This is how I compiled the codes for my project step by step

There's two big steps, data profiling with mapreduce and then data analysis in pyspark

All codes and classes for data profiling are in /profiling_code directory or in 1profiling folder in the source code zip file. Step 0 to 2 are for data profiling and MR_Job2 (clean) needs to be computed for later pyspark code to work

Pyspark codes are found in the zip file in ana_code directory called fullcode, fullcode is basically the entirety of my project and can be copy and pasted to pyspark to run the entire project in one go (if the copy paste worked properly, some errors may persist).

A big part of Fullcode is broken up to ETL_code.txt, ingest_code.txt, analysis_code.txt, and unused_code.txt all in their own directories for reasons. Ingest is step 4 for data ingestion, ETL is step 5 for data cleaning, analysis is for steps 6-8 for regression analysis.

Step 0, InputData

Gain access to the skillcraft.csv, it is in PBDproj/inputData/

hadoop fs -copyToLocal PBDproj/inputData/skillcraft.txt /home/ry2050/

Step 1, Profiling (count records)

#The count record classes are in PBDproj/profiling_code/MR_Job1/
#code below is what I used to compile the mapreduce count records

```
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job1/countR.jar /home/ry2050/
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job1/CountRecs.java
/home/ry2050/
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job1/CountRecsMapper.java
/home/ry2050/
        hadoop fs -copyToLocal
PBDproj/profiling_code/MR_Job1/CountRecsReducer.java /home/ry2050/
        hadoop fs -copyToLocal PBDproj/inputData/skillcraft.csv /home/ry2050/

javac -classpath `yarn classpath` -d . CountRecsMapper.java
javac -classpath `yarn classpath` -d . CountRecsReducer.java
javac -classpath `yarn classpath`:. -d . CountRecs.java
jar -cvf countR.jar *.class

hdfs dfs -rm -R -skipTrash output1
hadoop jar countR.jar CountRecs skillcraft.csv output1
hdfs dfs -cat output1/part-r-00000
```

Step 2, Profiling (clean)

#The clean classes are in PBDproj/profiling_code/MR_Job2/

#code below is what I used to compile the mapreduce clean, then save the results as skillclean.txt in HDFS

```
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job2/CleanC.jar /home/ry2050/
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job2/Clean.java /home/ry2050/
hadoop fs -copyToLocal PBDproj/profiling_code/MR_Job2/CleanMapper.java
/home/ry2050/
hadoop fs -copyToLocal PBDproj/inputData/skillcraft.csv /home/ry2050/

javac -classpath `yarn classpath` -d . CleanMapper.java
javac -classpath `yarn classpath`:. -d . Clean.java
jar -cvf CleanC.jar *.class

hdfs dfs -rm -R -skipTrash output1
hadoop jar CleanR.jar Clean skillcraft.csv output1
hdfs dfs -cat output1/part-r-00000
touch skillclean.txt
hdfs dfs -cat output1/part-r-00000 >/home/ry2050/skillclean.txt
hdfs dfs -put skillclean.txt PBDproj/inputData/
hdfs dfs -put skillclean.txt PBDproj/profiling_code/output/
```

Step 3, Pyspark setup

#update python to python 3

#set pyspark to use python 3, else some mlib libery can't be imported

```
export PYSPARK_PYTHON=python3
```

#open pyspark

```
pyspark --deploy-mode client
```

#code to import packages sql and mlib

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType
from pyspark.sql.functions import *
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import col, size
from pyspark.ml.stat import Correlation
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.stat import ChiSquareTest
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import Window
from pyspark.sql.functions import *
from pyspark.sql import functions as f
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator
```

```
bevaluator = BinaryClassificationEvaluator()
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
```

Step 4, Data Ingestion (ingest_code)

#code used to define schema

```
schema = StructType([\
    StructField("GameID", IntegerType()),\
    StructField("LeagueIndex", IntegerType()), \
    StructField("Age", IntegerType()),\
    StructField("Hours_Per_Week", IntegerType()),\
    StructField("TotalHours", IntegerType()),\
    StructField("APM", DoubleType()),\
    StructField("SelectByHotkeys", DoubleType()),\
    StructField("AssignToHotkeys", DoubleType()),\
    StructField("UniqueHotkeys", IntegerType()),\
    StructField("MinimapAttacks", DoubleType(), False),\
        StructField("MinimapRightClicks", DoubleType(), False),\
        StructField("NumberOfPACs", DoubleType()),\
        StructField("GapBetweenPACs", DoubleType()),\
        StructField("ActionLatency", DoubleType()),\
        StructField("ActionsInPAC", DoubleType()),\
        StructField("TotalMapExplored", IntegerType()),\
        StructField("WorkersMade", DoubleType()),\
        StructField("UniqueUnitsMade", IntegerType()),\
        StructField("ComplexUnitsMade", DoubleType(), False),\
        StructField("ComplexAbilitiesUsed", DoubleType(), False)
  ])
```

#read dataset with defined schema

```
dfn = spark.read.csv("skillcraft.csv", header=True, sep=",", schema=schema)
dfn.cache()
dfn.printSchema()
dfn.show(truncate=False)
```

#write new dataframe to hdfs for future use (ps: i never used it)

```
dfn.write.save('PBDproj/inputData', format='parquet', mode='append')
```

Step 5, ETL_code (cleaning)

#code below drops unwanted column game Id and total hours

