

Анализ временных рядов и предсказание количества поездок на неделю вперед

В этом ноутбуке данные группируются по количеству поездок за день, проводится анализ полученного временного ряда, делаются предсказания количества поездок на неделю вперед.

```
In [1]: 1 import scipy as sps
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import pandas as pd
6 from tqdm.notebook import tqdm
7
8 from sklearn.metrics import mean_squared_error as MSE
9
10 import plotly.graph_objects as go
11 import plotly.express as px
12 import plotly.offline
13
14 from statsmodels.tsa.seasonal import seasonal_decompose
15 import statsmodels.api as sm
16 from statsmodels.tsa.stattools import kpss
17 import gc
18 #from fbprophet import Prophet
19
20 import warnings
21
22 warnings.filterwarnings('ignore')
23 sns.set(font_scale=1.5)
```

Загрузим данные о поездках.

```
In [2]: 1 trips = pd.read_csv('cycle-share-dataset/trip.csv', error_bad_lines=False,
2 parse_dates=[1, 2])
```

```
b'Skipping line 50794: expected 12 fields, saw 20\n'
```

In [3]:

```
1 trips.head()
```

Out[3]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name	from_s
0	431	2014-10-13 10:31:00	2014-10-13 10:48:00	SEA00298	985.935	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
1	432	2014-10-13 10:32:00	2014-10-13 10:48:00	SEA00195	926.375	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
2	433	2014-10-13 10:33:00	2014-10-13 10:48:00	SEA00486	883.831	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
3	434	2014-10-13 10:34:00	2014-10-13 10:48:00	SEA00333	865.937	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
4	435	2014-10-13 10:34:00	2014-10-13 10:49:00	SEA00202	923.923	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	



Глава 1. Анализ временных рядов

1. Кол-во поездок в зависимости от даты

Отообразим количество поездок в зависимости от даты.

Преобразуем сначала данные для удобства подсчёта.

```
In [4]: 1 time_data = trips[['trip_id', 'starttime', 'stoptime']].copy()
2
3 time_data['start_year'] = time_data['starttime'].dt.year
4 time_data['start_month'] = time_data['starttime'].dt.month
5 time_data['start_day'] = time_data['starttime'].dt.day
6 time_data['start_date'] = time_data['starttime'].dt.date
7 time_data['start_time'] = time_data['starttime'].dt.time
8 time_data['start_weekday'] = time_data['starttime'].dt.dayofweek
9
10 time_data['stop_year'] = time_data['stoptime'].dt.year
11 time_data['stop_month'] = time_data['stoptime'].dt.month
12 time_data['stop_day'] = time_data['stoptime'].dt.day
13 time_data['stop_date'] = time_data['stoptime'].dt.date
14 time_data['stop_time'] = time_data['stoptime'].dt.time
15 time_data['stop_weekday'] = time_data['stoptime'].dt.dayofweek
16
17 time_data.drop(columns=['starttime', 'stoptime'], inplace=True)
18
19 time_data.head()
```

Out[4]:

	trip_id	start_year	start_month	start_day	start_date	start_time	start_weekday	stop_year	stop
0	431	2014	10	13	2014-10-13	10:31:00	0	2014	
1	432	2014	10	13	2014-10-13	10:32:00	0	2014	
2	433	2014	10	13	2014-10-13	10:33:00	0	2014	
3	434	2014	10	13	2014-10-13	10:34:00	0	2014	
4	435	2014	10	13	2014-10-13	10:34:00	0	2014	

```
In [5]: 1 trips_by_date = time_data.groupby('start_date').count()['trip_id']
2
3 trips_by_date
```

Out[5]:

start_date	
2014-10-13	818
2014-10-14	982
2014-10-15	626
2014-10-16	790
2014-10-17	588
...	
2016-08-27	333
2016-08-28	392
2016-08-29	369
2016-08-30	375
2016-08-31	319

Name: trip_id, Length: 689, dtype: int64

```
In [8]: 1 plt.figure(figsize=(20, 6))
2         sns.set_style('darkgrid')
3
4         plt.title('Зависимость количества поездок от даты')
5
6         plt.plot(trips_by_date.index, trips_by_date)
7         plt.xlabel('Дата начала поездки')
8         plt.ylabel('Количество')
9
10        plt.show()
```



