Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Курс «Технологии машинного обучения»
Отчёт по лабораторной работе №6

Выполнила:
студентка группы ИУ5-62Б Вешторт Е.С.
Подпись:

Проверил:
Гапанюк Ю.Е.
Подпись:

2025 г.

In [1]:
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingRegressor, RandomForestRegressor, GradientBoostingRegres
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
from gmdh import Combi, Mia
```

Подготовка датасета

Загрузка данных, для работы выбрала датасет с информацией о продаже машин

In [2]:
```
df = pd.read_csv("C://Users/dielo/OneDrive/добавить/Documents/Учебное/ТМО/Лаб1/all_anonymized
```

```
C:\Users\dielo\AppData\Local\Temp\ipykernel_12156\746799274.py:1: DtypeWarning: Columns (7,12) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("C://Users/dielo/OneDrive/добавить/Documents/Учебное/ТМО/Лаб1/all_anonymized_2015_11_2017_03.csv")
```

In [3]:
```
df.sample(5)
```

Out[3]:

	maker	model	mileage	manufacture_year	engine_displacement	engine_power	body_t
2110652	volkswagen	polo	151298.0	2006.0	1198.0	47.0	o
2923682	audi	200	NaN	NaN	NaN	NaN	o
1848017	ford	s-max	163000.0	2010.0	1997.0	120.0	o
579362	fiat	marea	NaN	1999.0	1616.0	76.0	com
3284050	mercedes-benz	ml320	220000.0	2001.0	NaN	NaN	o

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3552912 entries, 0 to 3552911
Data columns (total 16 columns):
 #   Column               Dtype  
---  ------               -----  
 0   maker                object 
 1   model                object 
 2   mileage              float64
 3   manufacture_year     float64
 4   engine_displacement  float64
 5   engine_power         float64
 6   body_type            object 
 7   color_slug           object 
 8   stk_year             float64
 9   transmission         object 
 10  door_count           float64
 11  seat_count           float64
 12  fuel_type            object 
 13  date_created         object 
 14  date_last_seen       object 
 15  price_eur            float64
dtypes: float64(8), object(8)
memory usage: 433.7+ MB
```

Столбцы с годами, количеством дверей и сидений, преобразуем в целые:

In [5]:
```
df[['manufacture_year', 'stk_year', 'door_count', 'seat_count']] = df[['manufacture_year', 's
```

In [6]:
```
missing_values = df.isnull().sum()
print("Пропущенные значения в каждом столбце:\n", missing_values[missing_values > 0])
```

```
Пропущенные значения в каждом столбце:
 maker                  518915
model                 1133361
mileage                362584
manufacture_year       370578
engine_displacement    743414
engine_power           554877
body_type             1122914
color_slug            3343411
stk_year              3016807
transmission           741630
door_count            1090066
seat_count            1287099
fuel_type             1847606
dtype: int64
```

Сперва заполним пропуски в столбцах maker и model, т.к. в дальнейшем будем опираться на них для заполнения столбцов со свойствами машин Можно было бы заполнить каждый столбец модой, но тогда может получиться что-нибудь странное вроде opel octavia, что не будет соответсвовать ни одному существующему автомобилю и может испортить дальнейшие заполнения свойств Поэтому логика заполнения будет следующая: если известен производитель, берем самую частую его модель в датасете (если производитель больше в датасете не встречается, будем заполнять строкой 'unknown'), если известна модель, ищем самого частого ее производителя (на случай совпадения названий), иначе также пишем 'unknow' Если нет ни того, ни другого, то ищем самую частую пару производитель-модель, и заполняем этими значениями столбцы

