

Задание

Домашнее задание по дисциплине направлено на решение комплексной задачи машинного обучения. Домашнее задание включает выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.


```
1 import numpy as np
2 import pandas as pd
3
4 data = pd.read_csv('Data/HW/car_ad.csv', sep=',', encoding='latin-1')
5 data2 = pd.read_csv('Data/HW/car_ad.csv', sep=',', encoding='latin-1')
6 data.head(10)
```




	car	price	body	mileage	engV	engType	registration	year	model
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Ku
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Cla
2	Mercedes-Benz	35000.0	other	135	5.5	Petrol	yes	2008	CL 5
3	Mercedes-	17800.0	van	162	1.8	Diesel	yes	2012	B 1

▼ 2. Проведение разведочного анализа данных.


```
1 data.shape
```

 (9576, 10)

```
1 data.dtypes
```


 car object
price float64
body object
mileage int64
engV float64
engType object
registration object
year int64
model object
drive object
dtype: object

```
1 # Проверка на пустые значения
2 data.isnull().sum()
```

 car 0
price 0
body 0
mileage 0
engV 434
engType 0
registration 0
year 0
model 0
drive 511
dtype: int64

▼ Удаление строк с пропусками в данных.

```
1 #Пустые значения составляют примерно 5 процентов, поэтому не будем их удалять
2 # Удаление строк, содержащих пустые значения
3 data_new = data.dropna(axis=0, how='any')
4 (data.shape, data_new.shape)
```

 ((9576, 10), (8739, 10))

▼ Построение графиков, необходимых для понимания структуры данных.

```
1 #основные статические характеристики набора данных
2 data_new.describe()
```



	price	mileage	engV	year
count	8739.000000	8739.000000	8739.000000	8739.000000
mean	15733.542261	140.095434	2.588607	2006.609681
std	24252.904810	97.892213	5.416670	6.968947
min	0.000000	0.000000	0.100000	1959.000000
25%	5000.000000	71.000000	1.600000	2004.000000
50%	9250.000000	130.000000	2.000000	2008.000000
75%	16800.000000	195.500000	2.500000	2012.000000
max	547800.000000	999.000000	99.990000	2016.000000

```
1 # Определим уникальные значения для целевого признака "тип кузова"
2 data_new['body'].unique()
```



```
array(['crossover', 'sedan', 'other', 'van', 'wagon', 'hatch'],
      dtype=object)
```

```
1 # Определим уникальные значения для целевого признака "Марка автомобиля"
2 data_new['car'].unique()
```



```
array(['Ford', 'Mercedes-Benz', 'Nissan', 'Honda', 'Renault', 'BMW',
      'Land Rover', 'Volkswagen', 'Audi', 'Chrysler', 'Jaguar',
      'Mitsubishi', 'Kia', 'Porsche', 'Toyota', 'Hyundai', 'Opel',
      'Chevrolet', 'Skoda', 'Daewoo', 'Mazda', 'Lexus', 'Infiniti',
      'Subaru', 'VAZ', 'Alfa Romeo', 'Smart', 'Peugeot', 'Suzuki',
      'Chery', 'Bentley', 'Volvo', 'ZAZ', 'Citroen', 'Dodge', 'Fiat',
      'Jeep', 'SsangYong', 'Seat', 'MINI', 'Dacia', 'Hummer', 'Geely',
      'Maserati', 'BYD', 'Cadillac', 'Acura', 'Aston Martin', 'Tesla',
      'Rover', 'GAZ', 'GMC', 'Lincoln', 'EUAZ', 'Moskvich-AZLK', 'FAW',
      'UAZ', 'Rolls-Royce', 'TATA', 'ZX', 'Lifan', 'Mercury', 'Groz',
      'Great Wall', 'Moskvich-Izh', 'Saab', 'Lancia', 'Aro', 'Ferrari',
      'Bogdan', 'Dadi', 'MG', 'Samand', 'JAC', 'Samsung', 'Lamborghini',
      'Daihatsu', 'Hafei', 'SMA', 'Isuzu', 'Huanghai', 'Wartburg',
      'Buick'], dtype=object)
```

```
1 # Определим уникальные значения для целевого признака "Тип привода"
2 data_new['drive'].unique()
```




```
array(['full', 'rear', 'front'], dtype=object)
```

```
1 data_new['engV'].unique()
```



```
array([ 2.5 ,  1.8 ,  5.5 ,  2.  ,  1.5 ,  2.2 ,  1.2 ,  4.8 ,  5.  ,
        3.  ,  4.4 ,  1.6 ,  2.98,  2.4 ,  2.8 ,  3.5 ,  2.99,  1.9 ,
        1.7 ,  4.5 ,  3.6 ,  1.4 ,  2.7 ,  4.  ,  3.8 ,  5.7 , 99.99,
        3.2 ,  3.7 ,  4.7 ,  1.  ,  4.6 ,  0.11,  4.2 ,  0.8 ,  2.3 ,
        1.3 ,  6.  ,  2.6 ,  0.6 ,  1.25,  5.46,  6.3 ,  5.6 ,  8.3 ,
        3.3 ,  1.1 ,  6.1 ,  0.65,  1.78,  2.1 ,  3.4 ,  7.  ,  5.2 ,
       75.  ,  6.5 ,  1.23,  8.  ,  6.2 ,  2.9 ,  1.34,  0.7 ,  1.39,
       90.  ,  5.3 ,  4.67, 20.  , 14.  , 11.5 ,  1.45,  9.  , 10.  ,
        4.66, 15.  ,  0.9 , 12.  ,  2.57,  1.91,  1.33,  2.49,  1.12,
```

