# CIS 5560 TERM PROJECT TUTORIAL

#### Authors: Ritika Joshi, Ketki Vyas, Sanchita Gawand

#### Instructor: [Jongwook Woo](https://www.linkedin.com/in/jongwook-woo-7081a85)

#### Date: 18/05/2019

**Lab Tutorial**



**Prediction of Obesity Risk Level Among Youth**

**Objectives**

In this hands-on lab, you will learn how to:

* Build machine learning models on Azure ML
* Choose best model for the prediction
* Due to restriction on Azure ML, implement model on big dataset using Databricks.
* Learn pyspark commands
* How to calculate RMSE and Accuracy for machine learning regression algorithms.
* Apply implementation on Oracle CLI for better efficiency of code execution.

**Platform Spec**

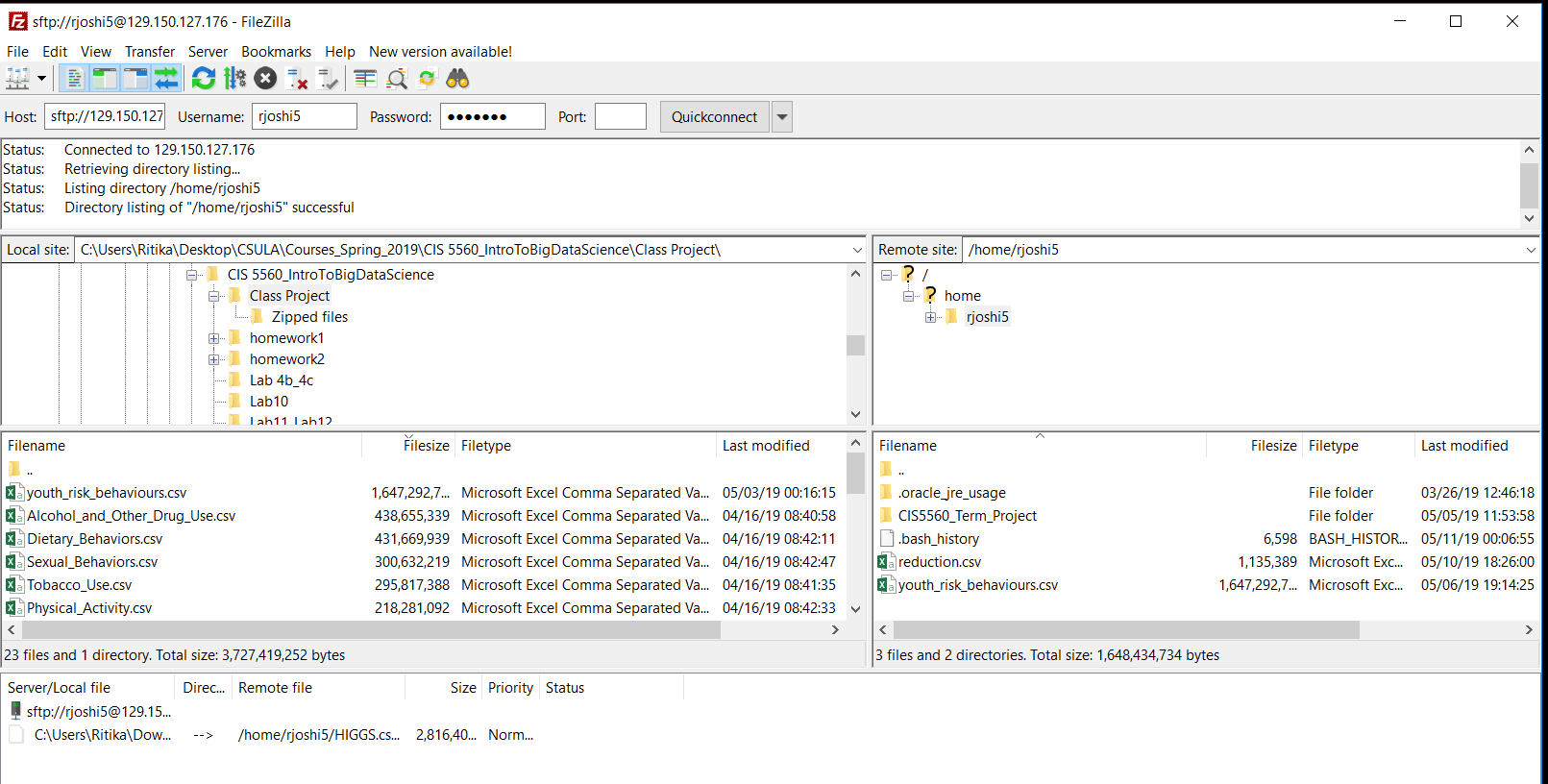
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Azure** | **Databricks** | | | | | | **Oracle** | |
| **Storage** | **10 GB** | **Cluster 5.2 (includes Apache Spark 2.4.0, Scala 2.11)** | | **682GB, 12 OCPUs** | | | | | |
| **Memory** |  | **6GB Memory , 0.88 Cores, 1 DBU** | | | | **180GB** | | | |
| **Node** | **1** | |  | | | | | | **6** |
| **Python Version** | **2.7.11** | **3.5.2** | | | | | **2.7.14** | | |
| **URL / IP Address** | [**https://studio.azureml.net**](https://studio.azureml.net/) | [**https://community.cloud.databricks.com/**](https://community.cloud.databricks.com/) | | | [**129.150.127.176**](mailto:yourusername@129.150.127.176) | | | | |

**Step 1: Download the dataset**

1. Click on the link <https://www.kaggle.com/raylo168/dash-yrbss-hs-2017>
2. Download all csv files.

**Step 2: Prepare the data**

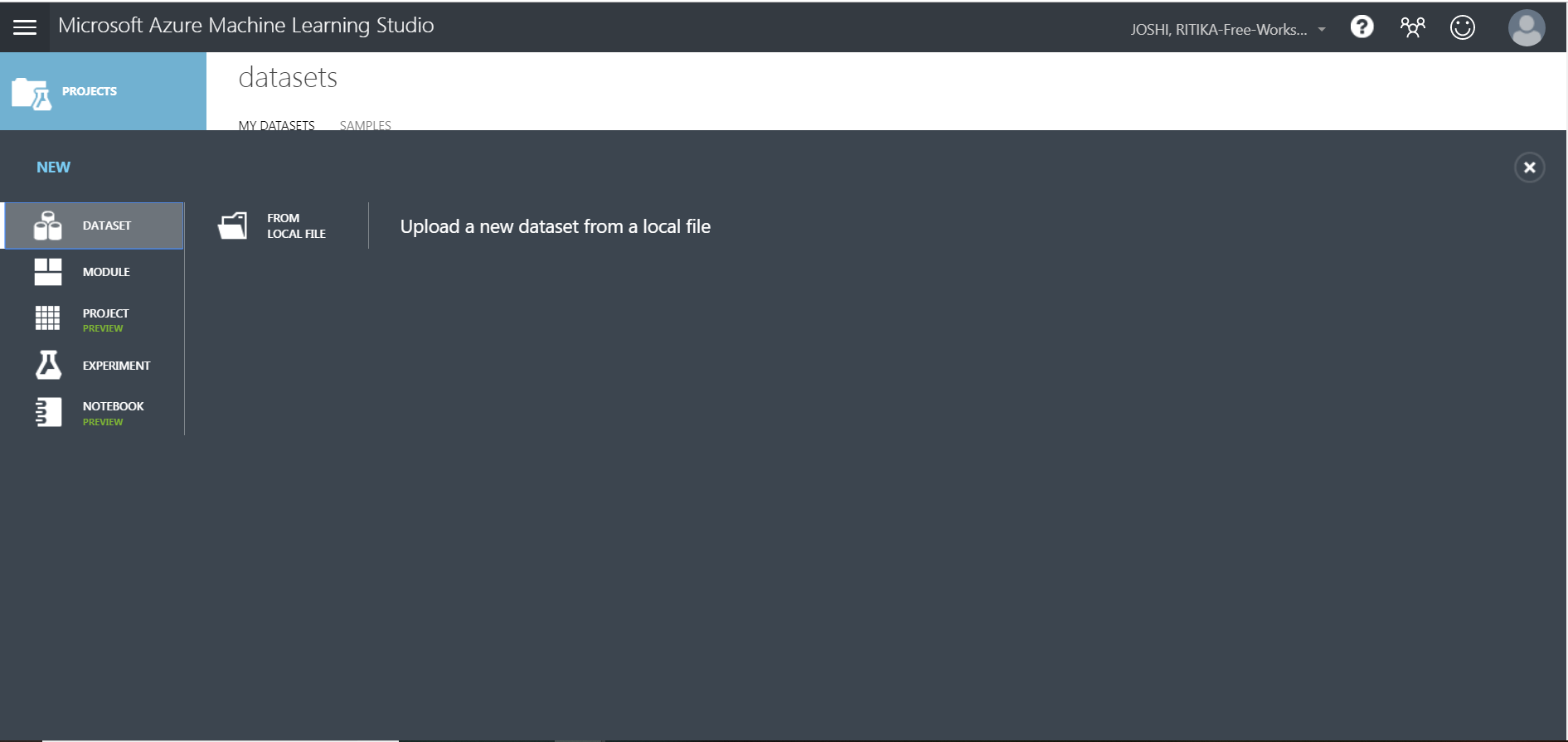
* 1. Upload all files on Oracle cluster using FileZilla and hdfs command.

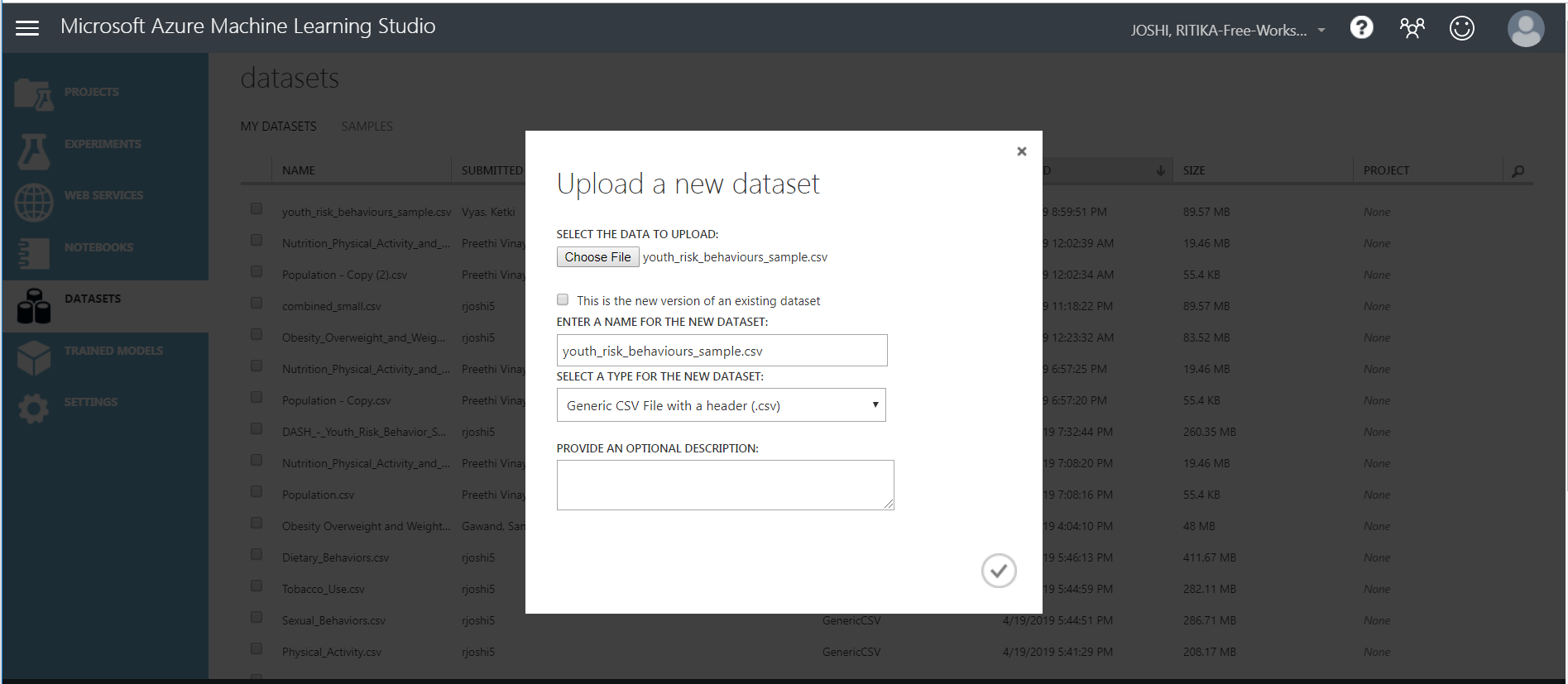


* 1. All files have same columns name and count so join all files and remove more than one headers.
  2. For Azure we need small dataset so prepare a csv file with equal number of columns from each file.

