Нам нужно защитить данные клиентов страховой компании «Хоть потоп». Разработаем такой метод преобразования данных, чтобы по ним было сложно восстановить персональную информацию.

Нужно защитить данные, чтобы при преобразовании качество моделей машинного обучения не ухудшилось.

1. Загрузка данных

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [2]:
```
df=pd.read_csv("/datasets/insurance.csv") #прочитаем файл с обучающей выборкой
```

In [3]:
```
df.info()
df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 5 columns):
Пол                  5000 non-null int64
Возраст              5000 non-null float64
Зарплата             5000 non-null float64
Члены семьи          5000 non-null int64
Страховые выплаты    5000 non-null int64
dtypes: float64(2), int64(3)
memory usage: 195.4 KB
```

Out[3]:

	Пол	Возраст	Зарплата	Члены семьи	Страховые выплаты
0	1	41.0	49600.0	1	0
1	0	46.0	38000.0	1	1
2	0	29.0	21000.0	0	0
3	0	21.0	41700.0	2	0
4	1	28.0	26100.0	0	0
...	...	...	...	...	...
4995	0	28.0	35700.0	2	0
4996	0	34.0	52400.0	1	0
4997	0	20.0	33900.0	2	0
4998	1	22.0	32700.0	3	0
4999	1	28.0	40600.0	1	0

5000 rows × 5 columns

In [4]:
```
features = df.drop('Страховые выплаты', axis=1)
target = df['Страховые выплаты']
```

In [5]:
```
#создадим класс линейной регрессии

class LinearRegression:
    def fit(self, train_features, train_target):
        #найдем матрицу признаков
        X = np.concatenate((np.ones((train_features.shape[0], 1)), train_features), axis=1)
        #найдем вектор предсказаний
        y = train_target
        #найдем вектор весов
        w = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
        self.w = w[1:]
        self.w0 = w[0]

    def predict(self, test_features):
        return test_features.dot(self.w) + self.w0
```

In [6]:
```
#обучим модель и вычислим качество модели
model = LinearRegression()
model.fit(features, target)
predictions = model.predict(features)
print(r2_score(target, predictions))
```

```
0.4249455028666801
```

2. Умножение матриц

Обозначения:

- X — матрица признаков (нулевой столбец состоит из единиц)
- y — вектор целевого признака
- P — матрица, на которую умножаются признаки
- w — вектор весов линейной регрессии (нулевой элемент равен сдвигу)

In [7]:
```
X=features.values
```

In [8]:
```
X
```

Out[8]:
```
array([[1.00e+00, 4.10e+01, 4.96e+04, 1.00e+00],
       [0.00e+00, 4.60e+01, 3.80e+04, 1.00e+00],
       [0.00e+00, 2.90e+01, 2.10e+04, 0.00e+00],
       ...,
       [0.00e+00, 2.00e+01, 3.39e+04, 2.00e+00],
       [1.00e+00, 2.20e+01, 3.27e+04, 3.00e+00],
       [1.00e+00, 2.80e+01, 4.06e+04, 1.00e+00]])
```

In [9]:
```
#найдем вектор целевого признака
y=target.values
```

Предсказания:

$$a = Xw$$

Задача обучения:

$$w = \arg\min_w MSE(Xw, y)$$

Формула обучения:

$$w = (X^T X)^{-1} X^T y$$

$(AB)^T = B^T A^T$

In [10]:
```
# создадим матрицу из случайных чисел размером 4*4
P = np.random.rand(4,4)
#np.random.seed
```

лучше фиксировать генераторы псевдослучайных чисел через аргумент `seed` – это делает генерацию воспроизводимой, т.е. при запуске кода, получатся точно такие же числа. Это может оказаться полезным при поиске ошибок.