Наблюдения: видно что в зависимости от даты у кол-ва поездок меняется дисперсия. Также на многих промежутках времени явно прослеживается тренд, поэтому скорее всего ряд не стационарен

2. Стационарность

Проверим ряд на стационарность с помощью критерия KPSS.

```
In [9]: 1 kpss(trips_by_date)
```

```
Out[9]: (1.5431653515342676,
0.01,
20,
{'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

Наблюдение

Гипотеза о стационарности отверглась. Значит, имеет смысл исследовать ряд на тренд и сезонность.

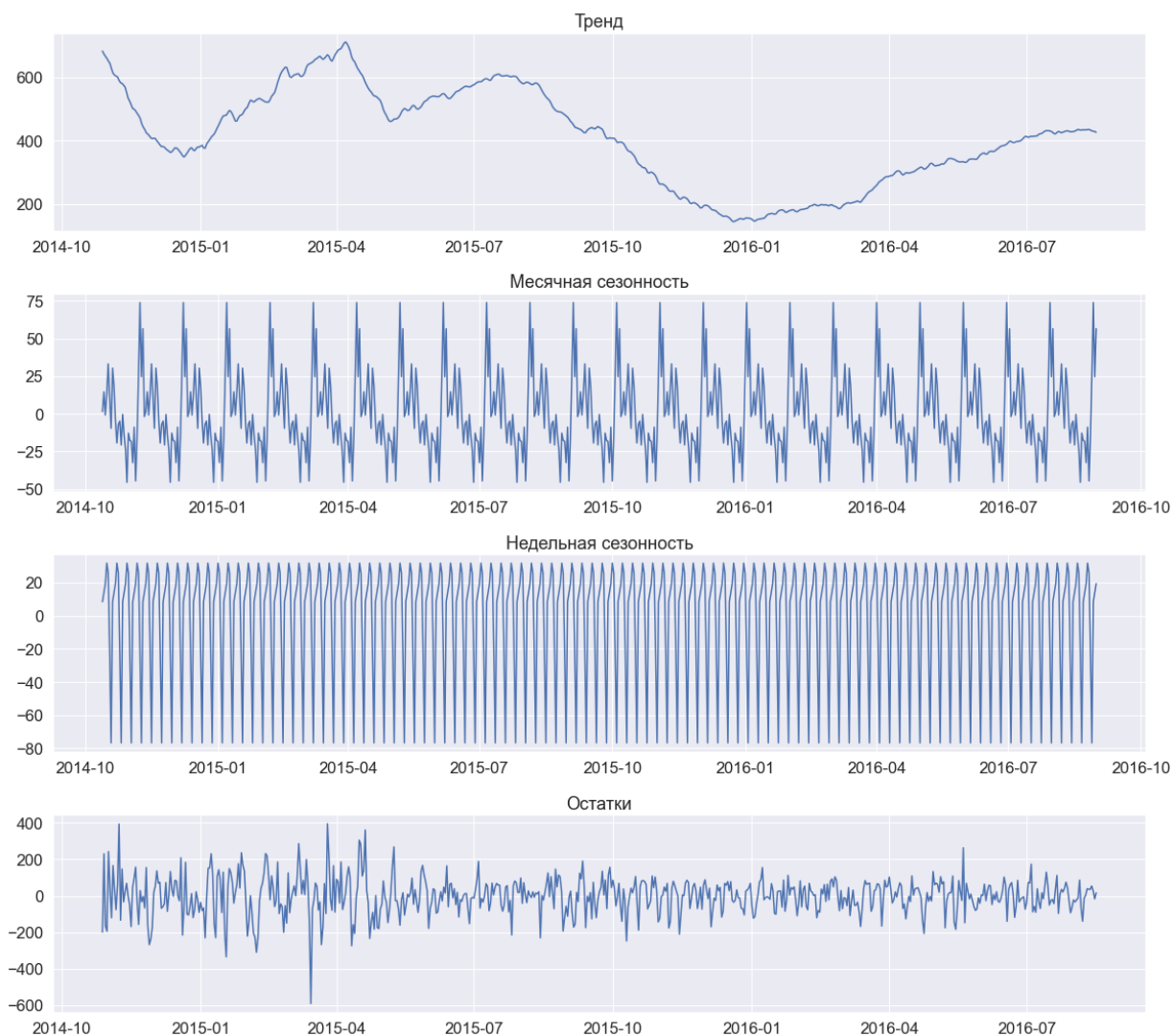
3. STL-декомпозиция

Применим STL-декомпозицию к полученному временному ряду.

