем дохода, полученного с пользователей,приведенных на сайт контекстной рекламой (medium = cpc) на полгода вперед. Опишите, как бы вы подошли к этой задаче и какие дополнительные данные вам понадобятся?

Для формирования прогноза необходимо подготовить датафрейм имеющихся данных (отфильтровать, сгруппировать по дате и агрегировать). Далее, по данным, имеющимся в распоряжении, обучается модель для прогноза значений, проверяется ее точность и предсказывается необходимый период. В идеале необходимо учитывать факторы, влияющие на тенденцию, такие как акции, которые собираются внедрять.

In [43]:
```
# Получим датафрейм с суммой продаж по дням
prediction_df = df.query('medium == "cpc"').groupby('date').agg({'revenue':
'sum'}).sort_values('date')
```

In [44]:
```
# Построим график продаж за имеющийся период
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
from plotly import graph_objs as go

# инициализируем plotly
init_notebook_mode(connected = False)

# опишем функцию, которая будет визуализировать все колонки dataframe в виде
line plot
def plotly_df(df, title = ''):
    data = []
```

```
    for column in df.columns:
        trace = go.Scatter(
            x = df.index,
            y = df[column],
            mode = 'lines',
            name = column
        )
        data.append(trace)

    layout = dict(title = title)
    fig = dict(data = data, layout = layout)
    iplot(fig, show_link=False)

plotly_df(prediction_df, title = 'Продажи за имеющийся период')
```



Реализация ниже для построения графика офлайн

In [57]:
```
# from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
iplot
# from plotly import graph_objs as go

# # инициализируем plotly
# init_notebook_mode(connected = False)

# # опишем функцию, которая будет визуализировать все колонки dataframe в
виде line plot
# def plotly_df(df, title = ''):
#     data = []

#     for column in df.columns:
#         trace = go.Scatter(
#             x = df.index,
#             y = df[column],
#             mode = 'lines',
#             name = column
#         )
#         data.append(trace)

#     layout = dict(title = title)
#     fig = go.Figure(data = data, layout=layout);
#     fig.update_layout(
#         autosize=False,
#         width=2000,
#         height=1000
#     ).show(renderer='jpeg')

# plotly_df(prediction_df, title = 'Продажи за имеющийся период')
```

In [46]:
```
#импортируем библиотеку для прогноза
from prophet import Prophet
import holidays
# задаем период, для которого проверяем предсказательную модель
predictions = 30

# приводим dataframe к нужному формату
pred_df = prediction_df.reset_index()
pred_df.columns = ['ds', 'y']

# отрезаем из обучающей выборки последние точки, на которых проверяем
предсказание
train_df = pred_df[:-predictions]
```

In [47]:
```
# задаем праздничные дни, чтобы модель их учитывала
holidays_dict = holidays.RU(years=(2020, 2021))
df_holidays = pd.DataFrame.from_dict(holidays_dict, orient='index') \
    .reset_index()
df_holidays = df_holidays.rename({'index':'ds', 0:'holiday'}, axis
='columns')
df_holidays['ds'] = pd.to_datetime(df_holidays.ds)
df_holidays = df_holidays.sort_values(by=['ds'])
df_holidays = df_holidays.reset_index(drop=True)
```

In [48]:
```
# Создаем объект класса Prophet и передадим ему тренировочные данные
m = Prophet(holidays=df_holidays, daily_seasonality=False,
weekly_seasonality=True, yearly_seasonality=False)
m.fit(train_df)
```

```
15:57:42 - cmdstanpy - INFO - Chain [1] start processing
15:57:42 - cmdstanpy - INFO - Chain [1] done processing
```

Out[48]:
```
<prophet.forecaster.Prophet at 0x20fcc519100>
```

In [49]:
```
# создаем датафрейм для предсказания 30 дней с помощью метода
make_future_dataframe
future = m.make_future_dataframe(periods=predictions)
forecast = m.predict(future)
```

In [50]:
```
# объединяем полученные значения модели с начальными данными
cmp_df = forecast.set_index('ds')[['yhat', 'yhat_lower',
'yhat_upper']].join(pred_df.set_index('ds'))
```

Проверяем качество модели по двум показателям: MAPE - mean absolute percentage error и MAP - mean absolute error. Чем ниже ошибка, тем точнее модель

