

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

---

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Кандидат технических наук,  
доцент  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

В.Ю. Скобцов  
\_\_\_\_\_  
инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

Разведочный и регрессионный анализ данных на основе  
нейросетевых моделей

по курсу: Интеллектуальный анализ данных на основе методов машинного обучения

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4134к

\_\_\_\_\_  
подпись, дата

Столяров Н.С.  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

## Задание

Дан многомерный размеченный набор данных. Необходимо выполнить регрессионный анализ данных на основе полносвязной нейросетевой модели и нейросетевой модели, указанной в варианте, в соответствии со следующей последовательностью этапов.

1. Загрузить необходимые пакеты и библиотеки.
2. Загрузить данные из указанного источника.
3. Выполнить разведочный анализ данных в соответствии с этапами описанными в файле Этапы проекта машинного обучения в примерах.pdf:
  - a. Ознакомление с данными с помощью методов описательной статистики;
  - b. Выполнить визуализацию данных одномерную для понимания распределения данных и многомерную для выяснения зависимостей между признаками;
  - c. При необходимости выполнить очистку данных одним из методов.
  - d. Проанализировать корреляционную зависимость между признаками;
  - e. Поэкспериментировать с комбинациями атрибутов. При необходимости добавить новые атрибуты в набор данных.
  - f. Выполнить отбор существенных признаков. Сформировать набор данных из существенных признаков.
  - g. При необходимости преобразовать текстовые или категориальные признаки одним из методов.
  - h. Выполнить преобразование данных для обоих наборов (исходного и сформированного) одним из методов по варианту.
4. Анализ выполняется для исходного набора данных, преобразованного исходного набора данных, построенного набора данных и преобразованного построенного набора данных. Во всех наборах данных выделить обучающую, проверочную (валидационную) и тестовую выборки данных.
5. Сравнить качество полносвязной нейросетевой регрессионной модели и регрессионной нейросетевой модели, указанной в варианте, на обучающей и валидационной выборках для всех наборов данных, включая их преобразованные варианты. Для оценки качества моделей использовать метрики: корень из среднеквадратичной ошибки, коэффициент детерминации  $R^2$ .
6. Для лучшей модели на лучшем наборе данных оценить качество на тестовом наборе.
7. Для лучшей модели на лучшем наборе данных выполнить Grid поиск лучших гиперпараметров регрессионной нейросетевой модели на обучающей и валидационной выборках. Определить значения лучших гиперпараметров.
8. Определить показатели качества полученной в результате Grid поиска регрессионной нейросетевой модели на тестовом наборе. Сравнить показатели качества лучшей модели на лучшем наборе данных до поиска гиперпараметров и после поиска гиперпараметров.
9. Сделать выводы по проведенному анализу.

## Вариант 8

Набор данных схемы пирамиды – определение прибыли или убытка. Схемы пирамид, запущенные в разных странах, часто соблазняют простых людей делать деньги в краткосрочной перспективе. Построить регрессионную модель прогностической оценки схемы пирамиды для целевого признака «profit» (выгода от схемы) от остальных входных признаков.

