МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | В. Ю. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| Регрессионный и разведочный анализ |
| по курсу: интеллектуальный анализ данных на основе методов машинного обучения |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

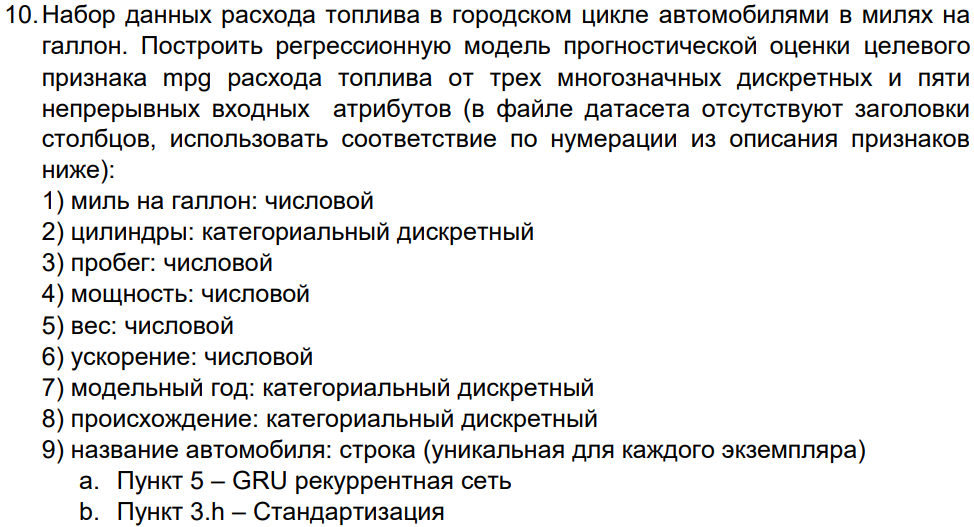
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4134 |  |  |  | Д. В. Самарин,  Р. Р. Усов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы**

