Spark ML Challenge 3

Predicting Bike Rental

MBD O2 Spark

Team G “DataSenseis”

Camillo Baratta | Rui Cassiolato | Charlotte Leysen | Amartya Sen |
Vilhelm Stiernstedt | Hernando Suarez | Theo Tortorici

Project Definition

Description

- Bike sharing is **growing**
- Major **cities** worldwide
- Massive **data** collected
- Multiple **benefits** (traffic, congestion, etc.)
- Scenario: Washington DC

Goal

- Build a **predictive model** (# rentals per hour)
- Enhance **accuracy** (through adding variables like weather or holidays)
- **Add value** to the rental experience

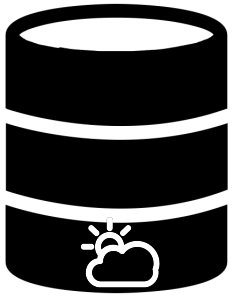
PLAN

1. Import data
2. Data Pre-processing
3. Merge dataset
4. Inspect data
5. Correlation
6. Data Engineering
7. Feature Selection
8. Modelling
9. Evaluation

Importing Data

Trip Data

```
root
 |-- duration_ms: integer (nullable = true)
 |-- start_date: timestamp (nullable = true)
 |-- end_date: timestamp (nullable = true)
 |-- start_station_nr: string (nullable = true)
 |-- start_station: string (nullable = true)
 |-- end_station_nr: string (nullable = true)
 |-- end_station: string (nullable = true)
 |-- bike_nr: string (nullable = true)
 |-- member_type: string (nullable = true)
```



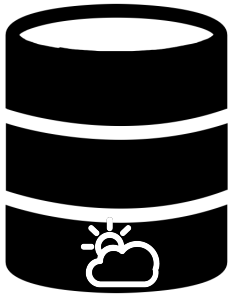
Weather Data

```
root
 |-- site4: string (nullable = true)
 |-- date: string (nullable = true)
 |-- source: string (nullable = true)
 |-- max_temp: integer (nullable = true)
 |-- avg_temp: integer (nullable = true)
 |-- hdds: double (nullable = true)
 |-- cdds: double (nullable = true)
 |-- precipitation: double (nullable = true)
 |-- snowfall: double (nullable = true)
 |-- snow_ice_depth: double (nullable = true)
 |-- random: double (nullable = true)
```

Pre-processing Data

Trip Data

1. Drop irrelevant columns.
2. Convert duration_ms to duration_min.
3. Remove +/- 2sd from duration_min.
4. Drop 'fake trips'
5. Split start_date to get start_date and start_time.
6. Split start_time to get start_hour.
7. Group data by date and hour.



Weather Data

1. Drop irrelevant columns.
2. Transform date to same format as start_date_1

Merge Datasets

```
# show columns
df_all.columns
```

```
['start_date_1',
 'start_hour',
 'count',
 'avg_duration',
 'max_temp',
 'avg_temp',
 'precipitation',
 'snowfall',
 'snow_ice_depth',
 'date']
```

63055 rows

1 Add day of the week to df_all

```
# add feature day of the week
df_all = df_all.select('start_date_1', 'start_hour', 'count', \
             'avg_duration', 'max_temp', 'avg_temp', \
             'precipitation', 'snowfall','snow_ice_depth', \
              date_format('start_date_1', 'E').alias('day_of_week'))
```

2 Transform start_hour from string to int.

```
# change start_hour form string to int
df_all = df_all.withColumn("start_hour", df_all["start_hour"].cast('int'))
```

3 Look for missing values.

```
# look for missing values
df_all.toPandas().isnull().sum()
```

```
start_date_1        0
start_hour          0
count               0
avg_duration        0
max_temp            0
avg_temp            0
precipitation       0
snowfall            0
snow_ice_depth     48
day_of_week         0
dtype: int64
```

```
# drop NaNs
df_all = df_all.na.drop()
```

```
# inspect results
df_all.toPandas().isnull().sum()
```

```
start_date_1        0
start_hour          0
count               0
avg_duration        0
max_temp            0
avg_temp            0
precipitation       0
snowfall            0
snow_ice_depth      0
day_of_week         0
dtype: int64
```

