databricksLinearRegression

Linear Regression

```
path = "/FileStore/tables/"
fcar = path + "carF1.csv"
dfcar1 = spark.read.csv(fcar, header=True, inferSchema=True)
dfcar1.printSchema()
```

```
root
 |-- Time: double (nullable = true)
 |-- Velocity: double (nullable = true)
```

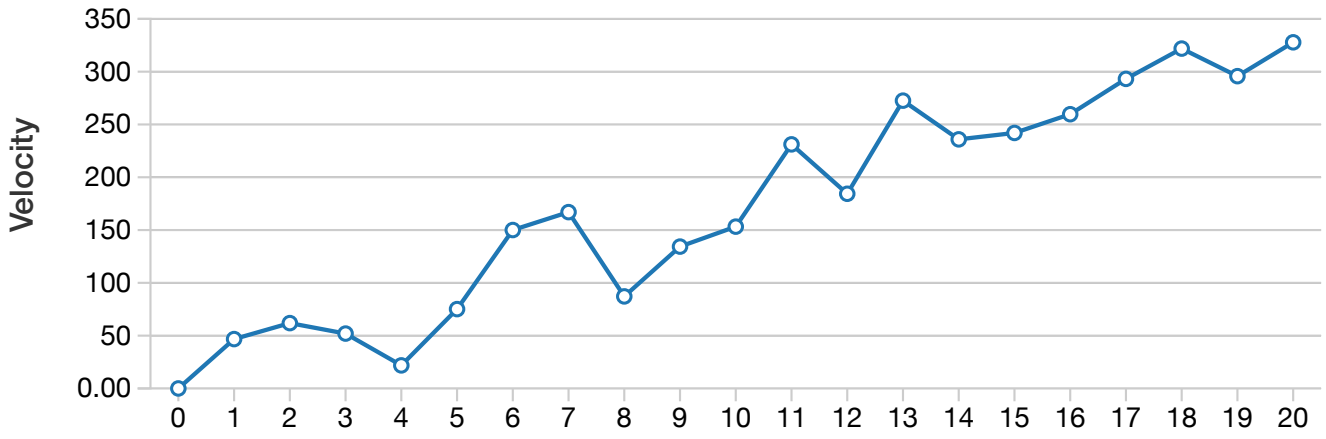
```
dfcar1.describe().show()
```

```
+-------+------------------+--------+
|summary|              Time|Velocity|
+-------+------------------+--------+
|  count|                26|      26|
|   mean|               6.5|     NaN|
| stddev|3.9772352206023727|     NaN|
|    min|               0.0|     0.0|
|    max|              13.0|     NaN|
+-------+------------------+--------+
```

```
dfcar2 = dfcar1.na.drop()
dfcar2.describe().show()
dfcar2.cache()
dfcar2.createOrReplaceTempView("tbl")
```

```
+-------+-----------------+------------------+
|summary|             Time|          Velocity|
+-------+-----------------+------------------+
|  count|               21|                21|
|   mean|              5.2|172.06951593689487|
| stddev|3.226515147957623| 104.9705738900087|
|    min|              0.0|               0.0|
|    max|             10.4|327.79294237718767|
+-------+-----------------+------------------+
```

```
%sql
select * from tbl
```



https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=vectorassembler#pyspark.ml.linalg.Vectors (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=vectorassembler#pyspark.ml.linalg.Vectors)

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
vecAssembler = VectorAssembler(inputCols=["Time"], outputCol="features")
dfcar3 = vecAssembler.transform(dfcar2)
dfcar3.printSchema()
```

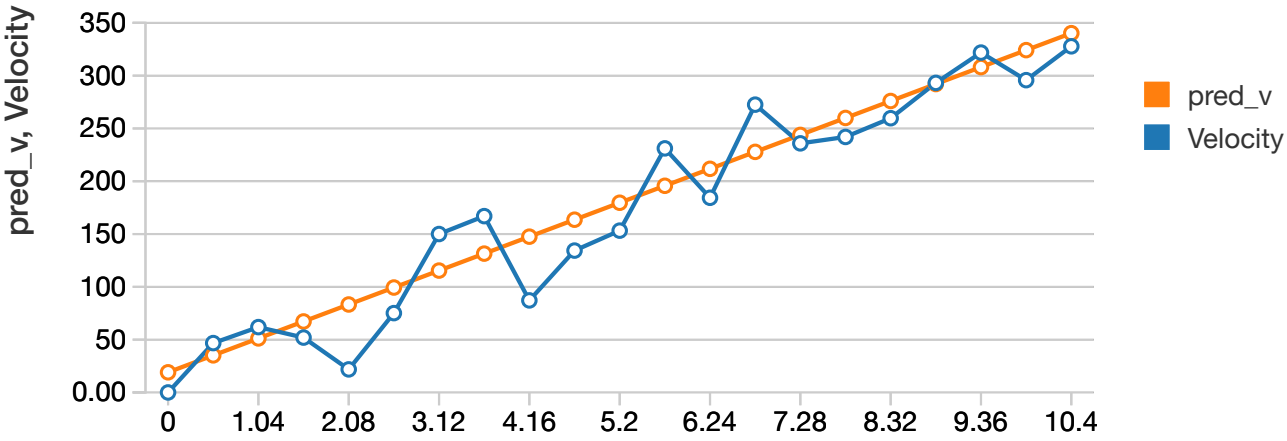
```
root
 |-- Time: double (nullable = true)
 |-- Velocity: double (nullable = true)
 |-- features: vector (nullable = true)
```

