```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV,
KFold, cross_val_score, cross_validate
from sklearn.svm import SVR
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_percentage_error, mean_absolute_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout,
BatchNormalization, Activation
from tensorflow.keras.models import Sequential


df =
pd.read_excel('/content/drive/MyDrive/DatasetsVKR/df_norm.xlsx').drop(
['Unnamed: 0'], axis = 1)
print(df.shape)
df.info()
```

```
(922, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 922 entries, 0 to 921
Data columns (total 13 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Соотношение матрица-наполнитель       922 non-null    float64
 1   Плотность, кг/м3                      922 non-null    float64
 2   Модуль упругости, ГПа                 922 non-null    float64
 3   Количество отвердителя, м.%           922 non-null    float64
 4   Содержание эпоксидных групп,%_2       922 non-null    float64
 5   Температура вспышки, С_2              922 non-null    float64
 6   Поверхностная плотность, г/м2         922 non-null    float64
 7   Модуль упругости при растяжении, ГПа  922 non-null    float64
 8   Прочность при растяжении, МПа         922 non-null    float64
 9   Потребление смолы, г/м2               922 non-null    float64
```

```
 10  Угол нашивки                                922 non-null    int64
 11  Шаг нашивки                                 922 non-null    float64
 12  Плотность нашивки                           922 non-null    float64
dtypes: float64(12), int64(1)
memory usage: 93.8 KB
```

```
# в соответствии с заданием ВКР необходимо обучить модели для прогноза
параметров модуля упругости при растяжении и прочности при растяжении
df
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 922,\n \"fields\": [\n   {\n   \"column\": \"\\u0421\\u043e\\u043e\\u0442\\u043d\\u043e\\u0448\\u0435\\u043d\\u0438\\u0435 \\u043c\\u0430\\u0442\\u0440\\u0438\\u0446\\u0430-\\u043d\\u0430\\u043f\\u043e\\u043b\\u043d\\u0438\\u0442\\u0435\\u043b\\u044c\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.8954717320562058,\n   \"min\": 0.547391007365624,\n   \"max\": 5.3141436851035,\n   \"num_unique_values\": 916,\n   \"samples\": [\n   2.71722263251102,\n   0.968086542252719,\n   3.72306752097199\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\u041f\\u043b\\u043e\\u0442\\u043d\\u043e\\u0441\\u0442\\u044c, \\u043a\\u0433/\\u043c3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 71.04064831235215,\n   \"min\": 1784.48224524858,\n   \"max\": 2161.56521646388,\n   \"num_unique_values\": 915,\n   \"samples\": [\n   2026.11047511909,\n   1899.37285197669,\n   1986.9445606882\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\u041c\\u043e\\u0434\\u0443\\u043b\\u044c \\u0443\\u043f\\u0440\\u0443\\u0433\\u043e\\u0441\\u0442\\u0438, \\u0413\\u041f\\u0430\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 327.60700751220486,\n   \"min\": 2.4369087535075,\n   \"max\": 1628.0,\n   \"num_unique_values\": 921,\n   \"samples\": [\n   718.492637623506,\n   1001.28579276492,\n   597.14151688518\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\u041a\\u043e\\u043b\\u0438\\u0447\\u0435\\u0441\\u0442\\u0432\\u043e \\u043e\\u0442\\u0432\\u0435\\u0440\\u0434\\u0438\\u0442\\u0435\\u043b\\u044f, \\u043c.%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 26.753227905108954,\n   \"min\": 38.6685003343557,\n   \"max\": 181.82844779488,\n   \"num_unique_values\": 905,\n   \"samples\": [\n   126.919151502633,\n   143.535238468246,\n   57.9497800545227\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\u0421\\u043e\\u0434\\u0435\\u0440\\u0436\\u0430\\u043d\\u0438\\u0435 \\u044d\\u043f\\u043e\\u043a\\u0441\\u0438\\u0434\\u043d\\u044b\\u0445 \\u0433\\u0440\\u0443\\u043f\\u043f,%_2\",\n   \"properties\": {\n

\"dtype\": \"number\",\n   \"std\": 2.3939260328968763,\n
\"min\": 15.6958938036288,\n   \"max\": 28.9550943746499,\n
\"num_unique_values\": 905,\n   \"samples\": [\n
21.2453708727899,\n   22.0241086190884,\n
25.3902830373374\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u0422\\u0435\\u043c\\u043f\\u0435\\u0440\\u0430\\u0442\\u0443\\u0440\\
u0430 \\u0432\\u0441\\u043f\\u044b\\u0448\\u043a\\u0438, \\u0421_2\",\
n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 39.42076363132895,\n   \"min\": 179.37439137039,\n
\"max\": 386.067991779505,\n   \"num_unique_values\": 904,\n
\"samples\": [\n   206.005127700645,\n
248.132367449544,\n   271.126312461753\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"\\u041f\\u043e\\u0432\\u0435\\
u0440\\u0445\\u043d\\u043e\\u0441\\u0442\\u043d\\u0430\\u044f \\
u043f\\u043b\\u043e\\u0442\\u043d\\u043e\\u0441\\u0442\\u044c, \\
u0433/\\u043c2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 280.437328786617,\n   \"min\":
0.603739925153945,\n   \"max\": 1291.34011463545,\n
\"num_unique_values\": 906,\n   \"samples\": [\n
660.247949897597,\n   384.207155178403,\n
334.700978120372\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u041c\\u043e\\u0434\\u0443\\u043b\\u044c \\u0443\\u043f\\u0440\\
u0443\\u0433\\u043e\\u0441\\u0442\\u0438 \\u043f\\u0440\\u0438 \\
u0440\\u0430\\u0441\\u0442\\u044f\\u0436\\u0435\\u043d\\u0438\\u0438,
\\u0413\\u041f\\u0430\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 3.0258643883622773,\n   \"min\":
65.7938449666054,\n   \"max\": 81.203146720828,\n
\"num_unique_values\": 906,\n   \"samples\": [\n
72.3662210329381,\n   70.916879218537,\n
78.0716510274565\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u041f\\u0440\\u043e\\u0447\\u043d\\u043e\\u0441\\u0442\\u044c \\
u043f\\u0440\\u0438 \\u0440\\u0430\\u0441\\u0442\\u044f\\u0436\\
u0435\\u043d\\u0438\\u0438, \\u041c\\u041f\\u0430\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
453.5647335545092,\n   \"min\": 1250.39280220501,\n
\"max\": 3654.43435901371,\n   \"num_unique_values\": 906,\n
\"samples\": [\n   2618.95064622698,\n
3636.8929917828,\n   2174.61538963401\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"\\u041f\\u043e\\u0442\\u0440\\
u0435\\u0431\\u043b\\u0435\\u043d\\u0438\\u0435 \\u0441\\u043c\\
u043e\\u043b\\u044b, \\u0433/\\u043c2\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 57.13747497853371,\n
\"min\": 72.5308733761696,\n   \"max\": 359.052219789673,\n
\"num_unique_values\": 905,\n   \"samples\": [\n

```
208.929445658,\n          239.457167408696,\n
227.924477322423\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0423\\u0433\\u043e\\u043b \\u043d\\u0430\\u0448\\u0438\\u0432\\
u043a\\u0438\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"\\u0428\\u0430\\u0433 \\u043d\\u0430\\u0448\\u0438\\
u0432\\u043a\\u0438\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.5141838419664064,\n        \"min\":
0.0376389366987437,\n        \"max\": 13.732404403383,\n
\"num_unique_values\": 891,\n        \"samples\": [\n
10.0671231646161,\n          6.80636570246087\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u041f\\u043b\\u043e\\u0442\\
u043d\\u043e\\u0441\\u0442\\u044c \\u043d\\u0430\\u0448\\u0438\\
u0432\\u043a\\u0438\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.122204405356028,\n        \"min\":
28.6616316278123,\n        \"max\": 86.0124270098611,\n
\"num_unique_values\": 890,\n        \"samples\": [\n
77.2777281396634,\n          74.8856664463349\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


Анализ данных показал отсутствие линейных зависимостей между переменными, что подтверждается низкими значениями коэффициентов корреляции.

С учетом характеристик исходных данных мы имеем три целевых признака которые выражены следующим образом: модуль упругости при растяжении и прочность при растяжении зависят от свойств матрицы, наполнителя и параметров процесса, а соотношение матрица-наполнитель определяется характеристиками матрицы, наполнителя и свойствами конечного композита. В связи с этим необходимо построить предсказывающие модели для первых двух признаков и разработать нейронную сеть для прогнозирования последнего.

Перед тем как передавать данные в модель, их необходимо привести к удобному формату. Для этого выполняем предобработку, учитывая особенности разных типов признаков.

Признак угла нашивки после преобразования дискретные значения, поэтому для их кодирования используем OrdinalEncoder.

Для остальных - числовых переменных - выполняем стандартизацию с помощью StandardScaler. Этот метод приводит данные к нормальному распределению со средним значением 0 и стандартным отклонением 1, что улучшает сходимость модели.

```
# деление датасетов по целевому признаку для каждой из задач

# для признака "Модуль упругости при растяжении"
```

```python
y1_columns = ['Модуль упругости при растяжении, ГПа']
x1_columns = [col for col in df.columns if col not in y1_columns]

y1 = df.loc[:, y1_columns]
x1 = df.loc[:, x1_columns]

# для признака "Прочность при растяжении"
y2_columns = ['Прочность при растяжении, МПа']
x2_columns = [col for col in df.columns if col not in y2_columns]

y2 = df.loc[:, y2_columns]
x2 = df.loc[:, x2_columns]

# для признака "Соотношение матрица-наполнитель"
y3_columns = ['Соотношение матрица-наполнитель']
x3_columns = [col for col in df.columns if col not in y3_columns]

y3 = df.loc[:, y3_columns]
x3 = df.loc[:, x3_columns]

# определяем категориальные и числовые признаки

categorial_feature = ['Угол нашивки']
num_features_x1 = list(set(x1_columns) - set(categorial_feature))
num_features_x2 = list(set(x2_columns) - set(categorial_feature))
num_features_x3 = list(set(x3_columns) - set(categorial_feature))

# создаем препроцессоры для разных задач

preproc_1 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x1),
        ("encode_categorical", OrdinalEncoder(), categorial_feature)
    ]
)

preproc_2 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x2),
        ("encode_categorical", OrdinalEncoder(), categorial_feature)
    ]
)

preproc_3 = ColumnTransformer(
    transformers=[
        ("scale_numeric", StandardScaler(), num_features_x3),
        ("encode_categorical", OrdinalEncoder(), categorial_feature)
    ]
)
```

Определим вспомогательные функции, с помощью которых можно будет сравнить метрики различных моделей

```
# перечень моделей, на которых будет проходить обучение

models = {
    'Dummy Regressor': DummyRegressor(strategy='mean'),
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'SVR': SVR(),
    'KNN': KNeighborsRegressor(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(random_state=42)
}

# функция для оценки моделей с базовыми параметрам с использованием
кросс-валидации

def evaluate_models(models, features, target):

    results = pd.DataFrame(columns=['R2', 'RMSE', 'MAE', 'MAPE'])

    # настройка кросс-валидации
    cv = KFold(n_splits=10, shuffle=True, random_state=42)

    # Определяем метрики для оценки
    metrics = {
        'R2': 'r2',
        'RMSE': 'neg_root_mean_squared_error',
        'MAE': 'neg_mean_absolute_error',
        'MAPE': 'neg_mean_absolute_percentage_error',
    }

    # проходим по каждой модели
    for model_name, model in models.items():

        # выполняем кросс-валидацию
        cv_results = cross_validate(model, features, target, cv=cv,
scoring=list(metrics.values()))

        # сохраняем средние значения метрик
        results.loc[model_name, 'R2'] = cv_results['test_r2'].mean()
        results.loc[model_name, 'RMSE'] = -
cv_results['test_neg_root_mean_squared_error'].mean()
        results.loc[model_name, 'MAE'] = -
cv_results['test_neg_mean_absolute_error'].mean()
        results.loc[model_name, 'MAPE'] = -
cv_results['test_neg_mean_absolute_percentage_error'].mean()
```

```
    return results

# функция для поиска оптимальных параметров моделей

def grid_search(model, params, x, y):
    pd.options.display.max_colwidth = 100 # чтобы полностью отобразить
оптимальные параметры при выводе

    results = pd.DataFrame()

    cv = KFold(10, shuffle=True, random_state=42)

    scoring = 'neg_root_mean_squared_error'
    searcher = GridSearchCV(model, params, cv=cv, scoring=scoring)

    searcher.fit(x, y)

    results.loc[:, 'best parameters'] = searcher.cv_results_['params']
    results.loc[:, 'RMSE'] = -searcher.cv_results_['mean_test_score']
    results.loc[:, 'rank'] = searcher.cv_results_['rank_test_score']

    return results, searcher.best_estimator_

# расчет метрик качества предсказания модели

def calculate_metrics(model_name, true_values, predicted_values):

    results = pd.DataFrame(index=[model_name], columns=['R2', 'RMSE',
'MAE', 'MAPE'])

    results.loc[model_name, 'R2'] = metrics.r2_score(true_values,
predicted_values)
    mse = metrics.mean_squared_error(true_values, predicted_values)
    results.loc[model_name, "RMSE"] = np.sqrt(mse)
    # results.loc[model_name, 'RMSE'] =
metrics.mean_squared_error(true_values, predicted_values,
squared=False)
    results.loc[model_name, 'MAE'] =
metrics.mean_absolute_error(true_values, predicted_values)
    results.loc[model_name, 'MAPE'] =
metrics.mean_absolute_percentage_error(true_values, predicted_values)

    return results

# Функция применяет стилизацию к DataFrame:
# Минимальные RMSE, MAE, MAPE — зеленым
# Максимальное R² — зеленым
# Максимальные RMSE, MAE, MAPE — синим
# Минимальный R² — синим
```

```
def style_model_results(df):
    return (df.style
            .highlight_min(subset=['RMSE', 'MAE', 'MAPE'], 
color="green")
            .highlight_max(subset=['R2'], color="green")
            .highlight_max(subset=['RMSE', 'MAE', 'MAPE'], 
color="blue")
            .highlight_min(subset=['R2'], color="blue"))
```

1ый целевой параметр - модуль упругости при растяжении

```
# разделяем выборки
x1_train_initial, x1_test_initial, y1_train, y1_test = 
train_test_split(x1, y1, test_size=0.3, random_state=42)

# преобразуем целевой параметр в массив
y1_train = y1_train['Модуль упругости при растяжении, ГПа'].values
y1_test = y1_test['Модуль упругости при растяжении, ГПа'].values

# препроцессинг
x1_train = preproc_1.fit_transform(x1_train_initial)
x1_test = preproc_1.transform(x1_test_initial)

results_1 = evaluate_models(models, x1_train, y1_train)
styled_results_1 = style_model_results(results_1)
styled_results_1

<pandas.io.formats.style.Styler at 0x7a3105f83690>
```

