In [1]:

```
import numpy as np
import pandas as pd
import warnings
from tqdm import tqdm_notebook

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

sns.set(font_scale=1.4, style="darkgrid", palette='Set2')
warnings.filterwarnings("ignore")
```

started 09:29:41 2020-03-27, finished in 1.21s

Задача 1

Будем работать с датасетом **"bikes_rent.csv"**, в котором по дням записаны календарная информация и погодные условия, характеризующие автоматизированные пункты проката велосипедов, а также число прокатов в этот день. Последнее мы будем предсказывать; таким образом, мы будем решать задачу регрессии.

Данные предоставлены компанией capital bikeshare.

Для каждого дня проката известны следующие признаки (как они были указаны в источнике данных):

- `_season_` : 1 - весна, 2 - лето, 3 - осень, 4 - зима
- `_yr_` : 0 - 2011, 1 - 2012
- `_mnth_` : от 1 до 12
- `_holiday_` : 0 - нет праздника, 1 - есть праздник
- `_weekday_` : от 0 до 6
- `_workingday_` : 0 - нерабочий день, 1 - рабочий день
- `_weathersit_` : оценка благоприятности погоды от 1 (чистый, ясный день) до 4 (ливень, туман)
- `_temp_` : температура в Цельсиях
- `_atemp_` : температура по ощущениям в Цельсиях
- `_hum_` : влажность
- `_windspeed(mph)_` : скорость ветра в милях в час
- `_windspeed(ms)_` : скорость ветра в метрах в секунду
- `_cnt_` : количество арендованных велосипедов (это целевой признак, его мы будем предсказывать)

Считайте данные и разделите на обучение и тест.

In [2]:

```
data = pd.read_csv("bikes_rent.csv")
X_train, X_test, y_train, y_test = train_test_split(
    data.iloc[:,0:-1], data.iloc[:,-1], random_state = 42
)
data.head()
```

started 09:29:42 2020-03-27, finished in 64ms

Out[2]:

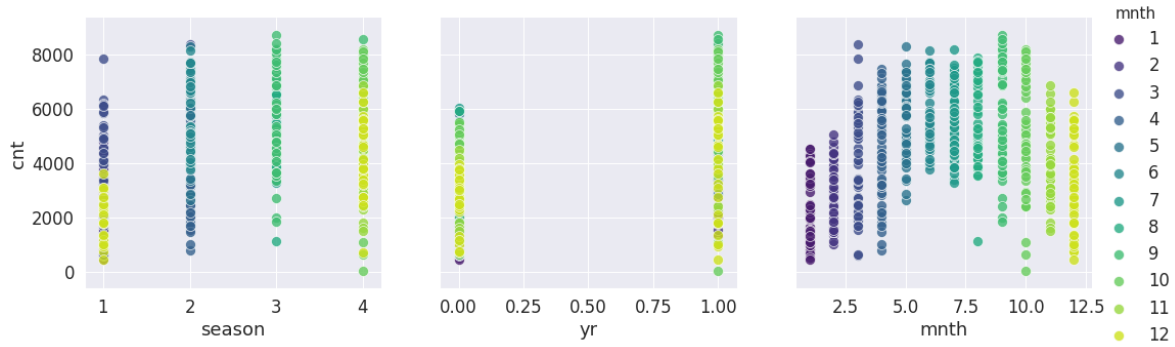
	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum
0	1	0	1	0	6	0	2	14.110847	18.18125	80.5833
1	1	0	1	0	0	0	2	14.902598	17.68695	69.6087
2	1	0	1	0	1	1	1	8.050924	9.47025	43.7273
3	1	0	1	0	2	1	1	8.200000	10.60610	59.0435
4	1	0	1	0	3	1	1	9.305237	11.46350	43.6957

