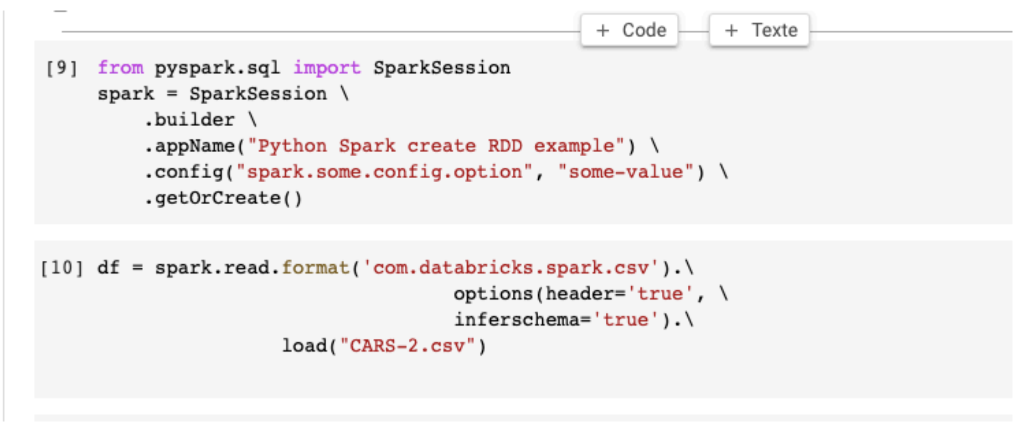
In the first data set and third data set we use : Random Forest Classifier .

A random forest is a machine learning classification algorithm. Random forests are generated collections of decision trees. We're also going to track the time it takes to train our model.

First :

Load the data set :

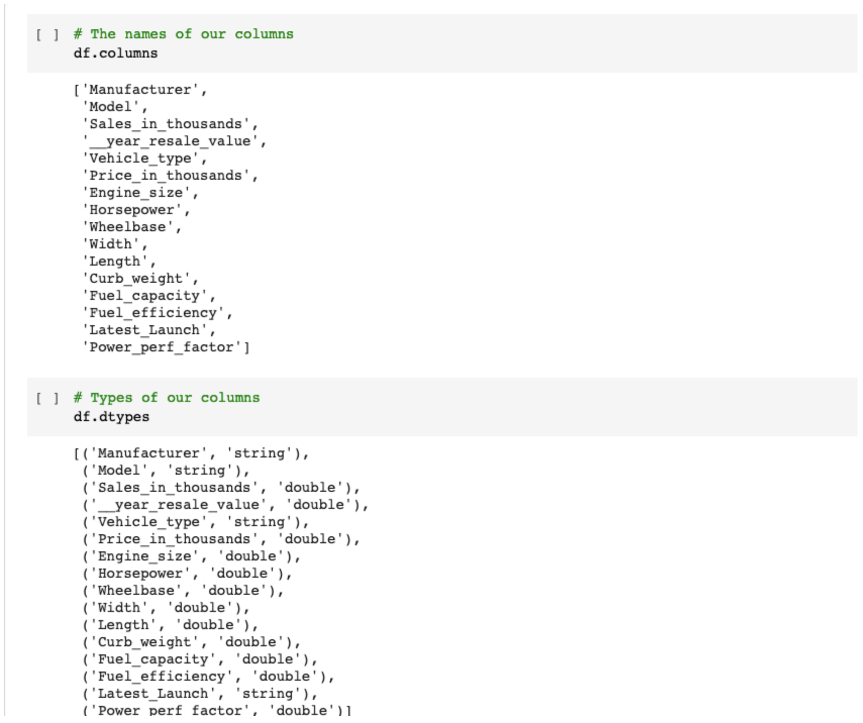


```
[9]  from pyspark.sql import SparkSession
     spark = SparkSession \
         .builder \
         .appName("Python Spark create RDD example") \
         .config("spark.some.config.option", "some-value") \
         .getOrCreate()
```

```
[10] df = spark.read.format('com.databricks.spark.csv').\
                               options(header='true', \
                               inferschema='true').\
                     load("CARS-2.csv")
```

Second:

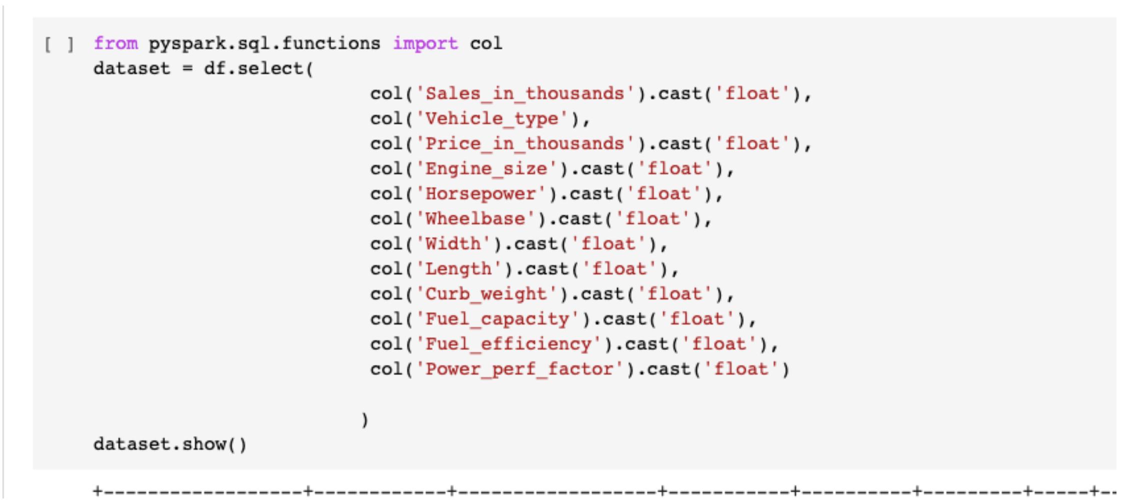
Data understanding :



```
[ ]  # The names of our columns
     df.columns

     ['Manufacturer',
      'Model',
      'Sales_in_thousands',
      '__year_resale_value',
      'Vehicle_type',
      'Price_in_thousands',
      'Engine_size',
      'Horsepower',
      'Wheelbase',
      'Width',
      'Length',
      'Curb_weight',
      'Fuel_capacity',
      'Fuel_efficiency',
      'Latest_Launch',
      'Power_perf_factor']
```

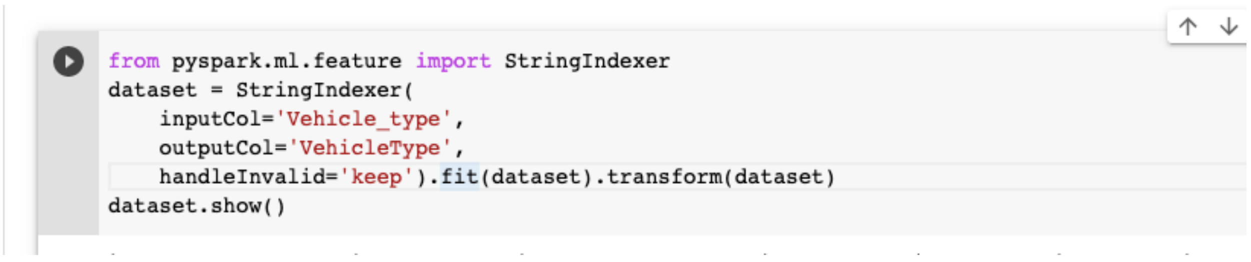
```
[ ]  # Types of our columns
     df.dtypes

     [('Manufacturer', 'string'),
      ('Model', 'string'),
      ('Sales_in_thousands', 'double'),
      ('__year_resale_value', 'double'),
      ('Vehicle_type', 'string'),
      ('Price_in_thousands', 'double'),
      ('Engine_size', 'double'),
      ('Horsepower', 'double'),
      ('Wheelbase', 'double'),
      ('Width', 'double'),
      ('Length', 'double'),
      ('Curb_weight', 'double'),
      ('Fuel_capacity', 'double'),
      ('Fuel_efficiency', 'double'),
      ('Latest_Launch', 'string'),
      ('Power_perf_factor', 'double')]
```

Cast the data :



