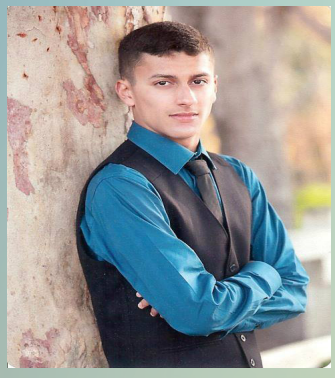
Predictions in a Mining Process

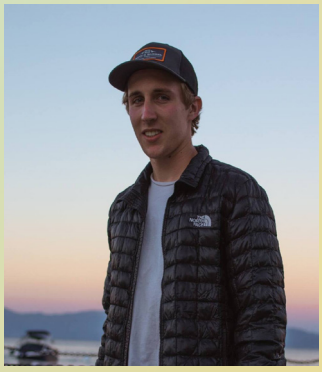
Team Members

Geovane Marquez

Ronak Parikh



Anthony Castillo



Sam Sheridan

Introduction

The target is to predict the % of Silica in the end of the process, which is the concentrate of iron ore and its impurity (which is the % of Silica).

The dataset in question specifically has the percentage of silica impurity as the response variable. Thus, the author of the dataset intended for users to research how much silica impurity is in the iron ore so manufacturing processes can be improved.

We perform random forest, decision trees, and gradient boosting tree algorithms to make the most accurate predictions.

DATA I

With over 737,453 records and 24 features, the analysis will consist of using only 22 of the features. The features 'date' and '% Iron Feed' will be excluded from the analysis. The feature of interest is the %
Silica Feed in apropos to the others.

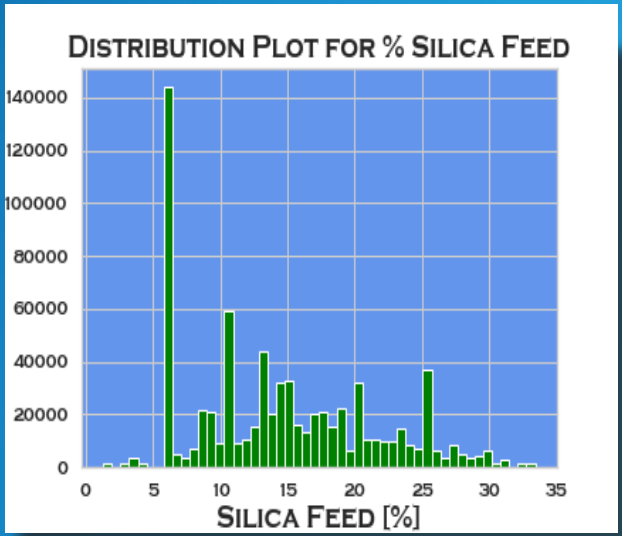
Data Source: https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process

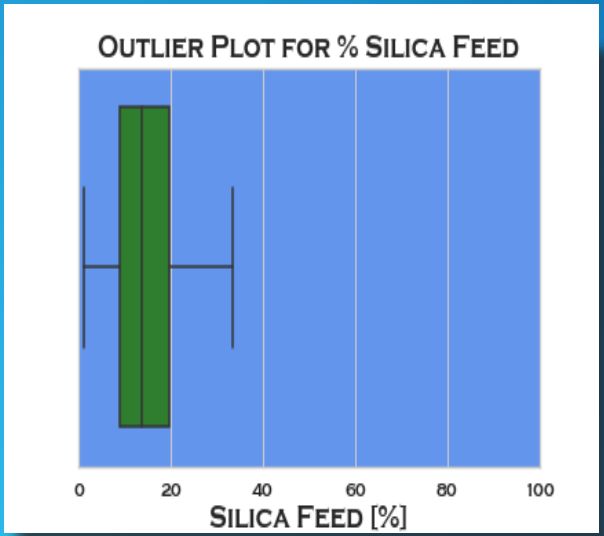
Features	Data Type
date	string
% iron feed	double
% silica feed	double
starch flow	double
amina flow	double
ore pulp flow	double
ore pulp pH	double
ore pulp density	double
Flotation Column 01 Air Flow	double
Flotation Column 02 Air Flow	double
Flotation Column 03 Air Flow	double
Flotation Column 04 Air Flow	double
Flotation Column 05 Air Flow	double
Flotation Column 06 Air Flow	double
Flotation Column 07 Air Flow	double
Flotation Column 01 Level	double
Flotation Column 02 Level	double
Flotation Column 03 Level	double
Flotation Column 04 Level	double
Flotation Column 05 Level	double
Flotation Column 06 Level	double
Flotation Column 07 Level	double
iron concentrate	double