Как можно видеть из результатов работы моделей коэффициент детерминации во всех моделях почти равен нулю (все они приняли отрицательное значение), то есть все они (за исключением лассо) показали себя хуже базовой модели.

```
# создаем dict с лучшими параметрами моделей
GS_best_models_1 = {}

# для обычной линейной регрессии нет возможности для перебора 
параметров, поэтому ее в данном случае не рассматриваем
# лучшие параметры для модели Ridge

params_1 = {
    'alpha': range(1, 10**6, 5000),
    'fit_intercept': [True, False],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sag', 'saga']
}
```

```
search, best_model = grid_search(Ridge(), params_1, x1_train,
y1_train)

# сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 2.9597817960771127,\n   \"max\": 2.9597817960771127,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   2.9597817960771127\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для модели Lasso

params_1 = {
    'alpha': [0.001, 0.01, 0.1, 0.05, 0.15, 0.2, 0.095, 1],
    'fit_intercept': [True, False],

}

search, best_model = grid_search(Lasso(), params_1, x1_train,
y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 2.959775640804863,\n   \"max\": 2.959775640804863,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   2.959775640804863\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для модели градиентного бустинга

params_1 = {
```

```
    'n_estimators': [5, 10, 25],
    'learning_rate': [0.05, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.5, 1.0]
}

search, best_model = grid_search(GradientBoostingRegressor(),
params_1, x1_train, y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 2.9594411022787837,\n   \"max\": 2.9594411022787837,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   2.9594411022787837\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для метода опорных векторов

params_1 = {
    'C': [0.001, 0.01, 0.05],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}


search, best_model = grid_search(SVR(), params_1, x1_train, y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 2.963759912160327,\n   \"max\": 2.963759912160327,\n

\"num_unique_values\": 1,\n   \"samples\": [\n
2.963759912160327\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для KNN

params_1 = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev'],
}

search, best_model = grid_search(KNeighborsRegressor(), params_1,
x1_train, y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\":
[\n   {\n   \"column\": \"best parameters\",\n
\"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
3.0574789069866073,\n   \"max\": 3.0574789069866073,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
3.0574789069866073\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для Decision Tree

params_1 = {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['squared_error', 'absolute_error', 'friedman_mse']
}

search, best_model = grid_search(DecisionTreeRegressor(), params_1,
x1_train, y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 3.073318067361942,\n   \"max\": 3.073318067361942,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   3.073318067361942\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для Random Forest

params_1 = {
    'n_estimators': [5, 8, 10],
    'max_depth': [3, 4, 6],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [2, 5, 7],
    'criterion': ['squared_error', 'absolute_error'],
    'bootstrap': [True, False]
}

search, best_model = grid_search(RandomForestRegressor(), params_1,
x1_train, y1_train)

# Сохранение лучшей модели в словарь
GS_best_models_1[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"best parameters\",\n   \"properties\": {\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 2.990872777822763,\n   \"max\": 2.990872777822763,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   2.990872777822763\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
mod_results_1 = evaluate_models(GS_best_models_1, x1_train, y1_train)
styled_mod_results_1 = style_model_results(mod_results_1)
styled_mod_results_1
```

```
<pandas.io.formats.style.Styler at 0x7a30f1d014d0>
```

```
best_model_1 = Lasso(
     alpha = 1
)
```

```
best_model_1.fit(x1_train, y1_train)
y1_best = best_model_1.predict(x1_test)

base_model_1 = DummyRegressor(strategy='mean')
base_model_1.fit(x1_train, y1_train)
y1_dummy_predicted = base_model_1.predict(x1_test)

diff_stats_1 = calculate_metrics('Базовая модель', y1_test,
y1_dummy_predicted)
diff_stats_1 = pd.concat([diff_stats_1, calculate_metrics('Лучшая
модель (Lasso)', y1_test, y1_best)], ignore_index=False)
diff_stats_1
```


```
{"summary":"{\n  \"name\": \"diff_stats_1\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"R2\",\n      \"properties\":
{\n        \"dtype\": \"date\",\n        \"min\": -
0.008878670806033773,\n        \"max\": -0.008878670806033773,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          -
0.008878670806033773\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RMSE\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 3.162017010185909,\n        \"max\": 3.162017010185909,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
3.162017010185909\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MAE\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 2.580192902566283,\n        \"max\": 2.580192902566283,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
2.580192902566283\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MAPE\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": 0.03503243786306619,\n        \"max\": 0.03503243786306619,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.03503243786306619\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"diff_stats_1"}
```


2ой целевой параметр - прочность при растяжении

```
# разделяем выборки
x2_train_initial, x2_test_initial, y2_train, y2_test =
train_test_split(x2, y2, test_size=0.3, random_state=42)

# преобразуем целевой параметр в массив
```

```python
y2_train = y2_train['Прочность при растяжении, МПа'].values
y2_test = y2_test['Прочность при растяжении, МПа'].values

# препроцессинг
x2_train = preproc_2.fit_transform(x2_train_initial)
x2_test = preproc_2.transform(x2_test_initial)

results_2 = evaluate_models(models, x2_train, y2_train)
styled_results_2 = style_model_results(results_2)
styled_results_2
```

```
<pandas.io.formats.style.Styler at 0x7a30e272c290>
```

```python
GS_best_models_2 = {}

# лучшие параметры для модели Ridge

params_2 = {
    'alpha': range(1, 10**6, 5000),
    'fit_intercept': [True, False],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sag', 'saga']
}

search, best_model = grid_search(Ridge(), params_2, x2_train,
y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


```
{"summary":"{\n  \"name\": \"search\",\n  \"rows\": 1,\n  \"fields\":
[\n    {\n      \"column\": \"best parameters\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RMSE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
448.5449034255527,\n        \"max\": 448.5449034255527,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
448.5449034255527\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```python
# лучшие параметры для модели Lasso

params_2 = {
    'alpha': [0.001, 0.01, 0.1, 0.05, 0.15, 0.2, 0.095, 1],
    'fit_intercept': [True, False],

}
```

```
search, best_model = grid_search(Lasso(), params_2, x2_train,
y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


```
{"summary":"{\n  \"name\": \"search\",\n  \"rows\": 1,\n  \"fields\":
[\n    {\n      \"column\": \"best parameters\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RMSE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
454.5899484746441,\n        \"max\": 454.5899484746441,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
454.5899484746441\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
# лучшие параметры для модели градиентного бустинга

params_2 = {
    'n_estimators': [5, 10, 25],
    'learning_rate': [0.05, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.5, 1.0]
}

search, best_model = grid_search(GradientBoostingRegressor(),
params_2, x2_train, y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


```
{"summary":"{\n  \"name\": \"search\",\n  \"rows\": 1,\n  \"fields\":
[\n    {\n      \"column\": \"best parameters\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RMSE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
447.24428094318154,\n        \"max\": 447.24428094318154,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
447.24428094318154\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
# лучшие параметры для метода опорных векторов

params_2 = {
    'C': [0.001, 0.01, 0.05],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}


search, best_model = grid_search(SVR(), params_2, x2_train, y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\":
[\n   {\n   \"column\": \"best parameters\",\n
\"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
448.4229056245975,\n   \"max\": 448.4229056245975,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
448.4229056245975\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}


```
# лучшие параметры для KNN

params_2 = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev'],
}

search, best_model = grid_search(KNeighborsRegressor(), params_2,
x2_train, y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\":
[\n   {\n   \"column\": \"best parameters\",\n
\"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
462.71324914626456,\n   \"max\": 462.71324914626456,\n

\"num_unique_values\": 1,\n   \"samples\": [\n
462.71324914626456\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для Decision Tree

params_2 = {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['squared_error', 'absolute_error', 'friedman_mse']
}

search, best_model = grid_search(DecisionTreeRegressor(), params_2,
x2_train, y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model

# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```

{"summary":"{\n \"name\": \"search\",\n \"rows\": 1,\n \"fields\":
[\n   {\n   \"column\": \"best parameters\",\n
\"properties\": {\n   \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
455.14233517603463,\n   \"max\": 455.14233517603463,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
455.14233517603463\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# лучшие параметры для Random Forest

params_2 = {
    'n_estimators': [5, 8, 10],
    'max_depth': [3, 4, 6],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [2, 5, 7],
    'criterion': ['squared_error', 'absolute_error'],
    'bootstrap': [True, False]
}

search, best_model = grid_search(RandomForestRegressor(), params_2,
x2_train, y2_train)

# Сохранение лучшей модели в словарь
GS_best_models_2[str(best_model)] = best_model
```

```
# Вывод результатов для лучшей модели
search.loc[search['rank'] == 1, ['best parameters', 'RMSE']]
```


```
{"summary":"{\n  \"name\": \"search\",\n  \"rows\": 1,\n  \"fields\":
[\n    {\n      \"column\": \"best parameters\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"RMSE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
449.9209798191905,\n        \"max\": 449.9209798191905,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
449.9209798191905\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
mod_results_2 = evaluate_models(GS_best_models_2, x2_train, y2_train)
styled_mod_results_2 = style_model_results(mod_results_2)
styled_mod_results_2
```

```
<pandas.io.formats.style.Styler at 0x7a3158f90550>
```

```
best_model_2 = GradientBoostingRegressor(
     n_estimators = 5,
     learning_rate = 0.05,
     min_samples_split = 7,
     min_samples_leaf = 4,
     subsample = 0.5,
     max_depth = 4
)

best_model_2.fit(x2_train, y2_train)
y2_best = best_model_2.predict(x2_test)

base_model_2 = DummyRegressor(strategy='mean')
base_model_2.fit(x2_train, y2_train)
y2_dummy_predicted = base_model_2.predict(x2_test)

diff_stats_2 = calculate_metrics('Базовая модель', y2_test,
y2_dummy_predicted)
diff_stats_2 = pd.concat([diff_stats_2, calculate_metrics('Лучшая
модель (GradientBoostingRegressor)', y2_test, y2_best)],
ignore_index=False)
styled_diff_stats_2 = style_model_results(diff_stats_2)
styled_diff_stats_2
```

```
<pandas.io.formats.style.Styler at 0x7a30e8115e90>
```

3ая целевая метрика - соотношение матрица-наполнитель

В соответствии с заданием необходимо создать нейронную сеть, рекомендующую значение данного параметра

Многослойный перцептрон (MLPRegressor)

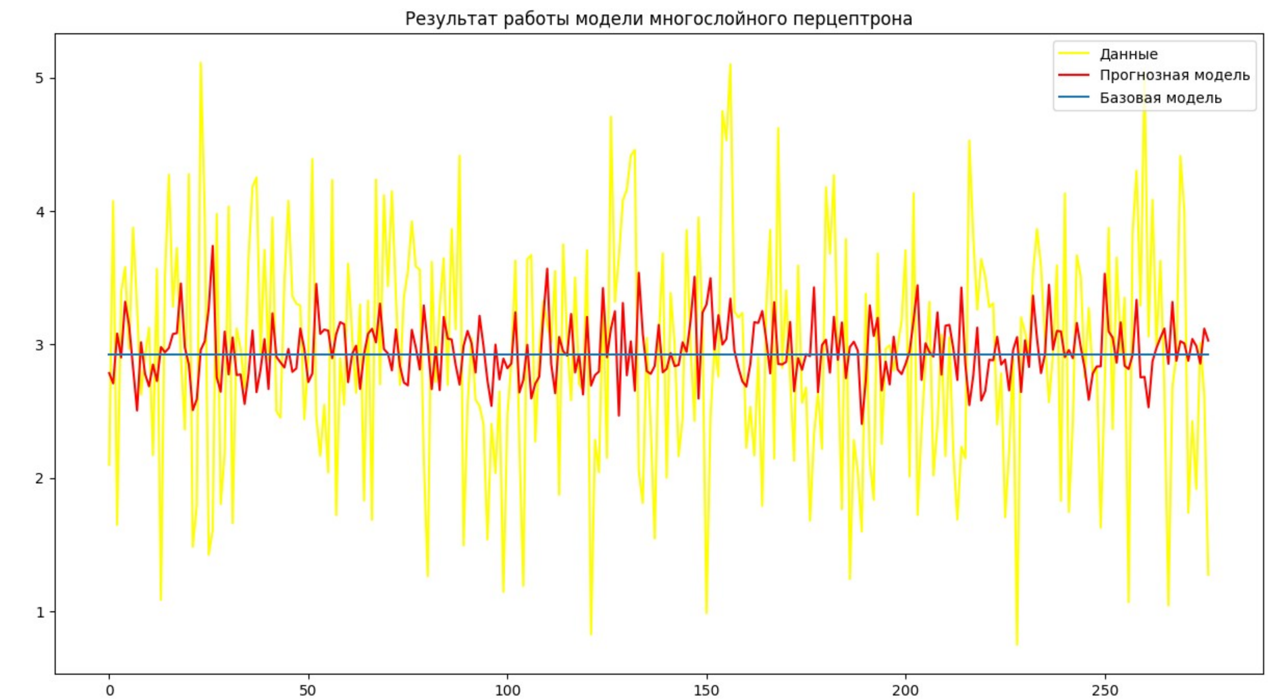
```
x3_train_initial, x3_test_initial, y3_train, y3_test =
train_test_split(x3, y3, test_size=0.3, random_state=42)
y3_train = y3_train['Соотношение матрица-наполнитель'].values
y3_test = y3_test['Соотношение матрица-наполнитель'].values

# препроцессинг
x3_train = preproc_3.fit_transform(x3_train_initial)
x3_test = preproc_3.transform(x3_test_initial)

base_3 = DummyRegressor(strategy='mean')
base_3.fit(x3_train, y3_train)
y3_dummy = base_3.predict(x3_test)

# модель многослойного перцептрона
mlp = MLPRegressor(
              hidden_layer_sizes = (64, 64, 32, 32, 16, 16, 8, 8),
              activation = 'relu',
              solver='adam',
              max_iter=1000,
              early_stopping = True,
              validation_fraction = 0.3,
              alpha = 0.01,
              random_state=42,
              verbose=True
)

mlp.fit(x3_train, y3_train)
```

```
Iteration 1, loss = 2.41823665
Validation score: -4.263452
Iteration 2, loss = 2.31530279
Validation score: -4.071077
Iteration 3, loss = 2.22585977
Validation score: -3.905421
Iteration 4, loss = 2.15128538
Validation score: -3.762145
Iteration 5, loss = 2.08413654
Validation score: -3.617629
Iteration 6, loss = 2.01643330
Validation score: -3.466933
Iteration 7, loss = 1.94611261
```

```
Validation score: -3.308277
Iteration 8, loss = 1.87062280
Validation score: -3.134350
Iteration 9, loss = 1.78739864
Validation score: -2.931636
Iteration 10, loss = 1.69045346
Validation score: -2.696407
Iteration 11, loss = 1.57669273
Validation score: -2.402364
Iteration 12, loss = 1.42465538
Validation score: -2.004647
Iteration 13, loss = 1.22367835
Validation score: -1.492637
Iteration 14, loss = 0.97049443
Validation score: -0.912732
Iteration 15, loss = 0.70352198
Validation score: -0.374532
Iteration 16, loss = 0.48818570
Validation score: -0.104601
Iteration 17, loss = 0.42508194
Validation score: -0.231417
Iteration 18, loss = 0.53460988
Validation score: -0.275385
Iteration 19, loss = 0.52118999
Validation score: -0.119796
Iteration 20, loss = 0.43547794
Validation score: -0.067268
Iteration 21, loss = 0.40475830
Validation score: -0.106666
Iteration 22, loss = 0.41832892
Validation score: -0.146272
Iteration 23, loss = 0.42911370
Validation score: -0.140742
Iteration 24, loss = 0.42479935
Validation score: -0.109156
Iteration 25, loss = 0.41177170
Validation score: -0.071776
Iteration 26, loss = 0.39937786
Validation score: -0.047420
Iteration 27, loss = 0.39364543
Validation score: -0.038717
Iteration 28, loss = 0.39289768
Validation score: -0.036034
Iteration 29, loss = 0.39147768
Validation score: -0.033946
Iteration 30, loss = 0.38923823
Validation score: -0.036993
Iteration 31, loss = 0.38745475
Validation score: -0.038980
```

```
Iteration 32, loss = 0.38591056
Validation score: -0.036075
Iteration 33, loss = 0.38386348
Validation score: -0.032529
Iteration 34, loss = 0.38196134
Validation score: -0.027685
Iteration 35, loss = 0.38033069
Validation score: -0.023576
Iteration 36, loss = 0.37859476
Validation score: -0.020965
Iteration 37, loss = 0.37703754
Validation score: -0.018823
Iteration 38, loss = 0.37540084
Validation score: -0.016057
Iteration 39, loss = 0.37386906
Validation score: -0.013343
Iteration 40, loss = 0.37207805
Validation score: -0.011108
Iteration 41, loss = 0.37017951
Validation score: -0.011137
Iteration 42, loss = 0.36812397
Validation score: -0.012644
Iteration 43, loss = 0.36692260
Validation score: -0.013741
Iteration 44, loss = 0.36533988
Validation score: -0.012645
Iteration 45, loss = 0.36343956
Validation score: -0.012298
Iteration 46, loss = 0.36143893
Validation score: -0.010437
Iteration 47, loss = 0.35963912
Validation score: -0.008559
Iteration 48, loss = 0.35760581
Validation score: -0.007186
Iteration 49, loss = 0.35574341
Validation score: -0.005474
Iteration 50, loss = 0.35396031
Validation score: -0.004307
Iteration 51, loss = 0.35180062
Validation score: -0.006469
Iteration 52, loss = 0.34987852
Validation score: -0.006629
Iteration 53, loss = 0.34745860
Validation score: -0.005677
Iteration 54, loss = 0.34504148
Validation score: -0.004491
Iteration 55, loss = 0.34419874
Validation score: -0.003150
Iteration 56, loss = 0.34096079
```

```
Validation score: -0.002707
Iteration 57, loss = 0.33798985
Validation score: -0.004262
Iteration 58, loss = 0.33676487
Validation score: -0.005538
Iteration 59, loss = 0.33352566
Validation score: -0.002757
Iteration 60, loss = 0.33032784
Validation score: -0.000586
Iteration 61, loss = 0.32846042
Validation score: -0.000754
Iteration 62, loss = 0.32735544
Validation score: -0.001222
Iteration 63, loss = 0.32377308
Validation score: -0.001525
Iteration 64, loss = 0.31976098
Validation score: -0.000793
Iteration 65, loss = 0.31648210
Validation score: -0.004445
Iteration 66, loss = 0.31256649
Validation score: -0.004598
Iteration 67, loss = 0.30837146
Validation score: -0.006361
Iteration 68, loss = 0.30453927
Validation score: -0.011227
Iteration 69, loss = 0.30121934
Validation score: -0.015957
Iteration 70, loss = 0.29647647
Validation score: -0.030242
Iteration 71, loss = 0.29514424
Validation score: -0.027668
Validation score did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.

MLPRegressor(alpha=0.01, early_stopping=True,
             hidden_layer_sizes=(64, 64, 32, 32, 16, 16, 8, 8),
max_iter=1000,
             random_state=42, validation_fraction=0.3, verbose=True)

y3_pred_skl = mlp.predict(x3_test)

fig, ax = plt.subplots(figsize=(15, 8))
ax.plot(y3_test, color = 'yellow', label='Данные')
ax.plot(y3_pred_skl, color = 'red', label='Прогнозная модель')
ax.plot(y3_dummy, label='Базовая модель')
ax.legend()
plt.title('Результат работы модели многослойного перцептрона')
plt.show()
```




```
diff_mlp = pd.DataFrame()

dummy_3 = calculate_metrics("Dummy Regressor", y3_test, y3_dummy)
diff_mlp = pd.concat([diff_mlp, dummy_3], ignore_index=False)

mlp_metrics = calculate_metrics("MLPRegressor", y3_test, y3_pred_skl)
diff_mlp = pd.concat([diff_mlp, mlp_metrics], ignore_index=False)

styled_diff_mlp = style_model_results(diff_mlp)
styled_diff_mlp
```

