Data Mining

Sayeh Sobhani

Student no. 99422102

Assignment 1.0

In this project, we are going to implement a linear regression model to predict the living space from the dataset we have on apartment rental offers in Germany.

This code is written with the Spyder editor 4.2.

This linear model implementation is going to be without using packages with ready to go linear regression models, but it is going to be compared to one.

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable causes the other (for example, higher SAT scores do not cause higher college grades), but that there is some significant association between the two variables.

A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not

indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model.

A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

Also a correlation matrix is implemented on the data for a better understanding of features correlation with the LivingSpace.

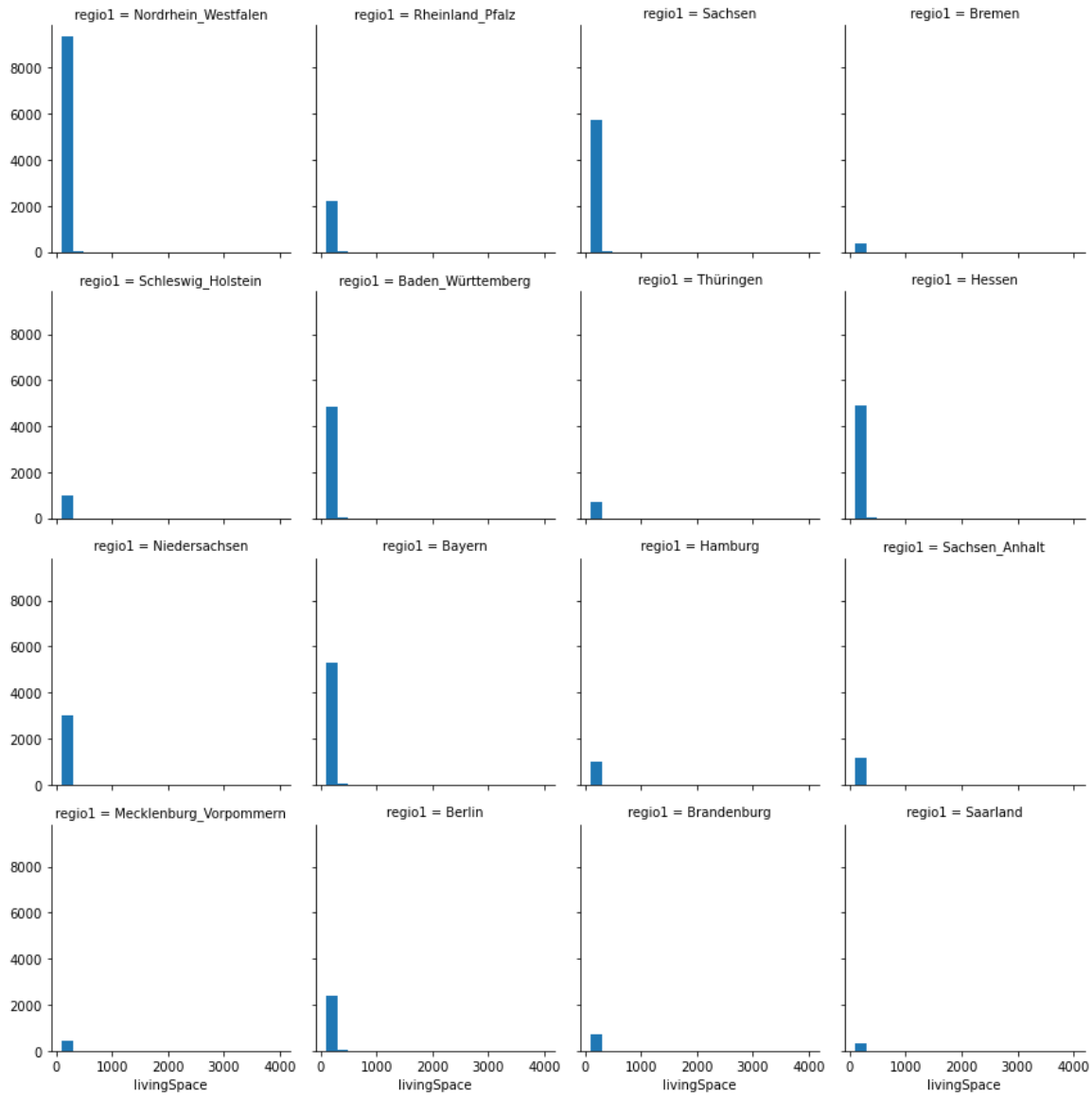
A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

Before we begin fitting out model to the data, some preprocessing is necessary.