- a. Пункт 5 – простая рекуррентная сеть
- b. Пункт 3.h – Нормализация

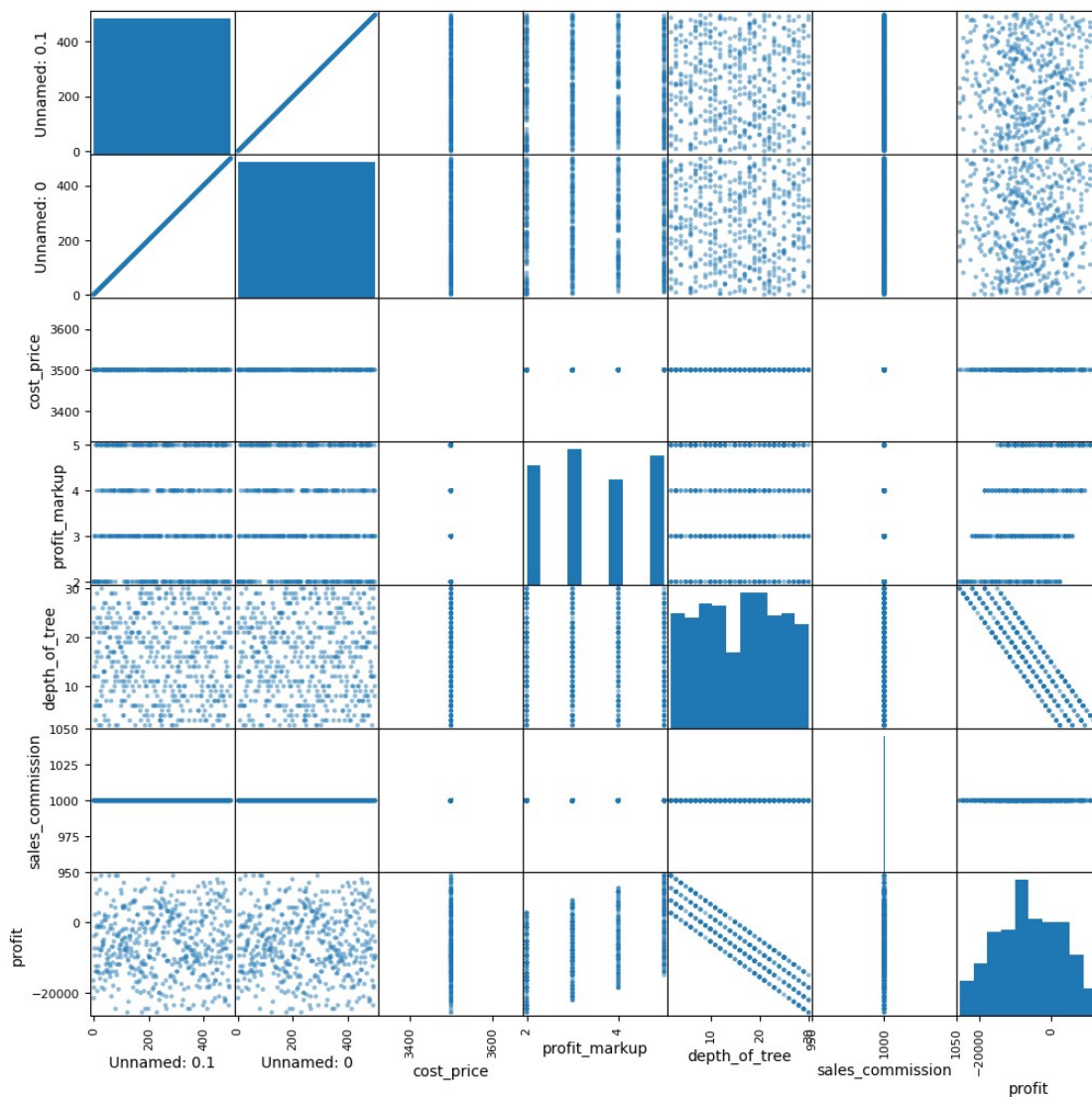
## Листинг

Сначала импортируем все необходимые библиотеки

```
import pandas as pd
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense
import os
from datetime import datetime
```

Загрузка данных из датасета, заполнение пустых значений и вывод гистограммы

```
sheet = pd.read_csv('V8.csv')
scatter_matrix(sheet, figsize=(12,12))
```



*#информация по данным датасета*

```
print(sheet.shape)
print(sheet.dtypes)
print(sheet.describe())
print(sheet.info())
```

(500, 7)

