Homework cover page

Analytic and machine learning-based modeling of dynamical systems - 036064

Dr. Maor Farid

☒ Homework no. ___3____

☐ Final project

Students details:

Chenhao Yang	941170706
Full name	ID no.
Jonathan Oh	941170383
Full name	ID no.

Before submitting your homework, you are requested to fill in the suitable survey on the course website. Submitting the survey is mandatory. Your feedback is highly important for the course's staff and serves us for optimizing the course contents for you and for maximizing the contribution of the HW and assimilation of the material learned. Of course, your answers do not influence your grade and evaluation in any way.

☒ I confirm that I have filled in and submitted the HW survey on the course website

Thanks for your collaboration and good luck!

Assignment 3 - machine learning methods for solving real-world regression problems and good practice techniques in Data Science

In [ ]:
```
import os
import re
import sys
import torch
import urllib
import shutil
import pathlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pprint import pprint

%load_ext autoreload
%autoreload 2
plt.rcParams.update({'font.size': 15,'axes.labelsize' : 15})
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
# !nvidia-smi
```

Part (I) Univariate data and linear regression

1. (8 points)Importing the data

In [2]:
```
# we upload the data file to github to
DATA_URL_1 = 'https://raw.githubusercontent.com/afiretony/NLDML_hw3/main/data/hw3_ex1.csv'
DATA_URL_2 = 'https://raw.githubusercontent.com/afiretony/NLDML_hw3/main/data/hw3_ex2.csv'
DATA_DIR = pathlib.Path.home().joinpath('datasets')

def download_data(out_path=DATA_DIR, url=DATA_URL_1, force=False):
    pathlib.Path(out_path).mkdir(exist_ok=True)
    out_filename = os.path.join(out_path, os.path.basename(url))

    if os.path.isfile(out_filename) and not force:
        print(f'DATA {out_filename} exists, skipping download.')
    else:
        print(f'Downloading {url}...')
        with urllib.request.urlopen(url) as response, open(out_filename, 'wb') as out_file:
            shutil.copyfileobj(response, out_file)
        print(f'Saved to {out_filename}.')
    return out_filename

DATA_path_1 = download_data(DATA_DIR, DATA_URL_1, False)
DATA_path_2 = download_data(DATA_DIR, DATA_URL_2, False)
```

```
Downloading https://raw.githubusercontent.com/afiretony/NLDML_hw3/main/data/hw3_ex1.csv...
Saved to C:\Users\chenhao\datasets\hw3_ex1.csv.
Downloading https://raw.githubusercontent.com/afiretony/NLDML_hw3/main/data/hw3_ex2.csv...
Saved to C:\Users\chenhao\datasets\hw3_ex2.csv.
```

In [3]:
```
# read data
df_1 = pd.read_csv(DATA_path_1)
x = df_1[['x']].values
y = df_1[['y']].values
m = x.size
print("First five entires of the table")
df_1.head()
```

```
First five entires of the table
```

Out[3]:

	x	y
0	0.00000	0.0276
1	0.00503	0.0632
2	0.01010	0.0853
3	0.01510	0.0662
4	0.02010	0.0325

