УДК 004

Прогнозирование стоимости автомобилей с использованием методов машинного обучения в Google Colab

Анишкова Анастасия Сергеевна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью исследования является прогнозирование стоимости автомобилей с использованием методов машинного обучения в Google Colab. Для реализации использовалась облачная платформа для создания и выполнения кода на Python Google Colab. Полученный результат можно использовать как учебное пособие.
Ключевые слова: прогнозирование стоимости**,** машинное обучение, Google Colab.

Predicting the cost of cars using machine learning methods in Google Colab

Anishkova Anastasia Sergeevna
Sholom Aleichem Priamurskiy State University
Student

Abstract
The purpose of the study is to predict the cost of cars using machine learning methods in Google Colab. For the implementation, Google Colab, a cloud-based platform for creating and executing Python code, was used. The result can be used as a textbook.
Key words: cost forecasting, machine learning, Google Colab.

1 Введение

1.1 Актуальность

Исследование в области прогнозирования стоимости автомобилей с применением методов машинного обучения в Google Colab является актуальным и востребованным как в научном, так и в практическом плане. Оно может привести к созданию более совершенных инструментов для принятия обоснованных решений при покупке, продаже и ценообразовании на автомобильном рынке.

1.2 Обзор исследований

В. Г. Винокурова, С. А. Тимаева продемонстрировали как совершить анализ и прогнозирование продаж ит-оборудования корпоративным заказчикам с помощью методов машинного обучения [1], прогнозирование инвестиционной привлекательности объектов недвижимости на основе

методов машинного обучения с использованием телеграмм-бота совершили А. А. Кириченко, С. А. Тимаева [2], А. А. Архипова применила нейронные сети в задаче прогнозирования финансовых временных рядов [3], разведочный анализ данных о прогнозировании энергопотребления провела А.С. Матвеева[4], А. А. Антонов провел прогнозирование размера заработной платы с использованием методов машинного обучения по обработке естественного языка[5].

2 Цель исследования

Основная цель данного исследования заключается в разработке и оценке эффективности моделей машинного обучения для прогнозирования стоимости автомобилей, используя облачную среду Google Colab.

3 Материалы и методы

В данном исследование используется Google Colab — сервис, созданный Google, который позволяет работать с кодом на языке Python через Jupyter Notebook.

4 Результаты

Первое, что необходимо сделать это импортировать библиотеки и осуществить настройку опций для анализа данных и машинного обучения в Python (см.рис.1).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, StackingRegressor

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

Рисунок 1. Импортирование библиотек и настройка опций

Используем библиотеку Pandas для чтения CSV-файла и создания DataFrame (см.рис.2). Данные можно скачать по ссылке https://cloud.mail.ru/public/47df/UZRJnvK44.

```
df = pd.read_csv('CarPrice_Assignment.csv')
```

Рисунок 2. Загрузка датасета

Просмотрим первые пять строк датасета (см.рис.3).

```
df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495.0
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500.0
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500.0
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950.0
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450.0

5 rows × 26 columns

Рисунок 3. Просмотр датасета

Затем следует загрузить данные в DataFrame и проверить их размер с помощью команды df.shape, чтобы убедиться в корректной загрузке и начать предварительный анализ данных (см.рис4).

```
df.shape
(205, 26)
```

Рисунок 4. Размер данных

Далее необходим код для проверки наличия пропущенных значений в DataFrame и подсчета количества таких значений в каждом столбце (см.рис.5).

```
#проверьте пропущенное значение
df.isnull().sum()
```

```
car_ID                0
symboling             0
CarName               0
fueltype              0
aspiration            0
doornumber            0
carbody               0
drivewheel            0
enginelocation        0
wheelbase             0
carlength             0
carwidth              0
carheight             0
curbweight            0
enginetype            0
cylindernumber        0
enginesize            0
fuelsystem            0
boreratio             0
stroke                0
compressionratio      0
horsepower            0
peakrpm               0
citympg               0
highwaympg            0
price                 0
brand                 0
model                 0
power_to_weight_ratio 0
dtype: int64
```

Рисунок 5. Проверка пропущенных значений

Затем важно проверить наличие повторяющихся значений (см.рис.6).

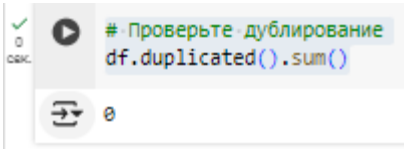


Рисунок 6. Проверка повторяющихся значений

Следующим шагом важно проверить тип данных каждого столбца в вашем DataFrame. Таким образом, вы сможете увидеть, какие типы данных используются для каждого столбца (см.рис.7).

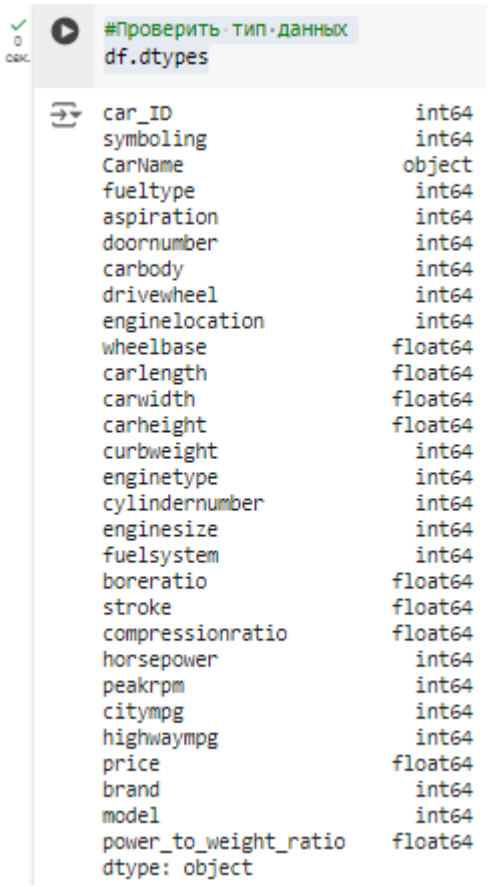


Рисунок 7. Проверка типа данных

Далее необходимо получить количество уникальных значений в каждом столбце вашего DataFrame. Метод nunique() возвращает количество уникальных значений для каждого столбца (см.рис.8).

```
# Проверьте количество уникальных значений в каждом столбце
df.nunique()
```

```
car_ID                205
symboling               6
CarName               147
fueltype                2
aspiration              2
doornumber              2
carbody                 5
drivewheel              3
enginelocation          2
wheelbase              53
carlength              75
carwidth               44
carheight              49
curbweight            171
enginetype              7
cylindernumber          7
enginesize             44
fuelsystem              8
boreratio              38
stroke                 37
compressionratio       32
horsepower             59
peakrpm                23
citympg                29
highwaympg             30
price                 189
brand                  28
model                 142
power_to_weight_ratio 185
dtype: int64
```

Рисунок 8. Проверка уникальных значений

Выведем уникальные значения для каждой категориальной колонки, перечисленной в списке categorical_columns (см.рис.9).

