Table of Contents

- MINE 432 - Quality Prediction of Iron Ore
  - 1. Background
  - 2. Importing the Dataset
  - 3. Importing Required Libraries
  - 4. Importing the Dataset
  - 5. Quick Review of the Data
  - 6. Exploratory Data Analysis (EDA)
    - 6.1. Check for Outliers
    - 6.2. Check for Correlation
  - 7. Data Preprocessing
    - 7.1. Remove Outliers
    - 7.2. Remove Correlated Columns
    - 7.3. Feature Scaling
    - 7.4. Splitting the Data into the Training set and Test set
    - 7.5. Quick Check of the Output Data
  - 8. Linear Regression Model
    - 8.1. Training the Model
    - 8.2. Predicting the Test set results
    - 8.3. Evaluating the Model Performance
  - 9. Other Models
    - 9.1. Linear Regression Model
    - 9.2. Stochastic Gradient Descent Regressor Model
    - 9.3. Decision Tree Regressor Model

MINE 432 - Quality Prediction of Iron Ore

1. Background

Iron ore is a vital input to the steelmaking industry, particularly mining plants. As, I would like to share the database with the community, please come from one of the most important parts of a mining process: a flotation plant!

The flotation is a process where the iron ore is separated from the waste material. In this case, the impurity is measured more than, if we can predict how much silica (impurity) is in a batch of iron ore before it is processed, we can adjust the parameters of the plant to make the most efficient use of the raw material. Hence, this will lead to less process waste and less pollution (because impurity, if not in the waste, will also delay the environment) extending the amount of iron that is going to be kept as you make silica in the iron concentrate).

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

2. Importing the Dataset

```
!wget -q https://raw.githubusercontent.com/dsrscg/DSRP-Datasets/master/MINE_Dataset.zip
```

3. Importing Required Libraries

```
import numpy as np
import pandas as pd
from zipfile import ZipFile as zf
```

```
!pip install -q -U keras_tuner
!pip install -q -U ydata-profiling
from sklearn.model_selection import train_test_split
import tensorflow as tf, keras_tuner, keras
import matplotlib.pyplot as plt, seaborn as sns
import ydata_profiling as pp, sklearn as sk
import numpy as np, pickle, os, zipfile, datetime
from google.colab import drive, files, data_transformer2
drive.mount('/content/drive')
```

4. Importing the Dataset

```
!unzipped_ref = zf('MINE_Dataset.zip', 'r').extract('MINE_Dataset.csv', path='.')
```

5. Quick Review of the Data

Show the dataframe

```
df1.head()
```

Output:

	date	% Iron Feed	% Silica Feed	Starch Flow	Amina Flow	Ore Pulp Flow	Ore Pulp pH	Ore Pulp Density	Flotation Column 01 Air Flow	Flotation Column 02 Air Flow	Flotation Column 03 Air Flow	Flotation Column 04 Air Flow	Flotation Column 05 Air Flow	Flotation Column 06 Air Flow	Flotation Column 07 Air Flow
0	2017-03-10 01:00:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
1	2017-03-10 01:00:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
2	2017-03-10 01:00:40	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
3	2017-03-10 01:01:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
4	2017-03-10 01:01:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551

5 rows × 24 columns

```
df1.shape
```

Output:

```
(737453, 24)
```

737453 rows × 24 columns

```
# get the type of each column
df1.dtypes
```

Output:

```
date                                   object
% Iron Feed                            float64
% Silica Feed                           float64
Starch Flow                            float64
Amina Flow                             float64
Ore Pulp Flow                          float64
Ore Pulp pH                            float64
Ore Pulp Density                       float64
Flotation Column 01 Air Flow           float64
Flotation Column 02 Air Flow           float64
Flotation Column 03 Air Flow           float64
Flotation Column 04 Air Flow           float64
Flotation Column 05 Air Flow           float64
Flotation Column 06 Air Flow           float64
Flotation Column 07 Air Flow           float64
Flotation Column 01 Level              float64
Flotation Column 02 Level              float64
Flotation Column 03 Level              float64
Flotation Column 04 Level              float64
Flotation Column 05 Level              float64
Flotation Column 06 Level              float64
Flotation Column 07 Level              float64
dtype: object
```

Check for missing data

```
df1.isnull().sum()
```

Output:

```
date                                   0
% Iron Feed                            0
% Silica Feed                           0
Starch Flow                            0
Amina Flow                             0
Ore Pulp Flow                          0
Ore Pulp pH                            0
Ore Pulp Density                       0
Flotation Column 01 Air Flow           0
Flotation Column 02 Air Flow           0
Flotation Column 03 Air Flow           0
Flotation Column 04 Air Flow           0
Flotation Column 05 Air Flow           0
Flotation Column 06 Air Flow           0
Flotation Column 07 Air Flow           0
Flotation Column 01 Level              0
Flotation Column 02 Level              0
Flotation Column 03 Level              0
Flotation Column 04 Level              0
Flotation Column 05 Level              0
Flotation Column 06 Level              0
Flotation Column 07 Level              0
dtype: int64
```

