# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Курсовой проект по курсу «Методы, средства и технологии мультимедиа»**


Студент: К. О. Вахрамян
Преподаватель: Б. В. Вишняков
Группа: М8О-406Б

Дата:
Оценка:
Подпись:


**Москва, 2021**

# 1 Веведение

**Задача:**
Выбрать задачу (классификация или регрессия), датасет и метрику качества. Выбранные данные необходимо визуализировать и проанализировать. После этого выполнить препроцессинг. Затем реализовать алгоритм линейной регрессии, проверить качество обучения, сравнить с моделью из sklearn.

**Описание данных:**
Датасет представляет из себя информацию об автомобилях марки Mercedes-Benz, реализуемых на вторичном рынке в Великобритании. Содержит следующие признаки:

- Модель автомобиля
- Год выпуска
- Цена на вторичном рынке
- Тип КПП
- Пробег
- Тип топлива
- Налог
- Расход топлива
- Объем двигателя

**Вариант:**
Линейная регрессия. Будем предсказывать цену автомобиля на основании остальных признаков.

# 2 Описание

**Линейная регрессия** - это модель зависимости переменной $y$ от переменных $x$ с линейной функцией зависимости.

$$a(x) = \omega_0 + \sum_{j=1}^{d} \omega_j x_j$$

Параметрами модели являются веса или коэффициенты $\omega_j$. Вес $\omega_0$ - свободный коэффициент или сдвиг.
Вход модели - объект с вектром $x$

$$a(x) = \omega_o + \langle \omega, x \rangle$$

$a(x)$ - оценка значения $y$
Чаще всего линейную регрессию обучают с использованием $MSE$.

$$\frac{1}{l} \sum_{i=1}^{l} (\langle \omega, x_i \rangle - y_i)^2 \to min$$

Получаем задачу оптимизации. Аналитическое решение:

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

Однако обращение матриц - сложная операция ($O(n^3)$), более того $X^T X$ может быть вырожденной или плохообусловленной. Эту задачу можно решить численно.

**Метод градиентного спуска:**
В основе метода - свойство антиградиента, он указыват в сторону наискорейшего убывания функции в данной точке.
Пусть $\omega^{(0)}$ - веса при инициализации. Тогда градиентный спуск имеет вид:

$$\omega^{(k)} = \omega^{(k-1)} - \eta \nabla Q(w^{(k-1)}) = \omega^{(k-1)} - \frac{2}{l} \langle X, \omega^{(k-1)} - y \rangle$$

**Регуляризация:**
Градиентный спуск очень общий метод обучения. Модель может переобучиться(качество на новых данных существенно хуже). Можно заметить, что при переобучении модели получаются очень большие коэффициенты при признаках. Будем штрафовать за слишком большую норму весов. Такой подход называется регуляризацией.

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

Регуляризатор для $L_2$ нормы:

$$R(\omega) = \|\omega\|_2^2 = \sum_{i=1}^{d} \omega_i^2$$

Итеративный процесс для градиентного спуска с регуляризацией имеет вид:

$$\omega^{(k)} = \omega^{(k-1)} - \frac{2}{l}\langle X, \omega^{(k-1)} - y\rangle + 2\omega^{(k-1)}$$

# 3 Ход работы

Загружаем датасет:

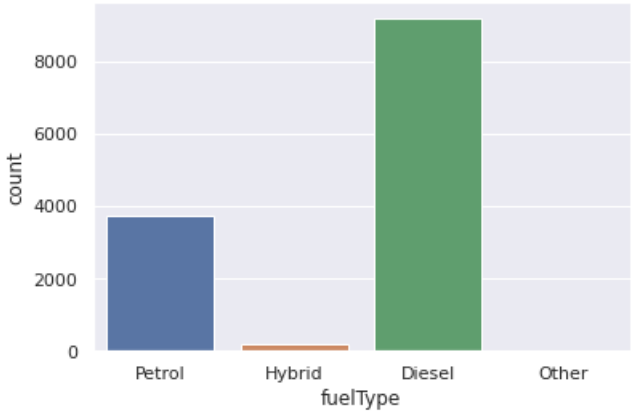
```
(13119, 9)
```

| | model | year | price | transmission | mileage | fuelType | tax | mpg | engineSize |
|---|---|---|---|---|---|---|---|---|---|
| **0** | SLK | 2005 | 5200 | Automatic | 63000 | Petrol | 325 | 32.1 | 1.8 |
| **1** | S Class | 2017 | 34948 | Automatic | 27000 | Hybrid | 20 | 61.4 | 2.1 |
| **2** | SL CLASS | 2016 | 49948 | Automatic | 6200 | Petrol | 555 | 28.0 | 5.5 |
| **3** | G Class | 2016 | 61948 | Automatic | 16000 | Petrol | 325 | 30.4 | 4.0 |
| **4** | G Class | 2016 | 73948 | Automatic | 4000 | Petrol | 325 | 30.1 | 4.0 |

Посмотрим на статистические данные:

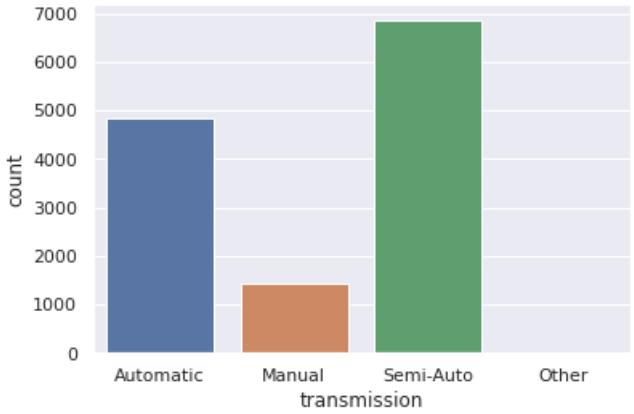
| | year | price | mileage | tax | mpg | engineSize |
|---|---|---|---|---|---|---|
| **count** | 13119.000000 | 13119.000000 | 13119.000000 | 13119.000000 | 13119.000000 | 13119.000000 |
| **mean** | 2017.296288 | 24698.596920 | 21949.559037 | 129.972178 | 55.155843 | 2.071530 |
| **std** | 2.224709 | 11842.675542 | 21176.512267 | 65.260286 | 15.220082 | 0.572426 |
| **min** | 1970.000000 | 650.000000 | 1.000000 | 0.000000 | 1.100000 | 0.000000 |
| **25%** | 2016.000000 | 17450.000000 | 6097.500000 | 125.000000 | 45.600000 | 1.800000 |
| **50%** | 2018.000000 | 22480.000000 | 15189.000000 | 145.000000 | 56.500000 | 2.000000 |
| **75%** | 2019.000000 | 28980.000000 | 31779.500000 | 145.000000 | 64.200000 | 2.100000 |
| **max** | 2020.000000 | 159999.000000 | 259000.000000 | 580.000000 | 217.300000 | 6.200000 |

Распределение числа автомобилей от типа топлива:




Большинство машин используют дизельное топливо.