The Summary Statistics for the % of Silica at the end of the process

Count	Min	Max	Average	STDDEV
737,453	1.31%	33.40%	14.65%	6.81%

DATA II





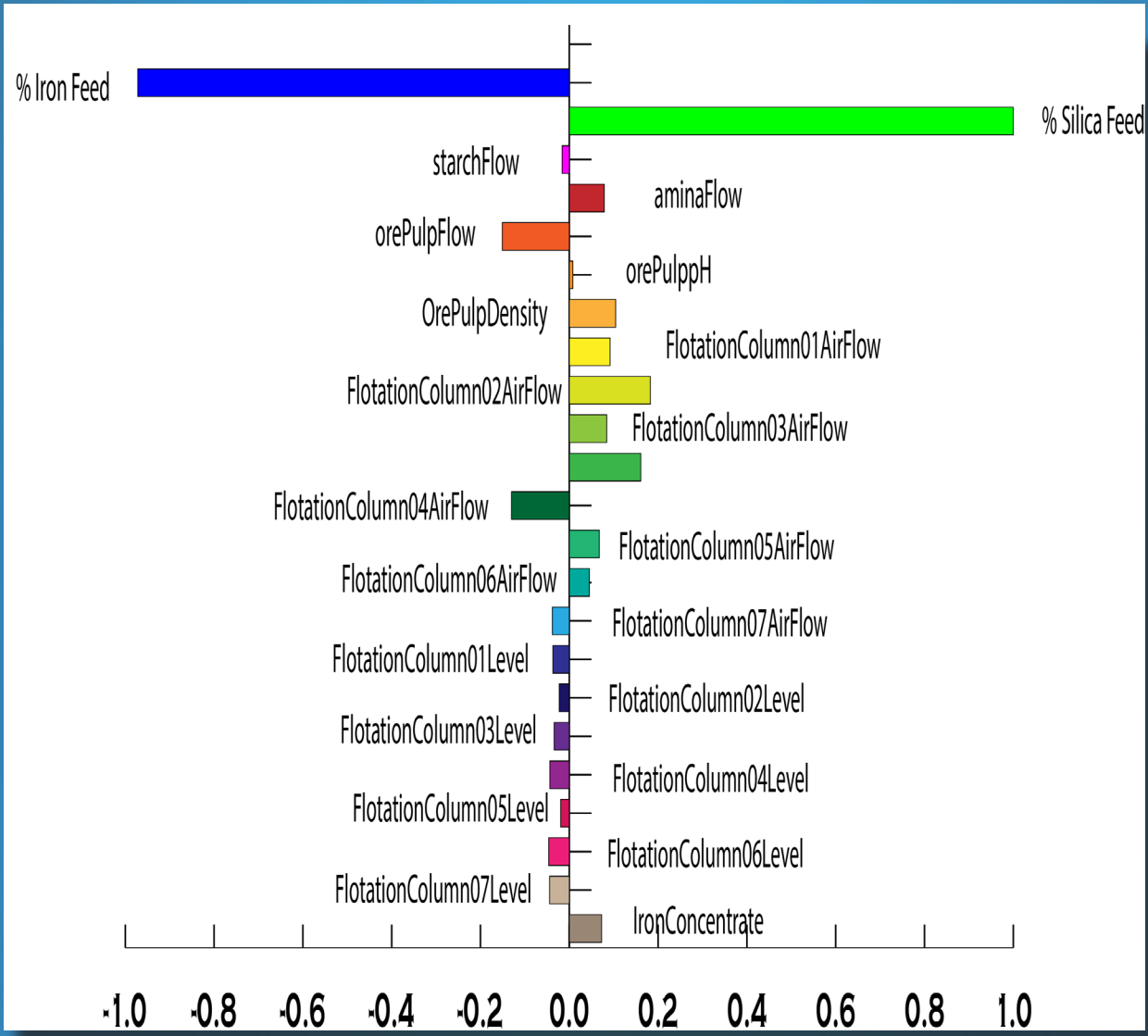
The target variable '% silica feed' has a range of percentages betwee 1.31 and 33.40 for the amount of silica found at the end of the mining process.
There is a high count around 6% of Silica feed.

Exploring further, no outliers were detected outside the minimum and maximum ranges of percentages. The data is consistent, and appears to be more dense between 10% and 20% range. Earlier, the average statistic was found to be 14.65%.

In the following pages, we explore the relationships that the target variable has with the other features. This becomes helpful in eliminating unneeded features.