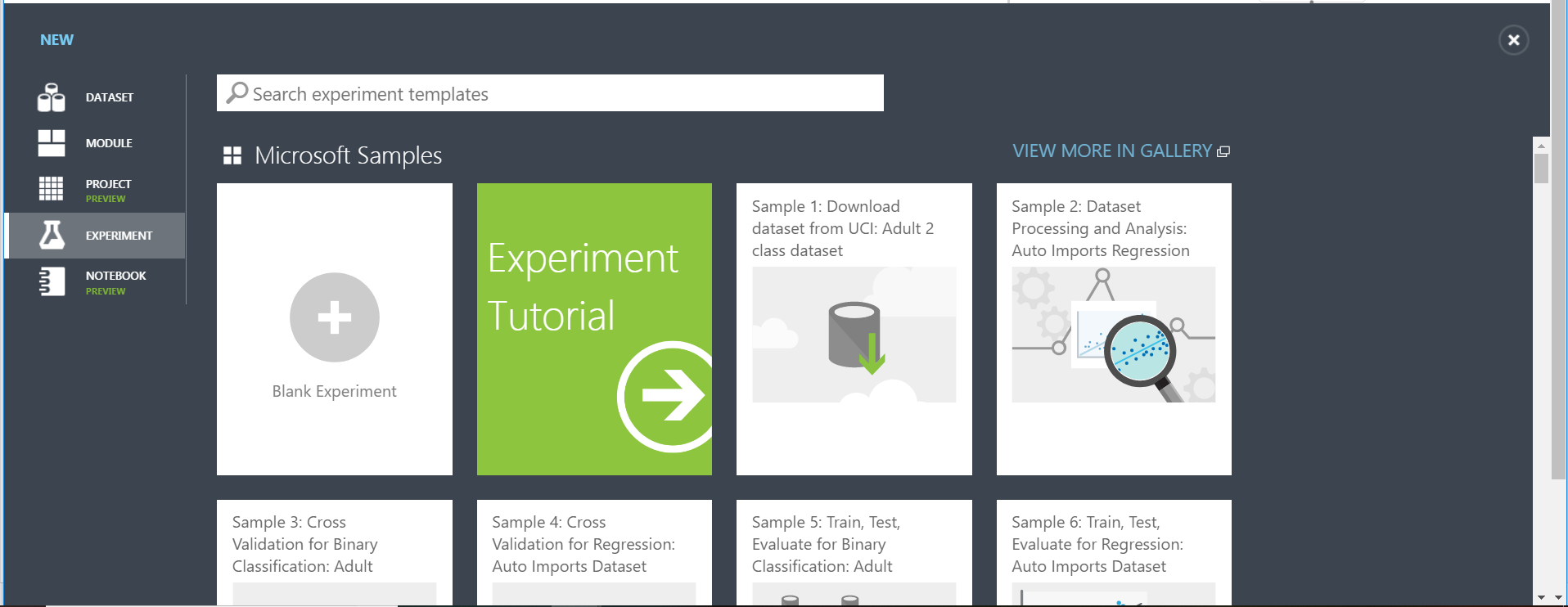
**Step 3: Implementation on Azure**

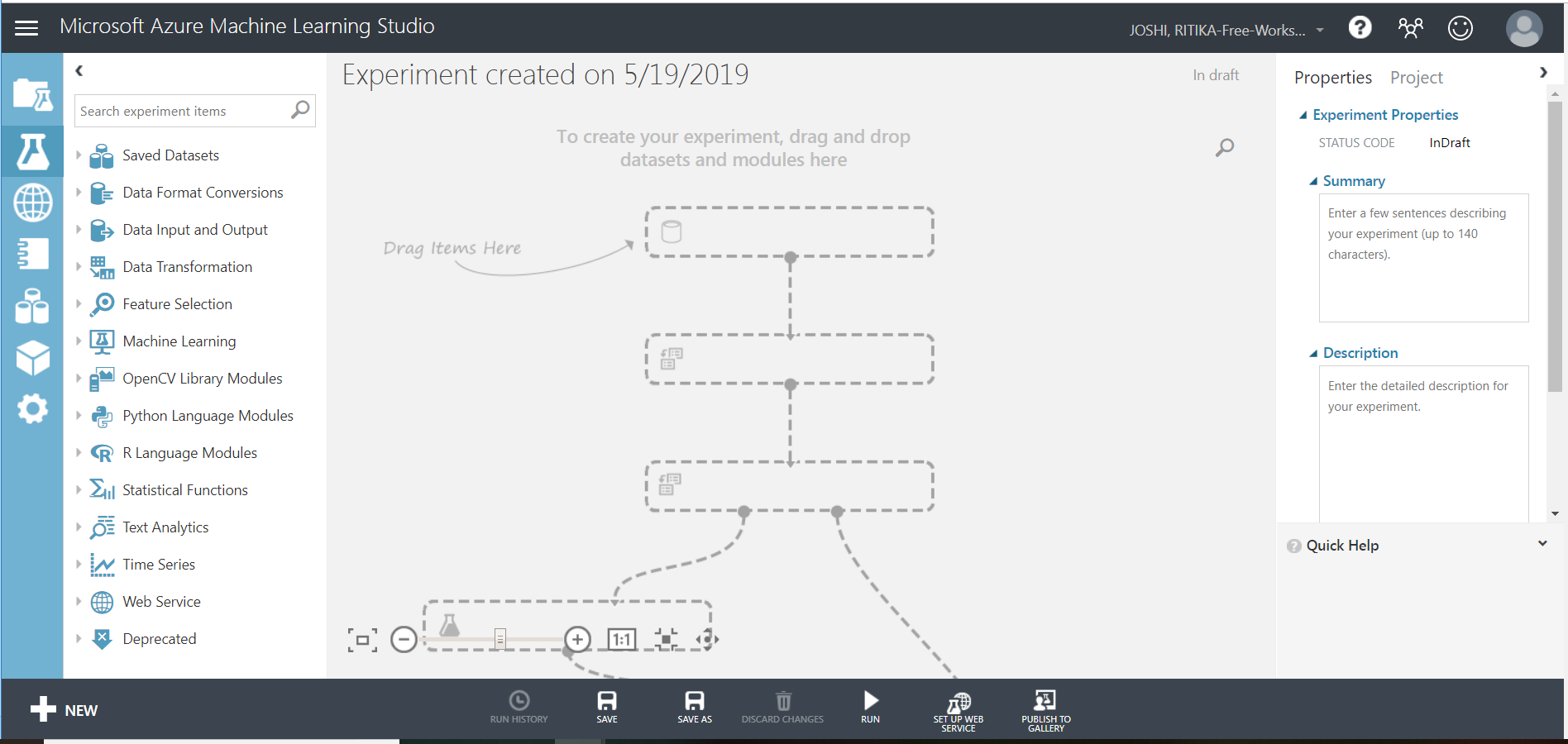
1. Login into Azure ML Studio <https://studio.azureml.net/> and upload smaller dataset created in previous step by clicking on ‘DATASET’. Choose dataset file and click on done. File upload will take few minutes to upload.



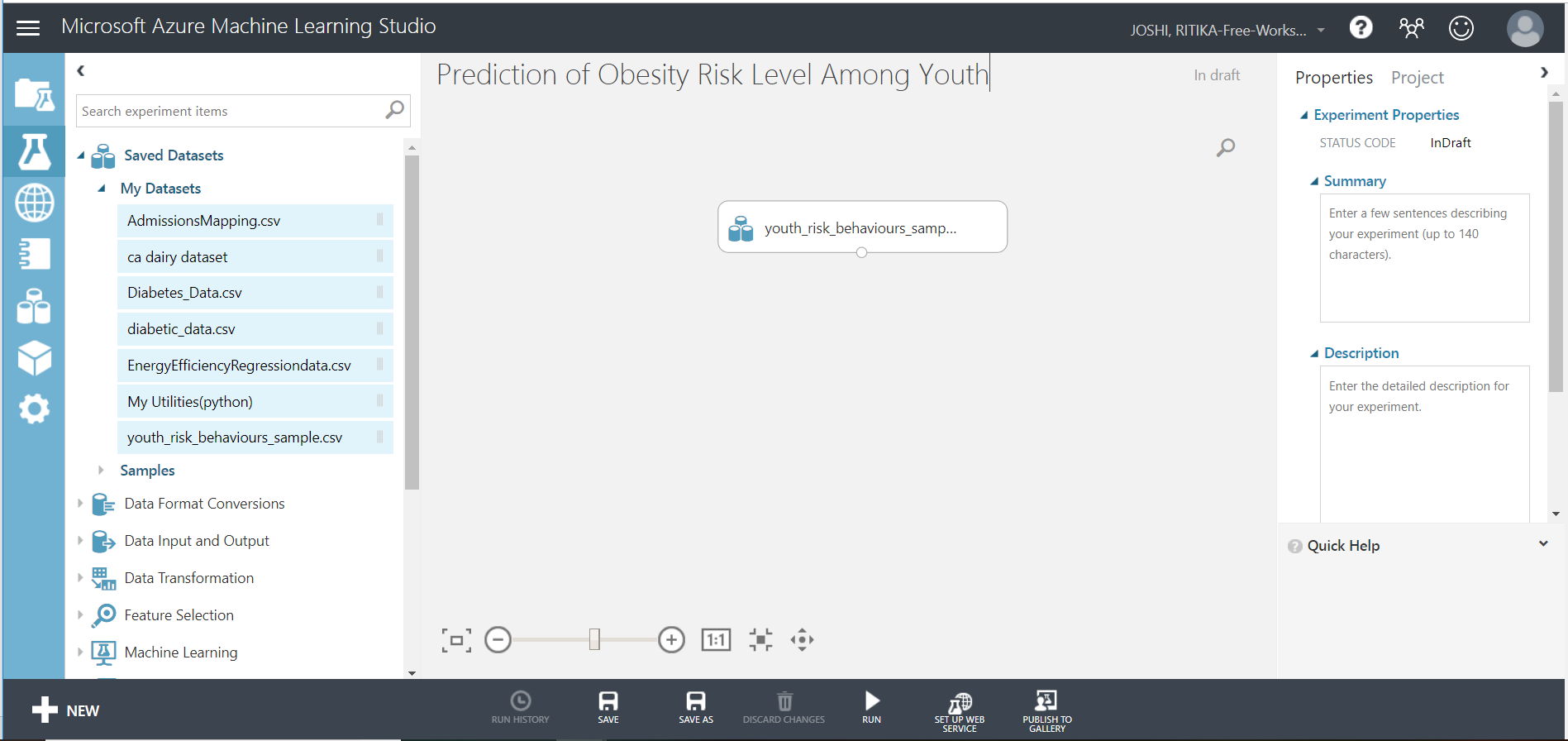


1. Create a new experiment page by clicking on add sign at the bottom left

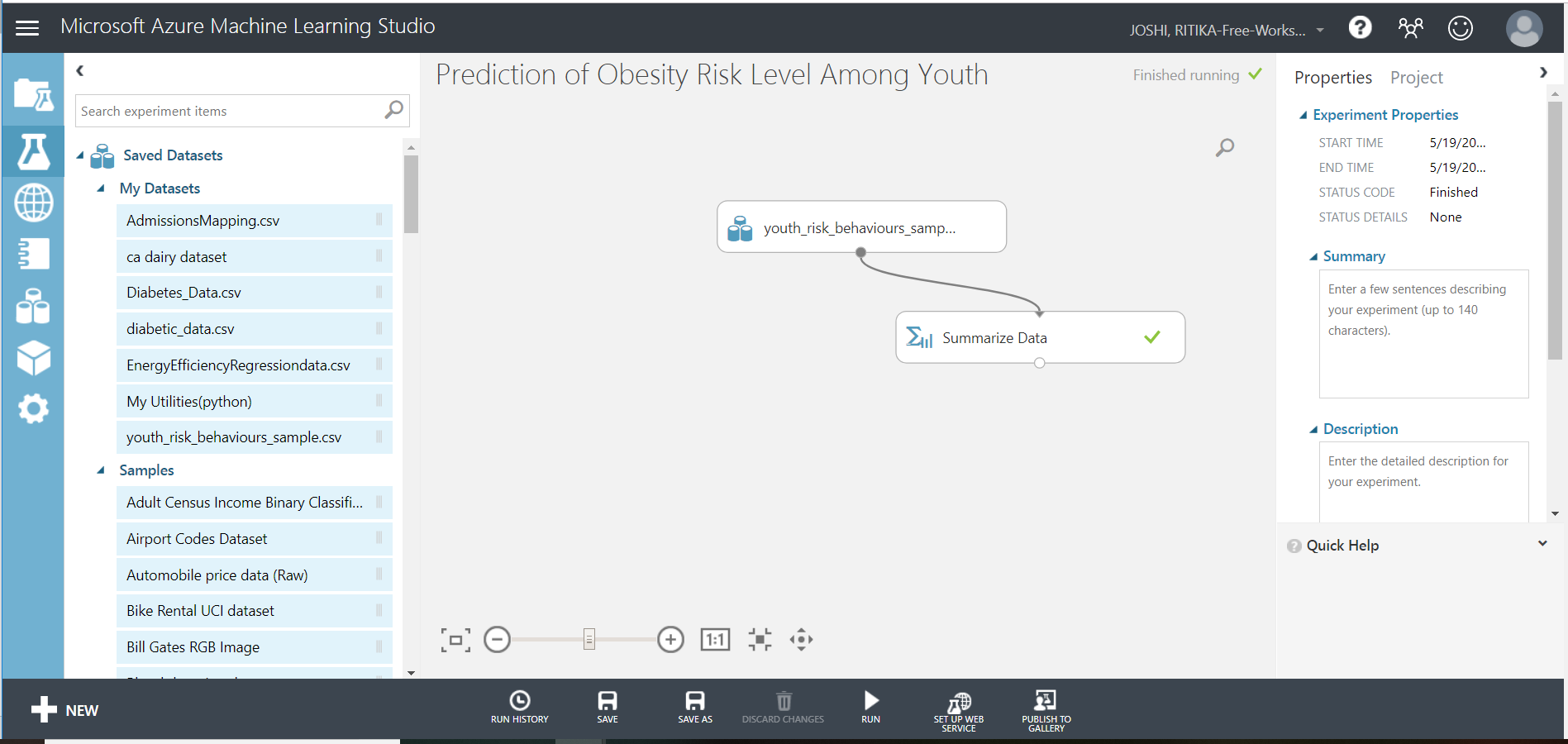


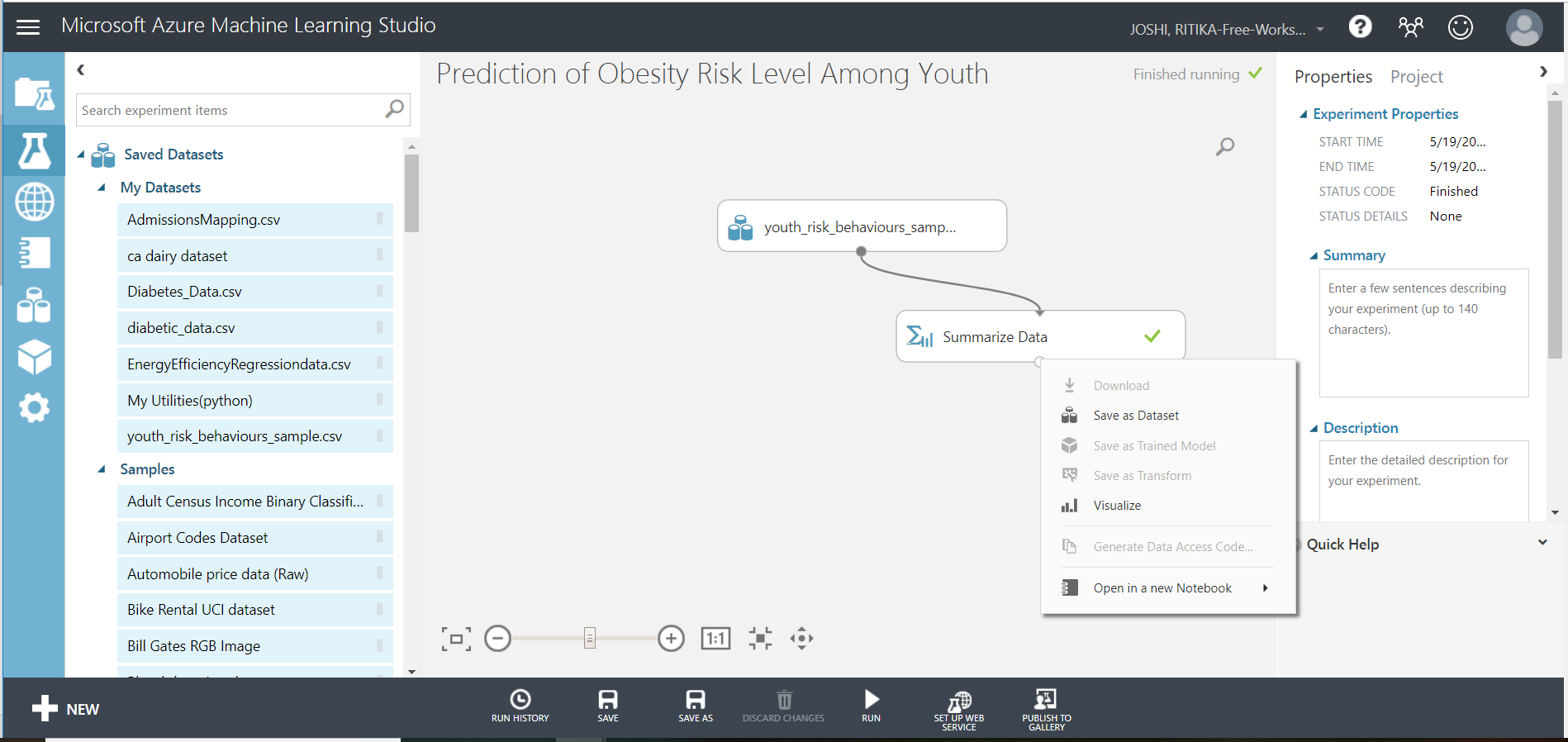


1. Name the experiment as ‘Prediction of Obesity Risk Level Among Youth’ and drag the dataset on to canvas.



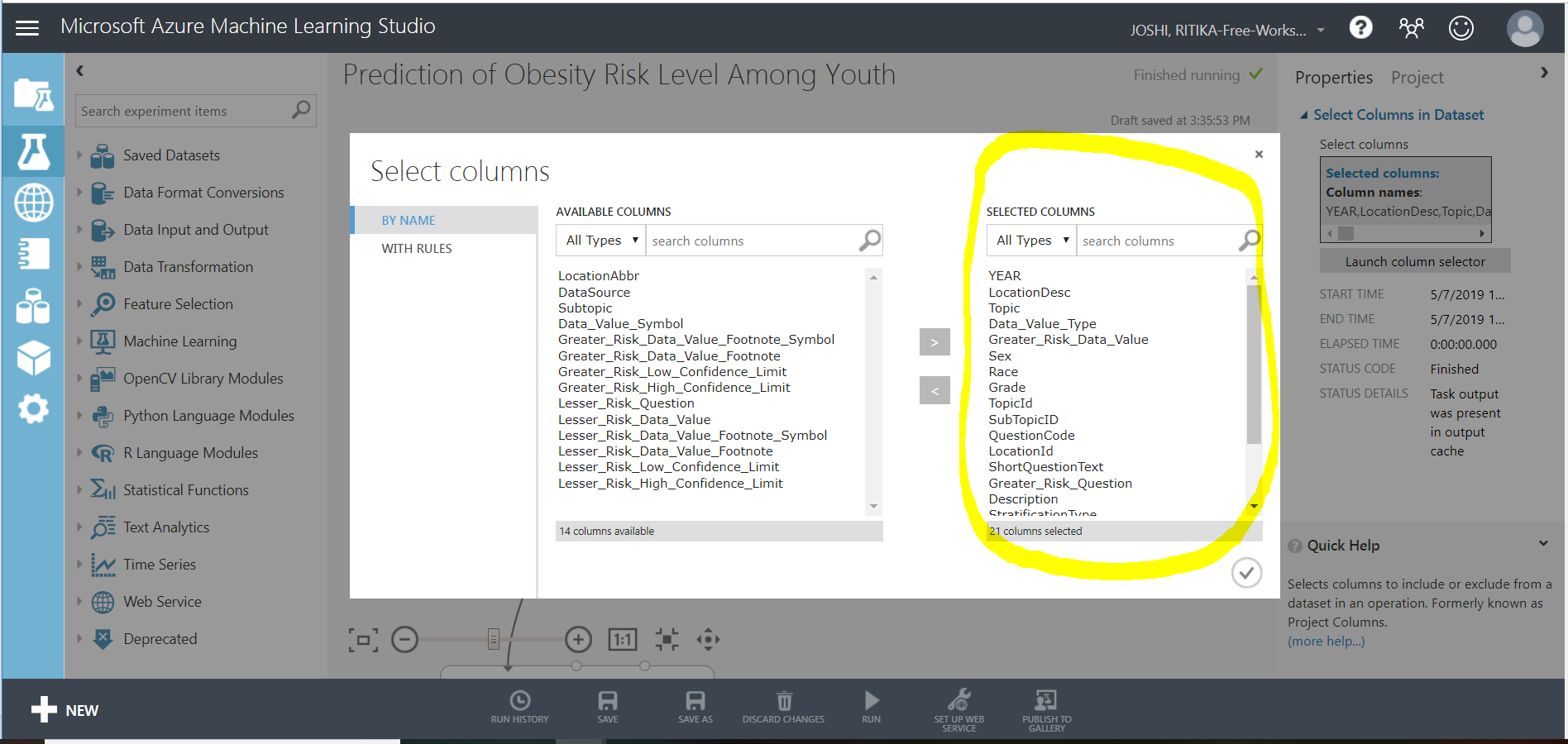
1. Select ‘Summarize Data’ and link output of dataset module with input of ‘Summarize Data’ module as shown in the screenshot below. Save and Run the experiment. You can now visualize statistics of all columns.