Решение задачи прогностического анализ данных на основе моделей нейронных сетей

**Вариант: 14**

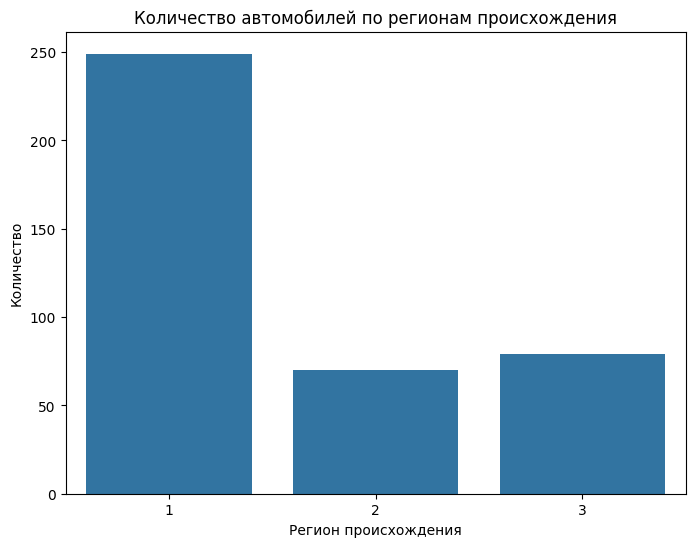


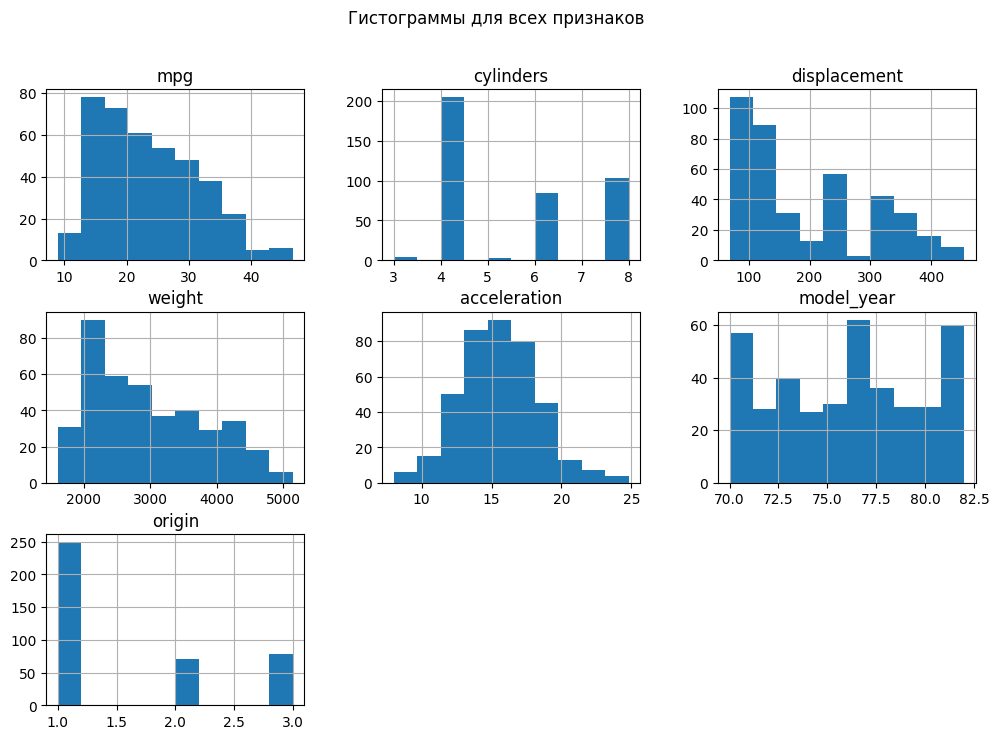
**Ход работы**

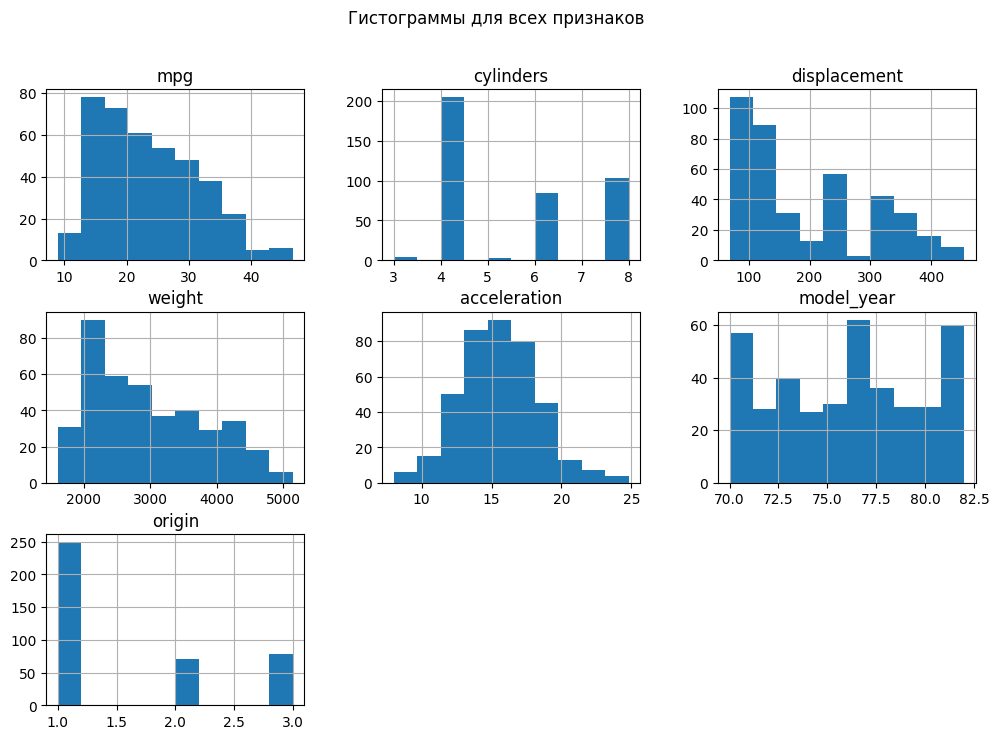
|  |
| --- |
| # Block 1: Импорт библиотек  # =============================================================  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.preprocessing import StandardScaler, OneHotEncoder, PolynomialFeatures  from sklearn.compose import ColumnTransformer  from sklearn.pipeline import Pipeline  from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, GRU, Dropout  from sklearn.linear\_model import LinearRegression  from tensorflow.keras.callbacks import EarlyStopping  import joblib  import os  from datetime import datetime  import tensorflow as tf  import logging  from pickle import dump, load  # Подавление предупреждений TensorFlow  tf.get\_logger().setLevel(logging.ERROR) |
| # Block 2: Загрузка данных и первичный анализ  # =============================================================  def load\_and\_inspect\_data(file\_path):  column\_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model\_year', 'origin', 'car\_name']  df = pd.read\_csv(file\_path, delim\_whitespace=True, names=column\_names)  print("\n========== Первые 5 строк датасета ==========")  print(df.head())  print("\n========== Размерность набора данных ==========")  print(f"Размерность набора данных: {df.shape}")  print("\n========== Типы атрибутов ==========")  print(df.dtypes)  print("\n========== Информация о данных ==========")  df.info()  print("\n========== Уникальные значения в столбце 'origin' ==========")  print(df['origin'].value\_counts())  print("\n========== Описание числовых признаков ==========")  print(df.describe())  return df |

