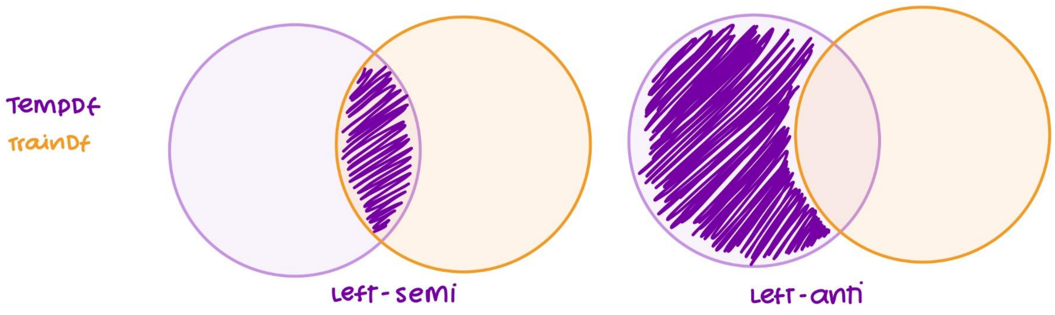
Working with Spark DataFrames - MLIB 1

1. **Which one of the following options is regarding the requirements of ML algorithms in MLib?**
    a. **All columns can be used as features**
    b. **Only the rating column can be used as feature because it’s dara type is Double**
    c. [✓ **movieId, userId, rating and timestamp can all be used as features because their data type is numeric**]
    d. **None of the above is correct**

2. **Draw the result of join operations left-semi and left-anti in a simple schema with two sets (red and yellow) with some elements in common:**



The left semi operation will return those elements from the left data frame that match in both the left and the right dataframe, whereas the left anti operation will return those elements from the left data frame that don’t match.

3. **Find the movies not rated by the previous user found. Consider that you can find all the ratings in ratingsDF. What information can we find there? What does it mean a user has not rated a particular movie?**

First we obtain the movies rated by the previous (“userId = 1”) user by filtering the ratingsDF and we store them into the variable ratings1:

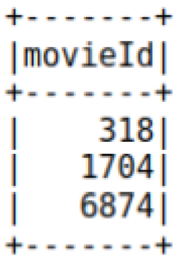
```
ratings1 = ratingsDF.where(expr(“userId = 1”)).
```

Then, we perform a left-anti join between the full ratingsDF and the ratings1 using the column movieId as the key. We will obtain a list of movie IDs that user 1 has not rated.

```
no_user1_ratings = ratingsDF.join(ratings1, [“movieId”],
            “left_anti”).select(“movieId)
```

And the output is:

We are just showing the first 3 movie IDs



```
+-------+
|movieId|
+-------+
|    318|
|   1704|
|   6874|
+-------+
```

When a user has not rated a particular movie, it means that there does not correspond any record in the ratingsDF that links the user's userId to the movie's movieId. This indicates that the user has not provided a rating for that specific movie.

4. **Show the predicted rating for the unrated movies of question 3.**

First, we use the code previously given to recommend the top movies. But instead of using just the top 5 movies, we will use the total number of movies that user 1 has not rated yet (`total_not_rated`).

```
user = usersRec.first().userId

usersRec = model.recommendForAllUsers(total_not_rated)
```

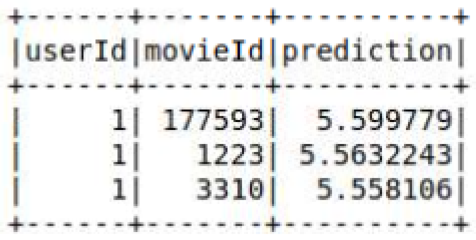
Then, we filter the recommendations for the specific user. After that, we select the relevant columns that are `movieId` and predicted `rating`. After joining this information with the movie details from `movieDF` DataFrame, we obtain a list of movie recommendations for that user, sorted by their predicted ratings in descending order.

```
movieRec1 = usersRec.filter(f"userId == {user}")

            .withColumn("recs",explode("recommendations"))

movieRec2 = movieRec1.select("userId", col("recs").movieId

            .alias("movieId"),col("recs").rating.alias("rating"))

movieRec = movieRec2.join(moviesDF, "movieId")

m = movieRec.orderBy("rating", descending=True)

            .select("movieId", "title", "rating")
```

