In [31]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns


from catboost import CatBoostRegressor, Pool, cv
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split,RandomizedSearchCV, Shuff
from sklearn.preprocessing import LabelEncoder



sns.set(font_scale=1.5)
%matplotlib inline
```

executed in 42ms, finished 13:07:52 2020-05-22

Формулировка на простом языке:

> Необходимо сделать предсказание количества совершаемых поездок в зависимости от погодных условий.

Формулировка на математическом языке:

> Ставится задача регрессии для количества совершаемых поездок по данным погодных условий.

In [4]:

```
def root_mean_squared_error(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

executed in 218ms, finished 12:44:50 2020-05-22

In [5]:

```
DATA_DIR = "../data/processed/"
```

executed in 135ms, finished 12:44:50 2020-05-22

In [6]:

```
trips = pd.read_csv(DATA_DIR+"trips.csv",
                    error_bad_lines=False,
                    index_col=0)
weather = pd.read_csv(DATA_DIR+"weather.csv")
```

executed in 5.79s, finished 12:44:56 2020-05-22

In [7]:

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

executed in 41ms, finished 12:44:56 2020-05-22

Out[7]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_statior
0	431	10/13/2014 10:31	10/13/2014 10:48	SEA00298	985.935	2nd Ave & Spring St	Occide Occident S
1	432	10/13/2014 10:32	10/13/2014 10:48	SEA00195	926.375	2nd Ave & Spring St	Occide Occident S
2	433	10/13/2014 10:33	10/13/2014 10:48	SEA00486	883.831	2nd Ave & Spring St	Occide Occident S
3	434	10/13/2014 10:34	10/13/2014 10:48	SEA00333	865.937	2nd Ave & Spring St	Occide Occident S
4	435	10/13/2014 10:34	10/13/2014 10:49	SEA00202	923.923	2nd Ave & Spring St	Occide Occident S

In [8]:

```
1 weather.head(3)
```

executed in 117ms, finished 12:44:56 2020-05-22

Out[8]:

	Date	Max_Temperature_F	Mean_Temperature_F	Min_TemperatureF	Max_Dew_Pc
0	10/13/2014	71	62.0	54	
1	10/14/2014	63	59.0	55	
2	10/15/2014	62	58.0	54	

3 rows × 21 columns

Подготовим данные и воспользуемся библиотекой град. бустинга CatBoost и случайным лесом из sklearn.

In [9]:

```
1 trips['starttime'] = pd.to_datetime(trips['starttime'])
2 trips['stoptime'] = pd.to_datetime(trips['stoptime'])
3 trips['Date'] = pd.to_datetime(trips['starttime'].dt.date)
4 weather['Date']= pd.to_datetime(weather['Date'])
```

executed in 43.7s, finished 12:45:40 2020-05-22

In [10]:

```
num_trips_per_day = trips.groupby(
    'Date'
).size().reset_index().rename(columns={0:'trips_counter'})
```

executed in 29ms, finished 12:45:40 2020-05-22

In [11]:

```
data = num_trips_per_day.merge(weather, on='Date')
```

executed in 1.41s, finished 12:45:41 2020-05-22

In [12]:

```
data.Events = data.Events.fillna('Nothing')
data = data.drop('Date', axis=1)
data = data.drop('Max_Gust_Speed_MPH', axis=1) # много пропущенных значений
for col in data.columns[1:]:
    data[col] = data[col].fillna(0)
```

executed in 243ms, finished 12:45:42 2020-05-22

In [13]:

```
data.head(3)
```

executed in 937ms, finished 12:45:43 2020-05-22

Out[13]:

	trips_counter	Max_Temperature_F	Mean_Temperature_F	Min_TemperatureF	Max_D
0	409	71	62.0	54	
1	491	63	59.0	55	
2	313	62	58.0	54	

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, 1:],
                                                    data.iloc[:, 0],
                                                    test_size=0.2,
                                                    random_state=44)
```

executed in 2.01s, finished 12:45:45 2020-05-22

Воспользуемся CatBoost и применим поиск по сетки для поиска оптимальных гиперпараметров.

In [15]:

```
train_pool = Pool(X_train, y_train, cat_features=['Events'])
eval_pool = Pool(X_test, y_test, cat_features=['Events'])
```

executed in 2.29s, finished 12:45:48 2020-05-22

In [16]:

```
model = CatBoostRegressor(loss_function='RMSE',
                          learning_rate = 0.05,
                          random_seed=44)
```

executed in 89ms, finished 12:45:48 2020-05-22

In [17]:

```
params_gs_catboost = {
    'n_estimators' : [50, 100, 150, 300, 500,
                      1000, 2000, 2500, 3000]
}
```

executed in 265ms, finished 12:45:49 2020-05-22

In [18]:

```
model.grid_search(params_gs_catboost, train_pool)
```

executed in 9m 3s, finished 12:54:52 2020-05-22

```
0:      loss: 93.9396119        best: 93.9396119 (0)    total: 2.58s
remaining: 20.7s
1:      loss: 86.4393408        best: 86.4393408 (1)    total: 2.81s
remaining: 9.82s
2:      loss: 86.0597480        best: 86.0597480 (2)    total: 3s
remaining: 6s
3:      loss: 85.7427382        best: 85.7427382 (3)    total: 3.45s
remaining: 4.32s
4:      loss: 85.7427382        best: 85.7427382 (3)    total: 4.21s
remaining: 3.37s
5:      loss: 85.7427382        best: 85.7427382 (3)    total: 5.92s
remaining: 2.96s
6:      loss: 85.7427382        best: 85.7427382 (3)    total: 9.39s
remaining: 2.68s
7:      loss: 85.7427382        best: 85.7427382 (3)    total: 15.6s
remaining: 1.96s
8:      loss: 85.7427382        best: 85.7427382 (3)    total: 21.3s
remaining: 0us
Estimating final quality...
```

In [19]:

```
model = CatBoostRegressor(loss_function='RMSE',
                          learning_rate = 0.1,
                          n_estimators=300,
                          random_seed=44)
```

executed in 11ms, finished 12:54:52 2020-05-22

In [20]:

```
params_gs_catboost = {
    'depth': [4, 6, 10],
    'l2_leaf_reg': [1, 3, 5, 7, 9]
}
```

executed in 4ms, finished 12:54:53 2020-05-22

In [21]:

```
1 model.grid_search(params_gs_catboost, train_pool)
```

executed in 6m 53s, finished 13:01:46 2020-05-22

```
0:      loss: 86.9262535        best: 86.9262535 (0)    total: 877ms
remaining: 12.3s
1:      loss: 86.3628331        best: 86.3628331 (1)    total: 1.66s
remaining: 10.8s
2:      loss: 87.5914956        best: 86.3628331 (1)    total: 2.5s
remaining: 9.98s
3:      loss: 87.8407200        best: 86.3628331 (1)    total: 3.21s
remaining: 8.82s
4:      loss: 87.2291908        best: 86.3628331 (1)    total: 3.73s
remaining: 7.47s
5:      loss: 85.8422771        best: 85.8422771 (5)    total: 4.51s
remaining: 6.76s
6:      loss: 85.3102779        best: 85.3102779 (6)    total: 5.1s
remaining: 5.83s
7:      loss: 86.8046319        best: 85.3102779 (6)    total: 5.55s
remaining: 4.86s
8:      loss: 85.0939817        best: 85.0939817 (8)    total: 6.07s
remaining: 4.04s
9:      loss: 85.1275243        best: 85.0939817 (8)    total: 6.58s
```

In [22]:

```
1 model = CatBoostRegressor(loss_function='RMSE',
2                           learning_rate = 0.05,
3                           n_estimators=300,
4                           depth=10,
5                           l2_leaf_reg=9,
6                           random_seed=44)
```

executed in 5ms, finished 13:01:46 2020-05-22

In [24]:

```
model.fit(train_pool,
          eval_set=eval_pool,
          plot=True, verbose=1000)
```

executed in 4.71s, finished 13:04:56 2020-05-22