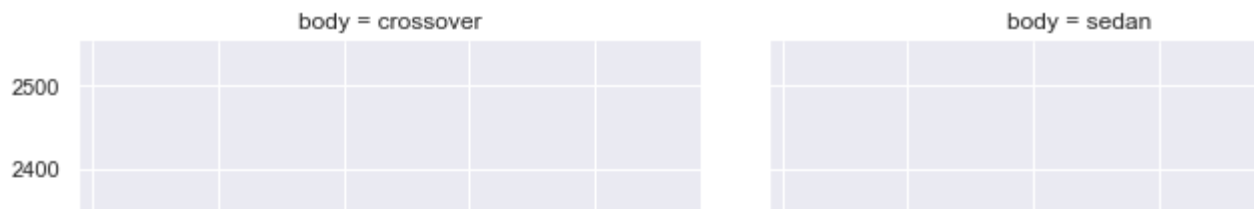
```
1 # Определим уникальные значения для целевого признака "Тип двигателя"
2 data_new['engType'].unique()
```

 array(['Gas', 'Petrol', 'Diesel', 'Other'], dtype=object)


```
1 import seaborn as sb
2 import matplotlib.pyplot as plt
3 import warnings
4 warnings.filterwarnings('ignore')
```

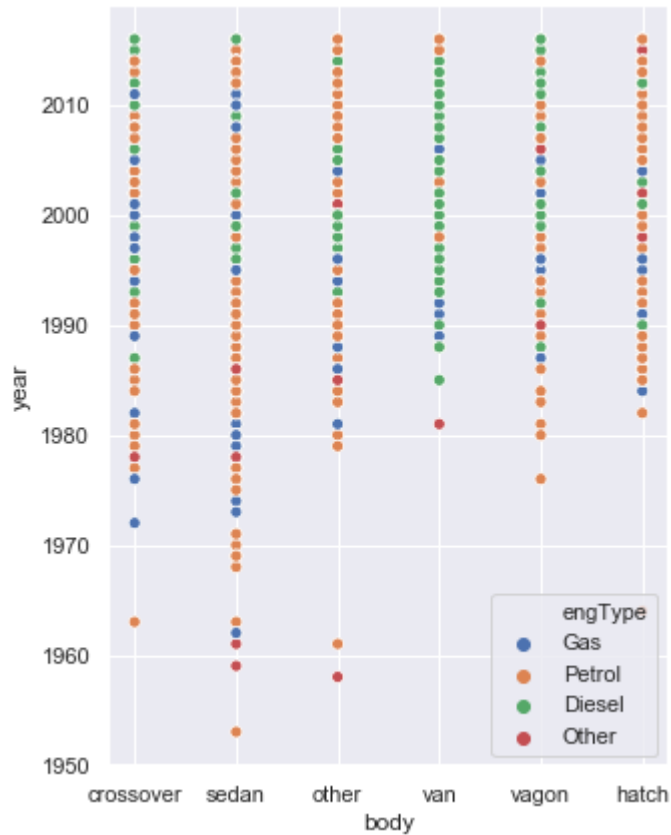
```
1 import seaborn as sns; sns.set(color_codes=True)
2 g = sns.lmplot(x="price", y="year", col="body", hue="body",
3               data=data2, col_wrap=3, height=5)
4
```





```
1 fig, ax = plt.subplots(figsize=(5,7))
2 sb.scatterplot(ax=ax, x='body', y='year', data=data2, hue='engType')
```