```
In [20]: 1 decomposed = seasonal_decompose(trips_by_date, freq=30)
          2
          3 day_decomp = seasonal_decompose(trips_by_date, freq=7)
```

In [21]:

```
1 plt.figure(figsize=(17, 15))
2
3 plt.subplot(411)
4 plt.plot(trips_by_date.index, decomposed.trend)
5 plt.title('Тренд')
6
7 plt.subplot(412)
8 plt.plot(trips_by_date.index, decomposed.seasonal)
9 plt.title('Месячная сезонность')
10
11 plt.subplot(413)
12 plt.plot(trips_by_date.index, day_decomp.seasonal)
13 plt.title('Недельная сезонность')
14
15 plt.subplot(414)
16 plt.plot(trips_by_date.index, decomposed.resid)
17 plt.title('Остатки')
18
19 plt.tight_layout()
```



Наблюдение

На графиках отчётливо прослеживается недельная и месячная сезонность. Исследовать годовую сезонность с помощью декомпозиции не представляется возможным, т.к. в датасете нет данных за полных два года.

4. Автокорреляция

Построим теперь графики автокорреляций для продифференцированного ряда, т.к. в данных, очевидно, наблюдается и тренд, и сезонность.

Для удаления тренда и всех сезонностей продифференцируем ряд несколько раз.

```
In [77]: 1 diff = trips_by_date.values[1:] - trips_by_date.values[:-1]
2 month_diff = diff[30:] - diff[:-30]
3 week_diff = month_diff[7:] - month_diff[:-7]
```

```
In [91]: 1 plt.figure(figsize=(14, 7))
2 plt.title('Ряд после дифференцирования')
3 plt.plot(trips_by_date.index.values[38:], week_diff)
4 plt.show()
```



Наблюдение: тренд и сезонность кажется убраны, но дисперсия все равно не стабильна.

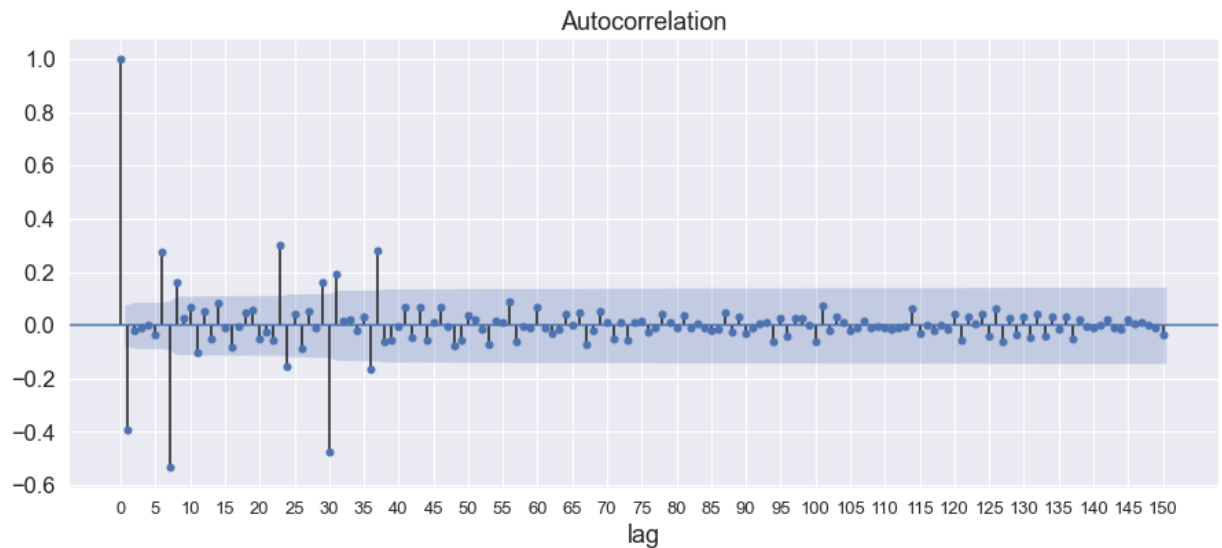
Проверим гипотезу о стационарности.

