чМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| Разведочный и регресионный анализ данных на основе нейросетевых моделей |
| по курсу: интеллектуальный анализ данных на основе методов машинного обучения |
|  |
|  |

РАБОТУ ВЫПОЛНИЛА

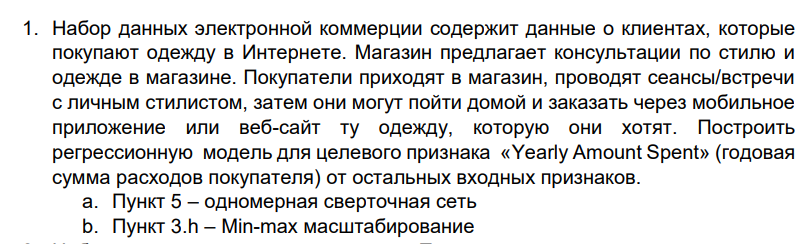
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4136 |  |  |  | Н.С. Бобрович |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы**

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.

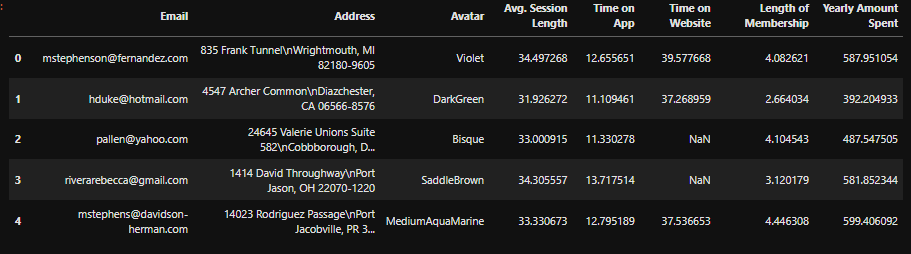
1. **Вариант 1:**



1. **Ход работы**
   1. **Анализ данных**

df = pd.read\_csv('V1.csv', index\_col=0)

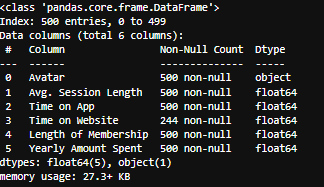
df.head()



Избавимся от текстовых данных

df = df.drop(['Email', 'Address'], axis=1)

df.info()



df['Avatar'].unique

Признак Avatar представляет так же текстовый признак и от него можно избавиться

df = df.drop(['Avatar'], axis=1)

Построим матрицу корреляции для оставшихся (числовых) признаков и оценим их влияение на целевой признак Yearly Amount Spent

df['Time on Website'].fillna(0, inplace=True)

corr = df.corr()

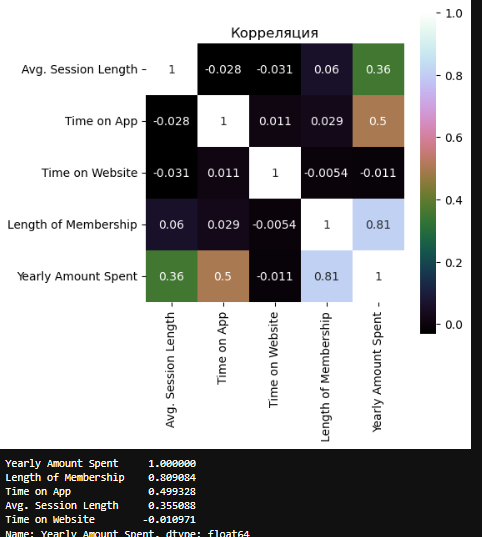
plt.figure(figsize=(5,5))

plt.title("Корреляция")

sns.heatmap(corr, vmax=1, square=True, annot = True, cmap='cubehelix')

plt.show()