Нейронная сеть на TensorFlow

```
# создаем аналогичную архитектуру нейросети

def keras_model():
    return tf.keras.Sequential([
      keras.layers.Input(shape=(12,), name='in'),
# 12 признаков
      keras.layers.Dense(64, activation='relu', name='dense_1'),
      keras.layers.Dense(64, activation='relu', name='dense_2'),
      keras.layers.Dense(32, activation='relu', name='dense_3'),
      keras.layers.Dense(32, activation='relu', name='dense_4'),
      keras.layers.Dense(16, activation='relu', name='dense_5'),
      keras.layers.Dense(16, activation='relu', name='dense_6'),
      keras.layers.Dense(8, activation='relu', name='dense_7'),
      keras.layers.Dense(8, activation='relu', name='dense_8'),
```

```
      keras.layers.Dense(1, name='out')
    ])

def compile_model(model):
    model.compile(
      optimizer=keras.optimizers.Adam(
            learning_rate=0.01,
      ),
      loss=keras.losses.MeanAbsolutePercentageError(),
      metrics=['mae', 'mape', 'root_mean_squared_error']
    )
    return model

# визуализация графиков ошибок

def plot_nn_loss(history):

    fig, axes = plt.subplots(1, 3, figsize=(18, 5))  # 3 графика в 1
ряду

    # MAPE
    axes[0].plot(history.history['loss'], label='train_loss (MAPE)',
color='blue')
    axes[0].plot(history.history['val_loss'], label='val_loss (MAPE)',
color='orange')
    axes[0].set_xlabel('Эпоха')
    axes[0].legend()
    axes[0].set_title('График MAPE')
    axes[0].grid(True)

    # RMSE
    axes[1].plot(history.history['root_mean_squared_error'],
label='train_loss (RMSE)', color='blue')
    axes[1].plot(history.history['val_root_mean_squared_error'],
label='val_loss (RMSE)', color='orange')
    axes[1].set_xlabel('Эпоха')
    axes[1].legend()
    axes[1].set_title('График RMSE')
    axes[1].grid(True)

    # MAE
    axes[2].plot(history.history['mae'], label='train_loss (MAE)',
color='blue')
    axes[2].plot(history.history['val_mae'], label='val_loss (MAE)',
color='orange')
    axes[2].set_xlabel('Эпоха')
    axes[2].legend()
    axes[2].set_title('График MAE')
    axes[2].grid(True)
```

```
    plt.show()

model_NN = keras_model()

model_NN = compile_model(model_NN) # компиляция нейросети

model_NN.summary()

Model: "sequential_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ dense_1 (Dense)                       │ (None, 64)                   │
832 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_2 (Dense)                       │ (None, 64)                   │
4,160 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_3 (Dense)                       │ (None, 32)                   │
2,080 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_4 (Dense)                       │ (None, 32)                   │
1,056 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_5 (Dense)                       │ (None, 16)                   │
528 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_6 (Dense)                       │ (None, 16)                   │
272 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_7 (Dense)                       │ (None, 8)                    │
136 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_8 (Dense)                       │ (None, 8)                    │
72 │
├───────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ out (Dense)                           │ (None, 1)                    │
```

```
9 │
└─────────────────────────────────────┴─────────────────────────────
─────────────────┘

 Total params: 9,145 (35.72 KB)

 Trainable params: 9,145 (35.72 KB)

 Non-trainable params: 0 (0.00 B)

model_NN_hist = model_NN.fit(
    x3_train,
    y3_train,
    epochs = 100,
    validation_split = 0.3,
    verbose = 1)

Epoch 1/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 95.5636 - mae: 2.8360
- mape: 95.5636 - root_mean_squared_error: 2.9803 - val_loss: 33.4639
- val_mae: 0.8572 - val_mape: 33.4639 - val_root_mean_squared_error:
1.0883
Epoch 2/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 36.2331 - mae: 0.9636 -
mape: 36.2331 - root_mean_squared_error: 1.1910 - val_loss: 36.9022 -
val_mae: 0.8009 - val_mape: 36.9022 - val_root_mean_squared_error:
1.0186
Epoch 3/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 28.3675 - mae: 0.8305 -
mape: 28.3675 - root_mean_squared_error: 1.0461 - val_loss: 36.2600 -
val_mae: 0.8004 - val_mape: 36.2600 - val_root_mean_squared_error:
1.0236
Epoch 4/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 28.9705 - mae: 0.7895 -
mape: 28.9705 - root_mean_squared_error: 0.9823 - val_loss: 33.2460 -
val_mae: 0.7851 - val_mape: 33.2460 - val_root_mean_squared_error:
1.0064
Epoch 5/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.7624 - mae: 0.7577 -
mape: 26.7624 - root_mean_squared_error: 0.9494 - val_loss: 32.6962 -
val_mae: 0.8014 - val_mape: 32.6962 - val_root_mean_squared_error:
1.0143
Epoch 6/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.4231 - mae: 0.7690
- mape: 26.4231 - root_mean_squared_error: 0.9716 - val_loss: 32.4349
- val_mae: 0.8345 - val_mape: 32.4349 - val_root_mean_squared_error:
1.0507
Epoch 7/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 27.0862 - mae: 0.7393 -
mape: 27.0862 - root_mean_squared_error: 0.9458 - val_loss: 33.2317 -
```

```
val_mae: 0.8037 - val_mape: 33.2317 - val_root_mean_squared_error:
1.0191
Epoch 8/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.6154 - mae: 0.7666 -
mape: 27.6154 - root_mean_squared_error: 0.9563 - val_loss: 33.0496 -
val_mae: 0.8572 - val_mape: 33.0496 - val_root_mean_squared_error:
1.0674
Epoch 9/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 25.2396 - mae: 0.7191 -
mape: 25.2396 - root_mean_squared_error: 0.9247 - val_loss: 32.8048 -
val_mae: 0.8293 - val_mape: 32.8048 - val_root_mean_squared_error:
1.0355
Epoch 10/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.2856 - mae: 0.6976 -
mape: 25.2856 - root_mean_squared_error: 0.8939 - val_loss: 33.3526 -
val_mae: 0.8023 - val_mape: 33.3526 - val_root_mean_squared_error:
0.9963
Epoch 11/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.2971 - mae: 0.7274 -
mape: 25.2971 - root_mean_squared_error: 0.9447 - val_loss: 33.5111 -
val_mae: 0.8435 - val_mape: 33.5111 - val_root_mean_squared_error:
1.0478
Epoch 12/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.9744 - mae: 0.7594 -
mape: 25.9744 - root_mean_squared_error: 0.9815 - val_loss: 33.5601 -
val_mae: 0.8416 - val_mape: 33.5601 - val_root_mean_squared_error:
1.0487
Epoch 13/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.8045 - mae: 0.7777
- mape: 26.8045 - root_mean_squared_error: 0.9766 - val_loss: 37.1431
- val_mae: 0.8239 - val_mape: 37.1431 - val_root_mean_squared_error:
1.0371
Epoch 14/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 26.6768 - mae: 0.7023 -
mape: 26.6768 - root_mean_squared_error: 0.8759 - val_loss: 33.6854 -
val_mae: 0.8356 - val_mape: 33.6854 - val_root_mean_squared_error:
1.0350
Epoch 15/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 23.9689 - mae: 0.7007 -
mape: 23.9689 - root_mean_squared_error: 0.9028 - val_loss: 34.7943 -
val_mae: 0.8242 - val_mape: 34.7943 - val_root_mean_squared_error:
1.0213
Epoch 16/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 26.6186 - mae: 0.7301 -
mape: 26.6186 - root_mean_squared_error: 0.9171 - val_loss: 33.9516 -
val_mae: 0.8435 - val_mape: 33.9516 - val_root_mean_squared_error:
1.0343
Epoch 17/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 24.8719 - mae: 0.7196 -
```

```
mape: 24.8719 - root_mean_squared_error: 0.9341 - val_loss: 33.5535 -
val_mae: 0.8838 - val_mape: 33.5535 - val_root_mean_squared_error:
1.0849
Epoch 18/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 23.5370 - mae: 0.6908 -
mape: 23.5370 - root_mean_squared_error: 0.9041 - val_loss: 33.6994 -
val_mae: 0.8410 - val_mape: 33.6994 - val_root_mean_squared_error:
1.0351
Epoch 19/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 23.4016 - mae: 0.6797 -
mape: 23.4016 - root_mean_squared_error: 0.8872 - val_loss: 33.3635 -
val_mae: 0.8641 - val_mape: 33.3635 - val_root_mean_squared_error:
1.0660
Epoch 20/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 22.3165 - mae: 0.6696 -
mape: 22.3165 - root_mean_squared_error: 0.8933 - val_loss: 34.2407 -
val_mae: 0.8864 - val_mape: 34.2407 - val_root_mean_squared_error:
1.0824
Epoch 21/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 22.1014 - mae: 0.6146 -
mape: 22.1014 - root_mean_squared_error: 0.8106 - val_loss: 34.9694 -
val_mae: 0.8582 - val_mape: 34.9694 - val_root_mean_squared_error:
1.0520
Epoch 22/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 22.2066 - mae: 0.6360 -
mape: 22.2066 - root_mean_squared_error: 0.8336 - val_loss: 35.1442 -
val_mae: 0.8921 - val_mape: 35.1442 - val_root_mean_squared_error:
1.0931
Epoch 23/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 22.5168 - mae: 0.6637 -
mape: 22.5168 - root_mean_squared_error: 0.8811 - val_loss: 34.9410 -
val_mae: 0.9391 - val_mape: 34.9410 - val_root_mean_squared_error:
1.1265
Epoch 24/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 23.1133 - mae: 0.6950 -
mape: 23.1133 - root_mean_squared_error: 0.9085 - val_loss: 34.1053 -
val_mae: 0.8598 - val_mape: 34.1053 - val_root_mean_squared_error:
1.0592
Epoch 25/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 22.4548 - mae: 0.6838 -
mape: 22.4548 - root_mean_squared_error: 0.9191 - val_loss: 34.6578 -
val_mae: 0.9536 - val_mape: 34.6578 - val_root_mean_squared_error:
1.1628
Epoch 26/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 21.3333 - mae: 0.6624 -
mape: 21.3333 - root_mean_squared_error: 0.8908 - val_loss: 33.7395 -
val_mae: 0.9033 - val_mape: 33.7395 - val_root_mean_squared_error:
1.1046
Epoch 27/100
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.5357 - mae: 0.6166 -
mape: 20.5357 - root_mean_squared_error: 0.8358 - val_loss: 35.4197 -
val_mae: 0.8908 - val_mape: 35.4197 - val_root_mean_squared_error:
1.1064
Epoch 28/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 21.0147 - mae: 0.6059 -
mape: 21.0147 - root_mean_squared_error: 0.8276 - val_loss: 38.1417 -
val_mae: 0.8601 - val_mape: 38.1417 - val_root_mean_squared_error:
1.0767
Epoch 29/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.1041 - mae: 0.6050 -
mape: 21.1041 - root_mean_squared_error: 0.7871 - val_loss: 33.5269 -
val_mae: 0.7908 - val_mape: 33.5269 - val_root_mean_squared_error:
0.9828
Epoch 30/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 19.8516 - mae: 0.5659
- mape: 19.8516 - root_mean_squared_error: 0.7652 - val_loss: 34.5083
- val_mae: 0.8759 - val_mape: 34.5083 - val_root_mean_squared_error:
1.0709
Epoch 31/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 18.2530 - mae: 0.5653 -
mape: 18.2530 - root_mean_squared_error: 0.7838 - val_loss: 34.7497 -
val_mae: 0.8441 - val_mape: 34.7497 - val_root_mean_squared_error:
1.0497
Epoch 32/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 21.1278 - mae: 0.6059 -
mape: 21.1278 - root_mean_squared_error: 0.7990 - val_loss: 35.8556 -
val_mae: 0.8252 - val_mape: 35.8556 - val_root_mean_squared_error:
1.0490
Epoch 33/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 20.9666 - mae: 0.5677 -
mape: 20.9666 - root_mean_squared_error: 0.7356 - val_loss: 34.6794 -
val_mae: 0.8283 - val_mape: 34.6794 - val_root_mean_squared_error:
1.0286
Epoch 34/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 17.9490 - mae: 0.5251
- mape: 17.9490 - root_mean_squared_error: 0.7042 - val_loss: 34.9339
- val_mae: 0.8747 - val_mape: 34.9339 - val_root_mean_squared_error:
1.0844
Epoch 35/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 16.9710 - mae: 0.5068 -
mape: 16.9710 - root_mean_squared_error: 0.6950 - val_loss: 35.8793 -
val_mae: 0.8856 - val_mape: 35.8793 - val_root_mean_squared_error:
1.1301
Epoch 36/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 17.8124 - mae: 0.5348
- mape: 17.8124 - root_mean_squared_error: 0.7345 - val_loss: 35.7071
- val_mae: 0.8790 - val_mape: 35.7071 - val_root_mean_squared_error:
1.0826
```

```
Epoch 37/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 17.3193 - mae: 0.5260 -
mape: 17.3193 - root_mean_squared_error: 0.7197 - val_loss: 35.9728 -
val_mae: 0.8851 - val_mape: 35.9728 - val_root_mean_squared_error:
1.1237
Epoch 38/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 16.5096 - mae: 0.5045
- mape: 16.5096 - root_mean_squared_error: 0.7132 - val_loss: 34.7705
- val_mae: 0.8834 - val_mape: 34.7705 - val_root_mean_squared_error:
1.0970
Epoch 39/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.4377 - mae: 0.5213 -
mape: 16.4377 - root_mean_squared_error: 0.7117 - val_loss: 34.5684 -
val_mae: 0.8328 - val_mape: 34.5684 - val_root_mean_squared_error:
1.0549
Epoch 40/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 15.5010 - mae: 0.4707 -
mape: 15.5010 - root_mean_squared_error: 0.6606 - val_loss: 38.0243 -
val_mae: 0.9019 - val_mape: 38.0243 - val_root_mean_squared_error:
1.1517
Epoch 41/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 15.4145 - mae: 0.4425 -
mape: 15.4145 - root_mean_squared_error: 0.6095 - val_loss: 35.0732 -
val_mae: 0.8631 - val_mape: 35.0732 - val_root_mean_squared_error:
1.0903
Epoch 42/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.2528 - mae: 0.4904 -
mape: 16.2528 - root_mean_squared_error: 0.6643 - val_loss: 35.0248 -
val_mae: 0.8695 - val_mape: 35.0248 - val_root_mean_squared_error:
1.0865
Epoch 43/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 14.9301 - mae: 0.4567 -
mape: 14.9301 - root_mean_squared_error: 0.6563 - val_loss: 35.4355 -
val_mae: 0.8573 - val_mape: 35.4355 - val_root_mean_squared_error:
1.0754
Epoch 44/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 13.0362 - mae: 0.3995 -
mape: 13.0362 - root_mean_squared_error: 0.5720 - val_loss: 36.3503 -
val_mae: 0.9024 - val_mape: 36.3503 - val_root_mean_squared_error:
1.1168
Epoch 45/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 14.4076 - mae: 0.4467 -
mape: 14.4076 - root_mean_squared_error: 0.6193 - val_loss: 34.7607 -
val_mae: 0.8518 - val_mape: 34.7607 - val_root_mean_squared_error:
1.0669
Epoch 46/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 13.7231 - mae: 0.4186 -
mape: 13.7231 - root_mean_squared_error: 0.5955 - val_loss: 37.0095 -
val_mae: 0.8660 - val_mape: 37.0095 - val_root_mean_squared_error:
```

```
1.0946
Epoch 47/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 12.1785 - mae: 0.3588 -
mape: 12.1785 - root_mean_squared_error: 0.5194 - val_loss: 36.6186 -
val_mae: 0.9052 - val_mape: 36.6186 - val_root_mean_squared_error:
1.1260
Epoch 48/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 15.3178 - mae: 0.4886 -
mape: 15.3178 - root_mean_squared_error: 0.6987 - val_loss: 38.3867 -
val_mae: 0.8902 - val_mape: 38.3867 - val_root_mean_squared_error:
1.1046
Epoch 49/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 14.3944 - mae: 0.4157 -
mape: 14.3944 - root_mean_squared_error: 0.5510 - val_loss: 36.1396 -
val_mae: 0.9041 - val_mape: 36.1396 - val_root_mean_squared_error:
1.1402
Epoch 50/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 13.5498 - mae: 0.4008 -
mape: 13.5498 - root_mean_squared_error: 0.5602 - val_loss: 35.1109 -
val_mae: 0.8530 - val_mape: 35.1109 - val_root_mean_squared_error:
1.0595
Epoch 51/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 13.9457 - mae: 0.4157
- mape: 13.9457 - root_mean_squared_error: 0.5930 - val_loss: 37.9483
- val_mae: 0.8775 - val_mape: 37.9483 - val_root_mean_squared_error:
1.1245
Epoch 52/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 13.9203 - mae: 0.4114 -
mape: 13.9203 - root_mean_squared_error: 0.5382 - val_loss: 36.3843 -
val_mae: 0.8888 - val_mape: 36.3843 - val_root_mean_squared_error:
1.1129
Epoch 53/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 14.9545 - mae: 0.4709 -
mape: 14.9545 - root_mean_squared_error: 0.6569 - val_loss: 36.4370 -
val_mae: 0.9134 - val_mape: 36.4370 - val_root_mean_squared_error:
1.1380
Epoch 54/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 12.8218 - mae: 0.4027 -
mape: 12.8218 - root_mean_squared_error: 0.5898 - val_loss: 40.0553 -
val_mae: 0.9599 - val_mape: 40.0553 - val_root_mean_squared_error:
1.1931
Epoch 55/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 12.6213 - mae: 0.3854 -
mape: 12.6213 - root_mean_squared_error: 0.5411 - val_loss: 35.2581 -
val_mae: 0.8688 - val_mape: 35.2581 - val_root_mean_squared_error:
1.0986
Epoch 56/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 14.0617 - mae: 0.4211 -
mape: 14.0617 - root_mean_squared_error: 0.5939 - val_loss: 38.9552 -
```

