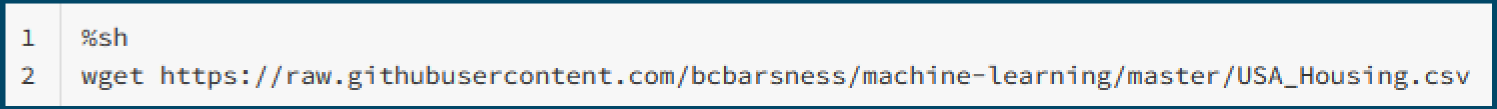
# PySpark Regression Example

For Python programmers

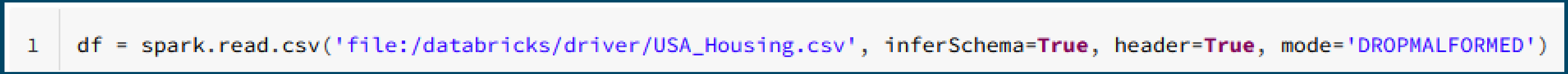
# Data process steps for regression

1. Read the data
2. Review the data
3. Split data into training & testing sets
4. Specify the regression (model / formula)
5. Fit the data to the model
6. Check the model fit

# Read data

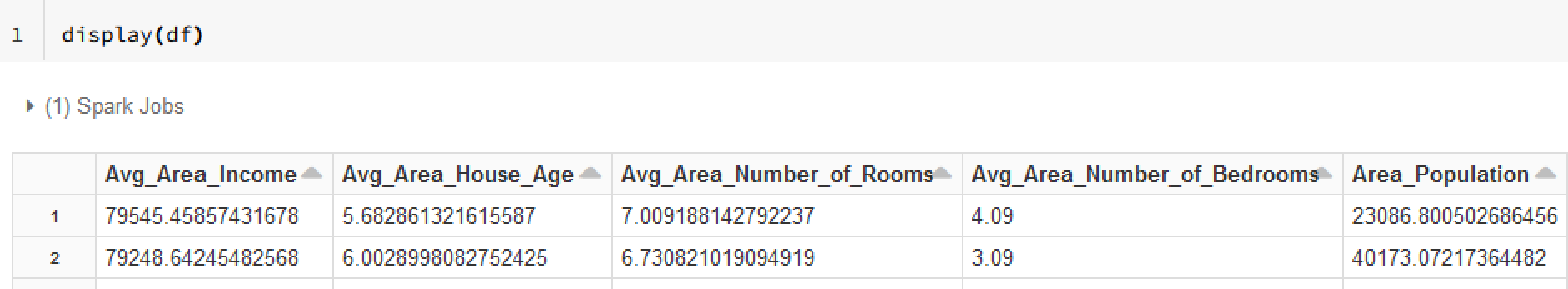


```
%sh
wget https://raw.githubusercontent.com/bcbarsness/machine-learning/master/USA_Housing.csv
```



```
df = spark.read.csv('file:/databricks/driver/USA_Housing.csv', inferSchema=True, header=True, mode='DROPMALFORMED')
```