After that, we join the **`movieRec`** DataFrame previously created with the list of movies that the user has not rated yet (**`no_user1_ratings_distinct`**). We select the **`userId`** and **`movieId`** columns and then we rename the column **`rating`** as **`prediction`**. We will obtain a DataFrame that shows the predicted ratings for the movies that the user has not rated.

```
unrated_prediction = movieRec.join(no_user1_ratings_distinct,
["movieId"], "inner").select("userId", "movieId", expr("rating as
prediction"))
```

And the output is:



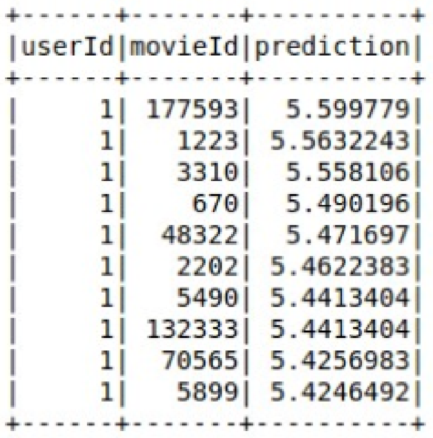
```
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|     1| 177593|  5.599779|
|     1|   1223| 5.5632243|
|     1|   3310|  5.558106|
+------+-------+----------+
```

5. **Show the top 10 movies with best predicted scores sorted by increasing values of ratings**

We create a DataFrame to store the top 10 predicted. To do that we sort the rating column in descending order to make sure that the movies with highest predicted scores appear at the top. Then we established the limit into 10 rows. We used the following code:

```
top10_predicted =
unrated_prediction.orderBy(desc(“prediction”)).limit(10)
```

And the output is:



```
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|     1| 177593|  5.599779|
|     1|   1223| 5.5632243|
|     1|   3310|  5.558106|
|     1|    670|  5.490196|
|     1|  48322|  5.471697|
|     1|   2202| 5.4622383|
|     1|   5490| 5.4413404|
|     1| 132333| 5.4413404|
|     1|  70565| 5.4256983|
|     1|   5899| 5.4246492|
+------+-------+----------+
```

Working with Spark DataFrames - MLIB 2

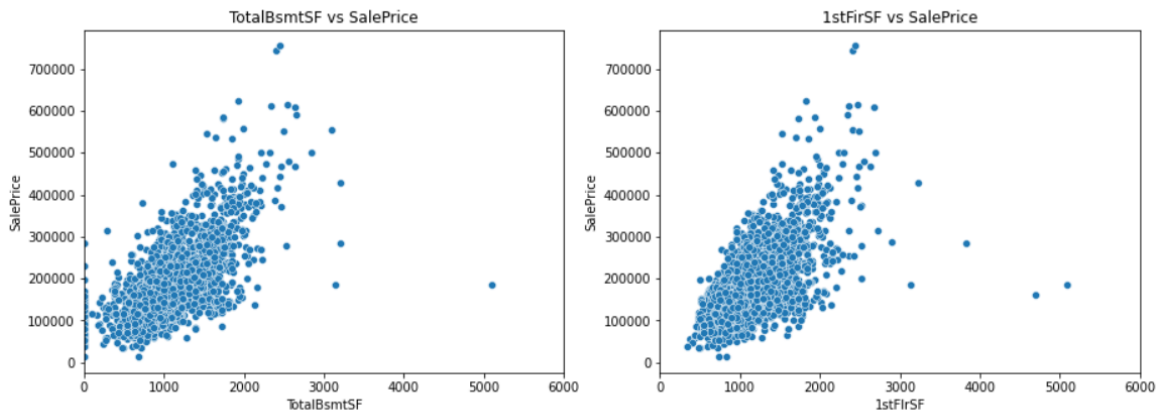
1. **Create similar scatter-plots for TotalBsmtSF vs SalePrice and 1stFlrSF vs SalePrice.**

First, we create a figure that will contain two different plots, one containing the scatter-plot for TotalBsmtSF vs SalePrice and another containing the scatter-plot for 1srFlrSF vs SalePrice.