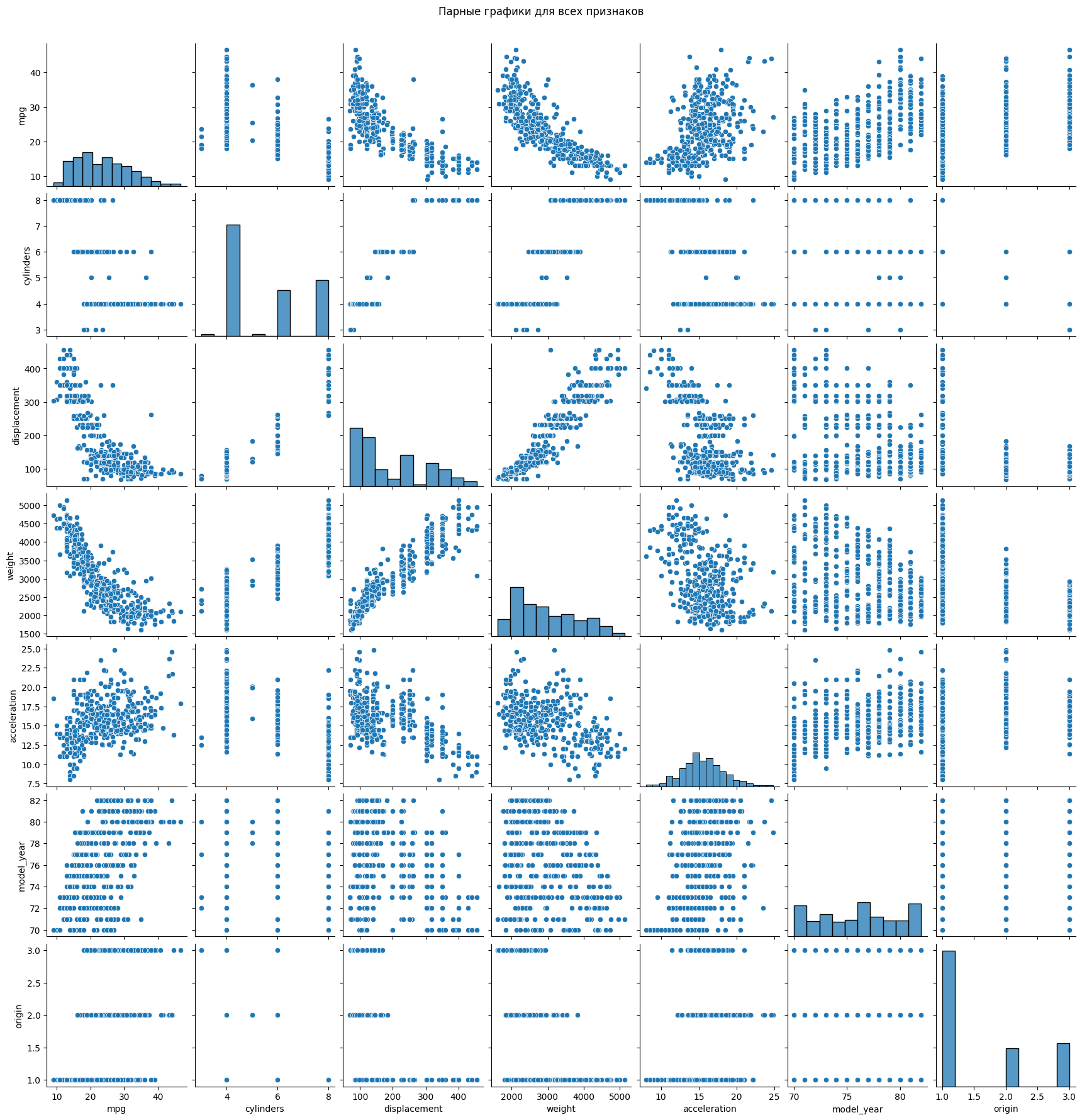
|  |
| --- |
| **========== Первые 5 строк датасета ==========**  **mpg cylinders displacement horsepower weight acceleration model\_year \**  **0 18.0 8 307.0 130.0 3504.0 12.0 70**  **1 15.0 8 350.0 165.0 3693.0 11.5 70**  **2 18.0 8 318.0 150.0 3436.0 11.0 70**  **3 16.0 8 304.0 150.0 3433.0 12.0 70**  **4 17.0 8 302.0 140.0 3449.0 10.5 70**  **origin car\_name**  **1 chevrolet chevelle malibu**  **1 1 buick skylark 320**  **2 1 plymouth satellite**  **3 1 amc rebel sst**  **4 1 ford torino**  **========== Размерность набора данных ==========**  **Размерность набора данных: (398, 9)**  **========== Типы атрибутов ==========**  **mpg float64**  **cylinders int64**  **displacement float64**  **horsepower object**  **weight float64**  **acceleration float64**  **model\_year int64**  **origin int64**  **car\_name object**  **dtype: object**  **========== Информация о данных ==========**  **<class 'pandas.core.frame.DataFrame'>**  **RangeIndex: 398 entries, 0 to 397**  **Data columns (total 9 columns):**  **# Column Non-Null Count Dtype**  **--- ------ -------------- -----**  **0 mpg 398 non-null float64**  **1 cylinders 398 non-null int64**  **2 displacement 398 non-null float64**  **3 horsepower 398 non-null object**  **4 weight 398 non-null float64**  **5 acceleration 398 non-null float64**  **6 model\_year 398 non-null int64**  **7 origin 398 non-null int64**  **8 car\_name 398 non-null object**  **dtypes: float64(4), int64(3), object(2)**  **memory usage: 28.1+ KB**  **========== Уникальные значения в столбце 'origin' ==========**  **origin**  **249**  **79**  **70**  **Name: count, dtype: int64**  **========== Описание числовых признаков ==========**  **mpg cylinders displacement weight acceleration \**  **count 398.000000 398.000000 398.000000 398.000000 398.000000**  **mean 23.514573 5.454774 193.425879 2970.424623 15.568090**  **std 7.815984 1.701004 104.269838 846.841774 2.757689**  **min 9.000000 3.000000 68.000000 1613.000000 8.000000**  **25% 17.500000 4.000000 104.250000 2223.750000 13.825000**  **50% 23.000000 4.000000 148.500000 2803.500000 15.500000**  **75% 29.000000 8.000000 262.000000 3608.000000 17.175000**  **max 46.600000 8.000000 455.000000 5140.000000 24.800000**  **model\_year origin**  **count 398.000000 398.000000**  **mean 76.010050 1.572864**  **std 3.697627 0.802055**  **min 70.000000 1.000000**  **25% 73.000000 1.000000**  **50% 76.000000 1.000000**  **75% 79.000000 2.000000**  **max 82.000000 3.000000** |