2. (8 points) Plot the loss function

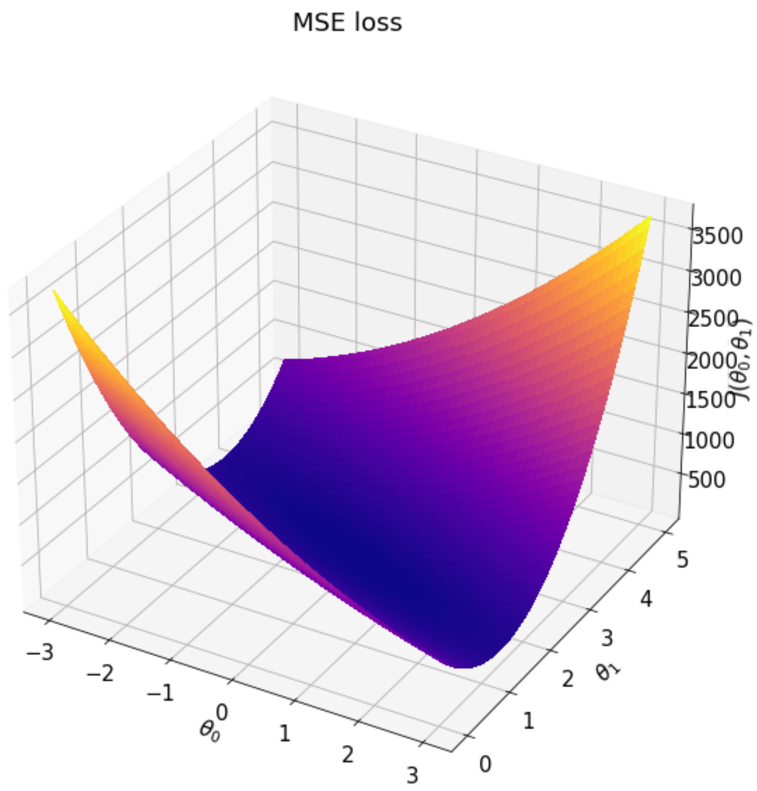
In [4]:
```
# create parameters
theta0 = np.linspace(-3,3,100)
theta1 = np.linspace(0,5,100)
J = np.zeros((theta0.size, theta1.size))
theta0_mesh, theta1_mesh = np.meshgrid(theta0, theta1)

def cost_func(x, y, theta0=0, theta1=0):
  y_pred = np.matmul(np.eye(m)*theta1, x) + np.ones((m,1))*theta0
  return np.matmul(np.transpose(y-y_pred), (y-y_pred))

for i in range(theta0.size):
```

```
    for j in range(theta1.size):
      J[i,j] = cost_func(x, y, theta0[i], theta1[j])
```

In [5]:
```
fig = plt.figure(figsize=(15,10))
ax = fig.gca(projection='3d')
# Plot the surface.
surf = ax.plot_surface(theta0_mesh, theta1_mesh, J,linewidth=0, antialiased=False, cmap='plasma')
# Customize the z axis.
# ax.set_zlim(-1.01, 1.01)
plt.title('MSE loss')
plt.xlabel(r'$\theta_0$')
plt.ylabel(r'$\theta_1$')
ax.set_zlabel(r'$J(\theta_0, \theta_1)$')
plt.show()
```



3. (8 points) Graphical approach

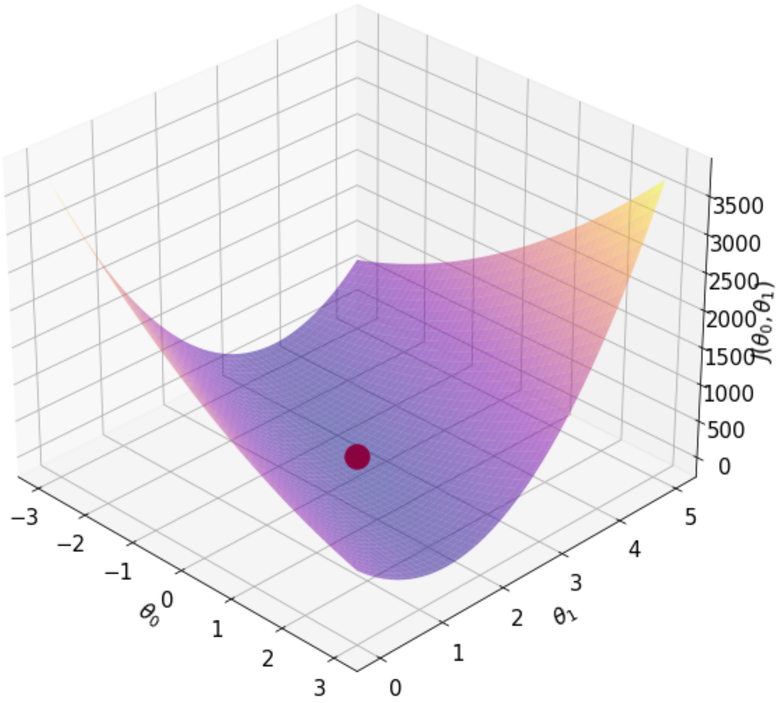
To get the indices of the minimum elements of a N-dimensional array, we used `argmin` method.

