Appendix B: Python code developed for different Machine Learning techniques.

General code:

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
import seaborn as sns
%matplotlib inline
df = pd.read_excel("input.xlsx")
df_test = pd.read_excel("input_testing.xlsx")
# take a look at the dataset
df.head()
df.columns
# select some features that we want to use for regression.
#'OperatingCondition', 'ViscosityRation'
cdf = df[['Temp', 'MW', 'Vol/int Ratio','CO2','C1', 'N2','H2S','C2', 'MMP']]
testing_data = df_test[['Temp', 'MW', 'Vol/int Ratio','CO2','C1', 'N2','H2S','C2']]
cdf.head(18)
cdf.corr()
x = ['Temp', 'MW', 'Vol/int Ratio', 'CO2', 'C1', 'N2', 'H2S', 'C2+']
y = [0.608687, 0.266845, 0.434933, -0.580074, 0.579177, 0.413702, -0.238834, 0.291129]
plt.bar(x, y)
ax = plt.gca()
ax.set_ylim([-1,1])
for index, value in enumerate(y):
    if value > 0:
        plt.text(index- .25,value+0.05,  str(round(value,2)))
    else:
        plt.text(index- .25,value - 0.1,  str(round(value,2)))
plt.ylabel ('Pearson Factor',fontsize =15)
```

```
plt.xticks(rotation =50,fontsize =10)
corr = cdf.corr()
# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
cdf.columns
train_dataset = cdf.sample(frac=0.8,random_state=42)
test_dataset = cdf.drop(train_dataset.index)
train_stats = train_dataset.describe()
train_stats.pop("MMP")
train_stats = train_stats.transpose()
train_stats
train_labels = train_dataset.pop("MMP")
test_labels = test_dataset.pop("MMP")
```

B-1 Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)
# fit the regressor with X and Y data
```

```
regressor.fit(normed_train_data, train_labels)
yh = regressor.predict(normed_test_data)
yh_testing = regressor.predict(normed_testing_data)
print(test_labels.values)
for i in range(len(test_labels.values)):
print(test_labels.values[i])
plt.plot(test_labels.values, color = 'red', label = 'Real data')
plt.plot(yh, color = 'blue', label = 'Predicted data')
plt.title('Prediction')
plt.legend()
plt.show()
score = np.sqrt(metrics.mean_squared_error(yh,test_labels))
print("Final score (RMSE): {}".format(score))
```

B-2 Random Forest

```
from sklearn.ensemble import RandomForestRegressor
ran = RandomForestRegressor()
ran.fit(normed_train_data, train_labels)
yhat = ran.predict(normed_test_data)
yhat_testing = ran.predict(normed_testing_data)
print(test_labels.values)
for i in range(len(test_labels.values)):
print(test_labels.values[i])
plt.plot(test_labels.values, color = 'red', label = 'Real data')
plt.plot(yhat, color = 'blue', label = 'Predicted data')
plt.title('Prediction')
plt.legend()
plt.show()
from sklearn.metrics import r2_score
r2_score(yhat,test_labels)
```

B-3 Deep Learning

```
from keras.preprocessing.text import Tokenizer
from keras import models
from keras import layers
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
train_dataset = sc.fit_transform(train_dataset)
test_dataset = sc.transform(test_dataset)
network = models.Sequential()
network.add(layers.Dense(units=32,
activation="relu",
input_shape=(train_dataset.shape[1],)))
network.add(layers.Dense(units=32, activation="relu"))
network.add(layers.Dense(units=32, activation="relu"))
network.add(layers.Dense(units=32, activation="relu"))
network.add(layers.Dense(units=32, activation="relu"))
network.add(layers.Dense(units=1))
network.compile(loss="mse",
optimizer="RMSprop",
metrics=["mse"])
history = network.fit(train_dataset,
train_labels,
epochs=1500,
verbose=0,
batch_size=100,
validation_data=(test_dataset, test_labels))
y_predict = network.predict(test_dataset)
plt.plot(test_labels.values, color = 'red', label = 'Real data')
```

```
plt.plot(y_predict, color = 'blue', label = 'Predicted data')
plt.title('Prediction')
plt.legend()
plt.show()
```