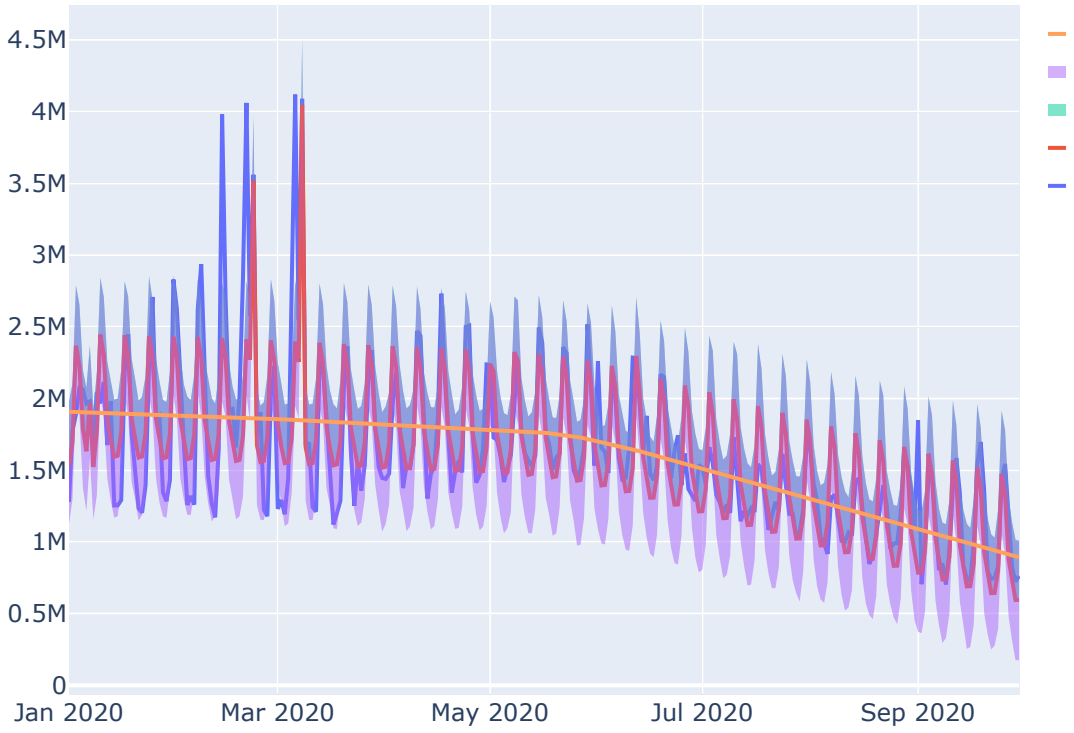
In [51]:
```
import numpy as np
cmp_df['e'] = cmp_df['y'] - cmp_df['yhat']
cmp_df['p'] = 100*cmp_df['e']/cmp_df['y']
print ('MAPE', np.mean(abs(cmp_df[-predictions:]['p'])))
print ('MAE', np.mean(abs(cmp_df[-predictions:]['e'])))
```

```
MAPE 12.913565554403016
MAE 156804.808333871
```

Точность модели составляет 12%, а ошибка в абсолютных значениях порядка 156 тысяч, визуализируем полученные данные

In [52]:

```
# офлайн график
data = [go.Scatter(x=pred_df['ds'], y=pred_df['y'], name='fact'),
        go.Scatter(x=forecast['ds'], y=forecast['yhat'], name='yhat'),
        go.Scatter(x=forecast['ds'], y=forecast['yhat_upper'],
fill='tonexty', mode='none', name='upper'),
        go.Scatter(x=forecast['ds'], y=forecast['yhat_lower'],
fill='tonexty', mode='none', name='lower'),
        go.Scatter(x=forecast['ds'], y=forecast['trend'], name='trend')
       ]
iplot(data)
# блок для отображения графика в github
# fig = go.Figure(data = data);
# fig.update_layout(
#     autosize=False,
#     width=2000,
#     height=1000
# ).show(renderer='jpeg')
```



Теперь спрогнозируем с помощью нашей модели данные на полгода вперед

In [53]:
```
# задаем интересующий нас период прогноза
future_predictions = 180
final_train_df = pred_df
```

