Assignment_2_Section_1

November 4, 2019

MIE1628 Assignment2 ---- Ziqi Zhang 1001374684

```
In [14]: from pyspark.sql import SQLContext, Window
         from pyspark.sql.functions import *
         from pyspark.sql.types import *
         from pyspark.ml.regression import LinearRegression
         from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import VectorAssembler
         from pyspark.ml.evaluation import RegressionEvaluator
         from pyspark.sql.functions import abs, sqrt
         from pyspark import SparkConf, SparkContext
```

Q1: Use apple close price data and create two lag features, lag1 and lag2. Lag1 should push the close price back by one day and lag2 should push the price back by two days. Show your code and data in the dataframe. (Window function)

```
In [ ]: #Initiate SparkContext
        sc = SparkContext()
```

```
In [17]: #Create and filter dateframe
         sqlContext = SQLContext(sc)
         df = sqlContext.read.csv("AAPL.csv", header='true', inferSchema='true')
         df_list = ['Date', 'Close']
         df = df \
             .select([banana for banana in df.columns if banana in df_list]) \
             .filter(df['Date'] >= lit("2016-01-01 00:00:00")) \
             .filter(df['Date'] <= lit("2017-12-31 00:00:00"))

         #Apply lag
         w = Window.orderBy("Date")
         lag1 = lag('Close',1).over(w)
         lag2 = lag('Close',2).over(w)
         df = df \
             .withColumn('Lag1', lag1)\
             .withColumn('Lag2', lag2) \
             .dropna()

         df.show(10)
```

```
+-------------------+----------+----------+----------+
|               Date|     Close|      Lag1|      Lag2|
+-------------------+----------+----------+----------+
|2016-01-06 00:00:00|100.699997|102.709999|105.349998|
|2016-01-07 00:00:00| 96.449997|100.699997|102.709999|
```

```
|2016-01-08 00:00:00| 96.959999| 96.449997|100.699997|
|2016-01-11 00:00:00| 98.529999| 96.959999| 96.449997|
|2016-01-12 00:00:00| 99.959999| 98.529999| 96.959999|
|2016-01-13 00:00:00| 97.389999| 99.959999| 98.529999|
|2016-01-14 00:00:00| 99.519997| 97.389999| 99.959999|
|2016-01-15 00:00:00| 97.129997| 99.519997| 97.389999|
|2016-01-19 00:00:00| 96.660004| 97.129997| 99.519997|
|2016-01-20 00:00:00| 96.790001| 96.660004| 97.129997|
+-------------------+----------+----------+----------+
only showing top 10 rows
```

Q2: Split dataframe into train and test (0.7/0.3) and train your model use linear regression on 70% of your data and test with the other 30 percent. Show your Code.(RandomSplit)

```
In [18]: #Assemble vector
         v_df = VectorAssembler(inputCols = ['Lag1','Lag2'], outputCol = 'Lags').transform(df)
         v_df = v_df.select(['Date','Close', 'Lags'])

         #Split dataset
         train_data,test_data = v_df.randomSplit([0.7, 0.3])

         #Build ML model
         regressor = LinearRegression(featuresCol = 'Lags', labelCol='Close', maxIter=50, regParam=0.2,
         model = regressor.fit(train_data)
         test = model.evaluate(test_data)
```

Q3: Create a prediction column and show your prediction. Show your code and dataframe.

```
In [19]: #Prediction
         predictions = model.transform(test_data)
         predictions.show(10)
```

```
+-------------------+----------+--------------------+------------------+
|               Date|     Close|                Lags|        prediction|
+-------------------+----------+--------------------+------------------+
|2016-01-06 00:00:00|100.699997|[102.709999,105.3...|103.39947135016119|
|2016-01-12 00:00:00| 99.959999|[98.529999,96.959...| 98.44322080275266|
|2016-01-20 00:00:00| 96.790001|[96.660004,97.129...| 96.96017949146108|
|2016-02-03 00:00:00| 96.349998|   [94.480003,96.43]| 95.06319703377144|
|2016-02-12 00:00:00| 93.989998|[93.699997,94.269...|  94.0271052187213|
|2016-02-18 00:00:00| 96.260002|[98.120003,96.639...| 98.05120478731347|
|2016-02-24 00:00:00| 96.099998|[94.690002,96.879...|  95.3175260232073|
|2016-03-02 00:00:00|    100.75|[100.529999,96.69...| 100.0128634788106|
|2016-03-03 00:00:00|     101.5| [100.75,100.529999]|100.90969613423478|
|2016-03-04 00:00:00|103.010002|      [101.5,100.75]| 101.5584305169391|
+-------------------+----------+--------------------+------------------+
only showing top 10 rows
```