B-4 Deep Learning with Early Stopping

```
optimizer = tf.keras.optimizers.RMSprop(0.0099)
model.compile(loss='mean_squared_error',optimizer=optimizer)
#Early stopping:
from keras.models import Sequential
import keras
import keras.backend as kb
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dropout
from sklearn.metrics import r2_score
monitor = EarlyStopping(monitor='loss', min_delta=1e-6,
                        patience=5, verbose=1, mode='auto',
                        restore_best_weights=True)
# ,callbacks=[monitor]
model.fit(normed_train_data,
train_labels,validation_data=(normed_test_data,test_labels),epochs=1500)
test_predictions = model.predict(normed_test_data).flatten()
a = plt.axes(aspect='equal')
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MMP]')
plt.ylabel('Predictions [MMP]')
lims = [0, 6000]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```

B-5 Deep Learning with k -fold Cross Validation

```
from sklearn.model_selection import KFold
num_folds = 5
# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []
r_per_fold =[]
rmse_per_fold = []
# inputs = np.concatenate((normed_train_data, train_labels), axis=0)
# # targets = np.concatenate((normed_test_data, test_labels), axis=0)
dataset = cdf.to_numpy()
dataset
inputs = dataset[:,0:8]
target = dataset[:,8]
target.shape
kfold = KFold(n_splits=num_folds, shuffle=True)
fold_no = 1
for train, test in kfold.split(inputs, targets):
   model = keras.Sequential([
   keras.layers.Dense(32, activation=tf.nn.relu, input_shape=(8,)), # Hidden 1
#         keras.layers.Dropout(0.2),
   keras.layers.Dense(32, activation=tf.nn.relu), # Hidden 2
   keras.layers.Dense(32, activation=tf.nn.relu), # Hidden 3
   keras.layers.Dense(32, activation=tf.nn.relu), # Hidden 4
   keras.layers.Dense(32, activation=tf.nn.relu), # Hidden 5
   keras.layers.Dense(1) #Output
  ])
   optimizer = tf.keras.optimizers.RMSprop(0.0099)
   model.compile(loss='mean_squared_error',optimizer=optimizer,metrics=['accuracy'])
   # Generate a print
```

```
    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')
    model.fit(inputs[train], targets[train],epochs=1500, batch_size = 50, verbose= 1)
    # Generate generalization metrics
    scores = model.evaluate(inputs[test], targets[test], verbose=0)
    print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]};
{model.metrics_names[1]} of {scores[1]*100}%')
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])
    test_predictions = model.predict(inputs[test]).flatten()
    test_labels = targets[test]
#     Final score (RMSE)
    score = np.sqrt(metrics.mean_squared_error(test_labels, test_predictions))

    rmse_per_fold.append(score)
    r_per_fold.append(r2_score(test_labels, test_predictions))
#     print(r2_score(test_labels, test_predictions))
  # Increase fold number
    fold_no = fold_no + 1
# # == Provide average scores ==
print('------------------------------------------------------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
    print('------------------------------------------------------------------------')
    print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print(f'> R^2: {np.mean(r_per_fold)}')
print(f'> RMSE: {np.mean(rmse_per_fold)}')
```

```
print('------------------------------------------------------------------------')
mse= model.evaluate(normed_test_data, test_labels, verbose=2)
print("Testing set Mean Square Error: {:5.2f} MMP".format(mse))
plt.plot(test_predictions, color = 'blue', label = 'Predicted data')
plt.plot(test_labels.values, color = 'red', label = 'Real data')
plt.title('Prediction')
plt.legend()
plt.show()
```