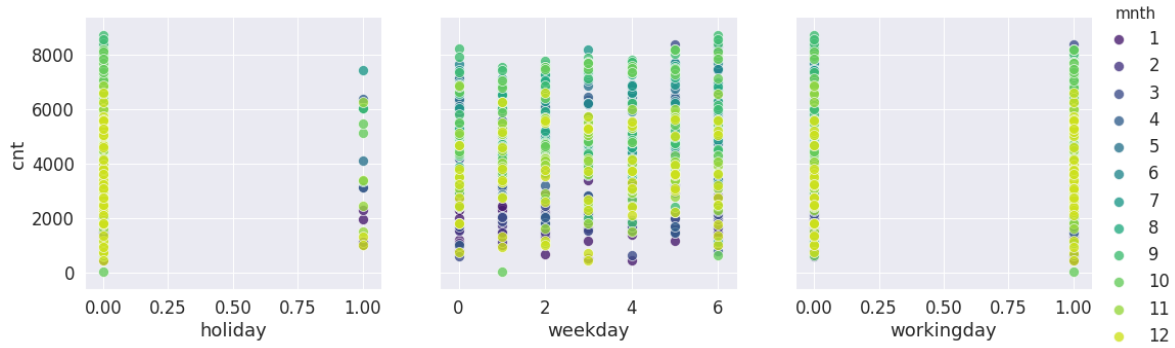
Посмотрите на графиках, как целевой признак зависит от остальных и поймите какой характер зависимости целевой переменной от остальных.

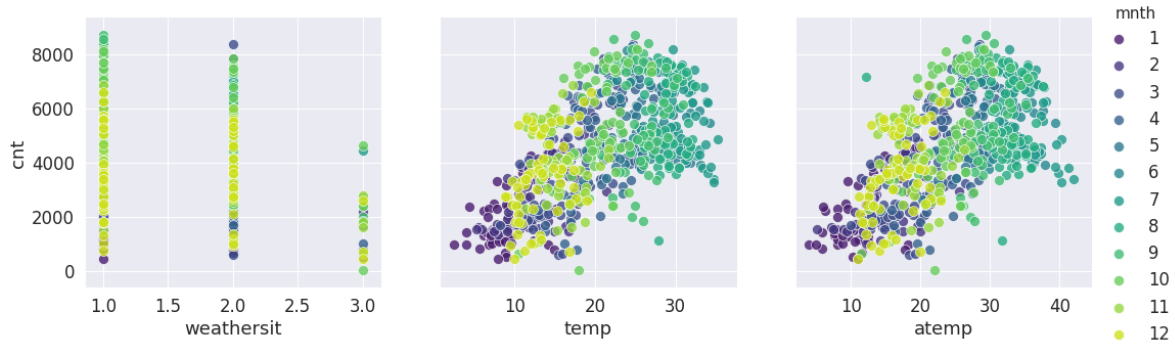
In [3]:

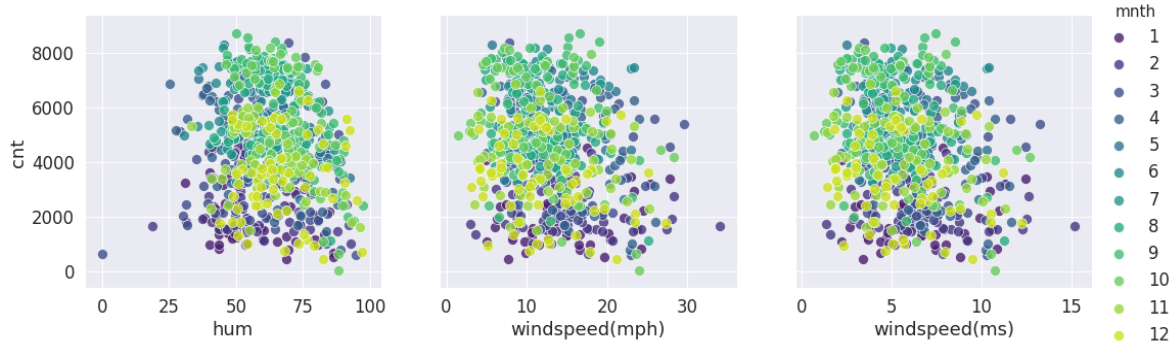
```
feature_names = np.array(data.columns[:-1])
sns.set(font_scale=1.5)
for i in range(0, 12, 3):
    sns.pairplot(
        data,
        y_vars=["cnt"],
        x_vars=feature_names[i:i+3],
        height=5, hue="mnth", palette="viridis",
        plot_kws = {"s": 100, "alpha": 0.8}
    );
```

started 09:29:42 2020-03-27, finished in 8.33s









Вывод:

- есть признаки, которые сильно зависят друг от друга: `windspeed(mph)` и `windspeed(ms)`; `temp` и `atemp`; `workingday` от `weekday`. В последствии нужно будет убрать один признак из каждой пары.
- `cnt` линейно зависит от `temp` и `atemp`
- линейно убывает в зависимости от `weathersit` и `windspeed`.
- если посмотреть на зависимость от месяца, то видно: летом арендуют чаще

Теперь посмотрите на среднее значение каждого признака. Что можно сказать? Какая тут проблема?

In [4]:

```
1  data.mean()
```

started 09:29:51 2020-03-27, finished in 6ms

Out[4]:

```
season              2.496580
yr                  0.500684
mnth                6.519836
holiday             0.028728
weekday             2.997264
workingday          0.683995
weathersit          1.395349
temp               20.310776
atemp              23.717699
hum                62.789406
windspeed(mph)     12.762576
windspeed(ms)       5.705220
cnt              4504.348837
dtype: float64
```

Проблема: разная размерность в данных. Из-за того, что у некоторых данных большое среднее значение, они могут неоправданно сильно влиять на результат во многих моделях. Тогда как данные с маленьким среднем практически не будут учитываться. Нужно нормализовать данные.

Поняв проблему, исправьте ее.

Чтобы не менять категориальные данные воспользуемся `MinMaxScaler`. При нормировке нужно сделать так, чтобы обучающая выборка не узнала ничего о тестовой выборке. Для этого нормируем обучающую выборку отдельно от тестовой, а потом тестовую с теми же параметрами.

In [5]:

```
1  scaler = MinMaxScaler()
2  X_train = scaler.fit_transform(X_train)
3  X_test = scaler.transform(X_test)
```

started 09:29:51 2020-03-27, finished in 7ms

Проверим результат:

In [6]:

```
1  X_train.mean(axis=0)
```

started 09:29:51 2020-03-27, finished in 9ms

Out[6]:

```
array([0.50243309, 0.52189781, 0.50033179, 0.02554745, 0.49270073,
       0.65693431, 0.19890511, 0.55116239, 0.52691277, 0.6571497 ,
       0.39865652, 0.39865652])
```

Обучите линейную регрессию на наших данных и посмотрите на веса признаков. Что в них не так? Почему так получилось? Какая здесь проблема и как ее можно решить?

In [7]:

```
1  model = LinearRegression()
2  model.fit(X_train, y_train)
3  model.coef_
```

started 09:29:51 2020-03-27, finished in 16ms

Out[7]:

```
array([ 1.58795380e+03,  2.02231842e+03, -4.37924143e+02, -3.97728570e
+02,
        4.55464905e+02,  1.51482021e+02, -1.21140749e+03, -2.10792873e
+03,
        6.76747508e+03, -9.21983059e+02,  7.59522255e+13, -7.59522255e
+13])
```

У некоторых признаков слишком большие веса.

В датасете есть линейно зависимые признаки: `windspeed(mph)` и `windspeed(ms)`; `temp` и `atemp`; `workingday` и `weekday`. Из-за этого матрица признаков получается вырожденной и формула МНК-решения перестает быть корректной, поскольку в ней берется обратная матрица от вырожденной.

Из `workingday` и `weekday` уберем `workingday`, т.к. он несет меньше информации.

Решите проблему, обучите линейную модель и снова посмотрите на веса? Стало ли лучше?