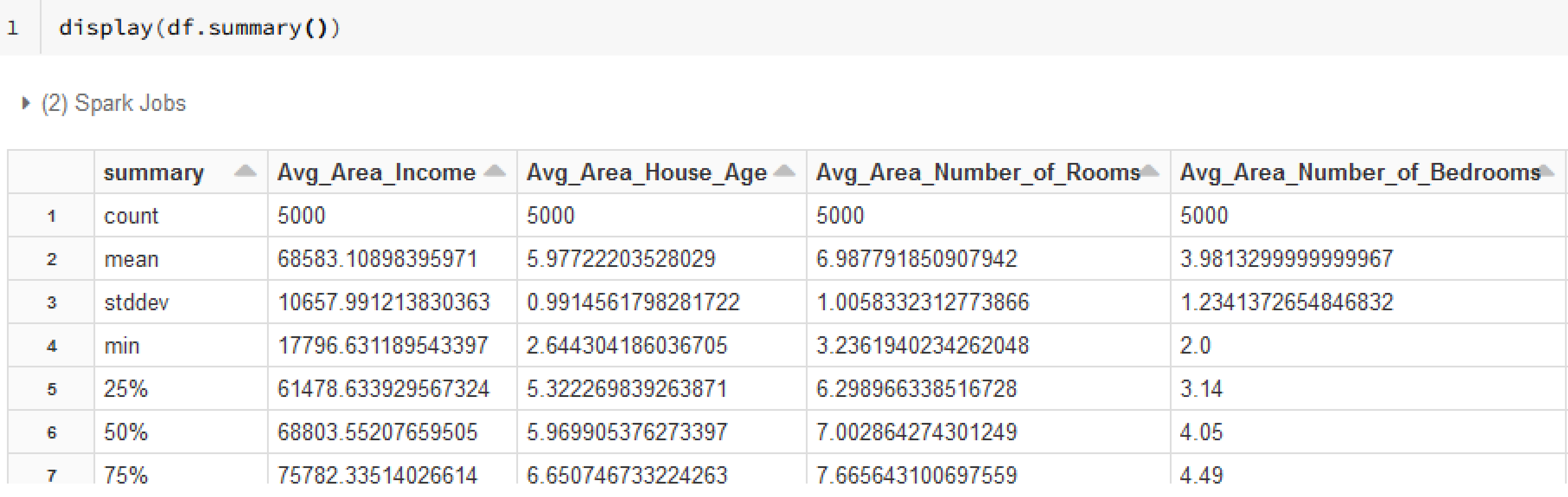
# Review data



```
display(df)
```

▸ (1) Spark Jobs

| | Avg_Area_Income | Avg_Area_House_Age | Avg_Area_Number_of_Rooms | Avg_Area_Number_of_Bedrooms | Area_Population |
|---|---|---|---|---|---|
| 1 | 79545.45857431678 | 5.682861321615587 | 7.009188142792237 | 4.09 | 23086.800502686456 |
| 2 | 79248.64245482568 | 6.0028998082752425 | 6.730821019094919 | 3.09 | 40173.07217364482 |

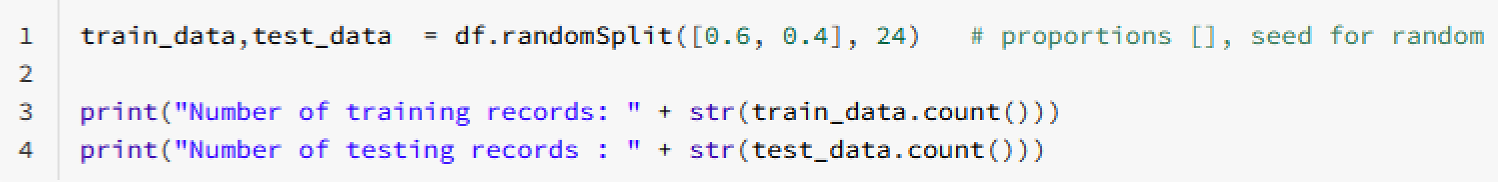


```
display(df.summary())
```

▸ (2) Spark Jobs

| | summary | Avg_Area_Income | Avg_Area_House_Age | Avg_Area_Number_of_Rooms | Avg_Area_Number_of_Bedrooms |
|---|---|---|---|---|---|
| 1 | count | 5000 | 5000 | 5000 | 5000 |
| 2 | mean | 68583.10898395971 | 5.97722203528029 | 6.987791850907942 | 3.9813299999999967 |
| 3 | stddev | 10657.991213830363 | 0.9914561798281722 | 1.0058332312773866 | 1.2341372654846832 |
| 4 | min | 17796.631189543397 | 2.644304186036705 | 3.2361940234262048 | 2.0 |
| 5 | 25% | 61478.633929567324 | 5.322269839263871 | 6.298966338516728 | 3.14 |
| 6 | 50% | 68803.55207659505 | 5.969905376273397 | 7.002864274301249 | 4.05 |
| 7 | 75% | 75782.33514026614 | 6.650746733224263 | 7.665643100697559 | 4.49 |