Unnamed: 0.1      int64

Unnamed: 0        int64

cost\_price        float64

profit\_markup     int64

depth\_of\_tree     int64

sales\_commission   int64

profit            int64

dtype: object

```
Unnamed: 0.1  Unnamed: 0  cost_price  profit_markup  depth_of_tree \
count  500.000000  500.000000    244.0    500.000000    500.000000
mean    249.500000  250.500000   3500.0     3.498000    15.896000
```

std	144.481833	144.481833	0.0	1.126292	8.095694
min	0.000000	1.000000	3500.0	2.000000	2.000000
25%	124.750000	125.750000	3500.0	3.000000	9.000000
50%	249.500000	250.500000	3500.0	3.000000	16.000000
75%	374.250000	375.250000	3500.0	5.000000	23.000000
max	499.000000	500.000000	3500.0	5.000000	30.000000

	sales_commission	profit
count	500.0	500.000000
mean	1000.0	-6153.000000
std	0.0	9080.528784
min	1000.0	-25500.000000
25%	1000.0	-12500.000000
50%	1000.0	-6500.000000
75%	1000.0	500.000000
max	1000.0	13000.000000

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 500 entries, 0 to 499

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
---	--------	----------------	-------

0	Unnamed: 0.1	500 non-null	int64
1	Unnamed: 0	500 non-null	int64
2	cost_price	244 non-null	float64
3	profit_markup	500 non-null	int64
4	depth_of_tree	500 non-null	int64
5	sales_commission	500 non-null	int64
6	profit	500 non-null	int64

dtypes: float64(1), int64(6)

memory usage: 27.5 KB

None

Выводы по матрице: После анализа графиков приходим к выводу, что наиболее выражено показатель profit зависит от depth\_of\_tree и после от profit\_markup depth\_of\_tree принимает значения от 3 до 30, а profit\_markup от 2 до 5 и является, скорее всего классификатором.

Теперь проверим показатели корреляции в виде карты температур для наглядности

```
correlation = sheet.corr().sort_values(by="profit",ascending=False)
sns.heatmap(correlation, vmax=1, square=True, annot=True)
print(correlation)
```

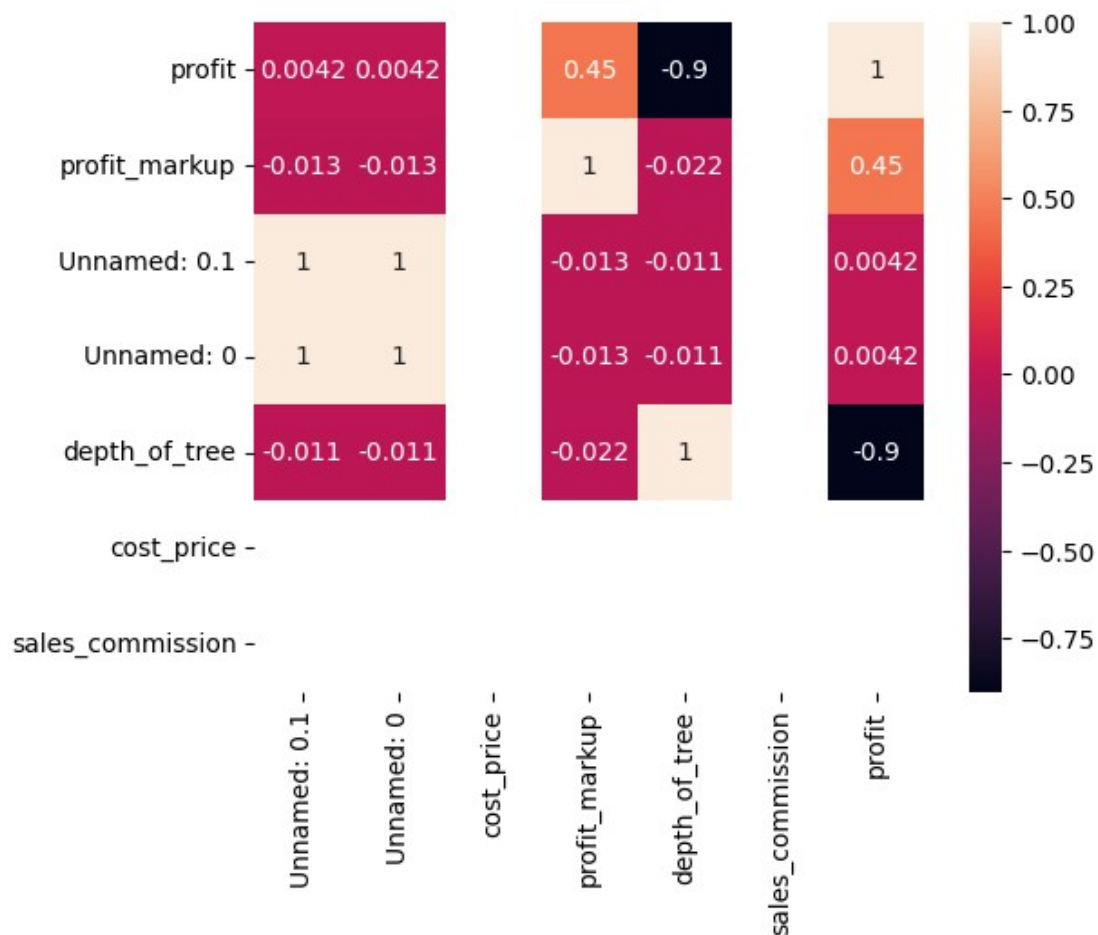
plt.figure(figsize=(5,5))