```
In [92]: 1 kpss(week_diff)
```

```
Out[92]: (0.022038395790409795,
0.1,
20,
{'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

Как видим теперь гипотеза о стационарности не отвергается

```
In [93]: 1 fig = plt.figure(figsize=(15, 6))
2
3 ax = fig.add_subplot(111)
4 fig = sm.graphics.tsa.plot_acf(week_diff, lags=150, ax=ax)
5 ax.set_xticks(np.arange(0, 151, 5))
6
7 plt.setp(ax.get_xticklabels(), fontsize=13)
8 ax.set_xlabel('lag')
9
10 plt.show()
```



Вывод

Значимыми оказались лаги 1, 6, 7, 23, 24, 30, 36, 37. Значения 7 и 30 учитывать не стоит, т.к. дифференцирование могло "снять" не всю сезонность. Остальные результаты помогут в подборе параметров предсказательных моделей. Таких, как, например, ARIMA.

Глава 2. Предсказание поездок на неделю вперед

0. Разбиение данных

Разобьем поездки на интервалы по 30 минут.

In [6]: 1 time_data.head()

Out[6]:

	trip_id	start_year	start_month	start_day	start_date	start_time	start_weekday	stop_year	stop
0	431	2014	10	13	2014-10-13	10:31:00	0	2014	
1	432	2014	10	13	2014-10-13	10:32:00	0	2014	
2	433	2014	10	13	2014-10-13	10:33:00	0	2014	
3	434	2014	10	13	2014-10-13	10:34:00	0	2014	
4	435	2014	10	13	2014-10-13	10:34:00	0	2014	

In [7]:

```
1 def time_round(s):
2     if int(s[0]) >= 3:
3         return '30'
4     else:
5         return '00'
6
7 time_data.start_time = time_data.start_time.values.astype(str)
8 time_data['time'] = [str(day) + ' ' + s[0:3] + time_round(s[3:4]) + ':00'
9                     for s, day in zip(time_data.start_time.values, time_data.start_time.index)]
```

In [8]: 1 time_data['time'] = [pd.to_datetime(s) for s in time_data.time.values]

In [9]: 1 series = time_data.groupby('time').count()['trip_id'].to_frame()

In [10]: 1 series['count'] = series.trip_id
2 series = series.drop(columns=['trip_id'])

```
In [11]: 1 series
```

Out[11]:

	count
time	
2014-10-13 10:30:00	12
2014-10-13 11:30:00	108
2014-10-13 12:00:00	42
2014-10-13 12:30:00	42
2014-10-13 13:00:00	36
...	...
2016-08-31 21:00:00	3
2016-08-31 21:30:00	2
2016-08-31 22:00:00	4
2016-08-31 22:30:00	4
2016-08-31 23:30:00	5

26195 rows × 1 columns

Будем предсказывать кол-во взятых в прокат велосипедов на неделю вперед по данным за последние полгода. Понятно, что так мы сможем учесть и месячную, и недельную сезонность, в то же время мы не можем ничего говорить о годовой сезонности, поэтому данные почти за два года будут излишними, более того, из-за них можно будет не уследить за краткосрочным трендом.

```
In [12]: 1 test_size = 48*7
          2 train_size = 48*30*6
          3 train = series.tail(test_size + train_size).head(train_size)
          4 test = series.tail(test_size)
```