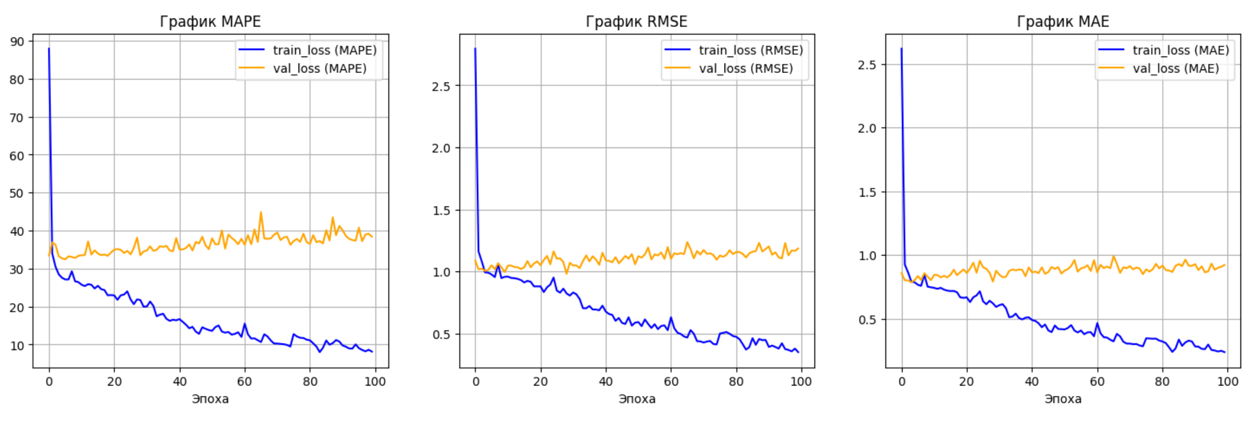
```
val_mae: 0.8944 - val_mape: 38.9552 - val_root_mean_squared_error:
1.1390
Epoch 57/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 12.9777 - mae: 0.3968 -
mape: 12.9777 - root_mean_squared_error: 0.5601 - val_loss: 38.0580 -
val_mae: 0.9015 - val_mape: 38.0580 - val_root_mean_squared_error:
1.1303
Epoch 58/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 12.7218 - mae: 0.3890
- mape: 12.7218 - root_mean_squared_error: 0.5618 - val_loss: 37.4063
- val_mae: 0.9185 - val_mape: 37.4063 - val_root_mean_squared_error:
1.1543
Epoch 59/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 12.5760 - mae: 0.3758 -
mape: 12.5760 - root_mean_squared_error: 0.5540 - val_loss: 36.4197 -
val_mae: 0.8731 - val_mape: 36.4197 - val_root_mean_squared_error:
1.1026
Epoch 60/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 11.4175 - mae: 0.3445 -
mape: 11.4175 - root_mean_squared_error: 0.5030 - val_loss: 37.8534 -
val_mae: 0.9588 - val_mape: 37.8534 - val_root_mean_squared_error:
1.1975
Epoch 61/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 16.0114 - mae: 0.4878 -
mape: 16.0114 - root_mean_squared_error: 0.6642 - val_loss: 36.2760 -
val_mae: 0.8630 - val_mape: 36.2760 - val_root_mean_squared_error:
1.1031
Epoch 62/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 12.2763 - mae: 0.3766 -
mape: 12.2763 - root_mean_squared_error: 0.5531 - val_loss: 38.8264 -
val_mae: 0.9226 - val_mape: 38.8264 - val_root_mean_squared_error:
1.1475
Epoch 63/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 11.4743 - mae: 0.3506 -
mape: 11.4743 - root_mean_squared_error: 0.5041 - val_loss: 36.4012 -
val_mae: 0.8966 - val_mape: 36.4012 - val_root_mean_squared_error:
1.1403
Epoch 64/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 10.7493 - mae: 0.3207 -
mape: 10.7493 - root_mean_squared_error: 0.4613 - val_loss: 40.3046 -
val_mae: 0.9088 - val_mape: 40.3046 - val_root_mean_squared_error:
1.1491
Epoch 65/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 11.0777 - mae: 0.3412
- mape: 11.0777 - root_mean_squared_error: 0.4753 - val_loss: 37.0151
- val_mae: 0.8958 - val_mape: 37.0151 - val_root_mean_squared_error:
1.1409
Epoch 66/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 10.5304 - mae: 0.3178 -
```

```
mape: 10.5304 - root_mean_squared_error: 0.4583 - val_loss: 44.8608 -
val_mae: 0.9910 - val_mape: 44.8608 - val_root_mean_squared_error:
1.2377
Epoch 67/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 13.4216 - mae: 0.3917 -
mape: 13.4216 - root_mean_squared_error: 0.5271 - val_loss: 37.9164 -
val_mae: 0.9307 - val_mape: 37.9164 - val_root_mean_squared_error:
1.1802
Epoch 68/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 11.9828 - mae: 0.3551 -
mape: 11.9828 - root_mean_squared_error: 0.4932 - val_loss: 37.8286 -
val_mae: 0.8589 - val_mape: 37.8286 - val_root_mean_squared_error:
1.1077
Epoch 69/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 10.8216 - mae: 0.3084 -
mape: 10.8216 - root_mean_squared_error: 0.4292 - val_loss: 37.9609 -
val_mae: 0.9033 - val_mape: 37.9609 - val_root_mean_squared_error:
1.1658
Epoch 70/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 9.7379 - mae: 0.2930 -
mape: 9.7379 - root_mean_squared_error: 0.4326 - val_loss: 38.8987 -
val_mae: 0.8956 - val_mape: 38.8987 - val_root_mean_squared_error:
1.1373
Epoch 71/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 10.5250 - mae: 0.3070 -
mape: 10.5250 - root_mean_squared_error: 0.4362 - val_loss: 39.4811 -
val_mae: 0.9119 - val_mape: 39.4811 - val_root_mean_squared_error:
1.1748
Epoch 72/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 9.6636 - mae: 0.2821 -
mape: 9.6636 - root_mean_squared_error: 0.4324 - val_loss: 37.5035 -
val_mae: 0.8866 - val_mape: 37.5035 - val_root_mean_squared_error:
1.1436
Epoch 73/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 10.7302 - mae: 0.3211 -
mape: 10.7302 - root_mean_squared_error: 0.4689 - val_loss: 38.2308 -
val_mae: 0.9014 - val_mape: 38.2308 - val_root_mean_squared_error:
1.1471
Epoch 74/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 9.1532 - mae: 0.2666 -
mape: 9.1532 - root_mean_squared_error: 0.3885 - val_loss: 38.3637 -
val_mae: 0.8930 - val_mape: 38.3637 - val_root_mean_squared_error:
1.1329
Epoch 75/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 8.7358 - mae: 0.2475 -
mape: 8.7358 - root_mean_squared_error: 0.3621 - val_loss: 36.2560 -
val_mae: 0.8492 - val_mape: 36.2560 - val_root_mean_squared_error:
1.0958
Epoch 76/100
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 13.1176 - mae: 0.3482 -
mape: 13.1176 - root_mean_squared_error: 0.4978 - val_loss: 37.2419 -
val_mae: 0.8868 - val_mape: 37.2419 - val_root_mean_squared_error:
1.1299
Epoch 77/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 10.9056 - mae: 0.3143 -
mape: 10.9056 - root_mean_squared_error: 0.4550 - val_loss: 37.8056 -
val_mae: 0.8688 - val_mape: 37.8056 - val_root_mean_squared_error:
1.1192
Epoch 78/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 11.7894 - mae: 0.3317 -
mape: 11.7894 - root_mean_squared_error: 0.5309 - val_loss: 37.0270 -
val_mae: 0.8871 - val_mape: 37.0270 - val_root_mean_squared_error:
1.1349
Epoch 79/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 10.9643 - mae: 0.3192 -
mape: 10.9643 - root_mean_squared_error: 0.4856 - val_loss: 39.1899 -
val_mae: 0.9301 - val_mape: 39.1899 - val_root_mean_squared_error:
1.1728
Epoch 80/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 10.7969 - mae: 0.3051 -
mape: 10.7969 - root_mean_squared_error: 0.4496 - val_loss: 36.9815 -
val_mae: 0.8941 - val_mape: 36.9815 - val_root_mean_squared_error:
1.1410
Epoch 81/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 9.9900 - mae: 0.2917 -
mape: 9.9900 - root_mean_squared_error: 0.4319 - val_loss: 36.4823 -
val_mae: 0.9163 - val_mape: 36.4823 - val_root_mean_squared_error:
1.1581
Epoch 82/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 11.0925 - mae: 0.3457 -
mape: 11.0925 - root_mean_squared_error: 0.4953 - val_loss: 38.7897 -
val_mae: 0.8810 - val_mape: 38.7897 - val_root_mean_squared_error:
1.1569
Epoch 83/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 10.3092 - mae: 0.3046 -
mape: 10.3092 - root_mean_squared_error: 0.4563 - val_loss: 36.9485 -
val_mae: 0.8781 - val_mape: 36.9485 - val_root_mean_squared_error:
1.1431
Epoch 84/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.4239 - mae: 0.2344 -
mape: 8.4239 - root_mean_squared_error: 0.3657 - val_loss: 37.2757 -
val_mae: 0.8660 - val_mape: 37.2757 - val_root_mean_squared_error:
1.1142
Epoch 85/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.0915 - mae: 0.2348 -
mape: 8.0915 - root_mean_squared_error: 0.3475 - val_loss: 36.6211 -
val_mae: 0.9141 - val_mape: 36.6211 - val_root_mean_squared_error:
1.1471
```

```
Epoch 86/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 10.4430 - mae: 0.3263 -
mape: 10.4430 - root_mean_squared_error: 0.4537 - val_loss: 40.1186 -
val_mae: 0.9285 - val_mape: 40.1186 - val_root_mean_squared_error:
1.1604
Epoch 87/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 9.6313 - mae: 0.2815 -
mape: 9.6313 - root_mean_squared_error: 0.3915 - val_loss: 37.3852 -
val_mae: 0.9124 - val_mape: 37.3852 - val_root_mean_squared_error:
1.1645
Epoch 88/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 9.8408 - mae: 0.2916 -
mape: 9.8408 - root_mean_squared_error: 0.4174 - val_loss: 43.5088 -
val_mae: 0.9634 - val_mape: 43.5088 - val_root_mean_squared_error:
1.2316
Epoch 89/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 11.5330 - mae: 0.3440 -
mape: 11.5330 - root_mean_squared_error: 0.4573 - val_loss: 38.7607 -
val_mae: 0.9161 - val_mape: 38.7607 - val_root_mean_squared_error:
1.1618
Epoch 90/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 11.1445 - mae: 0.3176 -
mape: 11.1445 - root_mean_squared_error: 0.4423 - val_loss: 41.2235 -
val_mae: 0.9145 - val_mape: 41.2235 - val_root_mean_squared_error:
1.1817
Epoch 91/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 9.4889 - mae: 0.2706 -
mape: 9.4889 - root_mean_squared_error: 0.3679 - val_loss: 40.1097 -
val_mae: 0.9275 - val_mape: 40.1097 - val_root_mean_squared_error:
1.2030
Epoch 92/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.9555 - mae: 0.2733 -
mape: 8.9555 - root_mean_squared_error: 0.3929 - val_loss: 38.6250 -
val_mae: 0.8812 - val_mape: 38.6250 - val_root_mean_squared_error:
1.1356
Epoch 93/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 8.8570 - mae: 0.2665 -
mape: 8.8570 - root_mean_squared_error: 0.4008 - val_loss: 37.8359 -
val_mae: 0.9106 - val_mape: 37.8359 - val_root_mean_squared_error:
1.1551
Epoch 94/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 8.5829 - mae: 0.2479 -
mape: 8.5829 - root_mean_squared_error: 0.3688 - val_loss: 37.5212 -
val_mae: 0.8640 - val_mape: 37.5212 - val_root_mean_squared_error:
1.1154
Epoch 95/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 9.3962 - mae: 0.2713 -
mape: 9.3962 - root_mean_squared_error: 0.4057 - val_loss: 37.3353 -
val_mae: 0.8708 - val_mape: 37.3353 - val_root_mean_squared_error:
```