```
# splitting train and tes sets
train_car, test_car = dfcar3.randomSplit([.7, .3])
print( "train_car.count : ", train_car.count() )
print( "test_car.count : ", test_car.count() )
```

```
train_car.count :  14
test_car.count :  7
```

```
from pyspark.ml.regression import LinearRegression
reg = LinearRegression( featuresCol="features", labelCol='Velocity',
predictionCol="pred_v" )
regFit = reg.fit(train_car)


#Print the weigths
print("Coefficients: ", regFit.coefficients )
print("Intercept: ", regFit.intercept )


Coefficients:  [31.755962069]
Intercept:  -1.594047189276496

#Predict on the test data
test_pred = regFit.transform( test_car )
test_pred.show()

+-----------------+------------------+-------------------+------------------+
|             Time|          Velocity|           features|            pred_v|
+-----------------+------------------+-------------------+------------------+
|             0.52|46.743616961008996|              [0.52]| 14.91905308658963|
|             3.64|166.97840637959908|              [3.64]| 113.9976547417864|
|              5.2| 153.2382121374094|               [5.2]| 163.5369555693848|
|5.720000000000001|231.14718847938798|[5.720000000000001]|180.05005584525094|
|             6.24|184.35533494139955|              [6.24]|196.56315612111703|
|             6.76|272.55853438444336|              [6.76]|213.07625639698315|
|             7.28|235.89496286842493|              [7.28]| 229.5893566728493|
+-----------------+------------------+-------------------+------------------+


#Find R2 for Linear Regression
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol="pred_v", \
                 labelCol="Velocity",metricName="r2")
evaluator.evaluate(test_pred)

Out[39]: 0.6873956890671913

#from pyspark.ml.functions import vector_to_array #spark 3

pred_all = regFit.transform( dfcar3 )
#pred_all = pred_all.withColumn( "time", vector_to_array("features")[0] )
pred_all.createOrReplaceTempView("tbl")
```

```
%sql
select * from tbl
```




```
# Put all the transformer(s) and the estimator in a pipeline
from pyspark.ml import Pipeline
pipe = Pipeline( stages=[ vecAssembler ] )
pipeFit = pipe.fit( dfcar2 ) # make a PipelineModel
train_car2, test_car2 = pipeFit.transform(dfcar2).randomSplit( [.7, .3] )
test_pred2 = reg.fit( train_car2 ).transform( test_car2 )
test_pred2.show()
```

```
+-----------------+------------------+-------------------+------------------+
|             Time|          Velocity|           features|            pred_v|
+-----------------+------------------+-------------------+------------------+
|             3.12|150.02505567606133|              [3.12]|101.70230828175954|
|             3.64|166.97840637959908|              [3.64]|118.21267417391478|
|             4.16| 87.16670585598823|              [4.16]|   134.72304006607|
|5.720000000000001|231.14718847938798|[5.720000000000001]|184.25413774253576|
|7.800000000000001| 241.9647914273659|[7.800000000000001]| 250.2956013111567|
|             8.32| 259.7002330704192|              [8.32]|  266.805967203312|
|             8.84|  293.168306802196|              [8.84]| 283.3163330954672|
+-----------------+------------------+-------------------+------------------+
```

```
# Exercise : Predict velocities for t = 8, 10, 12, 14 seconds
new_car = spark.createDataFrame( [ [8.], [10.], [12.] ], ["Time"] )
```

```
from pyspark.ml.feature import VectorAssembler
vassem = VectorAssembler( inputCols=["Time"], outputCol="features" )
new_car2 = vassem.transform(new_car)
new_car2.show()
```

```
+----+--------+
|Time|features|
+----+--------+
| 8.0|   [8.0]|
|10.0|  [10.0]|
|12.0|  [12.0]|
+----+--------+
```

```
pred_new = regFit.transform( new_car2 )
pred_new.show()
```

```
+----+--------+------------------+
|Time|features|            pred_v|
+----+--------+------------------+
| 8.0|   [8.0]|252.45364936251008|
|10.0|  [10.0]| 315.9655735004567|
|12.0|  [12.0]|379.47749763840335|
+----+--------+------------------+
```

```
# Or use a pipeline
from pyspark.ml import Pipeline
pip = Pipeline( stages=[ vecAssembler ] )
pipFit = pip.fit( dfcar2 )
new_car2 = pipFit.transform( new_car )
pred_new = regFit.transform( new_car2 )
pred_new.show()
```

```
+----+--------+------------------+
|Time|features|            pred_v|
+----+--------+------------------+
| 8.0|   [8.0]|252.45364936251008|
|10.0|  [10.0]| 315.9655735004567|
|12.0|  [12.0]|379.47749763840335|
+----+--------+------------------+
```