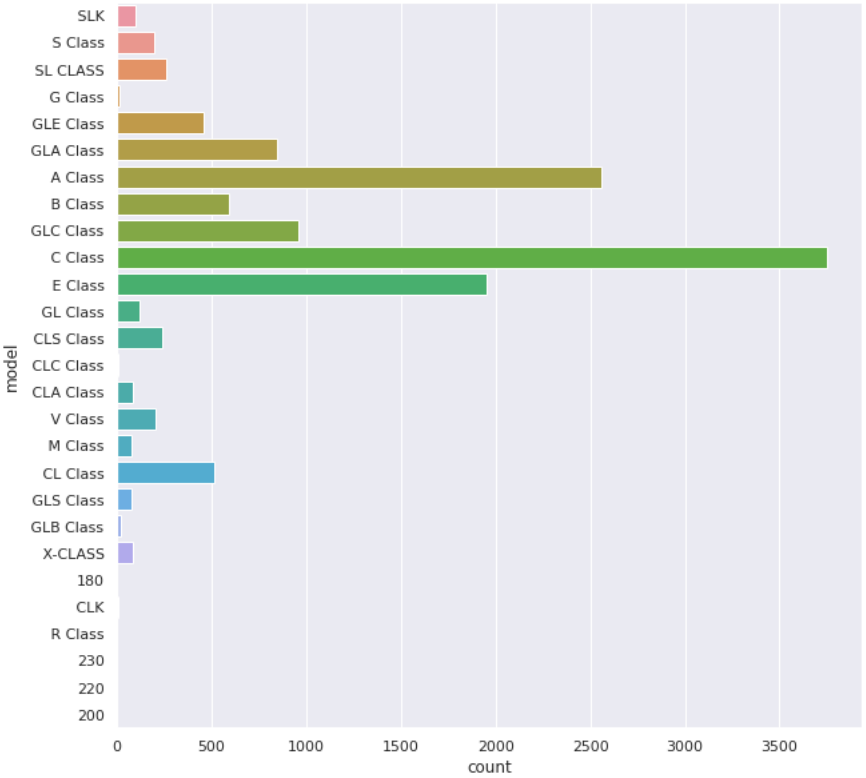
Распределение числа автомобилей от типа КПП:




Большинство машин с автоматической и полуавтоматической КПП.

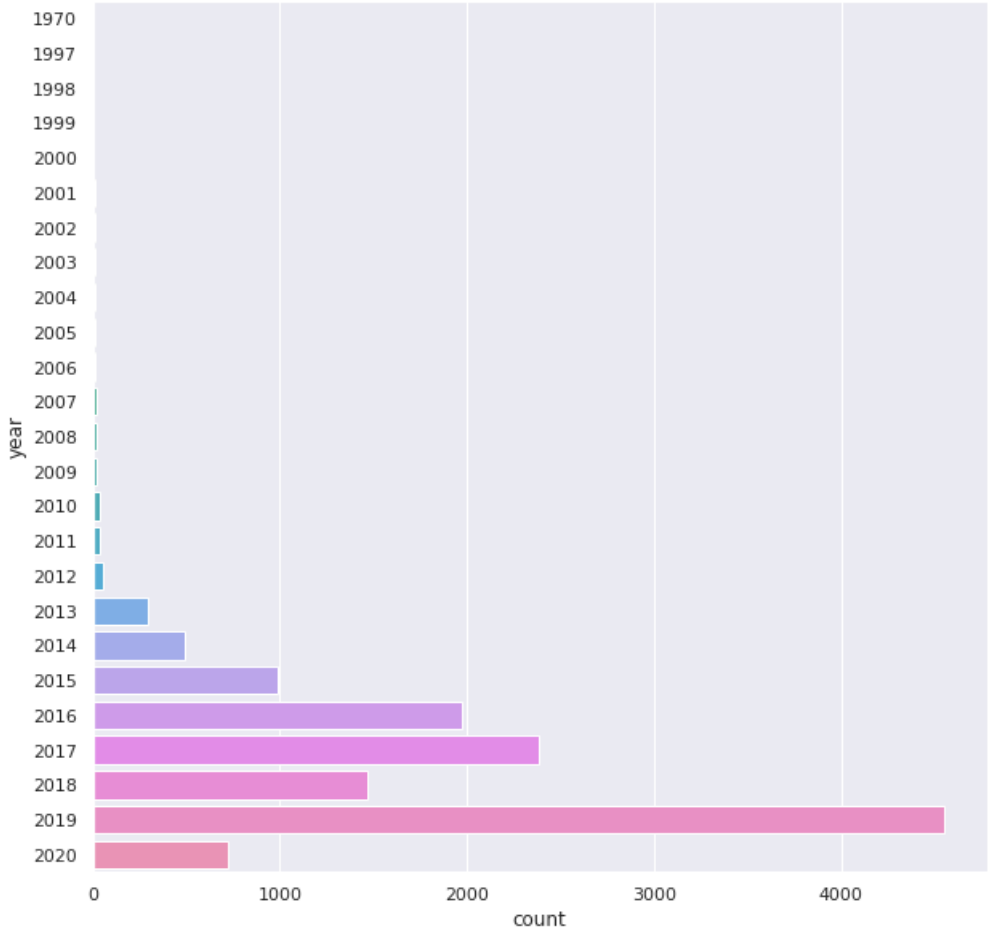
```
Semi-Auto 6848
Automatic 4825
Manual 1444
Other 2
Name: transmission, dtype: int64
```

Постмотрим на распределение моделей автоболией.

