

# Линейная регрессия

МАКСИМОВСКАЯ  
АНАСТАСИЯ

# Задача регрессии

---

**Регрессия** — класс задач обучения с учителем, когда по определённому набору признаков объекта нужно предсказать целевую переменную.

**Задача регрессии** — нахождение зависимостей между определяющими переменными и определяемой переменной, если она является непрерывным числом. Например, определить стоимость дома по его площади.

**Целевое значение** — любое действительное число.

**Задача линейной регрессии** — нахождение такой зависимости, если она линейная.

# Пример

---

# Общий вид

---

$$a(x) = w_0 + w_1 \cdot x_1 + \dots + w_d \cdot x_d = w_0 + \sum_{i=1}^d w_i \cdot x_i$$

# Линейная регрессия

---

$$a(x) = w_0 + \sum_{j=1}^d w_j x_j$$

Параметры линейной регрессии – веса (коэффициенты  $w_j$ ). Вес  $w_0$  называется свободным коэффициентом или сдвигом (bias). Заметим, что после знака суммы написано скалярное произведение. Также добавим в выборку  $w_{d+1}$  признак, равный единице, тогда необходимость в свободном коэффициенте отпадет. Перепишем формулу в более компактном виде:

$$a(x) = \langle w, x \rangle$$

# Преимущества линейной регрессии

---

- За счёт простой формы линейные модели достаточно быстро и легко обучаются, и поэтому популярны при работе с большими объёмами данных.
- Также у них мало параметров, благодаря чему удается контролировать риск переобучения и использовать их для работы с зашумлёнными данными и с небольшими выборками.
- Подходят для поиска простых взаимосвязей в данных.

# Обучение линейной регрессии

---

Чаще всего линейная регрессия обучается с использованием среднеквадратичной ошибки. В этом случае получаем задачу оптимизации (считаем, что среди признаков есть константный, и поэтому свободный коэффициент не нужен):

$$\frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 \rightarrow \min_w$$

Перепишем в матричном виде:

$$\frac{1}{\ell} \|Xw - y\|^2 \rightarrow \min_w$$

## Области применимости

$w_0 \cdot [t_0 \leq x_a < t_1]$  ...

$$a(x) = w_0 + w_1 \cdot (\text{площадь}) + w_2 \cdot (\text{количество комнат}) + w_3 \cdot (\text{расстояние до метро}) + w_4 \cdot (\text{район})$$

$$w_4 \cdot [\rho = 14\%] + w_5 \cdot [\rho = 103\%] +$$

} 0 1

} 0 0

# One-Hot Encoding

---

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

# One-Hot Encoding

---

$$a(x) = w_0 + w_1 \cdot [x_1 = c_1] + w_2 \cdot [x_1 = c_2] + \dots + w_n \cdot [x_1 = c_n])$$

# Бинаризация числовых признаков

---

$$a(x) = w_0 + w_1 \cdot [t_0 \leq x_1 < t_1] + w_2 \cdot [t_1 \leq x_2 < t_2] + \dots + w_n \cdot [t_n \leq x_n < t_{n+1}]$$

# Вывод

---

Для линейных моделей признаки надо готовить так, чтобы модель была осмысленной!

# Метрики для задачи регрессии

---

Mean Squared Error (среднеквадратичное отклонение):

$$\text{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

# Метрики для задачи регрессии

---

Root Mean Squared Error (корень из среднеквадратичного отклонения):

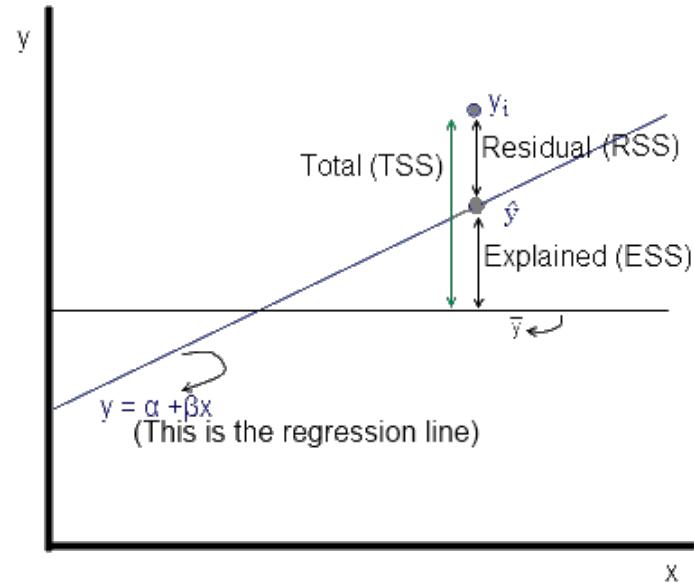
$$\text{RMSE}(a, X) = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2}$$

# Метрики для задачи регрессии

R-squared (коэффициент детерминации):

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2} = \\ = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

➤ Измеряет долю дисперсии, объясненную моделью.



Источник: [URL](#)

$\hat{y}$  is the predicted value of  $y$  given  $x$ , using the equation  $y = a + \beta x$ .

$y_i$  is the actual observed value of  $y$ .

$\bar{y}$  is the mean of  $y$ .