1 Examine bike parameters

```
# describe bike parameters
df_all.select('count', 'avg_duration').describe().show()
```

```
+-------+------------------+------------------+
|summary|             count|      avg_duration|
+-------+------------------+------------------+
|  count|             63007|             63007|
|   mean|244.14042884123987| 14.68291670561939|
| stddev|   266.34930793509|3.5432817577889106|
|    min|                 1|            5.0025|
|    max|              1747|           79.9471|
+-------+------------------+------------------+
```

2 Examine weather parameters

```
# describe weather paramenters
df_all.select('max_temp', 'precipitation', 'snowfall', 'snow_ice_depth').describe().show()
```

```
+-------+------------------+-----------------+-------------------+--------------------+
|summary|          max_temp|    precipitation|           snowfall|      snow_ice_depth|
+-------+------------------+-----------------+-------------------+--------------------+
|  count|             63007|            63007|              63007|               63007|
|   mean| 66.42341327154126|3.493500722141984|0.10900981160823407|0.048625073404542314|
| stddev|18.154156785046155|5.580150137498256| 0.3139070335205244| 0.47088381980781774|
|    min|                15|              0.0|                0.0|                 0.0|
|    max|               105|             25.5|               4.74|                11.7|
+-------+------------------+-----------------+-------------------+--------------------+
```

```
# create subset for numerical features
df_numerical = df_all.select('count', 'avg_duration', 'max_temp', 'start_hour', \
                             'precipitation', 'snowfall', 'snow_ice_depth')
```

3 Correlations

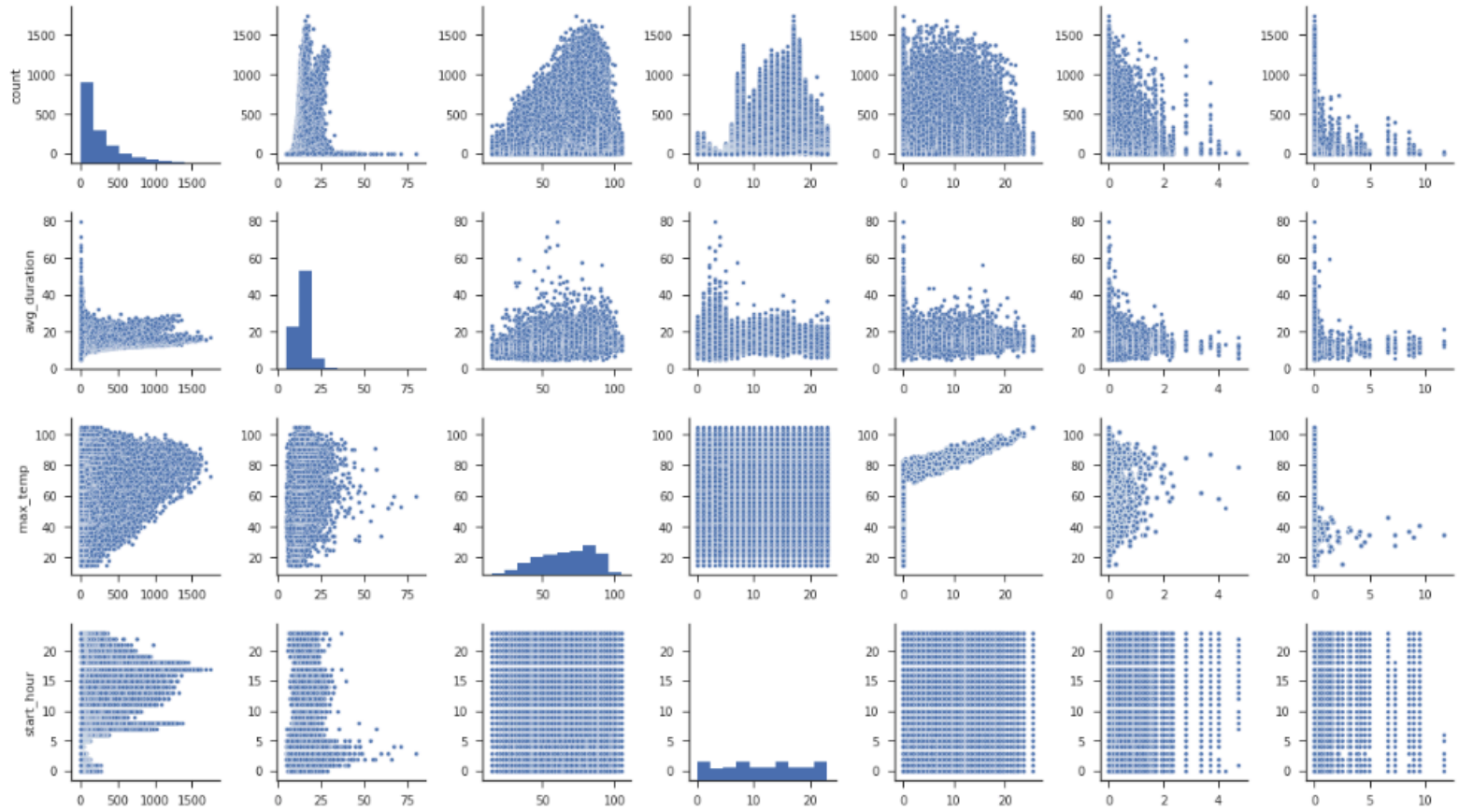
```
# function that computes the correlation of each column against the target
# Computes Pearson Correlation Coefficient between the two columns
def computeCorrelation(df, targetColumnName):
    for col in df.columns:
        r=df.stat.corr(col, targetColumnName)
        print("Pearson correlation : %s %s %f \n" %(col, targetColumnName , r))
```

```
# get correlation against bike count
computeCorrelation(df_numerical, 'count')
```