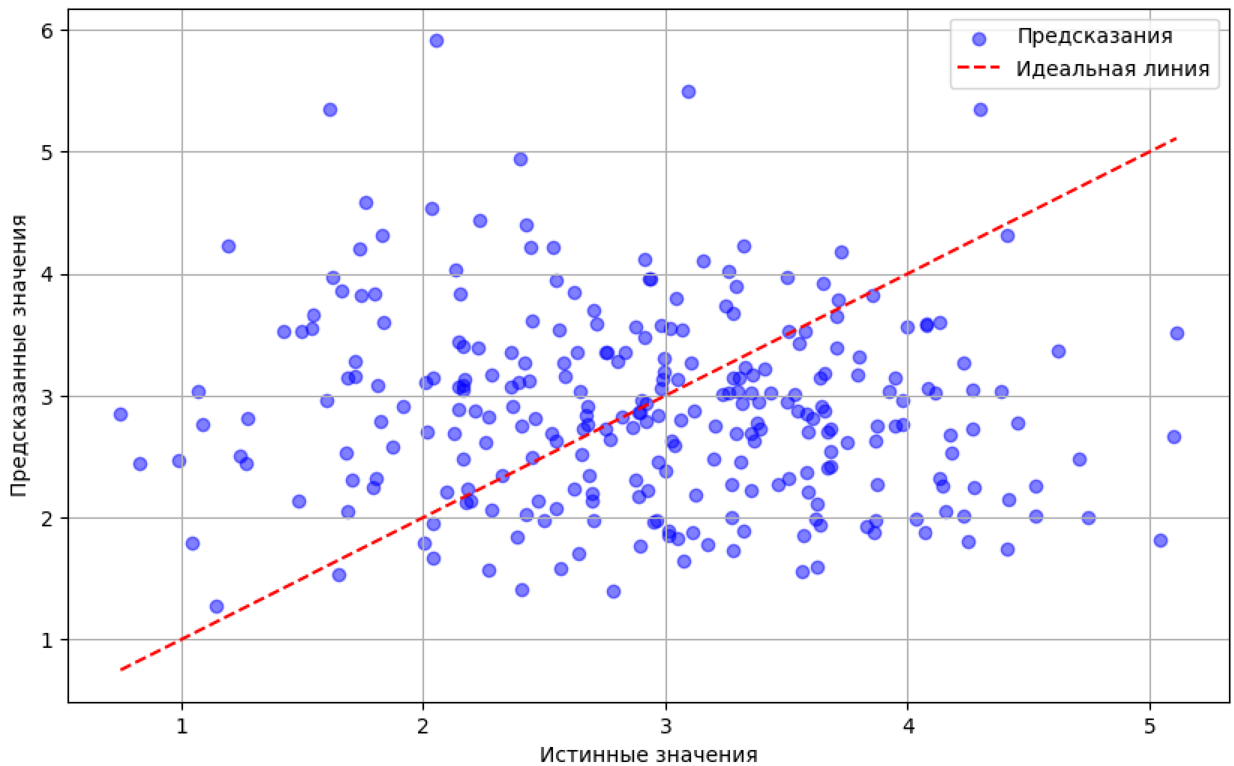
```
1.1061
Epoch 96/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 9.1598 - mae: 0.2541 -
mape: 9.1598 - root_mean_squared_error: 0.3790 - val_loss: 40.7900 -
val_mae: 0.9315 - val_mape: 40.7900 - val_root_mean_squared_error:
1.2306
Epoch 97/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 8.4988 - mae: 0.2525 -
mape: 8.4988 - root_mean_squared_error: 0.3730 - val_loss: 37.2183 -
val_mae: 0.8835 - val_mape: 37.2183 - val_root_mean_squared_error:
1.1307
Epoch 98/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 7.8484 - mae: 0.2362 -
mape: 7.8484 - root_mean_squared_error: 0.3474 - val_loss: 38.9462 -
val_mae: 0.8994 - val_mape: 38.9462 - val_root_mean_squared_error:
1.1704
Epoch 99/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 8.1256 - mae: 0.2220 -
mape: 8.1256 - root_mean_squared_error: 0.3376 - val_loss: 39.1766 -
val_mae: 0.9069 - val_mape: 39.1766 - val_root_mean_squared_error:
1.1668
Epoch 100/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 7.7087 - mae: 0.2259 -
mape: 7.7087 - root_mean_squared_error: 0.3385 - val_loss: 38.3827 -
val_mae: 0.9209 - val_mape: 38.3827 - val_root_mean_squared_error:
1.1862

plot_nn_loss(model_NN_hist)
```




```
y3_NN_pred = model_NN.predict(x3_test)

plt.figure(figsize=(10, 6))
plt.scatter(y3_test, y3_NN_pred, alpha=0.5, color="blue",
label="Предсказания")
plt.plot([min(y3_test), max(y3_test)], [min(y3_test), max(y3_test)],
color="red", linestyle="--", label="Идеальная линия")
```

```
plt.xlabel("Истинные значения")
plt.ylabel("Предсказанные значения")
plt.legend()
plt.grid(True)
plt.show()
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```



Постараюсь улучшить модель посредством добавления колбэка, дропаута, корректировки параметра коэффициента обучения и пакетной нормализации - нормализации входных данных слоев нейронки

```
from keras.layers import BatchNormalization
from keras.callbacks import EarlyStopping

def keras_model_2():
    return keras.Sequential([
        keras.layers.Input(shape=(12,), name='in'),

        keras.layers.Dense(64, name='dense_1'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_1'),

        keras.layers.Dense(32, name='dense_2'),
        keras.layers.BatchNormalization(),
```

```
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_2'),

        keras.layers.Dense(16, name='dense_3'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_3'),

        keras.layers.Dense(8, name='dense_4'),
        keras.layers.BatchNormalization(),
        keras.layers.Activation('relu'),
        keras.layers.Dropout(0.05, name='dropout_4'),

        keras.layers.Dense(1, name='out')
    ])

def compile_model_2(model):
    model.compile(
      optimizer=keras.optimizers.Adam(
            learning_rate=0.001,
      ),
      loss=keras.losses.MeanAbsolutePercentageError(),
      metrics=['mae', 'mape', 'root_mean_squared_error']
    )
    return model

# добавим колбэк, который остановит обучение, если ошибка на валидации
перестанет снижаться

early_stopping = EarlyStopping(
    monitor='val_loss',  # отслеживаемая метрика
    patience=20,         # количество эпох без улучшений
    restore_best_weights=True
)

model_NN_2 = keras_model_2()

model_NN_2 = compile_model_2(model_NN_2) # компиляция нейросети

model_NN_2.summary()

Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ dense_1 (Dense)                      │ (None, 64)                   │
832 │
```

```
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_4                │ (None, 64)                  │
256 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ activation_4 (Activation)            │ (None, 64)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 64)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_2 (Dense)                      │ (None, 32)                  │
2,080 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_5                │ (None, 32)                  │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ activation_5 (Activation)            │ (None, 32)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_2 (Dropout)                  │ (None, 32)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_3 (Dense)                      │ (None, 16)                  │
528 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_6                │ (None, 16)                  │
64 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ activation_6 (Activation)            │ (None, 16)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_3 (Dropout)                  │ (None, 16)                  │
```

```
0 │
├──────────────────────────────────────┼─────────────────────────────┼
────────────────┤
│ dense_4 (Dense)                      │ (None, 8)                   │
136 │
├──────────────────────────────────────┼─────────────────────────────┼
────────────────┤
│ batch_normalization_7                │ (None, 8)                   │
32 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
────────────────┤
│ activation_7 (Activation)            │ (None, 8)                   │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
────────────────┤
│ dropout_4 (Dropout)                  │ (None, 8)                   │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
────────────────┤
│ out (Dense)                          │ (None, 1)                   │
9 │
└──────────────────────────────────────┴─────────────────────────────┴
────────────────┘

 Total params: 4,065 (15.88 KB)

 Trainable params: 3,825 (14.94 KB)

 Non-trainable params: 240 (960.00 B)
```

```
model_NN_hist_2 = model_NN_2.fit(
    x3_train, y3_train,
    epochs=1000,
    validation_split = 0.3,
    verbose=1,
    callbacks=[early_stopping]
)
```

```
Epoch 1/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - loss: 88.1774 - mae: 2.6336
- mape: 88.1774 - root_mean_squared_error: 2.9034 - val_loss: 101.3080
- val_mae: 2.9400 - val_mape: 101.3080 - val_root_mean_squared_error:
3.0803
Epoch 2/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 85.1272 - mae: 2.5797 -
mape: 85.1272 - root_mean_squared_error: 2.8376 - val_loss: 97.3045 -
val_mae: 2.8419 - val_mape: 97.3045 - val_root_mean_squared_error:
2.9895
```