The distances that RSS, ESS and TSS represent are shown in the diagram to the left - but remember that the actual calculations are squares of these distances.

$$TSS = \sum (y_i - \bar{y})^2$$

$$RSS = \sum (y_i - \hat{y})^2$$

$$ESS = \sum (\hat{y} - \bar{y})^2$$

# Метрики для задачи регрессии

---

Mean Absolute Error (среднее абсолютное отклонение):

$$\text{MAE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|$$

Handwritten notes below the formula:

Left side:  $y$ ,  $a(x)$ ,  $\frac{y - a(x)}{2}$ ,  $|y - a(x)|$

Middle row:  $1$ ,  $1$

Bottom row:  $9960.04$ ,  $998$

$$\frac{\log x}{1000}$$

Left side:  $x$ ,  $y$ ,  $a(x)$ ,  $\frac{y - a(x)}{2}$ ,  $|y - a(x)|$

Middle row:  $2$ ,  $2$

Bottom row:  $9960.04$ ,  $998$

# Метрики для задачи регрессии

---

Mean Squared Logarithmic Error, MSLE (средняя логарифмическая ошибка):

$$L(y, a) = (\log(a + 1) - \log(y + 1))^2$$

Подходит для задач с неотрицательной целевой переменной. За счёт логарифмирования ответов и прогнозов мы скорее штрафуем за отклонения в порядке величин, чем за отклонения в их значениях. Также следует помнить, что логарифм не является симметричной функцией, и поэтому данная функция потерь штрафует заниженные прогнозы сильнее, чем завышенные.

# Метрики для задачи регрессии

---

Mean Absolute Percentage Error, MAPE (средняя абсолютная процентная ошибка):

$$L(y, a) = \left| \frac{y - a}{y} \right|$$

Часто используется для задач прогнозирования.

# Метрики для задачи регрессии

---

Symmetric Mean Absolute Percentage Error, SMAPE (симметричная средняя абсолютная процентная ошибка):

$$L(y, a) = \frac{|y - a|}{(|y| + |a|)/2}$$

Симметричная модификация MAPE.

# Метод наименьших квадратов (МНК)

---

$$\begin{aligned}\mathcal{L}(X, \vec{y}, \vec{w}) &= \frac{1}{2n} \sum_{i=1}^n \left( y_i - \vec{w}^T \vec{x}_i \right)^2 \\ &= \frac{1}{2n} \|\vec{y} - X\vec{w}\|_2^2 \\ &= \frac{1}{2n} (\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w})\end{aligned}$$

# Выход формулы весов

---

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \vec{w}} &= \frac{\partial}{\partial \vec{w}} \frac{1}{2n} \left( \vec{y}^T \vec{y} - 2\vec{y}^T X \vec{w} + \vec{w}^T X^T X \vec{w} \right) \\ &= \frac{1}{2n} (-2X^T \vec{y} + 2X^T X \vec{w})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \vec{w}} = 0 &\Leftrightarrow \frac{1}{2n} (-2X^T \vec{y} + 2X^T X \vec{w}) = 0 \\ &\Leftrightarrow -X^T \vec{y} + X^T X \vec{w} = 0 \\ &\Leftrightarrow X^T X \vec{w} = X^T \vec{y} \\ &\Leftrightarrow \vec{w} = (X^T X)^{-1} X^T \vec{y}\end{aligned}$$

# Матричные производные

---

$$\frac{\partial}{\partial x} x^T a = a$$

$$\frac{\partial}{\partial x} x^T A x = \left( A + A^T \right) x$$

$$\frac{\partial}{\partial A} x^T A y = x y^T$$

$$\frac{\partial}{\partial x} A^{-1} = -A^{-1} \frac{\partial A}{\partial x} A^{-1}$$

# Выход формулы весов

---

Учитывая все определения и условия описанные выше, мы можем утверждать, опираясь на теорему Гаусса-Маркова, что оценка МНК является лучшей оценкой параметров модели, среди всех *линейных и несмещенных* оценок, то есть обладающей наименьшей дисперсией.

# В чём проблема этой формулы?

---

- Обращение матрицы — сложная операция с кубической сложностью от количества признаков. Если в выборке тысячи признаков, то вычисления могут стать слишком трудоёмкими. Решить эту проблему можно путём использования численных методов оптимизации.
- Матрица  $X^T X$  может быть вырожденной или плохо обусловленной. В этом случае обращение либо невозможно, либо может привести к неустойчивым результатам. Проблема решается с помощью регуляризации, речь о которой пойдёт ниже.

# Расшифровка

---

- **Вырожденная матрица** – матрица, определитель которой равен 0.
- **Плохо обусловленная матрица** – определитель не равен 0, но число обусловленности очень велико.

# Регуляризация

---

Переобучение нередко приводит к большим значениям коэффициентов. Чтобы решить проблему, добавим к функционалу регуляризатор, который штрафует за слишком большую норму вектора весов:

$$Q_\alpha(w) = Q(w) + \alpha R(w)$$

# Регуляризация

---

Наиболее распространеными являются L2 и L1-регуляризаторы:

$$R(w) = \|w\|_2 = \sum_{i=1}^d w_i^2, \quad -\sqcup\sqcup$$

$$R(w) = \|w\|_1 = \sum_{i=1}^d |w_i|. \quad - \sqcup |$$

# Регуляризация

---

- Коэффициент  $\alpha$  называется параметром регуляризации и контролирует баланс между подгонкой под обучающую выборку и штрафом за излишнюю сложность. Значение параметра подбирается перебором под каждую задачу.
- Свободный коэффициент  $w_0$  нет смысла регуляризовывать — если мы будем штрафовать за его величину, то получится, что мы учитываем представления о близости целевой переменной к нулю и отсутствии необходимости в учёте её смещения.
- Особенно об этом следует помнить, если в выборке есть константный признак и коэффициент  $w_0$  обучается наряду с остальными весами. В этом случае исключаем константный признак из регуляризатора.