 <matplotlib.axes._subplots.AxesSubplot at 0x1a22a5d5c0>

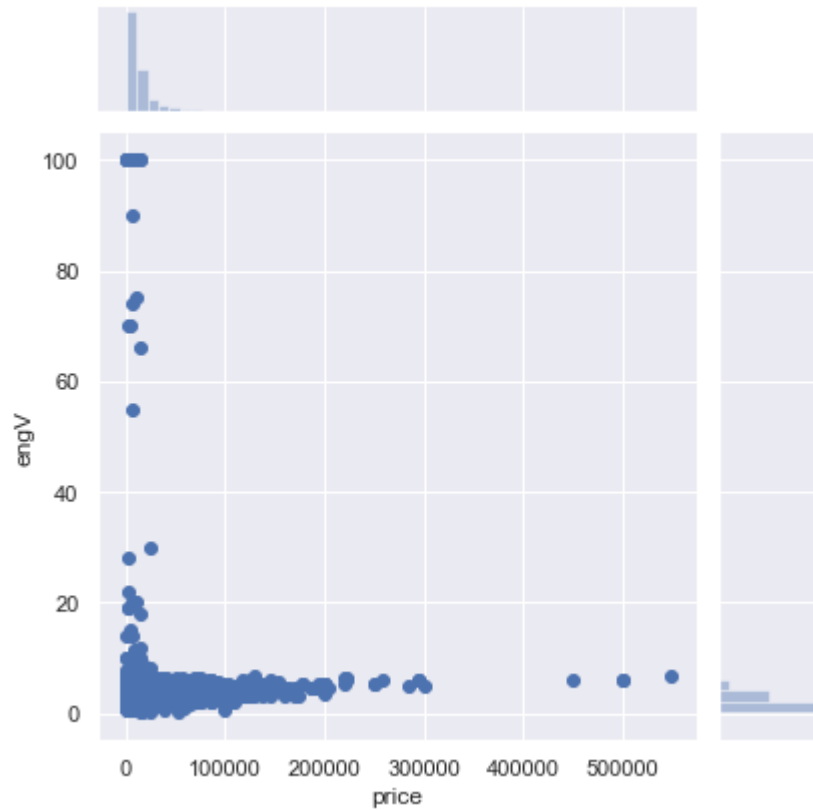


```
1 fig, ax = plt.subplots(figsize=(10,5))
2 sb.distplot(data_new['price'])
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22e278d0>  
1 | sb.jointplot(x='price', y='engV', data=data_new)
```

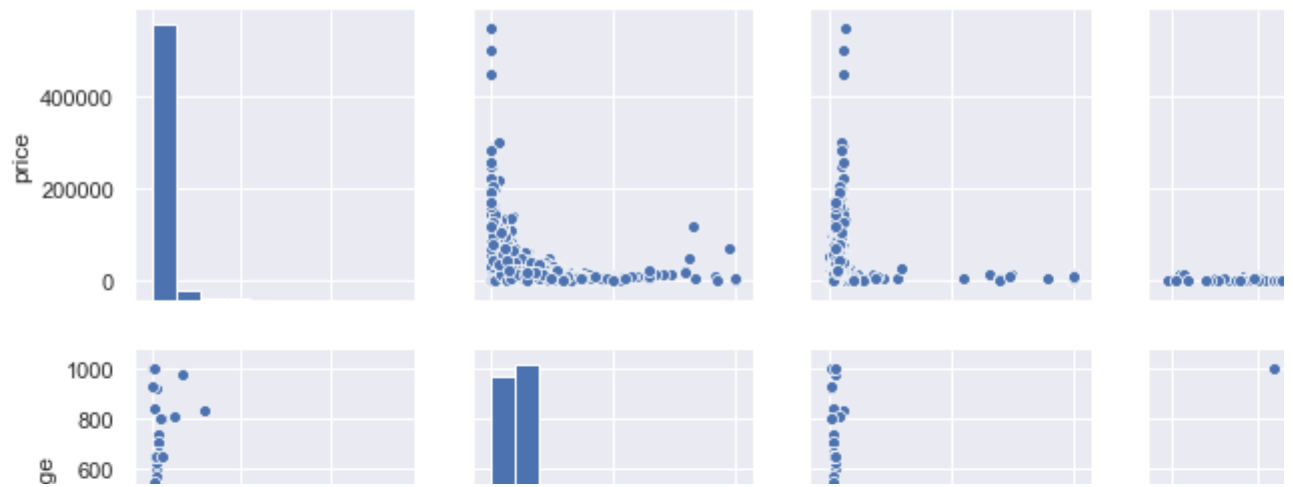
 <seaborn.axisgrid.JointGrid at 0x1a22e344e0>



```
1 | # Больше всего автомобилей до 10 тысяч с объемом двигателя меньше 10 литров  
2 | sb.pairplot(data_new)
```