```
Epoch 3/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 82.6406 - mae: 2.4761
- mape: 82.6406 - root_mean_squared_error: 2.7456 - val_loss: 92.0091
- val_mae: 2.7142 - val_mape: 92.0091 - val_root_mean_squared_error:
2.8732
Epoch 4/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 76.4391 - mae: 2.3288 -
mape: 76.4391 - root_mean_squared_error: 2.6428 - val_loss: 85.7786 -
val_mae: 2.5604 - val_mape: 85.7786 - val_root_mean_squared_error:
2.7348
Epoch 5/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 72.8013 - mae: 2.2118 -
mape: 72.8013 - root_mean_squared_error: 2.5220 - val_loss: 80.6540 -
val_mae: 2.4253 - val_mape: 80.6540 - val_root_mean_squared_error:
2.6122
Epoch 6/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 68.4226 - mae: 2.1004 -
mape: 68.4226 - root_mean_squared_error: 2.4141 - val_loss: 76.2125 -
val_mae: 2.2955 - val_mape: 76.2125 - val_root_mean_squared_error:
2.4895
Epoch 7/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 68.0012 - mae: 2.1230 -
mape: 68.0012 - root_mean_squared_error: 2.4435 - val_loss: 72.8693 -
val_mae: 2.1947 - val_mape: 72.8693 - val_root_mean_squared_error:
2.3945
Epoch 8/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 63.3918 - mae: 1.9586 -
mape: 63.3918 - root_mean_squared_error: 2.2864 - val_loss: 68.9921 -
val_mae: 2.0797 - val_mape: 68.9921 - val_root_mean_squared_error:
2.2899
Epoch 9/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 60.8759 - mae: 1.8903 -
mape: 60.8759 - root_mean_squared_error: 2.2099 - val_loss: 65.2807 -
val_mae: 1.9673 - val_mape: 65.2807 - val_root_mean_squared_error:
2.1897
Epoch 10/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 56.5094 - mae: 1.7879 -
mape: 56.5094 - root_mean_squared_error: 2.1386 - val_loss: 61.9958 -
val_mae: 1.8660 - val_mape: 61.9958 - val_root_mean_squared_error:
2.0959
Epoch 11/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 53.5538 - mae: 1.6795 -
mape: 53.5538 - root_mean_squared_error: 1.9887 - val_loss: 58.8579 -
val_mae: 1.7649 - val_mape: 58.8579 - val_root_mean_squared_error:
1.9972
Epoch 12/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 52.7610 - mae: 1.6468 -
mape: 52.7610 - root_mean_squared_error: 1.9896 - val_loss: 55.9505 -
val_mae: 1.6688 - val_mape: 55.9505 - val_root_mean_squared_error:
```

```
1.9039
Epoch 13/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 49.3046 - mae: 1.5228 -
mape: 49.3046 - root_mean_squared_error: 1.8566 - val_loss: 53.5609 -
val_mae: 1.5894 - val_mape: 53.5609 - val_root_mean_squared_error:
1.8331
Epoch 14/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 46.3477 - mae: 1.4810 -
mape: 46.3477 - root_mean_squared_error: 1.8571 - val_loss: 50.7471 -
val_mae: 1.4964 - val_mape: 50.7471 - val_root_mean_squared_error:
1.7486
Epoch 15/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 46.3511 - mae: 1.5121
- mape: 46.3511 - root_mean_squared_error: 1.8840 - val_loss: 48.5594
- val_mae: 1.4252 - val_mape: 48.5594 - val_root_mean_squared_error:
1.6792
Epoch 16/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 44.0581 - mae: 1.3986
- mape: 44.0581 - root_mean_squared_error: 1.7304 - val_loss: 47.0553
- val_mae: 1.3719 - val_mape: 47.0553 - val_root_mean_squared_error:
1.6294
Epoch 17/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 42.9683 - mae: 1.3217 -
mape: 42.9683 - root_mean_squared_error: 1.6566 - val_loss: 44.5727 -
val_mae: 1.2828 - val_mape: 44.5727 - val_root_mean_squared_error:
1.5369
Epoch 18/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 40.7468 - mae: 1.2860 -
mape: 40.7468 - root_mean_squared_error: 1.6133 - val_loss: 43.1449 -
val_mae: 1.2293 - val_mape: 43.1449 - val_root_mean_squared_error:
1.4854
Epoch 19/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 38.6380 - mae: 1.1911
- mape: 38.6380 - root_mean_squared_error: 1.4918 - val_loss: 41.4630
- val_mae: 1.1651 - val_mape: 41.4630 - val_root_mean_squared_error:
1.4247
Epoch 20/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 37.7236 - mae: 1.1631 -
mape: 37.7236 - root_mean_squared_error: 1.5189 - val_loss: 39.9851 -
val_mae: 1.1106 - val_mape: 39.9851 - val_root_mean_squared_error:
1.3767
Epoch 21/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 36.0410 - mae: 1.1402 -
mape: 36.0410 - root_mean_squared_error: 1.4499 - val_loss: 39.1776 -
val_mae: 1.0797 - val_mape: 39.1776 - val_root_mean_squared_error:
1.3480
Epoch 22/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 35.1636 - mae: 1.0787 -
mape: 35.1636 - root_mean_squared_error: 1.3570 - val_loss: 39.2778 -
```

```
val_mae: 1.0683 - val_mape: 39.2778 - val_root_mean_squared_error:
1.3317
Epoch 23/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 33.3855 - mae: 1.0279 -
mape: 33.3855 - root_mean_squared_error: 1.3335 - val_loss: 38.2919 -
val_mae: 1.0332 - val_mape: 38.2919 - val_root_mean_squared_error:
1.2950
Epoch 24/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 33.0588 - mae: 1.0379 -
mape: 33.0588 - root_mean_squared_error: 1.3239 - val_loss: 37.7014 -
val_mae: 1.0121 - val_mape: 37.7014 - val_root_mean_squared_error:
1.2709
Epoch 25/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 33.3764 - mae: 1.0168 -
mape: 33.3764 - root_mean_squared_error: 1.3077 - val_loss: 37.1197 -
val_mae: 0.9953 - val_mape: 37.1197 - val_root_mean_squared_error:
1.2551
Epoch 26/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 31.1198 - mae: 0.9654 -
mape: 31.1198 - root_mean_squared_error: 1.2609 - val_loss: 36.5082 -
val_mae: 0.9759 - val_mape: 36.5082 - val_root_mean_squared_error:
1.2372
Epoch 27/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 33.4410 - mae: 0.9961 -
mape: 33.4410 - root_mean_squared_error: 1.2347 - val_loss: 35.9189 -
val_mae: 0.9481 - val_mape: 35.9189 - val_root_mean_squared_error:
1.2065
Epoch 28/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 31.7753 - mae: 0.9425 -
mape: 31.7753 - root_mean_squared_error: 1.2162 - val_loss: 35.7250 -
val_mae: 0.9426 - val_mape: 35.7250 - val_root_mean_squared_error:
1.1983
Epoch 29/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 31.4163 - mae: 0.9098 -
mape: 31.4163 - root_mean_squared_error: 1.1791 - val_loss: 35.5074 -
val_mae: 0.9302 - val_mape: 35.5074 - val_root_mean_squared_error:
1.1859
Epoch 30/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 30.3380 - mae: 0.9241 -
mape: 30.3380 - root_mean_squared_error: 1.2162 - val_loss: 35.0743 -
val_mae: 0.9107 - val_mape: 35.0743 - val_root_mean_squared_error:
1.1664
Epoch 31/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 31.0265 - mae: 0.9274 -
mape: 31.0265 - root_mean_squared_error: 1.1816 - val_loss: 35.1166 -
val_mae: 0.9052 - val_mape: 35.1166 - val_root_mean_squared_error:
1.1583
Epoch 32/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 30.5795 - mae: 0.9020 -
```

```
mape: 30.5795 - root_mean_squared_error: 1.1594 - val_loss: 35.4193 -
val_mae: 0.9186 - val_mape: 35.4193 - val_root_mean_squared_error:
1.1754
Epoch 33/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 31.6779 - mae: 0.9362 -
mape: 31.6779 - root_mean_squared_error: 1.2053 - val_loss: 35.2969 -
val_mae: 0.9165 - val_mape: 35.2969 - val_root_mean_squared_error:
1.1747
Epoch 34/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 30.5743 - mae: 0.8858
- mape: 30.5743 - root_mean_squared_error: 1.1416 - val_loss: 34.9503
- val_mae: 0.9046 - val_mape: 34.9503 - val_root_mean_squared_error:
1.1656
Epoch 35/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 29.2796 - mae: 0.8756 -
mape: 29.2796 - root_mean_squared_error: 1.1244 - val_loss: 34.8146 -
val_mae: 0.9022 - val_mape: 34.8146 - val_root_mean_squared_error:
1.1585
Epoch 36/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 32.7498 - mae: 0.9630 -
mape: 32.7498 - root_mean_squared_error: 1.2497 - val_loss: 34.7851 -
val_mae: 0.9011 - val_mape: 34.7851 - val_root_mean_squared_error:
1.1563
Epoch 37/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 34.2036 - mae: 0.9369 -
mape: 34.2036 - root_mean_squared_error: 1.1902 - val_loss: 34.9569 -
val_mae: 0.9039 - val_mape: 34.9569 - val_root_mean_squared_error:
1.1616
Epoch 38/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 29.5941 - mae: 0.8516 -
mape: 29.5941 - root_mean_squared_error: 1.1215 - val_loss: 34.5328 -
val_mae: 0.8905 - val_mape: 34.5328 - val_root_mean_squared_error:
1.1429
Epoch 39/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 28.2663 - mae: 0.8353 -
mape: 28.2663 - root_mean_squared_error: 1.0904 - val_loss: 34.4031 -
val_mae: 0.8812 - val_mape: 34.4031 - val_root_mean_squared_error:
1.1274
Epoch 40/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 29.4189 - mae: 0.8682 -
mape: 29.4189 - root_mean_squared_error: 1.1131 - val_loss: 34.2038 -
val_mae: 0.8790 - val_mape: 34.2038 - val_root_mean_squared_error:
1.1184
Epoch 41/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 28.3999 - mae: 0.8497 -
mape: 28.3999 - root_mean_squared_error: 1.0977 - val_loss: 33.9775 -
val_mae: 0.8775 - val_mape: 33.9775 - val_root_mean_squared_error:
1.1087
Epoch 42/1000
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 33.2560 - mae: 0.9450 -
mape: 33.2560 - root_mean_squared_error: 1.1827 - val_loss: 33.8516 -
val_mae: 0.8832 - val_mape: 33.8516 - val_root_mean_squared_error:
1.1139
Epoch 43/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 26.7993 - mae: 0.7905 -
mape: 26.7993 - root_mean_squared_error: 1.0548 - val_loss: 33.7730 -
val_mae: 0.8827 - val_mape: 33.7730 - val_root_mean_squared_error:
1.1102
Epoch 44/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 30.8106 - mae: 0.8841 -
mape: 30.8106 - root_mean_squared_error: 1.1590 - val_loss: 33.5584 -
val_mae: 0.8696 - val_mape: 33.5584 - val_root_mean_squared_error:
1.0986
Epoch 45/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 29.6058 - mae: 0.8406
- mape: 29.6058 - root_mean_squared_error: 1.1102 - val_loss: 33.5060
- val_mae: 0.8573 - val_mape: 33.5060 - val_root_mean_squared_error:
1.0847
Epoch 46/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 28.2973 - mae: 0.8170
- mape: 28.2973 - root_mean_squared_error: 1.0982 - val_loss: 33.2588
- val_mae: 0.8498 - val_mape: 33.2588 - val_root_mean_squared_error:
1.0817
Epoch 47/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 32.2736 - mae: 0.8918
- mape: 32.2736 - root_mean_squared_error: 1.1402 - val_loss: 33.1146
- val_mae: 0.8441 - val_mape: 33.1146 - val_root_mean_squared_error:
1.0777
Epoch 48/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 30.4202 - mae: 0.8614
- mape: 30.4202 - root_mean_squared_error: 1.1206 - val_loss: 33.3478
- val_mae: 0.8518 - val_mape: 33.3478 - val_root_mean_squared_error:
1.0858
Epoch 49/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 27.6172 - mae: 0.8010
- mape: 27.6172 - root_mean_squared_error: 1.0633 - val_loss: 33.4437
- val_mae: 0.8519 - val_mape: 33.4437 - val_root_mean_squared_error:
1.0821
Epoch 50/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 28.9648 - mae: 0.8393
- mape: 28.9648 - root_mean_squared_error: 1.0809 - val_loss: 33.3490
- val_mae: 0.8443 - val_mape: 33.3490 - val_root_mean_squared_error:
1.0719
Epoch 51/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 29.6793 - mae: 0.8841
- mape: 29.6793 - root_mean_squared_error: 1.1641 - val_loss: 33.2222
- val_mae: 0.8350 - val_mape: 33.2222 - val_root_mean_squared_error:
1.0631
```

```
Epoch 52/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 27.6915 - mae: 0.7957 -
mape: 27.6915 - root_mean_squared_error: 1.0443 - val_loss: 33.0839 -
val_mae: 0.8324 - val_mape: 33.0839 - val_root_mean_squared_error:
1.0594
Epoch 53/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.4590 - mae: 0.7820 -
mape: 26.4590 - root_mean_squared_error: 1.0153 - val_loss: 33.0488 -
val_mae: 0.8423 - val_mape: 33.0488 - val_root_mean_squared_error:
1.0696
Epoch 54/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 27.9656 - mae: 0.8143
- mape: 27.9656 - root_mean_squared_error: 1.0394 - val_loss: 33.1343
- val_mae: 0.8473 - val_mape: 33.1343 - val_root_mean_squared_error:
1.0758
Epoch 55/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 28.2213 - mae: 0.8224 -
mape: 28.2213 - root_mean_squared_error: 1.0697 - val_loss: 33.2355 -
val_mae: 0.8535 - val_mape: 33.2355 - val_root_mean_squared_error:
1.0835
Epoch 56/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 29.3247 - mae: 0.8056
- mape: 29.3247 - root_mean_squared_error: 1.0284 - val_loss: 33.1380
- val_mae: 0.8456 - val_mape: 33.1380 - val_root_mean_squared_error:
1.0753
Epoch 57/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 27.5465 - mae: 0.7853 -
mape: 27.5465 - root_mean_squared_error: 1.0220 - val_loss: 33.2205 -
val_mae: 0.8454 - val_mape: 33.2205 - val_root_mean_squared_error:
1.0769
Epoch 58/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 29.0212 - mae: 0.8434 -
mape: 29.0212 - root_mean_squared_error: 1.0805 - val_loss: 33.0892 -
val_mae: 0.8369 - val_mape: 33.0892 - val_root_mean_squared_error:
1.0694
Epoch 59/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 27.2923 - mae: 0.7790 -
mape: 27.2923 - root_mean_squared_error: 0.9927 - val_loss: 33.0064 -
val_mae: 0.8300 - val_mape: 33.0064 - val_root_mean_squared_error:
1.0634
Epoch 60/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 27.7437 - mae: 0.8123 -
mape: 27.7437 - root_mean_squared_error: 1.0737 - val_loss: 32.9750 -
val_mae: 0.8352 - val_mape: 32.9750 - val_root_mean_squared_error:
1.0680
Epoch 61/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 27.4478 - mae: 0.8064 -
mape: 27.4478 - root_mean_squared_error: 1.0523 - val_loss: 32.9875 -
val_mae: 0.8327 - val_mape: 32.9875 - val_root_mean_squared_error:
```