In [6]:
```
min_ind = np.unravel_index(np.argmin(J, axis=None), J.shape)
print("Minimum value of the cost function J is", J[min_ind[0]][min_ind[1]])
print("When [theta 0, theta 1] = ", [theta0[min_ind[0]], theta1[min_ind[1]]])
```

```
Minimum value of the cost function J is 0.3925612032282422
When [theta 0, theta 1] =  [-0.030303030303030276, 2.525252525252525]
```

In [7]:
```
fig = plt.figure(figsize=(15,10))
ax = fig.gca(projection='3d')
surf = ax.plot_surface(theta0_mesh, theta1_mesh, J,linewidth=0, antialiased=True, alpha=0.5, cmap='plasma')

plt.xlabel(r'$\theta_0$')
plt.ylabel(r'$\theta_1$')
ax.set_zlabel(r'$J(\theta_0, \theta_1)$')
ax.scatter(theta0[min_ind[0]], theta1[min_ind[1]], J[min_ind],'r', s=300, color = 'red')
ax.view_init(30, -45)
plt.show()
```



4. (10 points) Calculate θ_0 and θ_1 using the linear regression algorithm from Scikit-Learn package

In [8]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(x, y)
theta0_pred, theta1_pred = reg.intercept_[0], reg.coef_[0][0]
print("When (theta 0, theta 1) =", (theta0_pred, theta1_pred))
print("The regression score is", reg.score(x, y))
```

```
When (theta 0, theta 1) = (0.0014203115286064438, 2.496092823473767)
The regression score is 0.9969629687544361
```

5. (8 points) Calculate the optimal θ_0 and θ_1 using the analytical approach

$$\theta_1^{opt} = \frac{mS_{xy} - S_x S_y}{mS_{xx} - S_x^2}, \ \theta_0^{opt} = \overline{y} - \theta_1^{opt}\overline{x}$$

In [9]:
```
Sx, Sy = x.sum(), y.sum()
Sxx = x.T @ x
Sxy = x.T @ y
theta_opt1 = (m*Sxy-Sx*Sy)/(m*Sxx-Sx**2)
theta_opt0 = y.mean() - theta_opt1*x.mean()
print("Using analytical method: (theta 0, theta 1) = ", (float(theta_opt0), float(theta_opt1)))
```

```
Using analytical method: (theta 0, theta 1) =  (0.0014203115286079981, 2.496092823473764)
```

6. (8 points) Compare the linear fitting curves obtained by the graphical approach, Sklearn package, and the analytical approach by plotting them on a single graph with the datapoints

In [10]:
```
y_pred1 = theta0[min_ind[0]] + theta1[min_ind[1]] * x
y_pred2 = theta0_pred + theta1_pred * x
y_pred3 = theta_opt0 + theta_opt1 * x

def RMSE(y=y, y_pred=y_pred1):
  diff = y - y_pred
  temp = diff.T @ diff / y.size
  return float(np.sqrt(temp))

print('The root mean square error of graphical method: %.5f' %RMSE(y, y_pred1))
print('The root mean square error of Scikit-Learn package:%.5f' %RMSE(y, y_pred2))
print('The root mean square error of analytical method:%.5f' %RMSE(y, y_pred3))
```

```
The root mean square error of graphical method: 0.04430
The root mean square error of Scikit-Learn package:0.03997
The root mean square error of analytical method:0.03997
```

RMSE(graphical method) > RMSE(Scikit_Learn) $\approx$ RMSE(analytical method)

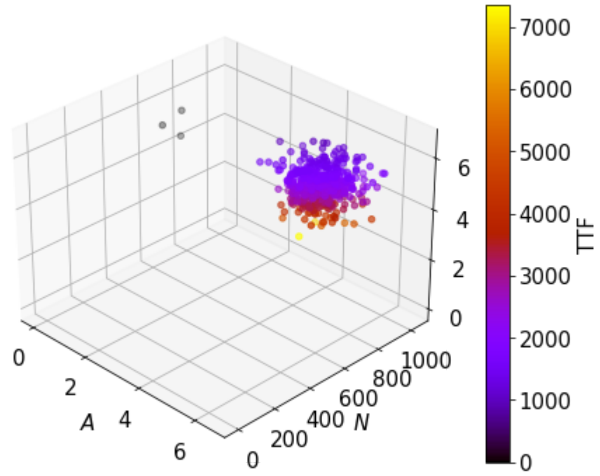
However, the difference between Scikit Learn and analytical method is very small $O(10^{-16})$, they are almost the same. But the root mean square error of graphical method is larger than those two, because the accuracy of graphical method mainly relies on the precision of parameter grid which is not high enough.

Part (II) 2. Multivariate regression- Artificial Neural Networks

1. (6 points) Data exploration and visualization

In [11]:
```
df_2 = pd.read_csv(DATA_path_2)
A = df_2[['A']].values
N = df_2[['N']].values
F = df_2[['F']].values
TTF = df_2[['TTF']].values
m = A.size
print("First five entires out of",m,"entries")
print(df_2.head())
fig = plt.figure(figsize=(10,6))
ax = fig.gca(projection='3d')
plt.xlabel(r'$A$')
plt.ylabel(r'$N$')
ax.set_zlabel(r'$F$')
ax.view_init(30, -45)
sc = ax.scatter(A,N,F,c=TTF,cmap='gnuplot')
cb = plt.colorbar(sc)
cb.set_label('TTF')
plt.show()
```