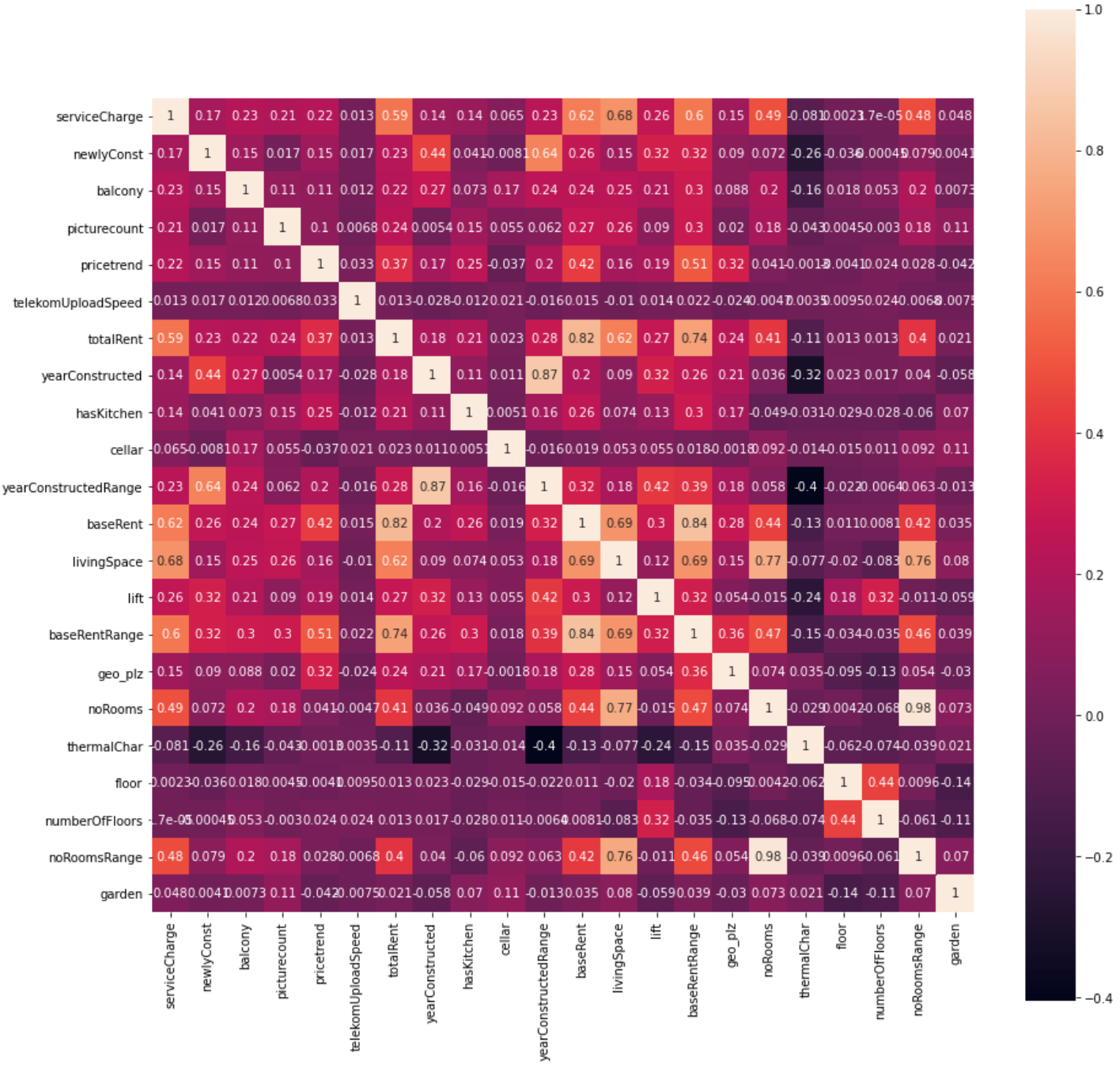
For this step:

First, I tried to clean and process the datasets given. For this purpose, I used the videos and codes that Mr. Sharifi has shared as a reference. For this part, I have removed some or replaces some of the null data, numeric and categorical. I have also removed and cleaned up the categorical features I believed couldn’t be of help.