In [7]:
```
def fill_maker_model(df):
    maker_to_model = (
        df.dropna(subset=['maker', 'model'])
          .groupby('maker')['model']
          .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else 'unknown')
          .to_dict()
    )

    model_to_maker = (
        df.dropna(subset=['maker', 'model'])
          .groupby('model')['maker']
          .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else 'unknown')
          .to_dict()
    )

    most_common_pair = (
        df.dropna(subset=['maker', 'model'])
          .groupby(['maker', 'model'])
          .size()
          .sort_values(ascending=False)
          .idxmax()
    )

    mask_model_missing = df['model'].isna() & df['maker'].notna()
    df.loc[mask_model_missing, 'model'] = df.loc[mask_model_missing, 'maker'].map(
        lambda maker: maker_to_model.get(maker)
    )

    mask_maker_missing = df['maker'].isna() & df['model'].notna()
    df.loc[mask_maker_missing, 'maker'] = df.loc[mask_maker_missing, 'model'].map(
        lambda model: model_to_maker.get(model)
    )

    mask_both_missing = df['maker'].isna() & df['model'].isna()
    df.loc[mask_both_missing, 'maker'] = most_common_pair[0]
    df.loc[mask_both_missing, 'model'] = most_common_pair[1]

    return df
```

In [8]:
```
df = fill_maker_model(df)
```

Теперь, когда у каждой машины есть производитель и модель, заполним свойства автомобилей, которые те имеют с момента производства (год производства, количество дверей и т.д.), наиболее часто встречающимися по машинам того же производителя той же модели Если модель в единственном экземпляре в датасете, то наиболее часто встречающимися по производителю А если производитель в свою очередь в единственном экземпляре - то модой по всему столбцу

In [9]:
```
def fill_car_properties(df, columns_to_fill):
    maker_model_mode = (
        df.groupby(['maker', 'model'])[columns_to_fill]
          .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else x.iloc[0])
          .reset_index()
    )

    maker_mode = (
        df.groupby('maker')[columns_to_fill]
          .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else x.iloc[0])
          .reset_index()
    )
```

```
    global_mode = df[columns_to_fill].mode().iloc[0]

    df = df.merge(maker_model_mode, on=['maker', 'model'], how='left', suffixes=('', '_mode')
    df = df.merge(maker_mode, on='maker', how='left', suffixes=('', '_maker_mode'))

    for col in columns_to_fill:
        df[col] = df[col].fillna(df[f'{col}_mode'])
        df[col] = df[col].fillna(df[f'{col}_maker_mode'])
        df[col] = df[col].fillna(global_mode[col])

    df = df.drop(columns=[f'{col}_mode' for col in columns_to_fill] + [f'{col}_maker_mode' fo

    return df
```

In [10]:
```
properties_columns = ['manufacture_year', 'stk_year', 'door_count', 'seat_count',
                      'engine_power', 'body_type', 'transmission', 'fuel_type']

df = fill_car_properties(df, properties_columns)
```

Оставшиеся столбцы заполним просто модой для категориальных и средним для числовых

In [11]:
```
categorical_cols = df.select_dtypes(include=['object']).columns
numerical_cols = df.select_dtypes(include=['float64']).columns
```

In [12]:
```
cat_imputer = SimpleImputer(strategy='most_frequent')
df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
```

In [13]:
```
num_imputer = SimpleImputer(strategy='mean')
df[numerical_cols] = num_imputer.fit_transform(df[numerical_cols])
```

Переведем даты в ts (int64):

In [14]:
```
df['date_last_seen'] = pd.to_datetime(df['date_last_seen'], format='ISO8601').astype('int64')
df['date_created'] = pd.to_datetime(df['date_created'], format='ISO8601').astype('int64')
```

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3552912 entries, 0 to 3552911
Data columns (total 16 columns):
 #   Column               Dtype  
---  ------               -----  
 0   maker                object 
 1   model                object 
 2   mileage              float64
 3   manufacture_year     Int64  
 4   engine_displacement  float64
 5   engine_power         float64
 6   body_type            object 
 7   color_slug           object 
 8   stk_year             Int64  
 9   transmission         object 
 10  door_count           Int64  
 11  seat_count           Int64  
 12  fuel_type            object 
 13  date_created         int64  
 14  date_last_seen       int64  
 15  price_eur            float64
dtypes: Int64(4), float64(4), int64(2), object(6)
memory usage: 447.3+ MB
```

Закодируем категориальные признаки при помощи OHE:

In [16]:
```
df_sample = df.sample(frac=0.01, random_state=42)

categorical_cols = df_sample.select_dtypes(include=['object']).columns

preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_cols)
    ],
    remainder='passthrough'
)

df_transformed = preprocessor.fit_transform(df_sample)