# Регуляризация

---

Квадратичный (или L2) регуляризатор достаточно прост в использовании в отличие от L1-регуляризатора, у которого нет производной в нуле. При этом L1-регуляризатор имеет интересную особенность: его использование приводит к занулению части весов.

# Ridge регрессия

---

## sklearn.linear\_model.Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

[\[source\]](#)

Linear least squares with l2 regularization.

Minimizes the objective function:

$$\|y - Xw\|^2 + \alpha * \|w\|^2$$

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n\_samples, n\_targets)).

# LASSO

---

## sklearn.linear\_model.Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True,  
max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')  
[source]
```

Linear Model trained with L1 prior as regularizer (aka the Lasso)

The optimization objective for Lasso is:

```
(1 / (2 * n_samples)) * ||y - Xw||^2 + alpha * ||w||_1
```

Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=1.0` (no L2 penalty).

# ElasticNet

---

## sklearn.linear\_model.ElasticNet

```
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, normalize=False, precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic') [source]
```

Linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

# Практический кейс

---

- Займемся предсказанием цены на машины (датасет [здесь](#)).

```
import pandas as pd  
import numpy as np
```

```
df = pd.read_csv('_____car_data.csv')  
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

# Практический кейс

---

➤ Посмотрим на описание выборки:

```
df.describe()
```

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

# Практический кейс

---

- Что у нас есть в данных?

```
df.dtypes
```

```
Car_Name          object
Year              int64
Selling_Price    float64
Present_Price    float64
Kms_Driven       int64
Fuel_Type         object
Seller_Type       object
Transmission     object
Owner             int64
dtype: object
```

```
for col in df.columns:
    print(col, ': ', df[col].nunique(), sep='')
```

```
Car_Name: 98
Year: 16
Selling_Price: 156
Present_Price: 147
Kms_Driven: 206
Fuel_Type: 3
Seller_Type: 2
Transmission: 2
Owner: 3
```

# Практический кейс

---

- Сразу сделаем несколько бинарных переменных (OneHotEncoding):

```
df['Fuel_Type'].value_counts()
```

```
Petrol      239  
Diesel       60  
CNG          2  
Name: Fuel_Type, dtype: int64
```

```
df['is_petrol'] = (df['Fuel_Type'] == 'Petrol').astype(int)  
df['is_diesel'] = (df['Fuel_Type'] == 'Diesel').astype(int)
```

# Практический кейс

---

## **sklearn.preprocessing.OneHotEncoder**

```
class sklearn.preprocessing. OneHotEncoder(*, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>,  
handle_unknown='error')
```

[\[source\]](#)

Encode categorical features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka ‘one-of-K’ or ‘dummy’) encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the `sparse` parameter)

# Практический кейс

---

```
df['Seller_Type'].value_counts()
```

```
Dealer      195  
Individual   106  
Name: Seller_Type, dtype: int64
```

```
df['is_dealer'] = (df['Seller_Type'] == 'Dealer').astype(int)
```

```
df['Transmission'].value_counts()
```

```
Manual      261  
Automatic    40  
Name: Transmission, dtype: int64
```

```
df['is_manual'] = (df['Transmission'] == 'Manual').astype(int)
```

```
df.drop(['Fuel_Type', 'Seller_Type', 'Transmission'], axis=1, inplace=True)
```

# Практический кейс

---

```
old_df = df.copy()
```

```
from sklearn.preprocessing import LabelEncoder

df['Car_Name'] = df['Car_Name'].astype('category')
encoder = LabelEncoder()
df['Car_Name'] = encoder.fit_transform(df['Car_Name'])
```