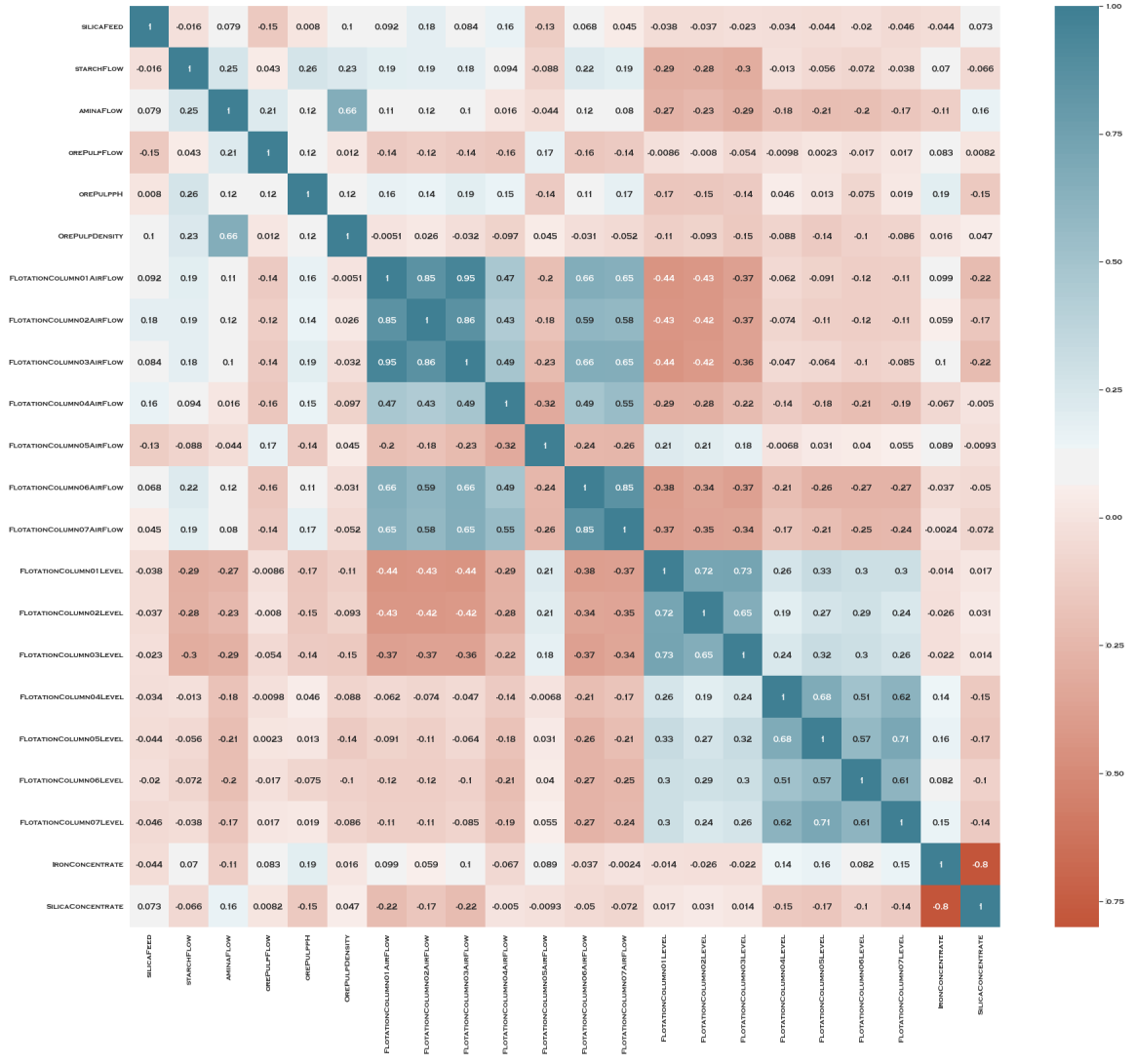
Mining Data I

Correlation built from the response: % Silica Feed



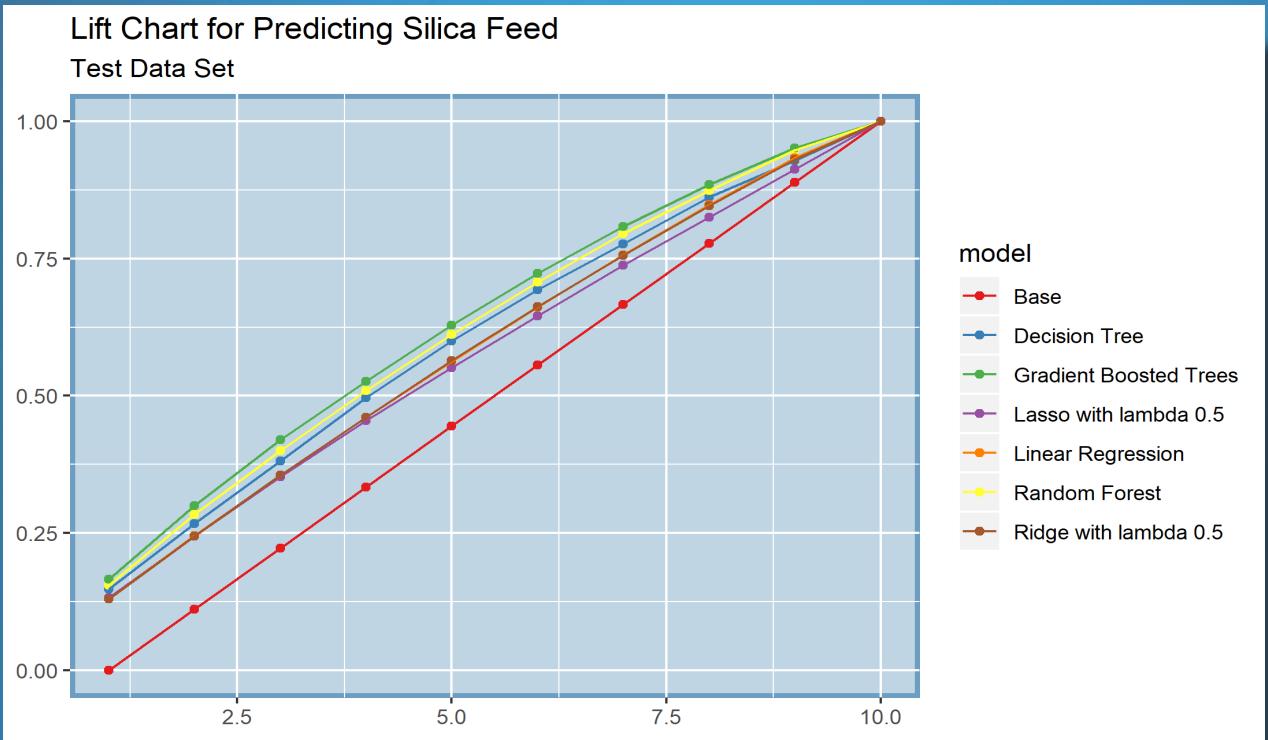
Iron Feed and silica feed are highly correlated. In this study, we are only interested in exploring the predictions for % silica feed, thus we removed % iron feed out of the model. In the next page, we explore these relations with a correlation heat matrix.

Mining Data II

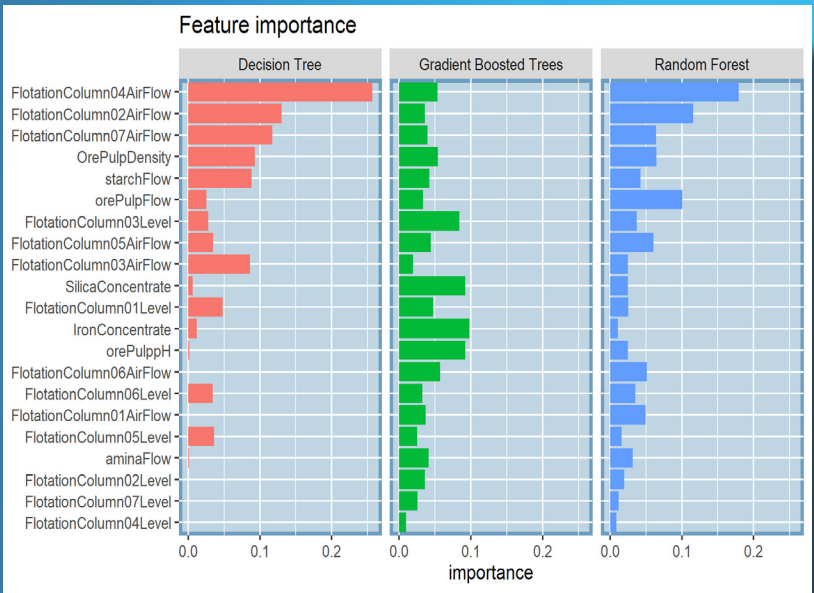


The plot shows the correlations between the features. We use this to find out which features affect % Silica Feed the most. The plot gives us a great overview of the feature correlation relationships.