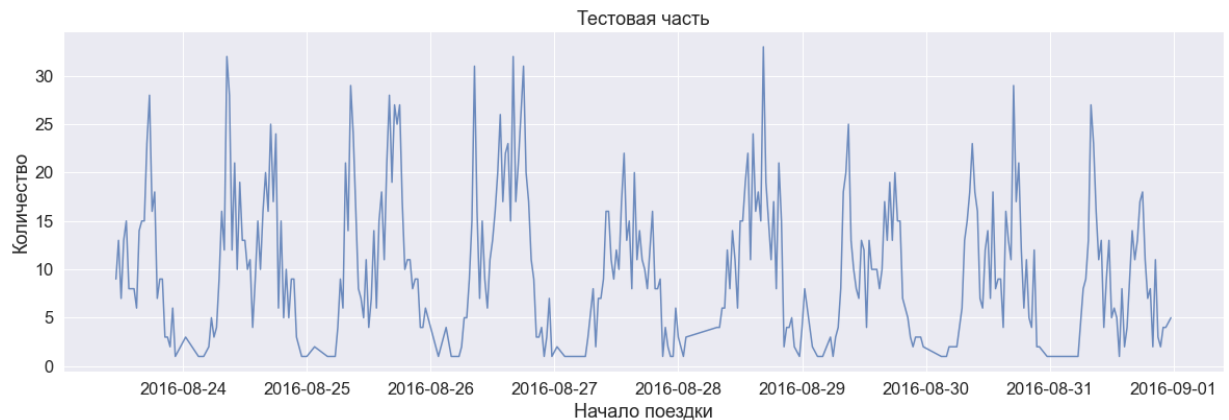
Взглянем на ряд

In [13]:

```
1 plt.figure(figsize=(20, 6))
2 sns.set_style('darkgrid')
3
4 plt.title('Обучающая часть')
5
6 plt.plot(train.index, train['count'].values, alpha=0.8)
7 plt.xlabel('Начало поездки')
8 plt.ylabel('Количество')
9
10 plt.show()
```



```
In [14]: 1 plt.figure(figsize=(20, 6))
2         sns.set_style('darkgrid')
3
4         plt.title('Тестовая часть')
5
6         plt.plot(test.index, test['count'].values, alpha=0.8)
7         plt.xlabel('Начало поездки')
8         plt.ylabel('Количество')
9
10        plt.show()
```



1. Бейзлайны - среднее и последнее наблюдаемое значение.

Во временных рядах сильными бейзлайнами являются среднее и последнее наблюдаемое значение. Благодаря ним поймем, на какой MSE стоит ориентироваться.

```
In [15]: 1 mean_preds = np.array([train['count'].mean()] * test_size)
```

```
In [16]: 1 def print_preds(test, preds, name):
2         plt.figure(figsize=(20, 6))
3         sns.set_style('darkgrid')
4
5         plt.title('Тестовая часть и предсказания')
6
7         plt.plot(test.index, test['count'].values, alpha=0.8,
8                 label='test')
9         plt.plot(test.index, preds, alpha=0.9,
10                label=name + ' preds')
11        plt.xlabel('Начало поездки')
12        plt.ylabel('Количество')
13        plt.legend()
14        plt.show()
```

```
In [17]: 1 mse = MSE(test['count'].values, mean_preds)
2 print(f'MSE = {mse}')
3 print_preds(test, mean_preds, name='mean')
```

MSE = 58.514083956434945



```
In [18]: 1 last_preds = np.array([train['count'].values[-1]] * test_size)
2 mse = MSE(test['count'].values, last_preds)
3 print(f'MSE = {mse}')
4 print_preds(test, mean_preds, name='last')
```