	Unnamed: 0.1	Unnamed: 0	cost_price	profit_markup \
profit	0.004159	0.004159	NaN	0.453341
profit_markup	-0.013048	-0.013048	NaN	1.000000
Unnamed: 0.1	1.000000	1.000000	NaN	-0.013048
Unnamed: 0	1.000000	1.000000	NaN	-0.013048

depth_of_tree	-0.011018	-0.011018	NaN	-0.021562
cost_price	NaN	NaN	NaN	NaN
sales_commission	NaN	NaN	NaN	NaN

	depth_of_tree	sales_commission	profit
profit	-0.900905	NaN	1.000000
profit_markup	-0.021562	NaN	0.453341
Unnamed: 0.1	-0.011018	NaN	0.004159
Unnamed: 0	-0.011018	NaN	0.004159
depth_of_tree	1.000000	NaN	-0.900905
cost_price	NaN	NaN	NaN
sales_commission	NaN	NaN	NaN

<Figure size 500x500 with 0 Axes>



<Figure size 500x500 with 0 Axes>

и Shape, dtypes

```
print(sheet.shape)
print(sheet.dtypes)
(500, 7)
Unnamed: 0.1      int64
Unnamed: 0        int64
cost_price        float64
profit_markup      int64
depth_of_tree      int64
```

```
sales_commission    int64
profit              int64
dtype: object
```

```
sheet = pd.read_csv('V8.csv')
filtered_sheet = sheet
filtered_sheet['cost_price'].fillna(3500, inplace=True)

sheet_x = filtered_sheet.iloc[:,6]
sheet_y= filtered_sheet['profit']#profit index in headers oh filtering stage

bestfeatures = SelectKBest(f_regression, k=3)
fit = bestfeatures.fit(sheet_x, sheet_y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(sheet_x.columns)
featureScores = pd.concat([dfcolumns, dfscores], axis =1)
print(featureScores)
```

```
      0      0
0  Unnamed: 0.1  0.008614
1  Unnamed: 0   0.008614
2   cost_price  0.000000
3  profit_markup 128.823664
4  depth_of_tree 2145.721436
5 sales_commission 0.000000
```

Все столбцы со значением модуля корреляции  $< 0.4$  исключаем из выборки. Оставляем только dept\_of\_tree и profit\_markup

```
filtered_sheet = sheet[['depth_of_tree', 'profit_markup']]
filtered_sheet["profit"] = sheet['profit']
```

Далее разбиваем датасет на тренировочную, тестовую и валидационную части:

1. Весь датасет в исходном виде
2. весь датасет в Нормализованном виде
3. Только столбцы со значением корреляции  $\geq 0.4$  в исходном виде
4. Только столбцы со значением корреляции  $\geq 0.4$  в Нормализованном виде

```
def get_raw_train_test_and_val_data():
    sheet = pd.read_csv('V8.csv')
    filtered_sheet = sheet
```

```

filtered_sheet['cost_price'].fillna(3500, inplace=True)

sheet_x = filtered_sheet.iloc[:,6]
sheet_y= filtered_sheet['profit']#profit index in headers oh filtering stage

seed =7
test_size = 0.2
val_size = 0.25
x_train, x_test, y_train, y_test = train_test_split(sheet_x, sheet_y, test_size = test_size,
random_state = seed)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = val_size,
random_state = seed)
return x_train, y_train, x_test, y_test, x_val, y_val

def get_normalized_train_test_and_val_data():
    sheet = pd.read_csv('V8.csv')
    filtered_sheet = sheet
    filtered_sheet['cost_price'].fillna(3500, inplace=True)

    #Нормализация
    scaler = Normalizer().fit(filtered_sheet)
    rescaled_sheet = pd.DataFrame(scaler.fit_transform(filtered_sheet))
    rescaled_sheet = rescaled_sheet.rename(columns={0:'Unnamed: 0', 1:"cost_price",
2:'profit_markup', 3: "3",
4:"depth_of_tree", 5:"sales_commission", 6:"profit"})

    #print(rescaled_sheet)

    #разделение наборов на тренировочный и тестовый и проверочный
    sheet_x = rescaled_sheet.iloc[:,6]
    sheet_y= rescaled_sheet['profit']#profit index in headers oh filtering stage

    seed =7
    test_size = 0.2
    val_size = 0.25
    x_train, x_test, y_train, y_test = train_test_split(sheet_x, sheet_y, test_size = test_size,
random_state = seed)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = val_size,
random_state = seed)

    return x_train, y_train, x_test, y_test, x_val, y_val

def get_raw_filtered_train_test_and_val_data():
    sheet = pd.read_csv('V8.csv')
    filtered_sheet = sheet[['depth_of_tree', 'profit_markup']]
    filtered_sheet["profit"] = sheet['profit']

    sheet_x = filtered_sheet.iloc[:,2]

```



```

sheet_y= filtered_sheet['profit']#profit index in headers oh filtering stage

seed =7
test_size = 0.2
val_size = 0.25
x_train, x_test, y_train, y_test = train_test_split(sheet_x, sheet_y, test_size = test_size,
random_state = seed)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = val_size,
random_state = seed)
return x_train, y_train, x_test, y_test, x_val, y_val

def get_normalized_filtered_train_test_and_val_data():
    sheet = pd.read_csv('V8.csv')
    filtered_sheet = sheet[['depth_of_tree', 'profit_markup']]
    #filtered_sheet["type_to_depth_rel"] = sheet['profit_markup']/sheet["depth_of_tree"]
    filtered_sheet["profit"] = sheet['profit']

    #Нормализация
    scaler = Normalizer().fit(filtered_sheet)
    rescaled_sheet = pd.DataFrame(scaler.fit_transform(filtered_sheet))
    rescaled_sheet = rescaled_sheet.rename(columns={0:'depth_of_tree', 1:"profit_markup",
2:'profit'})
    #print(rescaled_sheet)

    #разделение наборов на тренировочный и тестовый и проверочный
    sheet_x = rescaled_sheet.iloc[:, :2]
    sheet_y= rescaled_sheet['profit']#profit index in headers oh filtering stage

    seed =7
    test_size = 0.2
    val_size = 0.25
    x_train, x_test, y_train, y_test = train_test_split(sheet_x, sheet_y, test_size = test_size,
random_state = seed)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = val_size,
random_state = seed)

    return x_train, y_train, x_test, y_test, x_val, y_val

```

Теперь используем каждый набор для обучения и проверки для Линейной регрессионной модели и Простой рекуррентной сети, все результаты вывожу в main\_output.txt, который прикреплен ниже

```

for data_function in [
    get_normalized_filtered_train_test_and_val_data,
    get_normalized_train_test_and_val_data,