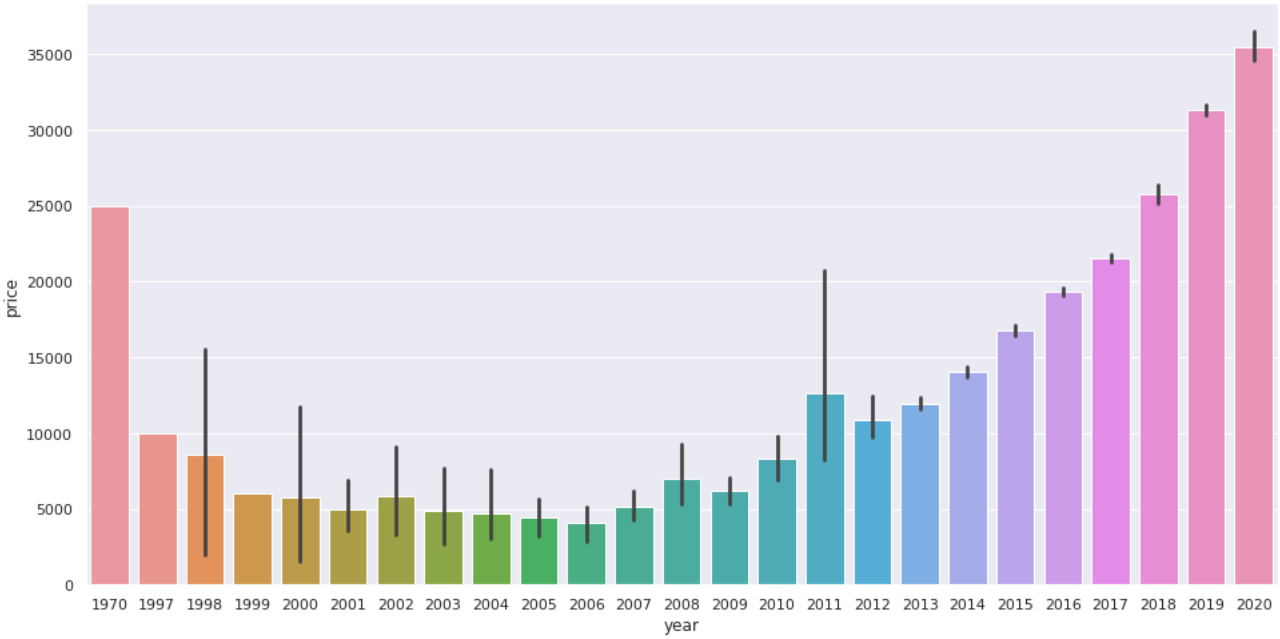



Самые популярные - C,A,E Class.

Также визуализируем распределение года выпуска автомобилей.

