



---

# Introduction to Data Mining on the Cloud

Department of Computer Science  
Kristianstad University  
Course: DT584C  
Master's in computer science

Student: Hazrat Ali <[react.dev.se@gmail.com](mailto:react.dev.se@gmail.com)>  
Teacher: Dawit Mengistu

---

---

## Table of contents

[Table of contents](#)

[Objective](#)

[Procedure](#)

[Report](#)

[PART 1](#)

[Google Cloud](#)

[Virtual Machines](#)

[Datalab](#)

[PART 2](#)

[Frequent items selection](#)

[PART 3](#)

[Regression](#)

[Solution](#)

[Exploratory Analysis](#)

[RESULT:](#)

---

## Objective

In this lab, you shall

1. Learn how to use Cloud Computing for your Data Mining tasks
2. Familiarize yourself with the Google Cloud Platform (GCP)
3. Experiment with Frequent Item Mining (FIM)

In order to perform the experiments in this lab, you need a reliable Internet connection.

There will be a tutorial session to help you get started.

## Procedure

### PART 1:

Once enrolled in the Cloud environment, read tutorials available online to understand how machine learning and data mining tasks can be executed on the Cloud.

### PART 2:

Create a Data Mining task of your own to analyze a dataset provided to you. In this task, you shall learn about **Frequent Item Mining** (FIM) algorithms. The general theory of the algorithms was discussed during the lectures. You shall use available open source libraries or write Python codes of your own to analyze the data. If you use the available libraries, you may have to customize the code to meet your requirements.

### PART 3:

Produce a **regression model** for the given dataset. You may use Octave, Matlab or another tool (Python based). Compare and evaluate different regression models. Which type of model is recommended? Why?

---

## Report

1. You shall write a report on what you learned in using GCP for machine learning applications (**part1**).
2. Write a report on how you solved the task in **Part2** and **Part3**. Upload your solution code together with your report (in separate files).

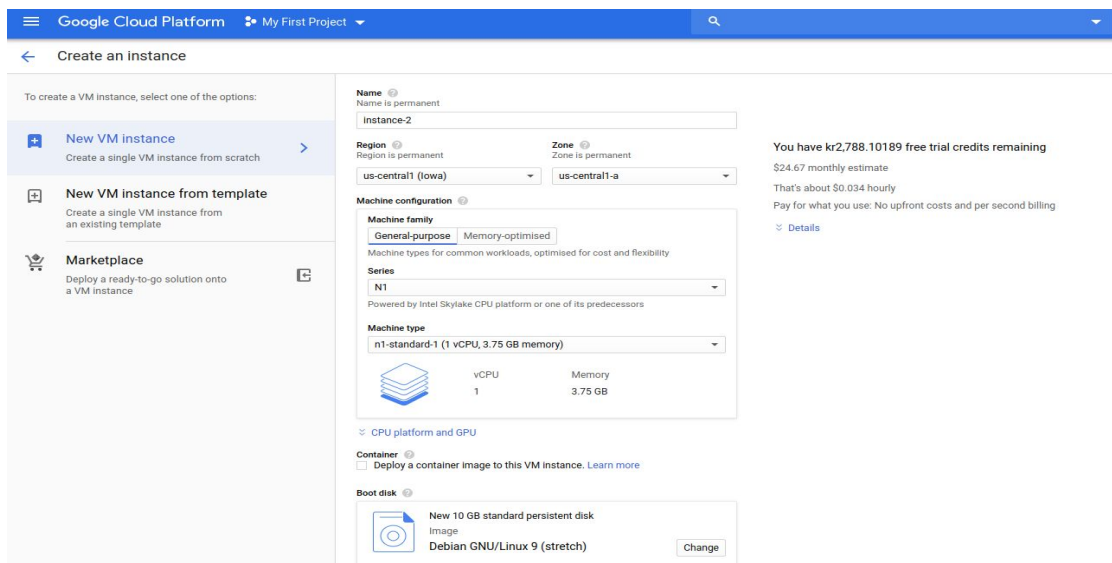
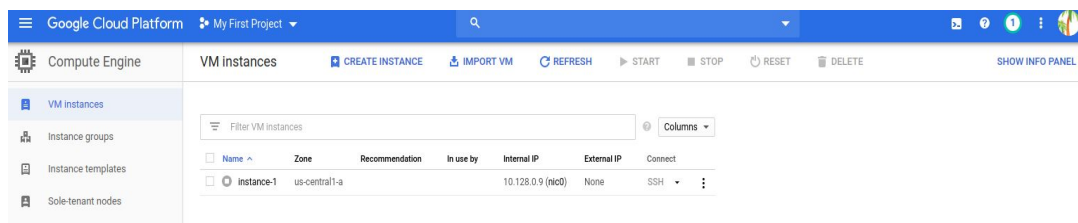
# 1. PART 1

## Google Cloud

Google cloud provides services for data mining, and we used two of them ie Data lab and Virtual private server. It has a Jupyter like interface. Google cloud provides a trial free services for learning purposes, which we use for this lab.

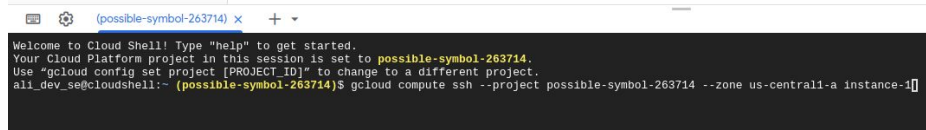
## Virtual Machines

Service: Compute > Compute engine



---

To connect to virtual machine use the following command:



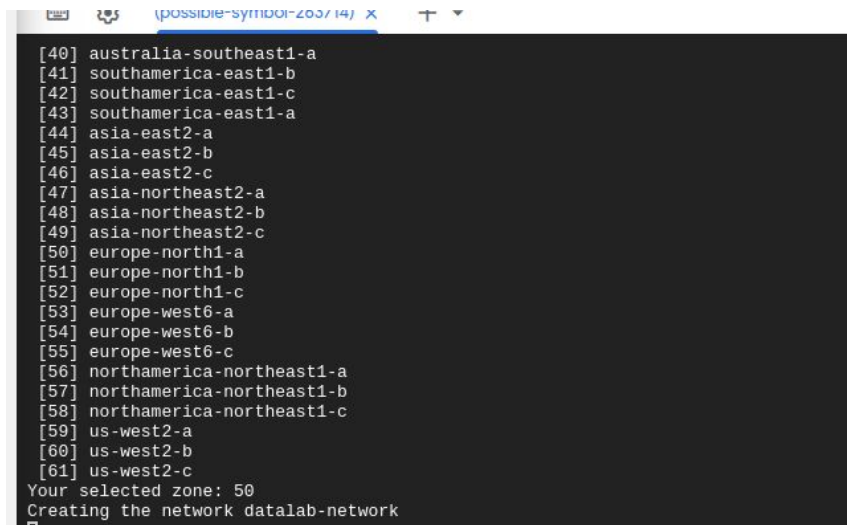
```
(possible-symbol-263714) x + v
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to possible-symbol-263714.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
ali_dev_se@cloudshell:~ (possible-symbol-263714)$ gcloud compute ssh --project possible-symbol-263714 --zone us-central1-a instance-1
```

## Datalab

Create an instance by using datalab command.

```
datalab create fis
```

The command will ask for region. Choose the one that suits you the best.

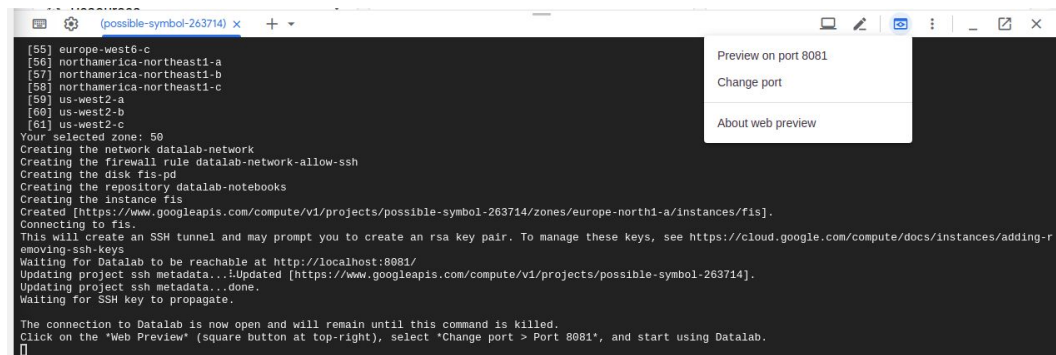


```
(possible-symbol-263714) x + v
[40] australia-southeast1-a
[41] southamerica-east1-b
[42] southamerica-east1-c
[43] southamerica-east1-a
[44] asia-east2-a
[45] asia-east2-b
[46] asia-east2-c
[47] asia-northeast2-a
[48] asia-northeast2-b
[49] asia-northeast2-c
[50] europe-north1-a
[51] europe-north1-b
[52] europe-north1-c
[53] europe-west6-a
[54] europe-west6-b
[55] europe-west6-c
[56] northamerica-northeast1-a
[57] northamerica-northeast1-b
[58] northamerica-northeast1-c
[59] us-west2-a
[60] us-west2-b
[61] us-west2-c
Your selected zone: 50
Creating the network datalab-network
```

It will take some time to create the environment.

```
[52] europe-north1-c
[53] europe-west6-a
[54] europe-west6-b
[55] europe-west6-c
[56] northamerica-northeast1-a
[57] northamerica-northeast1-b
[58] northamerica-northeast1-c
[59] us-west2-a
[60] us-west2-b
[61] us-west2-c
Your selected zone: 50
Creating the network datalab-network
Creating the firewall rule datalab-network-allow-ssh
Creating the disk fis-pd
Creating the repository datalab-notebooks
Creating the instance fis
Created [https://www.googleapis.com/compute/v1/projects/possible-symbol-263714/zones/europe-north1-a/instances/fis].
Connecting to fis.
This will create an SSH tunnel and may prompt you to create an rsa key pair. To manage these keys, see https://cloud.google.com/compute.
Removing-ssh-keys
Waiting for Datalab to be reachable at http://localhost:8081/
Updating project ssh metadata...Updated [https://www.googleapis.com/compute/v1/projects/possible-symbol-263714].
Updating project ssh metadata...done.
Waiting for SSH key to propagate.
```

Once ready it will guide you to start the service.



If for any reason you disconnect the datalab image, you can reconnect using:

`datalab connect fis`

Now data lab is ready to work with, create a notebook and happy coding.



---

## 2. PART 2

### Frequent items selection

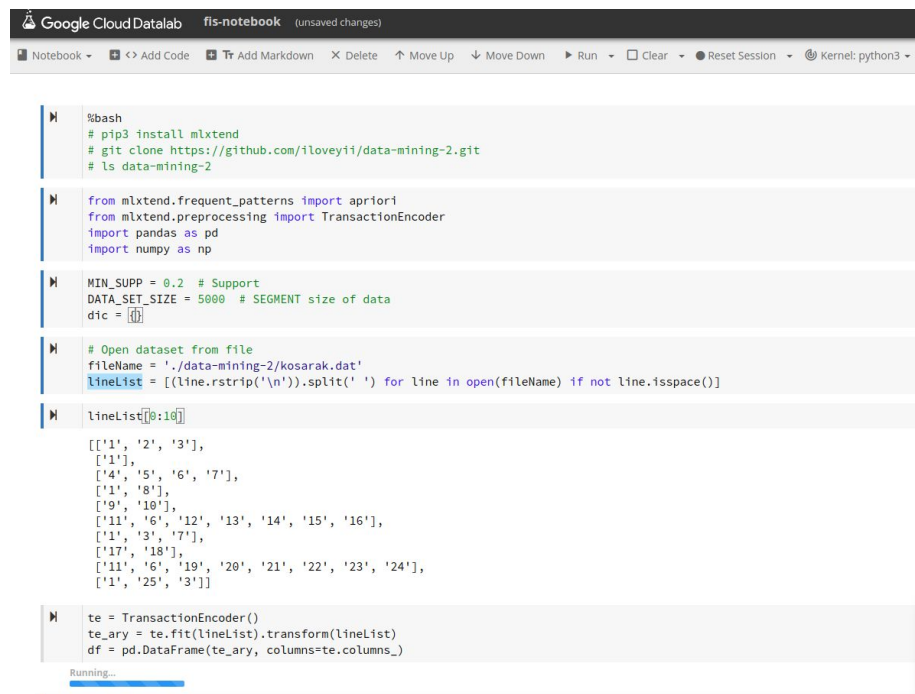
In this task we do basket analysis ( frequent item selection) using Apriori algorithm in Python.

I installed Python 3.6 on Ubuntu for this task. At first I also tried examples and sample code on Jupyter. However I did this task by coding using PyCharm IDE.

I tried to solve the problem on my computer but it required a lot of memory than my computer can support. Then I learned by practicing with smaller datasets.

After completing the code I used google cloud VPS with 560GB RAM and 16 CPUs but I got the error 'index out of range'.

Then I tried google cloud datalab learned in the previous section, but it had not mlxtend library available(which I later could fix it), and ran it successfully.



```
Google Cloud Datalab  fis-notebook  (unsaved changes)
Notebook  Add Code  Add Markdown  Delete  Move Up  Move Down  Run  Clear  Reset Session  Kernel: python3

%bash
# pip3 install mlxtend
# git clone https://github.com/iloveyfi/data-mining-2.git
# ls data-mining-2

from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd
import numpy as np

MIN_SUPP = 0.2 # Support
DATA_SET_SIZE = 5000 # SEGMENT size of data
dic = {}

# Open dataset from file
fileName = './data-mining-2/kosarak.dat'
lineList = [(line.rstrip('\n')).split(' ') for line in open(fileName) if not line.isspace()]

lineList[0:10]

[['1', '2', '3'],
 ['1'],
 ['4', '5', '6', '7'],
 ['1', '8'],
 ['9', '10'],
 ['11', '6', '12', '13', '14', '15', '16'],
 ['1', '3', '7'],
 ['17', '18'],
 ['11', '6', '19', '20', '21', '22', '23', '24'],
 ['1', '25', '3']]

te = TransactionEncoder()
te_ary = te.fit(lineList).transform(lineList)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

But it took too long to run the script.



---

To solve the above I refactored my algorithm and I partitioned the bigger dataset into smaller partitions and then compute them individually. At the end I averaged all the results to get the final results.

To make it quicker I also need to tailor MIN\_SUPPORT value.

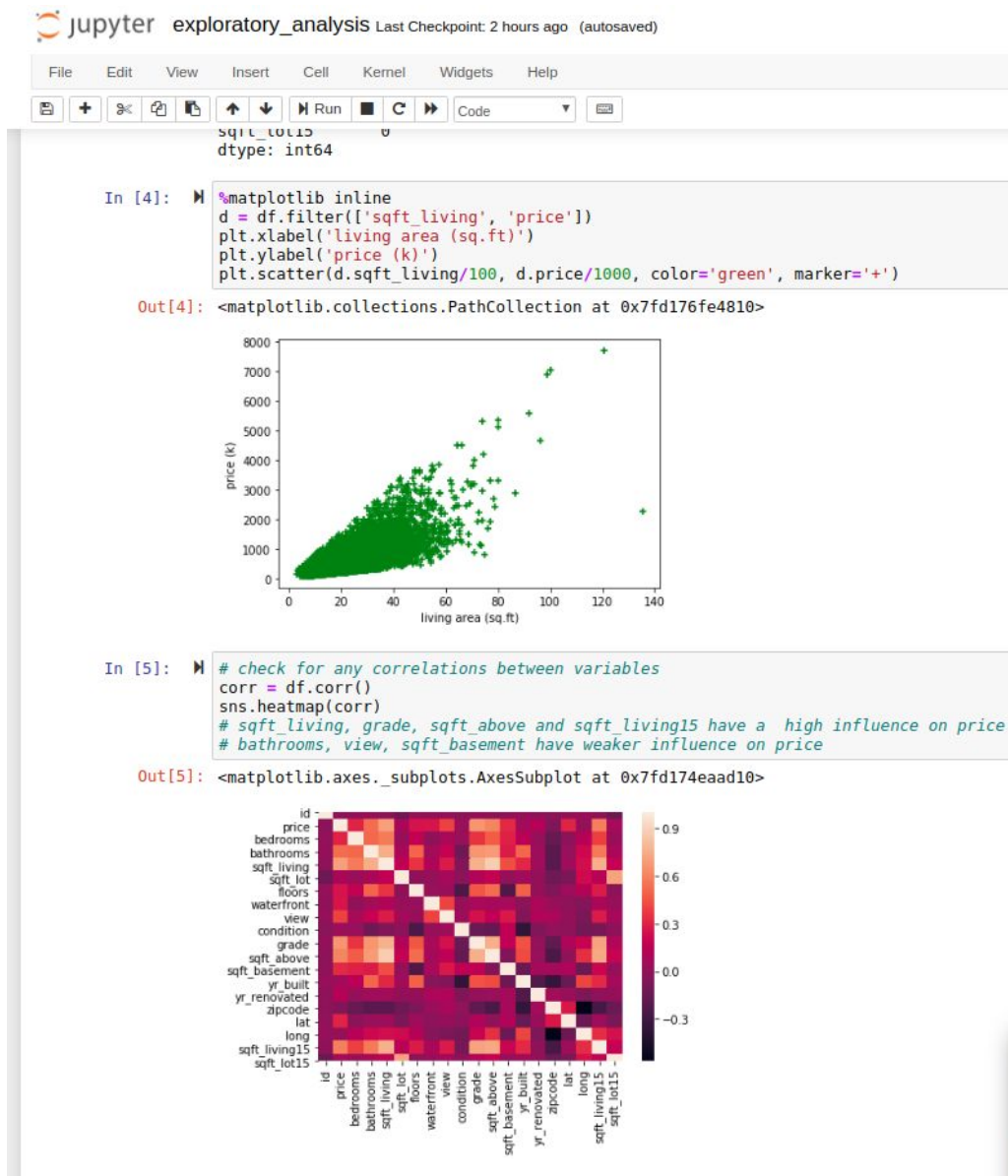
```
1 from mlxtend.frequent_patterns import apriori
2 from mlxtend.preprocessing import TransactionEncoder
3 import pandas as pd
4 import numpy as np
5
6 MIN_SUPP = 0.2 # Support
7 DATA_SET_SIZE = 5000 # SEGMENT size of data
8 dic = {}
9
10
11 def print_result():...
12
13
14
15
16
17
18
19
20
21 def add_to_dict(frequent_itemsets):...
22
23
24
25
26
27
28
29
30
31
32
33
34
35 def fis(dataset, split=DATA_SET_SIZE):...
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55 """ RUN CODE """
56 # Open dataset from file
57 fileName = './kosarak.dat'
58 lineList = [(line.rstrip('\n')).split(' ') for line in open(fileName) if not line.isspace()]
59 fis(lineList, DATA_SET_SIZE)
60
61 print_result()
```

Result:

```
items      :      support
-----
1          :      0.2046
11         :      0.3677
3          :      0.4548
6          :      0.6074
11-6       :      0.3273
3-6        :      0.2679
1056       :      0.5000
(venv) (base) alex@yoga-8104
```

## PART 3

In this task I have to calculate linear and polynomial regression for dataset about house prices using Python. I used Jupyter for this task.



### Regression

**Regression** is the process of investigating a relationship between an independent variable (x) and dependent variable (y). Therefore it is a predictive analysis method.

---

It gives an expression (model) than can be used to predict future values. The model that gives values on a straight line is termed as linear regression while others which can give values along a curved path is termed as polynomial regression.

**Linear regression** can be stated mathematically as:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Where

$\beta_0$  is the y intercept.

$\beta_1$  is the slope (  $x_2 - x_1 + y_2 - y_1$  )

$\varepsilon$  is the unobserved random error.

A **quadratic polynomial regression** can be expressed as:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$$

Here  $\beta_2$  is the coefficient

### Solution

I used Jupyter for this task since it gives very nice and easy plotting features. I learned to read the dataset using `panda.read_csv` method and for this i first converted the excel sheet to csv format.

After some experimentation and trying several examples I found that we can also read excel file directly:

```
dfx = pd.read_excel('dataset.xlsx', 'Sheet1') # file, sheet name
```

But to use this feature we need to install a pip package as follows:

```
pip2 install xlrd
```

A few commands that helped initially in understanding the data:

```
df.head() # top rows
```

---

```
Df[['col1','col2']] # select columns

pd.set_option('display.max_columns', 999) # show all columns

df.head(100).sort_values(by=['price']) # sort by price

df.describe() # provides stats info about df like mean, sd
```

### **Regression related functions are**

```
p1 = np.polyfit(x,y,1)

p2=np.polyfit(x,y,2)

p3=np.polyfit(x,y,3)
```

The above commands gives the expression/model of linear, quadratic and cubic regression respectively. While the commands below

This command simply plots the x and y variables

```
plot(x,y,'o')
```

These commands draw the linear, and polynomials

```
plot(xp, np.polyval(p1,xp), 'r-') # plot red line

plot(xp, np.polyval(p2,xp),'b--') # plot dashed blue line

plot(xp, np.polyval(p3,xp), 'g:') # plot green line
```

After some initial analysis it turned out that there is one column that gives prices but there are several columns that affect the price.

---

## Exploratory Analysis

### Top of the data set

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0

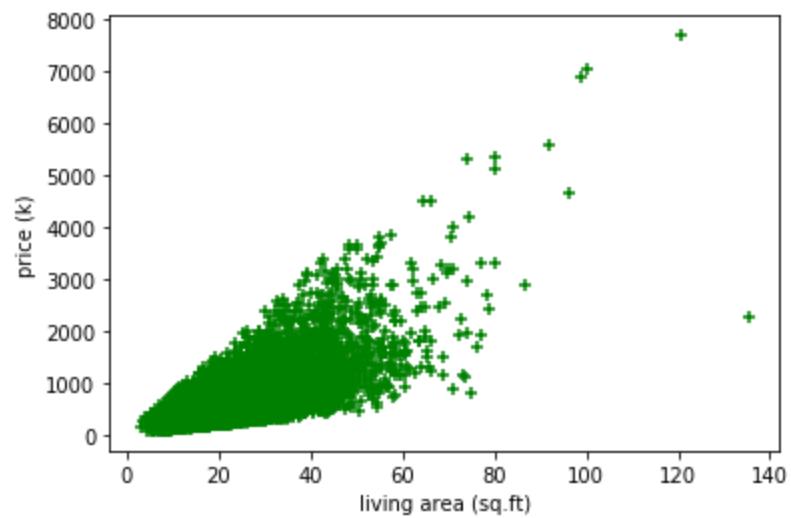
5 rows × 21 columns

### Data cleansing

```
Out[4]: id          0
        date        0
        price       0
        bedrooms    0
        bathrooms   0
        sqft_living  0
        sqft_lot     0
        floors       0
        waterfront   0
        view         0
        condition    0
        grade        0
        sqft_above   0
        sqft_basement 0
        yr_built     0
        yr_renovated  0
        zipcode      0
        lat          0
        long         0
        sqft_living15 0
        sqft_lot15   0
        dtype: int64
```

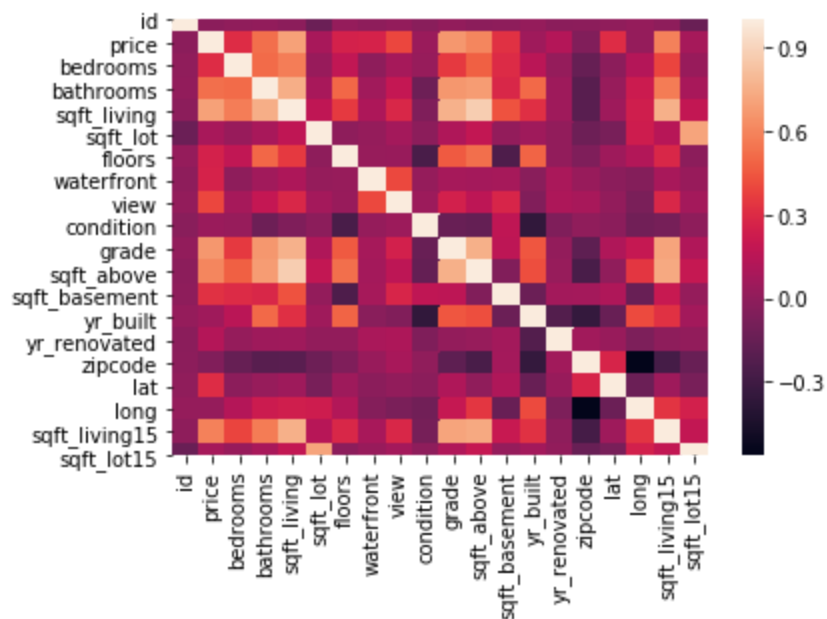
---

## Relationship between price and living area



The figure above show that living area has great influence on the price of the house.

## Correlation between variables



- sqft\_living, grade, sqft\_above and sqft\_living15 have a high influence on price
- bathrooms, view, sqft\_basement have weaker influence on price

## Linear regression

```
In [6]: reg = linear_model.LinearRegression()

In [7]: reg.fit(d[['sqft_living']], d.price)
Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [8]: reg.coef_
Out[8]: array([280.6235679])

In [9]: reg.intercept_
Out[9]: -43580.743094473844

In [10]: reg.predict([[1020]])
Out[10]: array([242655.29616092])

In [11]: reg.predict([[1180]])
Out[11]: array([287555.06702452])

In [12]: reg.predict([[1600]])
Out[12]: array([405416.96554144])

In [ ]:
```

## Single variable regression

First we computed Mean absolute error, MSE and R2 score for single feature (living area) by using 80% training data and 20% testing data.

```
X = df.sqft_living

X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(
    X, df.price, test_size=0.2, random_state = 5)
```

We got the following error and score.

```
print("Mean absolute error: %.2f" % np.mean(np.absolute(pred - y_test_new)))
print("Residual sum of squares (MSE): %.2f" % np.mean((pred - y_test_new) ** 2))
print("R2-score: %.2f" % r2_score(pred, y_test_new))

Mean absolute error: 174151.65
Residual sum of squares (MSE): 68199532519.86
R2-score: 0.02
```

---

## Multivariable regression

First we computed Mean absolute error, MSE and R2 score for multi features (except price, date) by using 80% training data and 20% testing data.

```
➤ X = df.drop('price', axis = 1)
  X = df.drop('date', axis = 1)

➤ X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(
  X, df.price, test_size=0.2, random_state = 5)
  # This creates a LinearRegression object
  linear_model = LinearRegression()

➤ # X_train = X_train.apply(pd.to_numeric, errors='coerce')
  # Y_train = Y_train.apply(pd.to_numeric, errors='coerce')
  linear_model.fit(X_train,Y_train)
```

And we got much better results.

```
➤ print("Mean absolute error: %.2f" % np.mean(np.absolute(pred - Y_test)))
  print("Residual sum of squares (MSE): %.2f" % np.mean((pred - Y_test) ** 2))
  print("R2-score: %.2f" % r2_score(pred , Y_test) )

Mean absolute error: 0.00
Residual sum of squares (MSE): 0.00
R2-score: 1.00
```

## RESULT:

Multivariable regression gives better price estimation because it takes into account the weightage of all influencing variables.