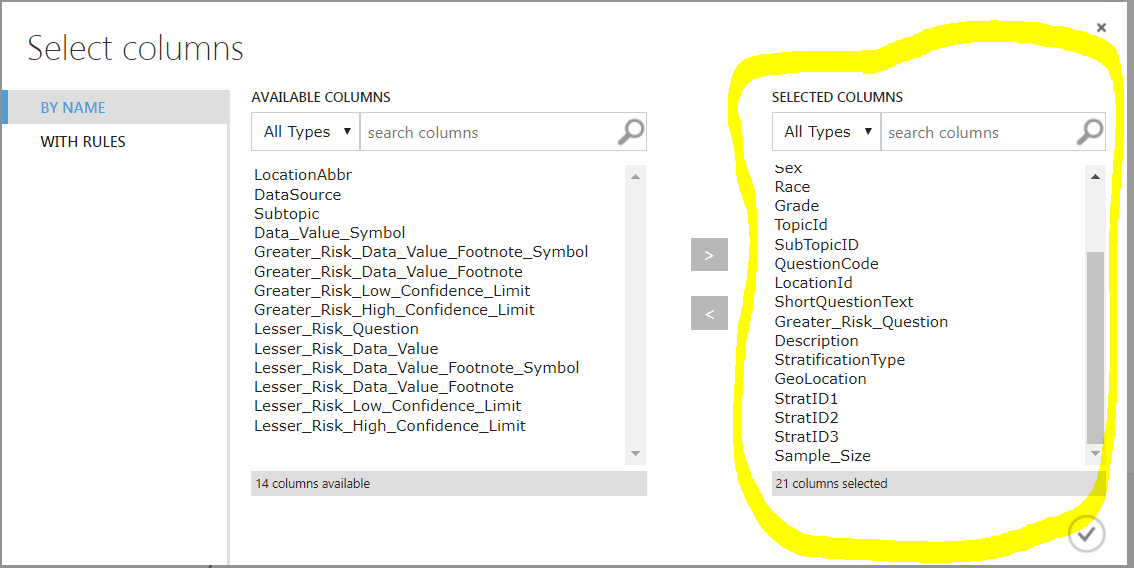




1. Drag ‘Select Columns in Dataset’ module on to canvas and click on it. Now click on ‘Launch Column Selector’ at the right and configure as shown below. Select 21 columns as shown at the right side

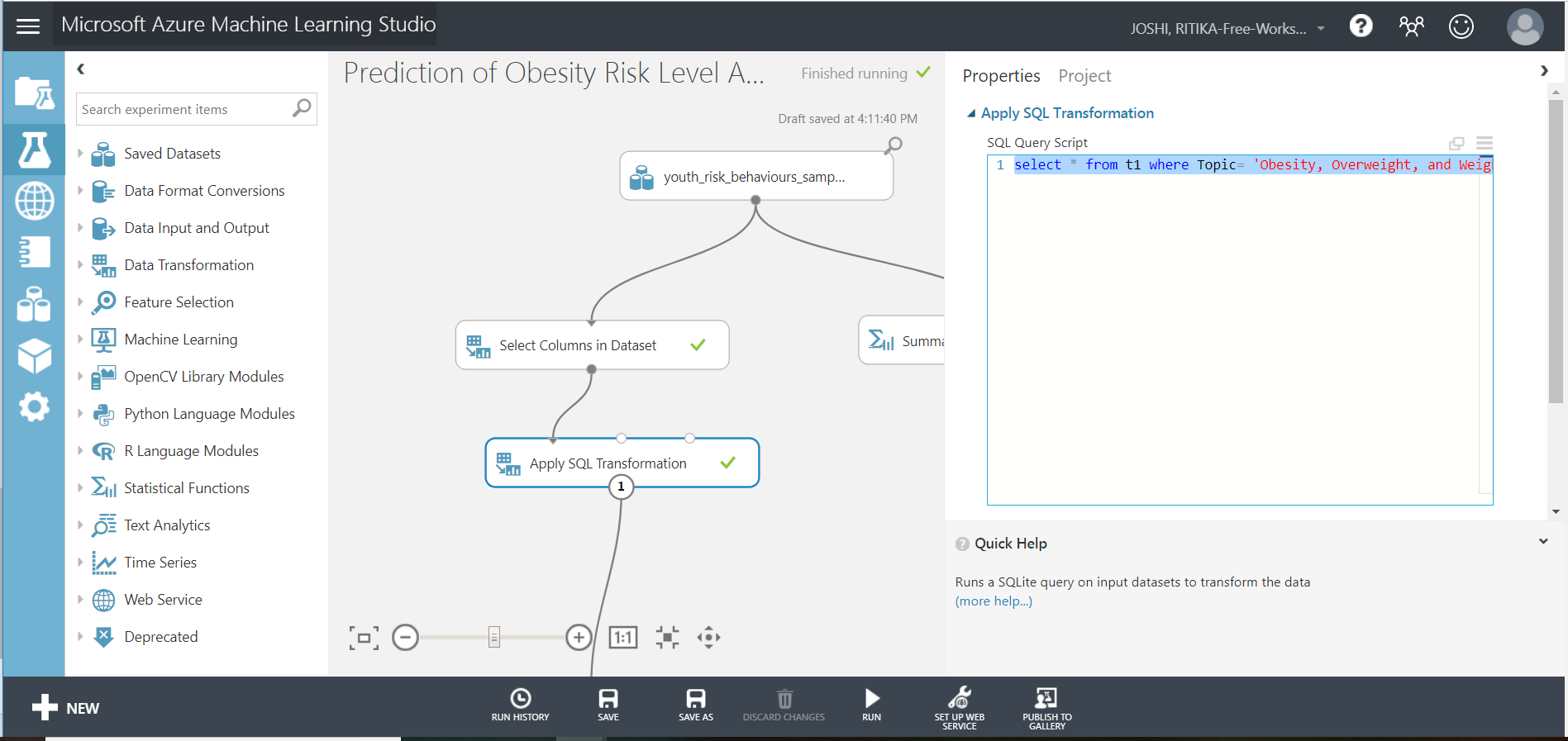






1. Drag ‘Apply SQL Transformation’ and edit SQL Query script at the right.

select \* from t1 where Topic= 'Obesity, Overweight, and Weight Control';



1. Drag ‘Clip Values’ and configure the module as below to fix outliers as 99th percentile or 1st percentile:

Set of thresholds: ClipPeaksAndSubpeaks

Threshold: Percentile

Percentile number of upper threshold: 99

Percentile number of upper threshold: 1

Substitute Value for peaks: Threshold

Substitute value for subpeaks: Missing

List of columns: Numeric, All

Overwrite Flag: Ticked



1. To remove rows with missing value, drag ‘Clean Missing Data’ module and configure as shown in the screenshot below.

Launch Column selector: Sample\_Size, YEAR

Minimum missing value ratio: 0

Maximum missing value ratio: 1

Cleaning mode: Custom substitution value

Replacement value: N/A

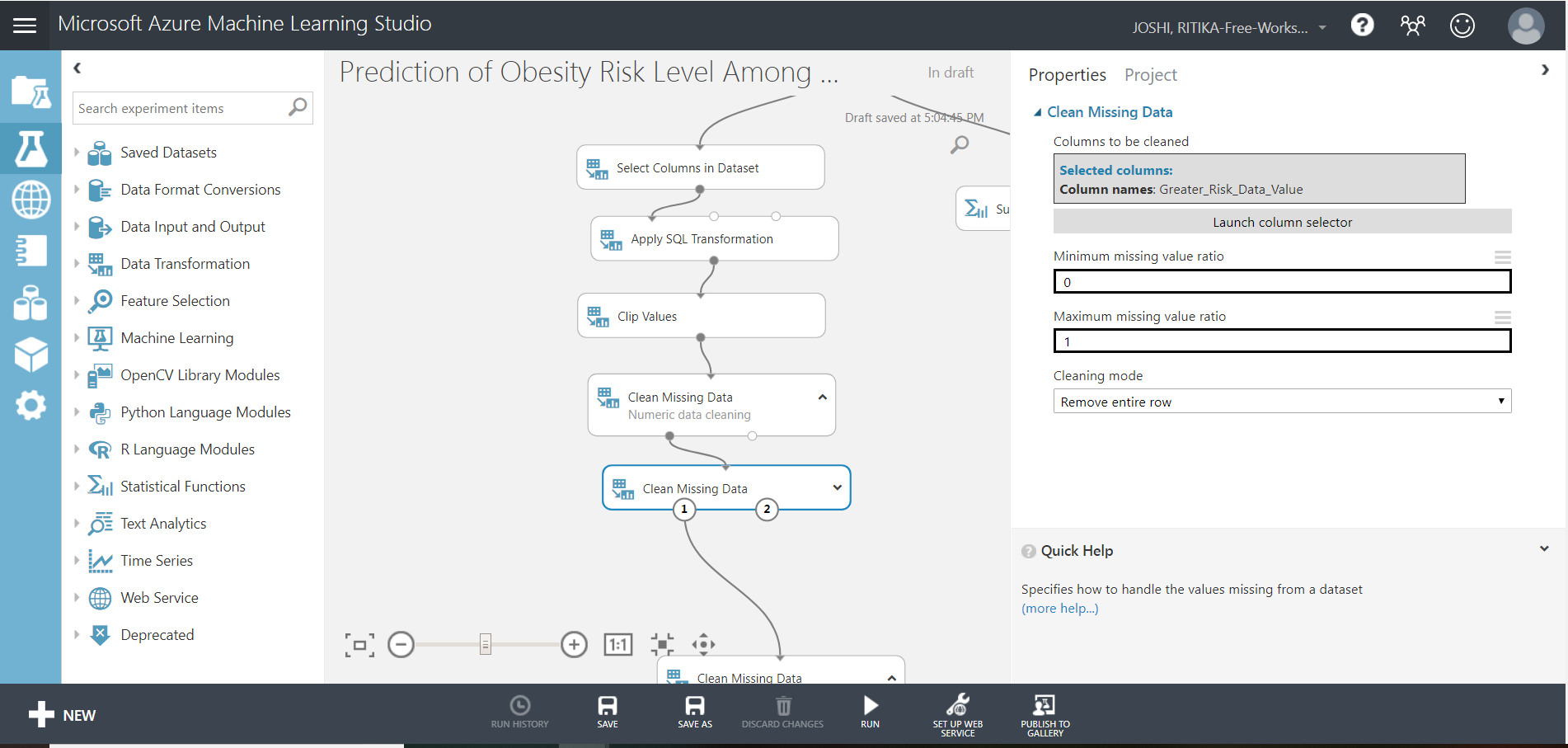


1. Clean data for the column that needs to be predicted i.e. ‘Greater Risk Data Value’

Minimum missing value ratio: 0

Maximum missing value ratio: 1

Cleaning mode: Remove Entire Row



1. Drag a new ‘Clean missing data’ module to remove rows for string type columns.

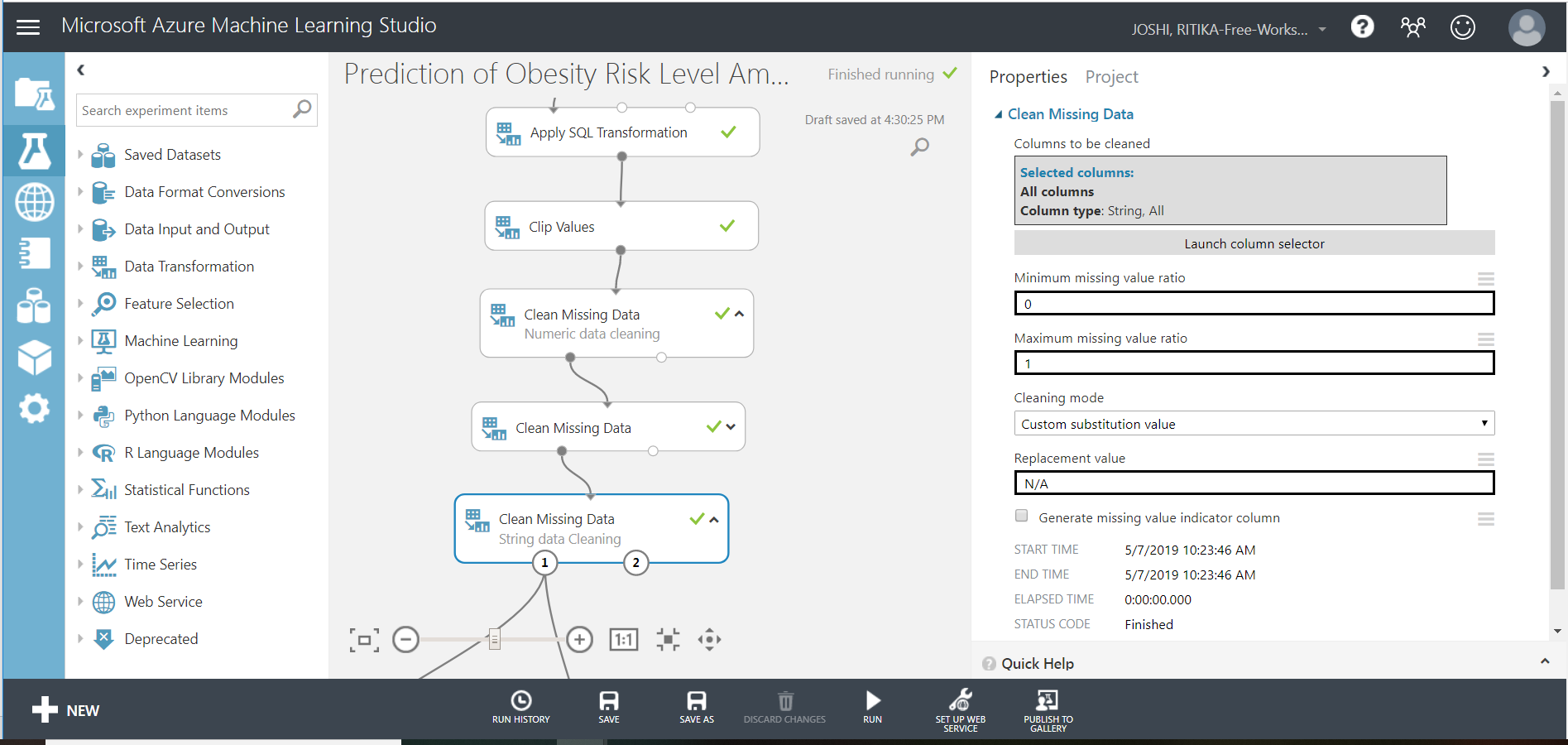
Launch Column selector: Column type: String

Minimum missing value ratio: 0

Maximum missing value ratio: 1

Cleaning mode: Custom substitution value

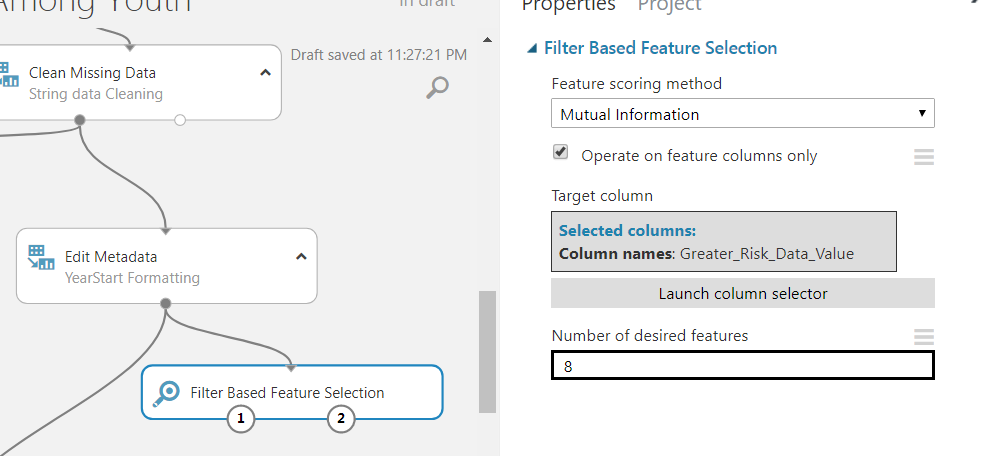
Replacement value: N/A



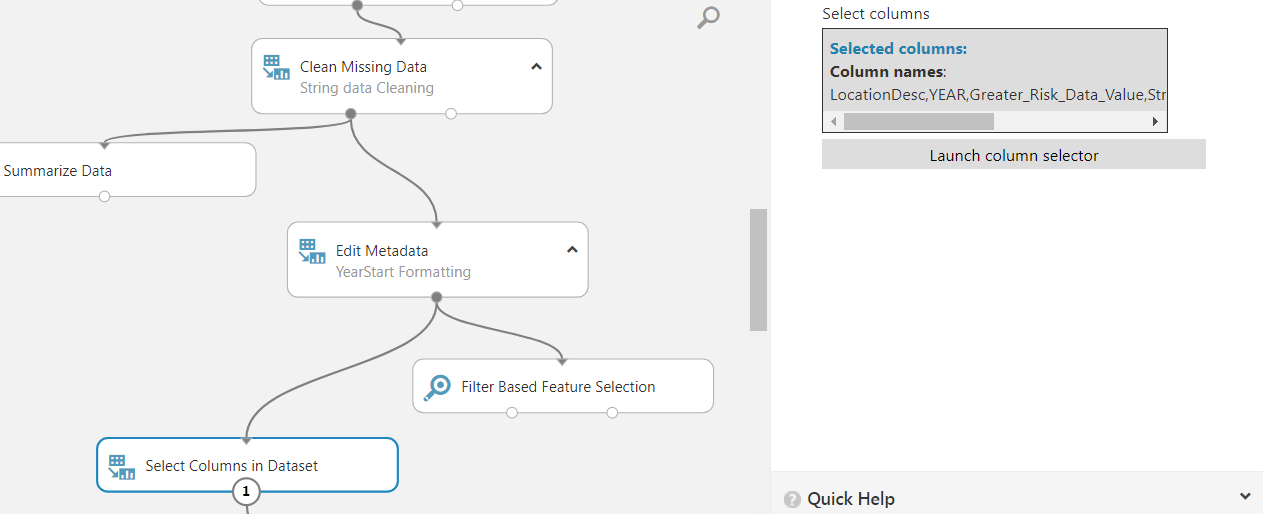
1. Drag ‘Edit Metadata’ to convert Year from integer data type to categorical