```
categorical_columns = ['fueltype','aspiration','doornumber','carbody','drivewheel','enginelocation','enginetype',
    'cylindernumber',
    'fuelsystem'
]

for col in categorical_columns:

    print(f"Category in {col} is : {df[col].unique()}")
```

```
Category in fueltype is : ['gas' 'diesel']
Category in aspiration is : ['std' 'turbo']
Category in doornumber is : ['two' 'four']
Category in carbody is : ['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
Category in drivewheel is : ['rwd' 'fwd' '4wd']
Category in enginelocation is : ['front' 'rear']
Category in enginetype is : ['dohc' 'ohcv' 'ohc' 'l' 'rotor' 'ohcf' 'dohcv']
Category in cylindernumber is : ['four' 'six' 'five' 'three' 'twelve' 'two' 'eight']
Category in fuelsystem is : ['mpfi' '2bbl' 'mfi' '1bbl' 'spfi' '4bbl' 'idi' 'spdi']
```

Рисунок 9. Категориальные переменные

Построим гистограммы для каждого числового признака, чтобы визуализировать распределение их значений. Это поможет лучше понять диапазоны значений, выявить наличие выбросов и общую структуру данных (см.рис.10).

```
# Distribution of Numerical Features
numerical_features = ['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
                      'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower',
                      'peakrpm', 'citympg', 'highwaympg', 'price']

plt.figure(figsize=(12, 8))
for feature in numerical_features:
    plt.subplot(3, 5, numerical_features.index(feature) + 1)
    sns.histplot(data=df[feature], bins=20, kde=True)
    plt.title(feature)
plt.tight_layout()
plt.show()
```

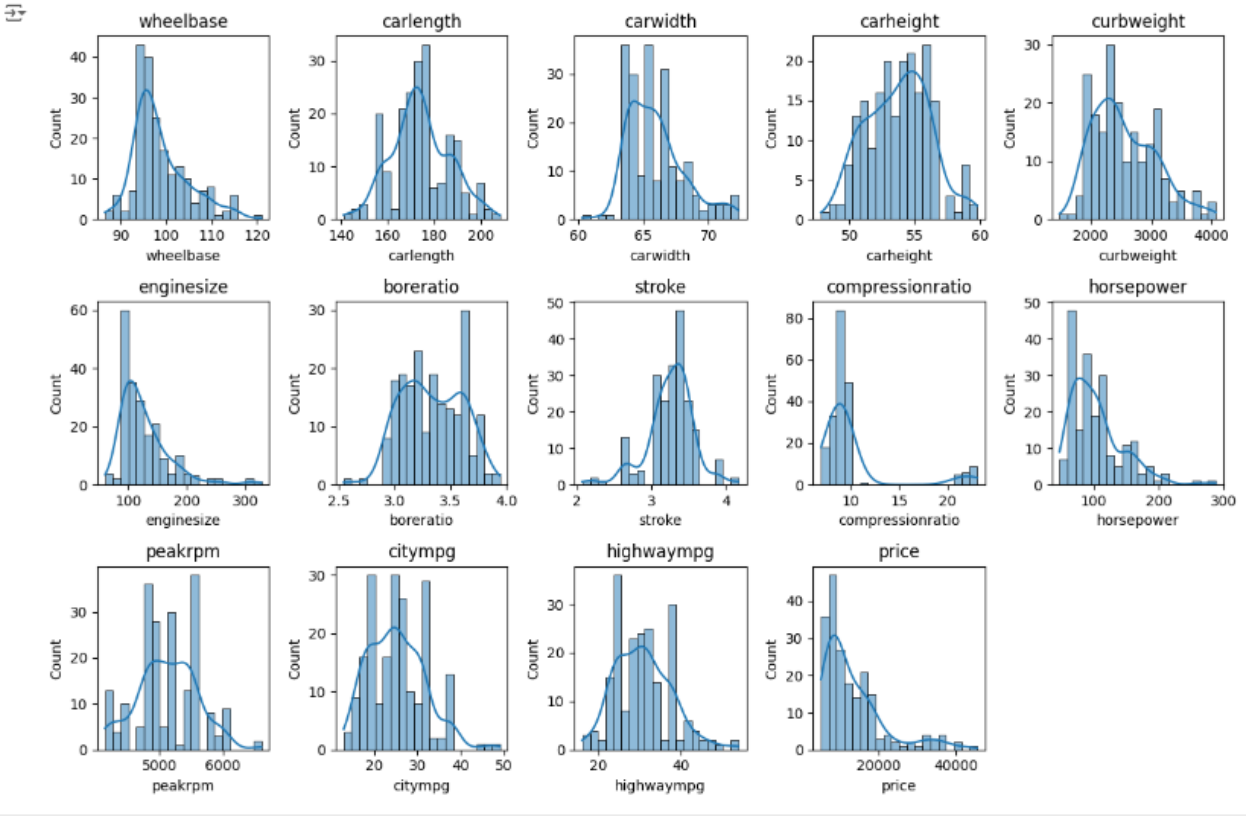


Рисунок 10. Визуализация

Построим графики рассеяния для каждого числового признака в наборе данных, чтобы визуально оценить их взаимосвязь с целевой переменной «price» (см. рис11).

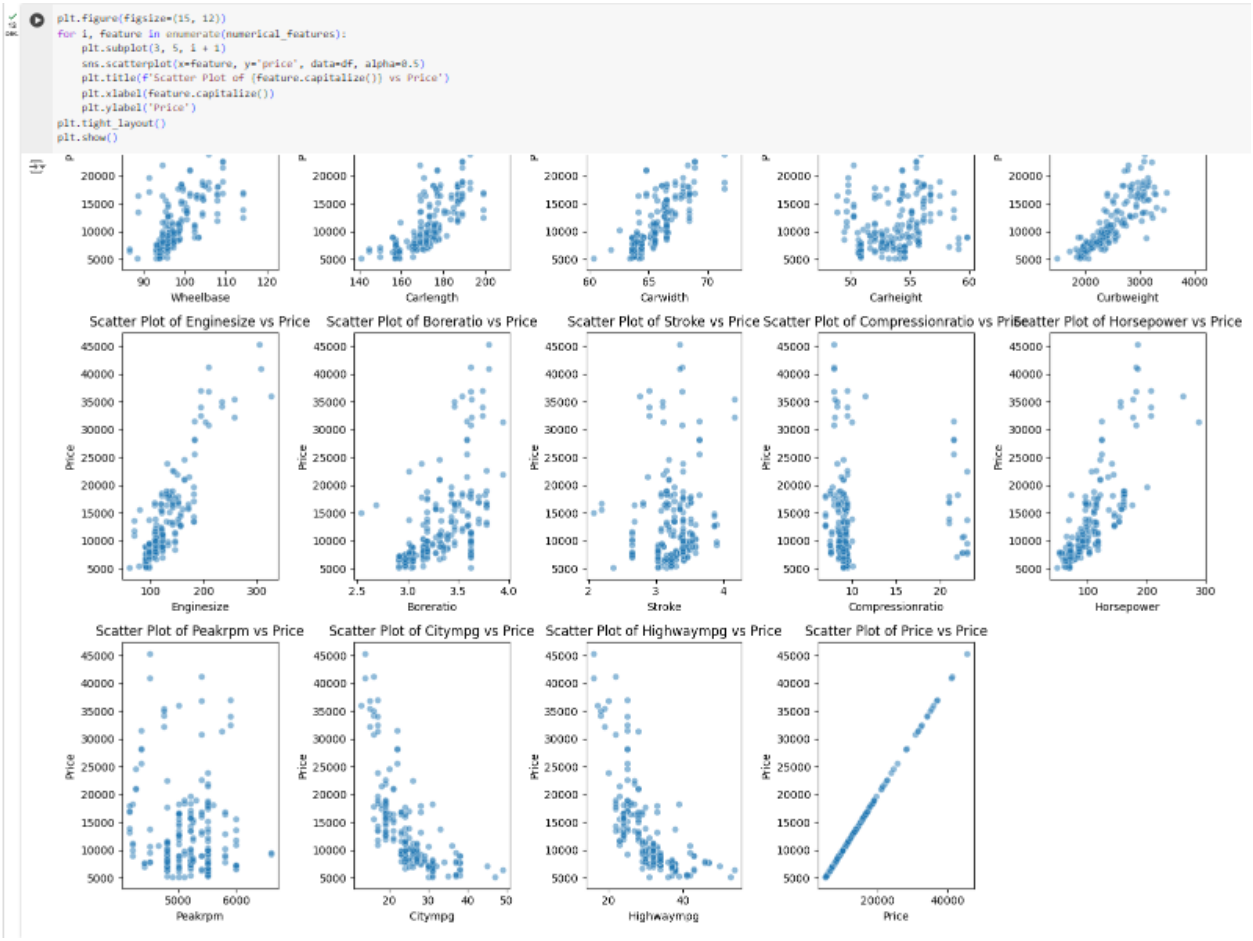


Рисунок 11. Графики рассеивания

Построит гистограмму распределения цен (price) с добавлением сглаженной линии оценки плотности распределения (см.рис.12).

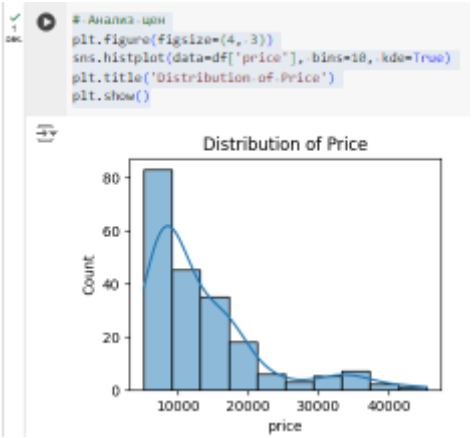


Рисунок 12. Гистограмма распределения

Создадим фигуру с несколькими подзаголовками, каждый из которых содержит гистограмму распределения значений для одного из указанных категориальных столбцов. Каждая гистограмма имеет надписи на столбцах, показывающие количество значений для каждой категории (см.рис.13).

