



**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 лабораторным работам №2-4**

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

Подпись:

Проверил:
Гапанюк Ю.Е.
Подпись:

2025 г.

```
In [26]: 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.tree import DecisionTreeClassifier
from tqdm import tqdm
```

Лаба 2, 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/Лаба1/all_anonymized

C:\Users\dielo\AppData\Local\Temp\ipykernel_20880\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/Лаба1/all_anonymize
d_2015_11_2017_03.csv")
```

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

```
Out[3]:
```

	maker	model	mileage	manufacture_year	engine_displacement	engine_power	body_type
3444530	land-rover	NaN	234000.0	2007.0	6200.0	112.0	other
2887531	bmw	x6	1.0	2016.0	2993.0	230.0	other
476748	NaN	NaN	NaN	NaN	NaN	NaN	compact
2187923	skoda	fabia	NaN	2007.0	1198.0	51.0	other
1305204	audi	a3	238500.0	2004.0	NaN	NaN	NaN

```
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 соответствовать 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	0.0	1.0
7	0.0	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	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 740 columns

Лаба 3, обучение модели ближайших соседей

In [19]: `final_df['price_category'] = pd.qcut(final_df['price_eur'], q=3, labels=[0, 1, 2])`

In [20]: `X = final_df.drop(columns=['price_category', 'price_eur'])`
`y = final_df['price_category']`

In [21]: `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`

Обучение с параметром K=5

In [22]: `import os`
`os.environ["LOKY_MAX_CPU_COUNT"] = "8"`

In [23]: `knn = KNeighborsClassifier(n_neighbors=5)`
`knn.fit(X_train, y_train)`
`y_pred = knn.predict(X_test)`

`print("Метрики для модели с K=5:")`
`print(classification_report(y_test, y_pred))`

Метрики для модели с K=5:

	precision	recall	f1-score	support
0	0.84	0.83	0.83	2334
1	0.68	0.70	0.69	2390
2	0.82	0.81	0.82	2382
accuracy			0.78	7106
macro avg	0.78	0.78	0.78	7106
weighted avg	0.78	0.78	0.78	7106

```
In [31]: param_grid = {'n_neighbors': range(1, 10)}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=KFold(n_splits=3), scoring=
    verbose=10 )
grid_search.fit(X_train, y_train)
best_k_grid = grid_search.best_params_['n_neighbors']

print(f"Лучшее K по GridSearchCV: {best_k_grid}")
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

C:\Users\dielo\AppData\Local\Programs\Python\Python313\Lib\site-packages\joblib\externals\loky\process_executor.py:752: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.

```
warnings.warn(
```

Лучшее K по GridSearchCV: 5

```
In [32]: knn_best_gs = KNeighborsClassifier(n_neighbors=best_k_grid)
knn_best_gs.fit(X_train, y_train)
y_pred_best_gs = knn_best_gs.predict(X_test)

print("Метрики для модели с оптимальным K по GridSearchCV:")
print(classification_report(y_test, y_pred_best_gs))
```

Метрики для модели с оптимальным K по GridSearchCV:

	precision	recall	f1-score	support
0	0.84	0.83	0.83	2334
1	0.68	0.70	0.69	2390
2	0.82	0.81	0.82	2382
accuracy			0.78	7106
macro avg	0.78	0.78	0.78	7106
weighted avg	0.78	0.78	0.78	7106

```
In [28]: random_search = RandomizedSearchCV(KNeighborsClassifier(), param_distributions=param_grid, n_
    verbose=10 )
random_search.fit(X_train, y_train)
best_k_random = random_search.best_params_['n_neighbors']

print(f"Лучшее K по RandomizedSearchCV: {best_k_random}")
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

Лучшее K по RandomizedSearchCV: 1

```
In [29]: knn_best_r = KNeighborsClassifier(n_neighbors=best_k_random)
knn_best_r.fit(X_train, y_train)
y_pred_best_r = knn_best_r.predict(X_test)

print("Метрики для модели с оптимальным K по RandomizedSearchCV:")
print(classification_report(y_test, y_pred_best_r))
```

Метрики для модели с оптимальным K по RandomizedSearchCV:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	2334
1	0.69	0.70	0.70	2390
2	0.82	0.83	0.82	2382
accuracy			0.78	7106
macro avg	0.79	0.78	0.79	7106
weighted avg	0.79	0.78	0.78	7106

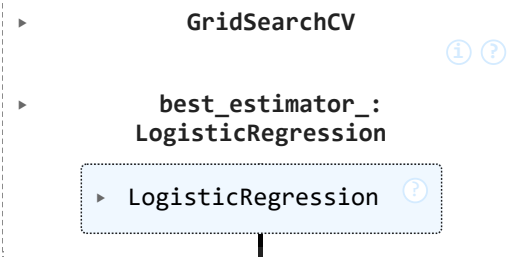
Лаба 4, линейные модели, SVM, деревья решений

```
In [34]: param_log_reg = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['lbfgs']
}

log_reg = LogisticRegression(max_iter=1000)
grid_log = GridSearchCV(log_reg, param_log_reg, cv=3, scoring='f1_weighted', n_jobs=-1, verbose=1)
grid_log.fit(X_train, y_train)
```

Fitting 3 folds for each of 4 candidates, totalling 12 fits

```
Out[34]:
```



```
In [ ]: param_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

svm = SVC()
grid_svm = GridSearchCV(svm, param_svm, cv=3, scoring='f1_weighted', n_jobs=-1, verbose = 10)
grid_svm.fit(X_train, y_train)
```

Fitting 3 folds for each of 12 candidates, totalling 36 fits

```
In [ ]: param_tree = {
    'max_depth': [3, 5, 10, 20],
    'min_samples_split': [2, 5, 10]
}

tree = DecisionTreeClassifier(random_state=42)
grid_tree = GridSearchCV(tree, param_tree, cv=3, scoring='f1_weighted', n_jobs=-1, verbose = 1)
grid_tree.fit(X_train, y_train)
```

```
In [ ]: best_models = {
    'Logistic Regression': grid_log.best_estimator_,
    'SVM': grid_svm.best_estimator_,
    'Decision Tree': grid_tree.best_estimator_
}

for name, model in best_models.items():
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{name} (Best): Accuracy = {acc:.4f}, F1-score = {f1:.4f}")
```