Then we set the x axis and then we create both plots, using **`sns.scatterplot()`** command. Inside this command we specify the data that we want to use (**`pandasDF`**), we set the x and y axis and we specify the title of each of the plots.

```
fig3,   axes   =   plt.subplots(1,   2,   sharex=True,   figsize=(15,5))
axes[0].set_xlim(0,6000)

sns.scatterplot(data=pandasDF, ax=axes[0], x="TotalBsmtSF",
y="SalePrice")

axes[0].set_title("TotalBsmtSF vs SalePrice")

sns.scatterplot(data=pandasDF, ax=axes[1], x="1stFlrSF",
y="SalePrice")

axes[1].set_title("1stFirSF vs SalePrice");
```

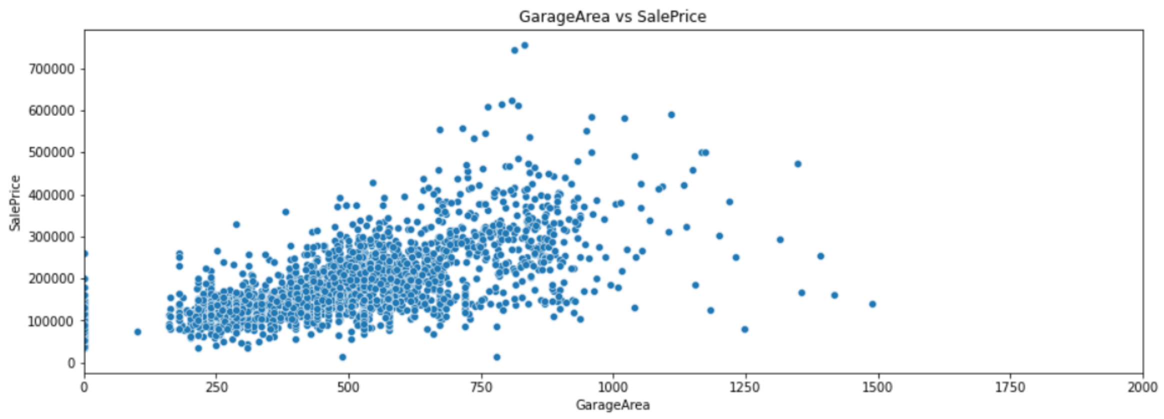
And the result is:



2. **Do you see any outliers from GarageArea vs SalePrice, can you remove them?**

First we create the plot GarageArea vs SalePrice by itself:

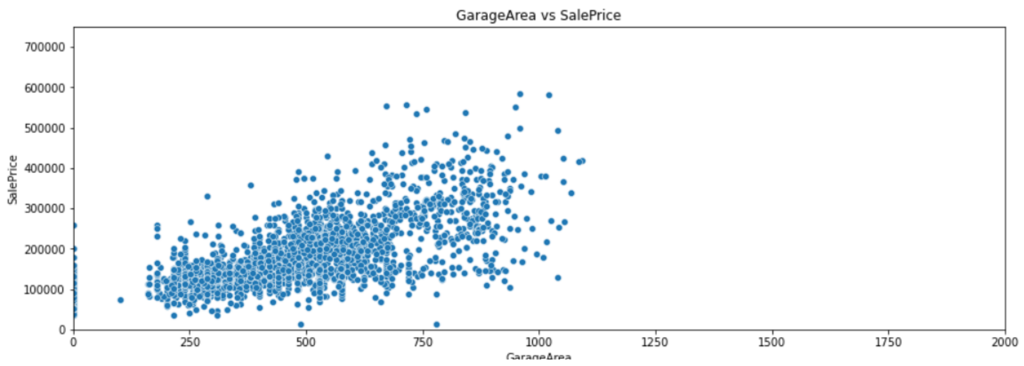
```
fig4, axes = plt.subplots(1, sharex=True, figsize=(15,5))

axes.set_xlim(0,2000)

sns.scatterplot(data=pandasDF, ax=axes, x="GarageArea",
y="SalePrice")

axes.set_title("GarageArea vs SalePrice");
```