```
# Определите список категориальных столбцов для анализа
categorical_columns = ['fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel',
                       'enginelocation', 'enginetype', 'cylindernumber', 'fuelsystem']

# Вычислите количество строк и столбцов, необходимых для подзаголовков
num_cols = min(len(categorical_columns), 5)
num_rows = (len(categorical_columns) - 1) // num_cols + 1

# Создание подзаголовков
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(12, num_rows * 2.5))
axes = axes.ravel()  # Flatten the 2D array of axes

# Пройдитесь по каждому категориальному столбцу
for i, column in enumerate(categorical_columns):
    if i < len(categorical_columns):
        sns.countplot(x=df[column], data=df, palette='Paired', ax=axes[i], saturation=0.95)
        for container in axes[i].containers:
            axes[i].bar_label(container, color='black', size=10)
        axes[i].set_title(f'Count Plot of {column.capitalize()}')
        axes[i].set_xlabel(column.capitalize())
        axes[i].set_ylabel('Count')
    else:
        axes[i].axis('off')

# Отрегулируйте макет и покажите графики
plt.tight_layout()
plt.show()
```

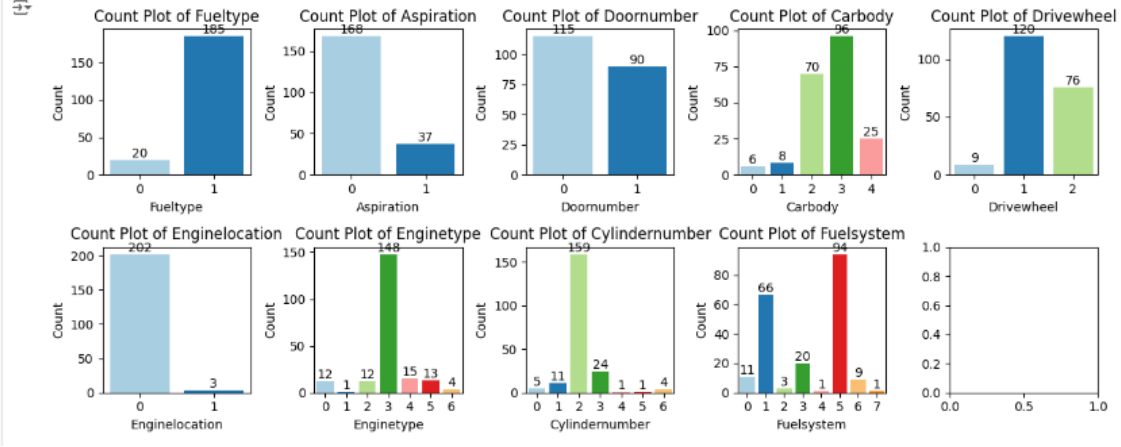


Рисунок 13. Гистограммы

Изобразим столбчатую диаграмму, на которой отображаются топ-20 автомобильных моделей по частоте их появления в датасете. Диаграмма будет иметь заголовок, метки осей и автоматически отрегулированный макет (см.рис.14).

```
n = 20  # Количество лучших моделей автомобилей для построения графика
top_car_models = df['CarName'].value_counts().head(n)

plt.figure(figsize=(6, 4))
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Frequency')
plt.xlabel('Frequency')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```

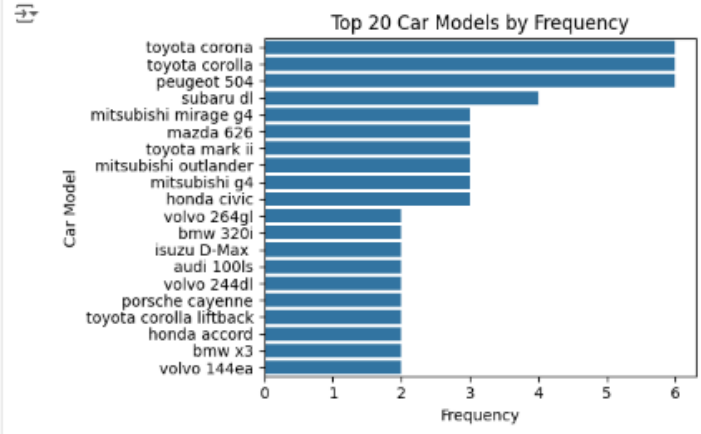
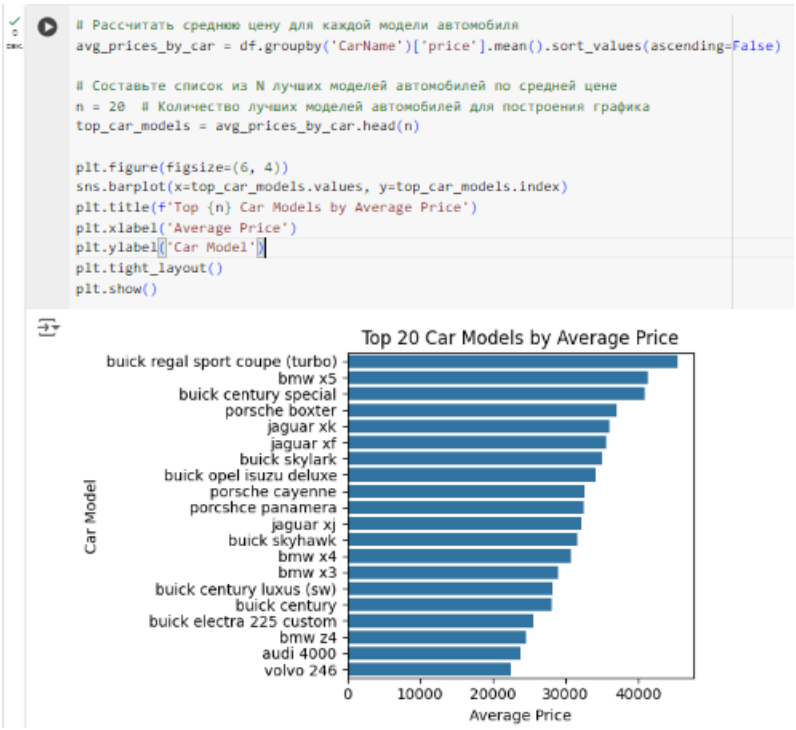