<seaborn.axisgrid.PairGrid at 0x1a23363c88>



```
1 | sb.pairplot(data_new, hue="body")
```



<seaborn.axisgrid.PairGrid at 0x11f38ee10>

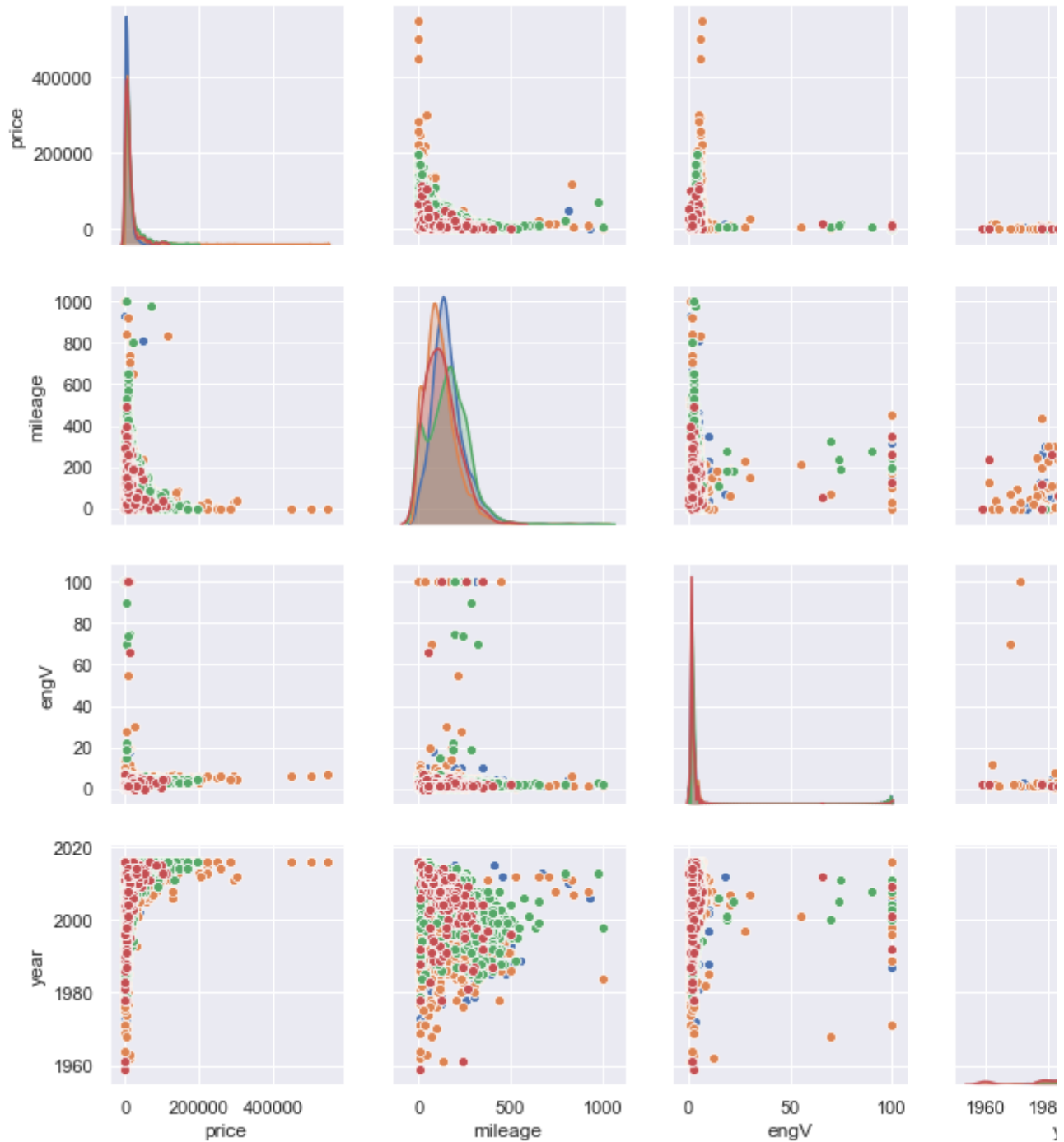
```
1 # Среди автомобилей преобладают хэтчбэки
```



```
1 sb.pairplot(data_new, hue="engType")
```



