MINISTRY OF EDUCATION OF REPUBLIC OF MOLDOVA

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automatics

Report

on Artificial Intelligence Fundamentals

Laboratory Work nr. III

Performed by: **Boaghe Mariana, Ejova Ecaterina**, FAF -171, FAF-172

Verified by: **Mihail Gavrilița**, asist. univ.

Chișinău, 2021

Contents

1 Introduction

We went to live at the ”Golden Rule” space station. Mariana and me have to find an apartment, but all the consulting companies ask for incredible amounts of money. Therefore, we turned to our mentor for advice. He our superman and cannot leave us in a difficult situation. This is where our adventures in search of a suitable apartment begin.

If you have any problems/difficulties finding an apartment, please contact us. We went through this situation and will help you!

2 The Task

The company is sending you to live at the ”Golden Rule” space station. And while the variable gravity doesn’t seem to bother you, the prices for housing there do. You would like to know how much an apartment could cost but all the consulting companies ask for incredible amounts of money, so the solution is thus obvious – you’ll need to find out yourself.

Being a qualified AI specialist you decide to over-engineer it a bit and create a program that could predict the price of an apartment given a set of characteristics of the apartment complex it is part of. You get your hands on some unformatted historical data about apartment complexes on the space station and their prices. The old man who supplied the data said that the fields that you might be interested are complexAge in column 3, totalRooms in column 4, totalBedrooms in column 5, complexInhabitants in column 6, apartmentsNr in column 7 and medianCompexValue in column 9.

Because running personal projects on production machines is a big no-no at the company, you’ve dug up an old server that can run Google Colab Notebooks and has Keras pre-installed as the only ML library (beggars can’t be choosers). You’ll need to create a model that would perform linear regression on the available data. As a result, the model should be able to predict the medianCompexValue of an apartment complex, given a set of it’s characteristics. Perform some simple statistical tests to determine the accuracy of your model.

3 Solution Description

First we need install Keras:

```
#install keras
!pip install -q keras
```

and setup libraries:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

Data loading preprocessing: the file must be loaded from the folder. For this we use the library and after launch, select the file that we need.

```
from google.colab import files
uploaded = files.upload()
```

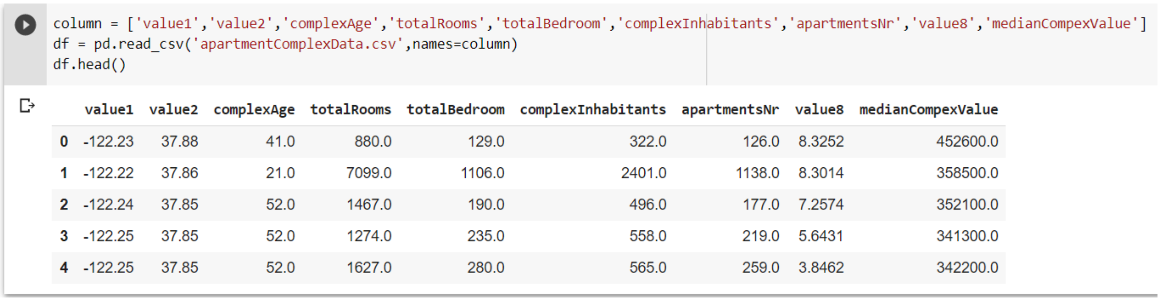
Since Keras does not read files, so we can change the file extension to csv and read it using pandas:

```
import pandas as pd

df = pd.read_csv('apartmentComplexData.csv')
```

Our file is clean, no spaces, no NaN, so cleaning is not needed.

Next, we create columns, as a parameter we pass to df. The first, second and eighth columns of the assignment do not have a title, so I gave them titles: value1, value2 and value8:

```
column = ['value1','value2','complexAge','totalRooms','totalBedroom','
    complexInhabitants','apartmentsNr','value8','medianCompexValue']
df = pd.read_csv('apartmentComplexData.csv',names=column)
print(df)
```



```
column = ['value1','value2','complexAge','totalRooms','totalBedroom','complexInhabitants','apartmentsNr','value8','medianCompexValue']
df = pd.read_csv('apartmentComplexData.csv',names=column)
df.head()
```

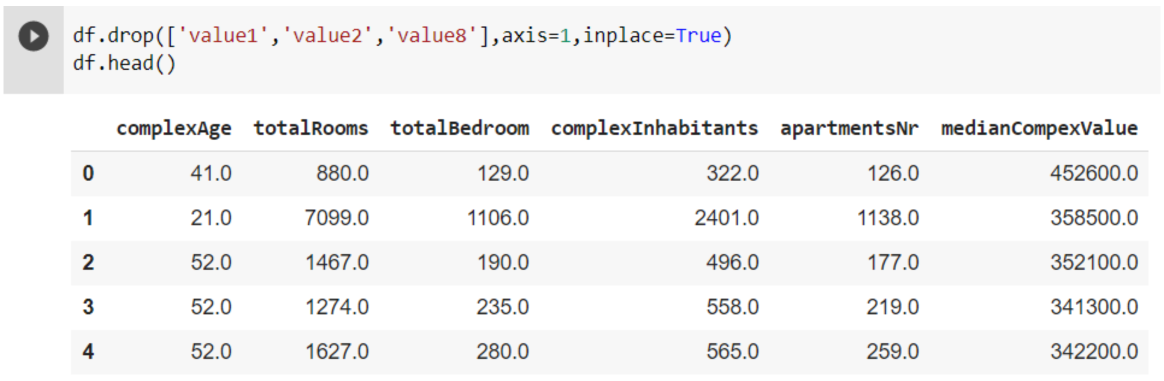
	value1	value2	complexAge	totalRooms	totalBedroom	complexInhabitants	apartmentsNr	value8	medianCompexValue
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

Pandas consist of drop function which is used in removing rows or columns from the CSV files:

```
df.drop(['value1','value2','value8'],axis=1,inplace=True)
df.head()
```

3.1 Linear Regression

We are given the initial data in the form of a table with columns complexAge, totalRooms, totalBedroom, complexInhabitants, apartmentsNr and medianCompexValue. And we will build a linear model of dependence of medianCompexValue on other columns, that is, the model will look like this:



```
df.drop(['value1','value2','value8'],axis=1,inplace=True)
df.head()
```

	complexAge	totalRooms	totalBedroom	complexInhabitants	apartmentsNr	medianCompexValue
0	41.0	880.0	129.0	322.0	126.0	452600.0
1	21.0	7099.0	1106.0	2401.0	1138.0	358500.0
2	52.0	1467.0	190.0	496.0	177.0	352100.0
3	52.0	1274.0	235.0	558.0	219.0	341300.0
4	52.0	1627.0	280.0	565.0	259.0	342200.0

$$y = b0 + b1x1 + b2x2 + b3x3 + b4x4 + b5x5 + \epsilon$$

where y is medianCompexValue; x1,x2,x3,x4 and x5 are other columns; b - model parameters, $\epsilon - randomvariable.$

Getting access by columns:

```
dataset = df[['complexAge','totalRooms','totalBedroom','complexInhabitants','
    apartmentsNr','medianCompexValue']]
```

We have 6 columns left after deleting unnecessary ones. Now the last column is medianCompexValue since our model should be able to predict the medianCompexValue of an apartment complex, given a set of it's characteristics. Therefore, we indicate the number 5 in dataset.iloc[:,0:5]. It gives us a 5-d dataframe (columns from 0 to 5):

```
feature_dataset = dataset.iloc[:,0:5]
target_dataset = dataset.iloc[1−,:]
```

And we pull out the last column like this: **dataset.iloc[1-,:]**. As in the list, access to data can be from the end using negative integers, that is the same here.

This is access to data in the dataframe line by line. ”i” allows us to specify a range of strings.

We need to use **StandardScaler**. The idea behind StandardScaler is that it transforms our data so that its distribution has a mean of 0 and a standard deviation of 1.

We need to smooth out the spread of data from minimum to maximum values and prepare the data for writing them to Keras.

```
from sklearn.preprocessing import StandardScaler

# scale feature dataset
sc_X = StandardScaler()
X = sc_X.fit_transform(np.array(feature_dataset))
# scale target dataset
sc_y = StandardScaler()
```

```
y = sc_y.fit_transform(np.array(target_dataset).reshape(-1,1))
```

However, the question is what to use: StandardScaler or MinMaxScaler. StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance. MinMaxScaler scales all the data features in the range [0, 1] or else in the range [-1, 1] if there are negative values in the dataset.

I believe MinMaxScaler gives an effective transformation. Because before MinMaxScaler, the model error was 0.5 and after it became 0.2 and below.Therefore, I use MinMaxScaler:

```
from sklearn.preprocessing import MinMaxScaler
sc_X = MinMaxScaler()
sc_y = MinMaxScaler()
X = sc_X.fit_transform(np.array(feature_dataset))
y = sc_y.fit_transform(np.array(target_dataset).reshape(-1,1))
```

Split arrays or matrices into random train and test subsets. **test size**: If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. I set to 0.20.

```
# train test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    shuffle=True)
```

This is where the learning takes place:

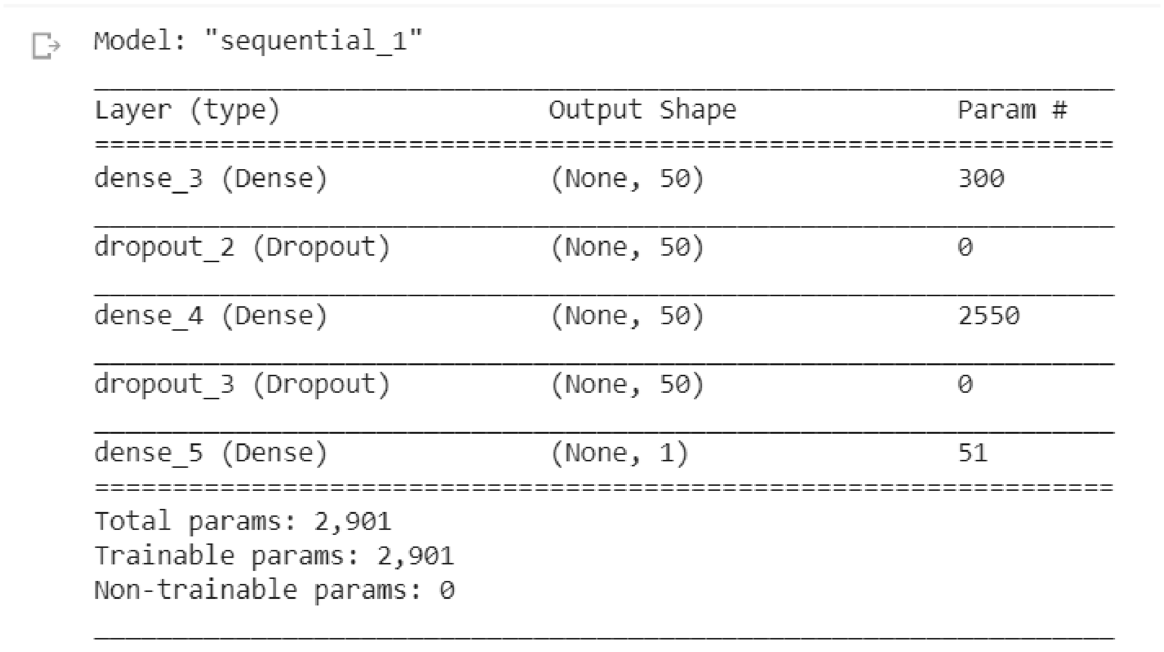
```
# train the model in 32 batch size and 100 epochs
model.fit(X_train, y_train, batch_size=32, epochs=300)
```

There are several neurons. Each neuron has a formula, the result of each formula is a coefficient. The input data (dataset) goes through the layers of the neural network and leaves a coefficient in each neuron. When the data is received, the system itself compares what happened and the target_dataset. Having received a discrepancy, it goes back along the same neurons and checks which of them messed up and got bad coefficients, corrects it and starts all over again. And so 100 times as indicated in the settings epochs = 300.

I create a Sequential model by passing a list of layer instances to the constructor and added layers via the .add() method. Dropout is a technique to address the issue of overfitting, and it randomly drops the hidden and visible units in the neural networks.

```
# Sequential Artificial Neural Network Model
from keras.models import Sequential
from keras.layers import Dense, Dropout

```

```
# define the keras model
model = Sequential()
# Layer 1
model.add(Dense(50, input_dim=X_train.shape[1], activation='sigmoid'))
# Dropout regularization is added to avoid overfitting
model.add(Dropout(0.1))
# Layer 2
model.add(Dense(50, activation='sigmoid'))
# Dropout regularization is added to avoid overfitting
model.add(Dropout(0.1))
# Layer 3
# Output Layer
model.add(Dense(1))
#Compile the model
model.compile(loss='mae', optimizer='adam', metrics=['mae'])

model.summary()
```

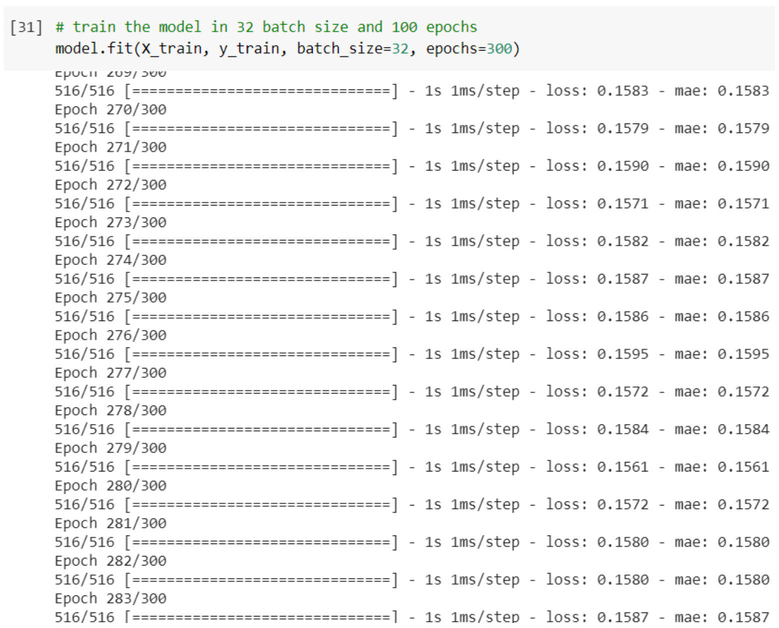


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 50)                300       
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550      
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51        
=================================================================
Total params: 2,901
Trainable params: 2,901
Non-trainable params: 0
_________________________________________________________________
```

Mean Absolute Error: mean_absolute_error, MAE, mae **Adam** optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The **Sigmoid** function (also known as the Logistic function) is one of the most widely used activation function. To use the Sigmoid activation function with Keras, we can simply pass 'sigmoid' to the argument activation.

Last layer use "sigmoid" activation, which means it will return an array of 50 probability scores (summing to 1). Each score will be the probability that the current digit image belongs to one of our 50 digit classes.

```
# train the model in 32 batch size and 100 epochs
model.fit(X_train, y_train, batch_size=32, epochs=300)
```



```
[31] # train the model in 32 batch size and 100 epochs
     model.fit(X_train, y_train, batch_size=32, epochs=300)
     Epoch 269/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1583 - mae: 0.1583
     Epoch 270/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1579 - mae: 0.1579
     Epoch 271/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1590 - mae: 0.1590
     Epoch 272/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1571 - mae: 0.1571
     Epoch 273/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1582 - mae: 0.1582
     Epoch 274/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1587 - mae: 0.1587
     Epoch 275/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1586 - mae: 0.1586
     Epoch 276/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1595 - mae: 0.1595
     Epoch 277/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1572 - mae: 0.1572
     Epoch 278/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1584 - mae: 0.1584
     Epoch 279/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1561 - mae: 0.1561
     Epoch 280/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1572 - mae: 0.1572
     Epoch 281/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1580 - mae: 0.1580
     Epoch 282/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1580 - mae: 0.1580
     Epoch 283/300
     516/516 [==============================] - 1s 1ms/step - loss: 0.1587 - mae: 0.1587
```

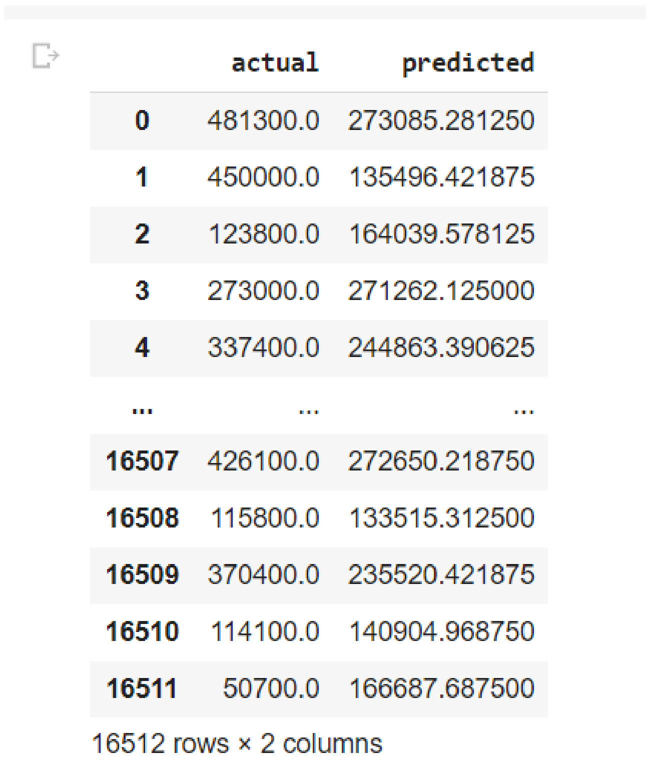
The data passed by the neuron leaves behind a weight or coefficient This weight remains on each neuron. We have 50 neurons in the layer and therefore 50 coefficients remain in the layer.

There is no guarantee that these coefficients will lead to the correct result. And therefore, the system or neural network goes from the first layer to the last two times: first to get the coefficients, and then in the reverse order so that the coefficients are corrected or unnecessary neurons are removed (which I assigned in our network). And the combination of these two actions is epocs. That is, we go here and there 300 times.

And all this in order to find an apartment !!!

```
# prediction on training dataset
train_prediction = model.predict(X_train)
train_result = pd.DataFrame(sc_y.inverse_transform(y_train))
train_result.columns = ['actual']
train_result['predicted'] = sc_y.inverse_transform(train_prediction)
train_result
```

The Output:

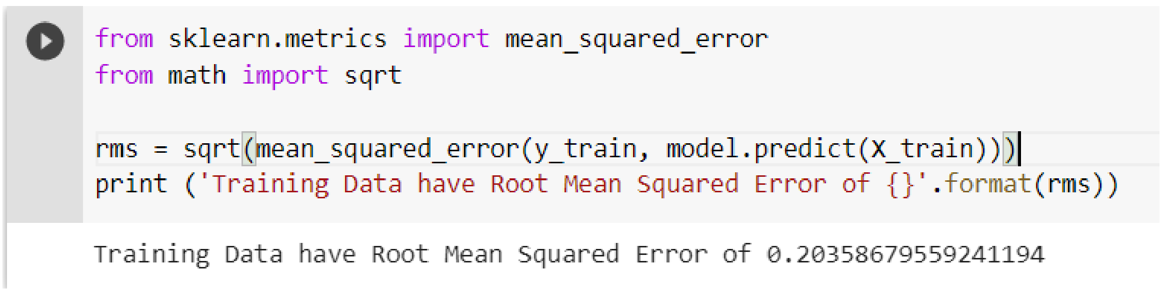


	actual	predicted
0	481300.0	273085.281250
1	450000.0	135496.421875
2	123800.0	164039.578125
3	273000.0	271262.125000
4	337400.0	244863.390625
...	...	...
16507	426100.0	272650.218750
16508	115800.0	133515.312500
16509	370400.0	235520.421875
16510	114100.0	140904.968750
16511	50700.0	166687.687500

16512 rows × 2 columns

Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general-purpose error metric for numerical predictions.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(y_train, model.predict(X_train)))
print ('Training Data have Root Mean Squared Error of {}'.format(rms))
```



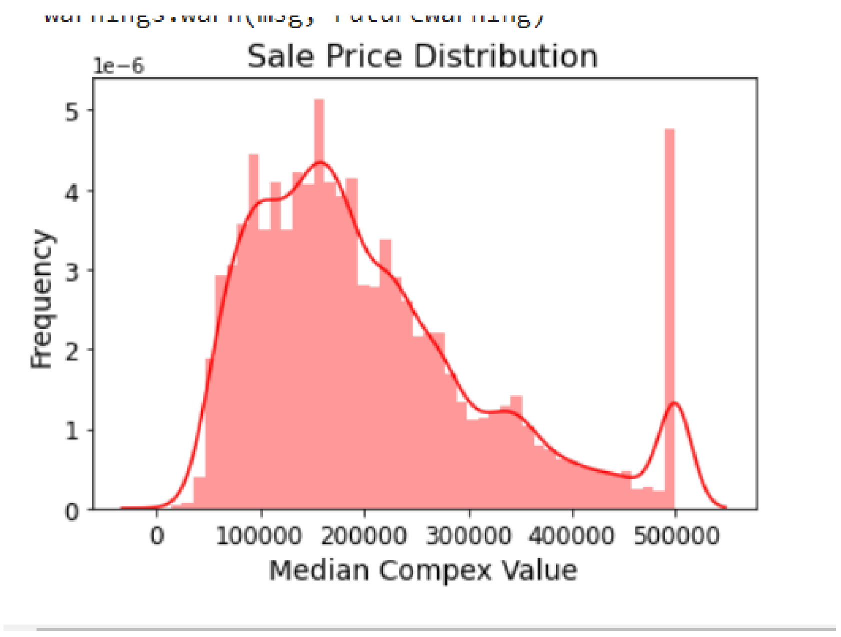
```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(y_train, model.predict(X_train)))
print ('Training Data have Root Mean Squared Error of {}'.format(rms))

Training Data have Root Mean Squared Error of 0.20358679559241194
```

Distribution plots are very useful to check how well a variable is distributed in the dataset. Let’s now produce a distribution plot using the ‘distplot’ function to check the distribution of the ‘medianCompexValue’ variable in the dataset.

```
#Distribution plot
import seaborn as sb # visualization
```

```

sb.distplot(df['medianCompexValue'], color = 'r')
plt.title('Sale␣Price␣Distribution', fontsize = 16)
plt.xlabel('Median␣Compex␣Value', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)

plt.show()
```

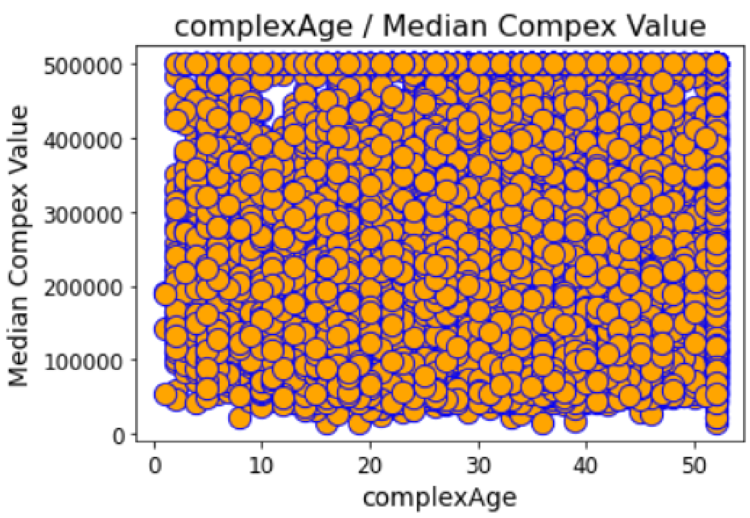
This graph shows that the data is not balanced.



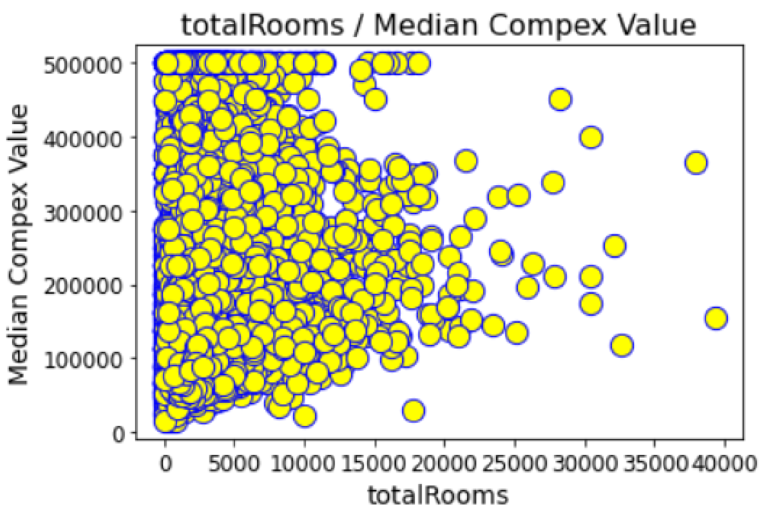
The following graphs show the ratio of each column to medianCompexValue.

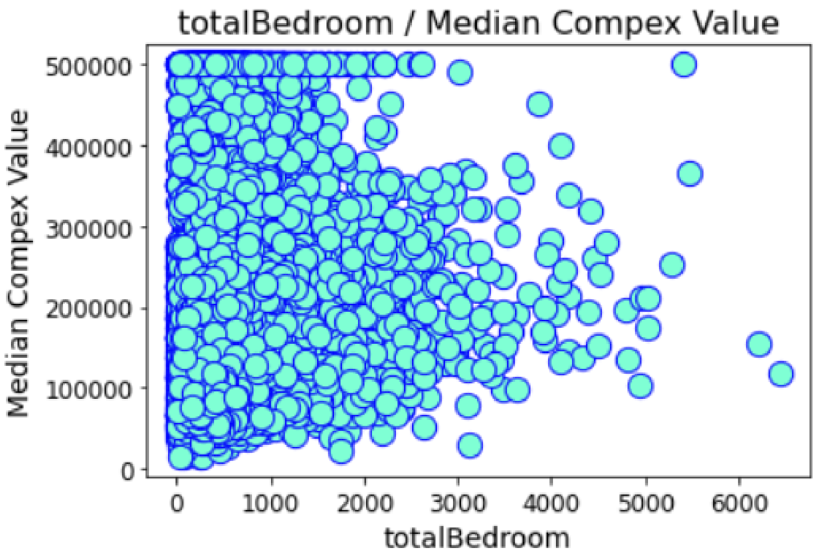
```python
# Scatter plot
import matplotlib.pyplot as plt # visualization
import seaborn as sb # visualization

def scatter_df(y_var):
    scatter_df = df.drop(y_var, axis = 1)
    i = df.columns

    plot1 = sb.scatterplot(i[0], y_var, data = df, color = 'orange', edgecolor =
    'b', s = 150)
    plt.title('{} / Median Compex Value'.format(i[0]), fontsize = 16)
    plt.xlabel('{}'.format(i[0]), fontsize = 14)
    plt.ylabel('Median Compex Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

    plot2 = sb.scatterplot(i[1], y_var, data = df, color = 'yellow', edgecolor =
    'b', s = 150)
    plt.title('{} / Median Compex Value'.format(i[1]), fontsize = 16)
    plt.xlabel('{}'.format(i[1]), fontsize = 14)
    plt.ylabel('Median Compex Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

    plot3 = sb.scatterplot(i[2], y_var, data = df, color = 'aquamarine',
   edgecolor = 'b', s = 150)
    plt.title('{} / Median Compex Value'.format(i[2]), fontsize = 16)
    plt.xlabel('{}'.format(i[2]), fontsize = 14)
    plt.ylabel('Median Compex Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

    plot4 = sb.scatterplot(i[3], y_var, data = df, color = 'deepskyblue',
   edgecolor = 'b', s = 150)
    plt.title('{} / Median Compex Value'.format(i[3]), fontsize = 16)
    plt.xlabel('{}'.format(i[3]), fontsize = 14)
    plt.ylabel('Median Compex Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

    plot5 = sb.scatterplot(i[4], y_var, data = df, color = 'crimson', edgecolor
   = 'white', s = 150)
    plt.title('{} / Median Compex Value'.format(i[4]), fontsize = 16)
    plt.xlabel('{}'.format(i[4]), fontsize = 14)
    plt.ylabel('Median␣Compex␣Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

    plot6 = sb.scatterplot(i[5], y_var, data = df, color = 'darkviolet', edgecolor = 'white', s = 150)
    plt.title('{}␣/␣Median␣Compex␣Value'.format(i[5]), fontsize = 16)
    plt.xlabel('{}'.format(i[5]), fontsize = 14)
    plt.ylabel('Median␣Compex␣Value', fontsize = 14)
    plt.xticks(fontsize = 12)
    plt.yticks(fontsize = 12)
    plt.show()

scatter_df('medianCompexValue')
```

```
  FutureWarning
```

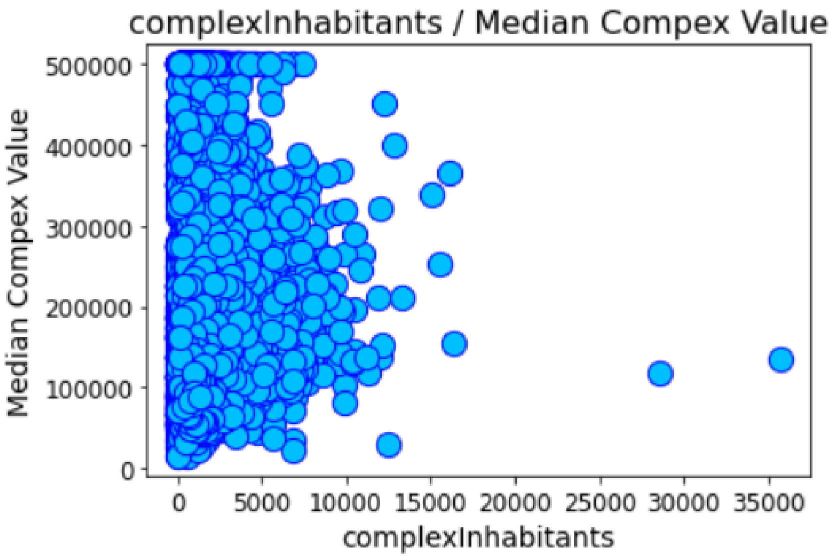


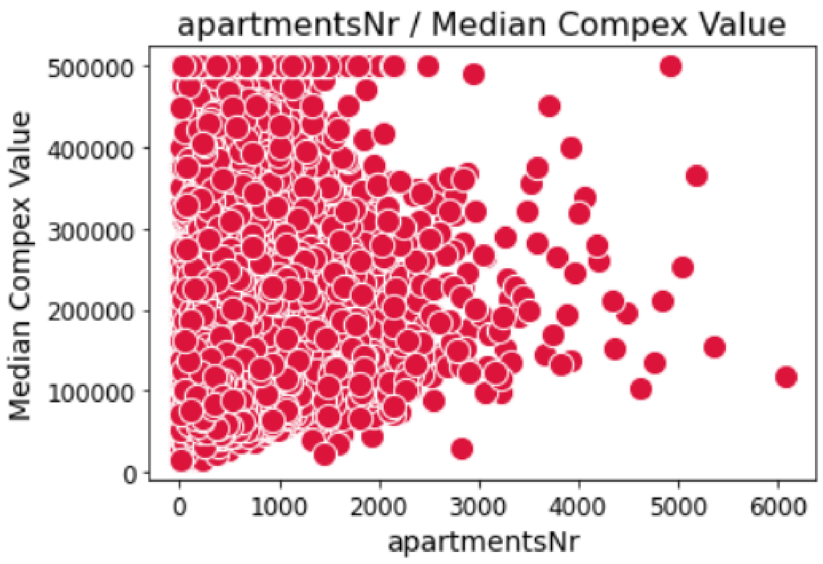
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:4
  FutureWarning
```





```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorator
  FutureWarning
```





Conclusions

We made a good mistake, but we didn't make the schedule. We believe that the data is not balanced and therefore the graph will be visually incorrect.

We are pleased with the latest schedules, as there is a sufficient number of apartments with bathrooms, as well as the age of the apartments. Can be selected in any segment.

References

[1] Source code for the laboratory work. Accessed February 25, 2021.

[2] Keras https://keras.io/api/.

[3] The Panda Libray https://malouche.github.io/teachingpython/panda_library.html

[4] https://faroit.com/keras-docs/2.0.2/getting-started/sequential-model-guide/

[5] https://keras.io/api/layers/activations/

[6] https://keras.io/api/optimizers/adam/

[7] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[8] https://www.sciencedirect.com/topics/engineering/root-mean-squared-error

[9] https://medium.com/codex/house-price-prediction-with-machine-learning-in-python-cf9

[10] https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-

[11] https://www.learndatasci.com/tutorials/predicting-housing-prices-linear-regression-u