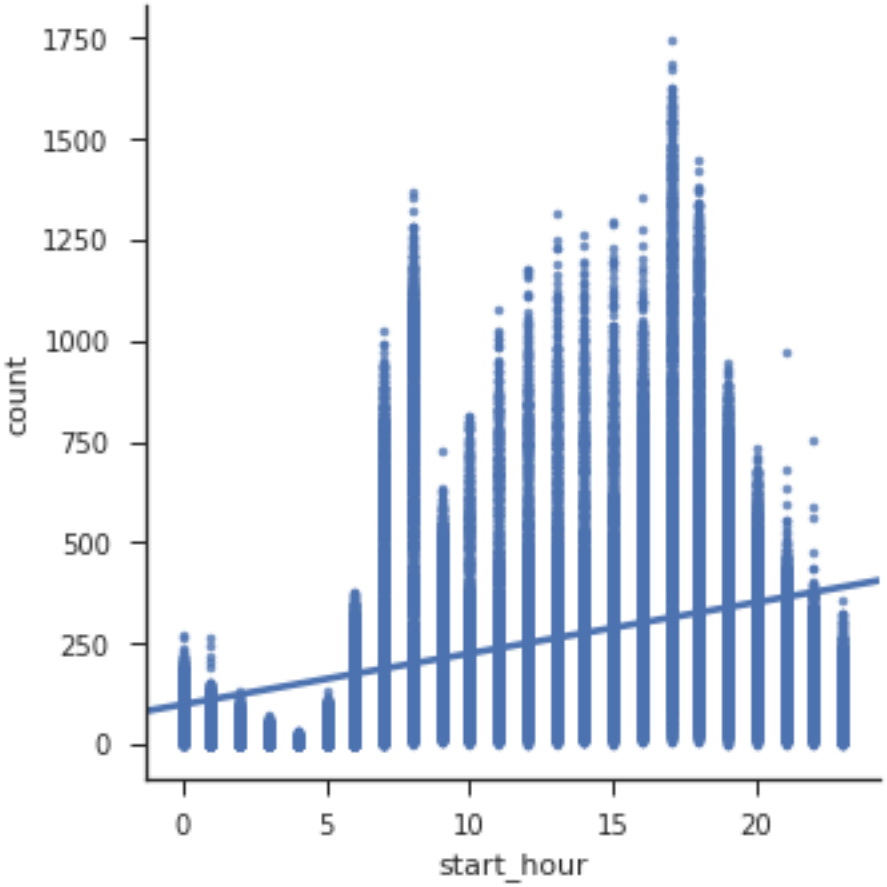
```
Pearson correlation : count count 1.000000

Pearson correlation : avg_duration count 0.330365

Pearson correlation : max_temp count 0.288379

Pearson correlation : start_hour count 0.328915

Pearson correlation : precipitation count 0.157065

Pearson correlation : snowfall count -0.086651

Pearson correlation : snow_ice_depth count -0.068109
```