Mining Data III

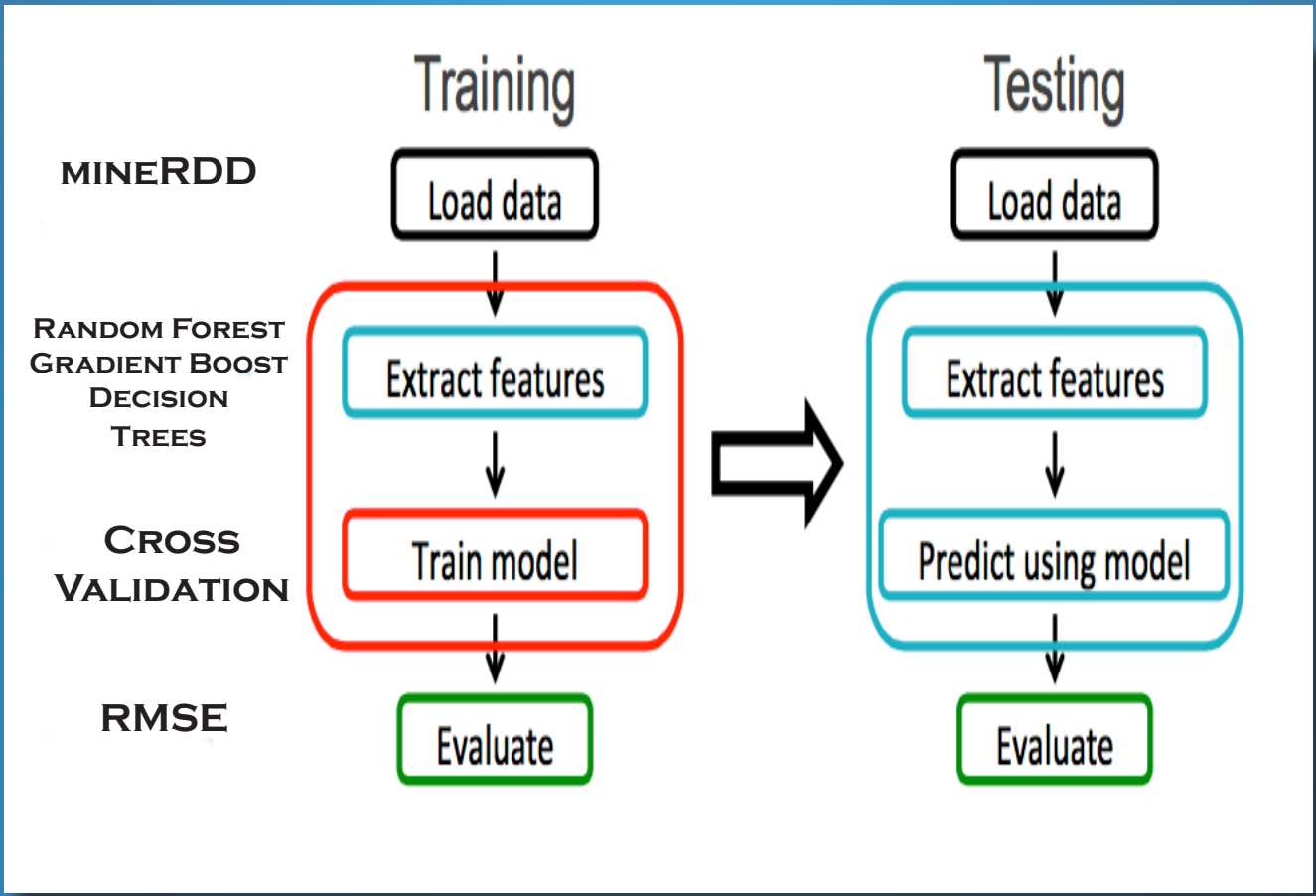


After splitting the data: 75% Training, 25% Test. several machine learning models were constructed with the training data and tested against the test data, as shown in the plot above. The plot above tells the story that the best predictions will come from gradient boosted trees, Random Forest, and Decision trees. Thus, fine-tuned models will be focused on these three.



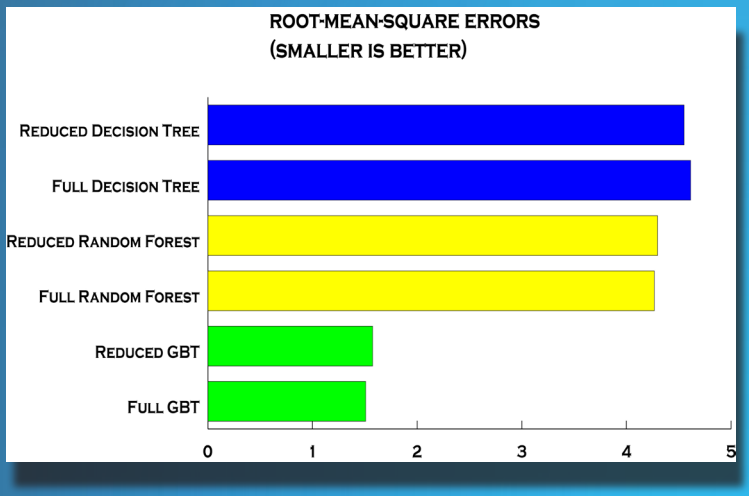
Based on the plot to the left, a reduced model will be constructed. The reduced model will then be tested against the full model. By observation, the least important features are: aminaFlow, and flotation Column levels 02,04,07. These three will be removed, and the rest of the features will be kept in the reduced model.

Pipelines



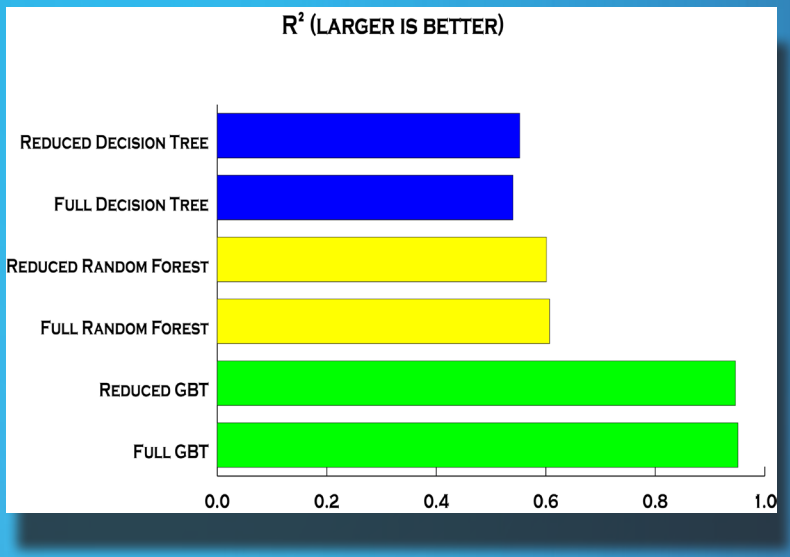
The graphic above illustrates the process that we took to generate a predictive fitted model. First we loaded the data into pyspark as an RDD. Secondly, we fitted three submodels using gradient boosting, decision, and random forest trees. Thirdly, we performed cross validations to generate the best parameter metrics. Lastly, the errors using RMSE were plotted against each other for comparison, and the model with the lowest error was chosen to be used to predict the % of silica Feed.