6. Exploratory Data Analysis (EDA)

```
df1.head(100)
```

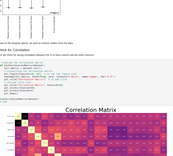
	date	% Iron Feed	% Silica Feed	Starch Flow	Amina Flow	Ore Pulp Flow	Ore Pulp pH	Ore Pulp Density	Flotation Column 01 Air Flow	Flotation Column 02 Air Flow	Flotation Column 03 Air Flow	Flotation Column 04 Air Flow	Flotation Column 05 Air Flow	Flotation Column 06 Air Flow	Flotation Column 07 Air Flow
0	2017-03-10 01:00:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
1	2017-03-10 01:00:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
2	2017-03-10 01:00:40	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
3	2017-03-10 01:01:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
4	2017-03-10 01:01:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	2017-03-10 01:32:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
96	2017-03-10 01:32:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
97	2017-03-10 01:32:40	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
98	2017-03-10 01:33:00	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551
99	2017-03-10 01:33:20	55.2820	5.178300	5735.000000	573.100000	396.021000	10.000000	1.689702	250.995	249.214	250.636	250.884	250.062	251.197	250.551

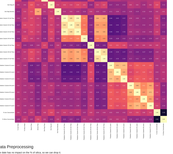
100 rows × 24 columns

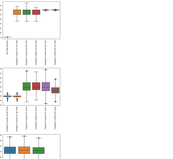
Check for Outliers

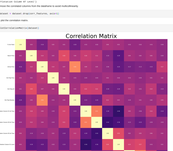
Checking the data, it was found that we are outliers

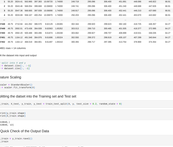
```
# Create boxplots for each column
for column in df1.select_dtypes(include=['float64', 'int64']):
  plt.figure(figure=(8, 2))
  sns.boxplot(data=df1[column], [-1.0:0])
  plt.show()
```

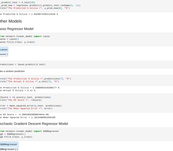












Based on the boxplots above, we want to remove outliers from the data.

Check for Correlation

Here we check for strong correlations between the % of silica column and the other columns.

```
# Create a correlation matrix
corr = df1.select_dtypes(include=['float64', 'int64'])
# Plot the correlation matrix
plt.figure(figsize=(20, 20))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix', fontsize=20)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.tight_layout()
plt.show()
plt.savefig('Correlation Matrix.')
```


7. Data Preprocessing

The data has to impact on the % of silica, so we can drop it.

```
df1.drop(['date'], axis=1, inplace=True)
```

Remove Outliers

Remove the outliers as much possible. We will use the IQR method.

```
# Remove outliers from all columns
for column in df1.select_dtypes(include=['float64', 'int64']):
  Q1 = df1[column].quantile(0.25)
  Q3 = df1[column].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  df1 = df1[(df1[column] >= lower_bound) & (df1[column] <= upper_bound)]
# Check the shape of the dataset after removing outliers
print("Shape of the dataset after removing outliers:", df1.shape)
```

Output:

```
Shape of the dataset after removing outliers: (737453, 24)
```

Remove Correlated Columns

Define a function to find correlated columns, given a threshold.

```
def find_correlated(df, threshold):
  corr = df.corr(numeric_only=True)
  # Find pairs of columns with correlation above the threshold
  correlated_pairs = []
  for i in range(len(corr.columns)):
    for j in range(i):
      if abs(corr.iloc[i, j]) > threshold:
        pair = (corr.columns[i], corr.columns[j])
        correlated_pairs.append(pair)
  return correlated_pairs
```

Call the function to find correlated columns.

```
correlated_pairs = find_correlated(df1, 0.9)
print("Correlated pairs:")
for pair in correlated_pairs:
  print(pair)
```

Output:

```
Correlated pairs:
('Flotation Column 01 Air Flow', '% Iron Feed')
('Flotation Column 02 Air Flow', '% Iron Feed')
('Flotation Column 03 Air Flow', '% Iron Feed')
('Flotation Column 04 Air Flow', '% Iron Feed')
('Flotation Column 05 Air Flow', '% Iron Feed')
('Flotation Column 06 Air Flow', '% Iron Feed')
('Flotation Column 07 Air Flow', '% Iron Feed')
```