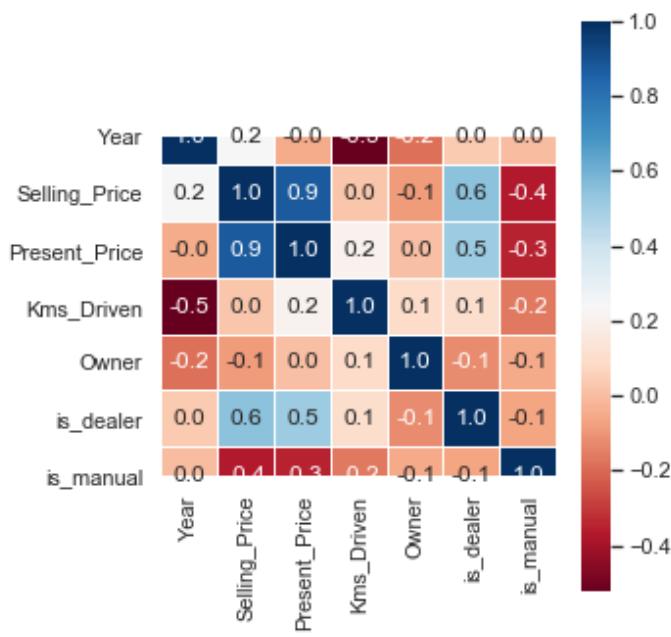
```
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Owner	is_petrol	is_diesel	is_cng	is_dealer	is_manual
0	90	2014	3.35	5.59	27000	0	1	0	0	1	1
1	93	2013	4.75	9.54	43000	0	0	1	0	1	1
2	68	2017	7.25	9.85	6900	0	1	0	0	1	1
3	96	2011	2.85	4.15	5200	0	1	0	0	1	1
4	92	2014	4.60	6.87	42450	0	0	1	0	1	1

# Практический кейс

---

- Проверим корреляционную матрицу, чтобы избежать мультиколлинеарности:



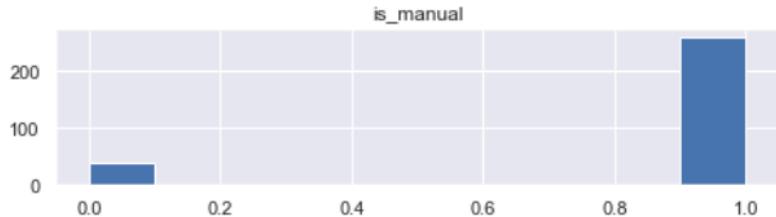
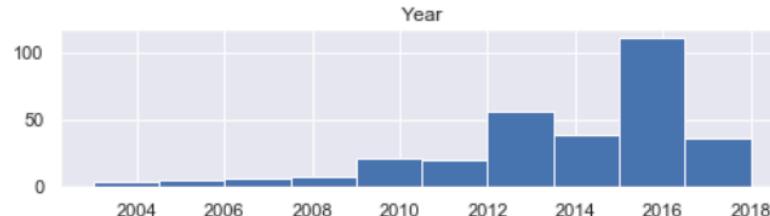
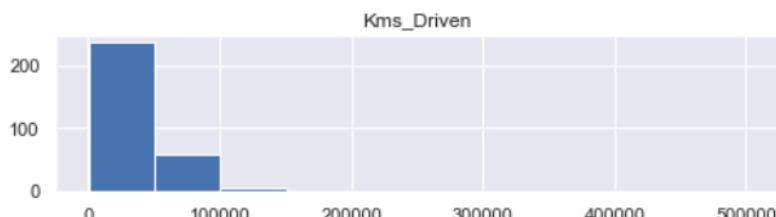
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(font_scale=1)
plt.subplots(figsize=(5, 5))
sns.heatmap(df.corr(), square=True,
            annot=True, fmt=".1f", linewidths=0.1, cmap="RdBu");
plt.tight_layout()
plt.savefig('car_corr.png')
```

# Практический кейс

---

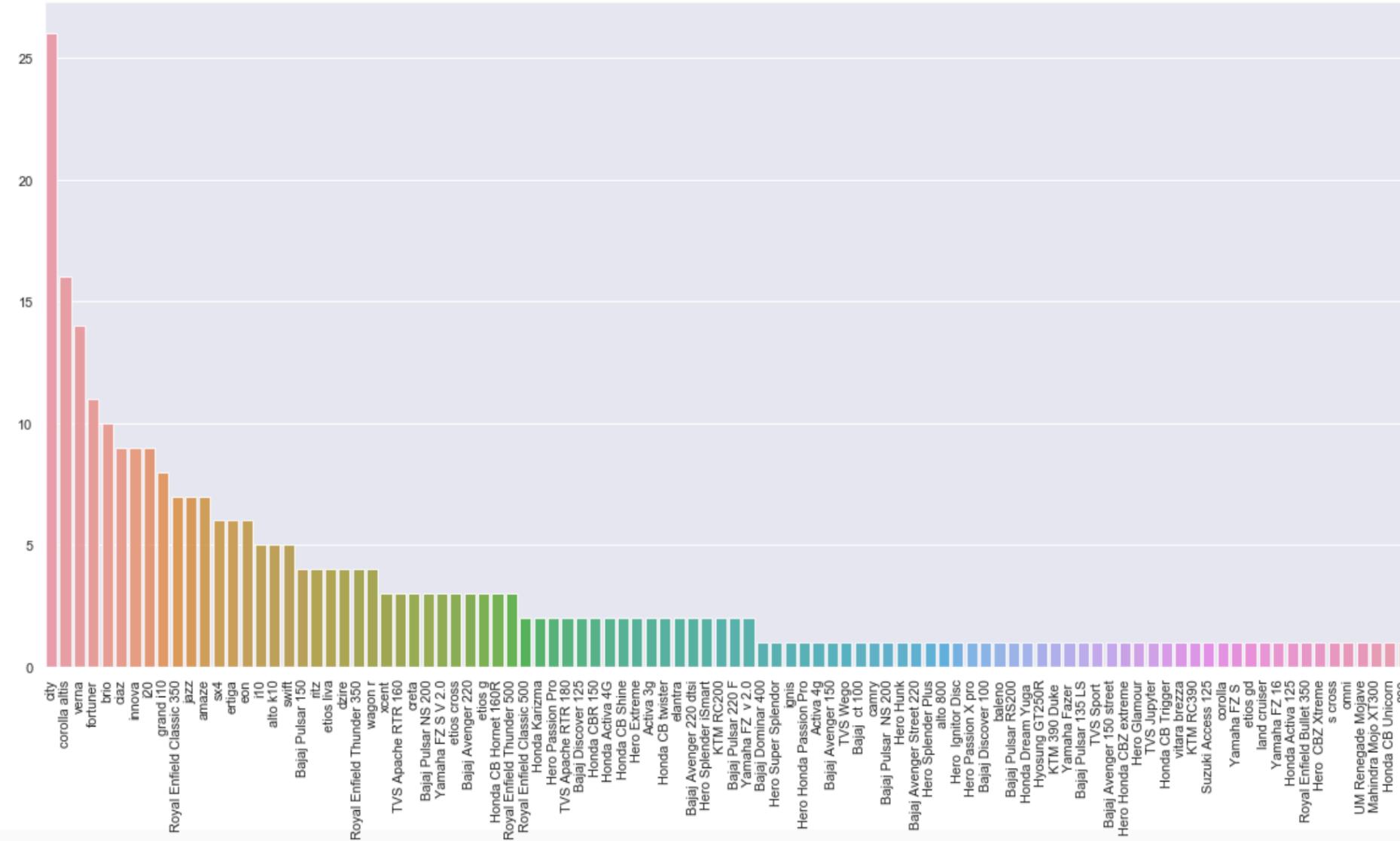
```
df.hist(figsize=(20, 6));  
plt.tight_layout()
```



```

car_type = df['Car_Name'].value_counts()
plt.figure(figsize=(20,10))
sns.barplot(car_type.index, car_type.values, alpha=0.9);
plt.xticks(rotation=90);