```

```
get_raw_filtered_train_test_and_val_data,  
get_raw_train_test_and_val_data]:
```

```
x_train, y_train, x_test, y_test, x_val, y_val = data_function()
```

```
#####  
#####
```

```
model = Sequential()  
model.add(SimpleRNN(150, activation='relu', input_shape=(x_train.shape[1], 1)))  
model.add(Dense(1)) # Выходной слой
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Обучение модели
```

```
model.fit(x_train, y_train, epochs=100, batch_size=32, validation_data=(x_val, y_val))
```

```
# Прогнозирование
```

```
y_train_pred = model.predict(x_train)
```

```
y_test_pred = model.predict(x_test)
```

```
y_val_pred = model.predict(x_val)
```

```
# Оценка модели
```

```
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
```

```
r2_train = r2_score(y_train, y_train_pred)
```

```
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
```

```
r2_test = r2_score(y_test, y_test_pred)
```

```
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))
```

```
r2_val = r2_score(y_val, y_val_pred)
```

```
with open('output.txt', 'a+') as output_file:
```

```
    output_file.write(f"Data function: {data_function.__name__}\n")
```

```
    output_file.write(f"Train RMSE for RNN: {rmse_train}\n")
```

```
    output_file.write(f"Train R2 Score for RNN: {r2_train}\n")
```

```
    output_file.write(f"Test RMSE for RNN: {rmse_test}\n")
```

```
    output_file.write(f"Test R2 Score for RNN: {r2_test}\n")
```

```
    output_file.write(f"Valid RMSE for RNN: {rmse_val}\n")
```

```
    output_file.write(f"Valid R2 Score for RNN: {r2_val}\n")
```

```
#####  
#####
```

```
model = Sequential()
```

```
model.add(Dense(60, activation='relu', input_dim=x_train.shape[1]))
```

```
model.add(Dense(150, activation='relu'))
```

```
model.add(Dense(1))
```

```

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=100, batch_size=32, validation_data=(x_val, y_val))

y_prediction = model.predict(x_train)
y_test_pred = model.predict(x_test)
y_val_pred = model.predict(x_val)
    # Оценка модели
rmse_train = np.sqrt(mean_squared_error(y_train, y_prediction))
r2_train = r2_score(y_train, y_prediction)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
r2_test = r2_score(y_test, y_test_pred)
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))
r2_val = r2_score(y_val, y_val_pred)

with open('output.txt', 'a+') as output_file:
    output_file.write("\n")
    output_file.write(f"Data function: {data_function.__name__}\n")
    output_file.write(f"Train RMSE for Dense: {rmse_train}\n")
    output_file.write(f"Train R2 Score for Dense: {r2_train}\n")
    output_file.write(f"Test RMSE for Dense: {rmse_test}\n")
    output_file.write(f"Test R2 Score for Dense: {r2_test}\n")
    output_file.write(f"Valid RMSE for Dense: {rmse_val}\n")
    output_file.write(f"Valid R2 Score for Dense: {r2_val}\n\n")
    output_file.write("#####\n\n")

```

## Результаты в output.txt

Data function: get\_raw\_filtered\_train\_test\_and\_val\_data Train RMSE for RNN:  
4518.686692937692 Train R2 Score for RNN: 0.7396427392959595 Test RMSE for RNN:  
4719.987685935309 Test R2 Score for RNN: 0.7633601427078247 Valid RMSE for RNN:  
4485.821421014299 Valid R2 Score for RNN: 0.7434529662132263

Data function: get\_raw\_filtered\_train\_test\_and\_val\_data Train RMSE for Dense:  
1377.3300917743718 Train R2 Score for Dense: 0.9758108258247375 Test RMSE for Dense:  
1372.2465549176563 Test R2 Score for Dense: 0.9799981117248535 Valid RMSE for Dense:  
1337.019611463902 Valid R2 Score for Dense: 0.9772092700004578

