Сегодня мы будем работать с линейной регрессией в библиотеке sklearn. Воспользуемся классами с различным типом регуляризации и подберем оптимальные гипер-параметры для этих моделей. Решать будем задачу с Kaggle про предсказание длины поездки в такси New York City Taxi Trip Duration (https://www.kaggle.com/c/nyc-taxi-trip-duration/overview).

Первым делом необходимо скачать данные. Воспользуйтесь следующей инструкцией: Как скачать данные с kaggle в colab. (https://medium.com/@saedhussain/google-colaboratory-and-kaggle-datasets-b57a83eb6ef8) Ниже есть необходимый код, вам нужно лишь запрость токен на kaggle и загрузить его.

In [ ]:

```
!pip install -q kaggle
```

In [ ]:

```
from google.colab import files
uploaded = files.upload()
```

In [ ]:

```
!mkdir /root/.kaggle
!mv kaggle.json /root/.kaggle/kaggle.json
!kaggle competitions download -c nyc-taxi-trip-duration
```

Распакуем данные, которые мы загрузили. Работать будем только с train частью, так как там имеются значения предсказываемой переменной.

In [ ]:

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

df = pd.read_csv('train.zip', compression='zip', header=0, sep=',', quotechar='"')
```

In [ ]:

```
df.head()
```

Out[ ]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitu
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.9821
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.9804
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.9790
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.0100
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.9730

Мы видим информацию о каждой поездке. Нам известны координаты, время начала поездки, количество пассажиров и т.д. Удалим колонку, которая есть только в обучающей выборке `dropoff_datetime` . Из названия понятно, что используя эту колонку и `pickup_datetime` мы сможем восстановить длину поездки. Очевидно, что в начале поездки `dropoff_datetime` нам недоступна, а значит и для предсказания ее использовать нельзя.

In [ ]:

```
df = df.drop('dropoff_datetime', axis=1)
```

Сейчас даты записаны в виде строк. Давайте преобразуем их в питонячие datetime объекты. Таким образом мы сможем выполнять арифметические операции с датами и вытаскивать нужную информацию, не работая со строками.

In [ ]:

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
```

Давайте разобьем выборку на train и test. Применить функцию `train_test_split` в этот раз не получиться. Мы теперь имеем дело с временными данными и на практике наша модель должна уметь работать во временных периодах, которых нет в обучающей выборке. Поэтому разбивать мы будем датасет по хронологии. Для этого отсортируем датасет по дате и возьмем первые N строк.

In [ ]:

```
df = df.sort_values(by='pickup_datetime')
```

In [ ]:

```
df.head()
```

Out[ ]:

	id	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_la
96469	id0190469	2	2016-01-01 00:00:17	5	-73.981743	40.7
223872	id1665586	1	2016-01-01 00:00:53	1	-73.985085	40.7
713067	id1210365	2	2016-01-01 00:01:01	5	-73.965279	40.8
652463	id3888279	1	2016-01-01 00:01:14	1	-73.982292	40.7
722901	id0924227	1	2016-01-01 00:01:20	1	-73.970108	40.7

In [ ]:

```
df_train = df[:10 ** 6]
df_test = df[10 ** 6:]
```

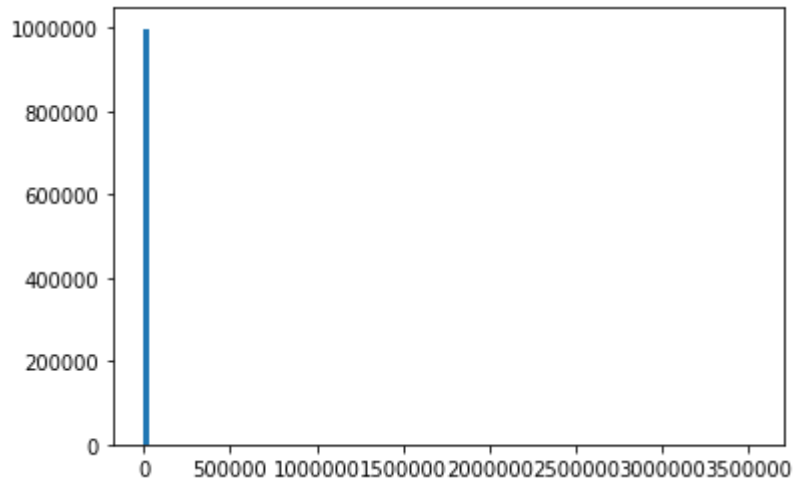
Напомню, что мы будем пресказывать переменную trip_duration. Посмотрим на target переменную.

