```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
# Installing Kaggle Library
!pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 65 bytes, last modified: 6/7/2023 - 100% done

```
Saving kaggle.json to kaggle.json
```

{'kaggle.json': b'{"username":"mehhrishi","key":"5106d4265061cadc32de7738609ffb88"}'}

```
# create a kaggle folder
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets list
```

```
ref                                                                 title                                               size  la
------------------------------------------------------------------  -------------------------------------------------  -----  --
arnabchaki/data-science-salaries-2023                               Data Science Salaries 2023 💸                        25KB  2
tawfikelmetwally/automobile-dataset                                 Car information dataset                             6KB  20
fatihb/coffee-quality-data-cqi                                      Coffee Quality Data (CQI May-2023)                 22KB  20
mohithsairamreddy/salary-data                                       Salary_Data                                        17KB  20
mauryansshivam/netflix-ott-revenue-and-subscribers-csv-file         Netflix OTT Revenue and Subscribers (CSV File)      2KB  20
omarsobhy14/mcdonalds-revenue                                       🍟💰From Flipping Burgers to Billions: McDonald's     565B
zsinghrahulk/rice-pest-and-diseases                                 Rice - Pest and Diseases                          312KB  20
iammustafatz/diabetes-prediction-dataset                            Diabetes prediction dataset                       734KB  20
vstacknocopyright/fruit-and-vegetable-prices                        Fruit and Vegetable Prices                          1KB  20
bilalwaseer/microsoft-stocks-from-1986-to-2023                      Microsoft Stocks from 1986 to 2023                120KB  20
darshanprabhu09/stock-prices-for                                    Stock prices of Amazon , Microsoft , Google, Apple 85KB  20
rajkumarpandey02/2023-world-population-by-country                    World Population by Country                       38KB  20
danishjmeo/karachi-housing-prices-2023                              Karachi_Housing_Prices_2023                         1MB  20
adityaramachandran27/world-air-quality-index-by-city-and-coordinates World Air Quality Index by City and Coordinates  372KB  20
dansbecker/melbourne-housing-snapshot                               Melbourne Housing Snapshot                        451KB  20
pushpakhinglaspure/oscar-dataset                                    Oscar Academy Award-winning films 1927-2022       161KB  20
aryansingh0909/weekly-patent-application-granted                    Patent Application Granted Dataset                  6MB  20
utkarshx27/heart-disease-diagnosis-dataset                          Heart Disease Prediction Dataset                    3KB  20
shreyanshverma27/water-quality-testing                              Water Quality Testing                               4KB  20
desalegngeb/conversion-predictors-of-cis-to-multiple-sclerosis      Multiple Sclerosis Disease                          3KB  20
```

```
!kaggle datasets download -d edumagalhaes/quality-prediction-in-a-mining-process
```

```
Downloading quality-prediction-in-a-mining-process.zip to /content
100% 50.9M/50.9M [00:02<00:00, 27.9MB/s]
100% 50.9M/50.9M [00:02<00:00, 21.4MB/s]
```

```
!unzip quality-prediction-in-a-mining-process.zip
```

```
Archive:  quality-prediction-in-a-mining-process.zip
  inflating: MiningProcess_Flotation_Plant_Database.csv
```