Performance and Validation



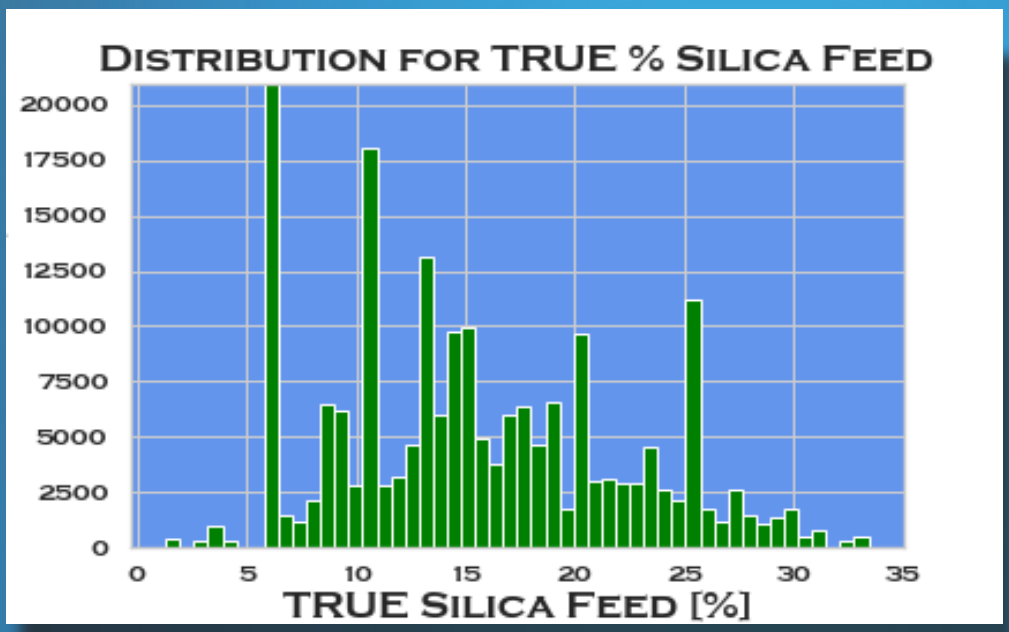
After performing cross validations using paragrid, as predicted by the earlier lift chart, the model with the smallest root-mean-square error (rmse) was the gradient boosting tree model. The full model was infinitesimally better than the reduced model when comparing their rmses.

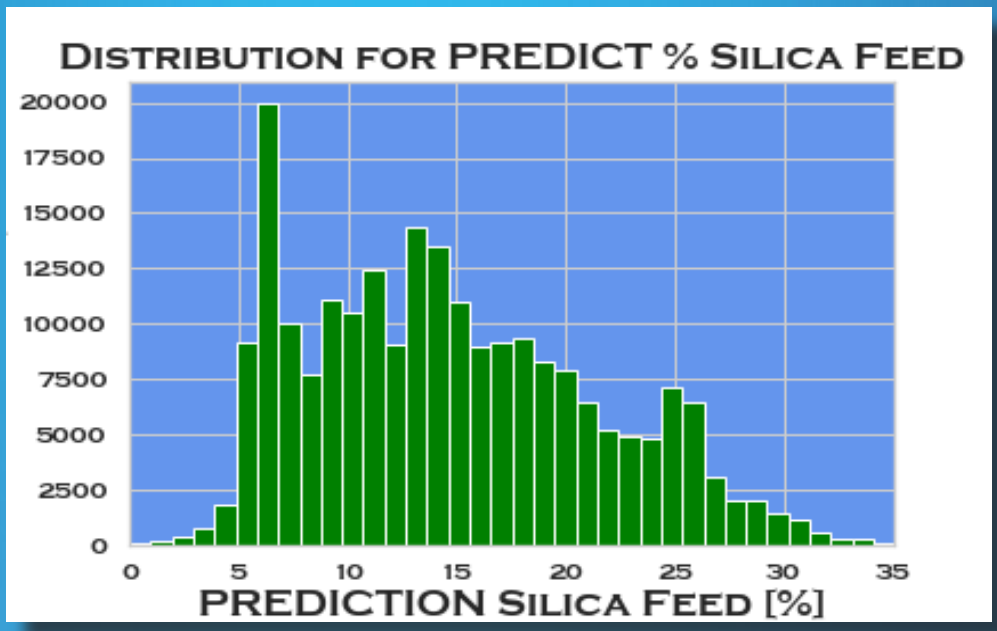
The proportion of variance explained by the model was highest in the gradient boosting tree models. Similarly, the full GBT model was infinitesimally better than the reduced model when comparing their R-squared



In conclusion, the model we chose to conduct the predictions for % silica Feed was determined to be the full model trained under a gradient boosting trees algorithm. The highly computational expensive model consisted of an ensemble of max depth of 10 of each decision tree. As well as, over 200 iterations for the number of trees in each boosting tree ensemble.

Predictions





Above we see the original true data for % silica Feed. Right below this graph we have the newly predicted data for % silica Feed. The shape of the prediction plot has slight skewness to it, however, the distribution looks nicely consistent.

Lastly, the predictions were constructed from a gradient boosting tree algorithm that utilized all the features (except date and % Iron Feed), and used % silica Feed as the Target. This trained model can help research how much silica impurity is in the iron ore so manufacturing processes can be improved.