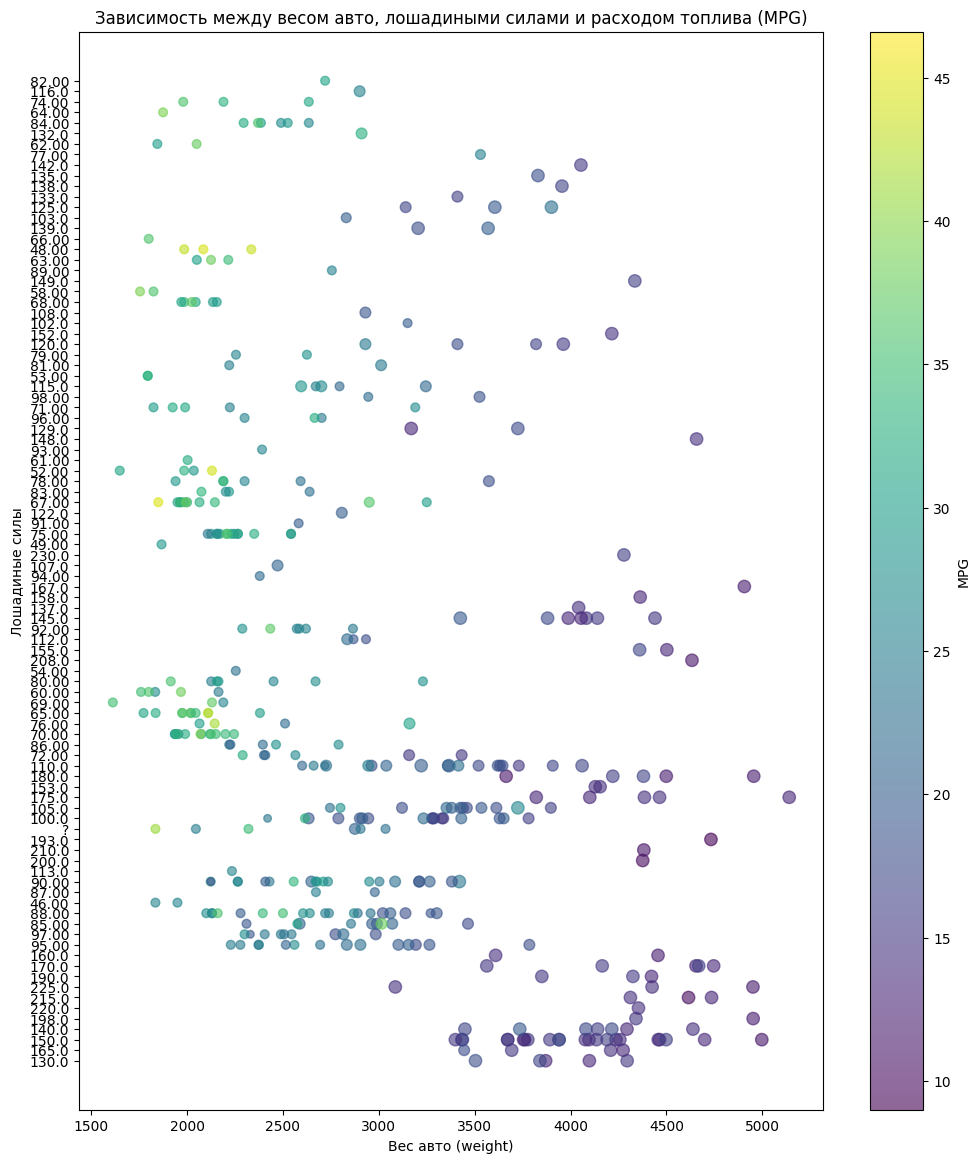
|  |
| --- |
| # Block 3: Визуализация данных  # =============================================================  def visualize\_data(df):  plt.figure(figsize=(8, 6))  sns.countplot(x='origin', data=df)  plt.title('Количество автомобилей по регионам происхождения')  plt.xlabel('Регион происхождения')  plt.ylabel('Количество')  plt.show()  df.hist(figsize=(12, 8))  plt.suptitle('Гистограммы для всех признаков')  plt.show()  sns.pairplot(df)  plt.suptitle('Парные графики для всех признаков', y=1.02)  plt.show()  plt.figure(figsize=(12, 14))  plt.scatter(df['weight'], df['horsepower'], c=df['mpg'], cmap='viridis', alpha=0.6, s=df['cylinders']\*10)  plt.colorbar(label='MPG')  plt.xlabel('Вес авто (weight)')  plt.ylabel('Лошадиные силы')  plt.title('Зависимость между весом авто, лошадиными силами и расходом топлива (MPG)')  plt.show() |

****

****

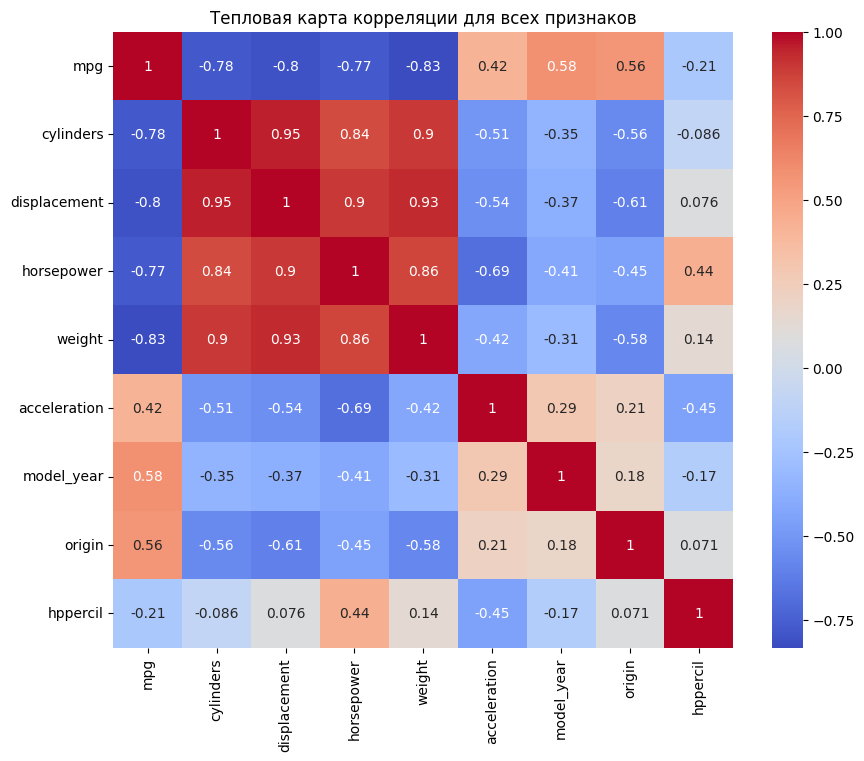
****

****

****

|  |
| --- |
| # Block 4: Очистка данных  # =============================================================  def clean\_data(df):  df['horsepower'] = pd.to\_numeric(df['horsepower'], errors='coerce')  df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())  df['hppercil'] = df['horsepower'] / df['cylinders']  df.drop(columns=['car\_name'], inplace=True)  for column in df.columns:  df[column] = df[column].replace('?', np.nan)  return df |