Q4: Evaluate your model with evaluation metrics (RMSE), show your code and print your result

```
In [20]: train_results = model.summary
         print("Training:")
```

```
        print("RSME: ", train_results.rootMeanSquaredError)
        print("r2: ", train_results.r2)
        print("\nTesting: ")
        print("RSME: ", test.rootMeanSquaredError)
        print("r2: ", test.r2)
```

```
Training:
RSME:  1.686669733628564
r2:  0.9958402224780161

Testing:
RSME:  1.5282749910847713
r2:  0.9961956837163369
```

Assignment_2_Section_2

November 4, 2019

In [ ]:
```
from pyspark.sql import SQLContext, Window
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark import SparkConf, SparkContext
```

Q1. Select two columns - games and seasons - and add a column with total goals (sum of home and away goals). Suggestion: use df.withColumn() function

In [ ]:
```
#Initiate SparkContext
sc = SparkContext()
sqlContext = SQLContext(sc)
```

In [129]:
```
#Create and filter dateframe
df = sqlContext.read.csv("game.csv", header='true', inferSchema='true')
df2 = df
df2 = df2.withColumn('total_goals', df2.away_goals + df2.home_goals)
df2_list = ['game_id', 'season','total_goals']
df2 = df2.select([banana for banana in df2.columns if banana in df2_list])
df2.show(10)
```

```
+----------+--------+-----------+
|   game_id|  season|total_goals|
+----------+--------+-----------+
|2011030221|20112012|          7|
|2011030222|20112012|          5|
|2011030223|20112012|          7|
|2011030224|20112012|          6|
|2011030225|20112012|          4|
|2011030411|20112012|          3|
|2011030412|20112012|          3|
|2011030413|20112012|          4|
|2011030414|20112012|          4|
|2011030415|20112012|          3|
+----------+--------+-----------+
only showing top 10 rows
```

Q2. Organize records in ascending order (by season)

In [130]:
```
df2 = df2.sort(asc("season"))
df2.show(10)
```

```
+----------+--------+-----------+
|   game_id|  season|total_goals|
+----------+--------+-----------+
|2010030311|20102011|          7|
|2010030244|20102011|          7|
|2010030312|20102011|         11|
|2010030313|20102011|          2|
|2010030314|20102011|          8|
|2010030315|20102011|          4|
|2010030316|20102011|          9|
|2010030317|20102011|          1|
|2010030241|20102011|          3|
|2010030242|20102011|          3|
+----------+--------+-----------+
only showing top 10 rows
```

Q3. Add a column with an average, min and max total score for each season.

```
In [131]: df2 = df2 \
              .withColumn('average',avg(df2['total_goals']).over(Window.partitionBy("season"))) \
              .withColumn('min',min(df2['total_goals']).over(Window.partitionBy("season"))) \
              .withColumn('max',max(df2['total_goals']).over(Window.partitionBy("season")))

          df2.show(10)
```

```
+----------+--------+-----------+-----------------+---+---+
|   game_id|  season|total_goals|          average|min|max|
+----------+--------+-----------+-----------------+---+---+
|2010030311|20102011|          7|5.586808188021228|  1| 15|
|2010030312|20102011|         11|5.586808188021228|  1| 15|
|2010030313|20102011|          2|5.586808188021228|  1| 15|
|2010030314|20102011|          8|5.586808188021228|  1| 15|
|2010030315|20102011|          4|5.586808188021228|  1| 15|
|2010030316|20102011|          9|5.586808188021228|  1| 15|
|2010030317|20102011|          1|5.586808188021228|  1| 15|
|2010030241|20102011|          3|5.586808188021228|  1| 15|
|2010030242|20102011|          3|5.586808188021228|  1| 15|
|2010030243|20102011|          7|5.586808188021228|  1| 15|
+----------+--------+-----------+-----------------+---+---+
only showing top 10 rows
```