In [8]:

```
def drop(data):
    new_data = data.copy()
    new_data = new_data.drop('windspeed(mph)',axis = 1)
    new_data = new_data.drop('temp',axis = 1)
    new_data = new_data.drop('workingday',axis = 1)
    return new_data

new_data = drop(data)
new_data.head()
```

started 09:29:51 2020-03-27, finished in 25ms

Out[8]:

	season	yr	mnth	holiday	weekday	weathersit	atemp	hum	windspeed(ms)	cnt
0	1	0	1	0	6	2	18.18125	80.5833	4.805490	985
1	1	0	1	0	0	2	17.68695	69.6087	7.443949	801
2	1	0	1	0	1	1	9.47025	43.7273	7.437060	1349
3	1	0	1	0	2	1	10.60610	59.0435	4.800998	1562
4	1	0	1	0	3	1	11.46350	43.6957	5.597810	1600

Также воспользуемся one-hot encoding для категориальных данных. При этом нужно выкинуть один из получившихся столбцов для каждого one-hot encoding, иначе они будут линейно зависимы.

In [9]:

```
def one_hot(new_data, prefixes=['mnth', 'weekday', 'weathersit']):
    for prefix in prefixes:
        one_hot = pd.get_dummies(new_data[prefix], prefix=prefix,
                                 drop_first=True)
        new_data = new_data.drop(prefix, axis=1)
        new_data = new_data.join(one_hot)
    return new_data

new_data = one_hot(new_data)

new_data.head()
```

started 09:29:51 2020-03-27, finished in 43ms

Out[9]:

	season	yr	holiday	atemp	hum	windspeed(ms)	cnt	mnth_2	mnth_3	mnth_4	...
0	1	0	0	18.18125	80.5833	4.805490	985	0	0	0	...
1	1	0	0	17.68695	69.6087	7.443949	801	0	0	0	...
2	1	0	0	9.47025	43.7273	7.437060	1349	0	0	0	...
3	1	0	0	10.60610	59.0435	4.800998	1562	0	0	0	...
4	1	0	0	11.46350	43.6957	5.597810	1600	0	0	0	...

5 rows × 26 columns

Посмотрим на имена полученных признаков

In [10]:

```
1  new_data.columns
```

started 09:29:51 2020-03-27, finished in 4ms

Out[10]:

```
Index(['season', 'yr', 'holiday', 'atemp', 'hum', 'windspeed(ms)', 'cn
t',
       'mnth_2', 'mnth_3', 'mnth_4', 'mnth_5', 'mnth_6', 'mnth_7', 'mn
th_8',
       'mnth_9', 'mnth_10', 'mnth_11', 'mnth_12', 'weekday_1', 'weekda
y_2',
       'weekday_3', 'weekday_4', 'weekday_5', 'weekday_6', 'weathersit
_2',
       'weathersit_3'],
      dtype='object')
```

Делим на обучение и тест, обучаем линейную регресси.

In [11]:

```
1  X_train, X_test, y_train, y_test = train_test_split(
2      new_data.loc[:, new_data.columns != 'cnt'],
3      new_data.loc[:,'cnt'], random_state=42
4  )
5
6  model = LinearRegression()
7  model.fit(X_train, y_train)
8  model.coef_
```

started 09:29:51 2020-03-27, finished in 10ms

Out[11]:

```
array([  557.51742744,  1995.48006212,  -575.28180307,   104.9708202 ,
         -13.97171969,   -77.25045188,    82.58702435,   740.62343999,
         702.22597715,   993.29286018,   632.00787568,  -447.98768535,
         -66.97586284,   729.79199746,   401.59760782,  -245.30786764,
        -194.83679472,   300.49372868,   303.93765865,   403.23725621,
         405.13747834,   496.59633734,   491.06285265,  -533.32492063,
       -1734.43797524])
```

Стало лучше, веса модели не сильно отличаются друг от друга

Обучите теперь `Lasso` и подберите оптимальный параметр `alpha` для него. Метрика --- MSE. Возьмите `alpha` от 0 до 100. Разделите выборку на 3 части и проведите 3 итерации для разных частей: для каждого `alpha` обучите `Lasso(alpha)` на двух частях и посмотрите на MSE на третей части. Визуализируйте 3 полученных графика завасимости MSE от alpha на разных данных. Сделайте выводы.

Будем пользоваться pipeline-ами при нормировании, чтобы информация из тестовой выборки не попала в обучение. Вернем исходные данные без нормировки, но с one_hot.