|  |
| --- |
| # Block 5: Корреляционный анализ  # =============================================================  def correlation\_analysis(df):  print("========== Матрица корреляции ==========")  corr\_matrix = df.corr()  print(corr\_matrix)  print("========== Корреляция с 'mpg' ==========")  correlation\_target = corr\_matrix['mpg'].sort\_values(ascending=False)  print(correlation\_target)  plt.figure(figsize=(10, 8))  sns.heatmap(df.select\_dtypes(include=[np.number]).corr(), annot=True, cmap='coolwarm')  plt.title('Тепловая карта корреляции для всех признаков')  plt.show() |
| **========== Матрица корреляции ==========**  **mpg cylinders displacement horsepower weight \**  **mpg 1.000000 -0.775396 -0.804203 -0.773453 -0.831741**  **cylinders -0.775396 1.000000 0.950721 0.841284 0.896017**  **displacement -0.804203 0.950721 1.000000 0.895778 0.932824**  **horsepower -0.773453 0.841284 0.895778 1.000000 0.862442**  **weight -0.831741 0.896017 0.932824 0.862442 1.000000**  **acceleration 0.420289 -0.505419 -0.543684 -0.686590 -0.417457**  **model\_year 0.579267 -0.348746 -0.370164 -0.413733 -0.306564**  **origin 0.563450 -0.562543 -0.609409 -0.452096 -0.581024**  **hppercil -0.213253 -0.085724 0.075692 0.437047 0.138797**  **acceleration model\_year origin hppercil**  **mpg 0.420289 0.579267 0.563450 -0.213253**  **cylinders -0.505419 -0.348746 -0.562543 -0.085724**  **displacement -0.543684 -0.370164 -0.609409 0.075692**  **horsepower -0.686590 -0.413733 -0.452096 0.437047**  **weight -0.417457 -0.306564 -0.581024 0.138797**  **acceleration 1.000000 0.288137 0.205873 -0.448055**  **model\_year 0.288137 1.000000 0.180662 -0.167030**  **origin 0.205873 0.180662 1.000000 0.071384**  **hppercil -0.448055 -0.167030 0.071384 1.000000**  **========== Корреляция с 'mpg' ==========**  **mpg 1.000000**  **model\_year 0.579267**  **origin 0.563450**  **acceleration 0.420289**  **hppercil -0.213253**  **horsepower -0.773453**  **cylinders -0.775396**  **displacement -0.804203**  **weight -0.831741**  **Name: mpg, dtype: float64** |

****

|  |
| --- |
| # Block 6: Разделение данных на обучающую и тестовую выборки  # =============================================================  def split\_data(df, target\_column='mpg'):  X\_full = df.drop(columns=[target\_column])  y = df[target\_column]  X\_filtered = df[['cylinders', 'horsepower', 'weight']]  return X\_full, X\_filtered, y |

|  |
| --- |
| # Block 7: Определение категориальных и числовых признаков  # =============================================================  def define\_features():  categorical\_features = []  numerical\_features\_full = ['cylinders', 'horsepower', 'weight']  numerical\_features\_filtered = ['cylinders', 'horsepower', 'weight']  return categorical\_features, numerical\_features\_full, numerical\_features\_filtered |

|  |
| --- |
| # Block 8: Предобработка данных  # =============================================================  def create\_preprocessors(numerical\_features\_full, numerical\_features\_filtered):  preprocessor\_full = ColumnTransformer(  transformers=[('num', StandardScaler(), numerical\_features\_full)]  )  preprocessor\_filtered = ColumnTransformer(  transformers=[('num', StandardScaler(), numerical\_features\_filtered)]  )  scaler\_full = StandardScaler()  scaler\_filtered = StandardScaler()  return preprocessor\_full, preprocessor\_filtered, scaler\_full, scaler\_filtered |

|  |
| --- |
| # Block 9: Определение моделей GRU  # =============================================================  def create\_gru\_model(input\_shape):  model = Sequential()  model.add(GRU(128, input\_shape=input\_shape, return\_sequences=True))  model.add(Dropout(0.2))  model.add(GRU(64))  model.add(Dropout(0.2))  model.add(Dense(1, activation='relu'))  model.compile(optimizer='RMSprop', loss='mse')  return model |