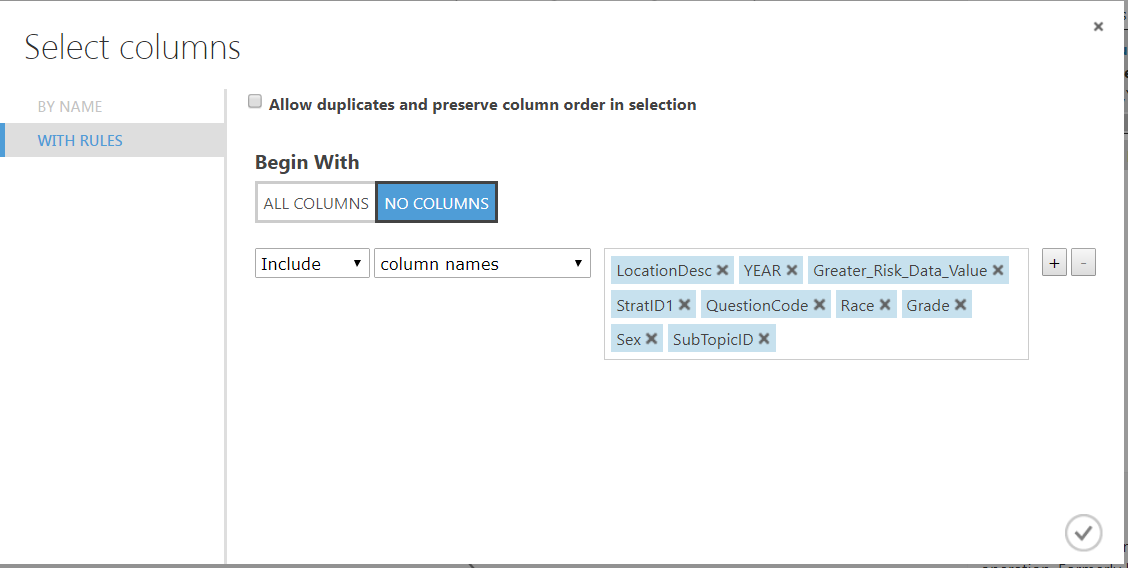


1. To select best eight columns that affect label, drag ‘filter based feature selection’ and chose column as ‘Grater Risk Data Value’. You may run the model and visualize what all columns are being selected by this module.

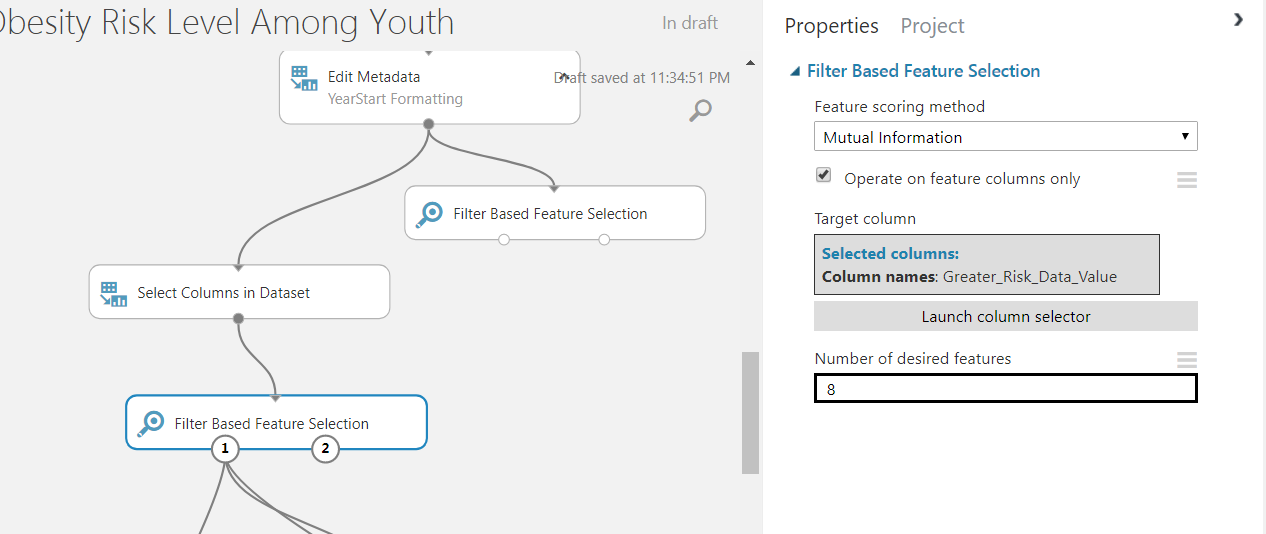


1. Drag ‘Select Columns in Dataset’ module to select only those columns that were received as output of previous module

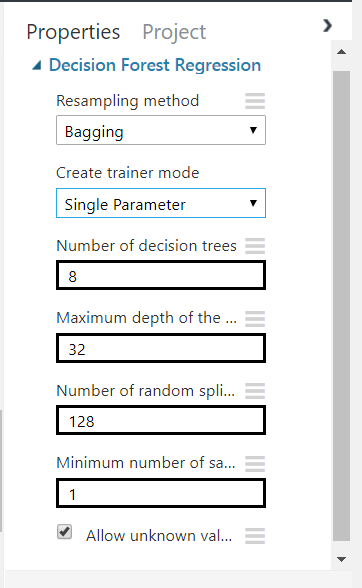
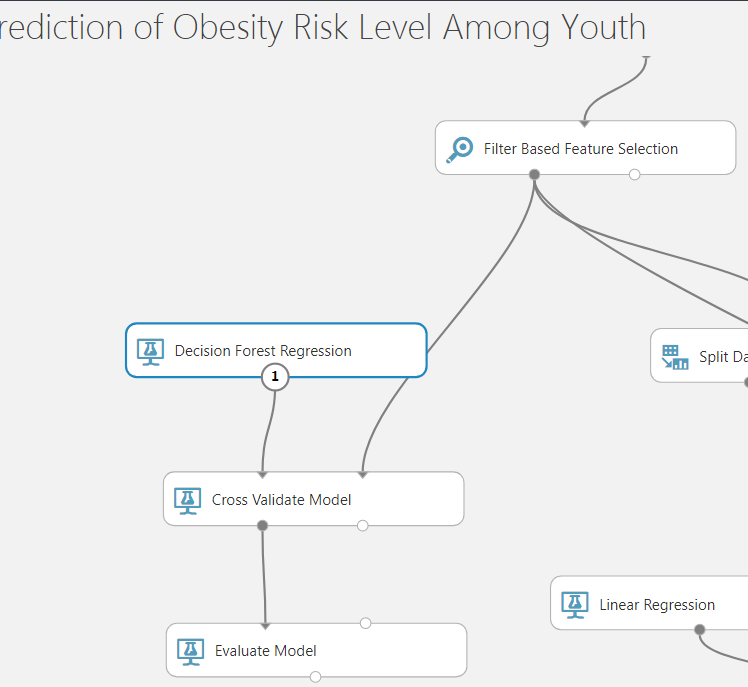




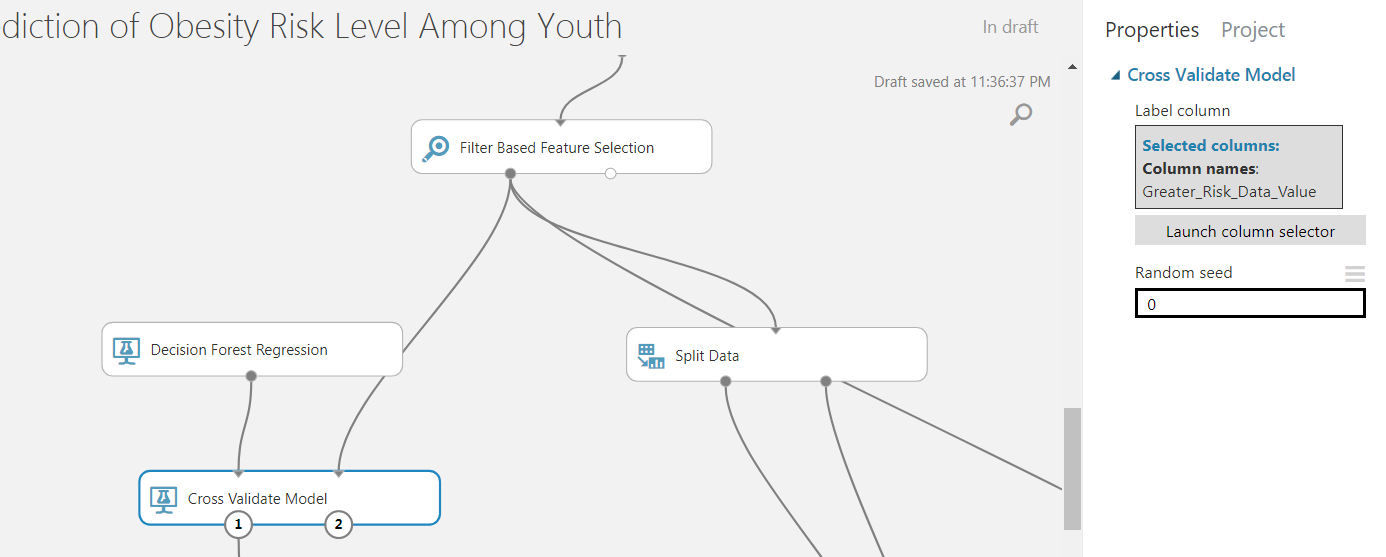
1. Another level check to find best columns to predict label values



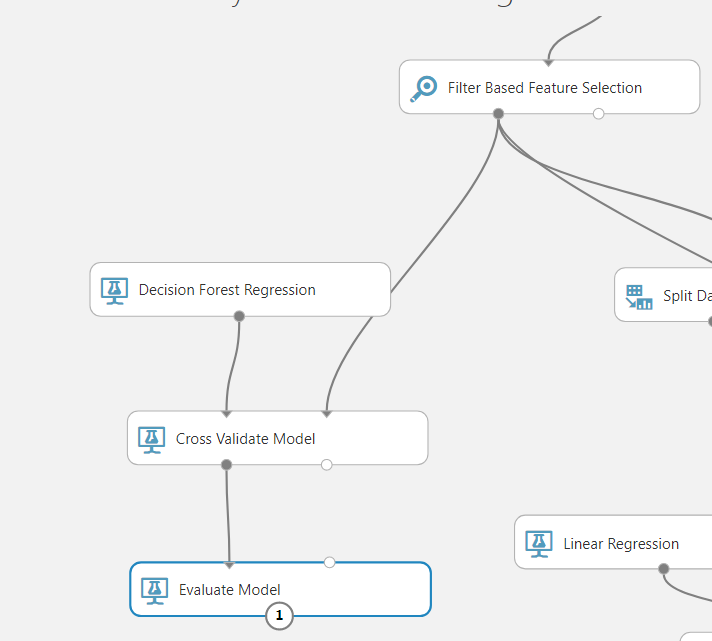
1. Implement Decision Forest with configuration shown in the snapshot.



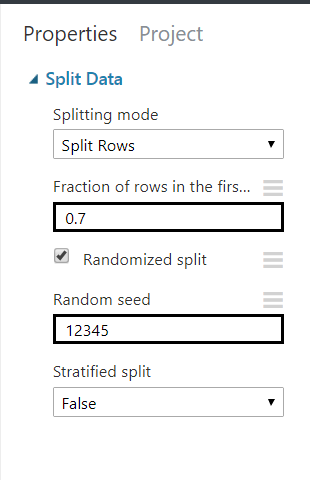
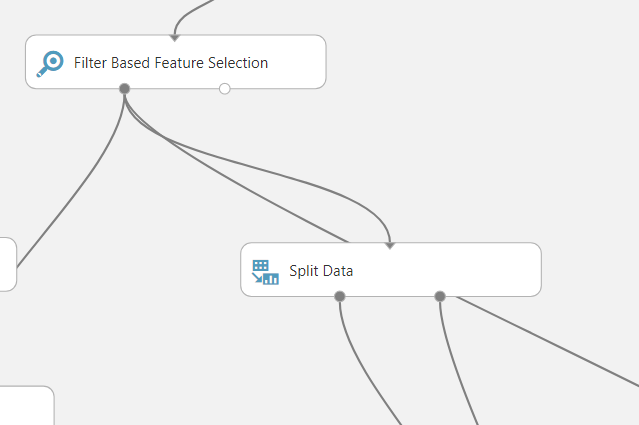
1. Drag ‘cross validate model’ and configure for ‘greater risk data value’



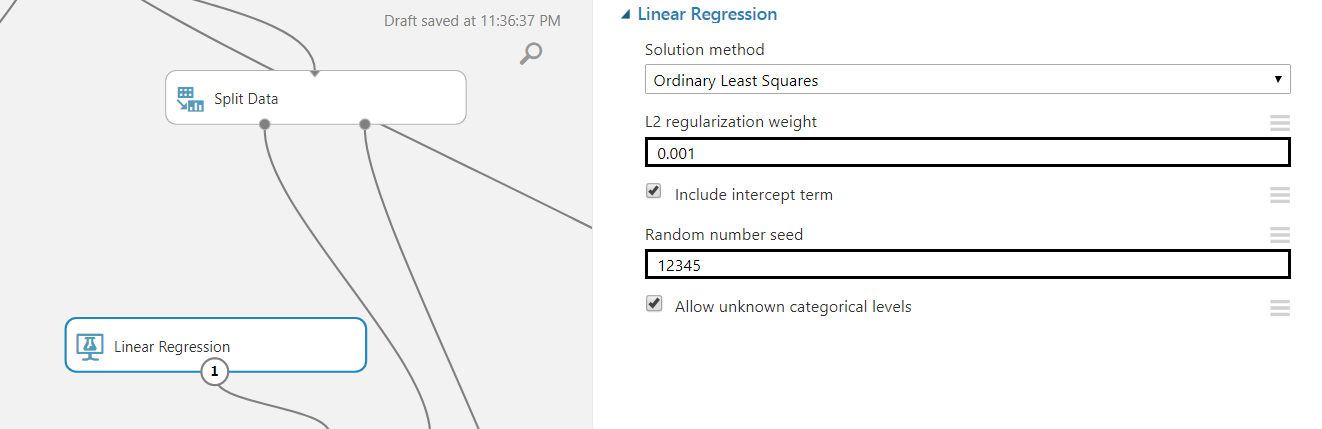
1. Add ‘Evaluate Model’