encoded_columns = preprocessor.transformers_[0][1].get_feature_names_out(categorical_cols)

final_df = pd.DataFrame(df_transformed, columns=list(encoded_columns) + [col for col in df_sa
```

In [17]:
```
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
numerical_cols = numerical_cols.drop('price_eur')

scaler = StandardScaler()
final_df[numerical_cols] = scaler.fit_transform(final_df[numerical_cols])
```

In [18]:
```
final_df.head(10)
```

Out[18]:

	maker_audi	maker_bentley	maker_bmw	maker_chevrolet	maker_chrysler	maker_citroen	maker_d
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	
5	1.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	0.0	0.0	0.0	0.0	1.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	

10 rows × 740 columns

In [19]:
```
X = final_df.drop(columns=['price_eur'])
y = final_df['price_eur']
```

In [21]:
```
categorical_cols = X.select_dtypes(include=['object']).columns
```

In [22]:
```
X[categorical_cols] = X[categorical_cols].astype('int64')
y = y.astype('float')
```

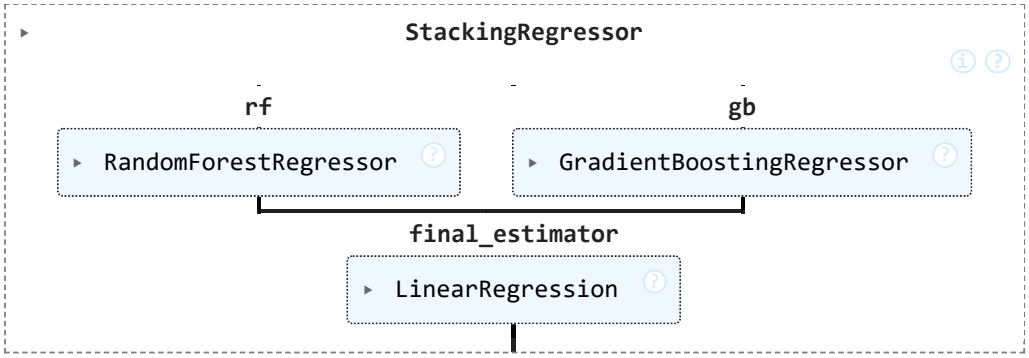
In [23]:
```
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=10)
X_selected = selector.fit_transform(X, y)
```

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_stat
```

In [25]:
```
stack_model = StackingRegressor(
    estimators=[
        ('rf', RandomForestRegressor(n_estimators=50, random_state=42)),
        ('gb', GradientBoostingRegressor(n_estimators=50, random_state=42))
    ],
    final_estimator=LinearRegression()
)
stack_model.fit(X_train, y_train)
```

Out[25]:



In [26]:

```
mlp = MLPRegressor(hidden_layer_sizes=(50, ), max_iter=2000, verbose = True, random_state=42,
mlp.fit(X_train, y_train)
```

```
Iteration 1, loss = 18367656740.09548950
Validation score: -0.427516
Iteration 2, loss = 18343366433.29713821
Validation score: -0.185694
Iteration 3, loss = 18304613455.67306900
Validation score: 0.092584
Iteration 4, loss = 18269312115.70410156
Validation score: 0.268786
Iteration 5, loss = 18246877869.32195663
Validation score: 0.358536
Iteration 6, loss = 18233953232.92250824
Validation score: 0.404291
Iteration 7, loss = 18225328522.12657166
Validation score: 0.431483
Iteration 8, loss = 18219603333.62150192
Validation score: 0.449229
Iteration 9, loss = 18215304969.45560074
Validation score: 0.459470
Iteration 10, loss = 18211764635.41940689
Validation score: 0.468554
Iteration 11, loss = 18208741951.88881302
Validation score: 0.476989
Iteration 12, loss = 18206069719.82094955
Validation score: 0.486078
Iteration 13, loss = 18203626432.36535263
Validation score: 0.492557
Iteration 14, loss = 18201270702.27122498
Validation score: 0.500033
Iteration 15, loss = 18199112629.58381653
Validation score: 0.506491
Iteration 16, loss = 18196990925.88211441
Validation score: 0.512900
Iteration 17, loss = 18194998461.06058502
Validation score: 0.517453
Iteration 18, loss = 18193063957.46667480
Validation score: 0.523425
Iteration 19, loss = 18191248564.93579865
Validation score: 0.529014
Iteration 20, loss = 18189492647.34416580
Validation score: 0.532362
Iteration 21, loss = 18187691125.10905075
Validation score: 0.537519
Iteration 22, loss = 18186105647.56687546
Validation score: 0.547281
Iteration 23, loss = 18184664860.37251663
Validation score: 0.542569
Iteration 24, loss = 18182959613.62076187
Validation score: 0.547311
Iteration 25, loss = 18181538590.02111053
Validation score: 0.544859
Iteration 26, loss = 18180095486.68863297
Validation score: 0.550383
Iteration 27, loss = 18178576867.03490829
Validation score: 0.546697
Iteration 28, loss = 18177291292.05794144
Validation score: 0.548741
Iteration 29, loss = 18175995165.81284714
Validation score: 0.551202
Iteration 30, loss = 18174625293.94470596
Validation score: 0.548451
Iteration 31, loss = 18173521455.94356155
Validation score: 0.548734
```

```
Iteration 32, loss = 18172264243.36838150
Validation score: 0.550418
Iteration 33, loss = 18171046539.33911896
Validation score: 0.549376
Iteration 34, loss = 18169835215.56095505
Validation score: 0.548564
Iteration 35, loss = 18168649252.38023758
Validation score: 0.551318
Iteration 36, loss = 18167574499.83300400
Validation score: 0.548070
Iteration 37, loss = 18166385300.25009155
Validation score: 0.547183
Iteration 38, loss = 18165357379.31843567
Validation score: 0.555605
Iteration 39, loss = 18164599493.84476471
Validation score: 0.544320
Iteration 40, loss = 18163033281.75092316
Validation score: 0.543816
Iteration 41, loss = 18161834222.80759430
Validation score: 0.545034
Iteration 42, loss = 18160845258.32625198
Validation score: 0.543804
Iteration 43, loss = 18159673839.31465530
Validation score: 0.541723
Iteration 44, loss = 18158723185.61074829
Validation score: 0.539469
Iteration 45, loss = 18157609341.42711258
Validation score: 0.536324
Iteration 46, loss = 18156507810.85210419
Validation score: 0.538208
Iteration 47, loss = 18155469154.72999954
Validation score: 0.534962
Iteration 48, loss = 18154426921.82211304
Validation score: 0.537207
Iteration 49, loss = 18153300472.99053192
Validation score: 0.537617
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

Out[26]:

MLPRegressor

```
MLPRegressor(early_stopping=True, hidden_layer_sizes=(50,),
             learning_rate='adaptive', learning_rate_init=0.02, max_iter=2000,
             random_state=42, verbose=True)
```

In [30]:

```python
gmdh_linear = Combi()
gmdh_nonlinear = Mia()

gmdh_linear.fit(X_train, y_train)
gmdh_nonlinear.fit(X_train, y_train)
```

Out[30]:

```
<gmdh.gmdh.Mia at 0x29cb97fc990>
```

In [42]:

```python
def evaluate(model, name):
    y_pred = model.predict(X_test)
    print(f"{name}:\n"
    f"  MAE:  {mean_absolute_error(y_test, y_pred):.4f}\n"
    f"  RMSE: {mean_squared_error(y_test, y_pred)**0.5:.4f}\n"
    f"  R²:   {r2_score(y_test, y_pred):.4f}\n")
```

In [43]:
```
evaluate(stack_model, "Stack")
evaluate(mlp, "MLP")
evaluate(gmdh_linear, "COMBI")
evaluate(gmdh_nonlinear, "MIA")
```

```
Stack:
  MAE:  17602.3924
  RMSE: 423945.3950
  R²:   -0.0002

MLP:
  MAE:  13407.4461
  RMSE: 423899.5553
  R²:   0.0000

COMBI:
  MAE:  16488.0649
  RMSE: 423841.9025
  R²:   0.0003

MIA:
  MAE:  15749.1855
  RMSE: 423758.3399
  R²:   0.0007
```