Рисунок 14. Столбчатая диаграмма

Выведем столбчатую диаграмму, которая отображает топ-20 автомобильных моделей (CarName) из датасета по средней цене (см.рис.15).




```
# Рассчитать среднюю цену для каждой модели автомобиля
avg_prices_by_car = df.groupby('CarName')['price'].mean().sort_values(ascending=False)

# Составьте список из N лучших моделей автомобилей по средней цене
n = 20  # Количество лучших моделей автомобилей для построения графика
top_car_models = avg_prices_by_car.head(n)

plt.figure(figsize=(6, 4))
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Average Price')
plt.xlabel('Average Price')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```

Рисунок 15. Столбчатая диаграмма

Создадим серию подзаголовков (subplots) с диаграммами «ящик с усами» (box plots), которые отображают зависимость цены от каждой из указанных категориальных характеристик в датасете (см.рис.16).

```
# Категориальная характеристика в зависимости от цены
plt.figure(figsize=(12, 8))
for feature in categorical_columns:
    plt.subplot(3, 3, categorical_columns.index(feature) + 1)
    sns.boxplot(data=df, x=feature, y='price')
    plt.title(f'{feature} vs. Price')
plt.tight_layout()
plt.show()
```

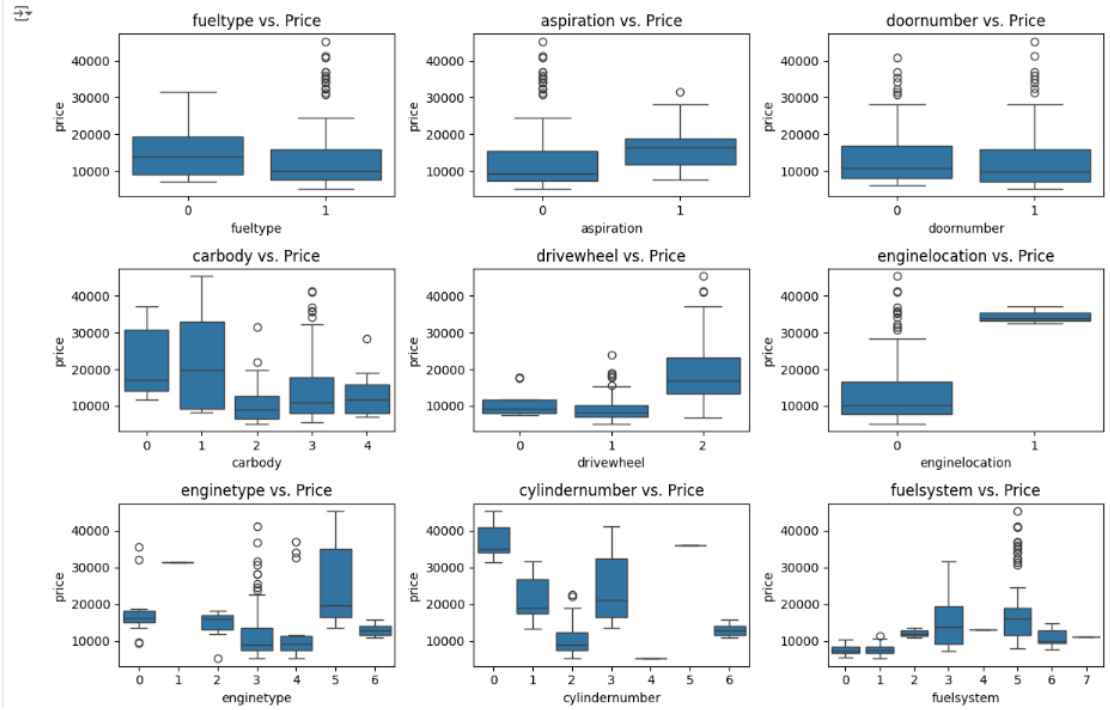


Рисунок 16. Диаграммы

Выведем тепловую карту (heatmap) корреляционной матрицы для числовых признаков (features) в датасете (см.рис.17).

```
# Корреляционный анализ
correlation_matrix = df[numerical_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