```
1.0605
Epoch 62/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 27.9396 - mae: 0.8131
- mape: 27.9396 - root_mean_squared_error: 1.0710 - val_loss: 33.0225
- val_mae: 0.8339 - val_mape: 33.0225 - val_root_mean_squared_error:
1.0597
Epoch 63/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 29.5187 - mae: 0.8217 -
mape: 29.5187 - root_mean_squared_error: 1.0378 - val_loss: 33.2124 -
val_mae: 0.8420 - val_mape: 33.2124 - val_root_mean_squared_error:
1.0661
Epoch 64/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 28.9234 - mae: 0.8181
- mape: 28.9234 - root_mean_squared_error: 1.0705 - val_loss: 33.1259
- val_mae: 0.8400 - val_mape: 33.1259 - val_root_mean_squared_error:
1.0629
Epoch 65/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.4513 - mae: 0.7957 -
mape: 27.4513 - root_mean_squared_error: 1.0465 - val_loss: 33.0138 -
val_mae: 0.8445 - val_mape: 33.0138 - val_root_mean_squared_error:
1.0699
Epoch 66/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 28.3816 - mae: 0.8270 -
mape: 28.3816 - root_mean_squared_error: 1.0522 - val_loss: 33.0627 -
val_mae: 0.8486 - val_mape: 33.0627 - val_root_mean_squared_error:
1.0761
Epoch 67/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 28.8367 - mae: 0.8210 -
mape: 28.8367 - root_mean_squared_error: 1.0562 - val_loss: 33.1063 -
val_mae: 0.8512 - val_mape: 33.1063 - val_root_mean_squared_error:
1.0789
Epoch 68/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 30.2124 - mae: 0.8593 -
mape: 30.2124 - root_mean_squared_error: 1.1283 - val_loss: 33.0981 -
val_mae: 0.8519 - val_mape: 33.0981 - val_root_mean_squared_error:
1.0797
Epoch 69/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 30.2030 - mae: 0.8642 -
mape: 30.2030 - root_mean_squared_error: 1.1146 - val_loss: 33.1220 -
val_mae: 0.8417 - val_mape: 33.1220 - val_root_mean_squared_error:
1.0646
Epoch 70/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.4452 - mae: 0.7750 -
mape: 26.4452 - root_mean_squared_error: 1.0322 - val_loss: 33.0844 -
val_mae: 0.8282 - val_mape: 33.0844 - val_root_mean_squared_error:
1.0445
Epoch 71/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.2439 - mae: 0.7683 -
mape: 26.2439 - root_mean_squared_error: 1.0361 - val_loss: 32.9106 -
```

```
val_mae: 0.8280 - val_mape: 32.9106 - val_root_mean_squared_error:
1.0427
Epoch 72/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 29.2650 - mae: 0.8431 -
mape: 29.2650 - root_mean_squared_error: 1.0939 - val_loss: 32.9183 -
val_mae: 0.8336 - val_mape: 32.9183 - val_root_mean_squared_error:
1.0466
Epoch 73/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.0673 - mae: 0.7504 -
mape: 26.0673 - root_mean_squared_error: 0.9910 - val_loss: 33.0930 -
val_mae: 0.8485 - val_mape: 33.0930 - val_root_mean_squared_error:
1.0644
Epoch 74/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 27.3177 - mae: 0.7786
- mape: 27.3177 - root_mean_squared_error: 1.0146 - val_loss: 32.9571
- val_mae: 0.8454 - val_mape: 32.9571 - val_root_mean_squared_error:
1.0625
Epoch 75/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.5126 - mae: 0.8067
- mape: 26.5126 - root_mean_squared_error: 1.0713 - val_loss: 32.7539
- val_mae: 0.8317 - val_mape: 32.7539 - val_root_mean_squared_error:
1.0464
Epoch 76/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.3566 - mae: 0.7873 -
mape: 26.3566 - root_mean_squared_error: 1.0491 - val_loss: 32.7062 -
val_mae: 0.8289 - val_mape: 32.7062 - val_root_mean_squared_error:
1.0420
Epoch 77/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.3363 - mae: 0.7520
- mape: 26.3363 - root_mean_squared_error: 1.0035 - val_loss: 32.6479
- val_mae: 0.8282 - val_mape: 32.6479 - val_root_mean_squared_error:
1.0419
Epoch 78/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 29.0963 - mae: 0.8278 -
mape: 29.0963 - root_mean_squared_error: 1.0590 - val_loss: 32.7647 -
val_mae: 0.8331 - val_mape: 32.7647 - val_root_mean_squared_error:
1.0469
Epoch 79/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.8037 - mae: 0.7654 -
mape: 26.8037 - root_mean_squared_error: 1.0029 - val_loss: 32.9306 -
val_mae: 0.8375 - val_mape: 32.9306 - val_root_mean_squared_error:
1.0498
Epoch 80/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.7419 - mae: 0.7559 -
mape: 25.7419 - root_mean_squared_error: 1.0043 - val_loss: 32.8307 -
val_mae: 0.8327 - val_mape: 32.8307 - val_root_mean_squared_error:
1.0424
Epoch 81/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.2780 - mae: 0.8022 -
```

```
mape: 26.2780 - root_mean_squared_error: 1.0542 - val_loss: 32.9372 -
val_mae: 0.8299 - val_mape: 32.9372 - val_root_mean_squared_error:
1.0405
Epoch 82/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.0601 - mae: 0.7711 -
mape: 27.0601 - root_mean_squared_error: 0.9888 - val_loss: 33.0358 -
val_mae: 0.8320 - val_mape: 33.0358 - val_root_mean_squared_error:
1.0445
Epoch 83/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.4572 - mae: 0.7027 -
mape: 25.4572 - root_mean_squared_error: 0.9312 - val_loss: 32.9890 -
val_mae: 0.8364 - val_mape: 32.9890 - val_root_mean_squared_error:
1.0499
Epoch 84/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 27.0344 - mae: 0.7770
- mape: 27.0344 - root_mean_squared_error: 1.0152 - val_loss: 32.8309
- val_mae: 0.8355 - val_mape: 32.8309 - val_root_mean_squared_error:
1.0521
Epoch 85/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 29.1586 - mae: 0.8120 -
mape: 29.1586 - root_mean_squared_error: 1.0518 - val_loss: 32.7775 -
val_mae: 0.8297 - val_mape: 32.7775 - val_root_mean_squared_error:
1.0473
Epoch 86/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.8780 - mae: 0.7093 -
mape: 24.8780 - root_mean_squared_error: 0.9377 - val_loss: 33.0244 -
val_mae: 0.8324 - val_mape: 33.0244 - val_root_mean_squared_error:
1.0497
Epoch 87/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 25.7002 - mae: 0.7072 -
mape: 25.7002 - root_mean_squared_error: 0.9475 - val_loss: 33.5374 -
val_mae: 0.8329 - val_mape: 33.5374 - val_root_mean_squared_error:
1.0441
Epoch 88/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.6164 - mae: 0.7763 -
mape: 26.6164 - root_mean_squared_error: 1.0185 - val_loss: 33.5078 -
val_mae: 0.8331 - val_mape: 33.5078 - val_root_mean_squared_error:
1.0450
Epoch 89/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.1373 - mae: 0.7937 -
mape: 27.1373 - root_mean_squared_error: 1.0144 - val_loss: 33.4078 -
val_mae: 0.8248 - val_mape: 33.4078 - val_root_mean_squared_error:
1.0341
Epoch 90/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 27.4686 - mae: 0.8033
- mape: 27.4686 - root_mean_squared_error: 1.0306 - val_loss: 33.2457
- val_mae: 0.8213 - val_mape: 33.2457 - val_root_mean_squared_error:
1.0284
Epoch 91/1000
```

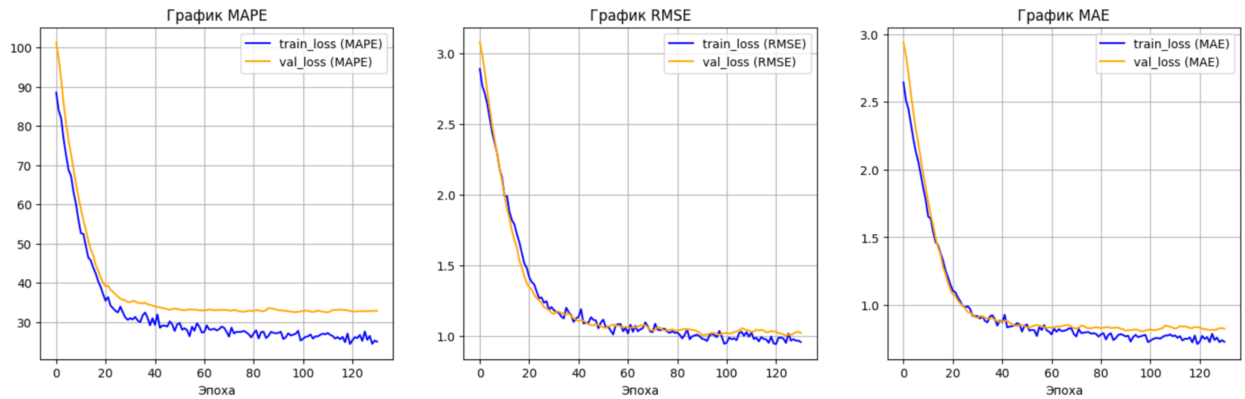
```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.1918 - mae: 0.7713 -
mape: 26.1918 - root_mean_squared_error: 0.9746 - val_loss: 33.0155 -
val_mae: 0.8098 - val_mape: 33.0155 - val_root_mean_squared_error:
1.0083
Epoch 92/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.6349 - mae: 0.7624 -
mape: 26.6349 - root_mean_squared_error: 0.9587 - val_loss: 32.9718 -
val_mae: 0.8144 - val_mape: 32.9718 - val_root_mean_squared_error:
1.0116
Epoch 93/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.9234 - mae: 0.7618 -
mape: 25.9234 - root_mean_squared_error: 1.0132 - val_loss: 32.9504 -
val_mae: 0.8138 - val_mape: 32.9504 - val_root_mean_squared_error:
1.0121
Epoch 94/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 26.2652 - mae: 0.7612 -
mape: 26.2652 - root_mean_squared_error: 0.9922 - val_loss: 32.8536 -
val_mae: 0.8183 - val_mape: 32.8536 - val_root_mean_squared_error:
1.0204
Epoch 95/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.3914 - mae: 0.7727 -
mape: 27.3914 - root_mean_squared_error: 1.0166 - val_loss: 32.7570 -
val_mae: 0.8194 - val_mape: 32.7570 - val_root_mean_squared_error:
1.0240
Epoch 96/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.3611 - mae: 0.7364
- mape: 26.3611 - root_mean_squared_error: 0.9455 - val_loss: 32.7008
- val_mae: 0.8161 - val_mape: 32.7008 - val_root_mean_squared_error:
1.0237
Epoch 97/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 25.3882 - mae: 0.7325 -
mape: 25.3882 - root_mean_squared_error: 0.9660 - val_loss: 32.5436 -
val_mae: 0.8077 - val_mape: 32.5436 - val_root_mean_squared_error:
1.0153
Epoch 98/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 26.4144 - mae: 0.7670 -
mape: 26.4144 - root_mean_squared_error: 1.0262 - val_loss: 32.5685 -
val_mae: 0.8060 - val_mape: 32.5685 - val_root_mean_squared_error:
1.0121
Epoch 99/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 27.9853 - mae: 0.7747 -
mape: 27.9853 - root_mean_squared_error: 0.9916 - val_loss: 32.6159 -
val_mae: 0.8122 - val_mape: 32.6159 - val_root_mean_squared_error:
1.0184
Epoch 100/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 25.8694 - mae: 0.7493 -
mape: 25.8694 - root_mean_squared_error: 0.9839 - val_loss: 32.7270 -
val_mae: 0.8159 - val_mape: 32.7270 - val_root_mean_squared_error:
1.0213
Epoch 101/1000
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 27.2935 - mae: 0.7575 -
mape: 27.2935 - root_mean_squared_error: 0.9732 - val_loss: 32.8364 -
val_mae: 0.8206 - val_mape: 32.8364 - val_root_mean_squared_error:
1.0255
Epoch 102/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.5826 - mae: 0.7582 -
mape: 25.5826 - root_mean_squared_error: 1.0231 - val_loss: 32.9176 -
val_mae: 0.8151 - val_mape: 32.9176 - val_root_mean_squared_error:
1.0214
Epoch 103/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.7900 - mae: 0.7147 -
mape: 25.7900 - root_mean_squared_error: 0.9166 - val_loss: 32.8988 -
val_mae: 0.8138 - val_mape: 32.8988 - val_root_mean_squared_error:
1.0212
Epoch 104/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 26.6201 - mae: 0.7409 -
mape: 26.6201 - root_mean_squared_error: 0.9739 - val_loss: 32.6720 -
val_mae: 0.8192 - val_mape: 32.6720 - val_root_mean_squared_error:
1.0304
Epoch 105/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 25.6123 - mae: 0.7494
- mape: 25.6123 - root_mean_squared_error: 0.9755 - val_loss: 32.5956
- val_mae: 0.8251 - val_mape: 32.5956 - val_root_mean_squared_error:
1.0361
Epoch 106/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.6368 - mae: 0.7697 -
mape: 25.6368 - root_mean_squared_error: 1.0238 - val_loss: 32.7508 -
val_mae: 0.8299 - val_mape: 32.7508 - val_root_mean_squared_error:
1.0382
Epoch 107/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 27.7775 - mae: 0.7579 -
mape: 27.7775 - root_mean_squared_error: 1.0122 - val_loss: 32.9041 -
val_mae: 0.8485 - val_mape: 32.9041 - val_root_mean_squared_error:
1.0581
Epoch 108/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 26.3000 - mae: 0.7524
- mape: 26.3000 - root_mean_squared_error: 0.9826 - val_loss: 32.7839
- val_mae: 0.8438 - val_mape: 32.7839 - val_root_mean_squared_error:
1.0517
Epoch 109/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 27.7415 - mae: 0.7849
- mape: 27.7415 - root_mean_squared_error: 1.0254 - val_loss: 32.6454
- val_mae: 0.8405 - val_mape: 32.6454 - val_root_mean_squared_error:
1.0459
Epoch 110/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 25.3992 - mae: 0.7329
- mape: 25.3992 - root_mean_squared_error: 0.9556 - val_loss: 32.5486
- val_mae: 0.8321 - val_mape: 32.5486 - val_root_mean_squared_error:
1.0402
```

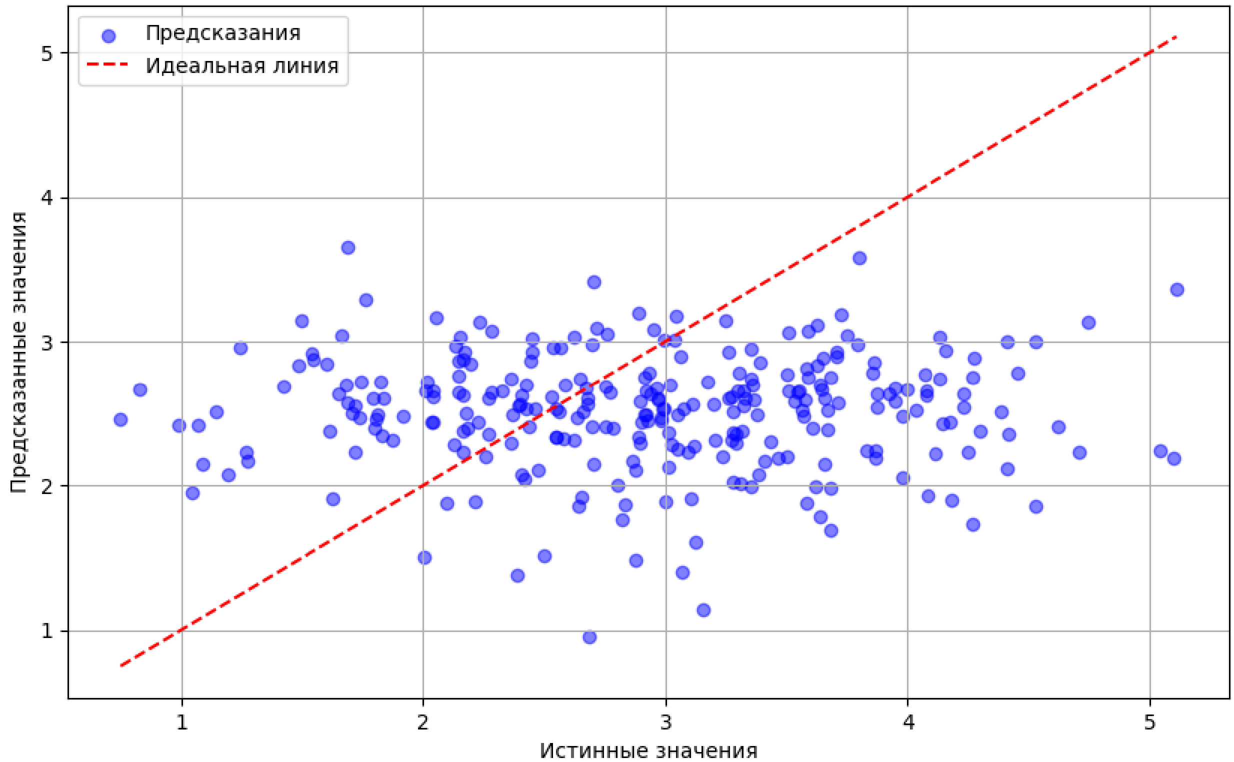
```
Epoch 111/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 26.2313 - mae: 0.7658
- mape: 26.2313 - root_mean_squared_error: 1.0197 - val_loss: 32.4753
- val_mae: 0.8265 - val_mape: 32.4753 - val_root_mean_squared_error:
1.0322
Epoch 112/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 28.1555 - mae: 0.8067 -
mape: 28.1555 - root_mean_squared_error: 1.0266 - val_loss: 32.7526 -
val_mae: 0.8311 - val_mape: 32.7526 - val_root_mean_squared_error:
1.0346
Epoch 113/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 25.9762 - mae: 0.7465
- mape: 25.9762 - root_mean_squared_error: 0.9719 - val_loss: 33.0756
- val_mae: 0.8450 - val_mape: 33.0756 - val_root_mean_squared_error:
1.0483
Epoch 114/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 26.6767 - mae: 0.7543
- mape: 26.6767 - root_mean_squared_error: 0.9760 - val_loss: 33.0809
- val_mae: 0.8398 - val_mape: 33.0809 - val_root_mean_squared_error:
1.0406
Epoch 115/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.9144 - mae: 0.7471 -
mape: 24.9144 - root_mean_squared_error: 0.9952 - val_loss: 33.1360 -
val_mae: 0.8430 - val_mape: 33.1360 - val_root_mean_squared_error:
1.0470
Epoch 116/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.9405 - mae: 0.8014
- mape: 26.9405 - root_mean_squared_error: 1.0219 - val_loss: 33.1961
- val_mae: 0.8380 - val_mape: 33.1961 - val_root_mean_squared_error:
1.0406
Epoch 117/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 27.1640 - mae: 0.7708 -
mape: 27.1640 - root_mean_squared_error: 0.9830 - val_loss: 33.1043 -
val_mae: 0.8291 - val_mape: 33.1043 - val_root_mean_squared_error:
1.0292
Epoch 118/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.7971 - mae: 0.7180 -
mape: 24.7971 - root_mean_squared_error: 0.9372 - val_loss: 33.0583 -
val_mae: 0.8298 - val_mape: 33.0583 - val_root_mean_squared_error:
1.0303
Epoch 119/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.9042 - mae: 0.7329 -
mape: 25.9042 - root_mean_squared_error: 0.9519 - val_loss: 32.9788 -
val_mae: 0.8375 - val_mape: 32.9788 - val_root_mean_squared_error:
1.0377
Epoch 120/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.6958 - mae: 0.7331 -
mape: 25.6958 - root_mean_squared_error: 0.9420 - val_loss: 32.8444 -
val_mae: 0.8351 - val_mape: 32.8444 - val_root_mean_squared_error:
```