```
dfcar2.unpersist()
```

```
Out[48]: DataFrame[Time: double, Velocity: double]
```

Polynomial Regression

```
path = "/FileStore/tables/"
fwind = path + "carF1_wind.csv"
df_wind = spark.read.csv( fwind, header=True, inferSchema=True )
df_wind.cache()
```

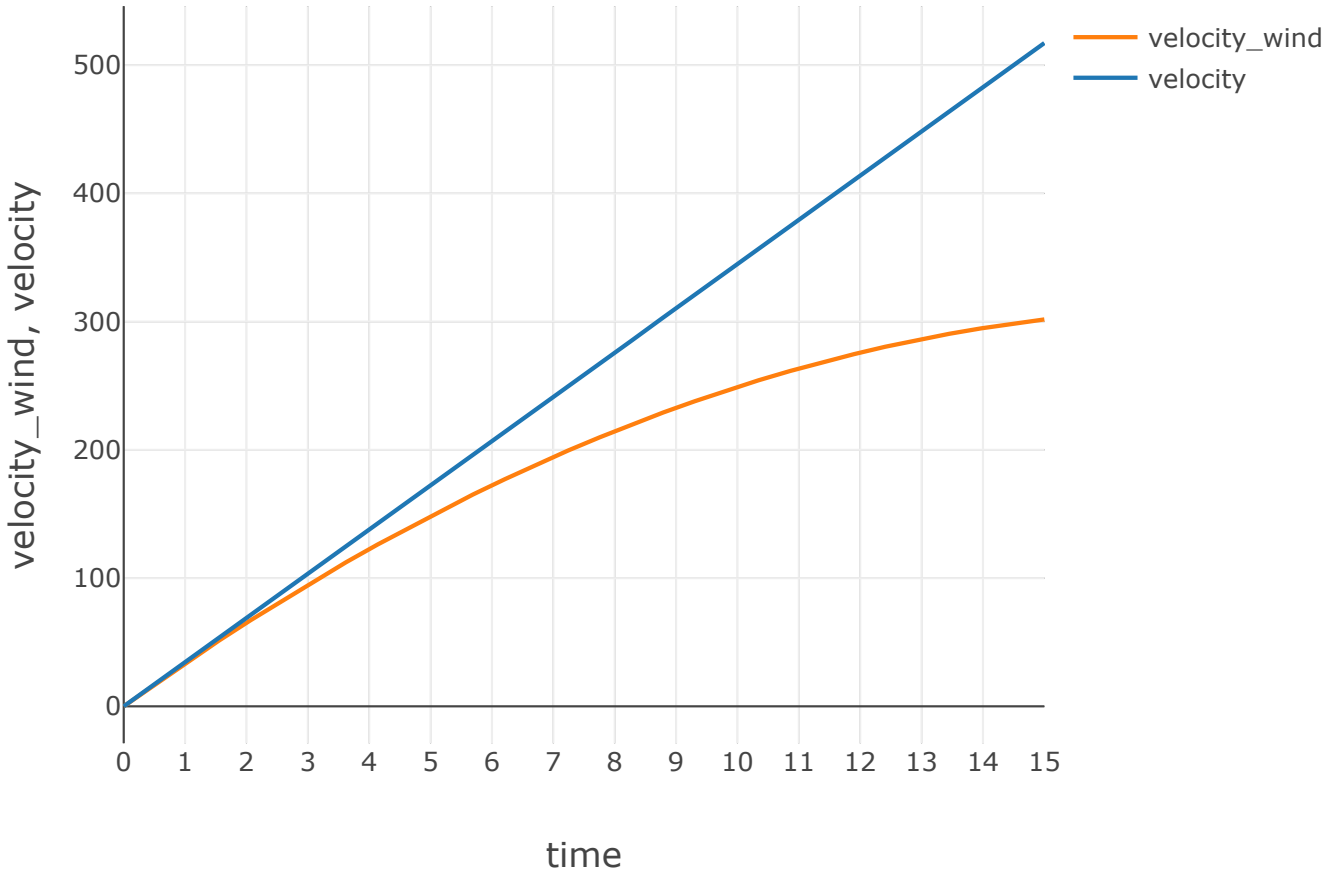
```
Out[54]: DataFrame[time: double, velocity: double, velocity_wind: double]
```

```
df_wind.describe().show()
```

```
+-------+-----------------+------------------+------------------+
|summary|             time|          velocity|     velocity_wind|
+-------+-----------------+------------------+------------------+
|  count|               30|                30|                30|
|   mean|7.499999999999999|258.62100000000004|185.54266666666666|
| stddev|4.553448863413768|157.01659396931316| 93.23750432810466|
|    min|              0.0|               0.0|               0.0|
|    max|             15.0|            517.24|            301.72|
+-------+-----------------+------------------+------------------+
```

```
df_wind.createOrReplaceTempView("tbl")
```

```
%sql
select * from tbl
```




```
#Adding the feature Vector to df_wind
from pyspark.ml.feature import VectorAssembler
vass = VectorAssembler(inputCols=["time"], outputCol="f1")
df_wind2 = vass.transform( df_wind )
df_wind2.show(5, False)
```