Drop the correlated columns, except the dependent variable (silica).

```
df1.drop(['Flotation Column 01 Air Flow', '% Iron Feed', 'Amina Flow', 'Ore Pulp Flow', 'Ore Pulp pH', 'Ore Pulp Density', 'Flotation Column 02 Air Flow', 'Flotation Column 03 Air Flow', 'Flotation Column 04 Air Flow', 'Flotation Column 05 Air Flow', 'Flotation Column 06 Air Flow', 'Flotation Column 07 Air Flow', 'Flotation Column 01 Level', 'Flotation Column 02 Level', 'Flotation Column 03 Level', 'Flotation Column 04 Level', 'Flotation Column 05 Level', 'Flotation Column 06 Level', 'Flotation Column 07 Level'], axis=1, inplace=True)
```

Do an the correlation matrix.

```
# Create a correlation matrix
corr = df1.select_dtypes(include=['float64', 'int64'])
# Plot the correlation matrix
plt.figure(figsize=(20, 20))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix', fontsize=20)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.tight_layout()
plt.show()
plt.savefig('Correlation Matrix.')
```


Feature Scaling

```
scaler = StandardScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])
X_test[X_train.columns] = scaler.transform(X_test[X_train.columns])
```

Splitting the dataset into the Training set and Test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(589962, 7)
(147491, 7)
(589962,)
(147491,)
```

Quick Check of the Output Data

```
y_train.iloc[0:5]
```

Output:

```
0    2.0
1    2.0
2    2.0
3    2.0
4    2.0
Name: % Silica Feed, dtype: float64
```

```
print(type(y_train))
print(type(y_test))
print(type(X_train))
print(type(X_test))
```

Output:

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

8. Linear Regression Model

Training the Model

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Output:

```
LinearRegression
LinearRegression()
```

Predicting the Test set results

```
y_pred = lr.predict(X_test)
```

Make a column prediction

```
y_test = y_test.reset()
y_test
```

Output:

```
0    2.0
1    2.0
2    2.0
3    2.0
4    2.0
Name: % Silica Feed, dtype: float64
```

```
y_test.shape, y_pred.shape
```

Output:

```
((147491,), (147491,))
```

This is a quick sanitiness to find what seem to be the % of silica.

Evaluating the Model Performance

```
from sklearn.metrics import r2_score, mean_absolute_error
```

```
r2score = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("R-squared Score:", r2score)
print("Mean Absolute Error:", mae)
```

Output:

```
R-squared Score: 0.9999999999999999
Mean Absolute Error: 0.0
```

```
y_test.iloc[0:5]
```

Output:

```
0    2.0
1    2.0
2    2.0
3    2.0
4    2.0
Name: % Silica Feed, dtype: float64
```

```
y_pred[0:5]
```

Output:

```
array([2.0000000 , 2.0000000 , 2.0000000 , 2.0000000 , 2.0000000 ])
```

```
y_test.iloc[0:5] - y_pred[0:5]
```

Output:

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: % Silica Feed, dtype: float64
```

Measuring the accuracy of the model

```
lr.score(X_test, y_test)
```

Output:

```
1.0
```

Using the Model to Predict the % of Silica

```
y_pred2 = lr.predict(X_test)
print("The Predicted % Silica 1 = ", y_pred2[0])
print("The Predicted % Silica 2 = ", y_pred2[1])
```

Output:

```
The Predicted % Silica 1 = 2.0000000000000000
The Predicted % Silica 2 = 2.0000000000000000
```

Other Models

Linear Regression Model

```
from sklearn.linear_model import Lasso
ls = Lasso()
ls.fit(X_train, y_train)
```

Output:

```
Lasso
Lasso()
```

```
y_pred3 = ls.predict(X_test)
```

Make a column prediction

```
print("The Predicted % Silica 1 = ", y_pred3[0])
print("The Predicted % Silica 2 = ", y_pred3[1])
```

Output:

```
The Predicted % Silica 1 = 2.0000000000000000
The Predicted % Silica 2 = 2.0000000000000000
```

Stochastic Gradient Descent Regressor Model

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor()
sgd.fit(X_train, y_train)
```

Output:

```
SGDRegressor
SGDRegressor()
```

```
y_pred4 = sgd.predict(X_test)
```

```
print("The Predicted % Silica 1 = ", y_pred4[0])
print("The Predicted % Silica 2 = ", y_pred4[1])
```

Output:

```
The Predicted % Silica 1 = 2.0000000000000000
The Predicted % Silica 2 = 2.0000000000000000
```

Decision Tree Regressor Model

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
```

Output:

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

```
y_pred5 = dtr.predict(X_test)
```

```
print("The Predicted % Silica 1 = ", y_pred5[0])
print("The Predicted % Silica 2 = ", y_pred5[1])
```

Output:

```
The Predicted % Silica 1 = 2.0000000000000000
The Predicted % Silica 2 = 2.0000000000000000
```