```
1.0351
Epoch 121/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.9440 - mae: 0.6913 -
mape: 24.9440 - root_mean_squared_error: 0.8880 - val_loss: 32.7418 -
val_mae: 0.8343 - val_mape: 32.7418 - val_root_mean_squared_error:
1.0336
Epoch 122/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.8853 - mae: 0.7460 -
mape: 24.8853 - root_mean_squared_error: 0.9656 - val_loss: 32.7088 -
val_mae: 0.8245 - val_mape: 32.7088 - val_root_mean_squared_error:
1.0224
Epoch 123/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 23.9087 - mae: 0.6987 -
mape: 23.9087 - root_mean_squared_error: 0.9468 - val_loss: 32.7137 -
val_mae: 0.8229 - val_mape: 32.7137 - val_root_mean_squared_error:
1.0212
Epoch 124/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.3294 - mae: 0.7623 -
mape: 25.3294 - root_mean_squared_error: 0.9948 - val_loss: 32.7839 -
val_mae: 0.8194 - val_mape: 32.7839 - val_root_mean_squared_error:
1.0128
Epoch 125/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 24.9491 - mae: 0.7008 -
mape: 24.9491 - root_mean_squared_error: 0.9202 - val_loss: 32.7672 -
val_mae: 0.8153 - val_mape: 32.7672 - val_root_mean_squared_error:
1.0075
Epoch 126/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 26.8374 - mae: 0.7588
- mape: 26.8374 - root_mean_squared_error: 0.9739 - val_loss: 32.7533
- val_mae: 0.8174 - val_mape: 32.7533 - val_root_mean_squared_error:
1.0119
Epoch 127/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 25.8426 - mae: 0.7628 -
mape: 25.8426 - root_mean_squared_error: 0.9973 - val_loss: 32.7835 -
val_mae: 0.8144 - val_mape: 32.7835 - val_root_mean_squared_error:
1.0111
Epoch 128/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 26.6987 - mae: 0.7399
- mape: 26.6987 - root_mean_squared_error: 0.9463 - val_loss: 32.8729
- val_mae: 0.8208 - val_mape: 32.8729 - val_root_mean_squared_error:
1.0180
Epoch 129/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 24.1812 - mae: 0.7166
- mape: 24.1812 - root_mean_squared_error: 0.9603 - val_loss: 32.7913
- val_mae: 0.8280 - val_mape: 32.7913 - val_root_mean_squared_error:
1.0296
Epoch 130/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 25.1603 - mae: 0.7435 -
mape: 25.1603 - root_mean_squared_error: 0.9659 - val_loss: 32.9401 -
```

```
val_mae: 0.8302 - val_mape: 32.9401 - val_root_mean_squared_error:
1.0312
Epoch 131/1000
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 24.0846 - mae: 0.6957 -
mape: 24.0846 - root_mean_squared_error: 0.9051 - val_loss: 32.8540 -
val_mae: 0.8249 - val_mape: 32.8540 - val_root_mean_squared_error:
1.0242

plot_nn_loss(model_NN_hist_2)
```




```
y3_NN_pred_2 = model_NN_2.predict(x3_test)

plt.figure(figsize=(10, 6))
plt.scatter(y3_test, y3_NN_pred_2, alpha=0.5, color="blue",
label="Предсказания")
plt.plot([min(y3_test), max(y3_test)], [min(y3_test), max(y3_test)],
color="red", linestyle="--", label="Идеальная линия")
plt.xlabel("Истинные значения")
plt.ylabel("Предсказанные значения")
plt.legend()
plt.grid(True)
plt.show()

9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```




```
models_diff = calculate_metrics('Dummy Regressor', y3_test, y3_dummy)

models_diff = pd.concat([
    models_diff,
    calculate_metrics('Обученная нейросеть', y3_test, y3_NN_pred),
    calculate_metrics('Нейросеть модифицированная', y3_test,
y3_NN_pred_2)
],
    ignore_index=False)

styled_models_diff = style_model_results(models_diff)
styled_models_diff

<pandas.io.formats.style.Styler at 0x7a30e1f95550>
```

Нейронные сети показали себя хуже базовой модели. Обычная нейросеть продемонстрировала высокий уровень ошибки и слабую обобщающую способность. Модифицированная нейросеть несколько снизила ошибки, но по коэффициенту детерминации также показала неудовлетворительный результат. В целом, ни одна из моделей не смогла превзойти базовую по коэффициенту детерминации, что говорит о необходимости доработки архитектуры или предобработки данных.

```
all_best_results = pd.DataFrame()

all_best_results_1 = pd.DataFrame()
y1_predicted_train = best_model_1.predict(x1_train)
```

```
all_best_results_1 = pd.concat([all_best_results_1,
calculate_metrics('Train по параметру Модуль упругости при
растяжении', y1_train, y1_predicted_train)], ignore_index=False)
y1_predicted_test = best_model_1.predict(x1_test)
all_best_results_1 = pd.concat([all_best_results_1,
calculate_metrics('Test по параметру Модуль упругости при растяжении',
y1_test, y1_predicted_test)], ignore_index=False)
all_best_results_1 = all_best_results_1.round(3)

all_best_results_2 = pd.DataFrame()
y2_predicted_train = best_model_2.predict(x2_train)
all_best_results_2 = pd.concat([all_best_results_2,
calculate_metrics('Train по параметру Прочность при растяжении',
y2_train, y2_predicted_train)], ignore_index=False)
y2_predicted_test = best_model_2.predict(x2_test)
all_best_results_2 = pd.concat([all_best_results_2,
calculate_metrics('Test по параметру Прочность при растяжении',
y2_test, y2_predicted_test)], ignore_index=False)

all_best_results_3 = pd.DataFrame()
y3_predicted_train = model_NN_2.predict(x3_train)
all_best_results_3 = pd.concat([all_best_results_3,
calculate_metrics('Train по параметру Соотношение матрица-
наполнитель', y3_train, y3_predicted_train)], ignore_index=False)
y3_predicted_test = model_NN_2.predict(x3_test)
all_best_results_3 = pd.concat([all_best_results_3,
calculate_metrics('Test по параметру Соотношение матрица-наполнитель',
y3_test, y3_predicted_test)], ignore_index=False)

all_best_results = pd.concat([all_best_results_1, all_best_results_2,
all_best_results_3], ignore_index=False)
all_best_results
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

{"summary":"{\n \"name\": \"all_best_results\",\n \"rows\": 6,\n \"fields\": [\n   {\n   \"column\": \"R2\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": -0.4382509310271079,\n   \"max\": 0.03948260899449807,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   0.0,\n   -0.008878670806033773,\n   -0.4382509310271079\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"RMSE\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 0.9046027280164337,\n   \"max\": 463.1120020431918,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   2.9649952494485126,\n   3.162017010185909,\n   1.0485625130410365\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"MAE\",\n   \"properties\": {\n

```
\"dtype\": \"date\",\n        \"min\": 0.6894679907674585,\n
\"max\": 369.7091106167727,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          2.388992079234711,\n
2.580192902566283,\n          0.8673059678893359\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MAPE\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": 0.03266885937660444,\n
\"max\": 0.32093217412107017,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          0.03266885937660444,\n
0.03503243786306619,\n          0.32093217412107017\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"all_best_results"}
```


Линейная модель Lasso показала отрицательное значение R^2 на тестовом датасете, что говорит о том, что модель работает хуже, чем простое среднее значение.

Хотя бы градиентный бустинг показал положительный R^2 на тренировочном датасете, то есть модель смогла уловить часть зависимости. На тестовом датасете из-за возможного переобучения коэф детерминации чуть ниже нуля. Значения RMSE, MAE и MAPE остаются аналогично высокими.

Нейросеть показала наихудший результат среди моделей: отрицательные значения R^2 как на тренировочном, так и на тестовом датасете. Настроенная мною нейросеть не смогла уловить зависимость в данных и, возможно, просто запомнила шум. Ошибки также остаются высокими.

```
# дополнительно найду максимальные значения ошибок, чтобы оценить
насколько релевантны лучшие модели

def calculate_max_error(model_name, true_values, predicted_values):

    results = pd.DataFrame(index=[model_name])
    results.loc[model_name, 'max_error'] =
metrics.max_error(true_values, predicted_values)

    return results

all_max_errors = pd.DataFrame()

max_error_1 = pd.DataFrame()
y1_predicted_test_error = best_model_1.predict(x1_test)
max_error_1 = pd.concat([max_error_1, calculate_max_error('Test по
параметру Модуль упругости при растяжении', y1_test,
y1_predicted_test_error)], ignore_index=False)

max_error_2 = pd.DataFrame()
y2_predicted_test_error = best_model_2.predict(x2_test)
max_error_2 = pd.concat([max_error_2, calculate_max_error('Test по
параметру Прочность при растяжении', y2_test,
y2_predicted_test_error)], ignore_index=False)
```

```
max_error_3 = pd.DataFrame()
y3_predicted_test = model_NN_2.predict(x3_test)
max_error_3 = pd.concat([max_error_3, calculate_max_error('Test по
параметру Соотношение матрица-наполнитель', y3_test,
y3_predicted_test)], ignore_index=False)

all_max_errors = pd.concat([max_error_1, max_error_2, max_error_3],
ignore_index=False)
all_max_errors
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

{"summary":"{\n \"name\": \"all_max_errors\",\n \"rows\": 3,\n
\"fields\": [\n   {\n   \"column\": \"max_error\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
707.8573616721742,\n   \"min\": 2.907570627117871,\n
\"max\": 1231.4713320765804,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   7.96088467072704,\n
1231.4713320765804,\n   2.907570627117871\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"all_max_errors"}

Приложение с графическим интерфейсом, выдающее прогноз на основании созданных моделей

В качестве модели для прогнозирования буду использовать нейронную сеть на tensorflow.

```
from flask import Flask, request, render_template
import numpy as np
import tensorflow as tf
from tensorflow import keras
import os
import pickle

# создаём папку models
os.makedirs("models", exist_ok=True)

model_path = "model_NN_2.keras"
model_NN_2.save(model_path)

x3_train_df = pd.DataFrame(x3_train, columns=['var2', 'var3', 'var4',
'var5', 'var6', 'var7', 'var8', 'var9', 'var10', 'var11', 'var12',
'var13'])
```

```python
# Создание и обучение препроцессора
preprocessor = ColumnTransformer([
    ('scaler', StandardScaler(), ['var2', 'var3', 'var4', 'var5',
'var6', 'var7', 'var8', 'var9', 'var10', 'var11', 'var12', 'var13'])
])

# Обучение препроцессора на тренировочных данных
preprocessor.fit(x3_train_df)

# Сохранение препроцессора
with open('models/Preprocessor_NN_scaler.pkl', 'wb') as f:
    pickle.dump(preprocessor, f)
```