Семинар: Линейная регрессия. Задачи

Задание: Линейная регрессия.

В этом задании мы рассмотрим метод линейной регрессии (метод наименьших квадратов - МНК). Мы будем работа
содержащим информацию о бриллиантах. Описание можно посмотреть [здесь (https://www.kaggle.com/shivam2503/d](https://www.kaggle.com/shivam2503/d)

Content

price - цена в долларах США (326 - 18,823)

carat - вес бриллианта (0.2 - 5.01)

cut - качество среза (Fair, Good, Very Good, Premium, Ideal)

color - цвет бриллианта (от J (худший) до D (лучший))

clarity - степень чистоты бриллианта (от I1 (худшая), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (лучшая))

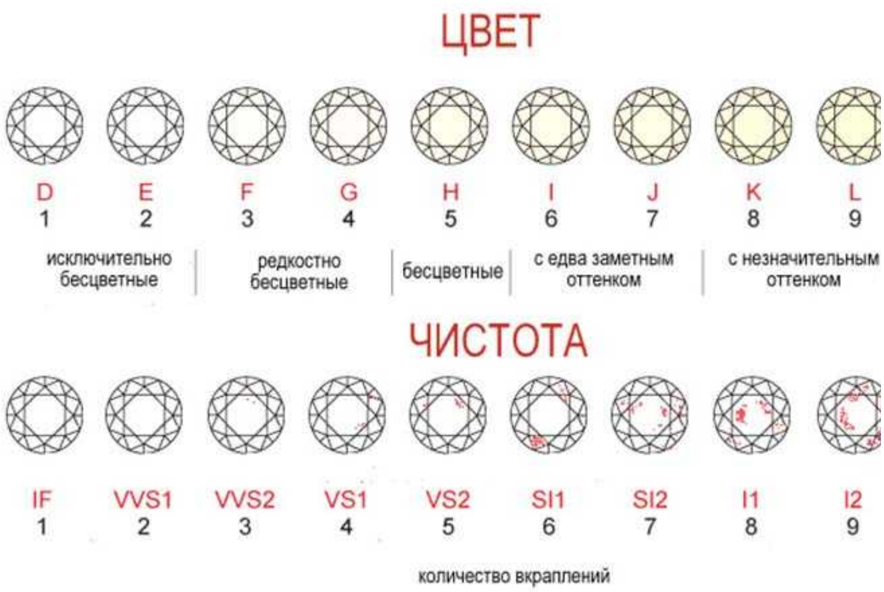
x - длина в миллиметрах (0 - 10.74)

y - ширина в миллиметрах (0 - 58.9)

z - глубина в миллиметрах (0 - 31.8)

depth - общий процент глубины = z / mean(x, y) = 2 * z / (x + y) (43 - 79)

table - ширина верхней плоскости бриллианта в самом широком месте (43 - 95)



Наглядная иллюстрация цвета и чистоты бриллианта с чуть более расширенными показателями (до M по цвету и д