```
+----+--------+-------------+------+
|time|velocity|velocity_wind|f1    |
+----+--------+-------------+------+
|0.0 |0.0     |0.0          |[0.0] |
|0.52|17.84   |17.58        |[0.52]|
|1.03|35.67   |34.65        |[1.03]|
|1.55|53.51   |51.2         |[1.55]|
|2.07|71.34   |67.24        |[2.07]|
+----+--------+-------------+------+
only showing top 5 rows
```

```
# Adding the second degree time dependency
from pyspark.ml.feature import PolynomialExpansion
polex = PolynomialExpansion(degree=2, inputCol="f1", outputCol="features")
df_wind3 = polex.transform( df_wind2 )
df_wind3.cache()
df_wind3.show(5, False)
```

```
+----+--------+-------------+------+--------------------------+
|time|velocity|velocity_wind|f1    |features                  |
+----+--------+-------------+------+--------------------------+
|0.0 |0.0     |0.0          |[0.0] |[0.0,0.0]                 |
|0.52|17.84   |17.58        |[0.52]|[0.52,0.27040000000000003]|
|1.03|35.67   |34.65        |[1.03]|[1.03,1.0609]             |
|1.55|53.51   |51.2         |[1.55]|[1.55,2.4025000000000003] |
|2.07|71.34   |67.24        |[2.07]|[2.07,4.2848999999999995] |
+----+--------+-------------+------+--------------------------+
only showing top 5 rows
```

```
#Splitting to training and test samples
wind_train, wind_test =  df_wind3.randomSplit( [.7, .3] )
print( wind_train.count(), wind_test.count(), df_wind3.count()  )
```

```
20 10 30
```

```
# Fit to traininf set
from pyspark.ml.regression import LinearRegression
reg2 = LinearRegression( featuresCol="features", labelCol="velocity_wind",
predictionCol="pred_v" )
reg2Fit = reg2.fit(wind_train)
```

```
#Print the weigths
print("Coefficients: ", reg2Fit.coefficients )
print("Intercept: ", reg2Fit.intercept )
```

```
Coefficients:  [34.464493205,-0.95679052325]
Intercept:  0.05964764261353271
```

```
# Predictions for the test ssample
test_pred = reg2Fit.transform( wind_test )
test_pred.show(10, False)
```

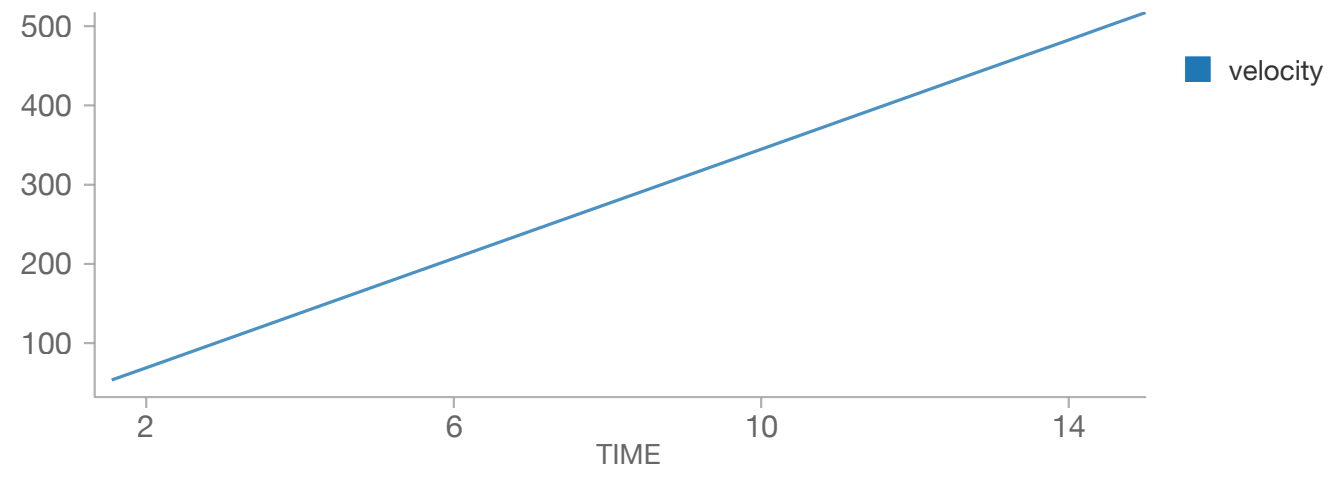
```
+----+--------+-------------+------+--------------------------+-------------------+
|time|velocity|velocity_wind|f1    |features                  |pred_v             |
+----+--------+-------------+------+--------------------------+-------------------+
```