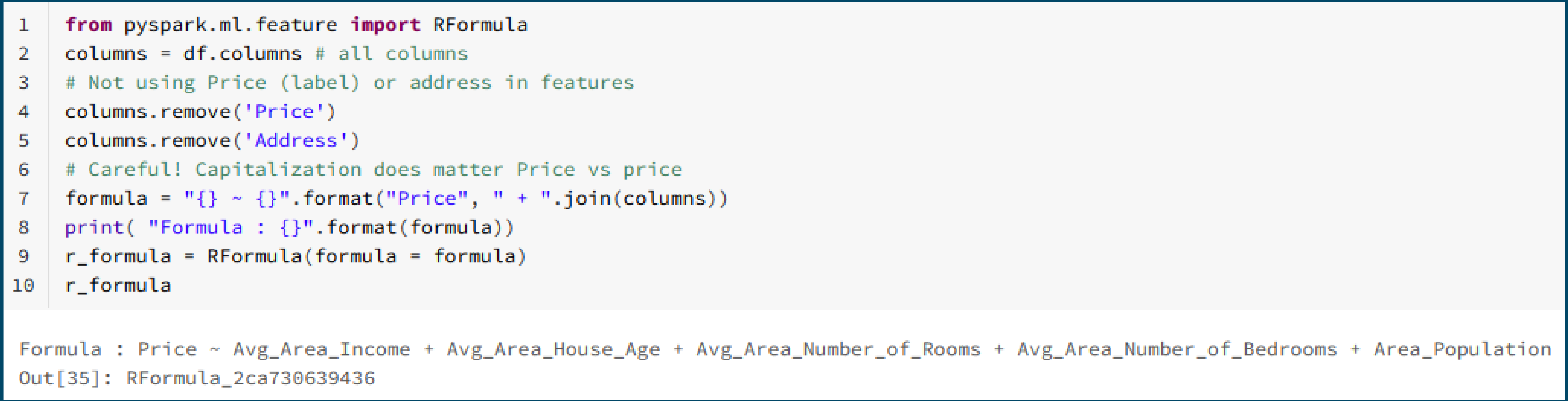
# Split data into training & testing sets



```
train_data,test_data  = df.randomSplit([0.6, 0.4], 24)   # proportions [], seed for random

print("Number of training records: " + str(train_data.count()))
print("Number of testing records : " + str(test_data.count()))
```

# Specify the regression formula



```python
from pyspark.ml.feature import RFormula
columns = df.columns # all columns
# Not using Price (label) or address in features
columns.remove('Price')
columns.remove('Address')
# Careful! Capitalization does matter Price vs price
formula = "{} ~ {}".format("Price", " + ".join(columns))
print( "Formula : {}".format(formula))
r_formula = RFormula(formula = formula)
r_formula
```

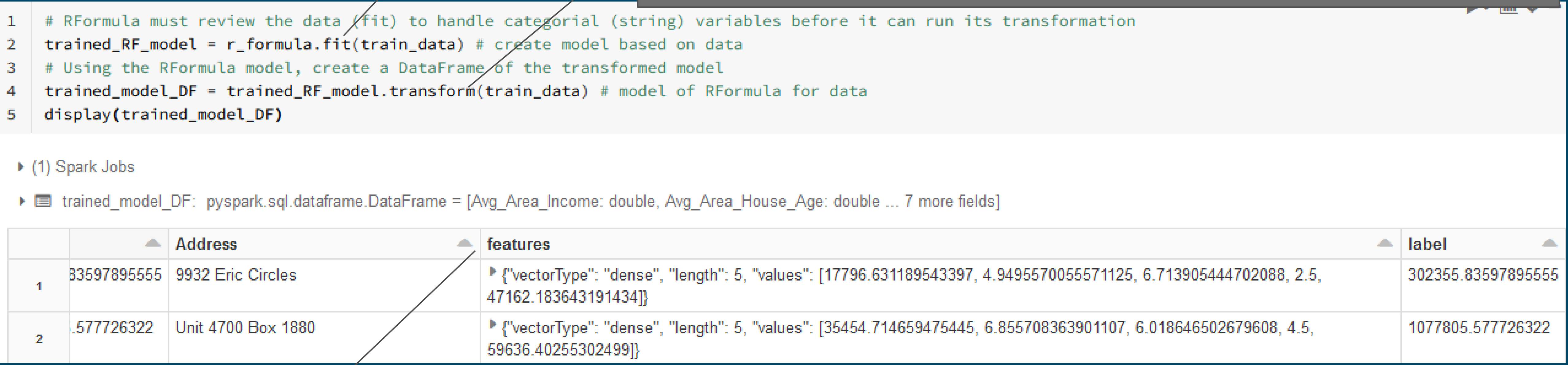
```
Formula : Price ~ Avg_Area_Income + Avg_Area_House_Age + Avg_Area_Number_of_Rooms + Avg_Area_Number_of_Bedrooms + Area_Population
Out[35]: RFormula_2ca730639436
```