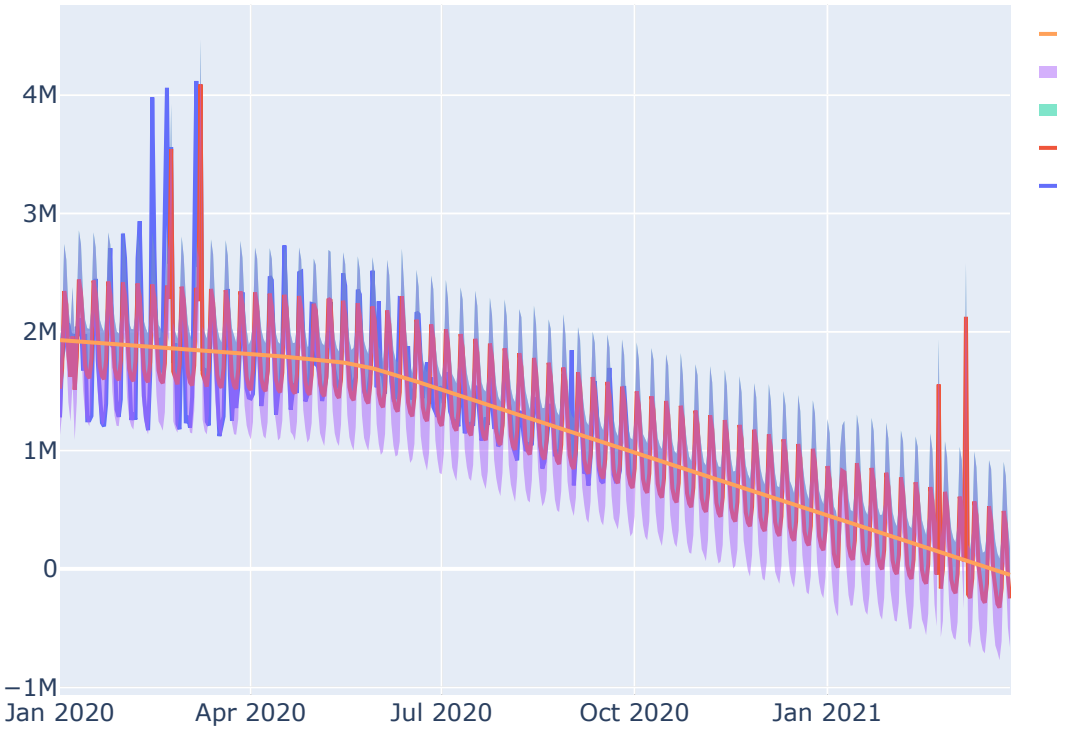
In [54]:
```
# теперь модель будет обучаться на всем интервале данных
f = Prophet(holidays=df_holidays, daily_seasonality=False,
weekly_seasonality=True, yearly_seasonality=False)
f.fit(final_train_df)
```

```
15:57:43 - cmdstanpy - INFO - Chain [1] start processing
15:57:43 - cmdstanpy - INFO - Chain [1] done processing
```

Out[54]:
```
<prophet.forecaster.Prophet at 0x20fc3acb9d0>
```

In [55]:
```
final_future = f.make_future_dataframe(periods=future_predictions)
final_forecast = f.predict(final_future)
```

In [56]:
```
data = [go.Scatter(x=pred_df['ds'], y=pred_df['y'], name='fact'),
    go.Scatter(x=final_forecast['ds'], y=final_forecast['yhat'],
name='yhat'),
    go.Scatter(x=final_forecast['ds'], y=final_forecast['yhat_upper'],
fill='tonexty', mode='none', name='upper'),
    go.Scatter(x=final_forecast['ds'], y=final_forecast['yhat_lower'],
fill='tonexty', mode='none', name='lower'),
    go.Scatter(x=final_forecast['ds'], y=final_forecast['trend'],
name='trend')
       ]
iplot(data)
# Также для отображения на github
# fig = go.Figure(data = data);
# fig.update_layout(
#     autosize=False,
#     width=3000,
#     height=1000
# ).show(renderer='jpeg')
```



Из-за падающей линии тренда и неполных данных за год, предсказанная модель имеет данный вид. Учтены выбросы в праздничные дни 14 и 23 февраля, а также 8 марта из-за добавления праздников в предсказательную модель.

Как было сказано ранее, при наличии бОльшего числа данных по времени, а также дополнительных параметров, влияющих на прибыль, модель будет точнее