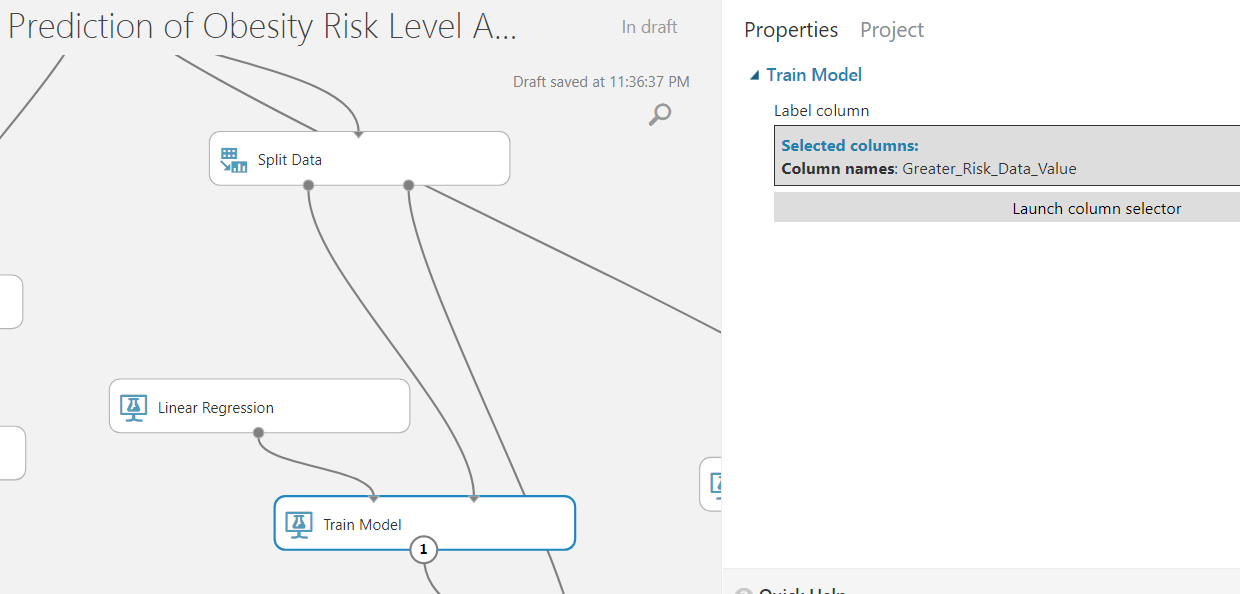
1. For linear regression split the data set into train and test



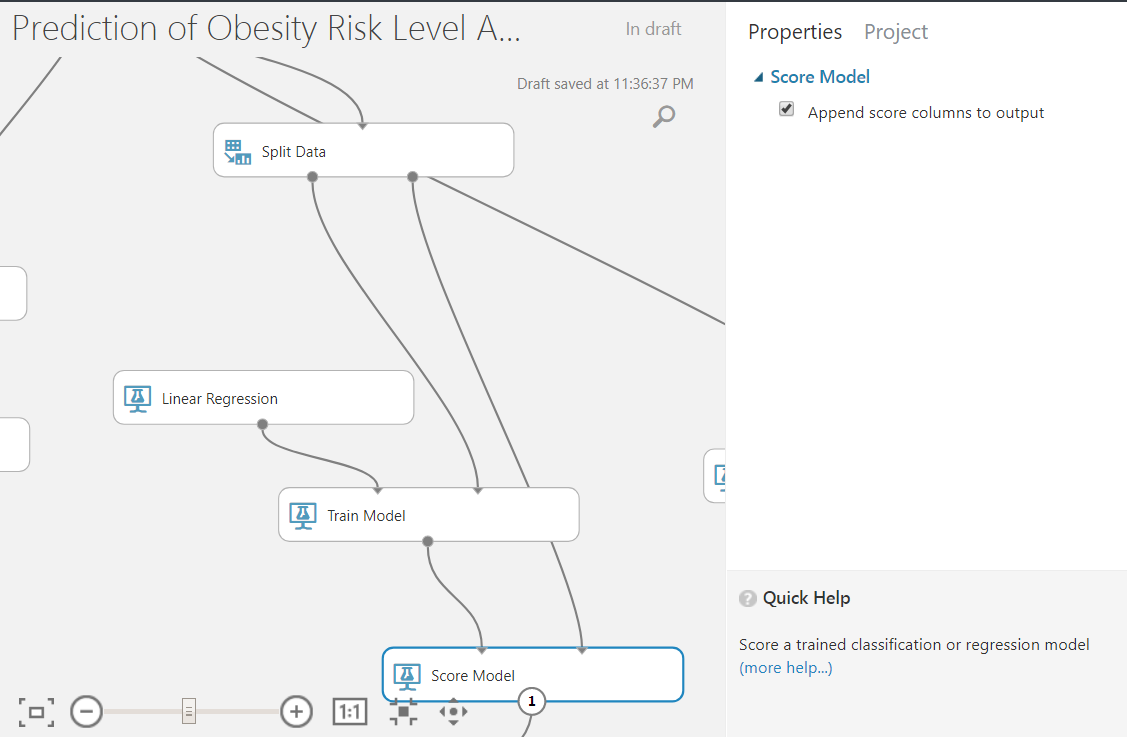
1. Add Linear Regression module with the config shown in screenshot below.



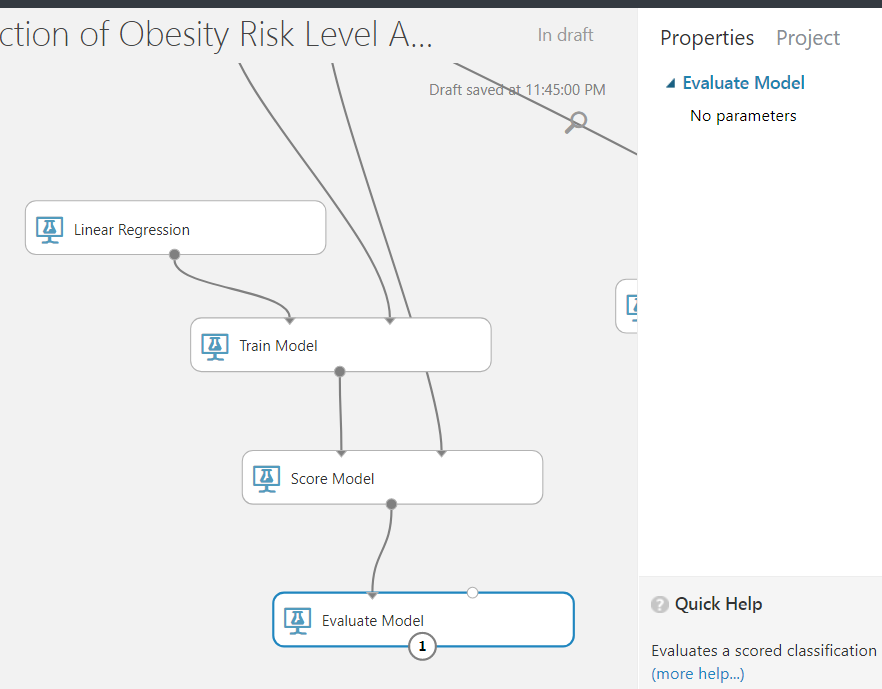
1. Add train model



1. Add ‘Score Model’ and link it as shown in the screenshot

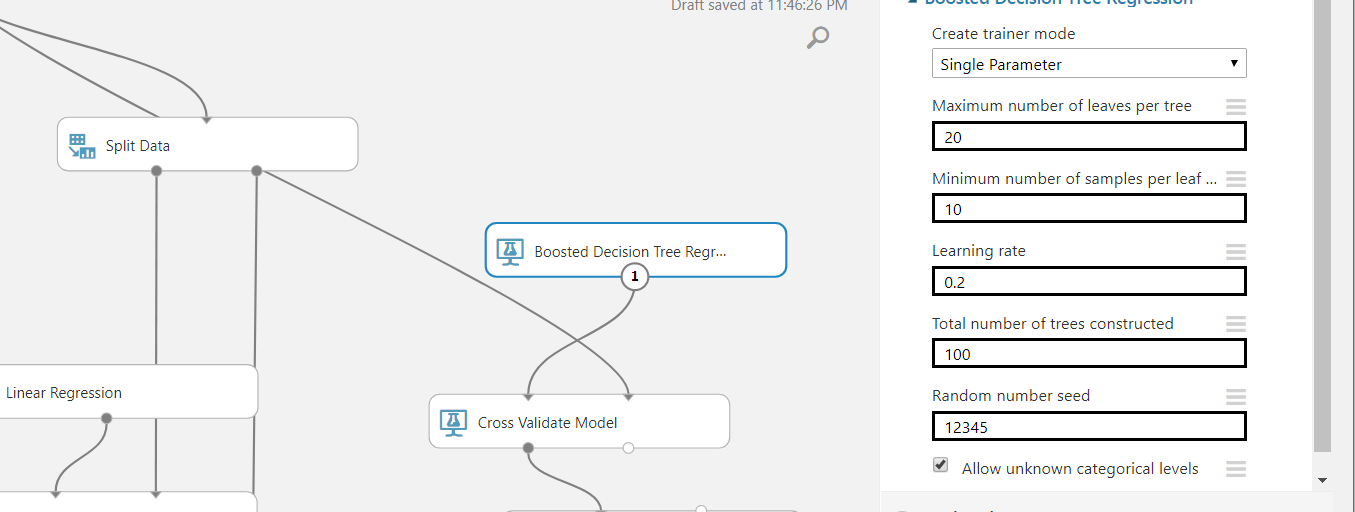


1. Evaluate model

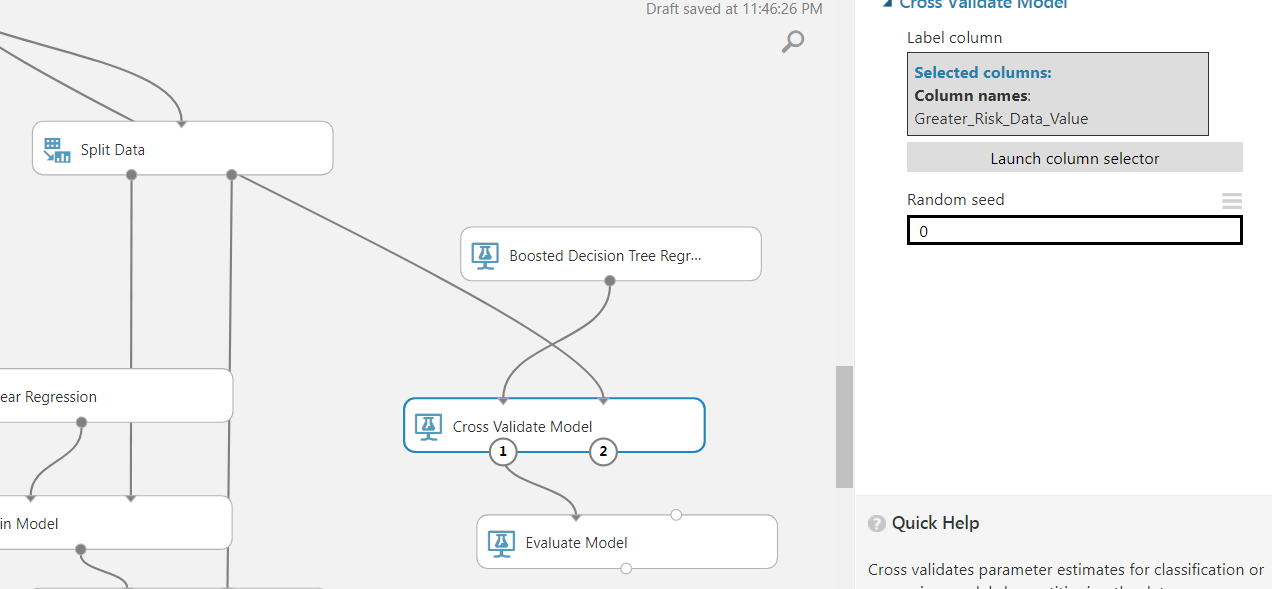


1. Boosted Decision Tree

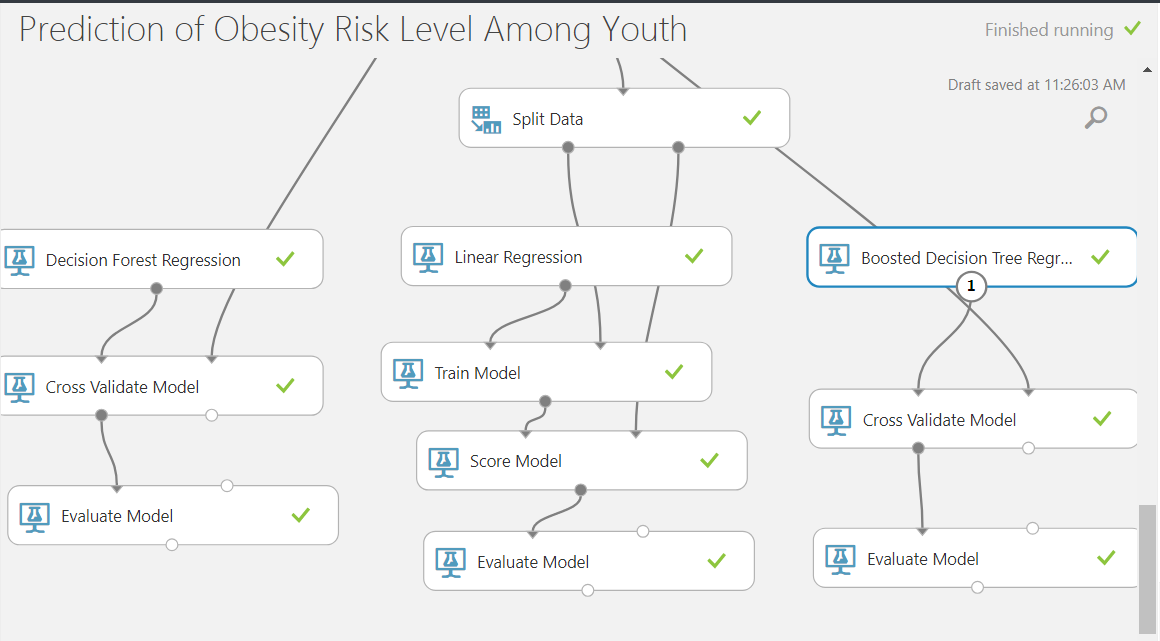
Drag and drop boosted decision tree module and configure it as shown in the screenshot