```
First five entires out of 500 entries
      A      N     F     TTF
0  0.00  770.0  4.39     0.0
1  4.56  962.0  5.58  1220.0
2  6.27  703.0  5.60  2330.0
3  5.73  698.0  6.31  1530.0
4  4.87  688.0  5.71  1740.0
```



We can see that there are several outliers. They meet the below criterion. TTF >3000. Also, we can see there are three gray dots on the left side of the figure. We can inspect this in details through data inspection.

2. (6 points) Data pre-processing, data cleaning

We just simply use `between` method to keep the data and discard the outliers.

In [12]:
```
# remove outliners
df_2 = pd.read_csv(DATA_path_2)
TTF = df_2['TTF']
m = TTF.size
removed_outliers = TTF.between(TTF.quantile(.05), TTF.quantile(.95))
df_2 = df_2[removed_outliers].reset_index(drop=True)
A = df_2['A']
removed_outliers = A.between(A.quantile(.01), A.quantile(.99))
df_2 = df_2[removed_outliers].reset_index(drop=True)
N = df_2['N']
removed_outliers = N.between(N.quantile(.01), N.quantile(.99))
df_2 = df_2[removed_outliers].reset_index(drop=True)
# F = df_2['F']
# removed_outliers = F.between(F.quantile(.03), F.quantile(.97))
# df_2 = df_2[removed_outliers].reset_index(drop=True)

m_new= df_2['TTF'].size
print("We removed",m-m_new, "outliers.")
```

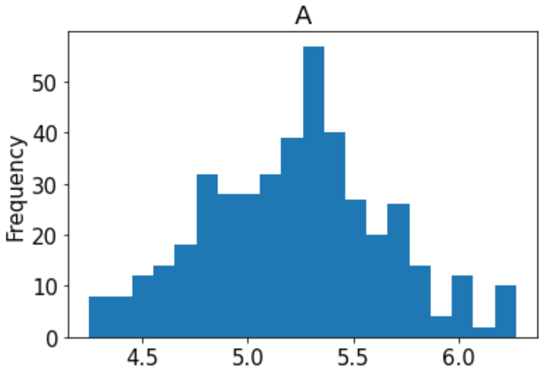
```
We removed 69 outliers.
```

3. (6 points) Data balance exploration

We can see most of the features follow normal distribution after removing the outliers. Some of the distribution might be skewed.

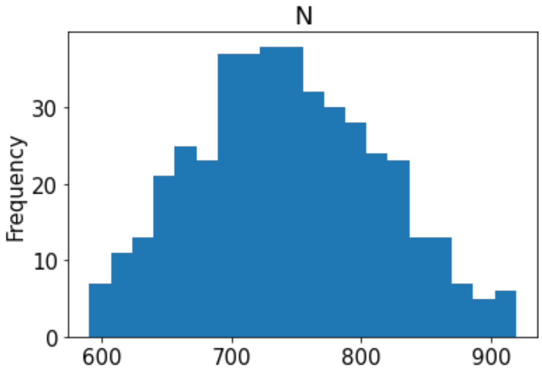
In [13]:
```
df_2['A'].plot.hist(bins=20,title='A')
```

Out[13]:
```
<AxesSubplot:title={'center':'A'}, ylabel='Frequency'>
```



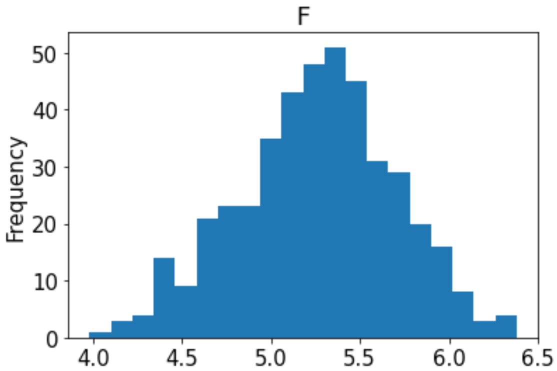
In [14]:
```
df_2['N'].plot.hist(bins=20,title='N')
```

Out[14]: <AxesSubplot:title={'center':'N'}, ylabel='Frequency'>



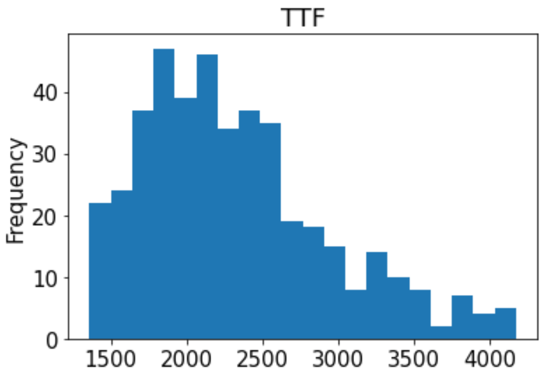
In [15]:
```
df_2['F'].plot.hist(bins=20,title='F')
```

Out[15]: <AxesSubplot:title={'center':'F'}, ylabel='Frequency'>



In [16]:
```
df_2['TTF'].plot.hist(bins=20,title='TTF')
```

Out[16]: <AxesSubplot:title={'center':'TTF'}, ylabel='Frequency'>



4. (6 points) Data normalization- Define a new dataframe, in which each column contains the z-score

In [17]:
```
# z-score normalization (Standardization)
normalized_df = (df_2-df_2.mean())/df_2.std()