```



# Практический кейс

---

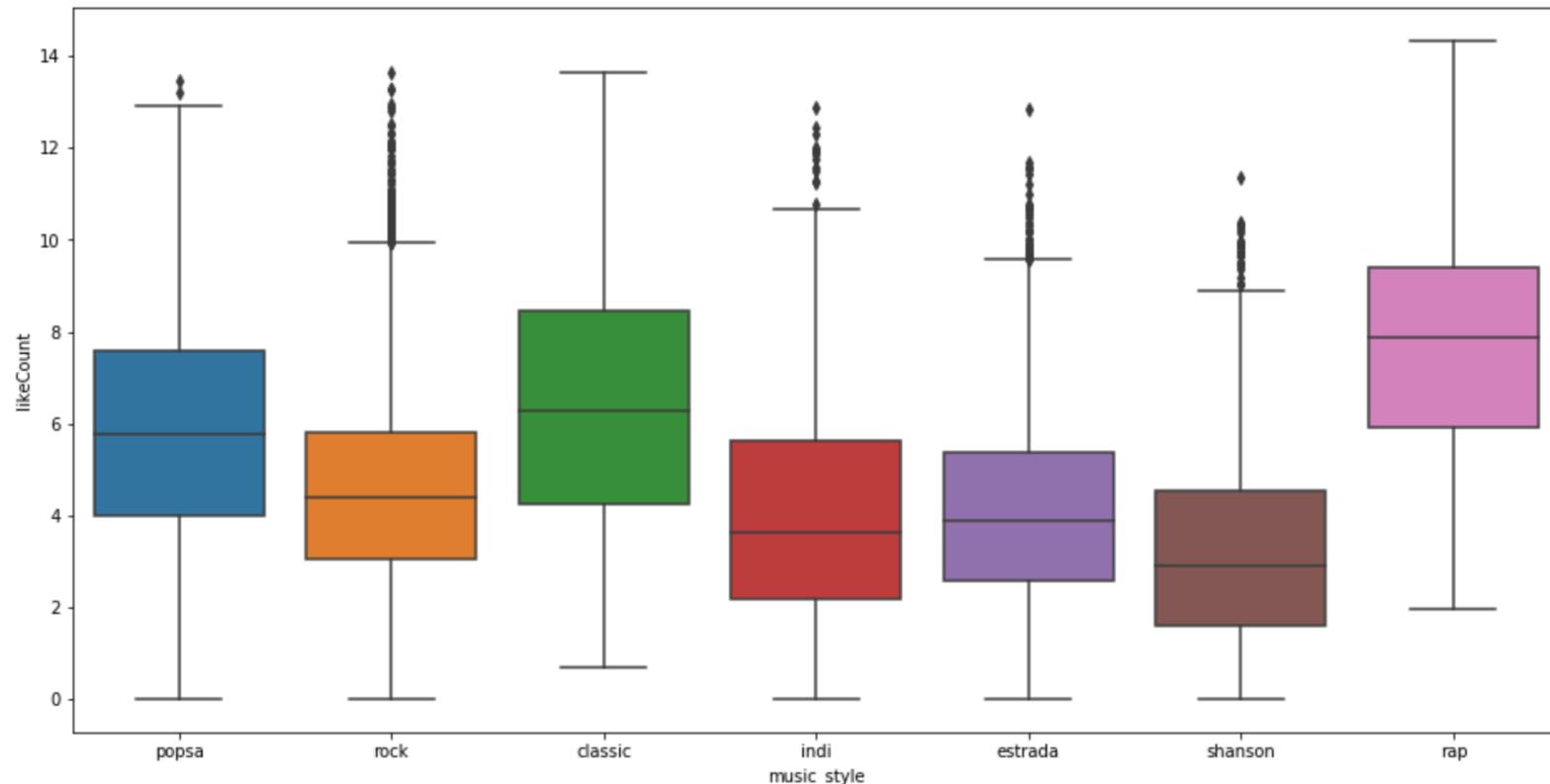
Шаги, которые в данном уроке мы делать не будем, но которые имеет смысл делать в реальных проектах:

1. Проверка разных гипотез (если в какой-то месяц цены выше, то сделать дамми на этот месяц)
2. Генерация новых признаков (например, из даты: год, является ли день рабочим, является ли месяц концом квартала)
3. Более подробная визуализация данных (scatterplot, pairplot, ящик с усами и т.д.)