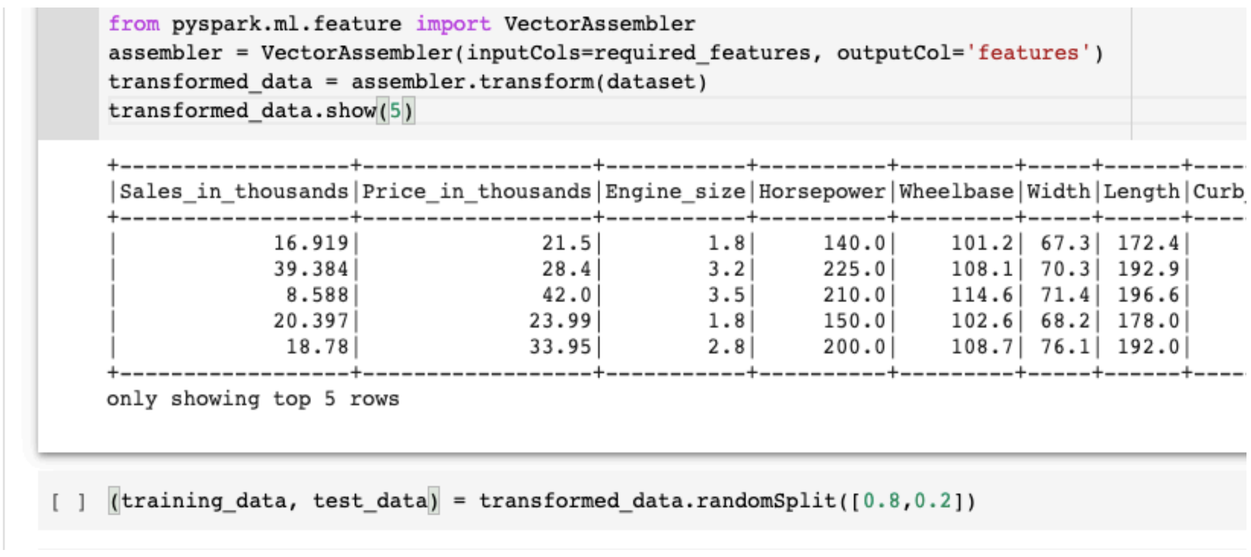
```
[ ] from pyspark.sql.functions import col
    dataset = df.select(
                              col('Sales_in_thousands').cast('float'),
                              col('Vehicle_type'),
                              col('Price_in_thousands').cast('float'),
                              col('Engine_size').cast('float'),
                              col('Horsepower').cast('float'),
                              col('Wheelbase').cast('float'),
                              col('Width').cast('float'),
                              col('Length').cast('float'),
                              col('Curb_weight').cast('float'),
                              col('Fuel_capacity').cast('float'),
                              col('Fuel_efficiency').cast('float'),
                              col('Power_perf_factor').cast('float')

                             )
    dataset.show()

    +------------------+------------+------------------+-----------+----------+---------+-----+--
```

Now, the Spark ML library only works with numeric data. But we still want to use the Sex and the Embarked column. For that, we will need to encode them. To do it let's use something called the StringIndexer



```
from pyspark.ml.feature import StringIndexer
dataset = StringIndexer(
    inputCol='Vehicle_type',
    outputCol='VehicleType',
    handleInvalid='keep').fit(dataset).transform(dataset)
dataset.show()
```

Assemble all the features with VectorAssembler , Additionally, we need to split the data into a training set and a test set. The training set will be used to create the model. The test set will be used to test the validity of the generated model.

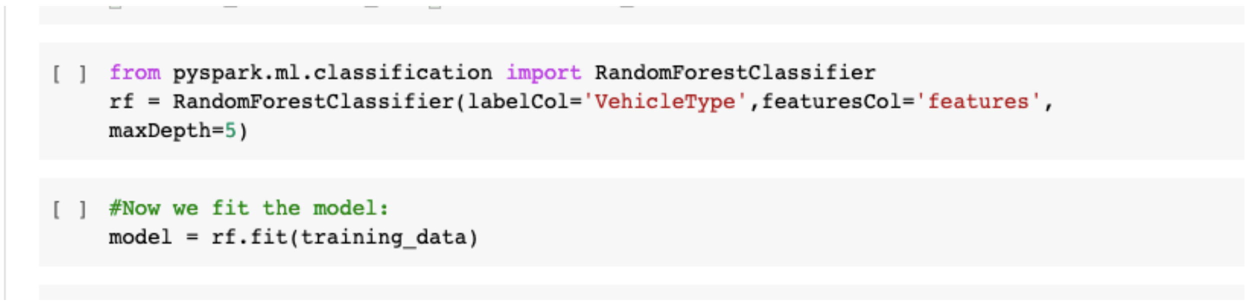


```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(dataset)
transformed_data.show(5)

+------------------+------------------+-----------+----------+---------+-----+------+----
|Sales_in_thousands|Price_in_thousands|Engine_size|Horsepower|Wheelbase|Width|Length|Curb
+------------------+------------------+-----------+----------+---------+-----+------+----
|            16.919|              21.5|        1.8|     140.0|    101.2| 67.3| 172.4|
|            39.384|              28.4|        3.2|     225.0|    108.1| 70.3| 192.9|
|             8.588|              42.0|        3.5|     210.0|    114.6| 71.4| 196.6|
|            20.397|             23.99|        1.8|     150.0|    102.6| 68.2| 178.0|
|             18.78|             33.95|        2.8|     200.0|    108.7| 76.1| 192.0|
+------------------+------------------+-----------+----------+---------+-----+------+----
only showing top 5 rows
```

```
[ ] (training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
```