Q4. Add a column that finds a difference between each game's total score and average for that season.n

```
In [132]: df2 = df2.withColumn('difference',df2.total_goals - df2.average)
          df2.show(10)
```

```
+----------+--------+-----------+-----------------+---+---+-------------------+
|   game_id|  season|total_goals|          average|min|max|         difference|
+----------+--------+-----------+-----------------+---+---+-------------------+
|2010030311|20102011|          7|5.586808188021228|  1| 15| 1.4131918119787716|
|2010030312|20102011|         11|5.586808188021228|  1| 15|  5.413191811978772|
```

```
|2010030313|20102011|          2|5.586808188021228|  1| 15|-3.5868081880212284|
|2010030314|20102011|          8|5.586808188021228|  1| 15| 2.4131918119787716|
|2010030315|20102011|          4|5.586808188021228|  1| 15|-1.5868081880212284|
|2010030316|20102011|          9|5.586808188021228|  1| 15| 3.4131918119787716|
|2010030317|20102011|          1|5.586808188021228|  1| 15| -4.586808188021228|
|2010030241|20102011|          3|5.586808188021228|  1| 15|-2.5868081880212284|
|2010030242|20102011|          3|5.586808188021228|  1| 15|-2.5868081880212284|
|2010030243|20102011|          7|5.586808188021228|  1| 15| 1.4131918119787716|
+----------+--------+-----------+-----------------+---+---+-------------------+
only showing top 10 rows
```

Q5. print top 10 records

```
In [69]: df2.sort(desc("difference")).limit(10).show()
```

```
+----------+--------+-----------+-----------------+---+---+------------------+
|   game_id|  season|total_goals|          average|min|max|        difference|
+----------+--------+-----------+-----------------+---+---+------------------+
|2011020128|20112012|         17|5.427051671732523|  1| 17|11.572948328267477|
|2016020661|20162017|         15|5.506454062262718|  1| 15|  9.49354593773728|
|2010020271|20102011|         15|5.586808188021228|  1| 15| 9.413191811978772|
|2018020912|20182019|         15|6.000736377025037|  1| 15| 8.999263622974963|
|2018020420|20182019|         15|6.000736377025037|  1| 15| 8.999263622974963|
|2016020434|20162017|         14|5.506454062262718|  1| 15|  8.49354593773728|
|2010020808|20102011|         14|5.586808188021228|  1| 15| 8.413191811978772|
|2012020306|20122013|         13|5.398263027295285|  1| 13| 7.601736972704715|
|2011030142|20112012|         13|5.427051671732523|  1| 17| 7.572948328267477|
|2011020398|20112012|         13|5.427051671732523|  1| 17| 7.572948328267477|
+----------+--------+-----------+-----------------+---+---+------------------+
```

Q6. List all team names (teamName) for teams that played as away team at TD Garden during seasons 2012-2013 and 2013-2014

```
In [126]: df3 = df \
              .filter("season BETWEEN '20122013' AND '20132014'") \
              .filter(df.venue == "TD Garden") \
              .drop_duplicates(subset=['away_team_id']) \
              .sort(asc('away_team_id')) \
              .select(['away_team_id'])

          ti = sqlContext.read.csv("team_info.csv", header='true', inferSchema='true')
          dif = df3.join(ti,df3["away_team_id"]==ti["team_id"],"left").select(['teamName'])

          dif.show(30)
```

```
+------------+
|    teamName|
+------------+
|      Devils|
|   Islanders|
|     Rangers|
```

```
|      Flyers|
|    Penguins|
|      Sabres|
|   Canadiens|
|    Senators|
| Maple Leafs|
|  Hurricanes|
|    Panthers|
|   Lightning|
|    Capitals|
|  Blackhawks|
|   Red Wings|
|   Predators|
|       Blues|
|      Flames|
|   Avalanche|
|      Oilers|
|     Canucks|
|       Ducks|
|       Stars|
|       Kings|
|     Coyotes|
|      Sharks|
|Blue Jackets|
|        Wild|
|        Jets|
+------------+
```

Q7. How many unique teams are on the list

```
In [127]: print(dif.count())
```

```
29
```

Assignment_2_Bonus

November 4, 2019

In [4]:
```
n = 17
```

In [5]:
```
for i in range(1,n+1):
    s = 1
    for j in range(1,i):
        if (i % j) == 0 and j != 1:
            s = 0
            break
    if s == 1:
        print(i)
```

```
1
2
3
5
7
11
13
17
```