We looked into SQL language and how to get some basic data preparation done. Today we will look into R and how to get started with data analytics.

Creating a data.frame (or getting data from SQL Table)

Create a new notebook (Name: Day11_R_AnalyticsTasks, Language: R) and let's go. Now we will get data from SQL tables and DBFS files.

We will be using a database from Day10 and the table called temperature.

```
%sql
USE Day10;

SELECT * FROM temperature
```

For getting SQL query result into R data.frame, we will use SparkR package.

```
library(SparkR)
```

Getting Query results in R data frame (using SparkR R library)

```
temp_df <- sql("SELECT * FROM temperature")
```

With this *temp_df* data.frame we can start using R or SparkR functions. For example viewing the content of the data.frame.

```
showDF(temp_df)
```

Cmd 7

```
showDF(temp_df)
```

(2) Spark Jobs

```
+----------+---------------+---------+
|      date|mean_daily_temp|     city|
+----------+---------------+---------+
|05/12/2020|              1|Ljubljana|
|06/12/2020|              2|Ljubljana|
|07/12/2020|              2|Ljubljana|
|08/12/2020|              1|Ljubljana|
|09/12/2020|             -1|Ljubljana|
|10/12/2020|             -2|Ljubljana|
|11/12/2020|              0|Ljubljana|
|12/12/2020|              1|Ljubljana|
|13/12/2020|              2|Ljubljana|
|14/12/2020|              3|Ljubljana|
|05/12/2020|              6|  Seattle|
|06/12/2020|              7|  Seattle|
|07/12/2020|              6|  Seattle|
|08/12/2020|              5|  Seattle|
|09/12/2020|              6|  Seattle|
|10/12/2020|              7|  Seattle|
```

Command took 0.43 seconds -- by tomaz.kastrun@gmail.com at 10/12

This is a SparkR data.frame. you can aslo create a R data.frame by using *as.data.frame* function.

```
df <- as.data.frame(temp_df)
```

Creating standard R data.frame and it can be used with any other R packages.

Importing CSV file into R data.frame

Another way to get data into R data.frame is to feed data from CSV file. And in this case, SparkR library will again come in handy. Once data in data.frame, it can be used with other R libraries.

```
Day6 <- read.df("dbfs:/FileStore/Day6Data_dbfs.csv", source = "csv",
header="true", inferSchema = "true")
head(Day6)
```

```
Cmd 12

Day6 <- read.df("dbfs:/FileStore/Day6Data_dbfs.csv", source = "csv", header="true", inferSchema = "true")
head(Day6)

▸ (3) Spark Jobs

        date mean_daily_temp      city
1 05/12/2020              1 Ljubljana
2 06/12/2020              2 Ljubljana
3 07/12/2020              2 Ljubljana
4 08/12/2020              1 Ljubljana
5 09/12/2020             -1 Ljubljana
6 10/12/2020             -2 Ljubljana

Command took 0.63 seconds -- by tomaz.kastrun@gmail.com at 10/12/2020, 22:32:33 on databbricks_cl1_Standard
```

Doing simple analysis and visualisations

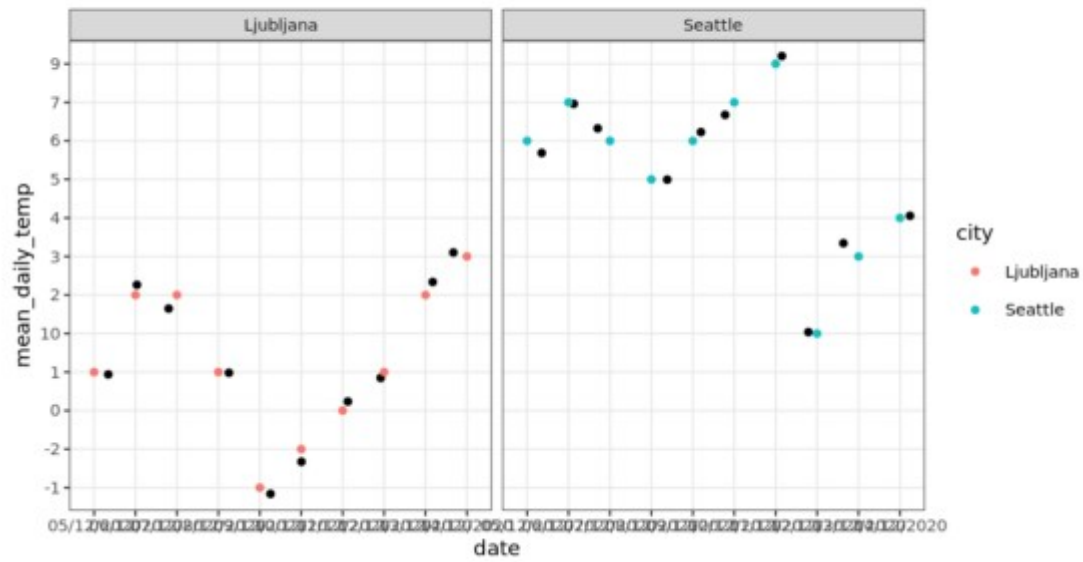
Once data is available in data.frame and it can be used for analysis and visualisations. Let's load ggplot2

```
library(ggplot2)
p <- ggplot(df, aes(date, mean_daily_temp))
p <- p + geom_jitter() + facet_wrap(~city)
p
```