Train the random forest :

A random forest is a machine learning classification algorithm. Random forests are generated collections of decision trees. We're also going to track the time it takes to train our model.

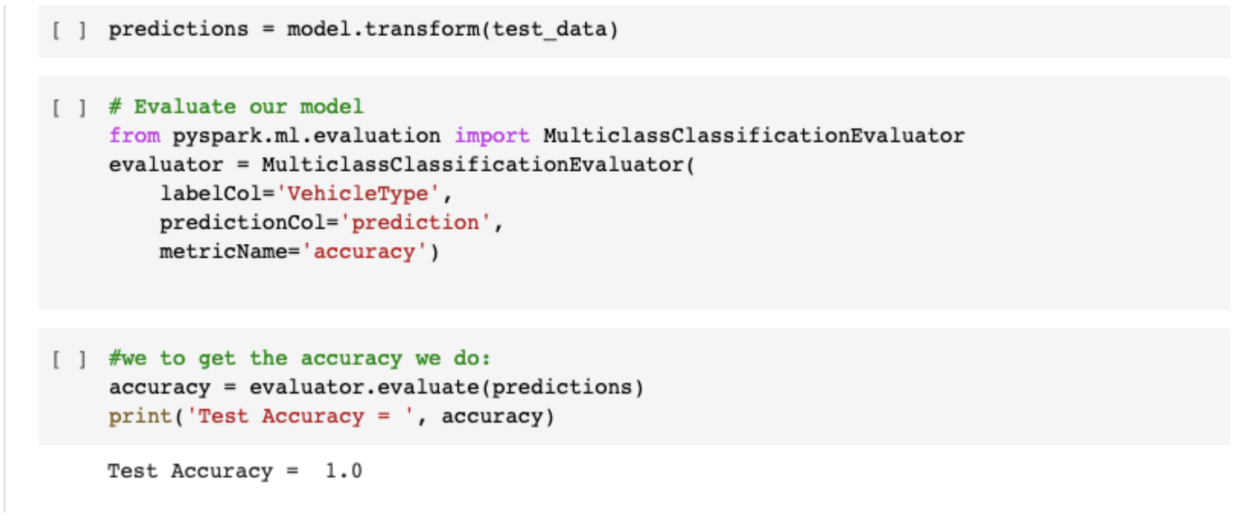


```
[ ] from pyspark.ml.classification import RandomForestClassifier
    rf = RandomForestClassifier(labelCol='VehicleType',featuresCol='features',
    maxDepth=5)
```

```
[ ] #Now we fit the model:
    model = rf.fit(training_data)
```

Make predictions and compute accuracy

Once we've trained our random forest model, we need to make predictions and test the accuracy of the model. Fortunately, there is a handy predict() function available. The accuracy is defined as the total number of correct predictions divided by the total number of predictions.



```
[ ] predictions = model.transform(test_data)
```

```
[ ] # Evaluate our model
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    evaluator = MulticlassClassificationEvaluator(
        labelCol='VehicleType',
        predictionCol='prediction',
        metricName='accuracy')
```

```
[ ] #we to get the accuracy we do:
    accuracy = evaluator.evaluate(predictions)
    print('Test Accuracy = ', accuracy)

    Test Accuracy =  1.0
```