```
df = pd.read_csv('MiningProcess_Flotation_Plant_Database.csv')
```

```
df
```

	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	...	Fl Co A
0	2017-03-10 01:00:00	55,2	16,98	3019,53	557,434	395,713	10,0664	1,74	249,214	253,235	...	
1	2017-03-10 01:00:00	55,2	16,98	3024,41	563,965	397,383	10,0672	1,74	249,719	250,532	...	
2	2017-03-10 01:00:00	55,2	16,98	3043,46	568,054	399,668	10,068	1,74	249,741	247,874	...	
3	2017-03-10 01:00:00	55,2	16,98	3047,36	568,665	397,939	10,0689	1,74	249,917	254,487	...	
4	2017-03-10 01:00:00	55,2	16,98	3033,69	558,167	400,254	10,0697	1,74	250,203	252,136	...	
...	...	...	...	...	...	...	...	...	...	...	...	
737448	2017-09-09 23:00:00	49,75	23,2	2710,94	441,052	386,57	9,62129	1,65365	302,344	298,786	...	
737449	2017-09-09 23:00:00	49,75	23,2	2692,01	473,436	384,939	9,62063	1,65352	303,013	301,879	...	
737450	2017-09-09 23:00:00	49,75	23,2	2692,2	500,488	383,496	9,61874	1,65338	303,662	307,397	...	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 737453 entries, 0 to 737452
Data columns (total 24 columns):
 #   Column                        Non-Null Count   Dtype 
---  ------                        --------------   ----- 
 0   date                          737453 non-null  object
 1   % Iron Feed                   737453 non-null  object
 2   % Silica Feed                 737453 non-null  object
 3   Starch Flow                   737453 non-null  object
 4   Amina Flow                    737453 non-null  object
 5   Ore Pulp Flow                 737453 non-null  object
 6   Ore Pulp pH                   737453 non-null  object
 7   Ore Pulp Density              737453 non-null  object
 8   Flotation Column 01 Air Flow  737453 non-null  object
 9   Flotation Column 02 Air Flow  737453 non-null  object
 10  Flotation Column 03 Air Flow  737453 non-null  object
 11  Flotation Column 04 Air Flow  737453 non-null  object
 12  Flotation Column 05 Air Flow  737453 non-null  object
 13  Flotation Column 06 Air Flow  737453 non-null  object
 14  Flotation Column 07 Air Flow  737453 non-null  object
 15  Flotation Column 01 Level     737453 non-null  object
 16  Flotation Column 02 Level     737453 non-null  object
 17  Flotation Column 03 Level     737453 non-null  object
 18  Flotation Column 04 Level     737453 non-null  object
 19  Flotation Column 05 Level     737453 non-null  object
 20  Flotation Column 06 Level     737453 non-null  object
 21  Flotation Column 07 Level     737453 non-null  object
 22  % Iron Concentrate            737453 non-null  object
 23  % Silica Concentrate          737453 non-null  object
dtypes: object(24)
memory usage: 135.0+ MB
```

- The main goal is to use this data to predict how much impurity is in the ore concentrate. As this impurity is measured every hour, if we can predict how much silica (impurity) is in the ore concentrate, we can help the engineers, giving them early information to take actions (empowering!). Hence, they will be able to take corrective actions in advance (reduce impurity, if it is the case) and also help the environment (reducing the amount of ore that goes to tailings as you reduce silica in the ore concentrate).

Content

- The first column shows time and date range (from march of 2017 until september of 2017). Some columns were sampled every 20 second. Others were sampled on a hourly base.
- The second and third columns are quality measures of the iron ore pulp right before it is fed into the flotation plant. Column 4 until column 8 are the most important variables that impact in the ore quality in the end of the process. From column 9 until column 22, we can see process data (level and air flow inside the flotation columns, which also impact in ore quality. The last two columns are the final iron ore pulp quality measurement from the lab.

- Target is to predict the last column, which is the % of silica in the iron ore concentrate.

Inspiration

- I have been working in this dataset for at least six months and would like to see if the community can help to answer the following questions:
    - Is it possible to predict % Silica Concentrate every minute?
    - How many steps (hours) ahead can we predict % Silica in Concentrate? This would help engineers to act in predictive and optimized way, mitigatin the % of iron that could have gone to tailings.
    - Is it possible to predict % Silica in Concentrate whitout using % Iron Concentrate column (as they are highly correlated)?

```
df.describe()
```

	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	...	F C
count	737453	737453	737453	737453	737453	737453	737453	737453	737453	737453	...	
unique	4097	278	293	409317	319416	180189	131143	105805	43675	80442	...	
top	2017-06-16 15:00:00	64,03	6,26	2562,5	534,668	402,246	10,0591	1,75	299,927	255,322	...	
freq	180	142560	142560	690	959	1735	1509	3214	13683	1487	...	