# Пример визуализации

---

```
sns.boxplot(x='music_style', y='likeCount', data=df_log);
```

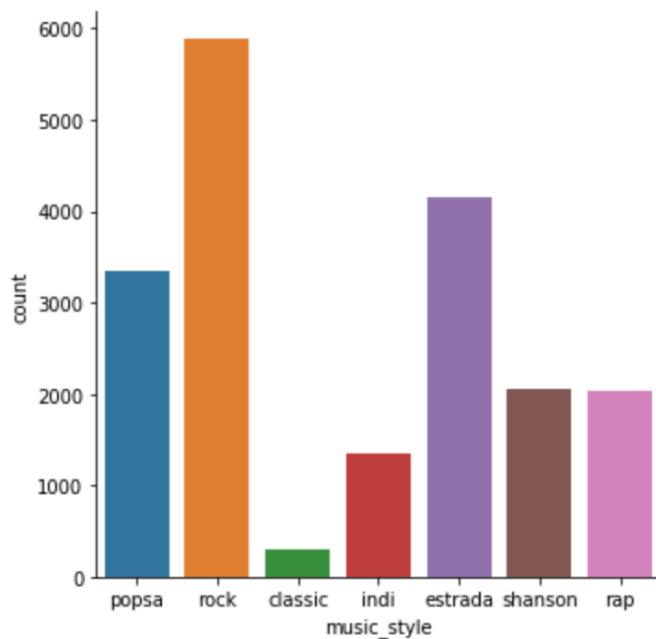


# Пример визуализации

---

```
sns.catplot('music_style', data=youtube, kind='count')|
```

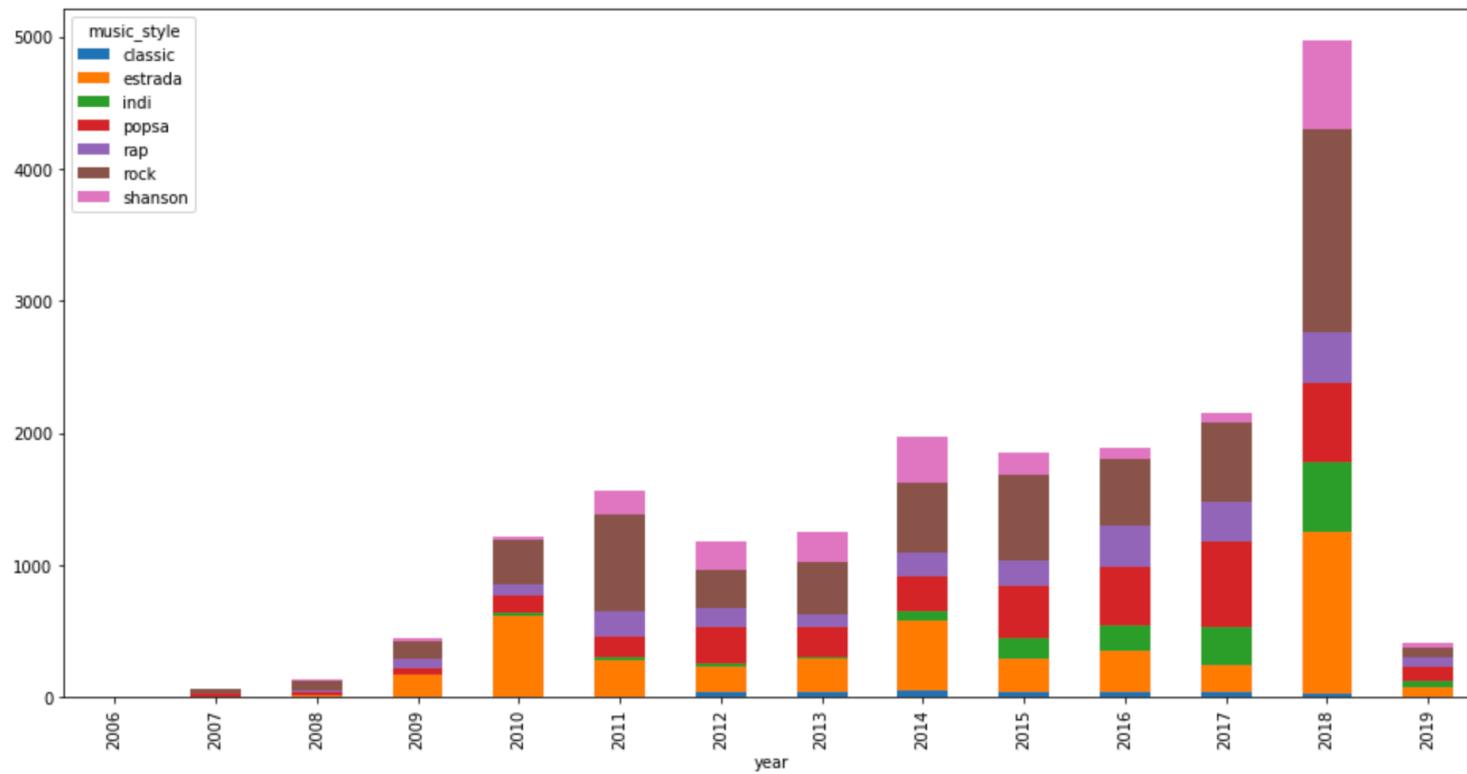
```
<seaborn.axisgrid.FacetGrid at 0x139cc1630>
```