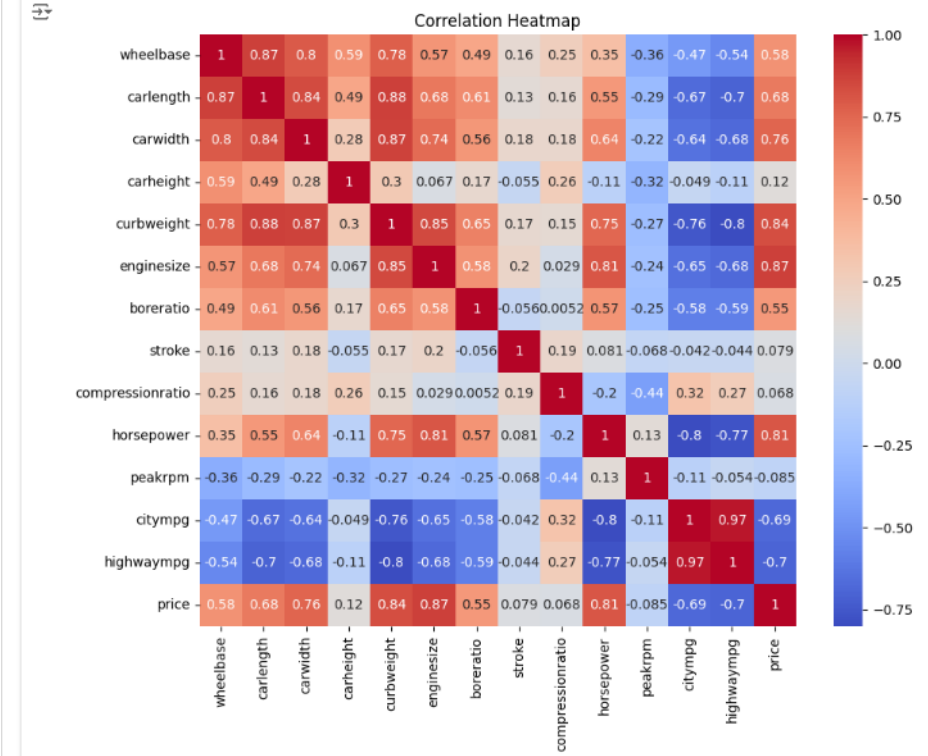


Рисунок 17. Корреляционная матрица

Разобьем названия автомобилей на марки и модели, затем кодирует категориальные переменные с помощью библиотеки sklearn LabelEncoder. Также создадим новый столбец "power_to_weight_ratio", который является отношением мощности к весу автомобиля. После выполнения этих действий выведем первые несколько строк датасета, включая новые столбцы «brand» и «model» (см.рис.18).

```
# Извлеките марку и модель из названия автомобиля
df['brand'] = df['CarName'].apply(lambda x: x.split(' ')[0])
df['model'] = df['CarName'].apply(lambda x: ' '.join(x.split(' ')[1:]))

# Определите категориальные и числовые столбцы
categorical_columns = ['fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel',
                       'enginelocation', 'enginetype', 'cylindernumber', 'fuelsystem', 'brand', 'model']
numerical_columns = ['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
                     'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower',
                     'peakrpm', 'citympg', 'highwaympg']

# Кодирование категориальных переменных
label_encoder = LabelEncoder()
for column in categorical_columns:
    df[column] = label_encoder.fit_transform(df[column])

# Разработка функциональных возможностей
df['power_to_weight_ratio'] = df['horsepower'] / df['curbweight']

df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price	brand	model	power_to_weight_ratio
0	1	3	alfa-romero giulia	1	0	1	0	2	0	88.6	...	2.68	9.0	111	5000	21	27	13495.0	1	79	0.043564
1	2	3	alfa-romero stelvio	1	0	1	0	2	0	88.6	...	2.68	9.0	111	5000	21	27	16500.0	1	123	0.043564
2	3	1	alfa-romero Quadrifoglio	1	0	1	2	2	0	94.5	...	3.47	9.0	154	5000	19	26	16500.0	1	29	0.054552
3	4	2	audi 100 ls	1	0	0	3	1	0	99.8	...	3.40	10.0	102	5500	24	30	13950.0	2	1	0.043646
4	5	2	audi 100s	1	0	0	3	0	0	99.4	...	3.40	8.0	115	5500	18	22	17450.0	2	2	0.040722

Рисунок 18. Вывод первых строк датасета

Создадим два набора данных для обучения (X_train) и тестирования (X_test), а также соответствующие им целевые переменные (y_train и y_test).

Далее применим стандартную масштабировку к обоим наборам данных (X_train и X_test), используя одну и ту же функцию масштабирования. Это означает, что все значения в каждом столбце будут нормированы так, чтобы их среднее значение было равно нулю, а стандартное отклонение - единице. Результатом выполнения этих действий будет три объекта: X_train, X_test и scaler. X_train и X_test теперь содержат данные, которые были нормализованы, а scaler — это объект, который может использоваться для нормализации новых данных (см.рис.19).

```
# Разделение набора данных
X = df.drop(['price', 'CarName','car_ID'], axis=1)  # Укажите технические характеристики и название машины
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Масштабирование объектов
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Рисунок 19. Разделение набора данных и масштабирование объектов

Сформируем и адаптируем простую модель линейной регрессии в качестве базовой линии, сделаем прогнозы на основе данных обучения и тестирования, а затем вычисляем R-квадрат (коэффициент детерминации) и среднеквадратичную ошибку (MSE) для оценки качества модели (см.рис.20).

```
# Создайте и адаптируйте простую модель линейной регрессии в качестве базовой линии
simple_linear_model = LinearRegression()
simple_linear_model.fit(X_train, y_train)

# Делайте прогнозы на основе данных обучения и тестирования.
y_train_pred_slr = simple_linear_model.predict(X_train)
y_test_pred_slr = simple_linear_model.predict(X_test)

# Вычислите R-квадрат и среднеквадратичную ошибку для оценки

slr_r2_train = r2_score(y_train, y_train_pred_slr)
slr_r2_test = r2_score(y_test, y_test_pred_slr)
slr_mse_train = mean_squared_error(y_train, y_train_pred_slr)
slr_mse_test = mean_squared_error(y_test, y_test_pred_slr)


print(f"Training R-squared: {slr_r2_train:.4f}, Training MSE: {slr_mse_train:.4f}")
print(f"Testing R-squared: {slr_r2_test:.4f}, Testing MSE: {slr_mse_test:.4f}")
```

```
Training R-squared: 0.9156, Training MSE: 5035232.8171
Testing R-squared: 0.8498, Testing MSE: 11856704.1656
```

Рисунок 20. Вычисление R-квадрат и среднеквадратичную ошибку

Создадим модели полиномиальной регрессии с разными степенями и оценит их производительность на обучающих и тестовых данных. В конце он найдет наилучшую степень, основываясь на результатах тестирования в R-квадрате (см.рис.21).

```
# Продолжайте со степенью 5
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
poly_features = PolynomialFeatures(degree=5)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
```

Рисунок 21. Модели полиномиальной регрессии

Исследуем влияние различных значений регуляризационного параметра (альфа) на модель линейной регрессии с регуляризацией (см.рис.22).

```
# Сильные стороны регуляризации (альфа-значения)
alphas = [0.001, 0.01, 0.1, 1, 10]
```

Рисунок 22. Регуляризационный параметр

Создадим и настраиваем три типа регрессионных моделей с регуляризацией: Ridge, Lasso и ElasticNet (см.рис.23).

```
# Создайте и подгоните регрессионные модели Ridge, Lasso и ElasticNet с различными значениями альфа-сигнала
ridge_models = []
ridge_r2_train_scores = []
ridge_r2_test_scores = []

lasso_models = []
lasso_r2_train_scores = []
lasso_r2_test_scores = []

lasso_mse_train = []
lasso_mse_test = []

elasticnet_models = []
elasticnet_r2_train_scores = []
elasticnet_r2_test_scores = []

for alpha in alphas:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train_poly, y_train)
    ridge_models.append(ridge_model)
```

Рисунок 23. Регрессионные модели

Сделаем прогнозы на основе данных обучения и тестирования для модели линейной регрессии с регуляризацией Ridge (см.рис.24).

```
# Делайте прогнозы на основе данных обучения и тестирования.
y_train_pred_ridge = ridge_model.predict(X_train_poly)
y_test_pred_ridge = ridge_model.predict(X_test_poly)
```

Рисунок 24. Прогнозы и тестирование

Вычислим коэффициент детерминации (R-квадрат) для модели линейной регрессии с регуляризацией Ridge, а затем повторяет аналогичный процесс для модели с регуляризацией Lasso (см.рис.25)

```
# Вычислить R-квадрат для оценки
ridge_r2_train = r2_score(y_train, y_train_pred_ridge)
ridge_r2_test = r2_score(y_test, y_test_pred_ridge)

ridge_r2_train_scores.append(ridge_r2_train)
ridge_r2_test_scores.append(ridge_r2_test)

lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X_train_poly, y_train)
lasso_models.append(lasso_model)
```

Рисунок 25. Вычисления

Сделаем прогнозы на основе данных обучения и тестирования для модели линейной регрессии с регуляризацией Lasso (см.рис.26).

```
# Делайте прогнозы на основе данных обучения и тестирования.
y_train_pred_lasso = lasso_model.predict(X_train_poly)
y_test_pred_lasso= lasso_model.predict(X_test_poly)
```