In [12]:

```
data = pd.read_csv("bikes_rent.csv")
new_data = drop(data)
new_data = one_hot(new_data)

X_train, X_test, y_train, y_test = train_test_split(
    new_data.loc[:, new_data.columns != 'cnt'],
    new_data.loc[:,'cnt'],  test_size=0.2, random_state=42
)
```

started 09:29:51 2020-03-27, finished in 27ms

Выполним кросс-валидациию для поиска оптимального значения гиперпараметра

In [13]:

```
steps = [('scaler', MinMaxScaler()), ('lasso_model', Lasso())]
pipeline = Pipeline(steps)

C = np.linspace(0.01, 100, 1000)
params = {'lasso_model__alpha': C}

gscv = GridSearchCV(pipeline, params, cv=3,
                    scoring='neg_mean_squared_error', n_jobs=4)
gscv.fit(X_train, y_train)
```

started 09:29:51 2020-03-27, finished in 29.2s

Out[13]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        MinMaxScaler(copy=True,
                                                     feature_range=(0,
1))),
                                       ('lasso_model',
                                        Lasso(alpha=1.0, copy_X=True,
                                              fit_intercept=True, max_
iter=1000,
                                              normalize=False, positiv
e=False,
                                              precompute=False,
                                              random_state=None,
                                              selection='cyclic', tol=
0.0001,
                                              warm_start=False))],
                                verbose=False),...
       9.80982883e+01, 9.81983784e+01, 9.82984685e+01, 9.83985586e+01,
       9.84986486e+01, 9.85987387e+01, 9.86988288e+01, 9.87989189e+01,
       9.88990090e+01, 9.89990991e+01, 9.90991892e+01, 9.91992793e+01,
       9.92993694e+01, 9.93994595e+01, 9.94995495e+01, 9.95996396e+01,
       9.96997297e+01, 9.97998198e+01, 9.98999099e+01, 1.00000000e+0
2])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='neg_mean_squared_error', verbose=0)
```

Нарисуем график зависимости ошибки по фолдам

In [14]:

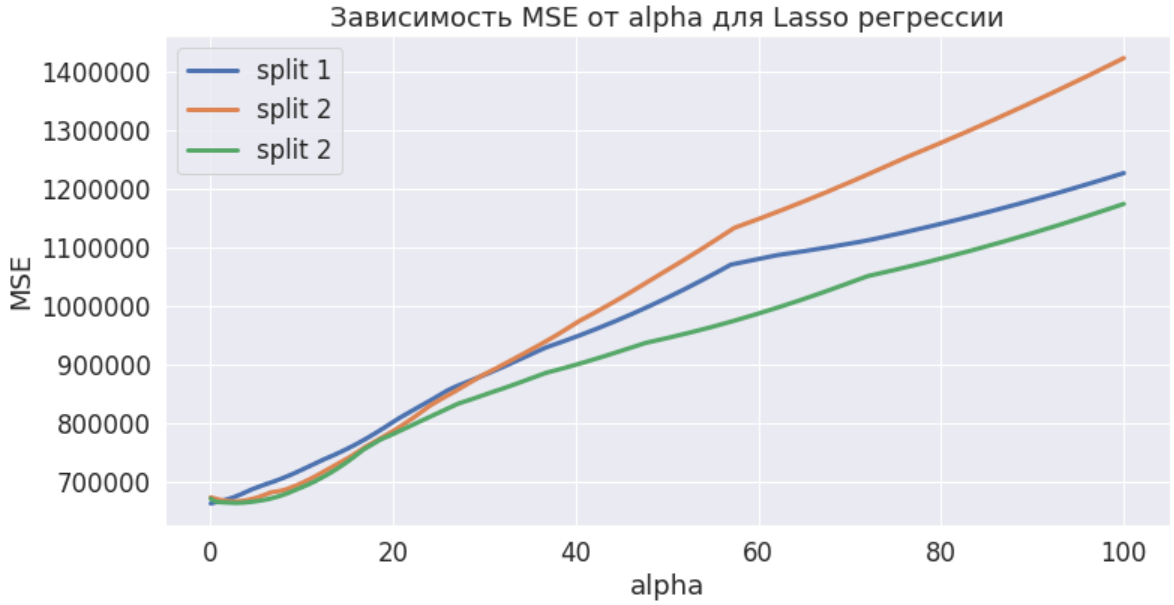
```
mean_test_score0 = gscv.cv_results_['split0_test_score']
mean_test_score1 = gscv.cv_results_['split1_test_score']
mean_test_score2 = gscv.cv_results_['split2_test_score']