SparklyR Appendix

Mining Project - PSTAT 135

```
library(sparklyr)
library(dplyr)
library(tidyverse)
```

```
spark_install(version='2.4')
sc <- spark_connect(master='local',version='2.4')
```

```
d <- spark_read_csv(sc, path='editedMine.csv', header=TRUE, infer_schema = TRUE)
```

```
mine_rdd = d
```

Feature Selection by CV

```
# source: https://www.eddjberry.com/post/2018-12-12-sparklyr-feature-selection/
get_feature_cols <- function(tbl, response, exclude = NULL) {

  # column names of the data
  columns <- colnames(tbl)

  # exclude the response/outcome variable and
  # exclude from the column names
  columns[!(columns %in% c(response, exclude))]
}
```

```
feature_cols <- get_feature_cols(mine_rdd, response ='silicaFeed')
#feature_cols
```

Split data

```
ds <- sdf_random_split(mine_rdd, train=0.75, test=0.25, seed=123)
names(ds)
```

```
train_data <- ds$train
test_data <- ds$test
```

Full Model

```
ml_formula <- formula(silicaFeed ~.)
```

Reduced Model

```
red.mod <- formula(silicaFeed ~ starchFlow+orePulpFlow+orePulppH+
                     OrePulpDensity+FlotationColumn01AirFlow +
                     FlotationColumn02AirFlow+FlotationColumn03AirFlow+
                     FlotationColumn04AirFlow+FlotationColumn05AirFlow+
                     FlotationColumn06AirFlow+FlotationColumn07AirFlow+
                     FlotationColumn01Level+FlotationColumn03Level+
                     FlotationColumn05Level+FlotationColumn06Level+
                     IronConcentrate+SilicaConcentrate)
```

```
reducedRDD <- select(mine_rdd, -c(aminaFlow,FlotationColumn02Level,
                                  FlotationColumn04Level,FlotationColumn07Level))
```

Pre-Tuning ML Models

```
# Decision Tree
ml_dt <- ml_decision_tree(train_data, ml_formula)

# Random Forest
ml_rf <- ml_random_forest(train_data, ml_formula)

# Gradient Boosted Tree
ml_gbt <- ml_gradient_boosted_trees(train_data, ml_formula)

# linear regression
ml_linreg <- ml_linear_regression(train_data, ml_formula)

# lasso
ml_lasso5 <- ml_linear_regression(train_data, formula = ml_formula,
                                  elastic_net_param=1, reg_param = 0.5)

# Ridge
ml_ridge5 = ml_linear_regression(train_data, ml_formula,
                                 elastic_net_param =0, reg_param = 0.5)
```

Validation

```
ml_models <- list(
  "Decision Tree" = ml_dt,
  "Random Forest" = ml_rf,
  "Gradient Boosted Trees" = ml_gbt,
  "Linear Regression" = ml_linreg,
  "Lasso with lambda 0.5" = ml_lasso5,
  "Ridge with lambda 0.5" = ml_ridge5
)

# Create a function for scoring
```

```
score_test_data <- function(model, data = test_data){
  pred <- sdf_predict(data, model)
  select(pred, silicaFeed, prediction)
}

# Score all the models
ml_score <- map(ml_models, score_test_data)
```

LIFT CHART

```
# Lift function
calculate_lift <- function(scored_data) {
  scored_data %>%
    mutate(bin = ntile(desc(prediction), 10)) %>%
    group_by(bin) %>%
    summarize(count = sum(silicaFeed)) %>%
    mutate(prop = count / sum(count)) %>%
    arrange(bin) %>%
    mutate(prop = cumsum(prop)) %>%
    select(-count) %>%
    collect() %>%
    as.data.frame()
}

# Initialize results
ml_gains <- tibble(
  bin = seq(from = 1, to = 10),
  prop = seq(0, 1, len = 10),
  model = "Base"
)

# Calculate lift
for(i in names(ml_score)){
  ml_gains <- ml_score[[i]] %>%
    calculate_lift %>%
    mutate(model = i) %>%
    bind_rows(ml_gains, .)
}

# Plot results
myplot = ggplot(ml_gains, aes(x = bin, y = prop, color = model)) +
  geom_point() +
  geom_line() +
  scale_color_brewer(palette = 'Set1') +
  labs(title = "Lift Chart for Predicting Silica Feed",
       subtitle = "Test Data Set",
       x = NULL,
       y = NULL) +
  theme(
  panel.background = element_rect(fill = "#BFD5E3", colour = "#6D9EC1",
                                  size = 2, linetype = "solid"),
```

```
  panel.grid.major = element_line(size = 0.5, linetype = 'solid',
                                  colour = "white"),
  panel.grid.minor = element_line(size = 0.25, linetype = 'solid',
                                  colour = "white")
  )
myplot
```

FEATURE IMPORTANCE CHART

```
# Initialize results
feature_importance <- tibble()

# Calculate feature importance
for(i in c("Decision Tree", "Random Forest", "Gradient Boosted Trees")){
  feature_importance <- ml_tree_feature_importance(ml_models[[i]]) %>%
    mutate(Model = i) %>%
    rbind(feature_importance, .)
}

# Plot results
feature_importance %>%
  ggplot(aes(reorder(feature, importance), importance, fill = Model)) +
  facet_wrap(~Model) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Feature importance",
       x = NULL) +
  theme( legend.position ="none",
  panel.background = element_rect(fill = "#BFD5E3", colour = "#6D9EC1",
                                  size = 2, linetype = "solid"),
  panel.grid.major = element_line(size = 0.5, linetype = 'solid',
                                  colour = "white"),
  panel.grid.minor = element_line(size = 0.25, linetype = 'solid',
                                  colour = "white")
  )
```

Spark Appendix

In [1]:

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession, SQLContext
import os
import pandas as pd
import os.path
#sc = SparkContext.getOrCreate()
conf = SparkConf().set("spark.cores.max", "8") \
    .set("spark.driver.memory", "16g") \
    .set("spark.executor.memory", "16g") \
    .set("spark.executor.memory_overhead", "16g") \
    .set("spark.driver.maxResultsSize", "0")
sc = SparkContext(appName="mineProject", conf=conf)
spark = SparkSession(sc)
```

Read Data

In [2]:

```
filename = 'MiningProcess_Flotation_Plant_Database.csv'
mine_pd = pd.read_csv(filename, decimal=',')
```

In [3]:

```
newColumns = ["date", "ironFeed","silicaFeed","starchFlow","aminaFlow","orePulpFlow",\
              "orePulppH",'OrePulpDensity','FlotationColumn01AirFlow',\
               'FlotationColumn02AirFlow','FlotationColumn03AirFlow',\
             'FlotationColumn04AirFlow','FlotationColumn05AirFlow', 'FlotationColumn06AirFlow','Flot
ationColumn07AirFlow','FlotationColumn01Level','FlotationColumn02Level','FlotationColumn03Level','
FlotationColumn04Level','FlotationColumn05Level','FlotationColumn06Level','FlotationColumn07Level'
,'IronConcentrate','SilicaConcentrate']
```

PANDAS AND SNS STUFF

In [4]:

```
#rename columns
mine_pd.columns = newColumns
```

In [5]:

```
mine_pd.drop(['ironFeed', 'date'], axis=1, inplace=True)
```

In [7]:

```
# Export csv for sparklyr
mine_pd.to_csv('/Users/Geo/Desktop/project_photos/editedMine.csv', index = False)
```

In [35]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Histogram of SilicaFeed

In [ ]:

```
sns.set(style="whitegrid", font='Copperplate Gothic Bold', rc={'axes.facecolor':'cornflowerblue', '
figure.facecolor':'cornflowerblue'})
ax=sns.distplot(mine_pd['silicaFeed'], color='green', hist_kws={ "alpha": 1}, kde=False)
plt.title('Distribution Plot for % Silica Feed', fontsize=18)
plt.ylabel('Relative Frequency', fontsize=18)
plt.xlabel('Silica Feed [%]', fontsize=18)
plt.show()
```

Log Transform Silica Feed

In [ ]:

```
ax = sns.boxplot(x=mine_pd["silicaFeed"], color="forestgreen")
ax.set_title('Outlier Plot for % Silica Feed', fontsize=18)
ax.set_xlabel('Silica Feed [%]', fontsize=18)
ax.set(xlim=(0,100))
ax.figure.savefig("plot2.png")
```

Heatmap for Silica Feed

In [ ]:

```
plt.figure(figsize=(30, 25))
mine_heatmap = sns.heatmap(mine_pd.corr(),annot=True, cmap=sns.diverging_palette(20, 220, n=200),)
mine_heatmap.figure.savefig("plot3.png")
```

In [6]:

```
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
os.environ['ARROW_PRE_0_15_IPC_FORMAT'] = "1"
```

In [8]:

```
# Drop unneed columns
mine_pd.drop(['aminaFlow','FlotationColumn02Level','FlotationColumn04Level','FlotationColumn07Level
' ], axis=1, inplace=True)
```

In [7]:

```
mineRDD = spark.createDataFrame(mine_pd)
```

Data Dimensions

In [8]:

```
print((mineRDD.count(), len(mineRDD.columns)))
```

```
(737453, 22)
```

Data Types

In [ ]:

```
mineRDD.dtypes
```

The goal is to predict the % of Silica impurity in iron ore concentrate

Summary Statistics

In [15]:

```
from pyspark.sql.functions import count, mean, stddev, min, max
```

In [ ]:

```
mineRDD.select(count('silicaFeed'),mean('silicaFeed'),stddev('silicaFeed'), \
               min('silicaFeed'), max('silicaFeed')).show()
```

Building a Correlation Matrix

We want to omit any highly correlated or unneccessary features

In [14]:

```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
```

In [10]:

```
def correlation_matrix(df, corr_columns, method='pearson'):
    vector_col = "corr_features"
    assembler = VectorAssembler(inputCols=corr_columns, outputCol=vector_col)
    df_vector = assembler.transform(df).select(vector_col)
    matrix = Correlation.corr(df_vector, vector_col, method)

    result = matrix.collect()[0]["pearson({})".format(vector_col)].values
    return pd.DataFrame(result.reshape(-1, len(corr_columns)), columns=corr_columns, index=corr_col
umns)
```

In [ ]:

```
correlation_matrix(mineRDD, mineRDD.columns[1:]).iloc[1:2].T
```

Keep all the features.

Pipelines

In [10]:

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator
```

Split training and test sets

In [11]:

```
train, test = mineRDD.randomSplit([0.7,0.3], seed = 123)
```

In [12]:

```
from pyspark.ml.feature import VectorAssembler, VectorIndexer
```

Vector Assembler

In [13]:

```
featuresCols = mineRDD.columns
featuresCols.remove('silicaFeed')

vectorAssembler = VectorAssembler(inputCols=featuresCols, outputCol="rawFeatures")
```

```
# This identifies categorical features and indexes them.
vectorIndexer = VectorIndexer(inputCol="rawFeatures", outputCol="features")
```

RANDOM FOREST - REDUCED MODEL

In [15]:

```
rf = RandomForestRegressor(labelCol='silicaFeed')
```

In [16]:

```
# Define a grid of hyperparameters to test:

paramGrid = ParamGridBuilder()\
  .addGrid(rf.numTrees, [2,10])\
  .addGrid(rf.maxDepth, [2,10])\
  .build()

#This tells CrossValidator how well we are doing by comparing the true labels with predictions.
evaluator = RegressionEvaluator(metricName="rmse", labelCol=rf.getLabelCol(), predictionCol=rf.getP
redictionCol())
# Declare the CrossValidator, which runs model tuning for us.
cv = CrossValidator(estimator=rf, evaluator=evaluator, estimatorParamMaps=paramGrid)
```

In [17]:

```
pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])
```

In [24]:

```
pipelineRFModel = pipeline.fit(train)
```

In [26]:

```
predictions = pipelineRFModel.transform(test)
```

In [27]:

```
rmse = evaluator.evaluate(predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 4.2974
```

R^2

In [29]:

```
y_true = predictions.select("silicaFeed").toPandas()
y_pred = predictions.select("prediction").toPandas()
```