1. Add ‘Cross Validate’ with random seed 0 and then add Evaluate model.



1. After all above configurations model should look similar to below. Save and run the experiment.

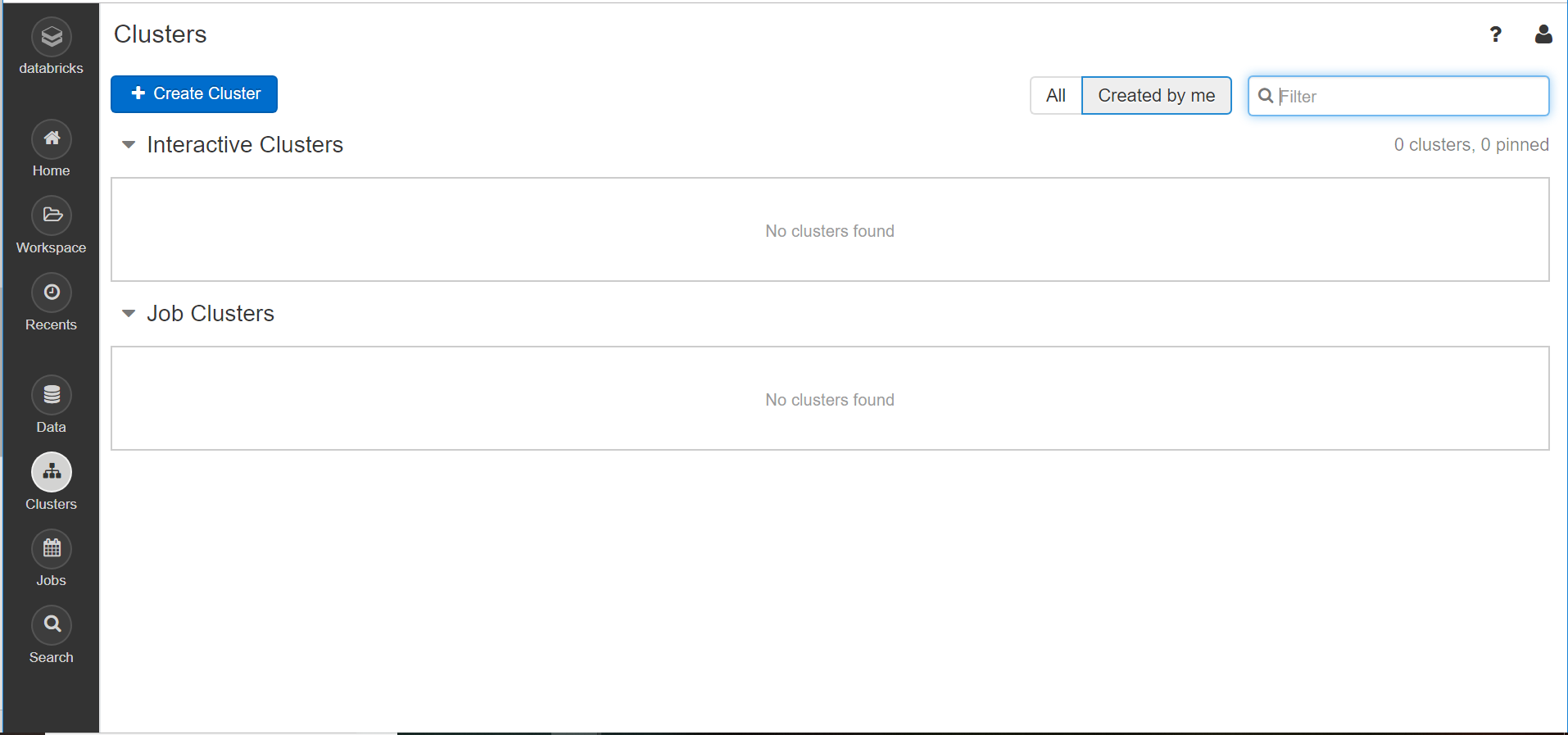


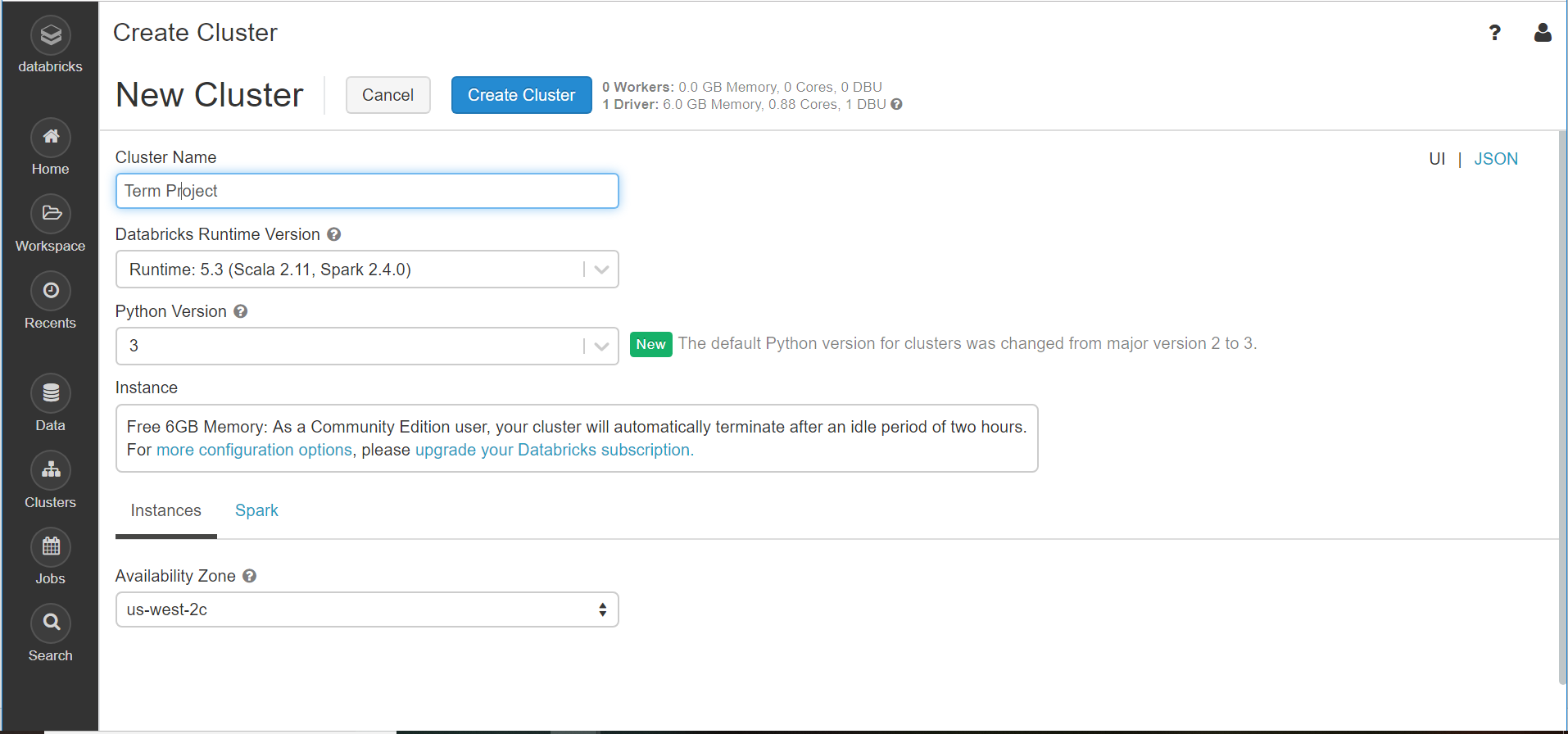
1. Visualize the RMSE of all the three ‘Evaluate Model’

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Decision Forest Regression** | **Linear Regression** | **Boosted Decision Tree Regression** |
| Mean Absolute Error | 2.91 | 5.64 | 2.86 |
| RMSE | 3.90 | 7.66 | 3.84 |
| Coefficient Of Determination | 0.95 | 0.83 | 0.95 |
| Relative Absolute Error | 0.18 | 0.36 | 0.18 |
| Relative Sqaured Error | 0.042 | 0.16 | 0.04 |

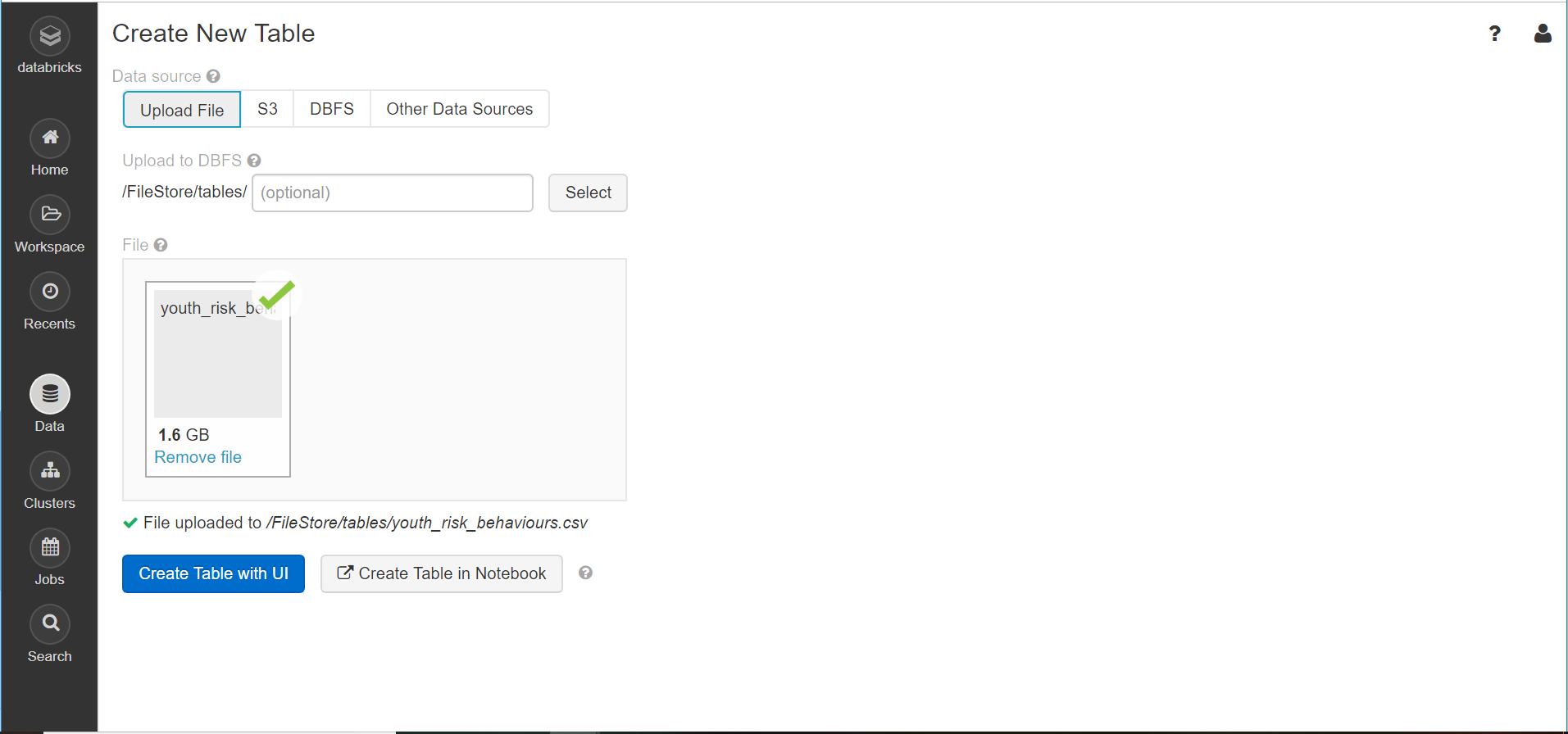
Step 4: Implementation on Databricks

1. Login to databricks using URL <https://community.cloud.databricks.com>
2. Create a new cluster named ‘Term Project’ and runtime version 5.3.

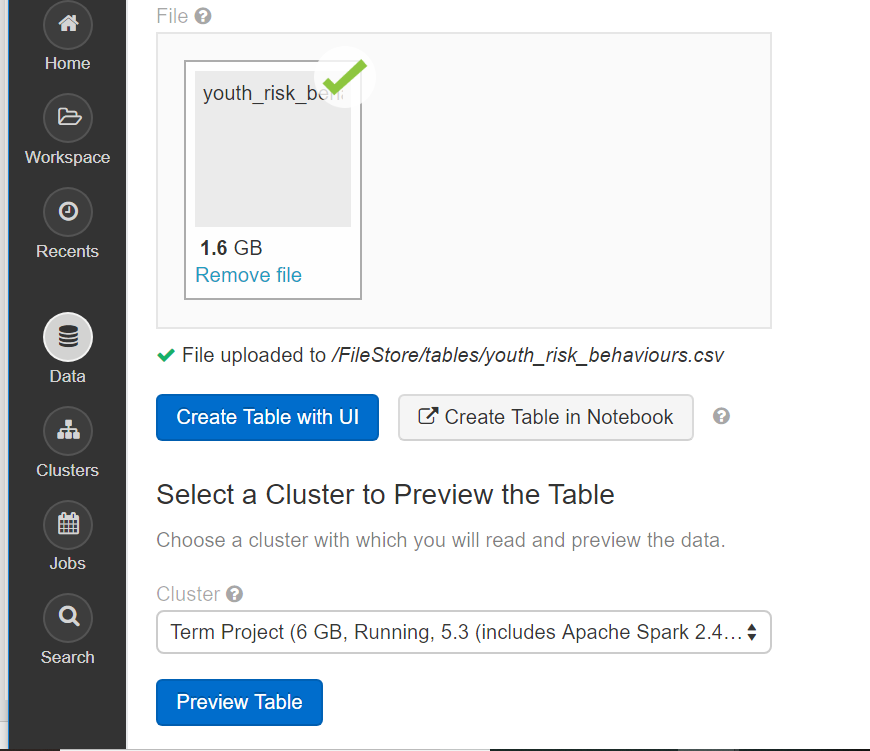




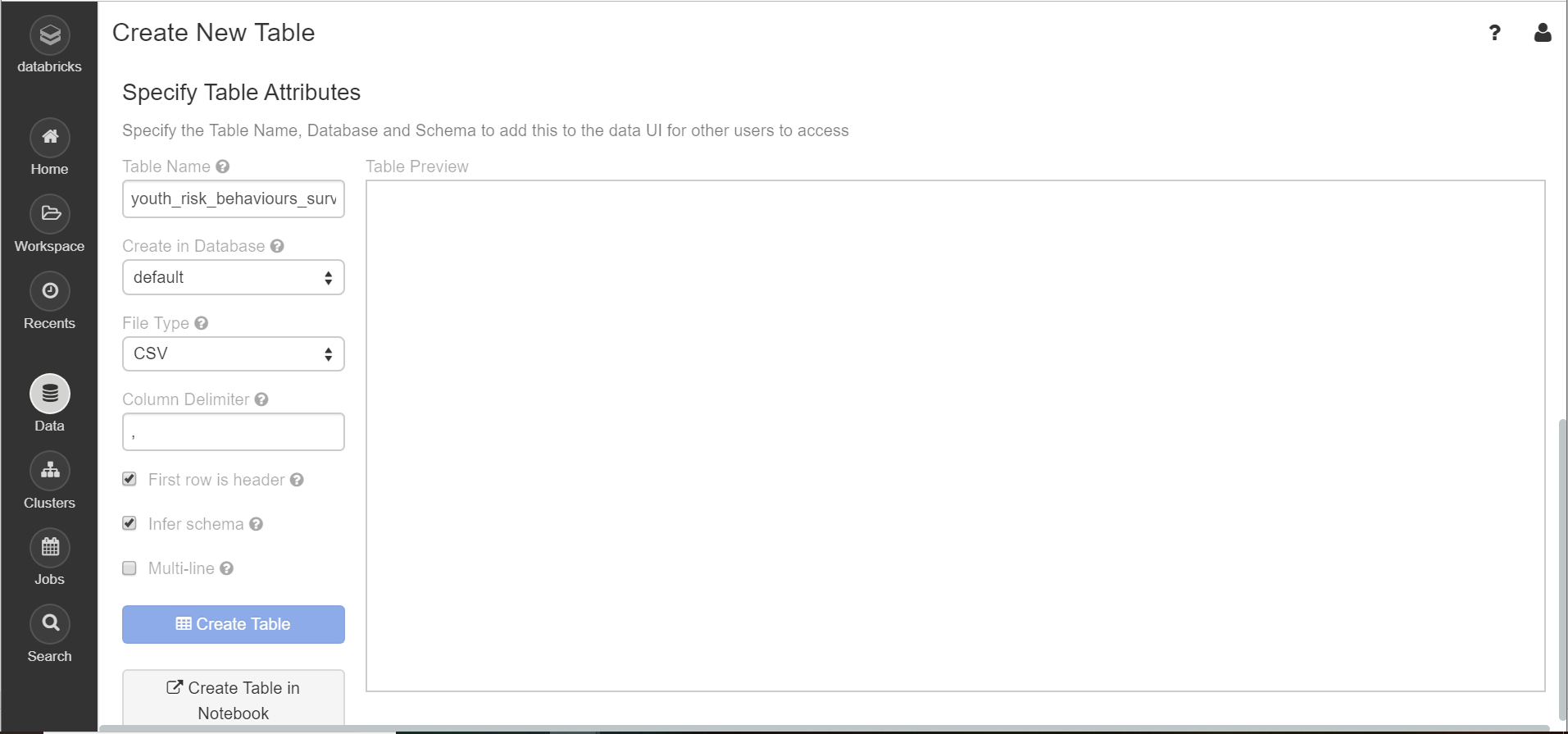
1. Click on databricks -> create table -> Upload file



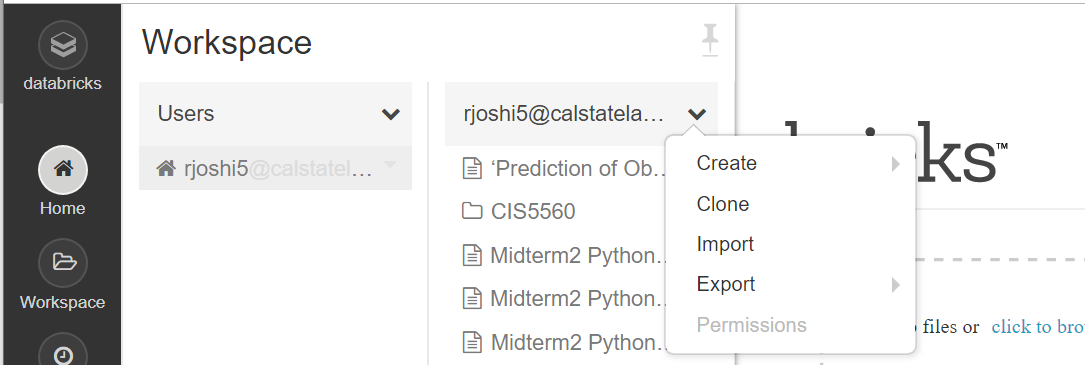
1. Create Table with UI -> Select previously created cluster -> Preview table



1. Give tablename as ‘youth\_risk\_behaviours\_surveillance\_csv’, tick options ‘first row as header’ and ‘infer schema’. Now click on ‘Create Table’



1. Now goto ‘Home’ option on the top-left side of pane and import .ipydb file

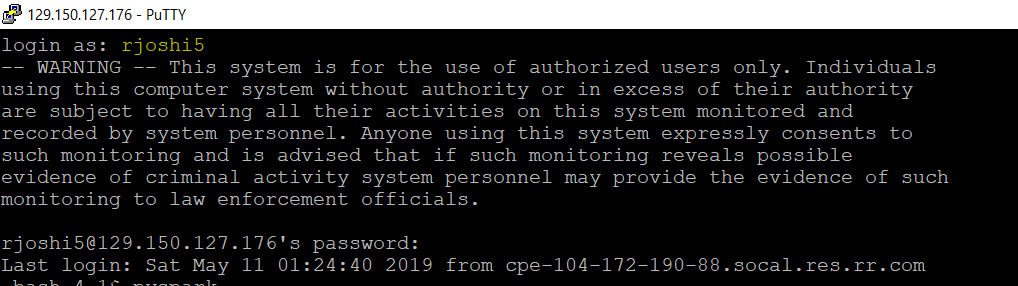


1. Save and run the file in order to view results.

Step 5: Implementation on Oracle CLI

1. Open Oracle CLI using putty by providing IP address of Oracle cluster and Port 22. Then provide username and password



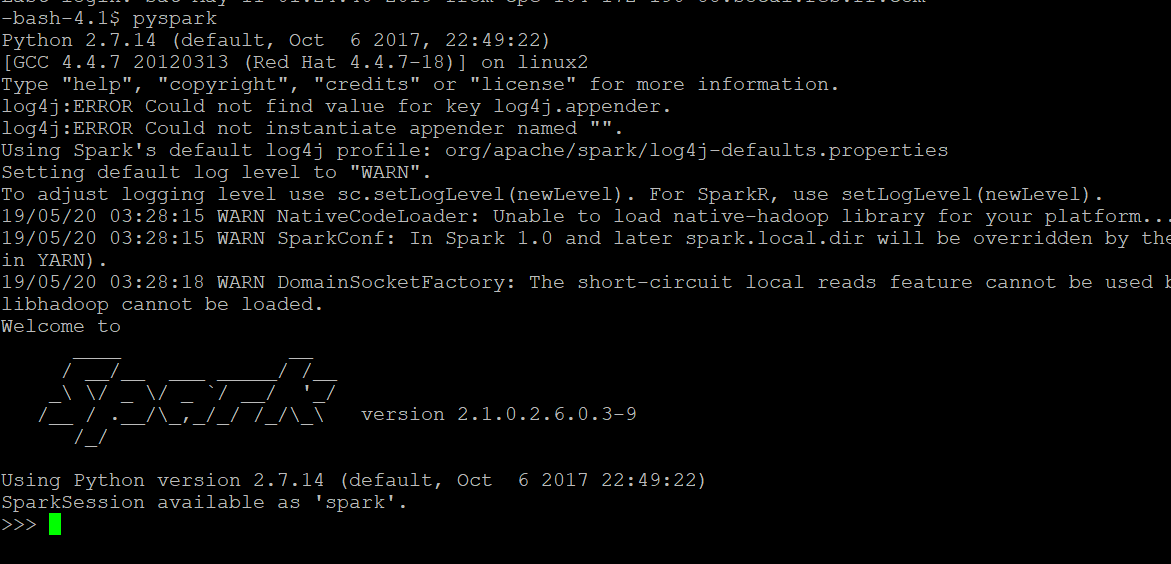


1. Upload file on hdfs

-bash-4.1$ hdfs dfs -put youth\_risk\_behaviours.csv .