Рисунок 26. Прогнозы и тестирование

Вычислим коэффициент детерминации (R-квадрат) и среднеквадратичную ошибку (MSE) для модели линейной регрессии с регуляризацией Lasso, а затем тренирует модели ElasticNet с различными значениями параметра l1_ratio (см.рис.27).

```
# Вычислить R-квадрат для оценки
lasso_r2_train = r2_score(y_train, y_train_pred_lasso)
lasso_r2_test = r2_score(y_test, y_test_pred_lasso)

lasso_r2_train_scores.append(lasso_r2_train)
lasso_r2_test_scores.append(lasso_r2_test)
lasso_mse_train_1 = mean_squared_error(y_train, y_train_pred_lasso)
lasso_mse_test_1 = mean_squared_error(y_test, y_test_pred_lasso)
lasso_mse_train.append(lasso_mse_train_1)  # Удалите лишнюю строку, если она не нужна

# Тренируйте модели ElasticNet с различным соотношением l1_ratio
for l1_ratio in [0.2, 0.5, 0.8]:
    elasticnet_model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)
    elasticnet_model.fit(X_train_poly, y_train)
    elasticnet_models.append(elasticnet_model)
```

Рисунок 27. Вычисления и тренировка модели

Этот код делает прогнозы на основе данных обучения и тестирования для модели линейной регрессии с регуляризацией ElasticNet, а затем вычисляет коэффициент детерминации (R-квадрат) для оценки производительности модели (см.рис.28).

```
# Делайте прогнозы на основе данных обучения и тестирования.
y_train_pred = elasticnet_model.predict(X_train_poly)
y_test_pred = elasticnet_model.predict(X_test_poly)

# Вычислить R-квадрат для оценки
elasticnet_r2_train = r2_score(y_train, y_train_pred)
elasticnet_r2_test = r2_score(y_test, y_test_pred)

elasticnet_r2_train_scores.append(elasticnet_r2_train)
elasticnet_r2_test_scores.append(elasticnet_r2_test)
```

Рисунок 28. Прогнозы и вычисления

Найдем лучшие значения регуляризационного параметра альфа для моделей Ridge, Lasso и ElasticNet, основываясь на их производительности (R-квадрат) на тестовых данных (см.рис.29).

```
# Найдите наилучшие альфа-значения для Ridge, Lasso и ElasticNet на основе результатов тестирования в R-квадрате.
best_alpha_ridge = alphas[np.argmax(ridge_r2_test_scores)]
best_alpha_lasso = alphas[np.argmax(lasso_r2_test_scores)]
best_alpha_elasticnet_index = np.argmax(elasticnet_r2_test_scores)
best_alpha_elasticnet = alphas[np.argmax(np.abs(elasticnet_r2_test_scores))]
best_l1_ratio_elasticnet = [0.2, 0.5, 0.8][np.argmax(elasticnet_r2_test_scores) // len(alphas)]
best_r2train_ridge_i=np.argmax(ridge_r2_train_scores)
best_r2train_lasso_i=np.argmax(lasso_r2_train_scores)
best_r2train_elasticnet_i=np.argmax(elasticnet_r2_train_scores)
best_r2test_ridge_i=np.argmax(ridge_r2_test_scores)
best_r2test_lasso_i=np.argmax(lasso_r2_test_scores)
best_r2test_elasticnet_i=np.argmax(elasticnet_r2_test_scores)
```

Рисунок 29. Поиск наилучших значений

Найдем индексы, соответствующие минимальным значениям среднеквадратичной ошибки (MSE) на тренировочных и тестовых данных для модели Lasso. Также извлекем лучшие значения коэффициента детерминации (R-квадрат) на тренировочных и тестовых данных для моделей Ridge, Lasso и ElasticNet (см.рис.30).

```
#лучшее_тестирование_lasso_i1=np.аргументов(lasso_mse_train)
#лучшее_тестирование_lasso_i1=np.аргументов(lasso_mse_test)

best_r2train_ridge=ridge_r2_train_scores[best_r2train_ridge_i]
best_r2train_lasso=lasso_r2_train_scores[best_r2train_lasso_i]
best_r2train_elasticnet=elasticnet_r2_train_scores[best_r2train_elasticnet_i]
best_r2test_ridge=ridge_r2_test_scores[best_r2test_ridge_i]
best_r2test_lasso=lasso_r2_test_scores[best_r2test_lasso_i]
best_r2test_elasticnet=elasticnet_r2_test_scores[best_r2test_elasticnet_i]
```

Рисунок 30. Тестирование

Выведем результаты для моделей линейной регрессии с регуляризацией (см.рис.31, 32).

```
#лучший_тренажер_класса_мсе=тренировка_класса_мсе[лучший_тренажер_класса_и1]
#лучший_класс_мсе_трест=лучший_класс_мсе_тест[лучший_класс_и1]

print(f"\nPolynomial Linear Regression with Regularization:")
print(f"Best Alpha (Ridge): {best_alpha_ridge:.4f}")
print(f"Best Alpha (Lasso): {best_alpha_lasso:.4f}")
print(f"Best Alpha (ElasticNet): {best_alpha_elasticnet:.4f}")
print(f"Best l1_ratio (ElasticNet): {best_l1_ratio_elasticnet:.1f}")
print(f"Ridge Training R-squared Scores: {best_r2train_ridge:.4f}")
print(f"Ridge Testing R-squared Scores: {best_r2test_ridge:.4f}")
print(f"Lasso Training R-squared Scores: {best_r2train_lasso:.4f}")
print(f"Lasso Testing R-squared Scores: {best_r2test_lasso:.4f}")
#print(f"Lasso Training MSE Scores: {best_lasso_mse_train:.2f}")
#print(f"Lasso Testing MSE Scores: {best_lasso_mse_test:.2f}")
print(f"ElasticNet Training R-squared Scores: {best_r2train_elasticnet:.4f}")
print(f"ElasticNet Testing R-squared Scores: {best_r2test_elasticnet:.4f}")
```

Рисунок 31. Вывод результатов

```
Polynomial Linear Regression with Regularization:
Best Alpha (Ridge): 0.0010
Best Alpha (Lasso): 0.0010
Best Alpha (ElasticNet): 0.0010
Best l1_ratio (ElasticNet): 0.2
Ridge Training R-squared Scores: 0.9998
Ridge Testing R-squared Scores: -0.2760
Lasso Training R-squared Scores: 0.9978
Lasso Testing R-squared Scores: 0.7207
ElasticNet Training R-squared Scores: 0.9961
ElasticNet Testing R-squared Scores: 0.5175
```

Рисунок 32. Результаты вывода

Основные выводы:

1. Лучшее значение регуляризационного параметра альфа:

- Для всех трех моделей (Ridge, Lasso, ElasticNet) лучшее значение альфа составляет 0.0010.

2. Лучшее значение параметра l1_ratio для модели ElasticNet:

- Лучшее значение l1_ratio для модели ElasticNet составляет 0.2, что означает, что модель использует больше регуляризации L1 (Lasso), чем регуляризации L2 (Ridge).

3. Производительность на тренировочных данных:

- Модель Ridge показывает самый высокий R-квадрат (0.9998) на тренировочных данных, что означает, что она очень хорошо подгоняется к обучающей выборке.

- Модель Lasso также показывает высокий R-квадрат (0.9978) на тренировочных данных.