```
|0.0 |0.0     |0.0          |[0.0] |[0.0,0.0]                  |0.05964764261353271|
|0.52|17.84   |17.58        |[0.52]|[0.52,0.27040000000000003]|17.722467951730877 |
|4.66|160.52  |139.77       |[4.66]|[4.66,21.715600000000002] |139.88690569125956 |
|5.17|178.36  |152.73       |[5.17]|[5.17,26.7289]             |152.66711929560356 |
|5.69|196.2   |165.19       |[5.69]|[5.69,32.3761]             |165.18546831930936 |
|6.72|231.87  |188.56       |[6.72]|[6.72,45.15839999999999]   |188.45391281512693 |
|8.28|285.37  |219.77       |[8.28]|[8.28,68.55839999999999]   |219.82962397088534 |
|8.79|303.21  |229.15       |[8.79]|[8.79,77.26409999999998]   |229.0769842471804  |
|9.31|321.05  |238.02       |[9.31]|[9.31,86.6761]             |237.99320830895397 |
|15.0|517.24  |301.72       |[15.0]|[15.0,225.0]               |301.7491779864448  |
+----+--------+-------------+------+---------------------------+-------------------+
```

```
# train and test fit evaluation
from pyspark.ml.evaluation import RegressionEvaluator
eval2 = RegressionEvaluator(predictionCol="pred_v", labelCol="velocity_wind",
metricName="r2")
print( " Test R^2 : ", eval2.evaluate(test_pred) )
print( " Train R^2 : ", eval2.evaluate(  reg2Fit.transform(wind_train) ) )

test_pred.createOrReplaceTempView( "tbl2" )
```

```
 Test R^2 :  0.9999992238760287
 Train R^2 :  0.999999607850485
```

```
%sql
select * from tbl2
```



```
# Exercise : put all in a pipeline
from pyspark.ml import Pipeline
pipe2 = Pipeline( stages=[ vass, polex ] )
pipe2Fit = pipe2.fit( df_wind )
wind_train2, wind_test2 =  df_wind.randomSplit( [.7, .3] )
wind_train2_R, wind_test2_R = pipe2Fit.transform( wind_train2 ),
pipe2Fit.transform( wind_test2 )
test_pred2 = reg2.fit( wind_train2_R ).transform( wind_test2_R )
eval2.evaluate(test_pred2)
```

```
Out[110]: 0.9999993934985519
```

```
df_wind.unpersist()
```

```
Out[68]: DataFrame[time: double, velocity: double, velocity_wind: double]
```

Linear Regression with Multiple Independent Variables (Features)

```
frent = path + "rentPriceNaN.csv"
df_rent = spark.read.csv( frent, header=True, inferSchema=True )
df_rent.cache()
df_rent.describe().show()
```

```
+-------+------------------+------------------+--------+------------------+
|summary|           surface|             floor|district|             price|
+-------+------------------+------------------+--------+------------------+
|  count|                27|                30|      30|                30|
|   mean|23.333333333333332|1.8666666666666667|    null|1093.4666666666667|
| stddev| 5.061164353721841|0.9371024061116424|    null|274.80321904119376|
|    min|              15.0|               1.0|       A|             526.0|
|    max|              30.0|               3.0|       B|            1581.0|
+-------+------------------+------------------+--------+------------------+
```

```
df_rent.show(30)
```

```
+-------+-----+--------+------+
|surface|floor|district| price|
+-------+-----+--------+------+
|   30.0|  1.0|       A|1553.0|
|   15.0|  3.0|       B| 526.0|
|   20.0|  3.0|       A| 834.0|
|   null|  3.0|       A| 824.0|
```

```
|   27.0|  1.0|       A|1416.0|
|   28.0|  3.0|       B|1184.0|
|   17.0|  3.0|       A| 712.0|
|   30.0|  1.0|       A|1581.0|
|   30.0|  3.0|       A|1371.0|
|   22.0|  1.0|       A|1139.0|
|   16.0|  2.0|       A| 759.0|
|   19.0|  1.0|       A|1000.0|
|   29.0|  1.0|       B|1396.0|
|   29.0|  3.0|       A|1344.0|
|   19.0|  1.0|       B| 866.0|
|   23.0|  1.0|       B|1097.0|
|   26.0|  3.0|       B|1061.0|
|   null|  1.0|       B|1129.0|
```

```
# Replacing the Null surfaces by the mean value
from pyspark.ml.feature import Imputer
imputer = Imputer(strategy='mean', missingValue=None, inputCols=["surface"],
outputCols=["surfaceR"])
impFit = imputer.fit(df_rent)
print( "average surface : \n " )
impFit.surrogateDF.show()

df_rent2 = impFit.transform( df_rent )
```

```
average surface :

+------------------+
|           surface|
+------------------+
|23.333333333333332|
+------------------+
```

```
df_rent2.describe().show()
```

```
+-------+------------------+------------------+--------+------------------+--------
----------+
|summary|           surface|             floor|district|             price|
surfaceR|
+-------+------------------+------------------+--------+------------------+--------
----------+
|  count|                27|                30|      30|                30|
30|
|   mean|23.333333333333332|1.8666666666666667|    null|1093.4666666666667|23.33333
3333333332|
| stddev| 5.061164353721841|0.9371024061116424|    null|274.80321904119376| 4.79223
5098717435|
```