В [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score

import warnings
warnings.filterwarnings("ignore")
```

В [5]:
```python
data = pd.read_csv('diamonds.csv')  # загружаем датафрейм
data.head()                          # выводим первые 5 строк датафрейма
```

Out[5]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

В [7]:
```
data.tail() # выводим последние 5 строк датафрейма
```

Out[7]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

Мы будем решать задачу предсказания цены бриллианта `price` в зависимости от его характеристик.

Задача 1.1. Есть ли в наборе данных пропущенные значения? Если да, удалите их.

В [3]:
```
data.isnull().any() # пропущенных значения в датафрейме отсутствуют
```

Out[3]:
```
Unnamed: 0    False
carat         False
cut           False
color         False
clarity       False
depth         False
table         False
price         False
x             False
y             False
z             False
dtype: bool
```

Задача 1.2. Есть ли в наборе данных бессмысленные столбцы (признаки, не несущие дополнительной информаци

В [4]:
```
data = data.drop('Unnamed: 0', axis=1)   # удаляем столбец "Unnamed: 0" с помощью функции drop
data.head()
```

Out[4]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Задача 1.3. Линейная регрессия основана на предположении о линейной связи между признаками и целевой перем выбором переменных для включения в модель имеет смысл проверить, насколько эта связь выполняется. Для сле потребуются выборочные корреляции между признаками. Выведите матрицу выборочных корреляций между всеми признаками и целевой переменной (то есть в этой матрице будет $k + 1$ строка, где k – количество вещественных п

Какие вещественные признаки коррелируют с целевой переменной больше всего?

В [5]:
```
(data.dtypes == "float").sum() # выводим колличесво вещественных признаков в датафрейме
```

Out[5]: 6

В [6]:
```
plt.figure(figsize = (12,6))  # создаем фигуру, в которой будет построен график и указываем её
sns.heatmap(data.corr(), annot=True, linewidths= 1)
# визуализизуем матрицу корреляции с помощью тепловой карты из библиотеки  seaborn,
# параметром annot=True включаем аннотацию к матице. в каждой клетке - коэф. кор.
```

Out[6]: <Axes: >

Вещественных признаков 6, а в матрице 7 строк (значит все правильно). Видно, что сильнее всего с ценой к carat, дальше идут, с небольшой разницей, параметры длины, ширины и глубины бриллианта. Признак dep переменной очень слабо.

Задача 1.4. Так как линейная модель складывает значения признаков с некоторыми весами, нам нужно аккуратно о признаки. Закодируйте категориальные переменные при помощи OneHot-кодирования.

В [7]:
```
y = data['price']                    # выделение зависимой переменной
X = data.drop('price', axis=1)    # удаление зависимой переменной из матрицы признаков
X.head()
```

Out[7]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75

В [8]:
```
X_dum = pd.get_dummies(X, drop_first=True)
# кодирую данные, передаю параметр drop_first=True, чтобы избежать линейной зависимости
X_dum # вывод датафрейма
```

Out[8]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...	color_H	color_I	color_J	c
0	0.23	61.5	55.0	3.95	3.98	2.43	0	1	0	0	...	0	0	0	
1	0.21	59.8	61.0	3.89	3.84	2.31	0	0	1	0	...	0	0	0	
2	0.23	56.9	65.0	4.05	4.07	2.31	1	0	0	0	...	0	0	0	
3	0.29	62.4	58.0	4.20	4.23	2.63	0	0	1	0	...	0	1	0	
4	0.31	63.3	58.0	4.34	4.35	2.75	1	0	0	0	...	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
53935	0.72	60.8	57.0	5.75	5.76	3.50	0	1	0	0	...	0	0	0	
53936	0.72	63.1	55.0	5.69	5.75	3.61	1	0	0	0	...	0	0	0	
53937	0.70	62.8	60.0	5.66	5.68	3.56	0	0	0	1	...	0	0	0	
53938	0.86	61.0	58.0	6.15	6.12	3.74	0	0	1	0	...	1	0	0	
53939	0.75	62.2	55.0	5.83	5.87	3.64	0	1	0	0	...	0	0	0	

53940 rows × 23 columns

Задача 1.5. Разделите выборку на тренировочную и тестовую. Долю тестовой выборки укажите равной 0.3.

В [9]:
```
from sklearn.model_selection import train_test_split   # импортирую нужный модуль
X_train, X_test, y_train, y_test = train_test_split(X_dum, y, test_size=0.3, random_state=1)
# разделяю выборку. За размер доли тестовой выборки отвечает параметр test_size, его, как указ
# кроме того передаю парметр random_state (аналог np.random.seed()), для того, чтобы разбиение
```

Задача 1.6. Зачастую при использовании линейных моделей вещественные признаки масштабируются. При этом о
теряют прямую статистическую интерпретацию ("при увеличении X_1 на 1, y увеличивается на w_1"), но приобретаю
задачах машинного обучения. В этой задаче масштабируйте вещественные признаки тренировочной и тестовой вы
`StandardScaler` .

В [10]:
```
from sklearn.preprocessing import StandardScaler           # импортирую модуль для нормировки

scaler = StandardScaler()                                  # создаю "нормировщик"
scaler.fit(X_train)                                        # обучаю его на данных из X_train
scaled_X_train = scaler.transform(X_train)                 # с помощью уже обученного "нормир
scaled_X_test = scaler.transform(X_test)                   # датафреймах X_train и X_test
```

Задача 1.7. Оцените линейную регрессию на тренировочной выборке. Выведите среднеквадратичную ошибку на тр
выборках.

В [11]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

В [12]:
```
# создаем модель линейной регрессии
regr = LinearRegression()