MSE = 72.96130952380952



Пока будем равняться на бейзлайн средних предсказаний

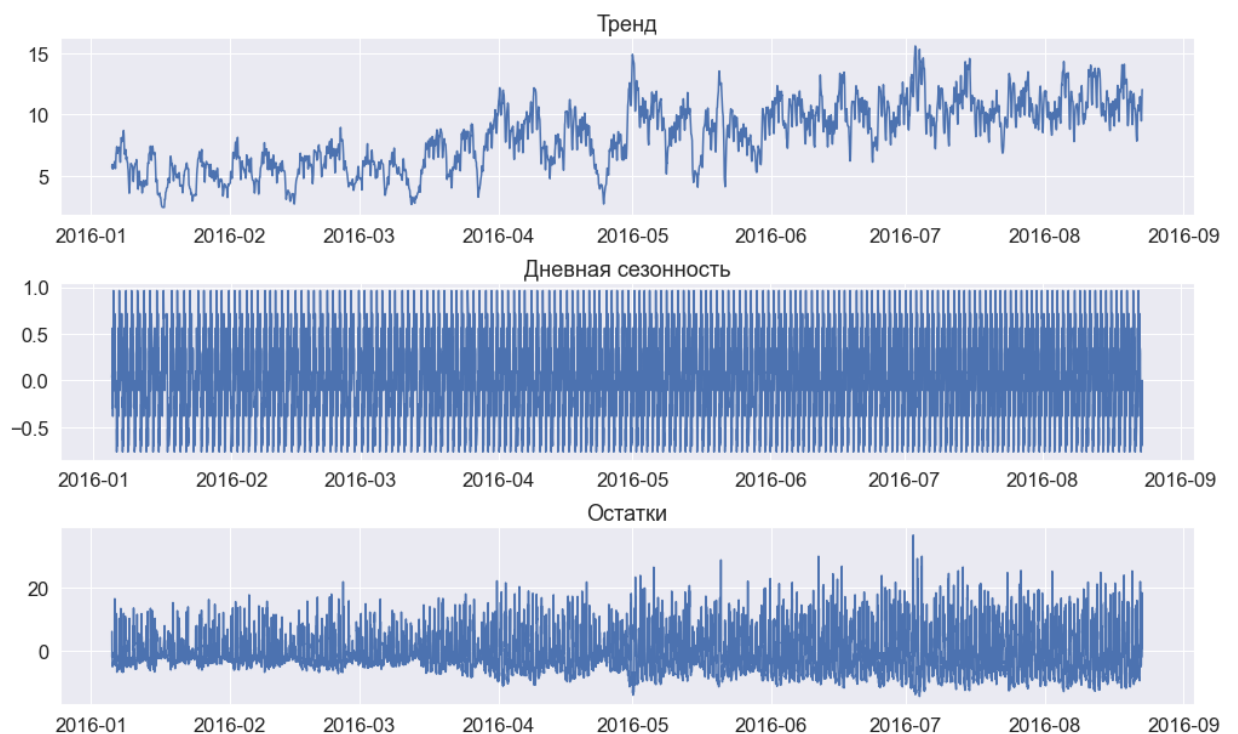
2. Подбор параметров для SARIMA

Из анализа проведенного выше понятно, что есть месячная и недельная сезонность. Рассмотрим дневную.

```

In [19]: 1 gc.collect()
2 decomposed = seasonal_decompose(train['count'].values, freq=48)
3
4 plt.figure(figsize=(14, 11))
5
6 plt.subplot(411)
7 plt.plot(train.index, decomposed.trend)
8 plt.title('Тренд')
9
10 plt.subplot(412)
11 plt.plot(train.index, decomposed.seasonal)
12 plt.title('Дневная сезонность')
13
14 plt.subplot(413)
15 plt.plot(train.index, decomposed.resid)
16 plt.title('Остатки')
17
18 plt.tight_layout()

```



Дневная сезонность действительно очевидна.

Для подбора параметров для SARIMA нужно смотреть на значимые лаги коррелограммы после приведения ряда к стационарному.

```

In [20]: 1 train_count = train['count'].values

```

```
In [21]: 1 kpss(train_count)
```

```
Out[21]: (13.559720625081978,  
0.01,  
37,  
{'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

Как видим, пока что гипотеза о стационарности отвергается

```
In [22]: 1 diff = train_count[1:] - train_count[:(-1)] # снимаем тренд  
2 diff = diff[(48*30):] - diff[:(-48*30)]  
3 diff = diff[(48*7):] - diff[:(-48*7)]  
4 diff = diff[(48):] - diff[:(-48)]
```