<seaborn.axisgrid.PairGrid at 0x1a249e0e80>

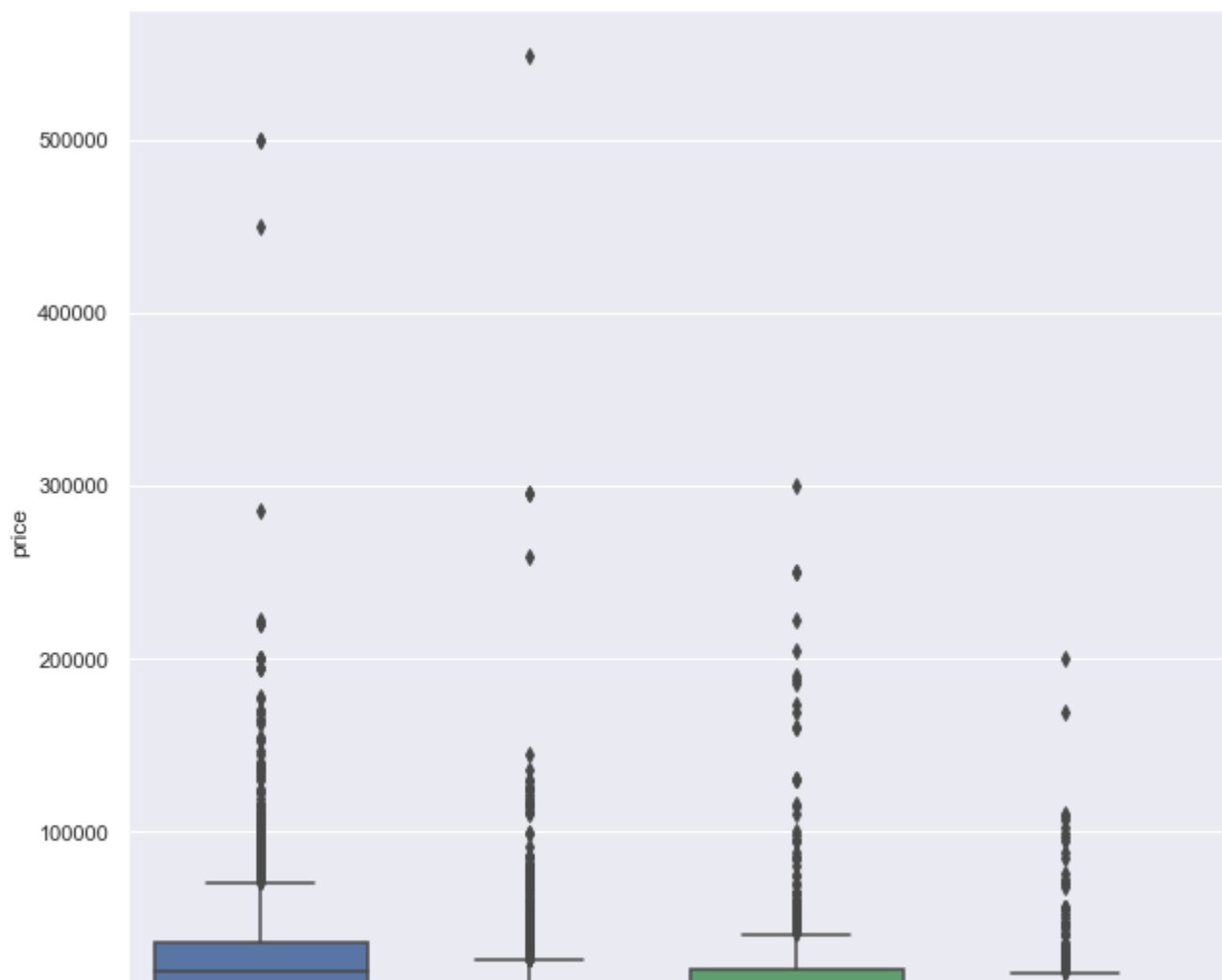


```
1 # Видно, что сейчас преимущественно используются в качестве топлива- бензин и дизель, газ испо
```

```
1 fig, ax = plt.subplots(1, 1, figsize=(15,10))
2 sb.boxplot(x='body', y='price', data=data_new)
```