|  |
| --- |
| # 10. Создание и обучение моделей  # =============================================================  def train\_models(X\_full, X\_filtered, y, preprocessor\_full, preprocessor\_filtered, scaler\_full, scaler\_filtered):  # разделение данных на обучающую и тестовую выборки с использованием train\_test\_split соответствует   этапу 5.1 в методических материалах.  # Разделение данных на обучающую и тестовую выборки (80% - обучение, 20% - тест)  X\_train\_full, X\_test\_full, y\_train\_full, y\_test\_full = train\_test\_split(X\_full, y, test\_size=0.2, random\_state=42)  X\_train\_filtered, X\_test\_filtered, y\_train\_filtered, y\_test\_filtered = train\_test\_split(X\_filtered, y, test\_size=0.2, random\_state=42)  # Стандартизация данных и создание наборов данных для GRU  X\_train\_full\_prepared = preprocessor\_full.fit\_transform(X\_train\_full)  X\_test\_full\_prepared = preprocessor\_full.transform(X\_test\_full)  X\_train\_filtered\_prepared = preprocessor\_filtered.fit\_transform(X\_train\_filtered)  X\_test\_filtered\_prepared = preprocessor\_filtered.transform(X\_test\_filtered)  # Стандартизация полных данных  X\_train\_full\_standardized = scaler\_full.fit\_transform(X\_train\_full)  X\_test\_full\_standardized = scaler\_full.transform(X\_test\_full)  # Стандартизация фильтрованных данных  X\_train\_filtered\_standardized = scaler\_filtered.fit\_transform(X\_train\_filtered[numerical\_features\_filtered])  X\_test\_filtered\_standardized = scaler\_filtered.transform(X\_test\_filtered[numerical\_features\_filtered])  # Добавление ранней остановки  early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)  # Обучение моделей GRU на различных данных  # GRU Model 1: Полная модель  gru\_model\_full = create\_gru\_model((X\_train\_full\_prepared.shape[1], 1))  gru\_model\_full.fit(np.expand\_dims(X\_train\_full\_prepared, axis=-1), y\_train\_full, epochs=240, batch\_size=32,  callbacks=[early\_stopping])  # GRU Model 2: Фильтрованная модель  gru\_model\_filtered = create\_gru\_model((X\_train\_filtered\_prepared.shape[1], 1))  gru\_model\_filtered.fit(np.expand\_dims(X\_train\_filtered\_prepared, axis=-1), y\_train\_filtered, epochs=240, batch\_size=32,  callbacks=[early\_stopping])  # GRU Model 3: Полная модель с стандартизированными данными  gru\_model\_full\_standardized = create\_gru\_model((X\_train\_full\_standardized.shape[1], 1))  gru\_model\_full\_standardized.fit(np.expand\_dims(X\_train\_full\_standardized, axis=-1), y\_train\_full, epochs=240, batch\_size=32,  callbacks=[early\_stopping])  # GRU Model 4: Фильтрованная модель с стандартизацией  gru\_model\_filtered\_standardized = create\_gru\_model((X\_train\_filtered\_standardized.shape[1], 1))  gru\_model\_filtered\_standardized.fit(np.expand\_dims(X\_train\_filtered\_standardized, axis=-1), y\_train\_filtered, epochs=240, batch\_size=32,  callbacks=[early\_stopping])  return (gru\_model\_full, gru\_model\_filtered, gru\_model\_full\_standardized, gru\_model\_filtered\_standardized), X\_train\_full\_prepared, X\_test\_full\_prepared, y\_train\_full, y\_test\_full, X\_train\_filtered\_prepared, X\_test\_filtered\_prepared, y\_train\_filtered, y\_test\_filtered, X\_train\_full\_standardized, X\_test\_full\_standardized, X\_train\_filtered\_standardized, X\_test\_filtered\_standardized  def train\_linear\_models(X\_train\_full\_prepared, X\_train\_filtered\_prepared, X\_train\_full\_standardized, X\_train\_filtered\_standardized, y\_train\_full, y\_train\_filtered):  lin\_reg\_full = LinearRegression()  lin\_reg\_filtered = LinearRegression()  lin\_reg\_full\_standardized = LinearRegression()  lin\_reg\_filtered\_standardized = LinearRegression()  # Обучение моделей линейной регрессии на различных наборах данных  lin\_reg\_full.fit(X\_train\_full\_prepared, y\_train\_full)  lin\_reg\_filtered.fit(X\_train\_filtered\_prepared, y\_train\_filtered)  lin\_reg\_full\_standardized.fit(X\_train\_full\_standardized, y\_train\_full)  lin\_reg\_filtered\_standardized.fit(X\_train\_filtered\_standardized, y\_train\_filtered)  return lin\_reg\_full, lin\_reg\_filtered, lin\_reg\_full\_standardized, lin\_reg\_filtered\_standardized  def perform\_grid\_search(X\_train\_filtered, y\_train\_filtered, X\_test\_filtered, y\_test\_filtered):  # Grid Search используется для поиска оптимальных гиперпараметров модели, как описано в разделе 5.2 методических материалов.  print("\n==================== Grid Search для Полиномиальной Регрессии ====================\n")  deg = 2  # Выбранная степень полинома  polynomial\_features = PolynomialFeatures(degree=deg)  housing\_X\_poly = polynomial\_features.fit\_transform(X\_train\_filtered)  model = LinearRegression()  model.fit(housing\_X\_poly, y\_train\_filtered)  housing\_Y\_poly\_pred = model.predict(polynomial\_features.transform(X\_test\_filtered))  rmse = np.sqrt(mean\_squared\_error(y\_test\_filtered, housing\_Y\_poly\_pred))  r2 = r2\_score(y\_test\_filtered, housing\_Y\_poly\_pred)  print(f"Poly degree = {deg}")  print(f"RMSE for Polynomial Regression: {rmse}")  print(f"R2 Score for Polynomial Regression: {r2}\n")  # =============================================================  #оценка моделей с использованием метрик MSE и R2 соответствует этапу 5.3  # =============================================================  def evaluate\_models(gru\_models, lin\_models, X\_test\_prepared, X\_test\_filtered, X\_test\_full\_standardized, X\_test\_filtered\_standardized, y\_test\_full, y\_test\_filtered):  gru\_model\_full, gru\_model\_filtered, gru\_model\_full\_standardized, gru\_model\_filtered\_standardized = gru\_models  lin\_reg\_full, lin\_reg\_filtered, lin\_reg\_full\_standardized, lin\_reg\_filtered\_standardized = lin\_models  # Оценка моделей GRU  y\_pred\_gru\_full = gru\_model\_full.predict(np.expand\_dims(X\_test\_prepared, axis=-1))  y\_pred\_gru\_filtered = gru\_model\_filtered.predict(np.expand\_dims(X\_test\_filtered, axis=-1))  y\_pred\_gru\_full\_standardized = gru\_model\_full\_standardized.predict(np.expand\_dims(X\_test\_full\_standardized, axis=-1))  y\_pred\_gru\_filtered\_standardized = gru\_model\_filtered\_standardized.predict(np.expand\_dims(X\_test\_filtered\_standardized, axis=-1))  evaluation\_results = []  evaluation\_results.append("\n======================== Оценка моделей GRU ========================\n")  evaluation\_results.append(f"GRU Model 1 (Полная):\n  - MSE: {mean\_squared\_error(y\_test\_full, y\_pred\_gru\_full):.2f}\n  - R2 Score: {r2\_score(y\_test\_full, y\_pred\_gru\_full):.2f}\n")  evaluation\_results.append(f"GRU Model 2 (Фильтрованная):\n  - MSE: {mean\_squared\_error(y\_test\_filtered, y\_pred\_gru\_filtered):.2f}\n  - R2 Score: {r2\_score(y\_test\_filtered, y\_pred\_gru\_filtered):.2f}\n")  evaluation\_results.append(f"GRU Model 3 (Полная с стандартизацией):\n  - MSE: {mean\_squared\_error(y\_test\_full, y\_pred\_gru\_full\_standardized):.2f}\n  - R2 Score: {r2\_score(y\_test\_full, y\_pred\_gru\_full\_standardized):.2f}\n")  evaluation\_results.append(f"GRU Model 4 (Фильтрованная с стандартизацией):\n  - MSE: {mean\_squared\_error(y\_test\_filtered, y\_pred\_gru\_filtered\_standardized):.2f}\n  - R2 Score: {r2\_score(y\_test\_filtered, y\_pred\_gru\_filtered\_standardized):.2f}\n")  # Оценка моделей линейной регрессии  y\_pred\_lin\_full = lin\_reg\_full.predict(X\_test\_prepared)  y\_pred\_lin\_filtered = lin\_reg\_filtered.predict(X\_test\_filtered)  y\_pred\_lin\_full\_standardized = lin\_reg\_full\_standardized.predict(X\_test\_full\_standardized)  y\_pred\_lin\_filtered\_standardized = lin\_reg\_filtered\_standardized.predict(X\_test\_filtered\_standardized)  evaluation\_results.append("\n==================== Оценка моделей Линейной Регрессии ====================\n")  evaluation\_results.append(f"Linear Regression Model 1 (Полная):\n  - MSE: {mean\_squared\_error(y\_test\_full, y\_pred\_lin\_full):.2f}\n  - R2 Score: {r2\_score(y\_test\_full, y\_pred\_lin\_full):.2f}\n")  evaluation\_results.append(f"Linear Regression Model 2 (Фильтрованная):\n  - MSE: {mean\_squared\_error(y\_test\_filtered, y\_pred\_lin\_filtered):.2f}\n  - R2 Score: {r2\_score(y\_test\_filtered, y\_pred\_lin\_filtered):.2f}\n")  evaluation\_results.append(f"Linear Regression Model 3 (Полная с стандартизацией):\n  - MSE: {mean\_squared\_error(y\_test\_full, y\_pred\_lin\_full\_standardized):.2f}\n  - R2 Score: {r2\_score(y\_test\_full, y\_pred\_lin\_full\_standardized):.2f}\n")  evaluation\_results.append(f"Linear Regression Model 4 (Фильтрованная с стандартизацией):\n  - MSE: {mean\_squared\_error(y\_test\_filtered, y\_pred\_lin\_filtered\_standardized):.2f}\n  - R2 Score: {r2\_score(y\_test\_filtered, y\_pred\_lin\_filtered\_standardized):.2f}\n")  # Печать результатов оценки  #for result in evaluation\_results:  #    print(result)  return evaluation\_results  # =============================================================  #Сохранение оценок и моделей  # =============================================================  def save\_evaluation\_results(evaluation\_results):  results\_dir = "model\_results"  os.makedirs(results\_dir, exist\_ok=True)  timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")  results\_file = os.path.join(results\_dir, f"evaluation\_{timestamp}.txt")  # Запись оценок в файл  with open(results\_file, 'w', encoding='utf-8') as f:  f.write(''.join(evaluation\_results))  # Вывод результатов оценки в консоль  print(''.join(evaluation\_results))  print(f"Оценки моделей сохранены в файл: {results\_file}")  def save\_gru\_models(gru\_models, timestamp):  models\_dir = "saved\_models"  os.makedirs(models\_dir, exist\_ok=True)  gru\_model\_full, gru\_model\_filtered, gru\_model\_full\_standardized, gru\_model\_filtered\_standardized = gru\_models  # Сохранение моделей GRU  gru\_full\_path = os.path.join(models\_dir, f"gru\_model\_full\_{timestamp}.h5")  gru\_model\_full.save(gru\_full\_path)  print(f"GRU Model Full сохранена в: {gru\_full\_path}")  gru\_filtered\_path = os.path.join(models\_dir, f"gru\_model\_filtered\_{timestamp}.h5")  gru\_model\_filtered.save(gru\_filtered\_path)  print(f"GRU Model Filtered сохранена в: {gru\_filtered\_path}")  gru\_full\_std\_path = os.path.join(models\_dir, f"gru\_model\_full\_standardized\_{timestamp}.h5")  gru\_model\_full\_standardized.save(gru\_full\_std\_path)  print(f"GRU Model Full Standardized сохранена в: {gru\_full\_std\_path}")  gru\_filtered\_std\_path = os.path.join(models\_dir, f"gru\_model\_filtered\_standardized\_{timestamp}.h5")  gru\_model\_filtered\_standardized.save(gru\_filtered\_std\_path)  print(f"GRU Model Filtered Standardized сохранена в: {gru\_filtered\_std\_path}")  def save\_linear\_models(lin\_models, timestamp):  models\_dir = "saved\_models"  os.makedirs(models\_dir, exist\_ok=True)  lin\_reg\_full, lin\_reg\_filtered, lin\_reg\_full\_standardized, lin\_reg\_filtered\_standardized = lin\_models  # Сохранение моделей линейной регрессии с помощью joblib  # Использование joblib для сохранения моделей рекомендуется для сериализации больших массивов NumPy, как описано в разделе 7.2 методических материалов.  lin\_reg\_full\_path = os.path.join(models\_dir, f"lin\_reg\_full\_{timestamp}.joblib")  joblib.dump(lin\_reg\_full, lin\_reg\_full\_path)  print(f"Linear Regression Model Full сохранена в: {lin\_reg\_full\_path}")  lin\_reg\_filtered\_path = os.path.join(models\_dir, f"lin\_reg\_filtered\_{timestamp}.joblib")  joblib.dump(lin\_reg\_filtered, lin\_reg\_filtered\_path)  print(f"Linear Regression Model Filtered сохранена в: {lin\_reg\_filtered\_path}")  lin\_reg\_full\_std\_path = os.path.join(models\_dir, f"lin\_reg\_full\_standardized\_{timestamp}.joblib")  joblib.dump(lin\_reg\_full\_standardized, lin\_reg\_full\_std\_path)  print(f"Linear Regression Model Full Standardized сохранена в: {lin\_reg\_full\_std\_path}")  lin\_reg\_filtered\_std\_path = os.path.join(models\_dir, f"lin\_reg\_filtered\_standardized\_{timestamp}.joblib")  joblib.dump(lin\_reg\_filtered\_standardized, lin\_reg\_filtered\_std\_path)  print(f"Linear Regression Model Filtered Standardized сохранена в: {lin\_reg\_filtered\_std\_path}")  def save\_polynomial\_model(model):  filename = 'finalized\_model.sav'  dump(model, open(filename, 'wb'))  print(f"Polynomial Regression Model сохранена в файл: {filename}")  # =============================================================  # Основная функция  #==========================================================================================================================#  if \_\_name\_\_ == "\_\_main\_\_":  # Загрузка и первичный анализ данных  df = load\_and\_inspect\_data('V10.txt')  # Визуализация данных  visualize\_data(df)  # Очистка данных  df = clean\_data(df)  # Корреляционный анализ  correlation\_analysis(df)  # 1. Разделение данных  X\_full, X\_filtered, y = split\_data(df)  # 2. Определение признаков  categorical\_features, numerical\_features\_full, numerical\_features\_filtered = define\_features()  # 3. Предобработка данных  preprocessor\_full, preprocessor\_filtered, scaler\_full, scaler\_filtered = create\_preprocessors(numerical\_features\_full, numerical\_features\_filtered)  # 4. Обучение моделей  gru\_models, X\_train\_full\_prepared, X\_test\_full\_prepared, y\_train\_full, y\_test\_full, X\_train\_filtered\_prepared, X\_test\_filtered\_prepared, y\_train\_filtered, y\_test\_filtered, X\_train\_full\_standardized, X\_test\_full\_standardized, X\_train\_filtered\_standardized, X\_test\_filtered\_standardized = train\_models(X\_full, X\_filtered, y, preprocessor\_full, preprocessor\_filtered, scaler\_full, scaler\_filtered)  # 5. Обучение линейных моделей  lin\_models = train\_linear\_models(X\_train\_full\_prepared, X\_train\_filtered\_prepared, X\_train\_full\_standardized, X\_train\_filtered\_standardized, y\_train\_full, y\_train\_filtered)  # 6. Выполнение Grid Search  perform\_grid\_search(X\_train\_filtered\_prepared, y\_train\_filtered, X\_test\_filtered\_prepared, y\_test\_filtered)  # 7. Оценка моделей и сохранение результатов  evaluation\_results = evaluate\_models(gru\_models, lin\_models, X\_test\_full\_prepared, X\_test\_filtered\_prepared, X\_test\_full\_standardized, X\_test\_filtered\_standardized, y\_test\_full, y\_test\_filtered)  save\_evaluation\_results(evaluation\_results)  # 9. Сохранение моделей  timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")  save\_gru\_models(gru\_models, timestamp)  save\_linear\_models(lin\_models, timestamp)  save\_polynomial\_model(LinearRegression())  #==========================================================================================================================# |