In [31]:

```
import sklearn.metrics
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
r2: 0.6011599040096292
```

gradient Boost Trees

In [16]:

```
from pyspark.ml.regression import GBTRegressor
```

In [17]:

```
# Takes the "features" column and learns to predict "silicaFeed"
gbt = GBTRegressor(labelCol="silicaFeed")
```

In [18]:

```
# Define a grid of hyperparameters to test:
paramGrid = ParamGridBuilder()\
  .addGrid(gbt.maxDepth, [2,10])\
  .addGrid(gbt.maxIter, [10,200])\
  .build()

#This tells CrossValidator how well we are doing by comparing the true labels with predictions.
evaluator = RegressionEvaluator(metricName="rmse", labelCol=gbt.getLabelCol(), predictionCol=gbt.ge
tPredictionCol())
# Declare the CrossValidator, which runs model tuning for us.
cv = CrossValidator(estimator=gbt, evaluator=evaluator, estimatorParamMaps=paramGrid)
```

In [19]:

```
pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])
```

In [ ]:

```
pipelineModel = pipeline.fit(train)
```

In [34]:

```
best_predictions = pipelineModel.transform(test)
```

RMSE

In [35]:

```
rmse = evaluator.evaluate(best_predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 1.57483
```

R^2

In [36]:

```
y_true = best_predictions.select("silicaFeed").toPandas()
y_pred = best_predictions.select("prediction").toPandas()
```

In [37]:

```
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
r2: 0.9464383854719923
```

Decision Trees

In [20]:

```
from pyspark.mllib.tree import DecisionTree
from pyspark.ml.regression import DecisionTreeRegressor
```

In [21]:

```
# decisionTree train model
dt = DecisionTreeRegressor(labelCol="silicaFeed")
```

In [22]:

```
# Define a grid of hyperparameters to test:
paramGrid = ParamGridBuilder()\
  .addGrid(dt.maxDepth, [2,10])\
  .build()

#This tells CrossValidator how well we are doing by comparing the true labels with predictions.
evaluator = RegressionEvaluator(metricName="rmse", labelCol=dt.getLabelCol(), predictionCol=dt.getP
redictionCol())
# Declare the CrossValidator, which runs model tuning for us.
cv = CrossValidator(estimator=dt, evaluator=evaluator, estimatorParamMaps=paramGrid)
```

In [23]:

```
pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])
```

In [33]:

```
pipelineTreeModel = pipeline.fit(train)
```

In [35]:

```
predictions = pipelineTreeModel.transform(test)
```

In [36]:

```
rmse = evaluator.evaluate(predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 4.55198
```

R^2

In [39]:

```
import sklearn.metrics
y_true = predictions.select("silicaFeed").toPandas()
y_pred = predictions.select("prediction").toPandas()
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
r2: 0.5525064735055689
```

FULL MODEL RMSE

RandomForest

MAKE SURE TO RELOAD THE DATA MINERDD BUT WITH FULL FEATURES

Reload mineRDD, drop date and % IronFeed, convert pandasDF to SparkRDD

Random Forest

In [18]:

```
pipelineRFModel = pipeline.fit(train)
predictions = pipelineRFModel.transform(test)
rmse = evaluator.evaluate(predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 4.26634
```

In [19]:

```
y_true = predictions.select("silicaFeed").toPandas()
y_pred = predictions.select("prediction").toPandas()
import sklearn.metrics
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
r2: 0.6069056057099547
```

Decision Tree

In [24]:

```
pipelineTreeModel = pipeline.fit(train)
predictions = pipelineTreeModel.transform(test)
rmse = evaluator.evaluate(predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 4.6144
```

In [25]:

```
y_true = predictions.select("silicaFeed").toPandas()
y_pred = predictions.select("prediction").toPandas()
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
r2: 0.5401485664010969
```

GBT

In [20]:

```
pipelineModel = pipeline.fit(train)
```

In [21]:

```
predictions = pipelineModel.transform(test)
```

In [22]:

```
rmse = evaluator.evaluate(predictions)
print("RMSE on our test set: %g" % rmse)
```

```
RMSE on our test set: 1.50837
```

In [24]:

```
import sklearn.metrics
y_true = predictions.select("silicaFeed").toPandas()
y_pred = predictions.select("prediction").toPandas()
r2 = sklearn.metrics.r2_score(y_true, y_pred)
print('r2: {0}'.format(r2))
```

```
print("r2: {0}".format(r2))
```

```
r2: 0.9508639723695737
```

Best Predictions - GBT Full Model

We convert RDD back to Pandas for easy graphics

In [33]:

```
# silicaFeed: the True % Silica Feed from real data
# prediction: our predicted % Silica Feed
rdd_pred = predictions.select('silicaFeed', 'prediction')
```

In [34]:

```
# Spark to Pandas
predictions_pd = rdd_pred.toPandas()
```

Bar Plot

In [ ]:

```
sns.set(style="whitegrid", font='Copperplate Gothic Bold', rc={'axes.facecolor':'cornflowerblue', '
figure.facecolor':'cornflowerblue'})
ax=sns.distplot(predictions_pd['silicaFeed'], color='green', hist_kws={ "alpha": 1}, kde=False)
plt.title('Distribution for TRUE % Silica Feed', fontsize=18)
plt.ylabel('Relative Frequency', fontsize=18)
plt.xlabel('TRUE Silica Feed [%]', fontsize=18)
plt.ylim((0,21000))
plt.show()
ax.figure.savefig("plotf1.png")
```

In [ ]:

```
sns.set(style="whitegrid", font='Copperplate Gothic Bold', rc={'axes.facecolor':'cornflowerblue', '
figure.facecolor':'cornflowerblue'})
ax=sns.distplot(predictions_pd['prediction'], color='green', hist_kws={ "alpha": 1}, kde=False)
plt.title('Distribution for PREDICT % Silica Feed', fontsize=18)
plt.ylabel('Relative Frequency', fontsize=18)
plt.xlabel('PREDICTION Silica Feed [%]', fontsize=18)
plt.xlim((0,35))
plt.show()
ax.figure.savefig("plotf2.png")
```