To get a better understanding of how regio1 feature categories should be handled I used this plot for visualization and merged some of the categories into a single category called others:



From the correlation matrix, some of the features mostly correlated with “Living Space” are number of rooms and the range of number of rooms, base rent range, total range and service charge, which all have correlations over 0.60. Whereas the number of floors has a negative correlation with living space with a score of -0.08.



1) In this step, we should introduce five hypothesizes and investigate the them with useful hypothesis test techniques.

   Some of the well-known hypothesis test methods are:

   Pearson's Correlation Coefficient

Spearman’s Rank Correlation

Kendall’s Rank Correlation

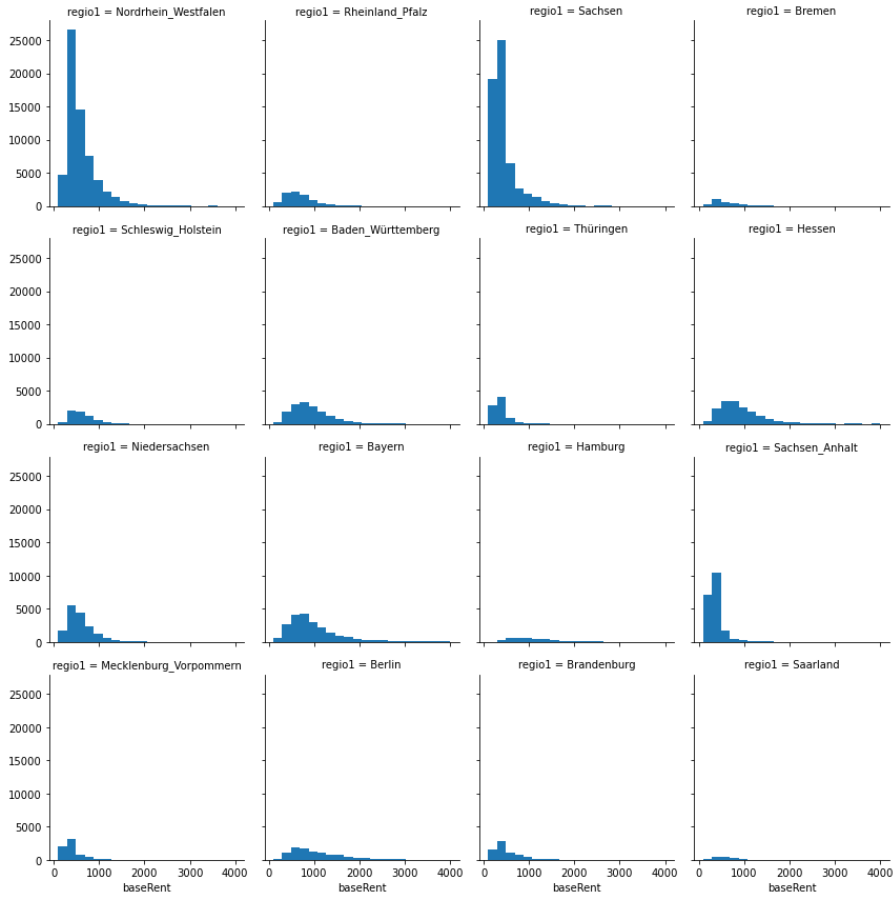
Chi-Squared Test

a. First, let’s investigate the relation of living space and the number of rooms.

The result for this hypothesis is SpearmanrResult (correlation=0.7815992067597107, pvalue=0.0)

b. To investigate the relation between regio1 and base rent, Hypothesis test Anova was run with the results “Oneway Anova totalRent ~ edit region1 F=5430.63, p-value=0.000000E+00”

   To better understand this feature, let’s see this grid face plot :



c. To investigate the relation of living space and no. of floors

   The results show negative relation: SpearmanrResult (correlation=-0.10258377605772608, pvalue=0.0)

d. The relation of having kitchen with living space was tested with the results: PointbiserialrResult (correlation=0.07421226391198485, pvalue=4.18269128783e-312)

e. Relation of having a kitchen and price of rent was tested and the results are: PointbiserialrResult (correlation=0.21400248713621559, pvalue=0.0)

Some of the hypothesizes we investigated had relations and were correct.

2) In this step, I implement a linear regression model without using scikitlearn or statsmodels.

   Before starting this step, replacing categorical data with dummy variables is necessary. I also split the target from features and divided the dataset into an 80 percent part train and a 20 percent part test after shuffling datapoints with this command

   df = shuffle(df)

   Then the regression model implementation was done with no packages other than NumPy. This part of the code can be seen in the #%%linear regr no package cell.

   The difference of some predicted values vs the real test data values can be seen in the output below:

Out[207]:

	test	pred
0	73.61	84.908897
1	59.00	59.868465
2	156.00	101.394185
3	60.88	65.675451
4	117.13	124.502588

4)For this part, the linear regression model is implemented with scikitlearn package and the MSE for this model for all features is

The MSE for this model with all features was 247.4 with a learning rate of 0.001.

The most correlated feature with living space is number of rooms with a correlation score of 0.77, I only run a linear regr with this feature in use and the MSE for this model was 459.0.

The most negatively correlated feature with living space is number of floors with a correlation score of -0.08, I only run a linear regr with this feature in use and the MSE for this model was 1032.3.

As implying PCA to reduce features, this is what we got:

with a pov of 0.99 we got these results for the test and train sets:

Train:(206602, 36)

Test:(51651, 36)

with a pov of 0.95 we got these results for the test and train sets:

Train:(206602, 24)

Test:(51651, 24)

with a pov of 0.90 we got these results for the test and train sets:

Train:(206602, 17)

Test:(51651, 17)

with a pov of 0.70 we got these results for the test and train sets:

Train:(206602, 8)

Test:(51651, 8)

For a margin of 0.1, we test our model's accuracy and the outcome is 246.6 for MSE and 45.1 accuracy which is not a very good score.

References:

1. Codes that the Teacher Assistants of this course have shared and Mr. Sharifi's videos.
2. sharpsightlabs.com
3. machinelearningmastery.com
4. medium.com, towardsdatascience.com
5. scikit-learn.org
6. kaggle.com

Appendix

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  @author: Sayeh Sobhani
4.  written in Spyder
5.  """
6.  #%%Importing packages and dataset
7.  import numpy as np
8.  import pandas as pd
9.  import seaborn as sns
10. import matplotlib.pyplot as plt
11. from scipy import stats
12. from sklearn.utils import shuffle
13. from sklearn.decomposition import PCA
14. from sklearn.linear_model import LinearRegression
15. from sklearn.metrics import mean_squared_error
16.
17. dfr = pd.read_csv(r'C:\Users\Asus\Desktop\Data Mining\01_assignment\1.0\immo_data.csv')
18.
19.
20. #%%Understanding and cleaning dataset
21.
22. #to get a better understanding of how our data looks we look at the first 10 rows
23. dfr.shape
24. dfr.head(10)
25.
26. #lets see some info about this dataset and its features
27. dfr.info()
28.
29. dfr.describe()
30.
31. #the mean for each numeric feature?
32. #miangin = dfr._get_numeric_data().mean()
33. #print(miangin)
34.
35. #useless columns deletion '',
36. dfr = dfr.drop(columns=['description'])
37. dfr = dfr.drop(columns=['facilities'])
38. dfr = dfr.drop(columns=['scoutId'])
39. dfr = dfr.drop(columns=['geo_bln','date'])
40. dfr = dfr.drop(columns=['houseNumber','livingSpaceRange','firingTypes'])
41. dfr = dfr.drop(columns=['streetPlain','street','geo_krs','regio2','regio3'])
42. dfr = dfr.drop(dfr[dfr['livingSpace'] == 0.0].index)
43. dfr = dfr.drop(dfr[dfr['totalRent'] == 0.0].index)
44.
45. #how null data looks like
46. dfr.isna()
47.
48. dfr.columns
49.
50. #columns with null data
51. dfr.isna().sum()/len(dfr)
52.
53. #which columns hve >50 percent null
54. dfr.columns[((dfr.isna().sum()/len(dfr)) > 0.50)]
55.
56. #removing those columns with >50 null
57. dfr = dfr.drop(columns=dfr.columns[((dfr.isna().sum()/len(dfr)) > 0.50)])
```

```
dfr.columns
dfr.shape

#filling the remaining data with means and max seen data
    #numeric
dfr._get_numeric_data().mean()
dfr.fillna(dfr._get_numeric_data().mean(),inplace = True)
dfr.isna().sum()
    #Categorical most seens
for cols in dfr.columns:
    if dfr[cols].dtype == 'object' or dfr[cols].dtype == 'bool':
        print('column : ',cols)
        print(dfr[cols].value_counts().head())

     #Fill categorical nan with most seen data in each feature
for cols in dfr.columns:
    if dfr[cols].dtype == 'object' or dfr[cols].dtype == 'bool':
        print('cols : {} , value : {}'.format(cols ,
   dfr[cols].value_counts().head(1).index[0]))
        dfr[cols].fillna(dfr[cols].value_counts().head(1).index[0],inplace = True)
#to check if nan left
dfr.isna().sum()
dfr.shape
dfr.info()

"Some Visualization"
#regio1 categorical by base rent amounts PLOT visualization
#dfr['regio1'].value_counts()
#g = sns.FacetGrid(dfr, col='regio1', col_wrap=4)
#g = g.map(plt.hist, 'livingSpace', bins=20, range=(100,4000))

#outliers finding and removing
for cols in dfr.columns:
    if dfr[cols].dtype == 'int64' or dfr[cols].dtype == 'float64':
        upper_range = dfr[cols].mean() + 3 * dfr[cols].std()
        lower_range = dfr[cols].mean() - 3 * dfr[cols].std()

        indexs = dfr[(dfr[cols] > upper_range) | (dfr[cols] < lower_range)].index
        dfr = dfr.drop(indexs)

dfr.info()
dfr.shape

 #%%categorical feature inspection
 #the number of categories in each feature
 for cols in dfr.columns:
     if dfr[cols].dtype == 'object' or dfr[cols].dtype == 'bool':
         print('cols : {} , unique values : {}'.format(cols,dfr[cols].nunique()))


 #the categories and their count in each feature
 for cols in dfr.columns:
     if dfr[cols].dtype == 'object' or dfr[cols].dtype == 'bool':
         print('cols : {} ,\n {}'.format(cols,dfr[cols].value_counts()))

 #regio1 edition based on plots from base rent amounts in line 75
 dfr['regio1'].value_counts()
 def edit_regio1(x):
     if x in ['Hamburg','Bremen','Saarland']:
         return 'other'
     else:
         return x
 dfr['regio1_'] = dfr['regio1'].apply(edit_regio1)
 dfr = dfr.drop(columns = ['regio1'])
 dfr['regio1_'].value_counts()*100 / len(dfr)
```

```
122.
123.     #visualization
124.  plt_regio1_= dfr['regio1_'].value_counts().plot(kind='bar')
125.
126.  #heatingType
127.  dfr['heatingType'].value_counts()
128.  others = list(dfr['heatingType'].value_counts().tail(12).index)
129.  def edit_heatingType(x):
130.      if x in others:
131.          return 'other'
132.      else:
133.          return x
134.
135.  dfr['heatingType_'] = dfr['heatingType'].apply(edit_heatingType)
136.  dfr = dfr.drop(columns = ['heatingType'])
137.  dfr['heatingType_'].value_counts()*100 / len(dfr)
138.
139.  #telekomTvOffer
140.  dfr['telekomTvOffer'].value_counts()
141.  others = list(dfr['telekomTvOffer'].value_counts().tail(2).index)
142.  def edit_telekomTvOffer(x):
143.      if x in others:
144.          return 'other'
145.      else:
146.          return x
147.
148.  dfr['telekomTvOffer_'] = dfr['telekomTvOffer'].apply(edit_telekomTvOffer)
149.  dfr = dfr.drop(columns = ['telekomTvOffer'])
150.  dfr['telekomTvOffer_'].value_counts()*100 / len(dfr)
151.
152.  #typeOfFlat
153.  dfr['typeOfFlat'].value_counts()
154.  others = list(dfr['typeOfFlat'].value_counts().tail(6).index)
155.  def edit_typeOfFlat(x):
156.      if x in others:
157.          return 'other'
158.      else:
159.          return x
160.
161.  dfr['typeOfFlat_'] = dfr['typeOfFlat'].apply(edit_typeOfFlat)
162.  dfr = dfr.drop(columns = ['typeOfFlat'])
163.  dfr['typeOfFlat_'].value_counts()*100 / len(dfr)
164.
165.  #condition
166.  dfr['condition'].value_counts()
167.  others = list(dfr['condition'].value_counts().tail(4).index)
168.  def edit_condition(x):
169.      if x in others:
170.          return 'other'
171.      else:
172.          return x
173.
174.  dfr['condition_'] = dfr['condition'].apply(edit_condition)
175.  dfr = dfr.drop(columns = ['condition'])
176.  dfr['condition_'].value_counts()*100 / len(dfr)
177.
178.  #%%normalization of numeric data
179.  for cols in dfr.columns:
180.      if dfr[cols].dtype == 'int64' or dfr[cols].dtype == 'float64':
181.           if cols != 'livingSpace':
182.                dfr[cols] = ((dfr[cols] - dfr[cols].mean())/(dfr[cols].std()))
183.
184.
185.  #%%correlation matrix
186.  corr = dfr.corr()
```

```
corr
f, ax = plt.subplots(figsize=(15, 15))

sns.heatmap(corr, square = True ,annot = True)

#%%Hypotheisi tests
dfr["regio1_"]
    #relation of noRoomsRange&livingSpace
stats.stats.spearmanr(dfr['livingSpace'],dfr['noRoomsRange'])

    #relation regio1 and rent
fstat, pval = stats.f_oneway(*[dfr.baseRent[dfr.regio1_ == s]
for s in dfr.regio1_.unique()])
print("Oneway Anova totalRent ~ edit region1 F=%.2f, p-value=%E" % (fstat, pval))

    #living space relation with no of floors
stats.stats.spearmanr(dfr['livingSpace'],dfr['numberOfFloors'])

    #relation of having kitchen vs living space
stats.pointbiserialr(dfr['hasKitchen'],dfr["livingSpace"])

    #relation btween owning a kitchen and paying rent in total
stats.pointbiserialr(dfr['totalRent'],dfr["hasKitchen"])

#%%Dummy variables for categorical
columns = []
for cols in dfr.columns:
    if dfr[cols].dtype == 'object' or dfr[cols].dtype == 'bool':
        columns.append(cols)
columns

dummies_feature = pd.get_dummies(dfr[columns])
dummies_feature.head()
dummies_feature.shape
dfr = pd.concat([dfr, dummies_feature], axis=1)
dfr.head()
dfr = dfr.drop(columns=columns)
dfr.head()
dfr.info()

#%%shuffling data and dividing test and train
dfr = shuffle(dfr)
y = dfr['livingSpace'].values
x = dfr.drop(columns = ['livingSpace']).values

print(x.shape)
print(y.shape)

#splitting test and train
train_size = int(0.8 * x.shape[0])
train_size

x_train = x[:train_size]
y_train = y[:train_size]

print(x_train.shape)
print(y_train.shape)

x_test = x[train_size:]
y_test = y[train_size:]

print(x_test.shape)
print(y_test.shape)

#%%linear regr no package
```

```
252.
253.  #y^=w1*x1+w2*x2+···+w51*x51+bias is the line we're lookin for
254.  #setting parameters "w" and "b" randomly
255.  np.random.seed(42)
256.  b = np.random.randn(1)
257.  w = np.random.randn(51)
258.  n = x_train.shape[0]
259.
260.  #Sets learning rate
261.  lr = 0.001
262.
263.  #number of epochs
264.  n_epochs = 500
265.
266.  test_error =  y_test - np.sum((w * x_test) + b , axis = 1)
267.  test_mse = (test_error**2).mean()
268.  print('init MSE : ',test_mse)
269.
270.  for epoch in range(n_epochs):
271.      error = y_train - np.sum((w * x_train) + b , axis = 1)
272.
273.      if epoch % 100 == 0:
274.          print('epoch {} , MSE : {}'.format(epoch,(error**2).mean()))
275.
276.      w_grad = [0] * x_train.shape[1]
277.      b_grad = 0
278.
279.      for i in range(x_train.shape[1]):
280.          w_grad[i] = -1 * (x_train[:,i] * error).mean()
281.          w[i] = w[i] - (lr * w_grad[i])
282.
283.      b_grad = -1 * error.mean()
284.      b = b - (lr*b_grad)
285.
286.
287.  test_error = y_test - np.sum((w * x_test) + b , axis = 1)
288.  test_mse = (test_error**2).mean()
289.  print('Final MSE : ',test_mse)
290.
291.  #comparison of the predicted value and the value in dataset
292.  y_pred = np.sum((w * x_test) + b , axis = 1)
293.  temp = pd.DataFrame({'test':y_test,'pred':y_pred})
294.  temp.head()
295.
296.  temp['upper_range'] = temp['test'] * 1.2
297.  temp['lower_range'] = temp['test'] * 0.8
298.
299.  temp[(temp['upper_range'] >=temp['pred']) & (temp['pred'] >=
    temp['lower_range'])].shape[0] * 100/temp.shape[0]
300.
301.  #%%only noRooms feature linear regr no package
302.
303.  X = dfr['noRooms'].values
304.  X.shape
305.  X_train = X[:train_size]
306.  print(X_train.shape)
307.  X_test = X[train_size:]
308.  print(X_test.shape)
309.
310.  np.random.seed(42)
311.  b = np.random.randn(1)
312.  w = np.random.randn(1)
313.  n = X_train.shape[0]
314.
315.  # Sets learning rate
```

```
lr = 0.1


n_epochs = 500
for epoch in range(n_epochs):
    error = y_train - ((w * X_train) + b)

    if epoch % 100 == 0:
        print('epoch {} , MSE : {}'.format(epoch,(error**2).mean()))

    # adoptive learning rate
    if epoch % 200 == 0:
        lr = lr * 0.1


    w_grad = 0
    b_grad = 0

    w_grad = -1 * (X_train * error).mean()
    w = w - (lr * w_grad)

    b_grad = -1 * error.mean()
    b = b - (lr*b_grad)

test_error = y_test - ((w * X_test) + b)
test_mse = (test_error**2).mean()
print('Final MSE : ',test_mse)
print('Final learning rate : ',lr)



#%%Linear regr with scikitlearn package all features
Linear = LinearRegression()
Linear.fit(x_train,y_train)
print("the coeficients are:")
print(Linear.coef_)
print("the intercept is:")
print(Linear.intercept_)

y_pred = Linear.predict(x_test)
mean_squared_error(y_pred,y_test)

temp = pd.DataFrame({'test':y_test,'pred':y_pred})
temp.head()

temp['upper_range'] = temp['test'] * 1.2
temp['lower_range'] = temp['test'] * 0.8

temp[(temp['upper_range'] >=temp['pred']) & (temp['pred'] >=
   temp['lower_range'])].shape[0] * 100/temp.shape[0]

#%%Linear regr package floor feature

X = dfr['numberOfFloors'].values
X.shape
X_train = X[:train_size]
print(X_train.shape)
X_test = X[train_size:]
print(X_test.shape)

Linear = LinearRegression()
Linear.fit(x_train,y_train)
print("the coeficients are:")
print(Linear.coef_)
print("the intercept is:")
```

```
print(Linear.intercept_)

y_pred = Linear.predict(x_test)
mean_squared_error(y_pred,y_test)

temp = pd.DataFrame({'test':y_test,'pred':y_pred})
temp.head()

temp['upper_range'] = temp['test'] * 1.2
temp['lower_range'] = temp['test'] * 0.8

temp[(temp['upper_range'] >=temp['pred']) & (temp['pred'] >=
   temp['lower_range'])].shape[0] * 100/temp.shape[0]

#%%number of floors feature linear regr no package

X = dfr['numberOfFloors'].values
X.shape
X_train = X[:train_size]
print(X_train.shape)
X_test = X[train_size:]
print(X_test.shape)

np.random.seed(42)
b = np.random.randn(1)
w = np.random.randn(1)
n = X_train.shape[0]

# Sets learning rate
lr = 0.1


n_epochs = 500
for epoch in range(n_epochs):
    error = y_train - ((w * X_train) + b)

    if epoch % 100 == 0:
        print('epoch {} , MSE : {}'.format(epoch,(error**2).mean()))

    # adoptive learning rate
    if epoch % 200 == 0:
        lr = lr * 0.1


    w_grad = 0
    b_grad = 0

    w_grad = -1 * (X_train * error).mean()
    w = w - (lr * w_grad)

    b_grad = -1 * error.mean()
    b = b - (lr*b_grad)

test_error = y_test - ((w * X_test) + b)
test_mse = (test_error**2).mean()
print('Final MSE : ',test_mse)
print('Final learning rate : ',lr)

#%%PCA

#pov 0.99
pca = PCA(0.99)
x_pca = pca.fit_transform(x)
x_pca.shape
x_pca_train = x_pca[:train_size]
```

```
print(x_pca_train.shape)
x_pca_test = x_pca[train_size:]
print(x_pca_test.shape)

#pov 0.95
pca = PCA(0.95)
x_pca = pca.fit_transform(x)
x_pca.shape
x_pca_train = x_pca[:train_size]
print(x_pca_train.shape)
x_pca_test = x_pca[train_size:]
print(x_pca_test.shape)

#pov 0.90
pca = PCA(0.90)
x_pca = pca.fit_transform(x)
x_pca.shape
x_pca_train = x_pca[:train_size]
print(x_pca_train.shape)
x_pca_test = x_pca[train_size:]
print(x_pca_test.shape)

#pov 0.70
pca = PCA(0.70)
x_pca = pca.fit_transform(x)
x_pca.shape
x_pca_train = x_pca[:train_size]
print(x_pca_train.shape)
x_pca_test = x_pca[train_size:]
print(x_pca_test.shape)


#%% linear regr no package different margin
#margin or 0.1
np.random.seed(42)
b = np.random.randn(1)
w = np.random.randn(51)
n = x_train.shape[0]

#Sets learning rate
lr = 0.001

#number of epochs
n_epochs = 500

test_error =  y_test - np.sum((w * x_test) + b , axis = 1)
test_mse = (test_error**2).mean()
print('init MSE : ',test_mse)

for epoch in range(n_epochs):
    error = y_train - np.sum((w * x_train) + b , axis = 1)

    if epoch % 100 == 0:
        print('epoch {} , MSE : {}'.format(epoch,(error**2).mean()))

    w_grad = [0] * x_train.shape[1]
    b_grad = 0

    for i in range(x_train.shape[1]):
        w_grad[i] = -1 * (x_train[:,i] * error).mean()
        w[i] = w[i] - (lr * w_grad[i])

    b_grad = -1 * error.mean()
    b = b - (lr*b_grad)

```

```
509.
510.  test_error = y_test - np.sum((w * x_test) + b , axis = 1)
511.  test_mse = (test_error**2).mean()
512.  print('Final MSE : ',test_mse)
513.
514.  #comparison of the predicted value and the value in dataset
515.  y_pred = np.sum((w * x_test) + b , axis = 1)
516.  temp = pd.DataFrame({'test':y_test,'pred':y_pred})
517.  temp.head()
518.
519.  temp['upper_range'] = temp['test'] * 1.1
520.  temp['lower_range'] = temp['test'] * 0.9
521.
522.  temp[(temp['upper_range'] >=temp['pred']) & (temp['pred'] >=
    temp['lower_range'])].shape[0] * 100/temp.shape[0]
523.
```