# Create RFormula for data

PySpark: it’s common to fit & transform, which customizes the model to the data and then fits the model to the data

fit()
Create RFormula model given the training data.
RFormula looks at the data (column types) to create the model

transform()
Create a DataFrame containing the fitted RFormula model



```
# RFormula must review the data (fit) to handle categorial (string) variables before it can run its transformation
trained_RF_model = r_formula.fit(train_data) # create model based on data
# Using the RFormula model, create a DataFrame of the transformed model
trained_model_DF = trained_RF_model.transform(train_data) # model of RFormula for data
display(trained_model_DF)
```

▸ (1) Spark Jobs

▸ trained_model_DF: pyspark.sql.dataframe.DataFrame = [Avg_Area_Income: double, Avg_Area_House_Age: double ... 7 more fields]

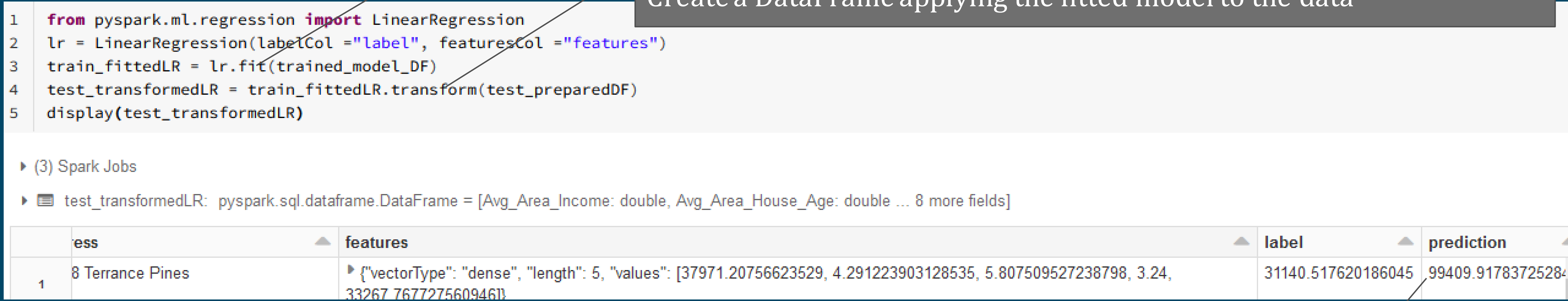
| | | Address | features | label |
|---|---|---|---|---|
| 1 | 33597895555 | 9932 Eric Circles | ▸ {"vectorType": "dense", "length": 5, "values": [17796.631189543397, 4.9495570055571125, 6.713905444702088, 2.5, 47162.183643191434]} | 302355.83597895555 |
| 2 | .577726322 | Unit 4700 Box 1880 | ▸ {"vectorType": "dense", "length": 5, "values": [35454.714659475445, 6.855708363901107, 6.018646502679608, 4.5, 59636.40255302499]} | 1077805.577726322 |

Two important columns:
features (the Xs from Python), as DenseVector of numbers, same order as data columns
label (the Y from Python)

# Apply regression with RFormula & test the model

fit()
Create a model given the data and RFormula model
LinearRegression will use the RFormula for the regression

transform()
Create a DataFrame applying the fitted model to the data

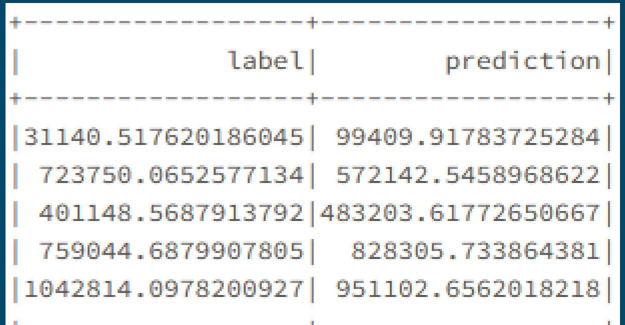


```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(labelCol ="label", featuresCol ="features")
train_fittedLR = lr.fit(trained_model_DF)
test_transformedLR = train_fittedLR.transform(test_preparedDF)
display(test_transformedLR)
```