# обучаем модель на train датасете, используя масштабированные признаки
regr.fit(scaled_X_train, y_train)

# предсказываем значения y_train и y_test с помощью обученной модели
y_train_pred = regr.predict(scaled_X_train)
y_test_pred = regr.predict(scaled_X_test)

# выводим значение средней квадратичной ошибки (MSE) для train и test датасетов
print("Train MSE: ")
print("Test MSE: ")

# выводим коэффициент квадрата детерминации (R2) для train и test датасетов, чтобы проверить m
print(f'\nTrain: {(r2_score(y_train, y_train_pred))}')
print(f'Test: {(r2_score(y_test, y_test_pred))}')
```

```
Train MSE: 1144.60
Test MSE: 1095.78

Train: 0.9189349877290272
Test: 0.9217238681492771
```

Задача 1.8. Изучите документацию модуля `LinearRegression` и выведите полученные оценки коэффициентов (ве
вещественные переменные, оценки коэффициентов которых по модулю на порядок превышают оценки прочих вещ

В [13]:
```
df = pd.DataFrame()                                        # создаю датафрейма
for i in range(len(list(X_train.columns.values))):         # для каждого признака нахожу соответс
    df[list(X_train.columns.values)[i]] = pd.Series(abs(regr.coef_[i]))

cat_features_mask_1 = (X.dtypes == "object").values        # создаю маску для отбора только вещес
df[list(X[X.columns[~cat_features_mask_1]].columns)].T.sort_values(0, ascending=False)
# делаю таблицу и осуществляю коэффиуиенты по убыванию
```

Out[13]:

	0
carat	5277.247129
x	1050.102162
depth	83.585954
table	56.869625
z	12.888928
y	0.654293

Среди вещественных переменных слишком большие веса получились у веса бриллианта и параметра x. Следующ
примерно одинаковые по соотношению веса, а вот последняя переменная (параметр y) имеет очень маленький вес

Задача 1.9. Как можно заметить из анализа корреляционной матрицы в задаче 3.3, *между некоторыми признакам*
корреляция, что может быть индикатором проблемы *мультиколлинеарности* (наличия линейной зависимости меж
переменными (факторами) модели). Различия в порядке коэффициентов, выявленные в предыдущей задаче также
присутствие. Как известно, для решения этой проблемы можно
либо исключить некоторые признаки из модели, *либо использовать регуляризацию*. Мы воспользуемся вторым в

Вспомним, что смысл регуляризации заключается в том, чтобы изменить функцию потерь так, чтобы устранить про
мультиколлинеарности. При L1-регуляризации предлагается минимизировать следующую функцию потерь:

$$\|y - X\hat{w}\|^2 + \alpha \sum_{i=1}^{k} |w_i|$$

Такая модель называется Lasso-регрессией.

При L2-регуляризации предлагается минимизировать следующую функцию потерь:

$$\|y - X\hat{w}\|^2 + \frac{1}{2}\alpha\|w\|^2$$

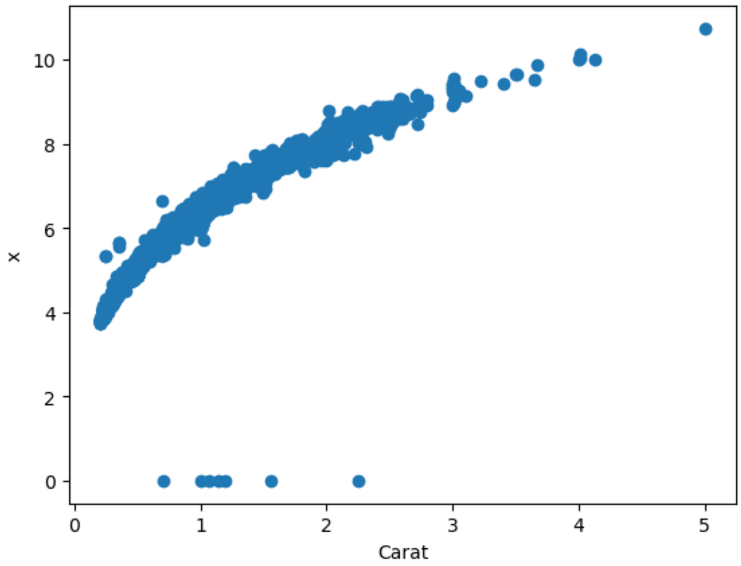
Такая модель называется Ridge-регрессией.

Выберите два признака, сильно коррелирующих между собой, и постройте зависимость (scatter) одного от другого.

Обучите Lasso-регрессию и Ridge-регрессию, уставновив гиперпараметр регуляризации равным 10. Для этого испо
`Ridge` из `sklearn` . Сильно ли уменьшились веса? Сделайте вывод о том, насколько сильно проблема мультикол
изначальной регрессии

В [14]:
```
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
```

В [15]:
```
plt.scatter((X_train['carat']), X_train['x'])
# на матрице корреляции было видно, что переменные carat и x между собой сильно коррелируют.
# далее вывожу scatter-plot, чтобы проследить их зависимсоть
plt.xlabel('Carat')
plt.ylabel('x')
plt.show();
```



В [16]:
```
from sklearn.linear_model import Lasso, Ridge                # импортирую Lasso и Ridge