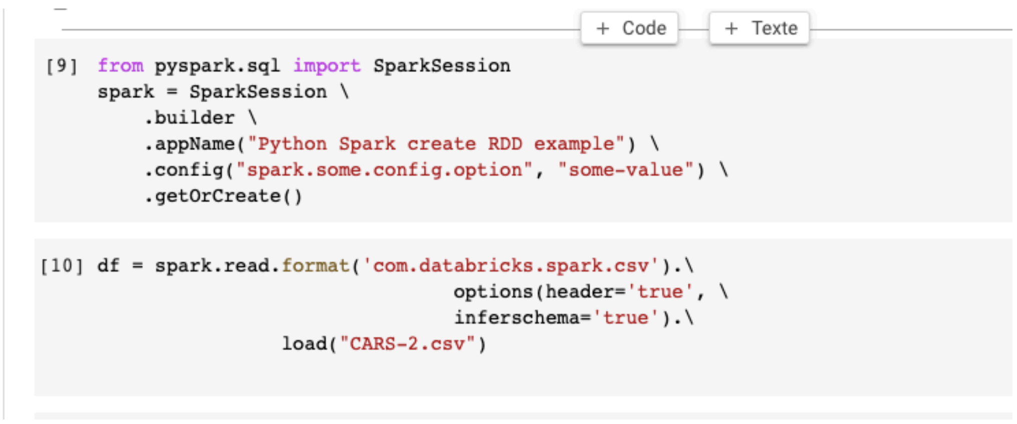
In the second data set we use LinearRegression:

The input data set contains data about details of various video games .

Based on the information provided, the goal is to come up with a model to predict Global_Sales value of a given games .

First :

Load the data set :

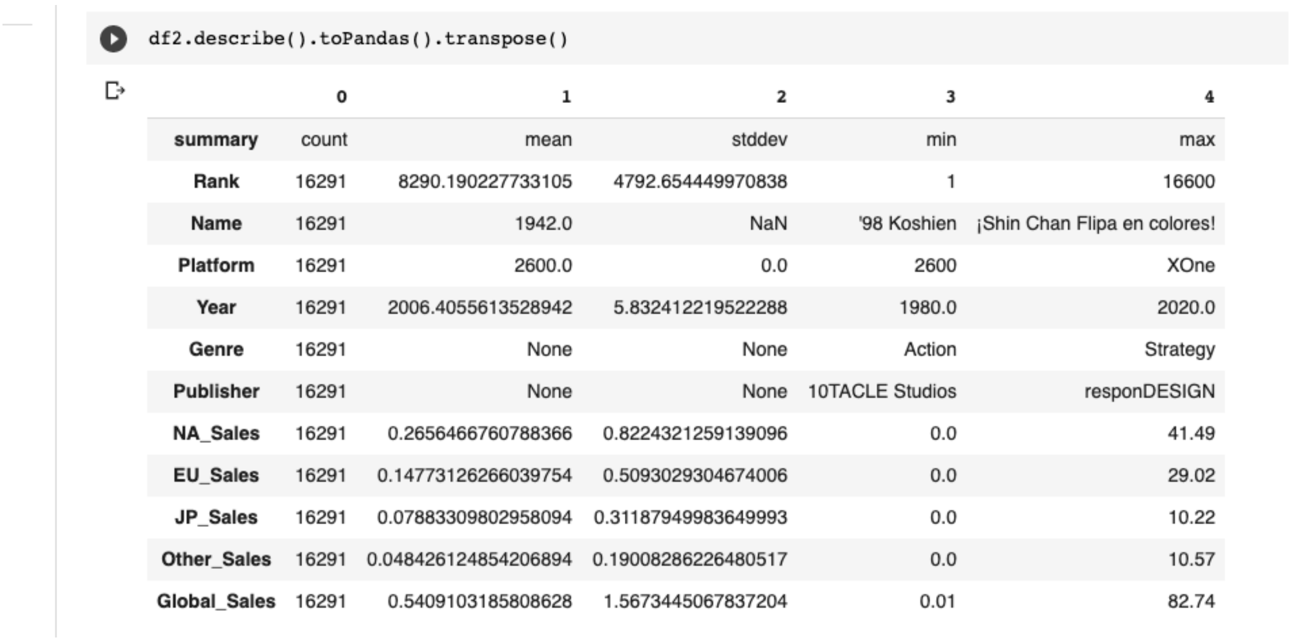


```
[9] from pyspark.sql import SparkSession
    spark = SparkSession \
        .builder \
        .appName("Python Spark create RDD example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