In [ ]:

```
df_train.trip_duration.hist(bins=100, grid=False, )
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa68dd420f0>
```



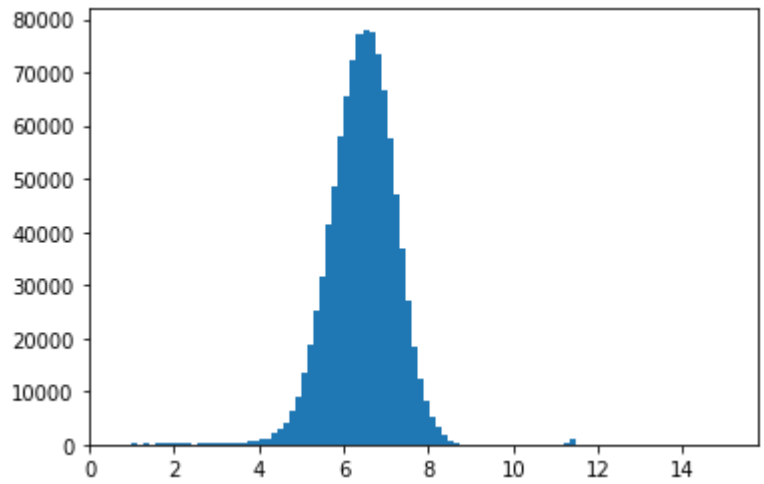
Что то пошло не так. Вероятно, есть очень длинные поездки и короткие. Попробуем взять `log(1 + x)` от длины поездки. Единицу мы прибавляем, чтобы избежать проблем с поездками, которые например мнгновенно завершились.

In [ ]:

```
import numpy as np
np.log1p(df_train.trip_duration).hist(bins=100, grid=False, )
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa68d728eb8>
```



Мы получили куда более ясную картину, распределение стало похоже на нормальное. Работать будем теперь с логарифмом. Так линейной регрессии будет куда проще выучить корректную зависимость. А если захотим вернуться к исходным данным, возведем предсказание в экспоненту.

In [ ]:

```
df_train['log_trip_duration'] = np.log1p(df_train.trip_duration)
df_test['log_trip_duration'] = np.log1p(df_test.trip_duration)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:

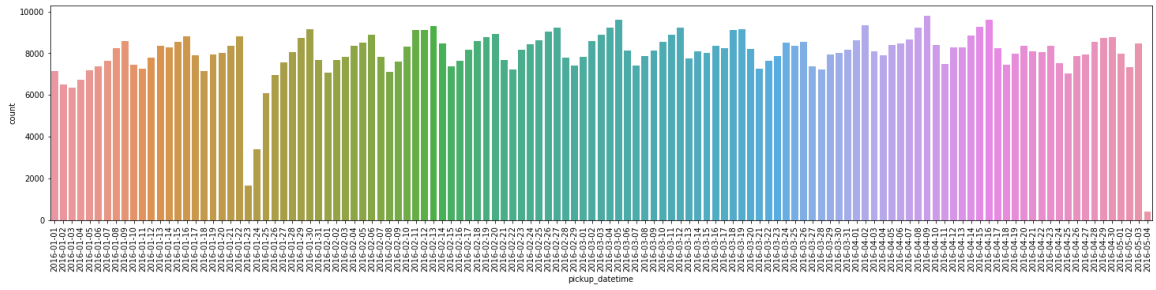
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
```

Посмотрим на наши признаки. Давайте нарисуем, как выглядит распределение количества поездок по дням.