```
|    min|              15.0|                1.0|       A|             526.0|
15.0|
|    max|              30.0|                3.0|       B|            1581.0|
30.0|
+-------+------------------+------------------+--------+------------------+--------
----------+
```

Out[76]: DataFrame[surface: double, floor: double, district: string, price: double, surfaceR: double]

```
from pyspark.ml.feature import StringIndexer # category to number
sindex = StringIndexer(inputCol="district", outputCol="districtNum" )
sindexFit = sindex.fit( df_rent2 )
df_rent3 = sindexFit.transform( df_rent2 )
df_rent3.show(5, False)
```

```
+-------+-----+--------+------+------------------+-----------+
|surface|floor|district|price |surfaceR          |districtNum|
+-------+-----+--------+------+------------------+-----------+
|30.0   |1.0  |A       |1553.0|30.0              |0.0        |
|15.0   |3.0  |B       |526.0 |15.0              |1.0        |
|20.0   |3.0  |A       |834.0 |20.0              |0.0        |
|null   |3.0  |A       |824.0 |23.333333333333332|0.0        |
|27.0   |1.0  |A       |1416.0|27.0              |0.0        |
+-------+-----+--------+------+------------------+-----------+
only showing top 5 rows
```

```
# OneHotEncode the categorical feature
from pyspark.ml.feature import OneHotEncoderEstimator # number to dummy variable
hot = OneHotEncoderEstimator( inputCols=["districtNum"], outputCols=["districtR"],
dropLast=True )
hotFit = hot.fit( df_rent3 )
df_rent4 = hotFit.transform( df_rent3 )
df_rent4.show(10, False)
```

```
+-------+-----+--------+------+------------------+-----------+-------------+
|surface|floor|district|price |surfaceR          |districtNum|districtR    |
+-------+-----+--------+------+------------------+-----------+-------------+
|30.0   |1.0  |A       |1553.0|30.0              |0.0        |(1,[0],[1.0])|
|15.0   |3.0  |B       |526.0 |15.0              |1.0        |(1,[],[])    |
|20.0   |3.0  |A       |834.0 |20.0              |0.0        |(1,[0],[1.0])|
|null   |3.0  |A       |824.0 |23.333333333333332|0.0        |(1,[0],[1.0])|
|27.0   |1.0  |A       |1416.0|27.0              |0.0        |(1,[0],[1.0])|
|28.0   |3.0  |B       |1184.0|28.0              |1.0        |(1,[],[])    |
|17.0   |3.0  |A       |712.0 |17.0              |0.0        |(1,[0],[1.0])|
|30.0   |1.0  |A       |1581.0|30.0              |0.0        |(1,[0],[1.0])|
|30.0   |3.0  |A       |1371.0|30.0              |0.0        |(1,[0],[1.0])|
```

```
|22.0   |1.0  |A       |1139.0|22.0               |0.0        |(1,[0],[1.0])|
+-------+-----+--------+------+-------------------+-----------+-------------+
only showing top 10 rows


df_rent4.printSchema()

root
 |-- surface: double (nullable = true)
 |-- floor: double (nullable = true)
 |-- district: string (nullable = true)
 |-- price: double (nullable = true)
 |-- surfaceR: double (nullable = true)
 |-- districtNum: double (nullable = false)
 |-- districtR: vector (nullable = true)


df_rent4.take(2)

Out[82]: [Row(surface=30.0, floor=1.0, district='A', price=1553.0, surfaceR=30.0, d
istrictNum=0.0, districtR=SparseVector(1, {0: 1.0})),
 Row(surface=15.0, floor=3.0, district='B', price=526.0, surfaceR=15.0, districtNum
=1.0, districtR=SparseVector(1, {}))]

# Definig the features vector
from pyspark.ml.feature import VectorAssembler
vessam = VectorAssembler( inputCols=[ "surfaceR", "floor", "districtR" ],
outputCol="features" )
df_rent5 = vessam.transform( df_rent4 )
df_rent5.show( 5, False )

+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+
|surface|floor|district|price |surfaceR          |districtNum|districtR    |feature
s                     |
+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+
|30.0   |1.0  |A       |1553.0|30.0              |0.0        |(1,[0],[1.0])|[30.0,1
.0,1.0]               |
|15.0   |3.0  |B       |526.0 |15.0              |1.0        |(1,[],[])    |[15.0,3
.0,0.0]               |
|20.0   |3.0  |A       |834.0 |20.0              |0.0        |(1,[0],[1.0])|[20.0,3
.0,1.0]               |
|null   |3.0  |A       |824.0 |23.333333333333332|0.0        |(1,[0],[1.0])|[23.333
333333333332,3.0,1.0]|
|27.0   |1.0  |A       |1416.0|27.0              |0.0        |(1,[0],[1.0])|[27.0,1
.0,1.0]               |
```

```
+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+
only showing top 5 rows


#Splitting to train and test samples
rent_train, rent_test = df_rent5.randomSplit( [.7, .3] )