As we can see in the previous output, we have to remove those values that are higher than 1100 in GarageArea and those above 600000 in SalePrice

```
pandasDF5 = pandasDF.loc[(pandasDF["GarageArea"] < 1100) &
(pandasDF["SalePrice"] < 600000)]

fig5, axes = plt.subplots(1, sharex=True, figsize=(15,5))

axes.set_xlim(0,2000)

axes.set_ylim(0,750000)

axes.set_title("GarageArea vs SalePrice")

sns.scatterplot(data=pandasDF5, ax=axes, x="GarageArea",
y="SalePrice")
```

And the result of the plot without the outliers is:



3.

   a. **load input airbnb data to estimate the price of a room from any combination of numerical columns like bedrooms, bathrooms, reviews, ...**

Using the code previously given for the first part of the MLIB Lab2, we create the `airbnbDF` dataframe. Then, we convert it to pandas.

```
airbnb_parquet="/home/alumno/Descargas/part-00000-tid-43204597469493
13749-5c3d407c-c844-4016-97ad-2edec446aa62-6688-1-c000.snappy.parque
t"

airbnbDF=spark.read.option("header",True).parquet(airbnb_parquet,
inferSchema=True)

pandasDF = airbnbDF.toPandas()
```

After that, we eliminate those columns with NA values from DataFrame.

```
naCols = pandasDF.columns[pandasDF.isna().any()].tolist()

curatedDF = airbnbDF[[i for i in pandasDF if i not in naCols]]
```

Then we redefine the DataFrame removing the non-string columns, leaving only the numerical ones.

```
numCols = [col for col, dtype in curatedDF.dtypes if dtype !=
'string']

indexedDF = curatedDF.select(numCols)
```

We split into training and validation separate sets of the input dataset to apply the ML methods asked.

```
(trainDF, validationDF) = indexedDF.randomSplit([0.8, 0.2], seed=1)
```

We use VectorAssembler transformer to put all of our features into a single vector.

```
vecAssembler= VectorAssembler(inputCols =
trainDF.drop("price").columns, outputCol = "features")

vecTrainDF = vecAssembler.transform(trainDF)
```

Then we use the StandardScaler function to standardise the feature columns.

```
scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures", withStd=True, withMean=False)

scalerModel = scaler.fit(vecTrainDF)

scTrainDF = scalerModel.transform(vecTrainDF)
```

b. create two models with LinearRegression and RandomForestRegressor

First we will build predictions with the linear regression model previously given but with our `airbnb` dataset.

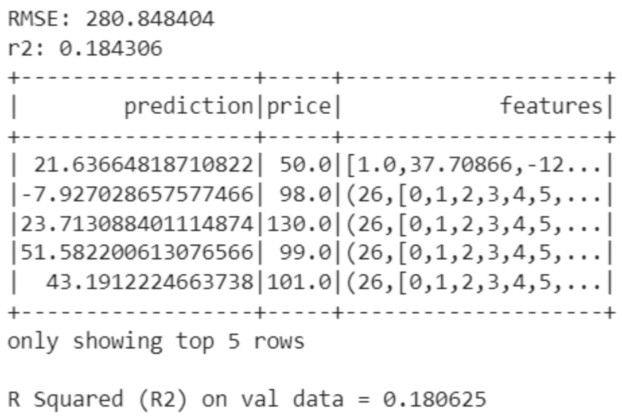
```
lr = (LinearRegression(featuresCol="features", labelCol="price",
maxIter= 50, regParam=0.001, elasticNetParam=0,))

lrModel = lr.fit(scTrainDF)
```