lasso = Lasso(10.0).fit(scaled_X_train, y_train)             # создаю Lasso-модель с λ = 10

ridge = Ridge(10.0).fit(scaled_X_train, y_train)             # делаю то же самое с Ridge
```

В [17]:
```
print(regr.coef_)
print('\n', lasso.coef_)
print('\n', ridge.coef_)
```

```
[ 5.27724713e+03 -8.35859541e+01 -5.68696252e+01 -1.05010216e+03
 -6.54293136e-01 -1.28889279e+01  1.71502354e+02  4.16023988e+02
  3.39486233e+02  3.13132588e+02 -8.45948129e+01 -1.02390339e+02
 -2.00244063e+02 -3.57144807e+02 -4.40474155e+02 -5.25835595e+02
  9.87614017e+02  1.59869330e+03  1.04783493e+03  1.67852566e+03
  1.82339758e+03  1.29267929e+03  1.46168957e+03]

 [4783.41094049  -88.18838824  -67.34812205 -605.4571011     -0.
   -9.04245448   45.90699967  207.92331741  150.09822521  139.4370833
  -12.65867611  -35.62012708 -122.72563295 -284.88080225 -369.04859265
 -467.2452535   669.45958233  863.02801363  409.79199968 1053.42448122
 1102.04028249  852.95943422  958.90274725]