## Поиск Гиперпараметров для Dense

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from scikeras.wrappers import KerasRegressor

```

```

# Load data
x_train, y_train, x_test, y_test, x_val, y_val = get_raw_filtered_train_test_and_val_data()

# Function to create model
def create_model(optimizer='adam', neurons=32):
    model = Sequential()
    model.add(Dense(neurons, activation='relu', input_dim=x_train.shape[1]))
    model.add(Dense(150, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
    return model

# KerasClassifier wrapper
model = KerasRegressor(model=create_model, verbose=0, neurons =32)

param_grid = {
    'model__neurons': [32, 64, 128, 150], # Количество нейронов
    'batch_size': [8, 16, 32], # Размер батча
    'model__optimizer': ['adam', 'RMSprop'], # Оптимизаторы
    'epochs': [100, 200] # Количество эпох
}

# Perform Grid Search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(x_train, y_train)

# Output best parameters
print("Лучшие параметры: ", grid_result.best_params_)
print("Лучший балл: ", grid_result.best_score_)

```

Лучшие параметры: {'batch\_size': 8, 'epochs': 200, 'model\_\_neurons': 128, 'model\_\_optimizer': 'RMSprop'} Лучший балл: 0.999963382879893

обучаем выбранную модель и делаем ее вырезку из памяти

```

import pandas as pd
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

```

```

from sklearn.preprocessing import PolynomialFeatures
import numpy as np
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense
from pickle import dump, load
from datetime import datetime

def get_raw_filtered_train_test_and_val_data():
    sheet = pd.read_csv('V8.csv')
    filtered_sheet = sheet[['depth_of_tree', 'profit_markup']]
    filtered_sheet["profit"] = sheet['profit']

    sheet_x = filtered_sheet.iloc[:,2]
    sheet_y= filtered_sheet['profit']#profit index in headers oh filtering stage

    seed =7
    test_size = 0.2
    val_size = 0.25
    x_train, x_test, y_train, y_test = train_test_split(sheet_x, sheet_y, test_size = test_size,
random_state = seed)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = val_size,
random_state = seed)
    return x_train, y_train, x_test, y_test, x_val, y_val

if __name__ == "__main__":
    with open('Dense_teaching_predictions.txt', 'w') as output_file:
        output_file.write(f"{datetime.now()}\n")

    x_train, y_train, x_test, y_test, x_val, y_val = get_raw_filtered_train_test_and_val_data()
    print(x_train)

    model = Sequential()
    model.add(Dense(128, activation='relu', input_dim=x_train.shape[1]))
    model.add(Dense(150, activation='relu'))
    model.add(Dense(1))

    model.compile(optimizer='RMSprop', loss='mean_squared_error')
    model.fit(x_train, y_train, epochs=200, batch_size=8, validation_data=(x_val, y_val))

# Прогнозирование
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)

# Оценка модели
    rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
    r2_train = r2_score(y_train, y_train_pred)
    rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
    r2_test = r2_score(y_test, y_test_pred)

```

```

with open('Dense_teaching_predictions.txt', 'a') as output_file:
    output_file.write(f"Train RMSE: {rmse_train}\n")
    output_file.write(f"Train R2 Score: {r2_train}\n")
    output_file.write(f"Test RMSE: {rmse_test}\n")
    output_file.write(f"Test R2 Score: {r2_test}\n")
    output_file.write(f"#####\n")

with open('Dense_teaching_predictions.txt', 'a') as f:
    f.write(f"actual, predicted\n")
    for actual, predicted in zip(y_test, y_test_pred):
        f.write(f"{actual}, {predicted} \n")

filename = 'Dense.sav'
dump(model, open(filename, 'wb'))

```

Подготовка дампа памяти модели для дальнейшего использования

```

from joblib import load
import numpy as np

model = load('Dense.sav')

x = np.array([[20, 4]]) #Сначала depth_of_tree, затем profit_markup

y_pred = model.predict(x)

print(y_pred)

```