```
0:      learn: 151.8570666      test: 146.6438892       best: 146.6438
892 (0) total: 9.47ms   remaining: 2.83s
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
299:    learn: 43.4372702       test: 72.3095433        best: 72.19036
25 (290)        total: 4.01s    remaining: 0us

bestTest = 72.19036249
bestIteration = 290

Shrink model to first 291 iterations.
```

Out[24]:

```
<catboost.core.CatBoostRegressor at 0x7fe024686310>
```

In [25]:

```
root_mean_squared_error(y_test, model.predict(X_test))
```

executed in 147ms, finished 13:04:58 2020-05-22

Out[25]:

```
72.19036249229482
```

In [37]:

```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

executed in 4ms, finished 13:30:23 2020-05-22

Посмотрим на MAPE.

In [38]:

```
mean_absolute_percentage_error(y_test, model.predict(X_test))
```

executed in 164ms, finished 13:30:49 2020-05-22

Out[38]:

```
28.444699777699572
```

Попробуем применить случайный лес.

In [26]:

```
label_encoder = LabelEncoder()
label_encoder.fit(X_train.iloc[:, -1])
```

executed in 89ms, finished 13:05:06 2020-05-22

Out[26]:

```
LabelEncoder()
```

In [27]:

```
X_train.iloc[:, -1] = label_encoder.transform(X_train.iloc[:, -1])
X_test.iloc[:, -1] = label_encoder.transform(X_test.iloc[:, -1])
```

executed in 11ms, finished 13:05:06 2020-05-22

In [29]:

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}
```

executed in 21ms, finished 13:06:36 2020-05-22

In [36]:

```
rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = random_grid,
                               cv = 3,
                               verbose=2,
                               random_state=44,
                               n_iter = 100,
                               n_jobs = -1)
rf_random.fit(X_train, y_train)
```

executed in 8m 59s, finished 13:19:56 2020-05-22

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  8.9min finished
```

Out[36]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='aut
o',
                                                   max_leaf_nodes=No
ne,
                                                   max_samples=None,
                                                   min_impurity_decr
ease=0.0,
                                                   min_impurity_spli
t=None,
                                                   min_samples_leaf=
1,
                                                   min_samples_split
=2,
                                                   min_weight_fracti
on_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_
score=Fals...
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 4
0, 50, 60,
                                                      70, 80, 90, 10
0, 110,
                                                      None],
                                        'max_features': ['auto', 'sq
rt'],
                                        'min_samples_leaf': [1, 2,
4],
                                        'min_samples_split': [2, 5,
10],
                                        'n_estimators': [200, 400, 6
00, 800,
                                                         1000, 1200,
```

```
1400, 1600,
                                                       1800, 200
0]},
                   pre_dispatch='2*n_jobs', random_state=44, refit=T
rue,
                   return_train_score=False, scoring=None, verbose=
2)
```

In [33]:

```
1 rf_random.best_params_
```

executed in 36ms, finished 13:10:19 2020-05-22

Out[33]:

```
{'n_estimators': 400,
 'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': False}
```

In [34]:

```
1 def root_mean_squared_error(y_true, y_pred):
2     return np.sqrt(mean_squared_error(y_true, y_pred))
```

executed in 16ms, finished 13:10:24 2020-05-22

In [35]:

```
1 root_mean_squared_error(y_test, rf_random.best_estimator_.predict(X_test))
```

executed in 141ms, finished 13:10:36 2020-05-22

Out[35]:

```
74.07526021739842
```

In [39]:

```
1 mean_absolute_percentage_error(y_test, rf_random.best_estimator_.predict(X_tes
```

executed in 108ms, finished 13:37:50 2020-05-22

Out[39]:

```
24.895983803381515
```

Как видим, CatBoost и случайный лес с подбором параметров показывают приблизительно схожие результаты со значением RMSE около 73-74(MAPE при этом 28.4% и 24.8% соответственно), что вообще говоря не очень хорошо, ведь в день совершается всего несколько сотен поездок, поэтому ошибка достаточно велика. Безусловно это связано с тем, что в модели учитываются лишь погодные условия (без дня недели, времени года и т.д.)