<matplotlib.axes._subplots.AxesSubplot at 0x1a25644be0>



```
1 data.corr()
```

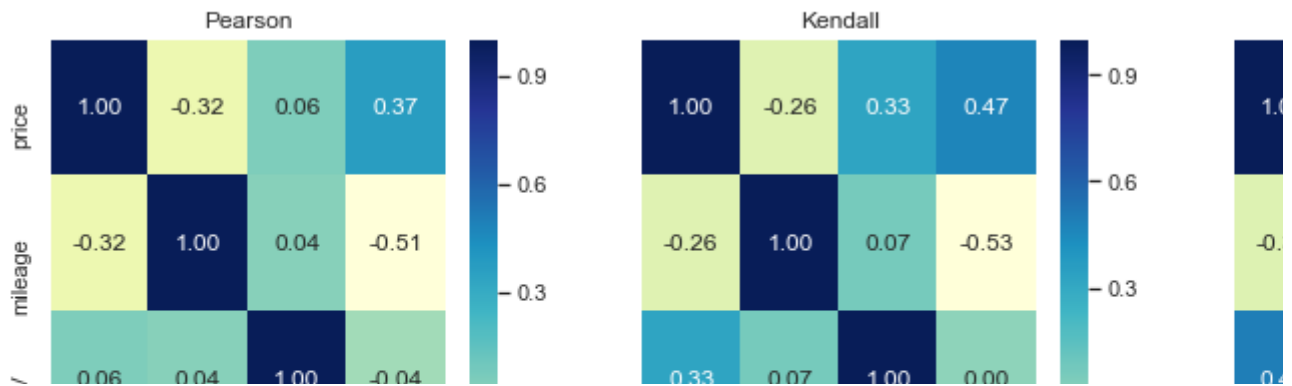


	price	mileage	engV	year
price	1.000000	-0.312415	0.051070	0.370379
mileage	-0.312415	1.000000	0.047070	-0.495599
engV	0.051070	0.047070	1.000000	-0.042251
year	0.370379	-0.495599	-0.042251	1.000000

```
1 fig, ax = plt.subplots(1, 3, sharex='col', sharey='row', figsize=(15,5))
2 sb.heatmap(data_new.corr(method='pearson'), ax=ax[0], cmap='YlGnBu', annot=True)
3 sb.heatmap(data_new.corr(method='kendall'), ax=ax[1],cmap='YlGnBu', annot=True,
4 sb.heatmap(data_new.corr(method='spearman'), ax=ax[2], cmap='YlGnBu', annot=Tru
5 fig.suptitle('Корреляционные матрицы, построенные различными методами')
6 ax[0].title.set_text('Pearson')
7 ax[1].title.set_text('Kendall')
8 ax[2].title.set_text('Spearman')
```



Корреляционные матрицы, построенные различными методами



3. Выбор признаков, подходящих для построения моделей.

Кодирование категориальных признаков.

↳ 3 cells hidden

Корреляционная матрица

↳ 2 cells hidden

Масштабирование данных.

↳ 3 cells hidden

Удаление лишних данных

↳ 6 cells hidden

5 Выбор метрик

<https://habr.com/ru/company/ods/blog/328372/>

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота). Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision — способность отличать этот класс от других классов. Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. Будем использовать F-мера — среднее гармоническое precision и recall. Выбранные метрики:

1. Precision
2. recall
3. F-мера

► 6. Выбор моделей для задачи классификации

1. SGDClassifier - стохастический градиентный спуск.
2. DecisionTreeClassifier - дерево решений.
3. RandomForestClassifier - случайный лес.