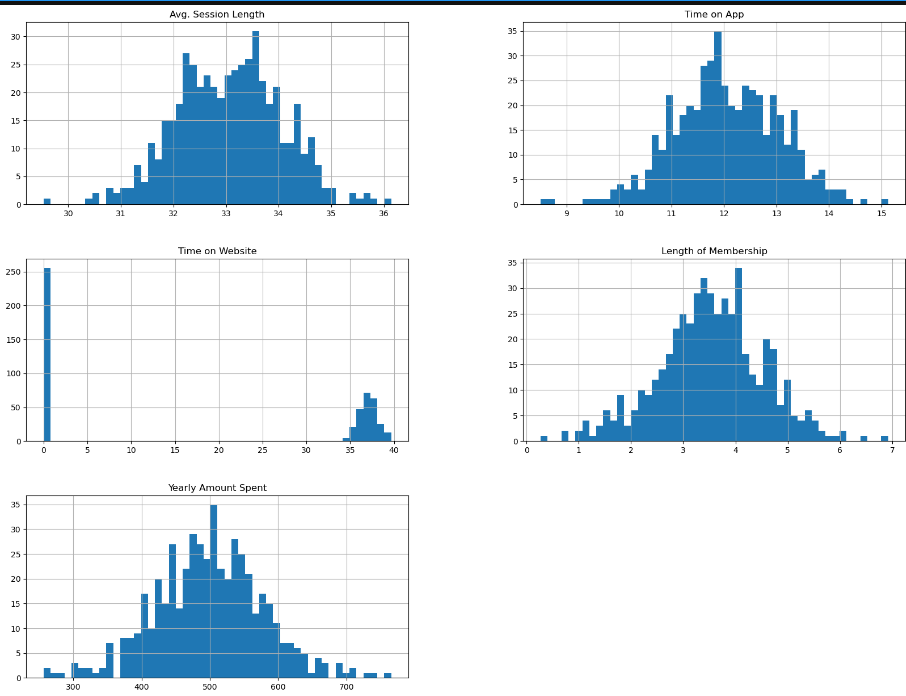
corr["Yearly Amount Spent"].sort\_values(ascending=False)



Все признаки оказывают существенное влияние на целевой признак

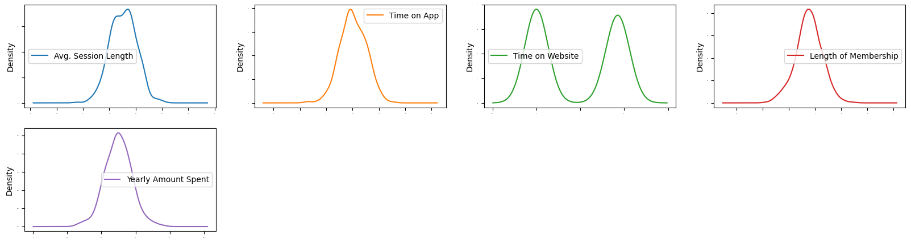
df.hist(bins=50,figsize=(20,15))

plt.show()



df.plot(kind='density', subplots=True, layout=(3,4), sharex=False, legend=True, fontsize=1, figsize=(20,8))

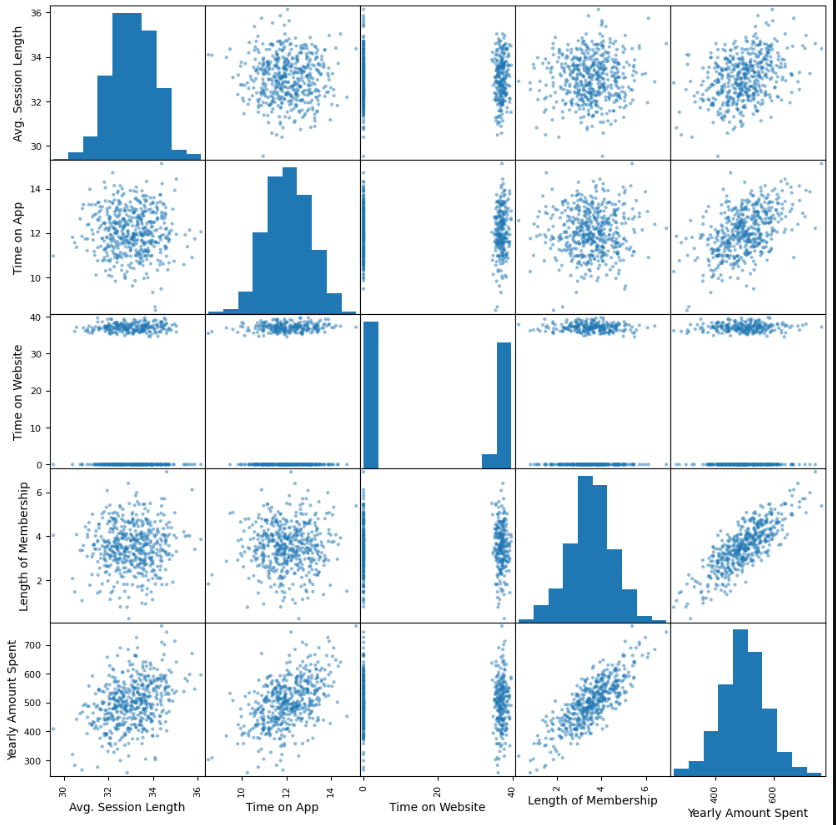
plt.show()