# Пример визуализации

---

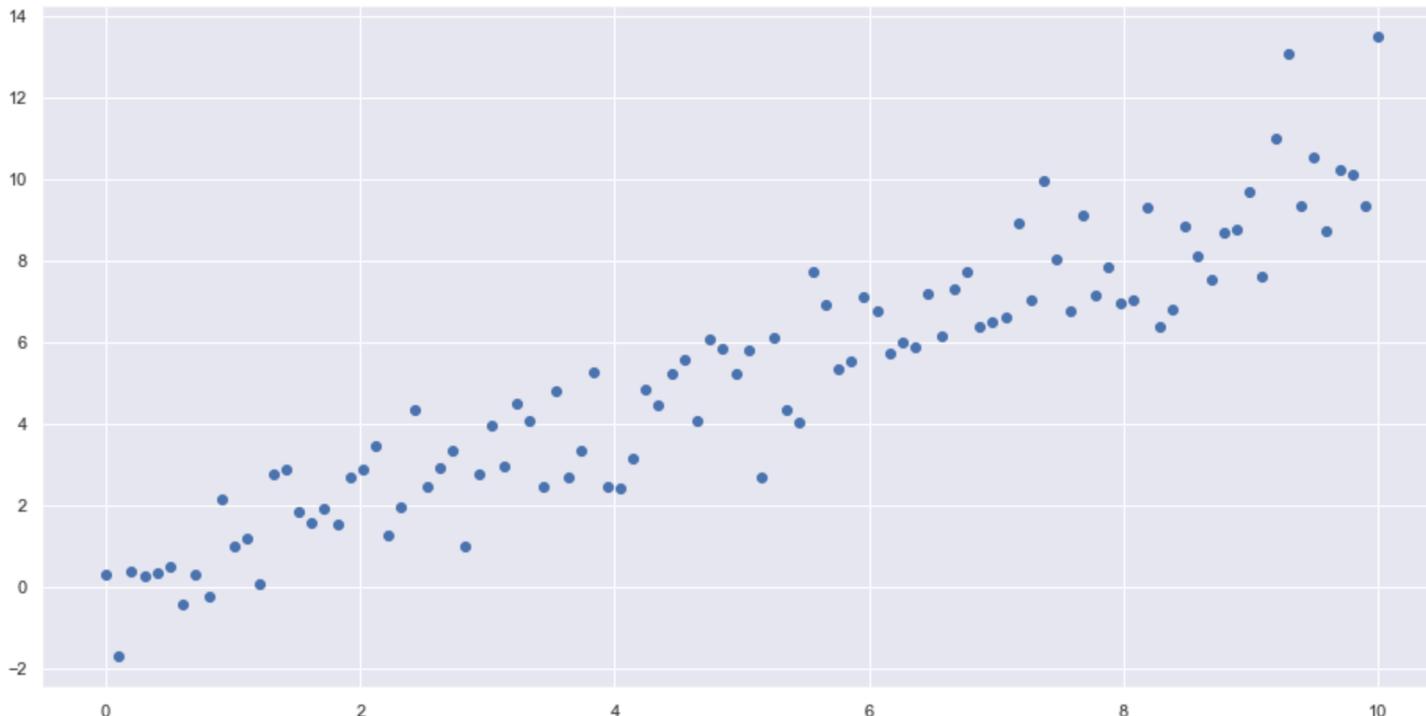
```
df.plot(kind='bar', stacked=True)  
<matplotlib.axes._subplots.AxesSubplot at 0x139ecd080>
```