1. Start spark session with command pyspark

-bash-4.1$ pyspark



1. Import all required the libraries

>>> import numpy as np

>>>

>>> from pyspark.sql.types import \*

>>> from pyspark.sql.functions import \*

>>> from pyspark.sql import functions as F

>>> from pyspark.sql import SQLContext

>>> sqlContext = SQLContext(sc)

>>> from pyspark.ml import Pipeline

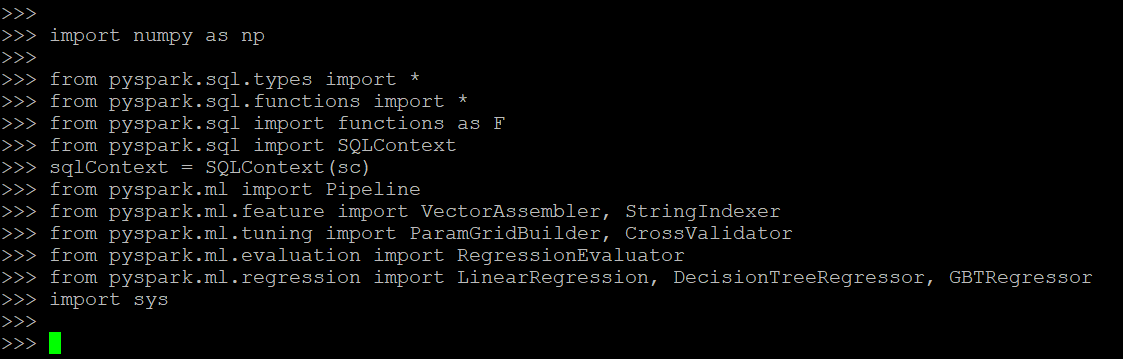
>>> from pyspark.ml.feature import VectorAssembler, StringIndexer

>>> from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

>>> from pyspark.ml.evaluation import RegressionEvaluator

>>> from pyspark.ml.regression import LinearRegression, DecisionTreeRegressor, GBTRegressor

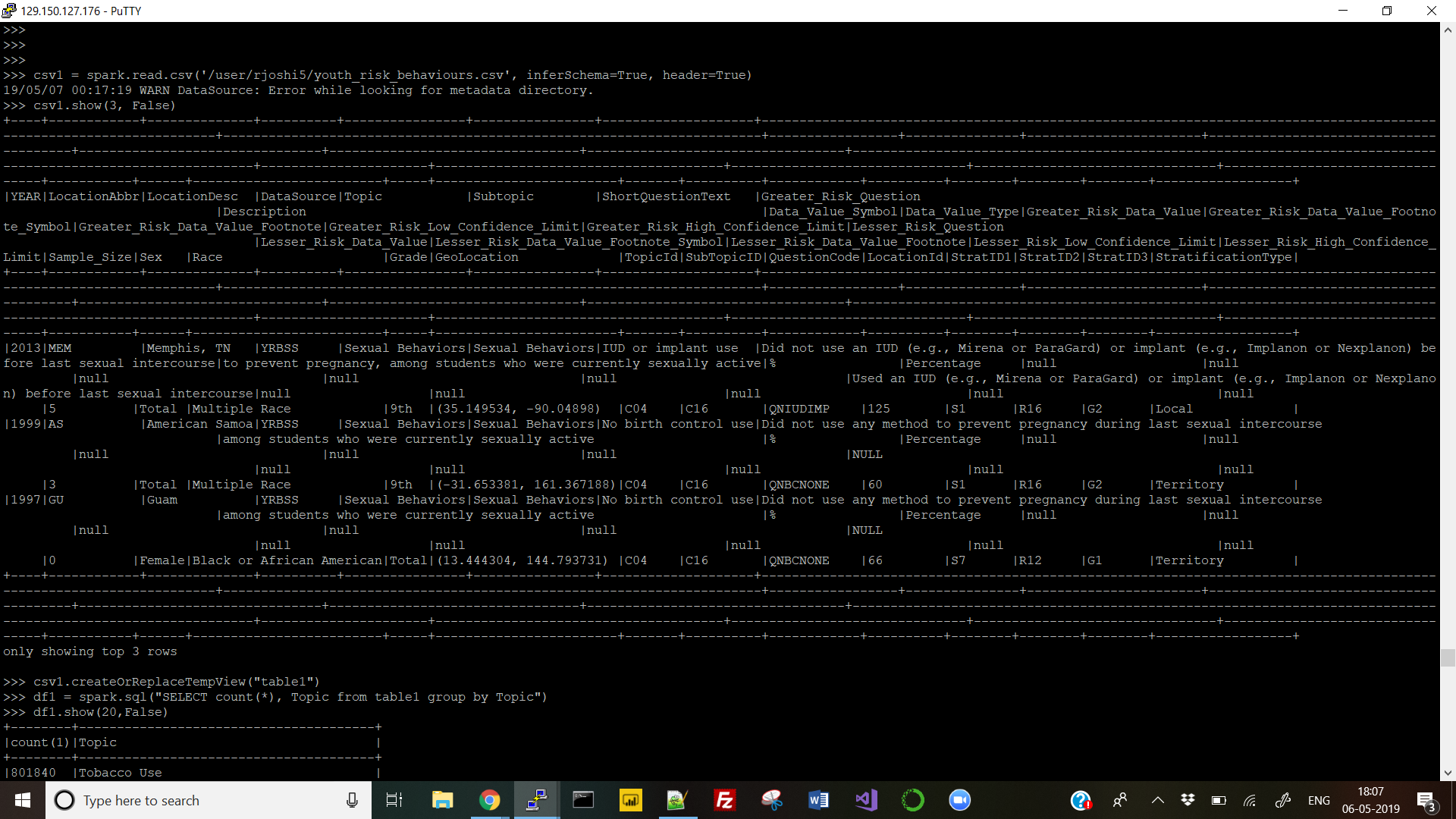
>>> import sys



1. Read the csv file that is loaded in hdfs

>>> csv1 = spark.read.csv('/user/rjoshi5/youth\_risk\_behaviours.csv', inferSchema=True, header=True)

>>> csv1.show(3, False)

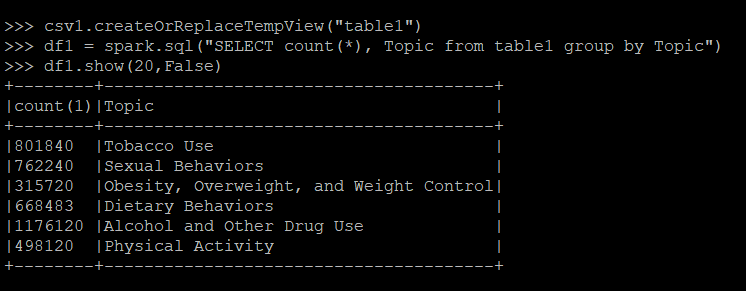


1. Create temp table from the read table and check how many rows are present in the table with respect to ‘Obesity Overweight’ for model to learn

>>> csv1.createOrReplaceTempView("table1")

>>> df1 = spark.sql("SELECT count(\*), Topic from table1 group by Topic")

>>> df1.show(20, False)

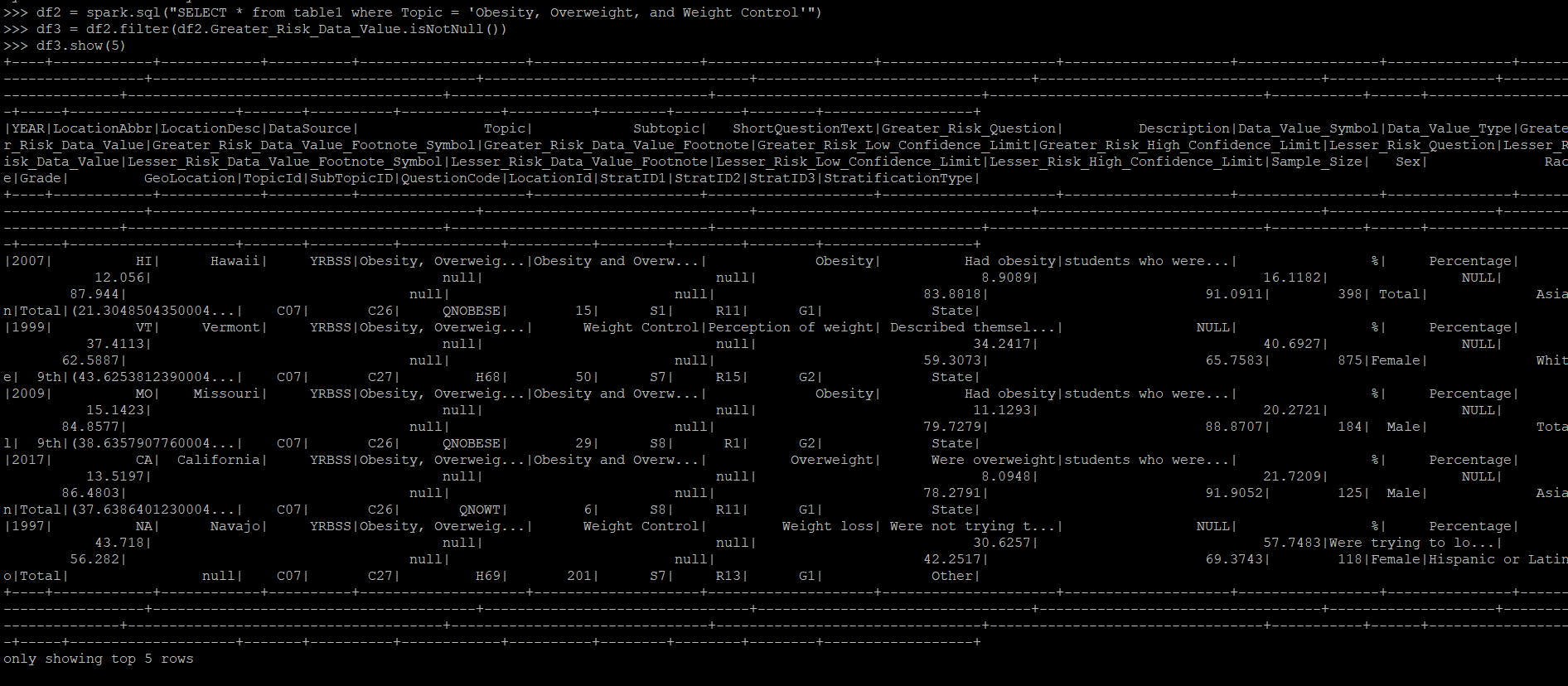


1. Create a new dataframe df2 with only the relevant rows of ‘Obesity, Overweight and Weight Control’. Further remove rows that have null values.

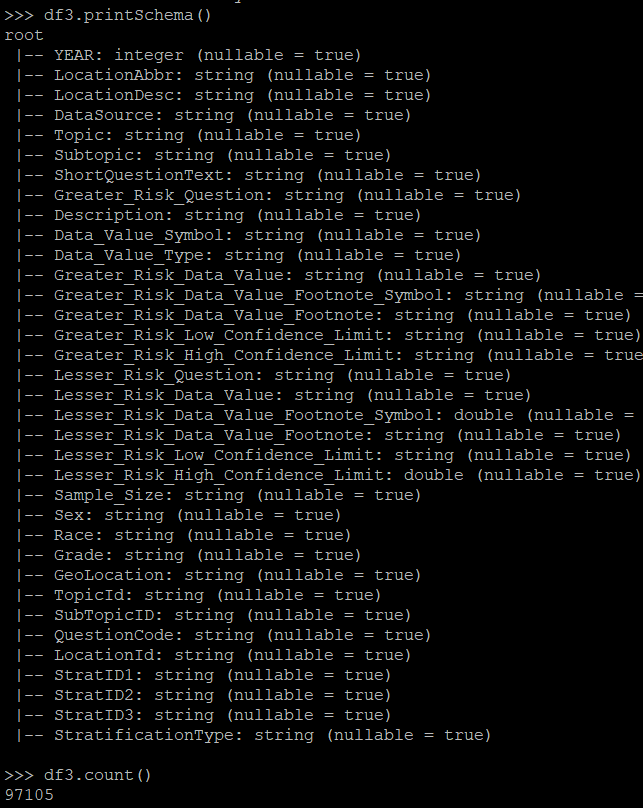
>>> df2 = spark.sql("SELECT \* from table1 where Topic = 'Obesity, Overweight, and Weight Control'")

>>> df3 = df2.filter(df2.Greater\_Risk\_Data\_Value.isNotNull())

>>> df3.show(5)



1. Check datatypes of all columns using printSchema function and count the number of rows available.



1. Choose columns that will be used to trian and test the model. Cast operator changes label data type from string to double.

>>> data = df3.select("YEAR", "LocationDesc", "Sex", "Race", "Grade", "SubTopicID", "QuestionCode", "LocationId", "StratID1", col("Greater\_Risk\_Data\_Value").cast("double").alias("label"))

>>> data = StringIndexer(inputCol='LocationDesc', outputCol='LocationDesc'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='Sex', outputCol='Sex'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='Race', outputCol='Race'+"\_index").fit(data).transform(data)

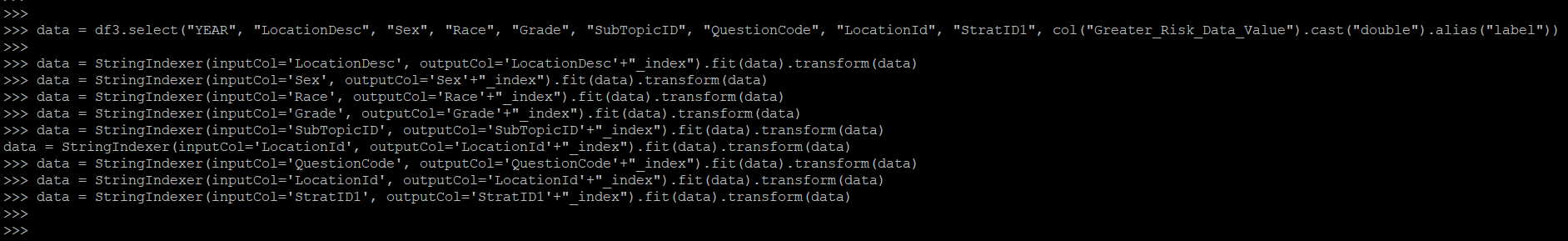
>>> data = StringIndexer(inputCol='Grade', outputCol='Grade'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='SubTopicID', outputCol='SubTopicID'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='QuestionCode', outputCol='QuestionCode'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='LocationId', outputCol='LocationId'+"\_index").fit(data).transform(data)

>>> data = StringIndexer(inputCol='StratID1', outputCol='StratID1'+"\_index").fit(data).transform(data)



1. Using random split, divide the data to be used as test and train in 70:30 for linear regression and decision tree evaluators used earlier in databricks.