 [ 5.23973797e+03 -8.18029757e+01 -5.70570935e+01 -1.00923042e+03
 -2.55973659e+00 -1.64756784e+01  1.72041187e+02  4.17140870e+02
  3.40412927e+02  3.14372969e+02 -8.41350398e+01 -1.02177926e+02
 -1.99692501e+02 -3.56285462e+02 -4.39199150e+02 -5.24544818e+02
  9.74870405e+02  1.56782640e+03  1.02157962e+03  1.65263349e+03
  1.79332538e+03  1.27483844e+03  1.44104250e+03]
```

Смотря на коэффициенты кажется, что Lasso справилось с задачей лучше, чем Ridge. В общем веса немног один коэффициент даже занулилися. Проблема мультиколлениарности была, но сказать, что она очень сил нельзя, но нам важно, что мы смогли ее подправить и улучшить результат.

Задача 3.10 Как обсуждалось на семинарах, Lasso-регрессию можно использовать для отбора наиболее информат следующих значений параметра регуляриазции α: 0.1, 1, 10, 100, 200 – обучите Lasso- и Ridge-регрессии и построй евклидовой нормы весов (`np.linalg.norm()` от вектора оценок коэффициентов) в зависимости от параметра α. К является численной характеристикой величины вектора, а потому по норме можно судить о том, насколько больши вектор оценок коэффициентов.

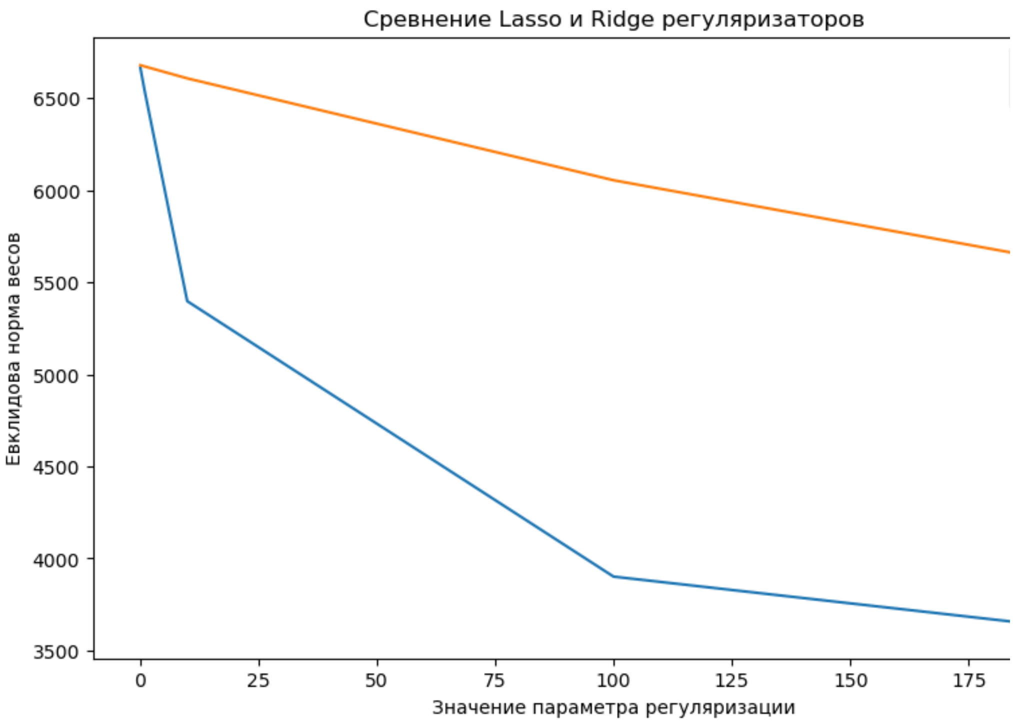
Какой метод сильнее уменьшает веса?

В [18]:
```
import numpy as np
```

В [19]:
```
alpha = [0.1, 1, 10, 100, 200]
lasso_evklid_coefs = []
ridge_evklid_coefs = []
for i in alpha:
    lasso_evklid_coefs.append(np.linalg.norm(Lasso(i).fit(scaled_X_train, y_train).coef_))
    ridge_evklid_coefs.append(np.linalg.norm(Ridge(i).fit(scaled_X_train, y_train).coef_))
```

В [20]:
```
fig, axes = plt.subplots(figsize=(10,6))                       # создаем поверхность и задае
axes.plot(alpha, lasso_evklid_coefs, alpha, ridge_evklid_coefs)  # рисуем нужные графики
axes.legend(['lasso', 'ridge'])                                # добавляем легенду
plt.xlabel('Значение параметра регуляризации')                 # подписываем оси графика
plt.ylabel('Евклидова норма весов')
plt.title('Сревнение Lasso и Ridge регуляризаторов')           # даем графику название
```

Out[20]:
```
Text(0.5, 1.0, 'Сревнение Lasso и Ridge регуляризаторов')
```



Из графика видно, что Lasso сильнее уменьшает веса как при маленьких значениях параметра регуляризации, так