4 Visualisation of correlations

```
# Visualize correlations
sns.set(style="ticks", color_codes=True )
sns.pairplot(df_numerical.toPandas(), markers=".")
plt.show()
```



Examine distribution of start_hour against count



Create the following bins

start_hour binning

- bin 1: 22:00-05:59
- bin 2: 06:00-09:59
- bin 3: 10:00-17:59
- bin 3: 18:00-21:59

binning weekday

- weekday bin: Mon-Fri
- weekend bin: Sat-Sun

binning season

- summer bin: apr-oct (04-10)
- winter bin: nov-mar (11-03)

1 Select features for regression

```
# select features for regression model (inc. target)
df_reg = df_all.select('count', 'max_temp', 'precipitation', 'snowfall', \
                       'weekday', 'weekend', 'hour_bin_1', 'hour_bin_2', 'hour_bin_3', 'hour_bin_4', \
                       'summer', 'winter')
```

2 Convert df to label and feature vectors

```
# Define the input_data
# The count (row[0]) is our target variable (the label)
# The rest of the values row[1:] our our features
data = df_reg.rdd.map(lambda row: (row[0], DenseVector(row[1:])))

# Replace df with the new DataFrame
df = spark.createDataFrame(data, ["label", "features"])
```

3 Split data

```
# Split the data into train and test sets
train_data, test_data = df.randomSplit([.8, .2],seed= 1234)
print('Training records : %d' % train_data.count())
print('Test records : %d ' % test_data.count())
```

```
Training records : 50523
Test records : 12484
```

Create model based on selected features

```
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(labelCol="label",
                      maxIter=10, elasticNetParam=0.8)

# Fit the data to the models
linearModelA = lr.fit(train_data,{lr.regParam:0.1})
linearModelB = lr.fit(train_data,{lr.regParam:0.3})
linearModelC = lr.fit(train_data,{lr.regParam:0.6})
linearModelD = lr.fit(train_data,{lr.regParam:0.9})

# Generate predictions for models
predictedA = linearModelA.transform(test_data)
predictedB = linearModelB.transform(test_data)
predictedC = linearModelC.transform(test_data)
predictedD = linearModelD.transform(test_data)
```

```
# Get the RMSE
linearModelA.summary.rootMeanSquaredError
```

```
209.82397609408608
```

```
# The R2
linearModelA.summary.r2
```

```
0.37783495603355177
```

```
linearModelB.summary.r2
```

```
0.377873379465696
```

```
linearModelC.summary.r2
```

```
0.37784068777704993
```

```
linearModelD.summary.r2
```

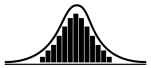
```
0.3778191906916345
```

Conclusion

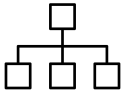
What we were able to do

We managed to **aggregate 19 million rows** into a format wherein we managed to produce a **linear regression model**. The four models show similar R2 score from where we can conclude that the **regularization parameter do not help us**. We most likely don't have an issue with overfitting.
Our **features are simply not enough**, linear speaking, to describe our target, count of bike rental per given day and hour.