In [ ]:

```
date_sorted = df_train.pickup_datetime.apply(lambda x: x.date()).sort_values()

plt.figure(figsize=(25, 5))
date_count_plot = sns.countplot(
  x=date_sorted,
)
date_count_plot.set_xticklabels(date_count_plot.get_xticklabels(), rotation=90);
```



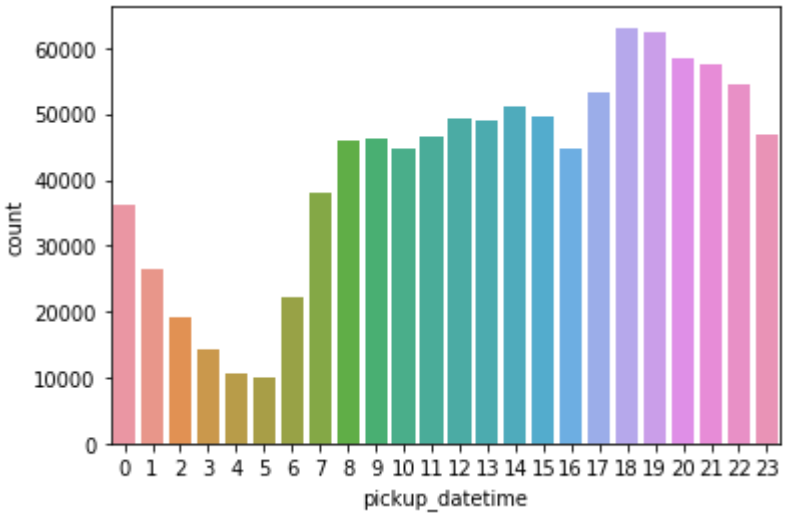
Мы можем увидеть паттерны, которые повторяются каждую неделю. Также мы можем наблюдать несколько аномальных правалов в количестве поездок. Посмотрим, как выглядит распределение по часам.

In [ ]:

```
sns.countplot(
  df_train.pickup_datetime.apply(lambda x: x.hour),
)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa68d5c5b00>
```



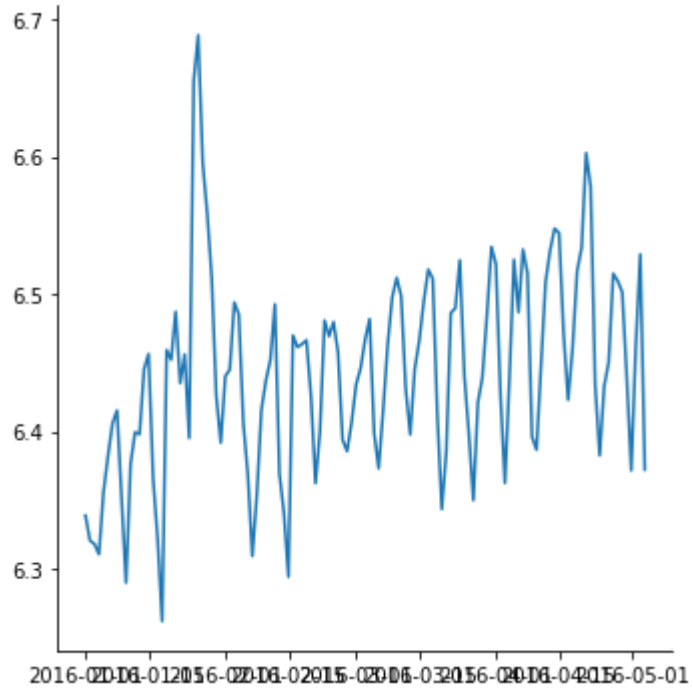
Теперь давайте посмотрим, как связан день и длина поездки.