- Модель ElasticNet имеет несколько более низкий R-квадрат (0.9961) на тренировочных данных по сравнению с моделями Ridge и Lasso.

4. Производительность на тестовых данных:

- Модель Lasso показывает лучший R-квадрат (0.7207) на тестовых данных, что указывает на ее лучшую способность обобщать на новые, неизвестные данные.

- Модель ElasticNet имеет более низкий R-квадрат (0.5175) на тестовых данных по сравнению с моделью Lasso.

- Модель Ridge показывает отрицательный R-квадрат (-0.2760) на тестовых данных, что свидетельствует о плохой способности модели обобщаться.

Основываясь на этих результатах, можно сделать вывод, что модель Lasso с лучшим R-квадратом на тестовых данных является наиболее подходящей регрессионной моделью с регуляризацией для данной задачи. Модель Ridge, несмотря на ее высокую производительность на тренировочных данных.

Создадим список базовых моделей, в котором хранятся модели линейной регрессии с регуляризацией, обученные с использованием лучших найденных значений параметров регуляризации (см.рис.33).

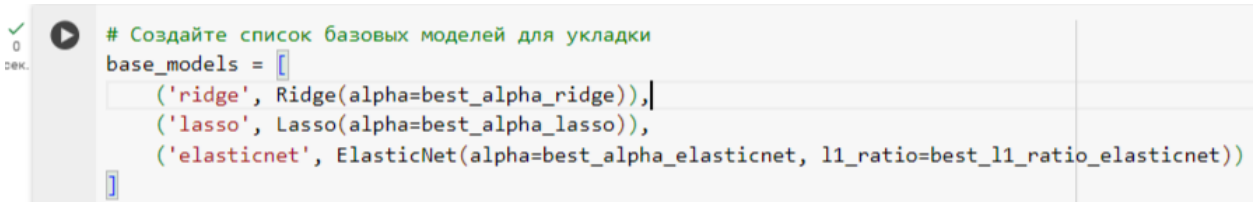


Рисунок 33. Создание списков

Установим регрессор укладки (Stacking Regressor) на основе списка базовых моделей, созданного в предыдущем шаге (см.рис.34).

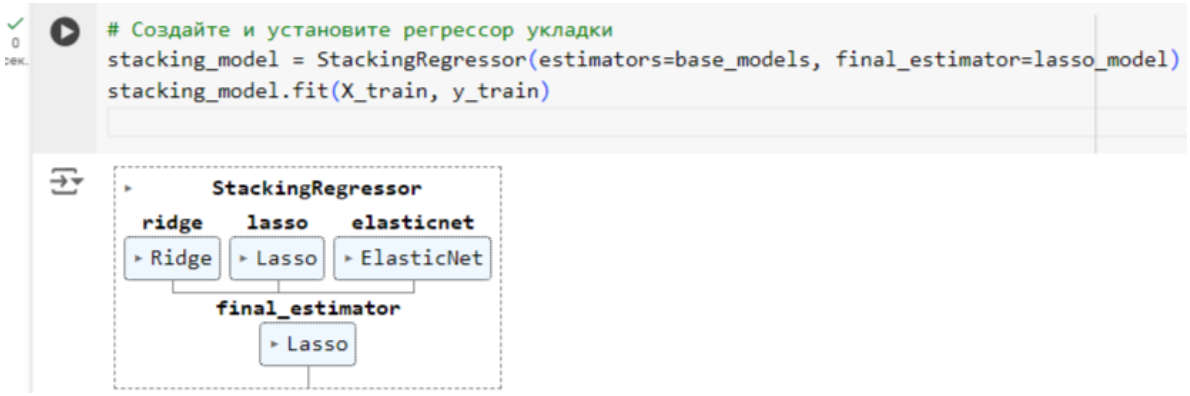


Рисунок 34. Установка регрессора

Обучим регрессор укладки (Stacking Regressor) для генерации прогнозов на тренировочных и тестовых данных (см.рис.35).

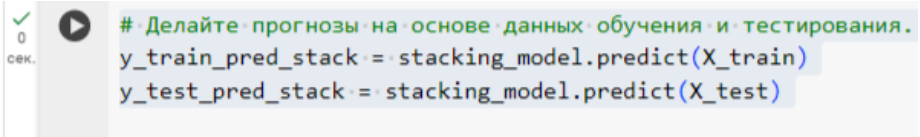


Рисунок 35. Прогнозы и тестирование

Вычислим коэффициент детерминации (R-квадрат) и среднеквадратичную ошибку (MSE) для регрессора укладки (Stacking Regressor) на тренировочных и тестовых данных, а затем выводит результаты (см.рис.36).

```
# Вычислите R-квадрат и среднеквадратичную ошибку для оценки
stack_r2_train = r2_score(y_train, y_train_pred_stack)
stack_r2_test = r2_score(y_test, y_test_pred_stack)
stack_mse_train = mean_squared_error(y_train, y_train_pred_stack)
stack_mse_test = mean_squared_error(y_test, y_test_pred_stack)

print(f"\nStacking Regressor (Stacking):")
print(f"Training R-squared: {stack_r2_train:.4f}, Training MSE: {stack_mse_train:.4f}")
print(f"Testing R-squared: {stack_r2_test:.4f}, Testing MSE: {stack_mse_test:.4f}")
```

```
Stacking Regressor (Stacking):
Training R-squared: 0.9130, Training MSE: 5187676.6900
Testing R-squared: 0.8413, Testing MSE: 12529716.9786
```

Рисунок 35. Вычисления

Создадим и обучим модель линейной регрессии с использованием метода стохастического градиентного спуска (SGD) с регуляризацией (см.рис.36).

```
from sklearn.linear_model import SGDRegressor

sgdr = SGDRegressor(max_iter=1000)
sgdr.fit(X_train, y_train)
y_train_pred_sgd = sgdr.predict(X_train)
y_test_pred_sgd = sgdr.predict(X_test)
```

Рисунок 36. Обучение модели линейной регрессии

Вычислит коэффициент детерминации (R-квадрат) и среднеквадратичную ошибку (MSE) для модели линейной регрессии, обученной с использованием метода стохастического градиентного спуска (SGD Regressor) (см.рис.37).

```
# Вычислите R-квадрат и среднеквадратичную ошибку для оценки
sgdr_r2_train = r2_score(y_train, y_train_pred_sgd)
sgdr_r2_test = r2_score(y_test, y_test_pred_sgd)
sgdr_mse_train = mean_squared_error(y_train, y_train_pred_sgd)
sgdr_mse_test = mean_squared_error(y_test, y_test_pred_sgd)
```

Рисунок 37. Вычисления

Выведем результаты оценки производительности модели SGD Regressor на тренировочных и тестовых данных (см.рис.38).

```
[91] print(f"\nSGDRegressor (SGD):")
     print(f"Training R-squared: {sgdr_r2_train:.4f}, Training MSE: {sgdr_mse_train:.4f}")
     print(f"Testing R-squared: {sgdr_r2_test:.4f}, Testing MSE: {sgdr_mse_test:.4f}")
```

Рисунок 38. Вывод

Проведем настройку гиперпараметров модели случайного леса с помощью поиска по сетке и затем оценим ее производительность на тестовых данных с использованием метрик среднеквадратичной ошибки и коэффициента детерминации (см.рис.39).