Suggestions

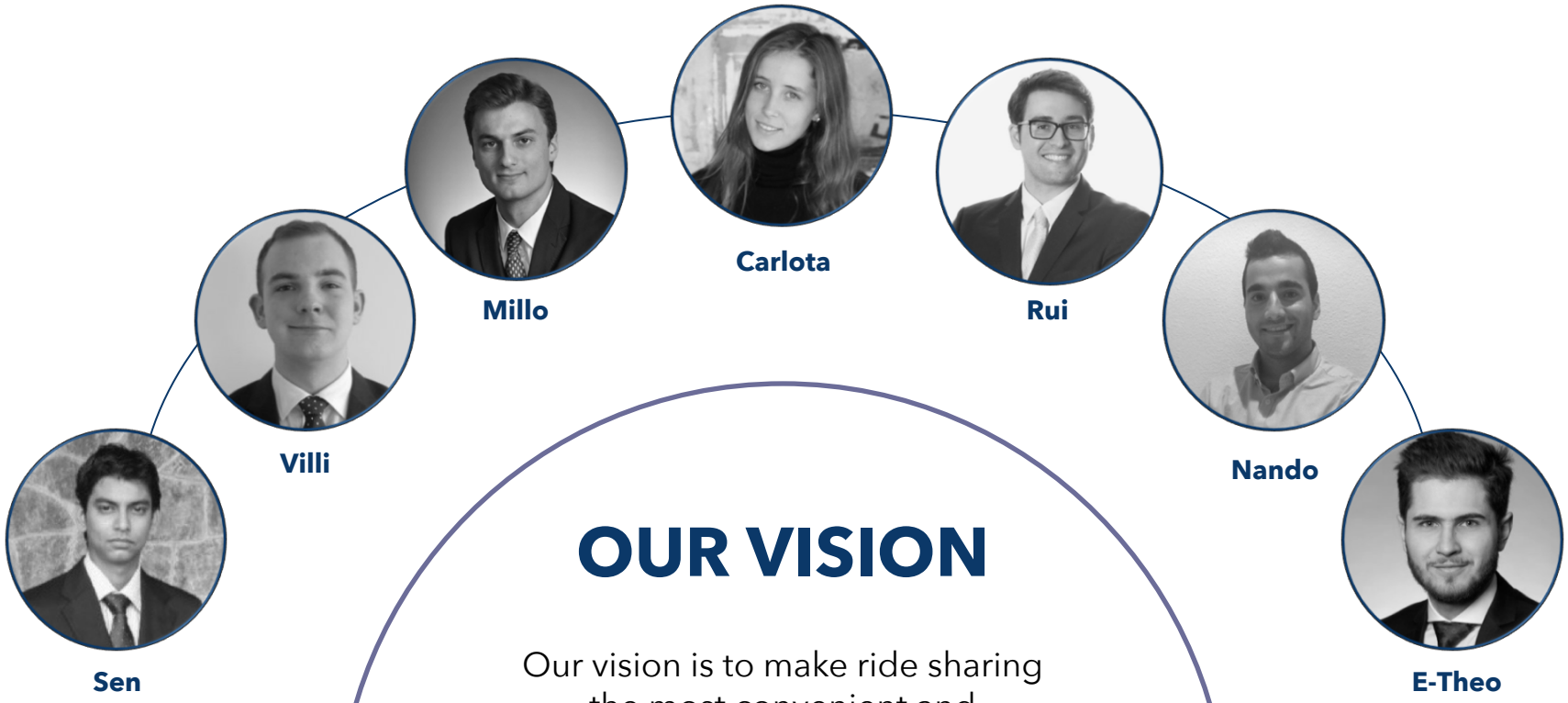


Scaling and log-transformation to **remove skewness**



Try more **complex models** i.e. RF / GBM

Introduce **more features** i.e. bank-holidays



OUR VISION

Our vision is to make ride sharing the most convenient and predictable experience to explore a city at the lowest possible cost.