In [2]:

```
import numpy as np
import pandas as pd


from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

executed in 1.19s, finished 13:50:29 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 в зависимости от погодных условй.

In [3]:

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

executed in 6ms, finished 13:50:29 2020-05-22

In [4]:

```
def get_data(DATA_DIR):↔
```

executed in 149ms, finished 13:50:30 2020-05-22

Выберем RMSE в качестве метрики(имеет ту же размерность, что и исходные данные, в отличие от MSE).

In [5]:

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

executed in 95ms, finished 13:50:30 2020-05-22

In [6]:

```
data = get_data(DATA_DIR)
data.head(5)
```

executed in 51.0s, finished 13:51:21 2020-05-22

Out[6]:

	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	
3	395	71	61.0	52	
4	294	64	60.0	57	

In [7]:

```
data = pd.concat((data.iloc[:, :-1], pd.get_dummies(data.iloc[:, -1])),axis=1)
```

executed in 14ms, finished 13:51:21 2020-05-22

Попробуем применить линейную регресиию, но не простую, а Elastic Net.

In [8]:

```
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 80ms, finished 13:51:21 2020-05-22

Посмотрим на размер тренировочной/тестовой выборки.

In [9]:

```
X_train.shape, X_test.shape
```

executed in 5ms, finished 13:51:21 2020-05-22

Out[9]:

```
((551, 28), (138, 28))
```

In [10]:

```
model = ElasticNet(max_iter=5000)
steps = [('scaler', StandardScaler()), ('model', model)]
pipeline = Pipeline(steps)
```

executed in 5ms, finished 13:51:21 2020-05-22

In [11]:

```
parameters_grid_elastic = {
    'model__alpha' : np.linspace(0.00001, 2, num=100),
    'model__l1_ratio' : np.linspace(0, 1, num=50)
}
```

```
executed in 6ms, finished 13:51:22 2020-05-22
```

In [12]:

```
ss = ShuffleSplit(n_splits=5, test_size=0.25, random_state=44)
```

```
executed in 3ms, finished 13:51:22 2020-05-22
```

In [13]:

```
gs = GridSearchCV(estimator=pipeline,
                  param_grid=parameters_grid_elastic,
                  scoring='neg_mean_squared_error',
                  cv=ss,
                  n_jobs=-1,
                  verbose=2)
```

```
executed in 5ms, finished 13:51:22 2020-05-22
```

In [14]:

```
gs.fit(X_train, y_train)
```

executed in 3m 1s, finished 13:54:23 2020-05-22

```
Fitting 5 folds for each of 5000 candidates, totalling 25000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:    9.3s
[Parallel(n_jobs=-1)]: Done 872 tasks      | elapsed:   18.4s
[Parallel(n_jobs=-1)]: Done 3136 tasks      | elapsed:   33.4s
[Parallel(n_jobs=-1)]: Done 6056 tasks      | elapsed:   54.8s
[Parallel(n_jobs=-1)]: Done 9616 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 13832 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 18688 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 24200 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 25000 out of 25000 | elapsed:  3.0min fini
shed
```

Out[14]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=5, random_state=44, test_size=0.
25, train_size=None),
             error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=Tru
e)),
                                       ('model',
                                        ElasticNet(alpha=1.0, copy_X=T
rue,
                                                   fit_intercept=True,
                                                   l1_ratio=0.5, max_i
ter=5000,
                                                   normalize=False,
                                                   positive=False,
                                                   precompute=False,
                                                   random...
       0.51020408, 0.53061224, 0.55102041, 0.57142857, 0.59183673,
       0.6122449 , 0.63265306, 0.65306122, 0.67346939, 0.69387755,
       0.71428571, 0.73469388, 0.75510204, 0.7755102 , 0.79591837,
       0.81632653, 0.83673469, 0.85714286, 0.87755102, 0.89795918,
       0.91836735, 0.93877551, 0.95918367, 0.97959184, 1.        ])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='neg_mean_squared_error', verbose=2)
```

In [15]:

```
root_mean_squared_error(y_test, gs.best_estimator_.predict(X_test))
```

executed in 8ms, finished 13:54:23 2020-05-22

Out[15]:

```
75.65636238907558
```

In [1]:

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

executed in 14ms, finished 13:50:20 2020-05-22

In [16]:

```
mean_absolute_percentage_error(y_test, gs.best_estimator_.predict(X_test))
```

executed in 11ms, finished 13:54:24 2020-05-22

Out[16]:

```
23.550110148977645
```

Получили результат близкий к тем, что мы получали у CatBoost/RandomForest, при этом MAPE оказывается меньше.