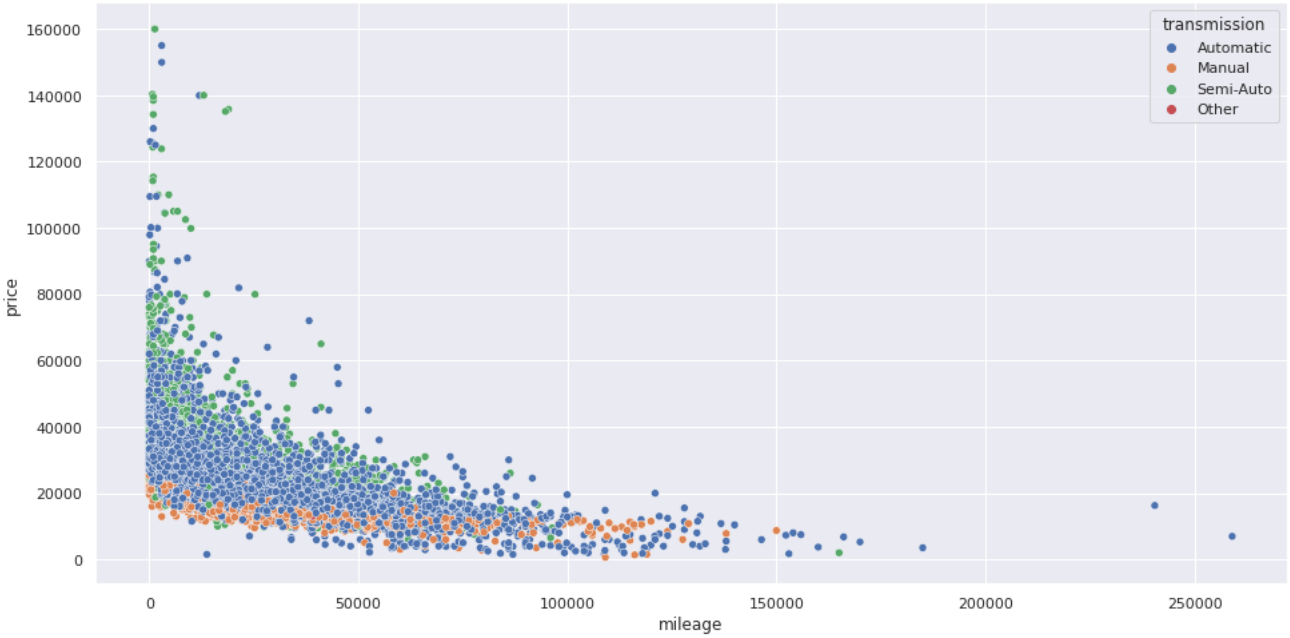



Посмотрим на зависимость цены автомобиля от года выпуска.




Видим, что в среднем, чем новее машина тем дороже. Однако, если идти по временному ряду назад, то выбросы становятся все больше. Это обусловленно тем, что за такой большой период сложно сохранить автомобиль и, чем лучше у него состояние, тем он дороже.

Зависимость цены от пробега при данной КПП:




Чем меньше пробег, тем машина дороже. Также механическая КПП показатель бо-

лее низкой цены, а полуавтоматическая - более высокой.

**Препроцессинг:**
Заменим категориальные признаки бинарными:

```
df_expanded = pd.get_dummies(df)
```

Разделим данные на обучающую и тестовую выборки.

```
X = df_expanded.drop(columns=['price'])
y = df_expanded['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =42)
```

Нормализуем данные:

```
pca = MinMaxScaler()
pca.fit(X_train)
X_train = pca.transform(X_train)
X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pca.transform(X_test)
X_test = pd.DataFrame(X_test, columns=X.columns)
```

**Метрика:**
Метрика R2 принимает значения от 0 до 1, чем ближе значение коэффициента к 1, тем сильнее модель соответствует данным. Эта метрика наиболее наглядна. Кроме того, используется в реализации линейной регрессии в sklearn.

```
def R2(y_actual, y_pred):
    ssr = np.sum((y_pred - y_actual)**2)
    sst = np.sum((y_actual - np.mean(y_actual))**2)
    r2_score = 1 - (ssr/sst)
    return r2_score
```

**Реализация линейной регрессии:**

```
class myLinearRegression:
  def __init__(self, lr=0.05, max_iter = 10000):
    self.max_iter = max_iter
    self.lr = lr
    self.w = np.array([])
    self.firstRun = True

  def fit(self, X, y):
    X = X.to_numpy()
    y = y.to_numpy()
    X = np.hstack((np.full((X.shape[0], 1), 1),X))

    if self.firstRun:
```