# Fit
from pyspark.ml.regression import LinearRegression
reg3 = LinearRegression( featuresCol="features", labelCol="price",
predictionCol="pred_price" )
reg3Fit = reg3.fit( rent_train )


# Parameters
print (reg3Fit.coefficients)
reg3Fit.intercept

[50.9845690896,-102.090471279,93.814725115]
Out[95]: 30.41848502408369

# test prediction
test_pred = reg3Fit.transform( rent_test )
test_pred.show(5, False)

+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+------------------+
|surface|floor|district|price |surfaceR          |districtNum|districtR    |feature
s                     |pred_price        |
+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+------------------+
|null   |3.0  |A       |1037.0|23.333333333333332|0.0        |(1,[0],[1.0])|[23.333
333333333332,3.0,1.0]|1007.6017417250798|
|20.0   |3.0  |A       |834.0 |20.0              |0.0        |(1,[0],[1.0])|[20.0,3
.0,1.0]               |837.653178093115  |
|27.0   |1.0  |A       |1416.0|27.0              |0.0        |(1,[0],[1.0])|[27.0,1
.0,1.0]               |1398.726104278726 |
+-------+-----+--------+------+------------------+-----------+-------------+-------
----------------------+------------------+
```

```
# Test evaluation
from pyspark.ml.evaluation import RegressionEvaluator
eval3 = RegressionEvaluator( labelCol="price", predictionCol="pred_price",
metricName="r2" )
eval3.evaluate( test_pred )
```

```
Out[100]: 0.9932617503552409
```

```
# Exersice : Put all in a Pipeline
from pyspark.ml import Pipeline
pipe3 = Pipeline( stages=[ imputer, sindex, hot, vessam ]  )
pipe3Fit = pipe3.fit( df_rent )
rent_train2, rent_test2 = df_rent.randomSplit( [.7, .3] )
rent_train_R2, rent_test_R2 = pipe3Fit.transform( rent_train2 ),
pipe3Fit.transform( rent_test2 )
test_pred2 = reg3.fit( rent_train_R2 ).transform( rent_test_R2 )
eval3.evaluate( test_pred2 )
```

```
Out[106]: 0.926529777638742
```

```
# Exercise : Predict the rent price for an appartement with surface of 21.5 m^2, at
second floor located at 'A' dsitrict.
```

```
rent_feature = spark.createDataFrame( [ [ 21.5, 2., "A"] ], [ "surfaceR", "floor",
"district"] )
rent_feature.show()
rent_feature.printSchema()
```

```
+--------+-----+--------+
|surfaceR|floor|district|
+--------+-----+--------+
|    21.5|  2.0|       A|
+--------+-----+--------+
```

```
root
 |-- surfaceR: double (nullable = true)
 |-- floor: double (nullable = true)
 |-- district: string (nullable = true)
```

```
rent_feature2 = hotFit.transform( sindexFit.transform( rent_feature ) )
rent_feature2.show()
```

```
+--------+-----+--------+-----------+-------------+
|surfaceR|floor|district|districtNum|    districtR|
+--------+-----+--------+-----------+-------------+
|    21.5|  2.0|       A|        0.0|(1,[0],[1.0])|
```

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

```
rent_feature3 = vessam.transform( rent_feature2 )
rent_feature3.show()
```

```
+--------+-----+--------+-----------+-------------+--------------+
|surfaceR|floor|district|districtNum|    districtR|      features|
+--------+-----+--------+-----------+-------------+--------------+
|    21.5|  2.0|       A|        0.0|(1,[0],[1.0])|[21.5,2.0,1.0]|
+--------+-----+--------+-----------+-------------+--------------+
```

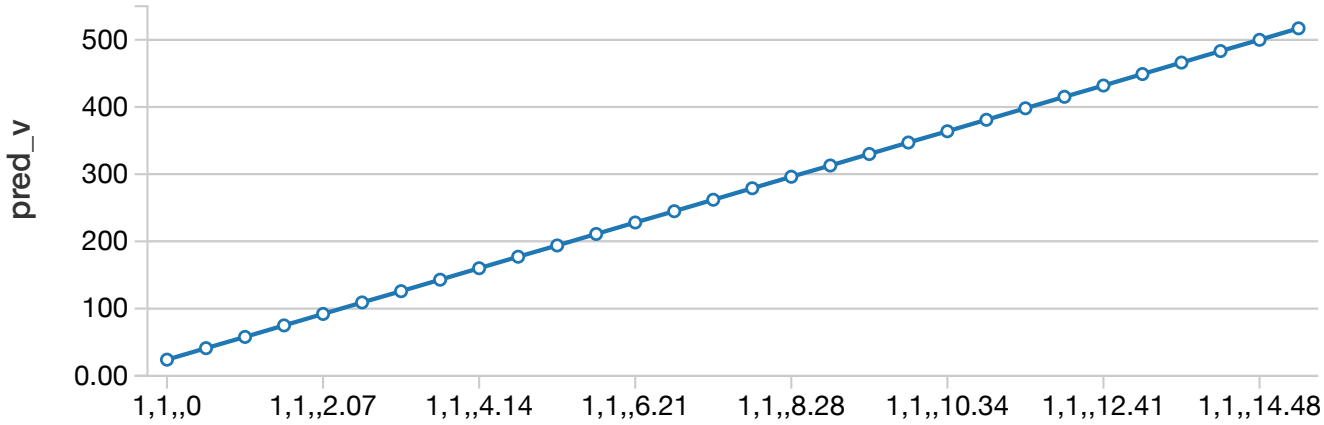
```
reg3Fit.transform( rent_feature3 ).show()
```