```
skillcraft = dfn.dropna().drop('GameID').drop('TotalHours')
skillcraft.cache()
skillcraft.printSchema()
skillcraft.show(truncate=False)
```

```
#code below finds and prints correlations between attributes and target variable
    for i in skillcraft.columns:
      c = skillcraft.stat.corr('LeagueIndex',i)
      print( "Correlation to Rank for:", i, " ", c)

#code below is a function named remove outliers from dataframe, only datapoints
outside of quantile 0.1 and 0.9 are counted as outliers
    def remove_outliers(df, columns):
      bounds = {}
      for i in columns:
        quantiles = df.approxQuantile(i,[0.10,0.90], 0)
        IQR = quantiles[1] - quantiles[0]
        bounds[i] = [quantiles[0] - 1.5 * IQR, quantiles[1] + 1.5 * IQR]
      quantile = f.udf(lambda x, y: 'yes' if x < bounds[y][0] or x > bounds[y][1] else 'no')
      for i in columns:
        df = df.withColumn(i + '_is_outlier', quantile(f.col(i), f.lit(i)))
        df = df.filter(df[i + '_is_outlier'] == 'no')
      df = df.drop(*[i + '_is_outlier' for i in columns])
      return df

#a new dataframe is created named cleaned - outliers are removed
    cleaned = remove_outliers(attributes, attributes.columns)
    print("Number of rows = " + str(cleaned.count()))
    print("Number of col = " + str(len(cleaned.columns)))
```

Step 6, Vector assembler

```
#Code below transforms dataframe to a new dataframe with two columns - stats and
LeagueIndex
#stats is vector column that's the combination of columns from the original df
#leagueindex is the target variable, rank
#trainer and tester are created from this df for ML uses

    va = VectorAssembler(inputCols = ['Age','Hours_Per_Week', 'APM',
    'SelectByHotkeys', 'AssignToHotkeys',
                         'UniqueHotkeys', 'MinimapAttacks', 'MinimapRightClicks',
                         'NumberOfPACs', 'GapBetweenPACs', 'ActionLatency',
                         "ActionsInPAC", 'TotalMapExplored','WorkersMade',
                         "UniqueUnitsMade", "ComplexUnitsMade",
    "ComplexAbilitiesUsed"], outputCol = 'stats')
    vd = va.transform(cleaned)
    vd = vd.select(['stats', 'LeagueIndex'])
    vector2 = vd.select('stats')
    vd.show(10)
    split = vd.randomSplit([0.7, 0.3])
    trainer = split[0]
    tester = split[1]
```

Step 7, Cross Validation

#code below estabishes a cross validator for linear regression with 10 folds

```
lr = LinearRegression(featuresCol = 'stats', labelCol= 'LeagueIndex', maxIter=10)
grid = ParamGridBuilder()\
   .addGrid(lr.regParam, [0.3])\
   .addGrid(lr.fitIntercept, [True])\
   .build()
lrevaluator = RegressionEvaluator(predictionCol="prediction",
labelCol="LeagueIndex", metricName="rmse")
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=lrevaluator,
   numFolds=10)
```

#The cross validator is fitted using the trainer, its model stats (pvalue, RMSE, etc) are printed

```
lrcvModel = cv.fit(trainer)

lrcvSummary = lrcvModel.bestModel.summary
print("P Values: " + str(lrcvSummary.pValues))
print("RMSE = %g" % lrcvSummary.rootMeanSquaredError)
print("R2 = %g" %lrcvSummary.r2)
print("MAE = %g" % lrcvSummary.meanAbsoluteError)
print("MSE = %g" % lrcvSummary.meanSquaredError)
print("R2adj = %g" % lrcvSummary.r2adj)
print("tValue = " + str(lrcvSummary.tValues))
```

Step 8, Create Regression Model

#code below the predicted regression model is created using the cv and tester, its R2 and RMSE are printed

```
reg_pred = lrcvModel.transform(tester)
reg_pred.select("prediction","LeagueIndex","stats").show(5)
r2_evaluator = RegressionEvaluator(predictionCol="prediction", \
            labelCol="LeagueIndex",metricName="r2")
predictions = r2_evaluator.evaluate(reg_pred)
print("R^2 (test)= %g" % predictions)
test_result = lrevaluator.evaluate(reg_pred)
print("Root Mean Squared Error (RMSE) on test data = %g" % test_result)
```

Step 9, More Regression Tests

```
dt = DecisionTreeRegressor(featuresCol ='stats', labelCol = 'LeagueIndex')
dt_model = dt.fit(trainer)
dt_predictions = dt_model.transform(tester)
dt_evaluator = RegressionEvaluator(
   labelCol="LeagueIndex", predictionCol="prediction", metricName="rmse")
rmse = dt_evaluator.evaluate(dt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
dt_model.featureImportances


gbt = GBTRegressor(featuresCol = 'stats', labelCol = 'LeagueIndex', maxIter=10)
gbt_model = gbt.fit(trainer)
gbt_predictions = gbt_model.transform(tester)
gbt_predictions.select('prediction', 'LeagueIndex', 'stats').show(2)
gbt_evaluator = RegressionEvaluator(
    labelCol="LeagueIndex", predictionCol="prediction", metricName="rmse")
rmse = gbt_evaluator.evaluate(gbt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```