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Рисунок 39. Настройка гиперпараметров

Определим сетку параметров для использования в GridSearchCV, которая будет использоваться для поиска оптимальных гиперпараметров модели RandomForestRegressor (см.рис.40).

```
# Определите сетку параметров для GridSearchCV
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 40. Определение сетки параметров

Настроим модель случайного леса для регрессии, используя оптимизацию гиперпараметров с помощью GridSearchCV, и оценим производительность модели на обучающем и тестовом наборах данных с использованием метрик R-квадрат и среднеквадратичной ошибки (см.рис.41).

```
[215] # Создайте регрессионную модель случайного леса
      rf_model = RandomForestRegressor(random_state=42)

[216] # Создать объект GridSearchCV с перекрестной проверкой
      grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

[217] # Сопоставить объект GridSearchCV с обучающими данными
      grid_search.fit(X_train, y_train)
      # Получите наилучшие параметры и наилучшую оценку
      best_params = grid_search.best_params_
      best_estimator = grid_search.best_estimator_

[218] # Делайте прогнозы на основе данных обучения и тестирования, используя наилучший оценщик.
      y_train_pred_rf = best_estimator.predict(X_train)
      y_test_pred_rf = best_estimator.predict(X_test)

      # Вычислите R-квадрат и среднеквадратичную ошибку для оценки
      rf_r2_train = r2_score(y_train, y_train_pred_rf)
      rf_r2_test = r2_score(y_test, y_test_pred_rf)
      rf_mse_train = mean_squared_error(y_train, y_train_pred_rf)
      rf_mse_test = mean_squared_error(y_test, y_test_pred_rf)
```

Рисунок 41. Модель случайного леса

Выведем результаты обучения и тестирования модели Random Forest Regression, после того как были найдены оптимальные гиперпараметры с помощью GridSearchCV (см.рис.42).

```
print("Best Parameters:", best_params)
print(f"\nRandom Forest Regression (Bagging):")
print(f"Training R-squared: {rf_r2_train:.4f}, Training MSE: {rf_mse_train:.4f}")
print(f"Testing R-squared: {rf_r2_test:.4f}, Testing MSE: {rf_mse_test:.4f}")
```

```
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1000}

Random Forest Regression (Bagging):
Training R-squared: 0.9831, Training MSE: 1009068.2845
Testing R-squared: 0.8900, Testing MSE: 8686267.7284
```

Рисунок 42. Вывод результата

Оценить качество модели Random Forest Regression (rf_model) с помощью перекрестной проверки (см.рис.43).

```
# Определите функцию для выполнения перекрестной проверки и вычисления среднего значения R-квадрата баллов.
def perform_cross_validation(model, X, y, cv):
    cv_scores = cross_val_score(model, X, y, scoring='r2', cv=cv)
    mean_cv_score = np.mean(cv_scores)
    return mean_cv_score

# Перекрестная проверка с 3-кратным повторением
cv_3folds_score = perform_cross_validation(rf_model, X, y, cv=3)
print(f"Cross Validation (3-folds) R-squared: {cv_3folds_score:.4f}")

# Перекрестная проверка с 5-кратным повторением
cv_5folds_score = perform_cross_validation(rf_model, X, y, cv=5)
print(f"Cross Validation (5-folds) R-squared: {cv_5folds_score:.4f}")

# Перекрестная проверка с 10-кратным увеличением
cv_10folds_score = perform_cross_validation(rf_model, X, y, cv=10)
print(f"Cross Validation (10-folds) R-squared: {cv_10folds_score:.4f}")
```

```
Cross Validation (3-folds) R-squared: 0.7820
Cross Validation (5-folds) R-squared: 0.3863
Cross Validation (10-folds) R-squared: 0.5521
```

Рисунок 43. Оценка качества модели

Проведем аналогичные действия, что и выше, но в данном случае используется алгоритм Gradient Boosting Regression вместо Random Forest Regression (см.рис.44).

```
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Определите сетку параметров
param_grid = {
    'n_estimators': [500, 1000],
    'learning_rate': [0.01, 0.02],
    'max_depth': [3, 5]
}

# Создайте регрессионную модель с ускорением градиента
gb_model = GradientBoostingRegressor(random_state=42)

# Инициализировать кросс-валидатор KFold
num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

# Создать объект GridSearchCV с перекрестной проверкой
grid_search = GridSearchCV(estimator=gb_model, param_grid=param_grid, cv=kf, scoring='neg_mean_squared_error', n_jobs=-1)

# Сопоставьте объект GridSearchCV с обучающими данными
grid_search.fit(X_train, y_train)

# Получите наилучшие параметры и наилучшую оценку
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Делайте прогнозы на основе данных обучения и тестирования, используя наилучший оценщик.
y_train_pred_gb = best_estimator.predict(X_train)
y_test_pred_gb = best_estimator.predict(X_test)

# Вычислите R-квадрат и среднеквадратичную ошибку для оценки
gb_r2_train = r2_score(y_train, y_train_pred_gb)
gb_r2_test = r2_score(y_test, y_test_pred_gb)
gb_mse_train = mean_squared_error(y_train, y_train_pred_gb)
gb_mse_test = mean_squared_error(y_test, y_test_pred_gb)

print("Best Parameters:", best_params)
print(f"\nGradient Boosting Regression (Boosting):")
print(f"Training R-squared: {gb_r2_train:.4f}, Training MSE: {gb_mse_train:.4f}")
print(f"Testing R-squared: {gb_r2_test:.4f}, Testing MSE: {gb_mse_test:.4f}")
```

```
Best Parameters: {'learning_rate': 0.02, 'max_depth': 3, 'n_estimators': 500}

Gradient Boosting Regression (Boosting):
Training R-squared: 0.9946, Training MSE: 321446.0692
Testing R-squared: 0.9144, Testing MSE: 6761369.1647
```

Рисунок 44. Оценка качества

Создадим сводную таблицу с R-квадрат значениями для различных регрессионных методов, примененных к набору данных о диабете. Затем визуализируем эти результаты в виде двух столбчатых диаграмм, одна для обучающего набора, другая для тестового набора (см.рис.45).

Рисунок 45. Сводная таблица

В результате исследования было проведено прогнозирование стоимости автомобилей при помощи машинного обучения. Материалы можно использовать для прогнозирования цен и как учебное пособие.

Ссылка на ноутбук https://colab.research.google.com/drive/1VwMuyKyEDdVjeTihMjK5ptrp5tREgzsK?usp=sharing

Библиографический список

1. Винокурова В. Г., Тимаева С. А. Анализ и прогнозирование продаж ит-оборудования корпоративным заказчикам с помощью методов машинного обучения //Цифровая экономика и информационные технологии. 2023. С. 294-303.
2. Кириченко А. А., Тимаева С. А. Прогнозирование инвестиционной привлекательности объектов недвижимости на основе методов машинного обучения с использованием телеграмм-бота //Цифровая экономика и

информационные технологии. 2023. С. 272-279.
3. Архипова А. А. Применение нейронных сетей в задаче прогнозирования финансовых временных рядов //Экономика и бизнес: теория и практика. – 2023. №. 6-1 (100). С. 18-22.
4. Матвеева А. С. Разведочный анализ данных о прогнозировании энергопотребления //Постулат. 2024. №. 1 январь.
5. Антонов А. А. Прогнозирование размера заработной платы с использованием методов машинного обучения по обработке естественного языка //лучшие исследовательские работы студентов 3. 2024. С. 30.