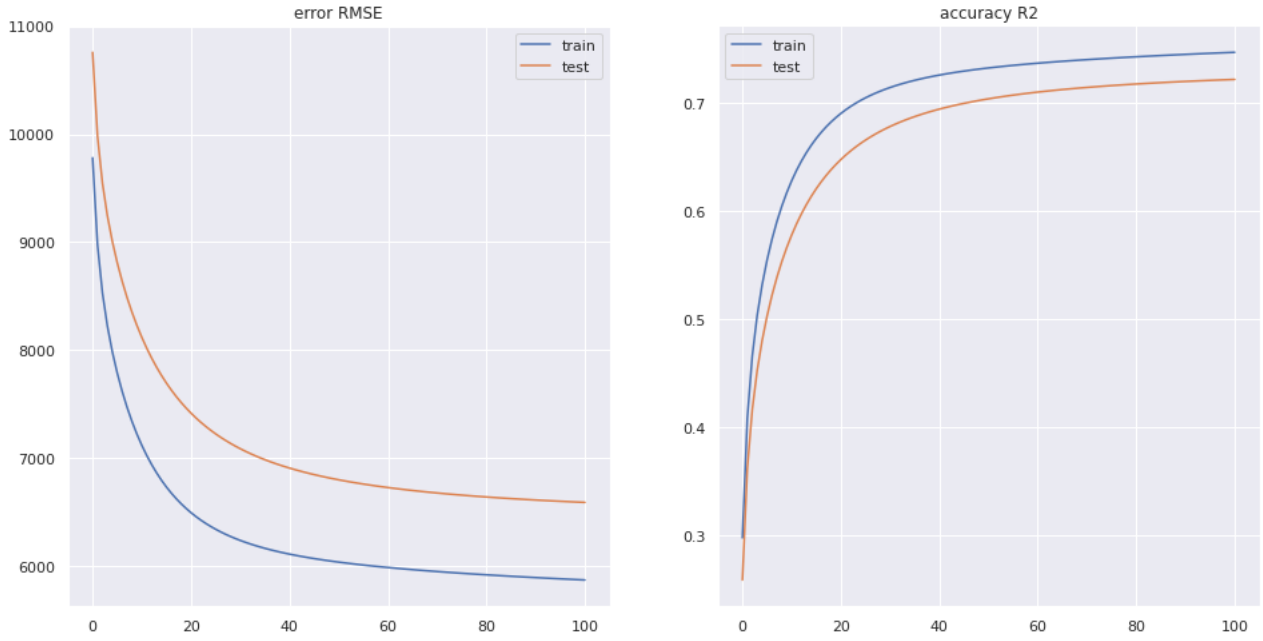
```
      self.w = np.random.rand(X.shape[1])
      self.firstRun = False;
    m = X.shape[0]

    for i in range(self.max_iter):
      y_pred = np.dot(X, self.w)
      diff = y_pred - y
      grad = 2 / m * np.dot(X.T, diff)
      self.w -= self.lr * grad
    return self

  def predict(self, X):
    X = X.to_numpy()
    X = np.hstack((np.full((X.shape[0], 1), 1),X))
    return np.dot(X, self.w)

  def score(self, X, y):
    y = y.to_numpy()
    y_pred = self.predict(X)
    return R2(y, y_pred)
```

**Результаты:**




| | ЛР№4 | КП | sklearn |
|---|---|---|---|
| train score | 0.74667 | 0.75954 | 0.77454 |
| test score | 0.72161 | 0.73486 | 0.74373 |

Лучших оценок удалось достичь благодарю тому, что категориальные признаки я преобразовал в бинарные. В 4 лабораторной было отображение:

```
fuel = {'Petrol':0, 'Hybrid':1, 'Diesel':2, 'Other':3}
df = df.replace({'fuelType' : fuel})
transmission = {'Automatic':0, 'Manual':1, 'Semi-Auto':2, 'Other':3}
df = df.replace({'transmission':transmission})
```

Это не совсем верно, т.к. увеличивает вес тем или иным объектам без определенной причины.

# 4 Выводы

Линейная регрессия - простой, но мощный механизм для аппроксимации линейных зависимостей. Модель является предшественником нелинейных методов, которые используют для обучения нейронных сетей. В ходе курсового проекта удалось улучшить реализацию модели, добавить регуляризацию, и добиться лучших оценок.