↳ 1 cell hidden

► 7. Формирование обучающей и тестовой выборки

↳ 3 cells hidden

► Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров

Стохастический градиентный спуск

↳ 9 cells hidden

<https://www.kaggle.com/iabhishekofficial/mobile-price-classification/downloads/mobile-price-classification.zip/1>

▼ Колонки датасета

1. battery_power- общая энергия, которую аккумулятор может хранить в mAh
2. blueHas - наличие bluetooth
3. clock_speed - тактовая частота, с которой микропроцессор выполняет инструкции
4. dual_sim - поддержка 2 сим-карт
5. fcFront Camera mega pixels - камера, мегапиксели
6. four_gHas 4G or not - наличие 4G
7. int_memoryInternal Memory in Gigabytes - внутренняя память
8. m_dep - толщина телефона в см
9. mobile_wt - вес телефона, г
10. n_cores- Количество ядер процессора
11. pc - Основная камера, мегапиксели
12. px_height- Пиксельное разрешение (высота)
13. px_width- Пиксельное разрешение(ширина)
14. ram- Оперативная память в мегабайтах
15. sc_h- Высота экрана мобильного в см
16. sc_w- Ширина экрана мобильного в см
17. talk_time - Максимальное время заряда аккумулятора
18. three_g - поддержка 3G
19. touch_screen- наличие сенсорного экрана
20. wifi- наличие wifi
21. price_range - это целевая переменная со значениями 0 (низкая стоимость), 1 (средняя стоимость), 2 (высокая стоимость) и 3 (очень высокая стоимость).

```
1 phone = pd.read_csv('Data/HW/phones.csv', sep=',', encoding='latin-1')
2 phone.head(2)
```



	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep
0	842	0	2.2	0	1	0	7	0.6
1	1021	1	0.5	1	0	1	53	0.7

2 rows x 21 columns

```
1 | phone.shape
```



(3000, 21)

```
1 | phone.dtypes
```



```
battery_power      int64
blue               int64
clock_speed        float64
dual_sim           int64
fc                int64
four_g            int64
int_memory         int64
m_dep             float64
mobile_wt         int64
n_cores           int64
pc               int64
px_height         int64
px_width         int64
ram              int64
sc_h             int64
sc_w             int64
talk_time        int64
three_g          int64
touch_screen     int64
wifi            int64
price_range      float64
dtype: object
```

```
1 | # Проверка на пустые значения
2 | phone.isnull().sum()
```



```
battery_power      0
blue               0
clock_speed        0
dual_sim           0
fc                 0
```

```
1 #Пустые значения составляют примерно 5 процентов, поэтому не будем их удалять
2 # Удаление строк, содержащих пустые значения
3 phone_new = phone.dropna(axis=0, how='any')
4 (phone_new.shape)
```

 (2000, 21)

```
1 phone_new= phone
  --_--
```

► Корреляционная матрица

↳ 2 cells hidden

► Графики

↳ 3 cells hidden

5 Выбор метрик

<https://habr.com/ru/company/ods/blog/328372/>

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота). Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision — способность отличать этот класс от других классов. Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. Будем использовать F-мера — среднее гармоническое precision и recall. Выбранные метрики:

1. Precision
2. recall
3. F-мера

6. Выбор моделей для задачи классификации

SVM - Машина опорных векторов
SGDClassifier - стохастический градиентный спуск.
DecisionTreeClassifier - дерево решений. RandomForestClassifier - случайный лес.

► 7. Формирование обучающей и тестовой выборки

↳ 4 cells hidden

► **MACHINE VECTOR(SVM) ALGORITHM**

↳ 1 cell hidden

► **Подбор параметров**

↳ 4 cells hidden

► **SGDClassifier**

↳ 1 cell hidden

► **RandomForestClassifier**

↳ 1 cell hidden

► **DecisionTreeClassifier**

↳ 2 cells hidden

► **Подбор гиперпараметров для RandomForestClassifier**

↳ 8 cells hidden

► **Подбор гиперпараметров для DecisionTreeClassifier**

↳ 4 cells hidden

► **Сравнение моделей после подбора гиперпараметров для DecisionTreeClassifier**

↳ 4 cells hidden

► **Подбор гиперпараметров для SGDClassifier**

↳ 5 cells hidden

► **Сравнение моделей после подбора гиперпараметров для SGDClassifier**

↳ 4 cells hidden

Вывод:

наибольшую точность показала машина опорных векторов при изначальном исследовании и после подбора гиперпараметров. Остальные модели показали приемлемые результаты, но ниже примерно на 10 процентов в большинстве случаев.

2. SGD: f1 - 66.8
3. RAndom Forest: f1 - 85.5
4. Decision Tree: f1 - 84

Таким образом, лучше всего подходит машина опорных векторов и случайный лес. При подборе гиперпараметров все модели показали прирост точности.