4 rows × 24 columns

```
# checking the nul values
df.isnull().sum()
```

```
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
% Iron Concentrate              0
% Silica Concentrate            0
dtype: int64
```

Replace the ',' with '.'

```
for i in df.columns:
    df[i] = df[i].apply(lambda x : x.replace(',','.'))
```

```
df
```

	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	...	Fl Co A
0	2017-03-10 01:00:00	55.2	16.98	3019.53	557.434	395.713	10.0664	1.74	249.214	253.235	...	
1	2017-03-10 01:00:00	55.2	16.98	3024.41	563.965	397.383	10.0672	1.74	249.719	250.532	...	
2	2017-03-10 01:00:00	55.2	16.98	3043.46	568.054	399.668	10.068	1.74	249.741	247.874	...	
3	2017-03-10 01:00:00	55.2	16.98	3047.36	568.665	397.939	10.0689	1.74	249.917	254.487	...	
4	2017-03-10 01:00:00	55.2	16.98	3033.69	558.167	400.254	10.0697	1.74	250.203	252.136	...	
...	...	...	...	...	...	...	...	...	...	...	...	
737448	2017-09-09 23:00:00	49.75	23.2	2710.94	441.052	386.57	9.62129	1.65365	302.344	298.786	...	
737449	2017-09-09 23:00:00	49.75	23.2	2692.01	473.436	384.939	9.62063	1.65352	303.013	301.879	...	
737450	2017-09-09 23:00:00	49.75	23.2	2692.2	500.488	383.496	9.61874	1.65338	303.662	307.397	...	
737451	2017-09-09	49.75	23.2	1164.12	491.548	384.976	9.61686	1.65324	302.55	301.959	...	
737452	09-09	49.75	23.2	1164.12	468.019	384.801	9.61497	1.6531	300.355	292.865	...	

737453 rows × 24 columns

Feature engineering

```
import re
```

```
df['date'] = df['date'].apply(lambda x : re.search('[0-9]*-[0-9]*' , x).group(0)) # for getting month and year
```

```
df
```

```
#year
df['Year']=df['date'].apply(lambda x : re.search('^[^-]*' , x).group(0))

#month
df['Month'] = df['date'].apply(lambda x : re.search('[^-]*$' , x).group(0))
```

```
df.drop(columns='date', axis=1,inplace=True)
```

```
df
```

	% 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	...
0	55.2	16.98	3019.53	557.434	395.713	10.0664	1.74	249.214	253.235	250.576	...
1	55.2	16.98	3024.41	563.965	397.383	10.0672	1.74	249.719	250.532	250.862	...
2	55.2	16.98	3043.46	568.054	399.668	10.068	1.74	249.741	247.874	250.313	...
3	55.2	16.98	3047.36	568.665	397.939	10.0689	1.74	249.917	254.487	250.049	...
4	55.2	16.98	3033.69	558.167	400.254	10.0697	1.74	250.203	252.136	249.895	...
...	...	...	...	...	...	...	...	...	...	...	...
737448	49.75	23.2	2710.94	441.052	386.57	9.62129	1.65365	302.344	298.786	299.163	...
737449	49.75	23.2	2692.01	473.436	384.939	9.62063	1.65352	303.013	301.879	299.487	...
737450	49.75	23.2	2692.2	500.488	383.496	9.61874	1.65338	303.662	307.397	299.487	...
737451	49.75	23.2	1164.12	491.548	384.976	9.61686	1.65324	302.55	301.959	298.045	...
737452	49.75	23.2	1164.12	468.019	384.801	9.61497	1.6531	300.355	292.865	298.625	...

737453 rows × 25 columns

```
df = df.astype('float64')
```

```
df.dtypes
```

```
% 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
% Iron Concentrate              float64
% Silica Concentrate            float64
Year                            float64
Month                           float64
dtype: object
```

```
df
```