По гистограммам и графикам плотности можно сделать вывод, что распределение всех признаков, примерно соответствует нормальному

scatter\_matrix(df, figsize=(12,12))

plt.show()



Все признаки хорошо коррелируют с целевым

X = df.drop(['Yearly Amount Spent'], axis=1)

Y = df['Yearly Amount Spent']

X

Мы создали исходный датасет. Создадим построенный, путём создания дополнительных признаков

df["Total time"] = df["Time on Website"] + df["Time on App"]

corr = df.corr()

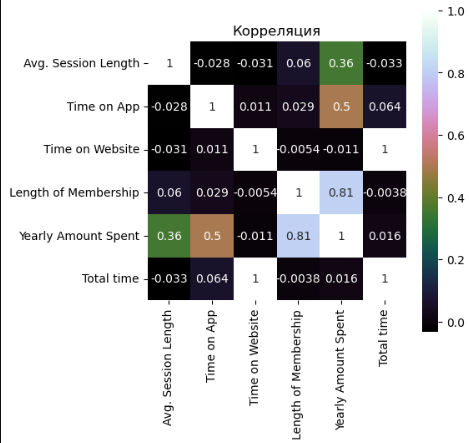
plt.figure(figsize=(5,5))

plt.title("Корреляция")

sns.heatmap(corr, vmax=1, square=True, annot = True, cmap='cubehelix')

plt.show()

corr["Yearly Amount Spent"].sort\_values(ascending=False)



Как можно заметить, созданный признак имеет влияение на целевой

X\_with\_new\_atribbute = df.drop(['Yearly Amount Spent'], axis=1)

Y\_with\_new\_atribbute = df['Yearly Amount Spent']

Используя MinMaxScaler выполним масштабирование построенного и исходного датасета в диапазон от 0 до 1

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

y\_scaled = scaler.fit\_transform(Y.values.reshape(-1, 1))

X\_with\_new\_atribbute\_scaled = scaler.fit\_transform(X\_with\_new\_atribbute)

Y\_with\_new\_atribbute\_scaled = scaler.fit\_transform(Y\_with\_new\_atribbute.values.reshape(-1, 1))

X\_scaled

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

X\_train\_extra, X\_test\_extra, y\_train\_extra, y\_test\_extra = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.2, random\_state=42)

X\_test\_extra, X\_dev\_extra, y\_test\_extra, y\_dev\_extra = train\_test\_split(X\_test\_extra, y\_test\_extra, test\_size=0.5, random\_state=42)

Создадим обучающу, валидационную и тестовую выборку. Поскольку размер датасета небольшой, размер обучающей выборки равен 60%, размер тестовой выборки равен 20% и размер валидационной выборки тоже 20%

* 1. **Создание и обучение моделей**

Создадим 4 модели

model - сверточная модель для данных без дополнительных параметров

model\_2 - сверточная модель для данных с дополнительным параметром

model\_dense - полносвязная модель для данных без дополнительных параметров

model\_2\_dense - полносвязная модель для данных с дополнительным параметром

Выполним обучения каждой модели на датасете без масштабирования и с масштабированием

model\_dense = Sequential()

model\_dense.add(Input(shape=(X\_train.shape[1],)))

model\_dense.add(Dense(units=64))

model\_dense.add(Dense(units=32))

model\_dense.add(Dense(units=1))

model\_2\_dense = Sequential()

model\_2\_dense.add(Input(shape=(X\_train\_extra.shape[1],)))

model\_2\_dense.add(Dense(units=64))

model\_2\_dense.add(Dense(units=32))

model\_2\_dense.add(Dense(units=1))

X\_train = X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1]).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1]).astype('float32')

model = Sequential()

model.add(Input(shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Conv1D(filters=32, kernel\_size=1))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(1000))

model.add(Dense(1))

X\_train\_extra = X\_train\_extra.reshape(X\_train\_extra.shape[0], 1, X\_train\_extra.shape[1]).astype('float32')

X\_test\_extra = X\_test\_extra.reshape(X\_test\_extra.shape[0], 1, X\_test\_extra.shape[1]).astype('float32')

model\_2 = Sequential()

model\_2.add(Input(shape=(X\_train\_extra.shape[1], X\_train\_extra.shape[2])))

model\_2.add(Conv1D(filters=32, kernel\_size=1))

model\_2.add(MaxPooling1D(pool\_size=1))

model\_2.add(Flatten())

model\_2.add(Dense(1000))

model\_2.add(Dense(1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

history = model\_dense.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model\_dense.evaluate(X\_test, y\_test)

predicted = model\_dense.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Dense, scaled mse: {mse}, r2: {r2}')

X\_train = X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1]).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1]).astype('float32')

X\_dev = X\_dev.reshape(X\_dev.shape[0], 1, X\_dev.shape[1]).astype('float32')

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Conv, scaled mse: {mse}, r2: {r2}')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model\_2\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

history = model\_2\_dense.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model\_2\_dense.evaluate(X\_test, y\_test)

predicted = model\_2\_dense.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Dense, scaled, extra mse: {mse}, r2: {r2}')

X\_train = X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1]).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1]).astype('float32')

X\_dev = X\_dev.reshape(X\_dev.shape[0], 1, X\_dev.shape[1]).astype('float32')

model\_2.compile(optimizer='adam', loss='mean\_squared\_error')

history = model\_2.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model\_2.evaluate(X\_test, y\_test)

predicted = model\_2.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Conv, scaled, extra mse: {mse}, r2: {r2}')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

history = model\_dense.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model\_dense.evaluate(X\_test, y\_test)

predicted = model\_dense.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Dense mse: {mse}, r2: {r2}')

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

X\_dev = np.array(X\_dev)

X\_train = X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1]).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1]).astype('float32')

X\_dev = X\_dev.reshape(X\_dev.shape[0], 1, X\_dev.shape[1]).astype('float32')

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Conv mse: {mse}, r2: {r2}')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute, Y\_with\_new\_atribbute, test\_size=0.5, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model\_2\_dense.compile(optimizer='adam', loss='mean\_squared\_error')

history = model\_2\_dense.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

mse = model\_2\_dense.evaluate(X\_test, y\_test)

predicted = model\_2\_dense.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print(f'Dense, extra mse: {mse}, r2: {r2}')

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

X\_dev = np.array(X\_dev)

X\_train = X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1]).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1]).astype('float32')

X\_dev = X\_dev.reshape(X\_dev.shape[0], 1, X\_dev.shape[1]).astype('float32')

model\_2.compile(optimizer='adam', loss='mean\_squared\_error')

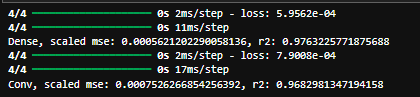
history = model\_2.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

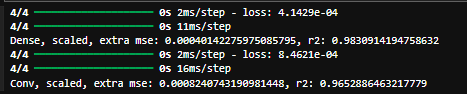
mse = model\_2.evaluate(X\_test, y\_test)

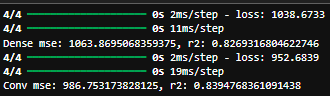
predicted = model\_2.predict(X\_test)

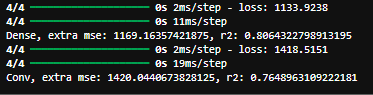
r2 = r2\_score(y\_test, predicted)

print(f'Conv, extra mse: {mse}, r2: {r2}')









Как можно заметить, лучший результат показала полносвязная нейронная сеть с массштабированием и дополнительным признаком

Для неё проведём Grid поиск параметров: количество нейронов в слое, количество эпох, лучший оптимизатор

1. **Grid поиск гиперпараметров**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

Deg = [2,3,4,5,6,7]

res = []

res\_r2 = []

names = []

for deg in Deg:

model = Sequential()

model.add(Input(shape=(X\_train.shape[1],)))

model.add(Dense(units=2\*\*deg))

model.add(Dense(units=2\*\*(deg-1)))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

rmse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

res.append(rmse)

res\_r2.append(r2)

names.append(deg)

print("Deg = ", deg)

print("RMSE = ", rmse)

print("R2 = ", r2, "\n-----------------")

print ("RMSEs: ", res)

print ("Names: ", names)

print ("R2s: ", res\_r2) 

Точность при 16, 32, 64, 128 нейронах достигается высокая. Самое высокое значение R2 и минимальное RMSE при 164 нейронах

Epohs = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

res = []

res\_r2 = []

names = []

for ep in Epohs:

model = Sequential()

model.add(Input(shape=(X\_train.shape[1],)))

model.add(Dense(units=128))

model.add(Dense(units=64))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

rmse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

res.append(rmse)

res\_r2.append(r2)

names.append(ep)

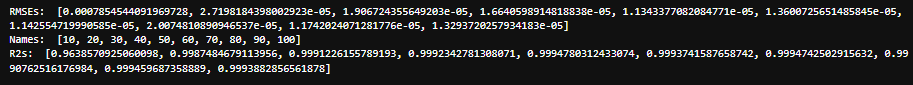
print("Epohs = ", ep)

print("RMSE = ", rmse)

print("R2 = ", r2, "\n-----------------")

print ("RMSEs: ", res)

print ("Names: ", names)

print ("R2s: ", res\_r2)

Точность при 100 эпохах самая лучшая.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

Opts = ['adam', 'SGD', 'Adamax']

res = []

res\_r2 = []

names = []

for op in Opts:

model = Sequential()

model.add(Input(shape=(X\_train.shape[1],)))

model.add(Dense(units=128))

model.add(Dense(units=64))

model.add(Dense(units=1))

model.compile(optimizer=op, loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=70, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

rmse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

res.append(rmse)

res\_r2.append(r2)

names.append(op)

print("Ops = ", op)

print("RMSE = ", rmse)

print("R2 = ", r2, "\n-----------------")

print ("RMSEs: ", res)

print ("Names: ", names)

print ("R2s: ", res\_r2)



Лучший результат показывает оптимизатор adam

Итоговая модель, с лучшими гиперпараметрами выглядит так:

- Полносвязная модель, использующая данные с масштабированием и дополнительным признаком

- Оптимизатор adam

- Количество эпох для обучения 100

- Количество нейронов в первом и втором слоях соответсвенно: 128, 64

Построим данную модель и вычислим её точность по метрике MSE и R2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model = Sequential()

model.add(Input(shape=(X\_train.shape[1],)))

model.add(Dense(units=128))

model.add(Dense(units=64))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=70, batch\_size=16, verbose=0, validation\_data=(X\_dev, y\_dev))

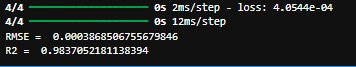
rmse = model.evaluate(X\_test, y\_test)

predicted = model.predict(X\_test)

r2 = r2\_score(y\_test, predicted)

print("RMSE = ", rmse)

print("R2 = ", r2)



Итоговые значение RMSE и R2 для данной модели: 0.0003 и 0.9837 соответсвенно

1. **Выводы**

В результате выполнения лабораторной работы были освоены навыки разворачивания сверточной сети, использования методов масштабирования тестовых данных, их анализа и подготовки к работе, заполнения и преобразования, разбиения на тестовую, валидационную и обучающую выборки и реализована сверточная сеть, работающая с этими данными и прогнозирующая целевой признак в зависимости от других. Лучший результат показала полносвязная сеть, использующая масштабированный измененный набор данных, с оптимизатором adam, количеством эпох 100 и количеством нейронов в верхнем слое равном 128