**Результат:**

|  |
| --- |
| **======================== Оценка моделей GRU ========================**  **GRU Model 1 (Полная):**  **MSE: 12.60**  **R2 Score: 0.77**  **GRU Model 2 (Фильтрованная):**  **- MSE: 13.49**  **- R2 Score: 0.75**  **GRU Model 3 (Полная с стандартизацией):**  **- MSE: 4.76**  **- R2 Score: 0.91**  **GRU Model 4 (Фильтрованная с стандартизацией):**  **- MSE: 12.00**  **- R2 Score: 0.78**  **==================== Оценка моделей Линейной Регрессии ====================**  **Linear Regression Model 1 (Полная):**  **MSE: 14.54**  **R2 Score: 0.73**  **Linear Regression Model 2 (Фильтрованная):**  **MSE: 14.54**  **R2 Score: 0.73**  **Linear Regression Model 3 (Полная с стандартизацией):**  **MSE: 7.87**  **R2 Score: 0.85**  **Linear Regression Model 4 (Фильтрованная с стандартизацией):**  **- MSE: 14.54**  **- R2 Score: 0.73** |

**Для дальнейшего использования**: предпочтительно использовать GRU Model 3, так как она показала наилучшие результаты.

Лучшие модели:

* GRU Model 3 (Полная с стандартизацией) является абсолютным лидером по качеству (R2=0.91 R^2 = 0.91R2=0.91, MSE = 4.76).
* Linear Regression Model 3 (Полная с стандартизацией) также выделяется (R2=0.85 R^2 = 0.85R2=0.85, MSE = 7.87), но уступает GRU.

**Выводы**

В ходе лабораторной работы, мы выполнили регрессионный анализ данных на основе полносвязной нейросетевой модели и нейросетевой модели, указанной в варианте.