plt.figure(figsize = (12, 6))

plt.plot(C, -mean_test_score0, label="split 1", lw=3)
plt.plot(C, -mean_test_score1, label="split 2", lw=3)
plt.plot(C, -mean_test_score2, label="split 2", lw=3)

plt.legend()
plt.title("Зависимость MSE от alpha для Lasso регрессии")
plt.xlabel("alpha")
plt.ylabel("MSE");
```

started 09:30:20 2020-03-27, finished in 428ms



Вывод: при построении графиков зависимости качества от alpha для разных фолдов можно заметить, что для разных фолдов оптимальные значения получились разными (да и сам mse сильно различается) отсюда вывод : лучше использовать кросс-валидацию, а не откладывать одну отдельную часть под валидацию.

Найдите оптимальное alpha

In [15]:

```
print(gscv.best_params_)
```

started 09:30:20 2020-03-27, finished in 6ms

```
{'lasso_model__alpha': 1.311171171171171}
```

Приступим к выбору модели:

Linear:

In [16]:

```
steps = [('scaler', MinMaxScaler()), ('linear_model', LinearRegression())]
pipeline = Pipeline(steps)

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

linear_mse = mean_squared_error(y_test, y_pred)
print("LinearRegression MSE: {:.1f}".format(linear_mse))
```

started 09:30:20 2020-03-27, finished in 20ms

```
LinearRegression MSE: 619381.6
```

Lasso:

In [17]:

```
steps = [('scaler', MinMaxScaler()), ('lasso_model', Lasso())]
pipeline = Pipeline(steps)
```

started 09:30:20 2020-03-27, finished in 3ms

In [18]:

```
params = {
    'lasso_model__alpha': np.linspace(0.01, 10, 1000)
}
gscv = GridSearchCV(pipeline, params, cv=3,
                    scoring='neg_mean_squared_error', n_jobs=4)

gscv.fit(X_train, y_train)
print(gscv.best_params_)
```

started 09:30:20 2020-03-27, finished in 20.1s

```
{'lasso_model__alpha': 1.3}
```

In [20]:

```
y_pred = gscv.best_estimator_.predict(X_test)

lasso_mse = mean_squared_error(y_test, y_pred)
print("Lasso MSE: {:.1f}".format(lasso_mse))
```

started 09:43:06 2020-03-27, finished in 14ms

```
Lasso MSE: 623310.8
```

Ridge:

In [21]:

```
steps = [('scaler', MinMaxScaler()), ('ridge_model', Ridge())]
pipeline = Pipeline(steps)
```

started 09:43:08 2020-03-27, finished in 5ms

In [22]:

```
params = {
    'ridge_model__alpha': np.linspace(0.01, 10, 1000)
}
gscv = GridSearchCV(pipeline, params, cv=3,
                    scoring='neg_mean_squared_error', n_jobs=4)

gscv.fit(X_train, y_train)
print(gscv.best_params_)
```

started 09:43:08 2020-03-27, finished in 20.9s

```
{'ridge_model__alpha': 0.53}
```

In [23]:

```
y_pred = gscv.best_estimator_.predict(X_test)

lasso_mse = mean_squared_error(y_test, y_pred)
print("Ridge MSE: {:.1f}".format(lasso_mse))
```

started 09:43:29 2020-03-27, finished in 12ms

```
Ridge MSE: 619580.4
```

Сравнение моделей:

In [24]:

```
print("LinearRegression MSE: {:.1f}".format(linear_mse))
print("Lasso MSE: {:.1f}".format(lasso_mse))
print("Ridge MSE: {:.1f}".format(ridge_mse))
```

started 09:43:29 2020-03-27, finished in 20ms

```
LinearRegression MSE: 619381.6
Lasso MSE: 619580.4
Ridge MSE: 623310.8
```

Вывод:

Линейная регрессия показала лучший результат