Here's the definition of a pipeline with the three previous steps to build a prediction model.

```
pipeline=Pipeline(stages=[vecAssembler, scaler, lr])

pipelineModel=pipeline.fit(trainDF)

predDF=pipelineModel.transform(validationDF)

predDF.select("prediction","price","features").show(5)
```

Now, we evaluate the model against the **`validationDF`**. To do it, we will use the R-squared evaluator.

```
lrEvaluator = (RegressionEvaluator(predictionCol="prediction",
labelCol="price", metricName="r2",))

r2=lrEvaluator.evaluate(predDF)

lrSummary = lrModel.summary

lrPredictions = pipelineModel.transform(validationDF)

lrPredictions.select("prediction","price","features").show(5)
```



```
RMSE: 280.848404
r2: 0.184306
+------------------+-----+--------------------+
|        prediction|price|            features|
+------------------+-----+--------------------+
| 21.63664818710822| 50.0|[1.0,37.70866,-12...|
|-7.927028657577466| 98.0|(26,[0,1,2,3,4,5,...|
|23.713088401114874|130.0|(26,[0,1,2,3,4,5,...|
|51.582200613076566| 99.0|(26,[0,1,2,3,4,5,...|
|  43.1912224663738|101.0|(26,[0,1,2,3,4,5,...|
+------------------+-----+--------------------+
only showing top 5 rows

R Squared (R2) on val data = 0.180625
```

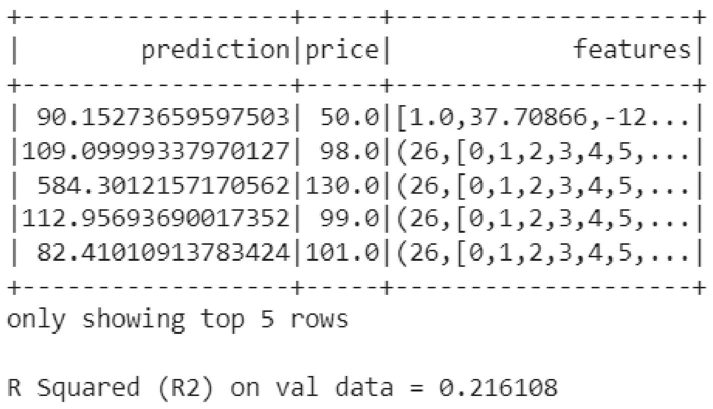
Now we'll repeat the process using the RandomForestRegressor model as a second approach.

```
rf = (RandomForestRegressor(featuresCol='features',
labelCol='price', maxDepth=10, minInstancesPerNode=2,
bootstrap=True, ))

pipeline=Pipeline(stages=[vecAssembler, rf])

pipelineModel=pipeline.fit(trainDF)

predDF=pipelineModel.transform(validationDF)

predDF.select("prediction","price","features").show(5)
```

Now, we will evaluate the quality of the prediction as we did before

```
rfEvaluator = (RegressionEvaluator(predictionCol="prediction",
labelCol="price", metricName="r2",))

rfPredictions = pipelineModel.transform(validationDF)
```

```
rfPredictions.select("prediction","price","features").show(5)
```



```
+------------------+-----+--------------------+
|        prediction|price|            features|
+------------------+-----+--------------------+
| 90.15273659597503| 50.0|[1.0,37.70866,-12...|
|109.09999337970127| 98.0|(26,[0,1,2,3,4,5,...|
| 584.3012157170562|130.0|(26,[0,1,2,3,4,5,...|
|112.95693690017352| 99.0|(26,[0,1,2,3,4,5,...|
| 82.41010913783424|101.0|(26,[0,1,2,3,4,5,...|
+------------------+-----+--------------------+
only showing top 5 rows

R Squared (R2) on val data = 0.216108
```

c. compare qualities of both methods

As we can see from the metrics observed in both models (R2 = 0.18 for LinearRegression and R2 = 0.21 for RandomForestRegressor), neither of them succeed in performing this task. Even though having tried different parameter combinations, we couldn't achieve any significant improvement. Thus, as a conclusion, we believe that these models are not the best option to perform this predictive task with the given AirBnB dataset.