```
In [23]: 1 kpss(diff)
```

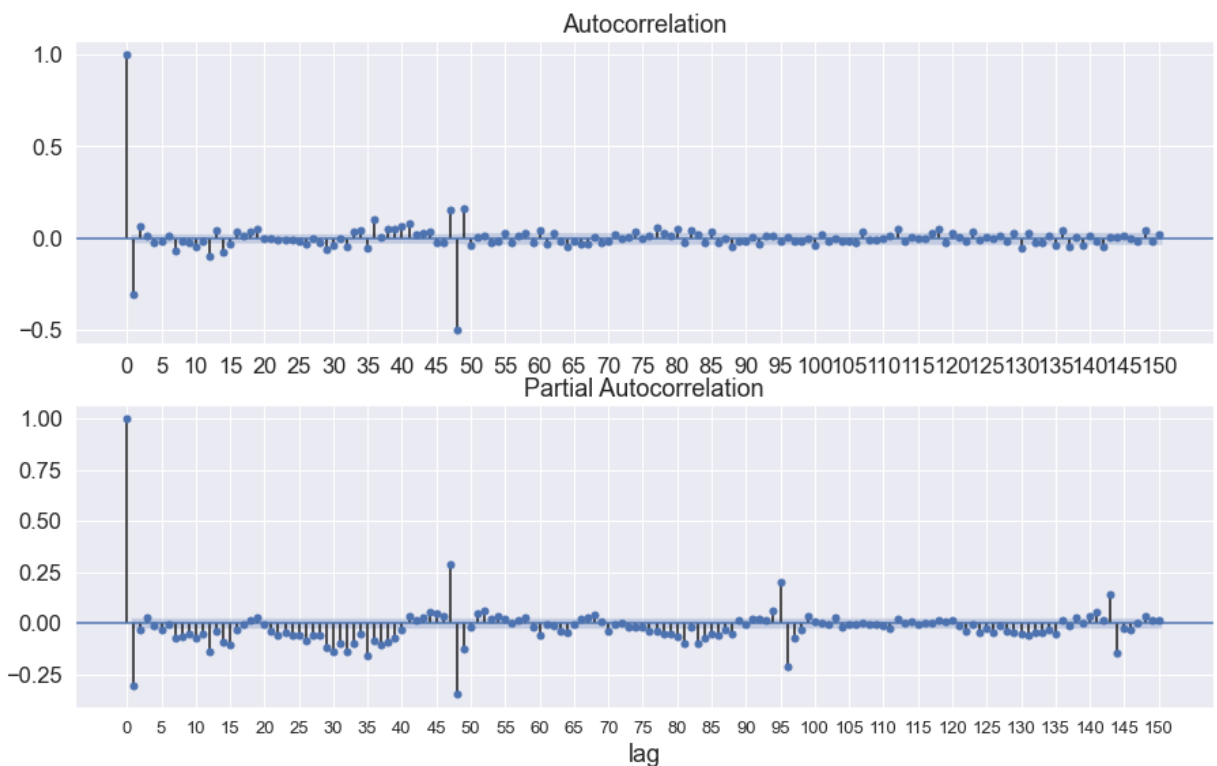
```
Out[23]: (0.005546095525755831,  
0.1,  
35,  
{'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

Теперь не отвергается

```

In [24]: 1 fig = plt.figure(figsize=(15, 9))
2
3 ax = fig.add_subplot(211)
4 fig = sm.graphics.tsa.plot_acf(diff, lags=150, ax=ax)
5 ax.set_xticks(np.arange(0, 151, 5))
6
7 ax = fig.add_subplot(212)
8 fig = sm.graphics.tsa.plot_pacf(diff, lags=150, ax=ax)
9 ax.set_xticks(np.arange(0, 151, 5))
10
11 plt.setp(ax.get_xticklabels(), fontsize=13)
12 ax.set_xlabel('lag')
13
14 plt.show()

```



Результат: автокорреляция принимает большие значения для первых 2-х лагов, поэтому возьмем $p = 2$. Частичная автокорреляция принимает большие значения для первых трех лагов, поэтому возьмем $q = 3$. На лаги, соответствующие сезонности, не обращаем внимание, так как дифференцируемость не обязательно снимает всю сезонность.

3. Обучение SARIMAX


```
In [25]: 1 %%time
2 param = (2, 1, 3)
3 param_seasonal = (2, 1, 2, 48)
4 model = sm.tsa.statespace.SARIMAX(train_count, order=param,
5                                   seasonal_order=param_seasonal)
6 model = model.fit()
7 print('ARIMA{x{x}} - AIC: {:.2f}'.format(param,
8                                           param_seasonal,
9                                           model.aic))
```

ARIMA(2, 1, 3)x(2, 1, 2, 48) - AIC: 50710.26
Wall time: 1h 43s

```
In [30]: 1 train_count.shape
```

Out[30]: (8640,)

```
In [35]: 1 preds = model.predict(start=0, end=8640, dynamic=True)
```

...

```
In [ ]: 1 preds
```

```
In [71]: 1 mse = MSE(test['count'].values, preds)
2 print(f'MSE = {mse}')
3 print_preds(test, preds, name='sarimax')
```

MSE = 109.65788942323996



4. fb prophet

```
In [ ]: 1
```