And make the graph smaller and give it a theme.

```
options(repr.plot.height = 500, repr.plot.res = 120)
p + geom_point(aes(color = city)) + geom_smooth() +
   theme_bw()
```

```
options(repr.plot.height = 500, repr.plot.res = 120)
p + geom_point(aes(color = city)) + geom_smooth() +
   theme_bw()
```



Command took 0.46 seconds -- by tomaz.kastrun@gmail.com at 10/12/2020, 22:49:16 on databbricks_cl1_Standard

Once again, we can use other data wrangling packages. Both *dplyr* and *ggplot2* are preinstalled on Databricks Cluster.

```
library(dplyr)
```

When you load a library, nothing might be returned as a result. In case of warning, Databricks will display the warnings. Dplyr package can be used as any other package absolutely normally, without any

limitations.

```
df %>%
  dplyr::group_by(city) %>%
  dplyr::summarise(
       n = dplyr::n()
      ,mean_pos = mean(as.integer(df$mean_daily_temp))
 )
#%>% dplyr::filter( as.integer(df$date) > "2020/12/01")
```

```
df %>%
  dplyr::group_by(city) %>%
  dplyr::summarise(
       n = dplyr::n()
      ,mean_pos = mean(as.integer(df$mean_daily_temp))
 )
#%>% dplyr::filter( as.integer(df$date) > "2020/12/01")
```

```
# A tibble: 2 x 3
  city          n mean_pos
  <chr>     <int>    <dbl>
1 Ljubljana    10      3.6
2 Seattle      10      3.6
```

Command took 0.02 seconds -- by tomaz.kastrun@gmail.com at 10/12/2020, 23:07:42 on databb

Cmd 18

But note(!), dplyr functions might not work, and it is due to the collision of function names with SparkR library. SparkR has same functions (*arrange, between, coalesce, collect, contains, count, cume_dist, dense_rank, desc, distinct, explain, filter, first, group_by, intersect, lag, last, lead, mutate, n, n_distinct, ntile,*
percent_rank, rename, row_number, sample_frac, select, sql, summarize, union). In other to solve this collision, either detach (*detach("package:dplyr")*) the dplyr package, or we instance the package by: dplyr::summarise instead of just summarise.

Creating a simple linear regression

We can also use many of the R packages for data analysis, and in this case I will run simple regression, trying to predict the daily temperature. Simply run the regression function lm().

```
model <- lm(mean_daily_temp ~ city + date, data = df)
model
```

And run *base* r function summary() to get model insights.

```
summary(model)
```

```
summary(model)
```

```
Call:
lm(formula = mean_daily_temp ~ city + date, data = df)

Residuals:
   Min     1Q Median     3Q    Max 
 -2.20  -1.05   0.00   1.05   2.20 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)     8.000e-01  1.606e+00   0.498 0.630313    
citySeattle     5.400e+00  9.684e-01   5.576 0.000345 ***
date06/12/2020  1.000e+00  2.165e+00   0.462 0.655176    
date07/12/2020  5.000e-01  2.165e+00   0.231 0.822552    
date08/12/2020 -5.000e-01  2.165e+00  -0.231 0.822552    
date09/12/2020 -1.000e+00  2.165e+00  -0.462 0.655176    
date10/12/2020 -1.000e+00  2.165e+00  -0.462 0.655176    
date11/12/2020  1.000e+00  2.165e+00   0.462 0.655176    
date12/12/2020  2.000e+00  2.165e+00   0.924 0.379779    
date13/12/2020 -1.000e+00  2.165e+00  -0.462 0.655176    
```

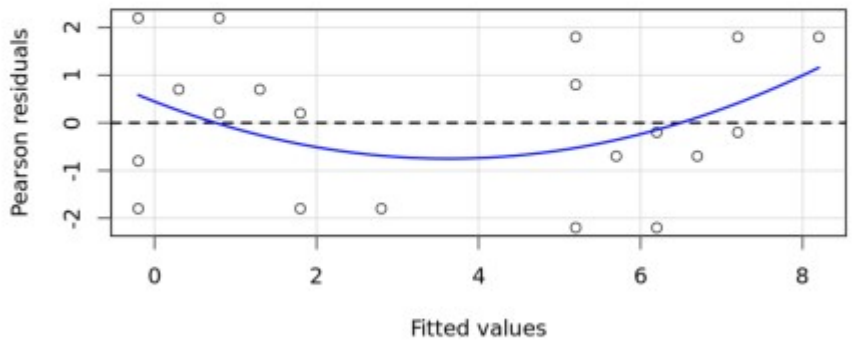
Command took 0.02 seconds -- by tomaz.kastrun@gmail.com at 10/12/2020, 23:11:

```
confint(model)
```

In addition, you can directly install any missing or needed package in notebook (R engine and Databricks Runtime environment version should be applied). In this case, I am running a *residualPlot()* function from extra installed package *car*.

```
install.packages("car")
library(car)
residualPlot(model)
```

```
install.packages("car")
library(car)
residualPlot(model)
```



Command took 0.55 minutes -- by tomaz.kastrun@gmail.com at 10/12/2020, 23:20:11 on databbricks_cll_Standard

Azure Databricks will generate RMarkdown notebook when using R Language as Kernel language….