▸ (3) Spark Jobs

▸ test_transformedLR: pyspark.sql.dataframe.DataFrame = [Avg_Area_Income: double, Avg_Area_House_Age: double ... 8 more fields]

| | ess | features | label | prediction |
|---|---|---|---|---|
| 1 | 8 Terrance Pines | ▸ {"vectorType": "dense", "length": 5, "values": [37971.20756623529, 4.291223903128535, 5.807509527238798, 3.24,<br>33267.767727560946]} | 31140.517620186045 | 99409.9178372528 |



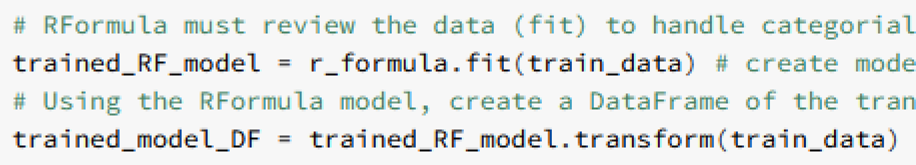
```
+------------------+------------------+
|             label|        prediction|
+------------------+------------------+
|31140.517620186045| 99409.91783725284|
| 723750.0652577134| 572142.5458968622|
| 401148.5687913792|483203.61772650667|
| 759044.6879907805|  828305.733864381|
|1042814.0978200927| 951102.6562018218|
```

Prediction column added by LinearRegression

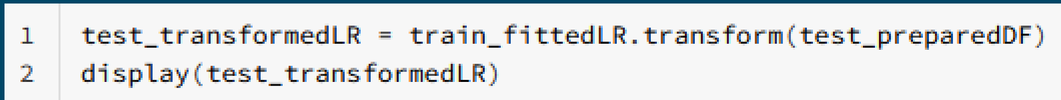
# Common data processing for modeling

## aka ML pipeline

- Begin with data
- Prepare a transformation, fit()
  - Model (estimator) customized to data
- Apply a transformation, transform()
  - Model applied to data to create DataFrame
- Prepare the next transformation, fit()
- Apply the next transformation, transform()
- …



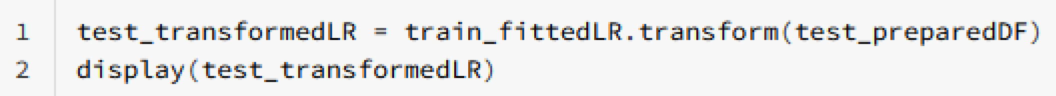
```
# RFormula must review the data (fit) to handle categorial
trained_RF_model = r_formula.fit(train_data) # create mode
# Using the RFormula model, create a DataFrame of the tran
trained_model_DF = trained_RF_model.transform(train_data)
```



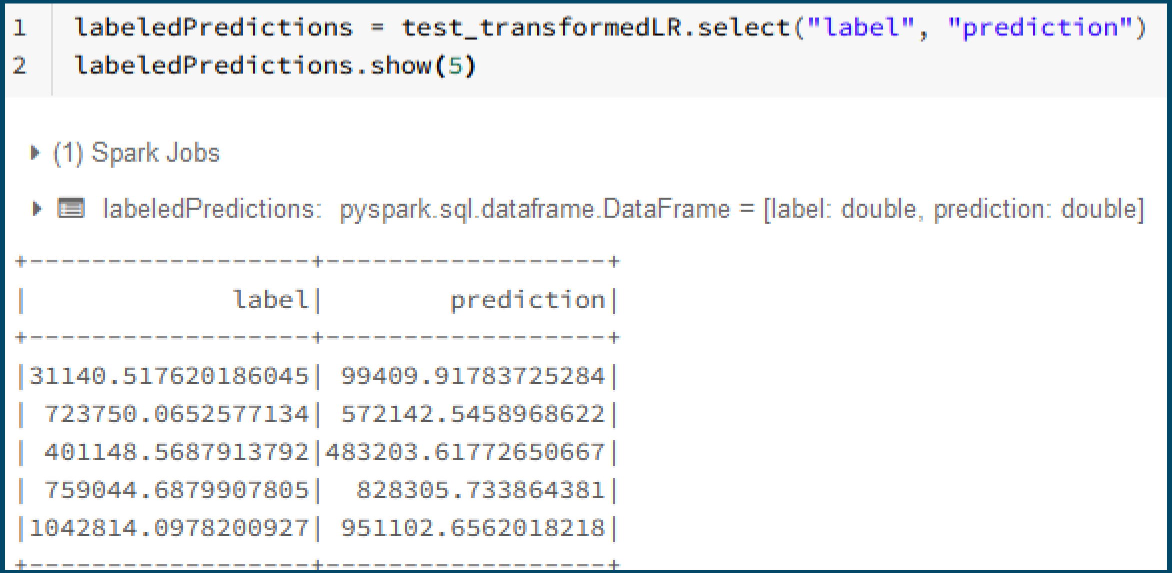
```
test_transformedLR = train_fittedLR.transform(test_preparedDF)
display(test_transformedLR)
```