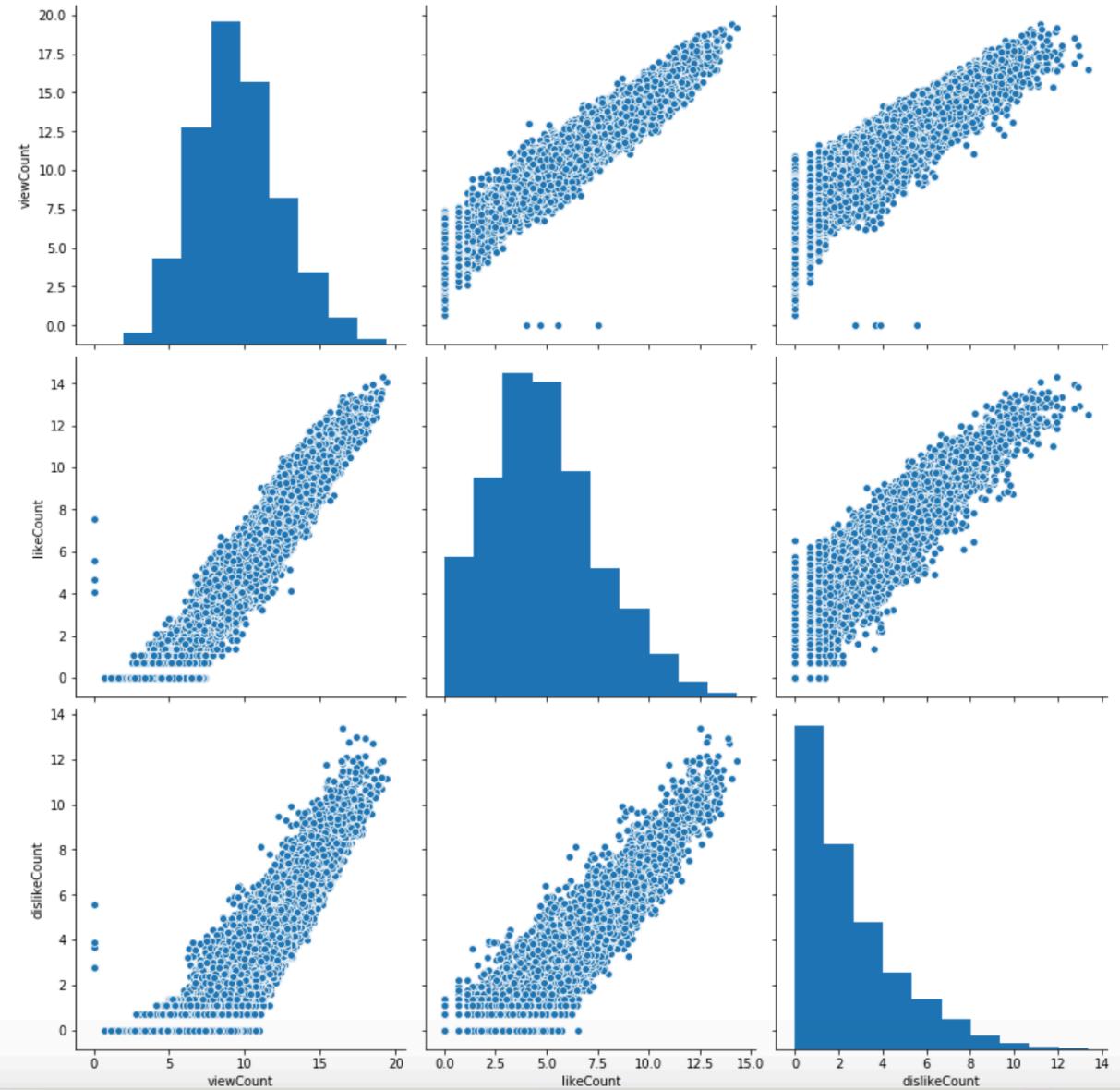
# Пример визуализации

---

```
X = np.linspace(0, 10, 100)
Y = X + np.random.normal(size = 100)
plt.scatter(X, Y);
```



```
columns = ['viewCount', 'likeCount', 'dislikeCount']
sns.pairplot(df_log[columns], height=4, aspect=1);
```



# Практический кейс

---

- Обработка пропусков: в нашем кейсе их нет, но в жизни бывают.
- Есть простые методы борьбы с ними: заполнение средним, медианой, средним по соседям (в случае временного ряда).
- Более продвинутые: обучить модель и ей предсказывать пропуски.

# Практический кейс

---

```
from sklearn.model_selection import train_test_split

X = df.drop('Selling_Price', axis=1)
y = df['Selling_Price']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

```
from sklearn.preprocessing import MinMaxScaler

to_scale = ['Year', 'Present_Price', 'Kms_Driven', 'Owner', 'Car_Name']
scaler = MinMaxScaler()
X_train.loc[:, to_scale] = scaler.fit_transform(X_train[to_scale])
X_test.loc[:, to_scale] = scaler.transform(X_test[to_scale])
```

# Практический кейс

---

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

def print_regression_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    print(f'RMSE = {rmse:.2f}, MAE = {mae:.2f}, R-sq = {r2:.2f}')
```

# Практический кейс

---

```
: from sklearn.linear_model import LinearRegression  
  
reg = LinearRegression().fit(X_train, y_train)  
y_pred = reg.predict(X_test)  
print_regression_metrics(y_test, y_pred)
```

RMSE = 1.88, MAE = 1.27, R-sq = 0.88

# Практический кейс

---

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

reg = Ridge()
param_grid_ = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
grid_ridge = GridSearchCV(estimator=reg, param_grid=param_grid_, scoring='neg_mean_squared_error', cv=5)
grid_ridge.fit(X_train, y_train)
grid_ridge.best_params_

{'alpha': 0.01}

reg = Ridge(alpha=0.01).fit(X_train, y_train)
y_pred = reg.predict(X_test)
print_regression_metrics(y_test, y_pred)

RMSE = 1.89, MAE = 1.27, R-sq = 0.87
```

# Пакеты для линейных методов

---

1. Linear Regression в scikit-learn (а также ее вариации: Ridge, Lasso, ElasticNet)
2. Vowpal-Wabbit

# Источники

---

1. Лекции по машинному обучению на ФКН ([URL](#) и [URL](#))