	% Iron Feed	% Silica Feed	Starch Flow	Amina Flow	Ore Pulp Flow	Ore Pulp pH	Ore Pulp Density	Flotation Column 01 Air Flow	Flotati Column Air Fl
0	55.20	16.98	3019.53	557.434	395.713	10.06640	1.74000	249.214	253.2
1	55.20	16.98	3024.41	563.965	397.383	10.06720	1.74000	249.719	250.5
2	55.20	16.98	3043.46	568.054	399.668	10.06800	1.74000	249.741	247.8
3	55.20	16.98	3047.36	568.665	397.939	10.06890	1.74000	249.917	254.4
4	55.20	16.98	3033.69	558.167	400.254	10.06970	1.74000	250.203	252.1
...	...	...	...	...	...	...	...	...	
737448	49.75	23.20	2710.94	441.052	386.570	9.62129	1.65365	302.344	298.7
737449	49.75	23.20	2692.01	473.436	384.939	9.62063	1.65352	303.013	301.8
737450	49.75	23.20	2692.20	500.488	383.496	9.61874	1.65338	303.662	307.3

```
x = df.drop(columns='% Silica Concentrate').values
```

```
x
```

```
array([[5.52000e+01, 1.69800e+01, 3.01953e+03, ..., 6.69100e+01,
        2.01700e+03, 3.00000e+00],
       [5.52000e+01, 1.69800e+01, 3.02441e+03, ..., 6.69100e+01,
        2.01700e+03, 3.00000e+00],
       [5.52000e+01, 1.69800e+01, 3.04346e+03, ..., 6.69100e+01,
        2.01700e+03, 3.00000e+00],
       ...,
       [4.97500e+01, 2.32000e+01, 2.69220e+03, ..., 6.42700e+01,
        2.01700e+03, 9.00000e+00],
       [4.97500e+01, 2.32000e+01, 1.16412e+03, ..., 6.42700e+01,
        2.01700e+03, 9.00000e+00],
       [4.97500e+01, 2.32000e+01, 1.16412e+03, ..., 6.42700e+01,
        2.01700e+03, 9.00000e+00]])
```

```
y = df.iloc[:,-3].values
```

```
y
```

```
array([1.31, 1.31, 1.31, ..., 1.71, 1.71, 1.71])
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

```
x
```

```
array([[-0.21225167,  0.34202086,  0.12375664, ...,  1.66266665,
         0.        , -1.827297  ],
       [-0.21225167,  0.34202086,  0.12777243, ...,  1.66266665,
         0.        , -1.827297  ],
       [-0.21225167,  0.34202086,  0.14344883, ...,  1.66266665,
         0.        , -1.827297  ],
       ...,
       [-1.26891589,  1.25572783, -0.14560578, ..., -0.69733357,
         0.        ,  1.80686228],
       [-1.26891589,  1.25572783, -1.40307479, ..., -0.69733357,
         0.        ,  1.80686228],
       [-1.26891589,  1.25572783, -1.40307479, ..., -0.69733357,
         0.        ,  1.80686228]])
```

```
# Separating the x and y for Training purpose
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1)
```

```
xtrain
```

```
array([[ 0.05336851, -0.17359192, -1.89555611, ...,  0.2055453 ,
         0.        , -1.22160379],
       [-1.34646923,  1.50545482,  0.13211738, ...,  0.25918167,
         0.        ,  0.59547585],
       [-1.08472672,  1.25132089,  0.84143925, ...,  0.12509075,
         0.        , -0.01021736],
       ...,
```

```
       [-0.09592166,  0.11873555,  0.70774142, ..., -1.03703058,
         0.        ,  0.59547585],
       [ 0.09214518, -0.04138352,  0.53451109, ...,  0.34857562,
         0.        ,  0.59547585],
       [-0.21806817, -0.04432148, -0.70139774, ..., -0.79566692,
         0.        , -1.22160379]])
```

```
df.columns.size
```

```
25
```

```
# Importing libraries of Neural network
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scipy.stats.morestats import optimize
```

```
ann = Sequential()
ann.add(Dense(units = 20, activation = 'relu')) # hidden Layer 1
ann.add(Dense(units = 20, activation = 'relu')) # hidden Layer 2

ann.add(Dense(units = 1, activation = 'linear')) # output layer
# where activation function is by default set as 'linear'.

ann.compile(optimizer='adam', loss = 'mse')