```
+--------+-----+--------+-----------+-------------+--------------+-----------------
-+
|surfaceR|floor|district|districtNum|    districtR|      features|        pred_pric
e|
+--------+-----+--------+-----------+-------------+--------------+-----------------
-+
|    21.5|  2.0|       A|        0.0|(1,[0],[1.0])|[21.5,2.0,1.0]|1009.111810522266
8|
+--------+-----+--------+-----------+-------------+--------------+-----------------
-+
```

```
df_rent.unpersist()
```

```
Out[104]: DataFrame[surface: double, floor: double, district: string, price: double
]
```

```
# Exercise : From data frame df_wind select only two columns time and velocity.
Replace the velocity data for 4 s < time < 6 s by the mean velocity computed from
the other lines. Plot the velocity vs. time. Make a linear fit. Plot the data and
the fitted line. What do you conclude ?

dfw1 = df_wind.drop( "velocity_wind" )

from pyspark.sql import Window
from pyspark.sql import functions as fun
win = Window.rangeBetween( Window.unboundedPreceding, Window.unboundedFollowing )
dfw2 = dfw1.withColumn( "newV",
                        fun.when( ((dfw1.time>4.) & (dfw1.time<6.)),
fun.round(fun.avg(dfw1.velocity).over(win), 2) )\
                           .otherwise(dfw1.velocity) )\
            .drop("velocity")\
            .withColumnRenamed( "newV", "velocity" )

from pyspark.ml.feature import VectorAssembler
vass = VectorAssembler(inputCols=["time"], outputCol="features")
dfw3 = vass.transform( dfw2 )

from pyspark.ml.regression import LinearRegression
reg4 = LinearRegression( featuresCol="features", labelCol="velocity",
predictionCol="pred_v" )
reg4Fit = reg4.fit( dfw3 )

from pyspark.ml.evaluation import RegressionEvaluator
regeval = RegressionEvaluator( labelCol="velocity", predictionCol="pred_v",
metricName="r2" )
print (regeval.evaluate( reg4Fit.transform(dfw3) ) )

reg4Fit.transform(dfw3).createOrReplaceTempView("tbl4")

0.95930316835405

%sql
select * from tbl4
```



Exercise : Open the file 50_Startups.csv. It comtains 5 columns : R&D Spend, Administration, Marketing Spend, State, Profit.Take the first four columns as features and fit the multiple linear regression to the profit column.

Further notes

```
from pyspark.ml.linalg import Vectors
tt = Vectors.dense( [1, 2, 3] )
tt.toArray()
```

```
Out[80]: array([ 1.,  2.,  3.])
```

```
class pyspark.ml.feature.PolynomialExpansion(degree=2, inputCol=None,
outputCol=None)
 from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([0.5, 2.0]),)], ["dense"])
>>> px = PolynomialExpansion(degree=2, inputCol="dense", outputCol="expanded")
>>> px.transform(df).head().expanded
DenseVector([0.5, 0.25, 2.0, 1.0, 4.0])
```

```
class pyspark.ml.feature.PCA(k=None, inputCol=None, outputCol=None)
from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.sparse(5, [(1, 1.0), (3, 7.0)]),),
...     (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
...     (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
>>> df = spark.createDataFrame(data,["features"])
>>> pca = PCA(k=2, inputCol="features", outputCol="pca_features")
>>> model = pca.fit(df)
>>> model.transform(df).collect()[0].pca_features
DenseVector([1.648..., -4.013...])
>>> model.explainedVariance
```

```
class pyspark.ml.feature.OneHotEncoderEstimator(inputCols=None, outputCols=None,
handleInvalid='error', dropLast=True)
from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(0.0,), (1.0,), (2.0,)], ["input"])
>>> ohe = OneHotEncoderEstimator(inputCols=["input"], outputCols=["output"])
>>> model = ohe.fit(df)
>>> model.transform(df).head().output
SparseVector(2, {0: 1.0})
```

```
class pyspark.ml.feature.MinMaxScaler(min=0.0, max=1.0, inputCol=None,
outputCol=None)
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([0.0]),), (Vectors.dense([2.0]),)],
["a"])
>>> mmScaler = MinMaxScaler(inputCol="a", outputCol="scaled")
>>> model = mmScaler.fit(df)
>>> model.originalMin
DenseVector([0.0])
>>> model.originalMax
DenseVector([2.0])
>>> model.transform(df).show()
+-----+------+
|    a|scaled|
+-----+------+
|[0.0]| [0.0]|
|[2.0]| [1.0]|
+-----+------+

class pyspark.ml.feature.MaxAbsScaler(inputCol=None, outputCol=None)
```