In [ ]:

```
group_by_weekday = df_train.groupby(df_train.pickup_datetime.apply(lambda x: x.date()))
sns.relplot(data=group_by_weekday.log_trip_duration.aggregate('mean'), kind='line');
```

```
/usr/local/lib/python3.6/dist-packages/pandas/plotting/_matplotlib/convert
er.py:103: FutureWarning: Using an implicitly registered datetime converte
r for a matplotlib plotting method. The converter was registered by pandas
on import. Future versions of pandas will require you to explicitly regist
er matplotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



Мы видим явный тренд. Более того, наблюдается такая вещь как сезонность: повторяющиеся временные паттерны. В нашем случае период равен неделе.

In [ ]:

Теперь подготовим датасет. Включим в него день года и час дня. Для этого напишем функцию `create_features`, которая будет собирать нам нужные признаки в отдельный `pandas.DataFrame`. В итоге, мы сможем воспользоваться этой функцией, как для train подвыборки, так и для test.

In [ ]:

```
import datetime
def create_features(data_frame):
  X = pd.concat([
      data_frame.pickup_datetime.apply(lambda x: x.timetuple().tm_yday),
      data_frame.pickup_datetime.apply(lambda x: x.hour),
     ], axis=1, keys=['day', 'hour',]
  )

  return X, data_frame.log_trip_duration
```

In [ ]:

```
X_train, y_train = create_features(df_train)
X_test, y_test = create_features(df_test)
```

In [ ]:

```
X_train.head()
```

Out[ ]:

	day	hour
96469	1	0
223872	1	0
713067	1	0
652463	1	0
722901	1	0

Переменная час, хоть и является целым числом, не может трактоваться как вещественная. Дело в том, что после 23 идет 0, и что будет означать коэффициент регрессии в таком случае, совсем не ясно. Поэтому применим к этой переменной one -hot кодирование. В тоже время, переменная день должна остаться вещественной, так как значения из обучающей выборке не встреться нам на тестовом подмножестве.

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

In [ ]:

```
ohe = ColumnTransformer([("One hot", OneHotEncoder(sparse=False),[1])], remainder="pass through")
```

In [ ]:

```
X_train = ohe.fit_transform(X_train)
X_test = ohe.transform(X_test)
```

Воспользуемся классом `Ridge` и обучим модель.

In [ ]:

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
```

In [ ]:

```
ridge = Ridge(alpha=1000).fit(X_train, y_train)
```

In [ ]:

```
mean_squared_error(ridge.predict(X_test), y_test)
```

Out[ ]:

```
0.6539687516878999
```

Давайте попробуем сделать лучше и подберем гиперпараметры модели.

In [ ]:

```
from sklearn.model_selection import GridSearchCV

grid_searcher = GridSearchCV(Ridge(),
                             param_grid={'alpha': np.linspace(100, 750, 10)},
                             cv=5).fit(X_train, y_train)
```

In [ ]:

```
mean_squared_error(grid_searcher.predict(X_test), y_test)
```

Out[ ]:

```
0.6538703424304909
```

In [ ]:

```
grid_searcher.best_params_
```

Out[ ]:

```
{'alpha': 383.839}
```

In [ ]:

```

```

Задание 1

Постройте график соответствующий количеству поездок в зависимости от дня недели по обучающей выборке. Какой из этих графиков соответствует правильному?

In [ ]:

Задание 2

Добавьте к признакам бинарную переменную, которая равна 1 для двух аномальных дней и 0 во все остальные дни. Для этого вам понадобиться модифицировать функцию create_features.

In [ ]:

Задание 3

1. Добавьте день недели в качестве признака для обучения. Удобнее всего - модифицировать функцию create_features.
2. Заново проведите one-hot кодирование. Сколько признаков у вас получилось?

In [ ]:

Вопрос 4

1. Отмасштабируйте единственный вещественный признак.
2. Обучите на полученных данных Lasso регрессию, в качества параметра `alpha` возьмите 2.65e-05.

Какое качество в терминах MSE вы получили?

Сколько признаков было отобрано? В качестве критерия зануления признака используйте сравнение с 10^{-6} (с точностью до 3 знаков после запятой).

In [ ]: