PHONE
+385 914001982
email
zvonimir.klenovic@gmail.com

ZVONIMIR
KLENOVIC
master engineer of graphic technology

820422-9150
ADRESS
Etsarvagen 10, Johanneshov 12143
Stockholm

PRESENTATION
DATA HANDLING

DATA CLEANING AND PREPARATION

1. Creating a DataBase in SQL

   Accessing a DataBase
   Creating a Backup

2. Exploring a DataBase with SQL

3. DATA cleaning with Python and Pandas

   Detecting missing values
   Summarizing missing values
   Replacing missing values

5. Joining Multiple DataBases

SIMPLE LINEAR REGRESSION

7. Exploring Correlations between Variables

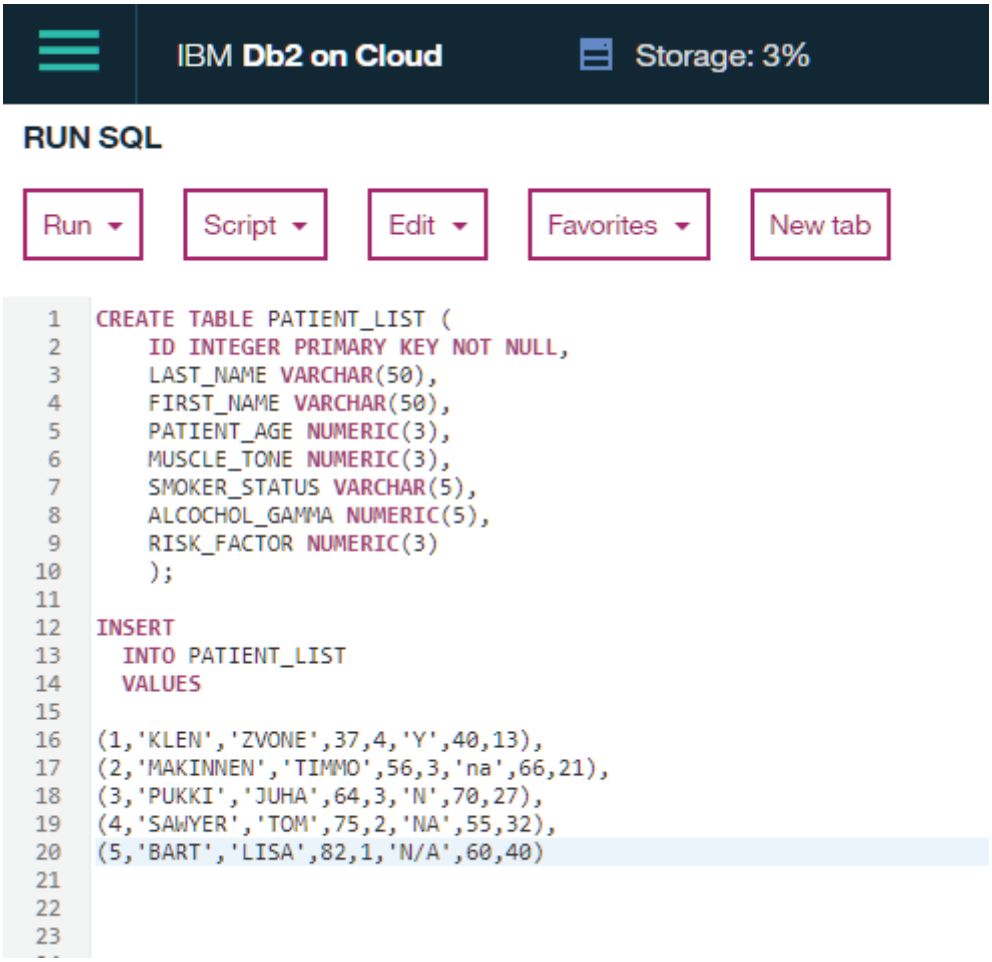
8. Creating Train and Test set

9. Using Sklearn Package for Data Modeling

   Evaluation

CREATING A DATABASE IN SQL

I created 2 imaginary patient Databases in SQL on IBM Cloud




```sql
CREATE TABLE PATIENT_LIST (
    ID INTEGER PRIMARY KEY NOT NULL,
    LAST_NAME VARCHAR(50),
    FIRST_NAME VARCHAR(50),
    PATIENT_AGE NUMERIC(3),
    MUSCLE_TONE NUMERIC(3),
    SMOKER_STATUS VARCHAR(5),
    ALCOCHOL_GAMMA NUMERIC(5),
    RISK_FACTOR NUMERIC(3)
    );

INSERT
  INTO PATIENT_LIST
  VALUES

(1,'KLEN','ZVONE',37,4,'Y',40,13),
(2,'MAKINNEN','TIMMO',56,3,'na',66,21),
(3,'PUKKI','JUHA',64,3,'N',70,27),
(4,'SAWYER','TOM',75,2,'NA',55,32),
(5,'BART','LISA',82,1,'N/A',60,40)
```

NRM17357.PATIENTS

Delete Table Export to CSV

	ID INTEGER	LAST_NAME VARCHAR(50)	FIRST_NAME VARCHAR(50)	PATIENT_AGE DECIMAL(3, 0)	MUSCLE_TONE DECIMAL(3, 0)	SMOKER_STATUS VARCHAR(5)	ALCOCHOL_GAMMA DECIMAL(5, 0)	RISK_FACTOR DECIMAL(3, 0)
1	1	KLEN	ZVONE	37	4	Y	40	13
2	2	MAKINNEN	TIMMO	56	3	na	66	21
3	3	PUKKI	JUHA	64	3	N	70	27
4	4	SAWYER	TOM	75	2	NA	55	32
5	5	BART	LISA	82	1	N/A	60	40

NRM17357.PATIENTS2

Delete Table Export to CSV

	ID INTEGER	LAST_NAME VARCHAR(50)	FIRST_NAME VARCHAR(50)	PATIENT_AGE DECIMAL(3, 0)	MUSCLE_TONE DECIMAL(3, 0)	WEIGHT DECIMAL(3, 0)	ALCOCHOL_GAMMA DECIMAL(5, 0)	RISK_FACTOR DECIMAL(3, 0)
1	1	BARA	MARKO	38	4	85	40	10
2	2	JAMES	HENRY	54	3	92	100	19
3	3	LARSSON	ALEX	68	2	80	120	39
4	4	SANDERS	TOM	77	1	99	111	47
5	5	BERG	JANNE	82	1	80	100	60

I can access them remotely via link or save as .csv on a local hard drive

Service credentials-1 12. OŽU. 2019 - 04:51:11 PM View credentials


```
{
  "hostname": "dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net",
  "password": "5gzf+8btmhwg1nv7",
  "https_url": "https://dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:8443",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=nrm17357;PWD=5gzf+8btmhwg1nv7;Security=SSL;",
  "host": "dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB",
  "uri": "db2://nrm17357:5gzf%2B8btmhwg1nv7@dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB",
  "db": "BLUDB",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net;PORT=50000;PROTOCOL=TCPIP;UID=nrm17357;PWD=5gzf+8btmhwg1nv7;",
  "username": "nrm17357",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50001/BLUDB:sslConnection=true;"
}
```

CONNECTING TO A DATABASE REMOTELY

We are sucessfully connected to a DataBase on IBM Cloud in Jupyter Notebook loading external SQL in Python

```
# Load the SQL extension and establish a connection with the database

%load_ext sql
```

```
# Enter the connection string for your Db2 on Cloud database instance below

%sql ibm_db_sa://nrm17357:5gzf%2B8btmhwg1nv7@dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB
```

```
'Connected: nrm17357@BLUDB'
```

EXPLORING THE DATABASE

We can display the full count of patients and the whole DataBase (in this case PATIENTS2)

```
%sql SELECT COUNT (*) FROM PATIENTS2
```

```
 * ibm_db_sa://nrm17357:***@dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB
Done.
```

1
5

```
%sql SELECT * FROM PATIENTS2
```

```
 * ibm_db_sa://nrm17357:***@dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB
Done.
```

id	last_name	first_name	patient_age	muscle_tone	weight	alcochol_gamma	risk_factor
1	BARA	MARKO	38	4	85	40	10
2	JAMES	HENRY	54	3	92	100	19
3	LARSSON	ALEX	68	2	80	120	39
4	SANDERS	TOM	77	1	99	111	47
5	BERG	JANNE	82	1	80	100	60

We can find out specific information from the DataBase

In this example a list of all patients is displayed with Gamma value higher than 100

```
%sql SELECT last_name, first_name FROM PATIENTS2 WHERE alcochol_gamma>100
```

```
 * ibm_db_sa://nrm17357:***@dashdb-txn-sbox-yp-lon02-01.services.eu-gb.bluemix.net:50000/BLUDB
Done.
```

last_name	first_name
LARSSON	ALEX
SANDERS	TOM

DATA CLEANING WITH PYTHON AND PANDAS

Loading the DataBase from a Local hard drive

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

```
df
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	SMOKER_STATUS	ALCOCHOL_GAMMA	RISK_FACTOR
0	1	KLEN	ZVONE	37	4	Y	40	13
1	2	MAKINNEN	TIMMO	56	3	na	66	21
2	3	PUKKI	JUHA	64	3	N	70	27
3	4	SAWYER	TOM	75	2	NaN	55	32
4	5	BART	LISA	82	1	NaN	60	40

DETECTING AND SUMMARIZING MISSING VALUES

Here we are dealing with some Non Standard missing values (SMOKER_STATUS column)

```
print (df.isnull().sum())
```

```
ID                0
LAST_NAME         0
FIRST_NAME        0
PATIENT_AGE       0
MUSCLE_TONE       0
SMOKER_STATUS     2
ALCOCHOL_GAMMA    0
RISK_FACTOR       0
dtype: int64
```

```
print (df['SMOKER_STATUS'])
```

```
0      Y
1     na
2      N
3    NaN
4    NaN
Name: SMOKER_STATUS, dtype: object
```

```
print (df['SMOKER_STATUS'].isnull())
```

```
0    False
1    False
2    False
3     True
4     True
Name: SMOKER_STATUS, dtype: bool
```

```
missing_values = ["na"]
df = pd.read_csv("PATIENTS.csv", na_values = missing_values)
```

```
print (df['SMOKER_STATUS'])
```

```
0      Y
1    NaN
2      N
3    NaN
4    NaN
Name: SMOKER_STATUS, dtype: object
```

In the SMOKER_STATUS column the correct entries should be Y (yes) or N (no) and the missing values should be labeled as NaN, some people enter values differently if the information is not available (na, n/a, NA, N/A), what we have done here is just converted non standard missing values to standard missing values

If we display the DataBase now we can see that the missing values are now all standard (SMOKER_STATUS)

```
df
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	SMOKER_STATUS	ALCOCHOL_GAMMA	RISK_FACTOR
0	1	KLEN	ZVONE	37	4	Y	40	13
1	2	MAKINNEN	TIMMO	56	3	NaN	66	21
2	3	PUKKI	JUHA	64	3	N	70	27
3	4	SAWYER	TOM	75	2	NaN	55	32
4	5	BART	LISA	82	1	NaN	60	40

Here I have added the missing value for patient "Makinnen" changing it to Smoker (Y). I located him under the index number (1).

```
df.loc[1,'SMOKER_STATUS'] = 'Y'
```

```
df
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	SMOKER_STATUS	ALCOCHOL_GAMMA	RISK_FACTOR
0	1	KLEN	ZVONE	37	4	Y	40	13
1	2	MAKINNEN	TIMMO	56	3	Y	66	21
2	3	PUKKI	JUHA	64	3	N	70	27
3	4	SAWYER	TOM	75	2	NaN	55	32
4	5	BART	LISA	82	1	NaN	60	40

I have also changed the ALCOCHOL_GAMMA column name which was written incorrectly to GAMMA.

```
df.rename(columns={'ALCOCHOL_GAMMA':'GAMMA'}, inplace=True)
df
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	SMOKER_STATUS	GAMMA	RISK_FACTOR
0	1	KLEN	ZVONE	37	4	Y	40	13
1	2	MAKINNEN	TIMMO	56	3	Y	66	21
2	3	PUKKI	JUHA	64	3	N	70	27
3	4	SAWYER	TOM	75	2	NaN	55	32
4	5	BART	LISA	82	1	NaN	60	40

I am now accessing the second DataBase, checking for missing values and correcting ALCOCHOL_GAMMA column name.

```
df2 = pd.read_csv('PATIENTS2.csv')
df2
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	WEIGHT	ALCOCHOL_GAMMA	RISK_FACTOR
0	1	BARA	MARKO	38	4	85	40	10
1	2	JAMES	HENRY	54	3	92	100	19
2	3	LARSSON	ALEX	68	2	80	120	39
3	4	SANDERS	TOM	77	1	99	111	47
4	5	BERG	JANNE	82	1	80	100	60

```
print (df2.isnull().sum())
```

```
ID              0
LAST_NAME       0
FIRST_NAME      0
PATIENT_AGE     0
MUSCLE_TONE     0
WEIGHT          0
ALCOCHOL_GAMMA  0
RISK_FACTOR     0
dtype: int64
```

```
df2.rename(columns={'ALCOCHOL_GAMMA':'GAMMA'}, inplace=True)
df2
```

	ID	LAST_NAME	FIRST_NAME	PATIENT_AGE	MUSCLE_TONE	WEIGHT	GAMMA	RISK_FACTOR
0	1	BARA	MARKO	38	4	85	40	10
1	2	JAMES	HENRY	54	3	92	100	19
2	3	LARSSON	ALEX	68	2	80	120	39
3	4	SANDERS	TOM	77	1	99	111	47
4	5	BERG	JANNE	82	1	80	100	60

Before joining PATIENTS1 and PATIENTS2 DataBases I am interested in doing some investigating.
I can find out which columns appear in both DataBases and which columns differ and are specific.

```
Intersection = df.columns & df2.columns
Intersection
```

```
Index(['ID', 'LAST_NAME', 'FIRST_NAME', 'PATIENT_AGE', 'MUSCLE_TONE', 'GAMMA',
       'RISK_FACTOR'],
      dtype='object')
```

```
P1 = df.columns
P1
```

```
Index(['ID', 'LAST_NAME', 'FIRST_NAME', 'PATIENT_AGE', 'MUSCLE_TONE',
       'SMOKER_STATUS', 'GAMMA', 'RISK_FACTOR'],
      dtype='object')
```

```
P2 = df2.columns
P2
```

```
Index(['ID', 'LAST_NAME', 'FIRST_NAME', 'PATIENT_AGE', 'MUSCLE_TONE', 'WEIGHT',
       'GAMMA', 'RISK_FACTOR'],
      dtype='object')
```

```
P1.difference(P2)
```

```
Index(['SMOKER_STATUS'], dtype='object')
```

```
P2.difference(P1)
```

```
Index(['WEIGHT'], dtype='object')
```

Here we can notice that SMOKER_STATUS only appears in DataBase PATIENTS1 and that WEIGHT column only appears in DataBase PATIENTS2.

I am now joining two DataBases and displaying the resulting new DataBase

```
df_new = pd.concat([df, df2])
df_new
```

In the newly formed DataBase we can see that the index values arent specific enough.
We can also notice that most of our missing values appear to be in the SMOKER_STATUS column.
Also I find that the ID column is irrelevant.

	FIRST_NAME	GAMMA	ID	LAST_NAME	MUSCLE_TONE	PATIENT_AGE	RISK_FACTOR	SMOKER_STATUS	WEIGHT
0	ZVONE	40	1	KLEN	4	37	13	Y	NaN
1	TIMMO	66	2	MAKINNEN	3	56	21	na	NaN
2	JUHA	70	3	PUKKI	3	64	27	N	NaN
3	TOM	55	4	SAWYER	2	75	32	NaN	NaN
4	LISA	60	5	BART	1	82	40	NaN	NaN
0	MARKO	40	1	BARA	4	38	10	NaN	85.0
1	HENRY	100	2	JAMES	3	54	19	NaN	92.0
2	ALEX	120	3	LARSSON	2	68	39	NaN	80.0
3	TOM	111	4	SANDERS	1	77	47	NaN	99.0
4	JANNE	100	5	BERG	1	82	60	NaN	80.0

For easier retreival of a patient I have decided to switch index with the LAST_NAME.

```
df_new.set_index('LAST_NAME', inplace=True)
df_new
```

LAST_NAME	FIRST_NAME	GAMMA	ID	MUSCLE_TONE	PATIENT_AGE	RISK_FACTOR	SMOKER_STATUS	WEIGHT
KLEN	ZVONE	40	1	4	37	13	Y	NaN
MAKINNEN	TIMMO	66	2	3	56	21	Y	NaN
PUKKI	JUHA	70	3	3	64	27	N	NaN
SAWYER	TOM	55	4	2	75	32	NaN	NaN
BART	LISA	60	5	1	82	40	NaN	NaN
BARA	MARKO	40	1	4	38	10	NaN	85.0
JAMES	HENRY	100	2	3	54	19	NaN	92.0
LARSSON	ALEX	120	3	2	68	39	NaN	80.0
SANDERS	TOM	111	4	1	77	47	NaN	99.0
BERG	JANNE	100	5	1	82	60	NaN	80.0

My new DataBase now has 10 patients and 8 variables

```
df_new.shape
```

```
(10, 8)
```

```
df_new.columns.values
```

```
array(['FIRST_NAME', 'GAMMA', 'ID', 'MUSCLE_TONE', 'PATIENT_AGE',
       'RISK_FACTOR', 'SMOKER_STATUS', 'WEIGHT'], dtype=object)
```

```
df_new.drop(['ID','SMOKER_STATUS'], axis=1, inplace=True)
```

I have decided to drop the SMOKER_STATUS column with most missing values and also the ID column which I find irrelevant.

Our new DataBase now only contains standard missing values for the WEIGHT variable.

LAST_NAME	FIRST_NAME	GAMMA	MUSCLE_TONE	PATIENT_AGE	RISK_FACTOR	WEIGHT
KLEN	ZVONE	40	4	37	13	NaN
MAKINNEN	TIMMO	66	3	56	21	NaN
PUKKI	JUHA	70	3	64	27	NaN
SAWYER	TOM	55	2	75	32	NaN
BART	LISA	60	1	82	40	NaN
BARA	MARKO	40	4	38	10	85.0
JAMES	HENRY	100	3	54	19	92.0
LARSSON	ALEX	120	2	68	39	80.0
SANDERS	TOM	111	1	77	47	99.0
BERG	JANNE	100	1	82	60	80.0

One way of populating the missing values is by adding the average value. (In a real life scenario we wouldnt use this method because WEIGHT is a big factor in determing the Risk Factor for the surgery and recovery)

```
avg_weight = df_new['WEIGHT'].astype("float").mean(axis = 0)
print("Average WEIGHT:", avg_weight)
df_new['WEIGHT'].replace(np.nan, avg_weight, inplace = True)
```

```
Average WEIGHT: 87.2
```

```
df_new
```

LAST_NAME	FIRST_NAME	GAMMA	MUSCLE_TONE	PATIENT_AGE	RISK_FACTOR	WEIGHT
KLEN	ZVONE	40	4	37	13	87.2
MAKINNEN	TIMMO	66	3	56	21	87.2
PUKKI	JUHA	70	3	64	27	87.2
SAWYER	TOM	55	2	75	32	87.2
BART	LISA	60	1	82	40	87.2
BARA	MARKO	40	4	38	10	85.0
JAMES	HENRY	100	3	54	19	92.0
LARSSON	ALEX	120	2	68	39	80.0
SANDERS	TOM	111	1	77	47	99.0
BERG	JANNE	100	1	82	60	80.0

EXPLORING CORRELATIONS BETWEEN VARIABLES

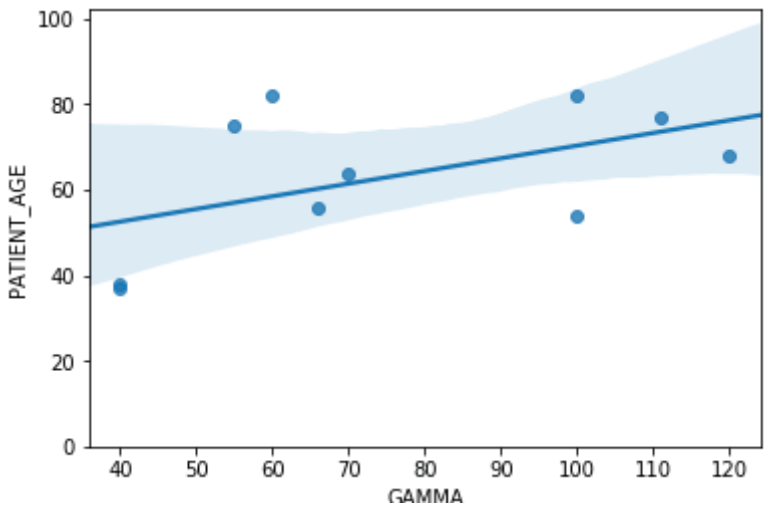
I am now looking at correlations between variables

```
df_new.corr()
```

	GAMMA	MUSCLE_TONE	PATIENT_AGE	RISK_FACTOR	WEIGHT
GAMMA	1.000000	-0.580465	0.519565	0.640618	0.113859
MUSCLE_TONE	-0.580465	1.000000	-0.963191	-0.937472	-0.097875
PATIENT_AGE	0.519565	-0.963191	1.000000	0.906272	0.007120
RISK_FACTOR	0.640618	-0.937472	0.906272	1.000000	-0.132504
WEIGHT	0.113859	-0.097875	0.007120	-0.132504	1.000000

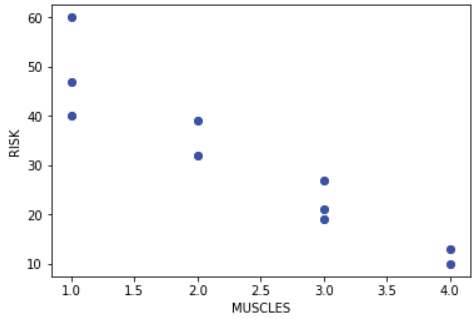
I am finding a somewhat positive linear relationship between age and gamma displayed in the scatterplot

```
sns.regplot(x='GAMMA', y='PATIENT_AGE', data=df_new)
plt.ylim(0,)
```

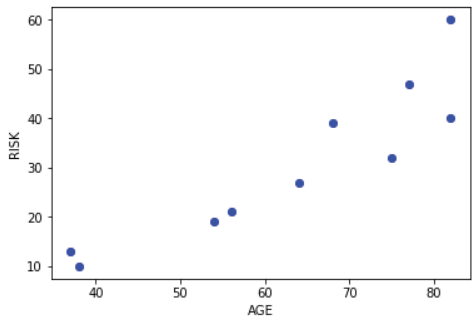


We can plot each of the variables (MUSCLE_TONE, AGE, WEIGHT) against RISK_FACTOR to see how linear is the relation.

```
plt.scatter(df_new.MUSCLE_TONE, df_new.RISK_FACTOR,  color='blue')
plt.xlabel('MUSCLES')
plt.ylabel('RISK')
plt.show()
```




```
plt.scatter(df_new.PATIENT_AGE, df_new.RISK_FACTOR,  color='blue')
plt.xlabel('AGE')
plt.ylabel('RISK')
plt.show()
```

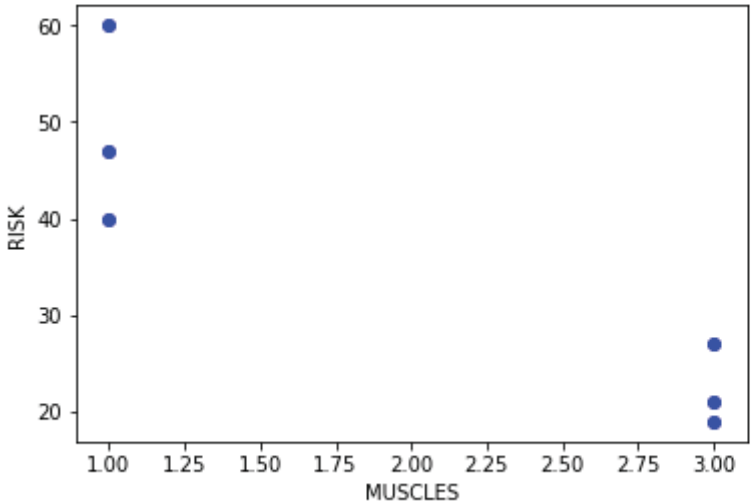


CREATING TRAIN AND TEST DATA SET

Creating train and test dataset: 80pcnt is used for training and 20pcnt for testing

```
msk = np.random.rand(len(df_new)) < 0.8
train = df_new[msk]
test = df_new[~msk]
```

```
plt.scatter(train.MUSCLE_TONE, train.RISK_FACTOR,  color='blue')
plt.xlabel('MUSCLES')
plt.ylabel('RISK')
plt.show()
```



We can see the train data distribution

SIMPLE LINEAR REGRESSION

USING SKLEARN PACKAGE TO FOR DATA MODELING

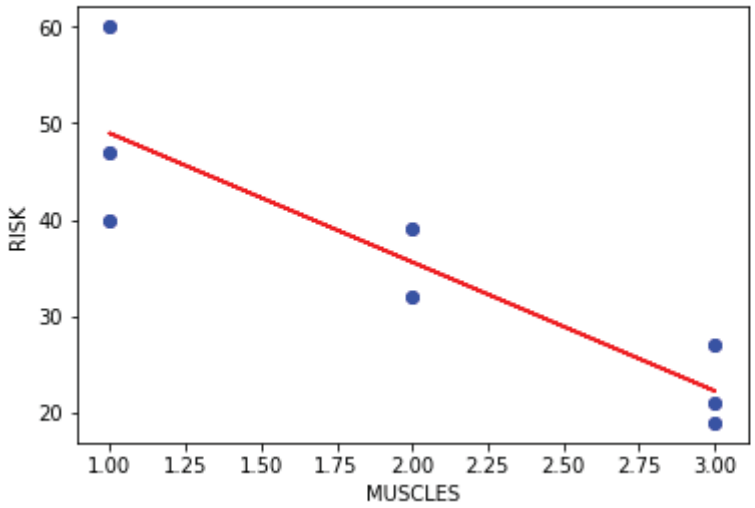
Coefficient and Intercept in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['MUSCLE_TONE']])
train_y = np.asanyarray(train[['RISK_FACTOR']])
regr.fit (train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[-14.68253968]]
Intercept:  [66.74603175]
```

```
plt.scatter(train.MUSCLE_TONE, train.RISK_FACTOR,  color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel('MUSCLES')
plt.ylabel('RISK')
```

```
Text(0, 0.5, 'RISK')
```



EVALUATION

We compare the actual values and predicted values to calculate the accuracy of a regression model.

```
from sklearn.metrics import r2_score

test_x = np.asanyarray(test[['MUSCLE_TONE']])
test_y = np.asanyarray(test[['RISK_FACTOR']])
test_y_hat = regr.predict(test_x)

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

```
Mean absolute error: 8.52
Residual sum of squares (MSE): 85.18
R2-score: 0.82
```