ann.fit(xtrain,ytrain,validation_data = (xtest,ytest) ,epochs = 10)
```

```
Epoch 1/10
16132/16132 [==============================] - 32s 2ms/step - loss: 0.3003 - val_loss: 0.2471
Epoch 2/10
16132/16132 [==============================] - 26s 2ms/step - loss: 0.2377 - val_loss: 0.2286
Epoch 3/10
16132/16132 [==============================] - 26s 2ms/step - loss: 0.2249 - val_loss: 0.2224
Epoch 4/10
16132/16132 [==============================] - 26s 2ms/step - loss: 0.2184 - val_loss: 0.2132
Epoch 5/10
16132/16132 [==============================] - 25s 2ms/step - loss: 0.2148 - val_loss: 0.2135
Epoch 6/10
16132/16132 [==============================] - 31s 2ms/step - loss: 0.2115 - val_loss: 0.2060
Epoch 7/10
16132/16132 [==============================] - 25s 2ms/step - loss: 0.2088 - val_loss: 0.2058
Epoch 8/10
16132/16132 [==============================] - 25s 2ms/step - loss: 0.2067 - val_loss: 0.2061
Epoch 9/10
16132/16132 [==============================] - 26s 2ms/step - loss: 0.2051 - val_loss: 0.2065
Epoch 10/10
16132/16132 [==============================] - 25s 2ms/step - loss: 0.2039 - val_loss: 0.2029
<keras.callbacks.History at 0x7f760a19f1c0>
```

```
model_loss = ann.history.history
model_loss
```

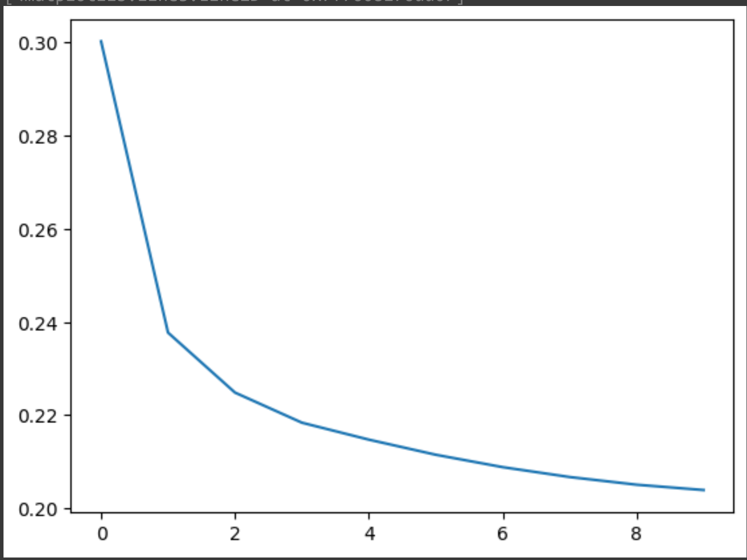
```
{'loss': [0.3002714514732361,
  0.23773451149463654,
  0.22486205399036407,
  0.2183936983346939,
  0.21475359797477722,
  0.2115083932876587,
  0.20884431898593903,
  0.20672336220741272,
  0.2050764560699463,
  0.2039332538843155],
 'val_loss': [0.24714542925357819,
  0.22859543561935425,
  0.22244992852210999,
  0.21318714320659637,
  0.21351295709609985,
  0.20598040521144867,
  0.20583955943584442,
  0.20606382191181183,
  0.20652467012405396,
  0.20287956297397614]}
```

```
df_loss = pd.DataFrame(model_loss)
df_loss
```

	loss	val_loss
0	0.300271	0.247145
1	0.237735	0.228595
2	0.224862	0.222450
3	0.218394	0.213187
4	0.214754	0.213513
5	0.211508	0.205980
6	0.208844	0.205840

```
plt.plot(df_loss['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f7608176dd0>]
```




```
ypred = ann.predict(xtest)
```

```
6914/6914 [==============================] - 6s 842us/step
```

```
ypred
```

```
array([[3.1971667],
       [1.7415532],
       [3.913054 ],
       ...,
       [3.1401117],
       [2.6461346],
       [4.389356 ]], dtype=float32)
```

```
# the reason behind using flatten is yped is 2d array
# flatten cahnges 2d array to 1d array.
```

```
pd.DataFrame({'Actual value': ytest, 'Predicted value': ypred.flatten()})
```

	Actual value	Predicted value
0	2.610000	3.197167
1	1.117835	1.741553
2	4.290000	3.913054
3	2.070000	1.706998
4	1.620000	2.503230
...	...	...
221231	3.360000	3.063824
221232	1.250000	1.432341
221233	3.923518	3.140112
221234	2.070000	2.646135
221235	5.050000	4.389356

221236 rows × 2 columns

Model Evaluation

```
from sklearn.metrics import r2_score
```

```
r2_score(ytest,ypred)
```

```
0.8402268375532407
```

✓ 0s completed at 4:01 PM