```
[10] df = spark.read.format('com.databricks.spark.csv').\
                               options(header='true', \
                               inferschema='true').\
                load("CARS-2.csv")
```

Second:

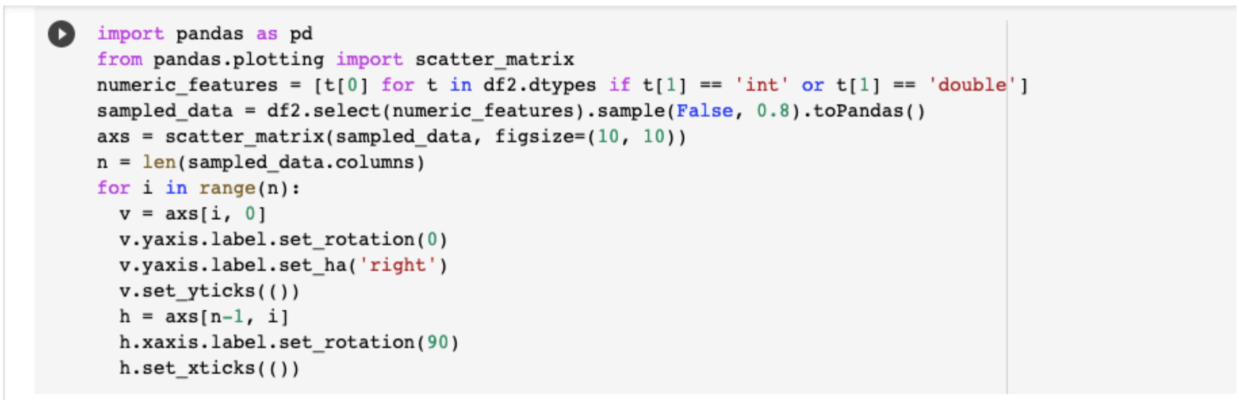
Perform descriptive analytics :



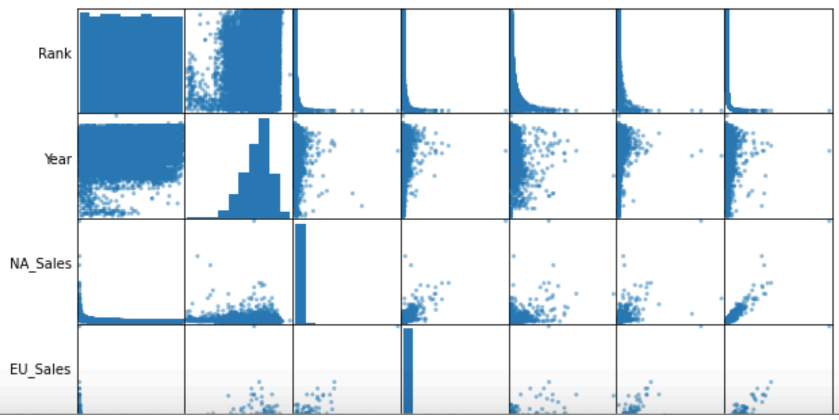
```
df2.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
Rank	16291	8290.190227733105	4792.654449970838	1	16600
Name	16291	1942.0	NaN	'98 Koshien	¡Shin Chan Flipa en colores!
Platform	16291	2600.0	0.0	2600	XOne
Year	16291	2006.4055613528942	5.832412219522288	1980.0	2020.0
Genre	16291	None	None	Action	Strategy
Publisher	16291	None	None	10TACLE Studios	responDESIGN
NA_Sales	16291	0.2656466760788366	0.8224321259139096	0.0	41.49
EU_Sales	16291	0.14773126266039754	0.5093029304674006	0.0	29.02
JP_Sales	16291	0.07883309802958094	0.31187949983649993	0.0	10.22
Other_Sales	16291	0.048426124854206894	0.19008286226480517	0.0	10.57
Global_Sales	16291	0.5409103185808628	1.5673445067837204	0.01	82.74