# Check the model fit

- Recall, we applied the trained model to the test data



```
test_transformedLR = train_fittedLR.transform(test_preparedDF)
display(test_transformedLR)
```

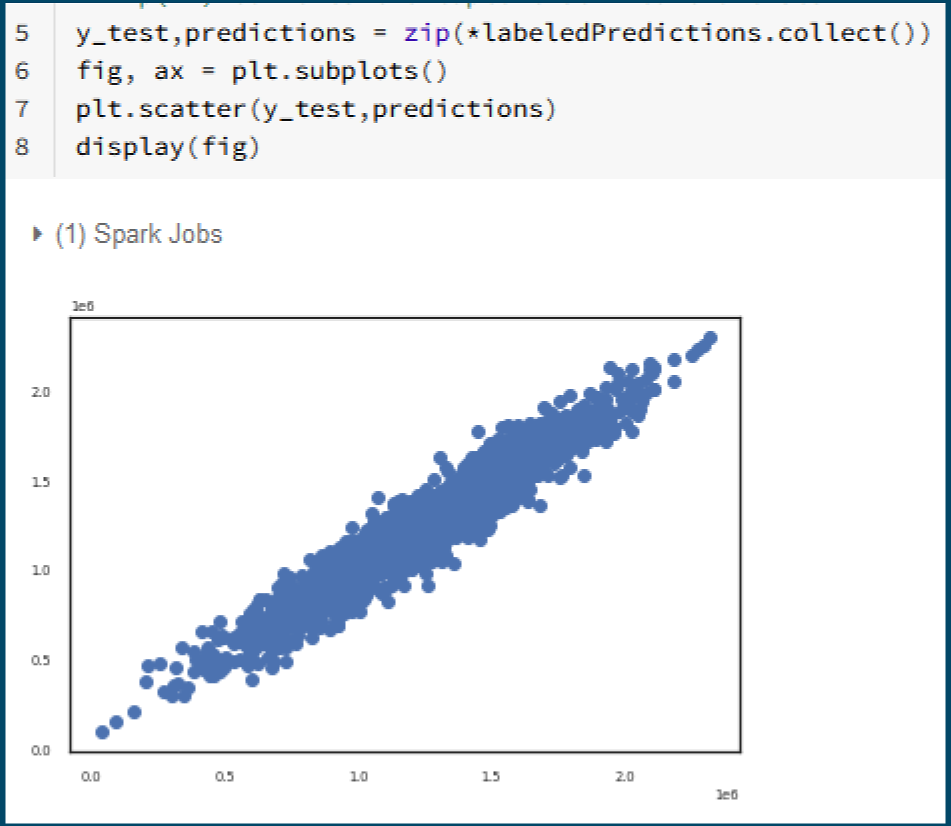


```
labeledPredictions = test_transformedLR.select("label", "prediction")
labeledPredictions.show(5)
```

(1) Spark Jobs

labeledPredictions: pyspark.sql.dataframe.DataFrame = [label: double, prediction: double]

```
+------------------+------------------+
|             label|        prediction|
+------------------+------------------+
|31140.517620186045| 99409.91783725284|
| 723750.0652577134| 572142.5458968622|
| 401148.5687913792|483203.61772650667|
| 759044.6879907805|  828305.733864381|
|1042814.0978200927| 951102.6562018218|
+------------------+------------------+
```



```
y_test,predictions = zip(*labeledPredictions.collect())
fig, ax = plt.subplots()
plt.scatter(y_test,predictions)
display(fig)
```

(1) Spark Jobs

# Important to remember

- PySpark regression is like Python regression
- RFormula must be fitted & transformed before using it in Regression
- Model parameters and data are placed into a single Features DataFrame column, using a DenseVector
- PySpark modeling assumes a pipeline
  - An estimator prepares model parameters, like RFormula
  - A transformation updates a DataFrame using a prepared model
  - Like Python’s ML pipeline