>>> splits = data.randomSplit([0.7, 0.3])

>>>

>>> lr\_train = splits[0]

>>> lr\_test = splits[1].withColumnRenamed("label", "trueLabel")

>>>

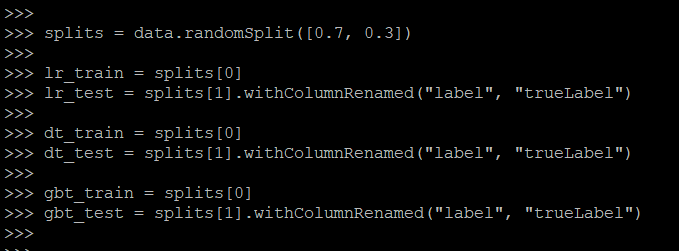
>>> dt\_train = splits[0]

>>> dt\_test = splits[1].withColumnRenamed("label", "trueLabel")

>>>

>>> gbt\_train = splits[0]

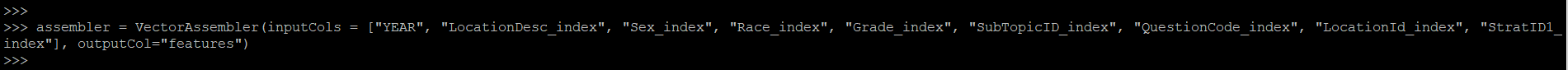
>>> gbt\_test = splits[1].withColumnRenamed("label", "trueLabel")



1. Below set of commands uses vector assembler to group columns as ‘features’. This ‘features’

column is further used in regression algorithms.

>>> assembler = VectorAssembler(inputCols = ["YEAR", "LocationDesc\_index", "Sex\_index", "Race\_index", "Grade\_index", "SubTopicID\_index", "QuestionCode\_index", "LocationId\_index", "StratID1\_index"], outputCol="features")



1. Linear Regression Implementation

>>> lr = LinearRegression(featuresCol = 'features', labelCol='label', maxIter=12345, regParam=0.4, elasticNetParam=0.7)

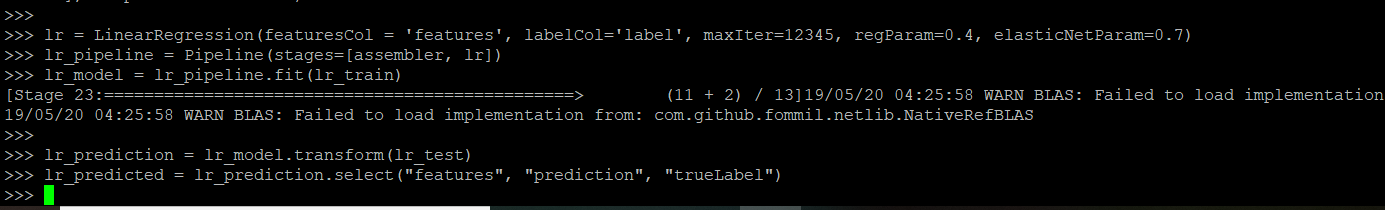
>>> lr\_pipeline = Pipeline(stages=[assembler, lr])

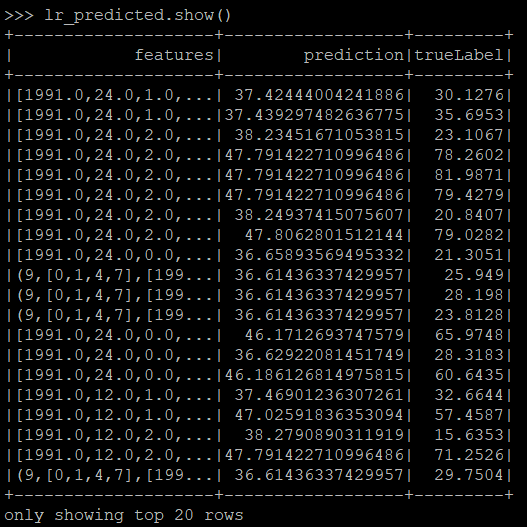
>>> lr\_model = lr\_pipeline.fit(lr\_train)

>>> lr\_prediction = lr\_model.transform(lr\_test)

>>> lr\_predicted = lr\_prediction.select("features", "prediction", "trueLabel")

>>> lr\_predicted.show()





1. Calculate and print Root mean square error and Accuracy for linear regression

>>> lr\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")

lr\_accuracy = lr\_evaluator.evaluate(lr\_prediction, {lr\_evaluator.metricName: "r2"})

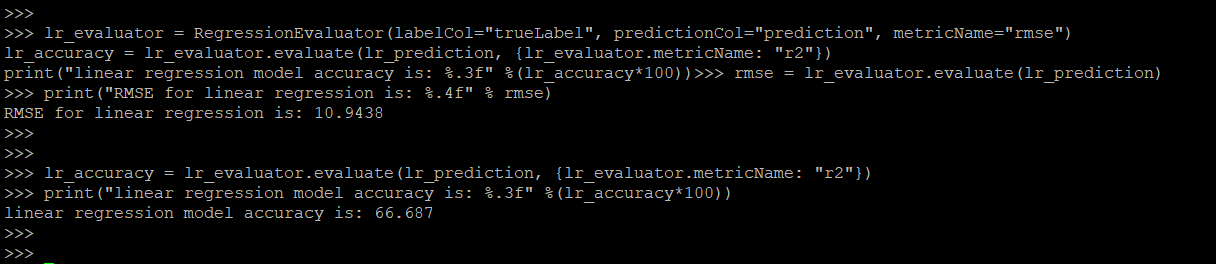
print("linear regression model accuracy is: %.3f" %(lr\_accuracy\*100))>>> rmse = lr\_evaluator.evaluate(lr\_prediction)

>>> print("RMSE for linear regression is: %.4f" % rmse)

>>>

>>> lr\_accuracy = lr\_evaluator.evaluate(lr\_prediction, {lr\_evaluator.metricName: "r2"})

>>> print("linear regression model accuracy is: %.3f" %(lr\_accuracy\*100))



1. Decision Tree application using cross validator

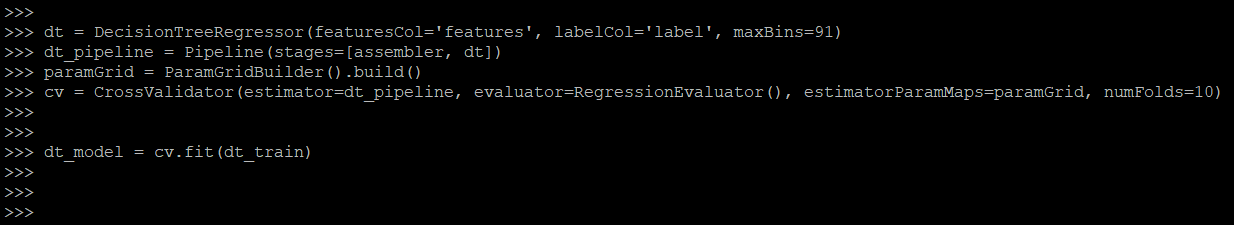
>>> dt = DecisionTreeRegressor(featuresCol='features', labelCol='label', maxBins=91)

dt\_model = cv.fit(dt\_train)>>> dt\_pipeline = Pipeline(stages=[assembler, dt])

>>> paramGrid = ParamGridBuilder().build()

>>> cv = CrossValidator(estimator=dt\_pipeline, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid, numFolds=10)

>>> dt\_model = cv.fit(dt\_train)

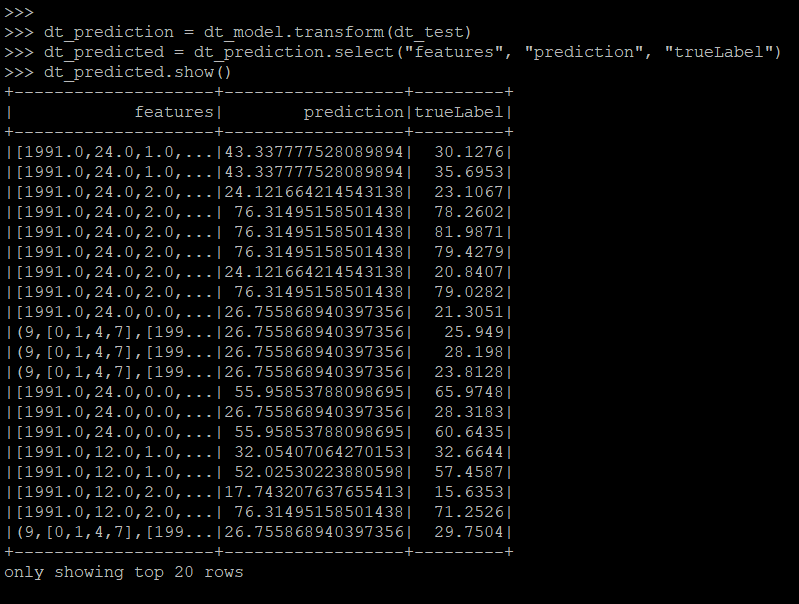


1. Compare predicted and label values

>>> dt\_prediction = dt\_model.transform(dt\_test)

>>> dt\_predicted = dt\_prediction.select("features", "prediction", "trueLabel")

>>> dt\_predicted.show()



1. Evaluate and print RMSE and Accuracy for Decision Tree Regression

>>> dt\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")

>>> dt\_rmse = dt\_evaluator.evaluate(dt\_prediction)

>>> print ("Root Mean Square Error (RMSE):", dt\_rmse)

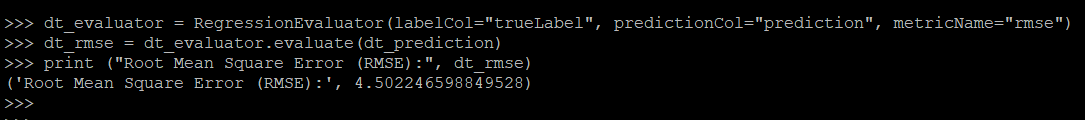
>>>

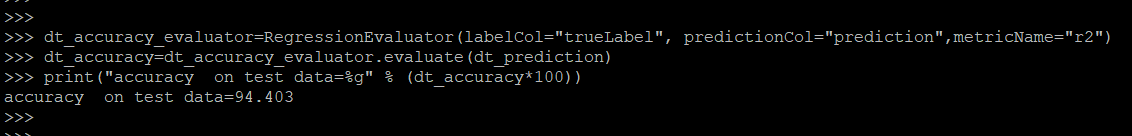
>>>

>>> dt\_accuracy\_evaluator=RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction",metricName="r2")

>>> dt\_accuracy=dt\_accuracy\_evaluator.evaluate(dt\_prediction)

>>> print("accuracy on test data=%g" % (dt\_accuracy\*100))





1. Gradient Boosted Decision Tree

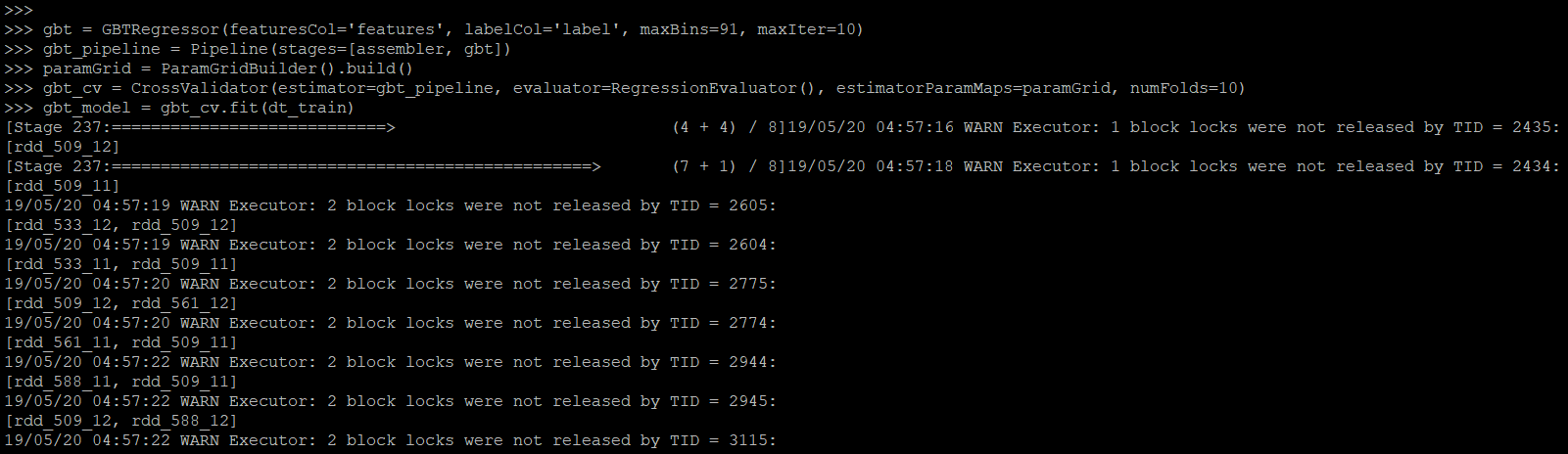
>>> gbt = GBTRegressor(featuresCol='features', labelCol='label', maxBins=91, maxIter=10)

>>> gbt\_pipeline = Pipeline(stages=[assembler, gbt])

>>> paramGrid = ParamGridBuilder().build()

>>> gbt\_cv = CrossValidator(estimator=gbt\_pipeline, evaluator=RegressionEvaluator(), estimatorParamMaps=paramGrid, numFolds=10)

>>> gbt\_model = gbt\_cv.fit(dt\_train)

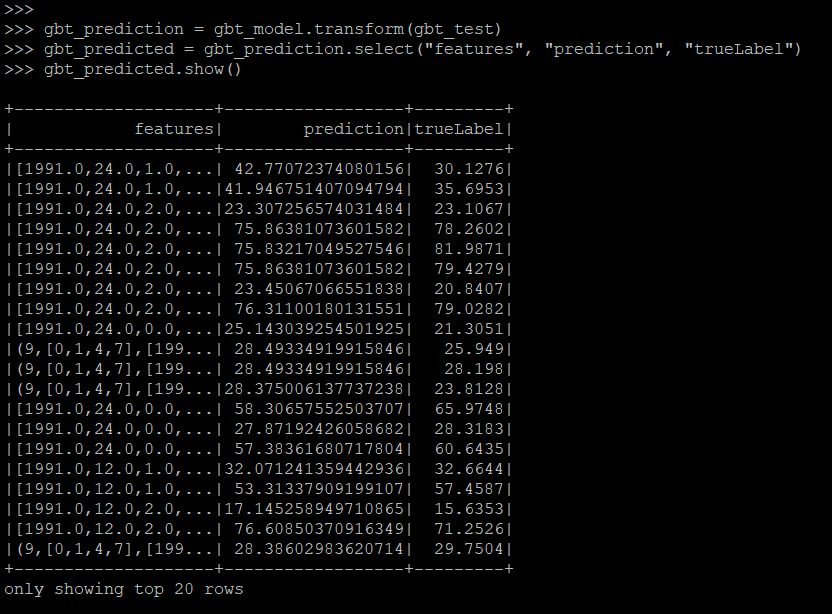


1. Compare predicted and actual values of the label

>>> gbt\_prediction = gbt\_model.transform(gbt\_test)

>>> gbt\_predicted = gbt\_prediction.select("features", "prediction", "trueLabel")

>>> gbt\_predicted.show()



1. Evaluate and print RMSE and Accuracy for gradient boosted decision tree.

>>> gbt\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")

>>> gbt\_rmse = gbt\_evaluator.evaluate(gbt\_prediction)

>>> print ("Root Mean Square Error (RMSE):", gbt\_rmse)

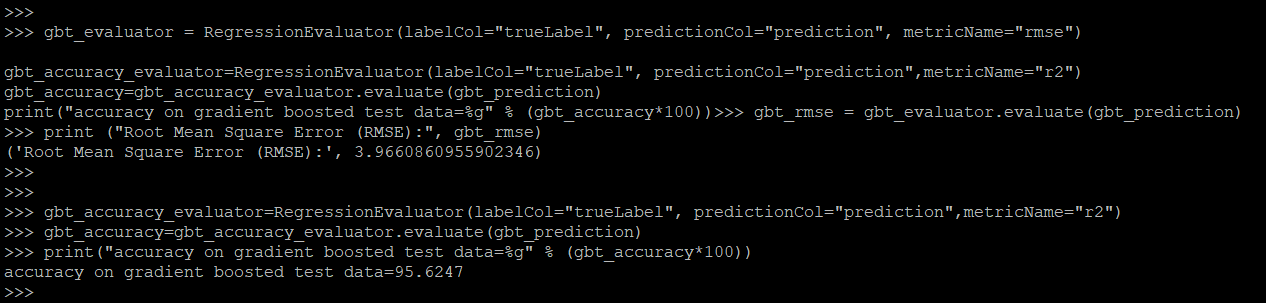
>>>

>>>

>>> gbt\_accuracy\_evaluator=RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction",metricName="r2")

>>> gbt\_accuracy=gbt\_accuracy\_evaluator.evaluate(gbt\_prediction)

>>> print("accuracy on gradient boosted test data=%g" % (gbt\_accuracy\*100))



References

* 1. URL of Data Source, <https://www.kaggle.com/raylo168/dash-yrbss-hs-2017>
  2. Github - <https://github.com/rjoshi5/Obesity-Risk-Level-Among-Youth>
  3. Published Azure ML model link - <https://gallery.cortanaintelligence.com/Experiment/Prediction-of-Obesity-Risk-Level-Among-Youth>
  4. <https://www.cdc.gov/healthyyouth/data/yrbs/overview.htm>
  5. <https://spark.apache.org/docs/2.1.0/mllib-classification-regression.html>
  6. <https://towardsdatascience.com/building-a-linear-regression-with-pyspark-and-mllib-d065c3ba246a>
  7. <https://blog.epigno.systems/2018/02/18/machine-learning-with-pyspark-linear-regression/>