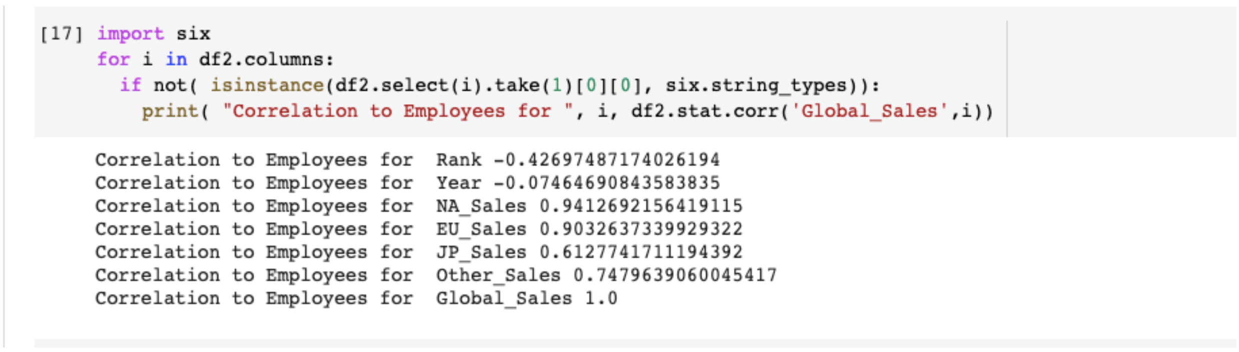
Scatter matrix is a great way to roughly determine if we have a linear correlation between multiple independent variables.



```
import pandas as pd
from pandas.plotting import scatter_matrix
numeric_features = [t[0] for t in df2.dtypes if t[1] == 'int' or t[1] == 'double']
sampled_data = df2.select(numeric_features).sample(False, 0.8).toPandas()
axs = scatter_matrix(sampled_data, figsize=(10, 10))
n = len(sampled_data.columns)
for i in range(n):
  v = axs[i, 0]
  v.yaxis.label.set_rotation(0)
  v.yaxis.label.set_ha('right')
  v.set_yticks(())
  h = axs[n-1, i]
  h.xaxis.label.set_rotation(90)
  h.set_xticks(())
```



It’s hard to see. Let’s find correlation between independent variables and target variable.

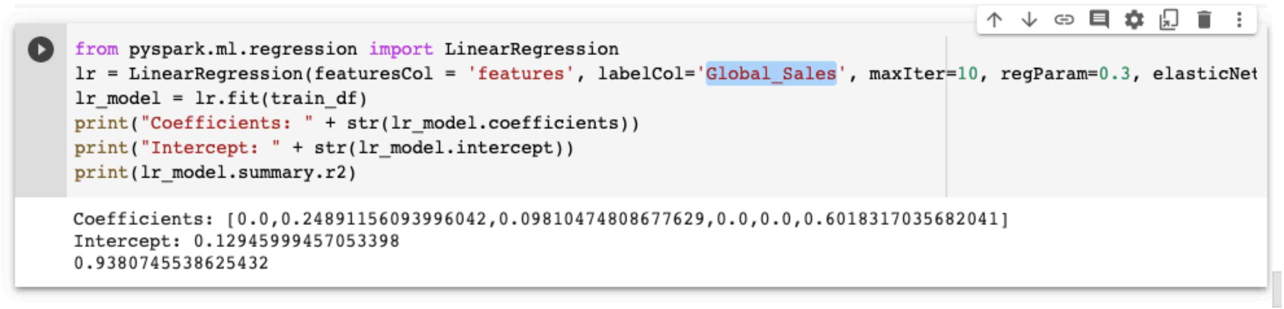


```
[17] import six
     for i in df2.columns:
       if not( isinstance(df2.select(i).take(1)[0][0], six.string_types)):
         print( "Correlation to Employees for ", i, df2.stat.corr('Global_Sales',i))
```

```
Correlation to Employees for  Rank -0.42697487174026194
Correlation to Employees for  Year -0.07464690843583835
Correlation to Employees for  NA_Sales 0.9412692156419115
Correlation to Employees for  EU_Sales 0.9032637339929322
Correlation to Employees for  JP_Sales 0.6127741711194392
Correlation to Employees for  Other_Sales 0.7479639060045417
Correlation to Employees for  Global_Sales 1.0
```

The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation; for example, the median value tends to go up when the number of rooms goes up. When the coefficient is close to –1, it means that there is a strong negative correlation; the median value tends to go down when the percentage of the lower status of the population goes up. Finally, coefficients close to zero mean that there is no linear correlation.

Linear Regression



```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol = 'features', labelCol='Global_Sales', maxIter=10, regParam=0.3, elasticNet
lr_model = lr.fit(train_df)
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
print(lr_model.summary.r2)
```

```
Coefficients: [0.0,0.24891156093996042,0.09810474808677629,0.0,0.0,0.6018317035682041]
Intercept: 0.12945999457053398
0.9380745538625432
```