In [11]:
```
P
```

Out[11]:
```
array([[0.685858  , 0.83050326, 0.71367305, 0.52132436],
       [0.44094502, 0.54607719, 0.80053109, 0.8716164 ],
       [0.39590423, 0.40982009, 0.21409886, 0.81704216],
       [0.25142865, 0.93672662, 0.87454674, 0.51102953]])
```

In [12]:
```
#Найдем обратимую матрицу, если выдаст ошибку, значит матрица необратимая

P_inv=np.linalg.inv(P)
```

In [13]:
```
P_inv
#матрица обратима
```

Out[13]:
```
array([[ 2.31619732,  0.21610786, -0.47526018, -1.97159961],
       [ 0.08702433, -2.39159394,  1.3826955 ,  1.77967028],
       [-0.09184679,  2.19597346, -2.11229248, -0.27460742],
       [-1.14191458,  0.51944663,  1.13418011,  0.13464806]])
```

In [14]:
```
#умножим матрицу X на матрицу P
XP=X@P
```

In [15]:
```
XP
```

Out[15]:
```
array([[19655.86592199, 20351.23264403, 10653.7136694 , 40562.05958388],
       [15064.89570203, 15599.21953295,  8173.45582457, 31088.20732719],
       [ 8326.77627009,  8622.05803766,  4519.29155428, 17183.16216001],
       ...,
       [13430.47521042, 13905.69590132,  7275.71121891, 27716.18348916],
       [12957.20930914, 13416.77118278,  7021.98186359, 26738.50848815],
       [16086.99555194, 16655.7528695 ,  8716.41698552, 33197.34916307]])
```

In [16]:
```
#создадим из матрицы датафрейм
XP_df=pd.DataFrame(XP)
```

In [17]:
```
XP_df
```

Out[17]:

	0	1	2	3
0	19655.865922	20351.232644	10653.713669	40562.059584
1	15064.895702	15599.219533	8173.455825	31088.207327
2	8326.776270	8622.058038	4519.291554	17183.162160
3	16518.969162	17102.838647	8946.482892	34089.983926
4	10346.132764	10712.424901	5611.108905	21349.726866
...	...	...	...	...
4995	14146.630388	14647.740673	7667.493424	29193.832302
4996	20760.625297	21494.075840	11246.873099	42843.154983
4997	13430.475210	13905.695901	7275.711219	27716.183489
4998	12957.209309	13416.771183	7021.981864	26738.508488
4999	16086.995552	16655.752869	8716.416986	33197.349163

5000 rows × 4 columns

In [18]:
```
#обучим модель и вычислим качество модели
model_P = LinearRegression()
model_P.fit(XP_df, target)
predictions = model_P.predict(XP_df)
print(r2_score(target, predictions))
```

```
0.4249455028666595966
```

Обоснование:

$\omega = (X^T X)^{-1} X^T a$

$\omega' = ((XP)^T XP)^{-1} (XP)^T a'$

$\omega' = ((P^T X^T XP)^{-1} P^T X^T a'$

$\omega' = P^{-1} (X^T X)^{-1} (P^T)^{-1} P^T X^T a'$

$(P^T)^{-1} P^T$ сократятся

$\omega' = P^{-1} (X^T X)^{-1} X^T a'$

После умножения на обратимую матрицу наша формула регрессии будет иметь вид

$$a' = (XP)w'$$

В формуле регрессии участвует вектор y – наш целевой признак, а вектор предсказаний a получается из вектора ω и матрицы прихнаков X: $a = X\omega$. Вместо a и a' должен быть y. В итоге, если сравнить выражения для ω' и ω, то окажется, что $\omega' = P^{-1}\omega$. А дальше:

$$a' = (XP)\omega' = XPP^{-1}\omega = X\omega = a.$$

3. Алгоритм преобразования

Алгоритм

In [19]:
```
def back_to_X (X, P):
    back_X=X.dot(P).dot(np.linalg.inv(P))
    return back_X
```

Обоснование

умножаем на обратную матрицу P

4. Проверка алгоритма

In [20]:
```
back_X=back_to_X (X, P)
```

In [21]:
```
back_X=pd.DataFrame(back_X)
```

In [22]:
```
back_X=back_X.round()
```

In [23]:
```
#обучим модель и вычислим качество модели
model_X = LinearRegression()
model_X.fit(back_X, target)
predictions = model_X.predict(back_X)
print(r2_score(target, predictions))
```

```
0.42494550286668
```