# save for future use
# reverse: normalized_df * std + mean
```

```
mean = df_2.mean()
std = df_2.std()

A = normalized_df[['A']].values
N = normalized_df[['N']].values
F = normalized_df[['F']].values
TTF = normalized_df[['TTF']].values
m = A.size
normalized_df.head()
```

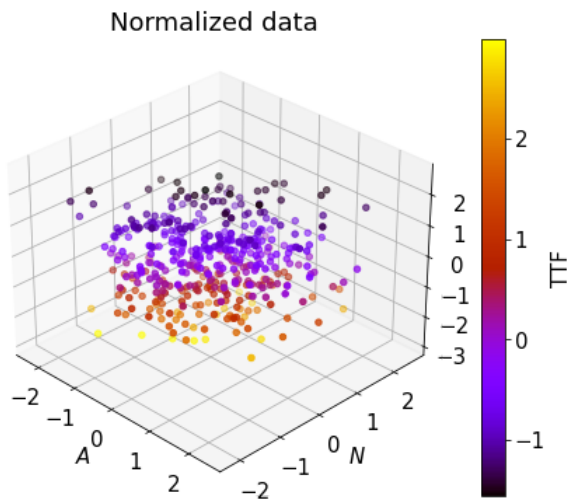
Out[17]:

	A	N	F	TTF
0	2.439062	-0.569937	0.795151	0.010228
1	1.177579	-0.640621	2.410528	-1.286281
2	-0.831449	-0.781990	1.045421	-0.945947
3	0.967332	0.037950	1.159180	-0.832503
4	2.322258	1.055806	0.021590	0.139879

5. (6 points) Last check before prediction

In [18]:

```
fig = plt.figure(figsize=(10,6))
ax = fig.gca(projection='3d')
plt.xlabel(r'$A$')
plt.ylabel(r'$N$')
plt.title(r'Normalized data')
ax.set_zlabel(r'$F$')
ax.view_init(30, -45)
sc = ax.scatter(A,N,F,c=TTF,cmap='gnuplot')
cb = plt.colorbar(sc)
cb.set_label('TTF')
plt.show()
```

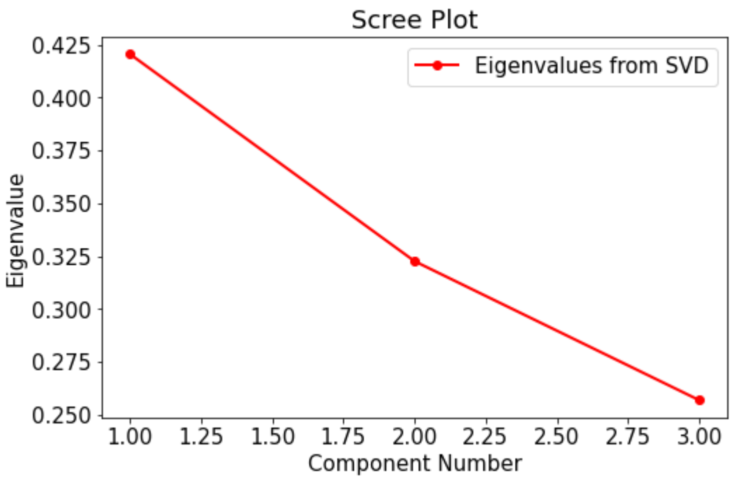


In [19]:

```
mat = np.hstack((A, N))
mat = np.hstack((mat, F))

U, S, V = np.linalg.svd(mat)
eigvals = S**2 / np.sum(S**2)

fig = plt.figure(figsize=(8,5))
sing_vals = np.arange(3) + 1
plt.plot(sing_vals, eigvals, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Component Number')
plt.ylabel('Eigenvalue')
leg = plt.legend(['Eigenvalues from SVD'], loc='best')
plt.show()
```



From the plot, we can see that the relative variances of all the features are not similar. The eigen value of feature `A`, `N`, `F` is 0.42059232, 0.32258397, 0.2568237 respectly. So feature `A` has the highest relative varience and feature `F` has the lowest varience.

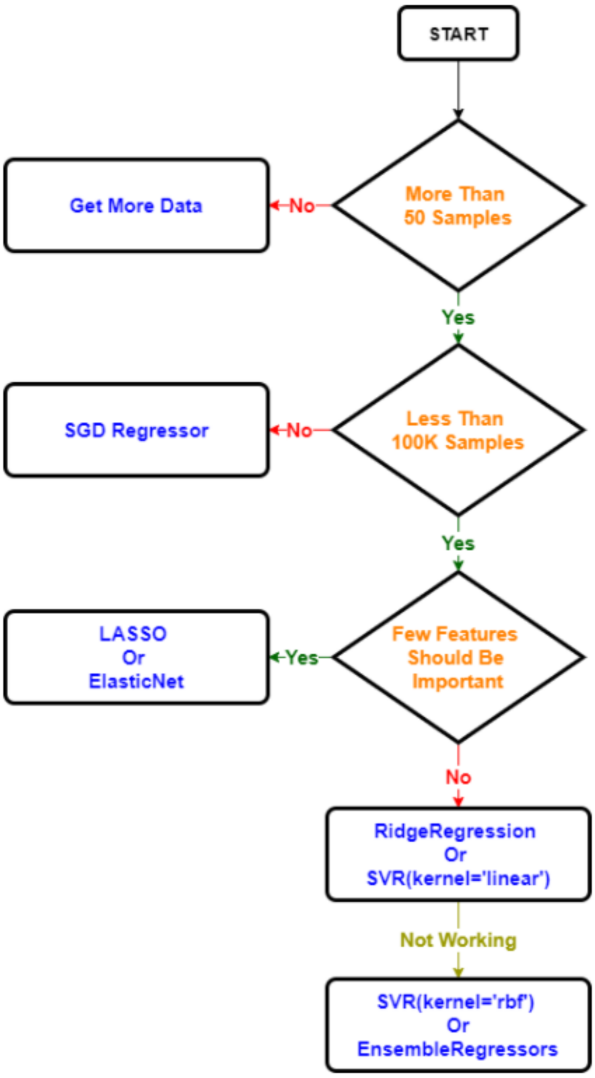
6. (6 points) Data splitting

In [20]:
```
import sklearn
ds_train, ds_test = sklearn.model_selection.train_test_split(normalized_df,test_size=0.2)
print("There are ",len(ds_train),"train sets and ",len(ds_test),"test sets.")
```

```
There are  344 train sets and  87 test sets.
```

7. (6 points) Linear Regression

We have 300 train sets and all of the features are important to learn. For this size of datasets, we can utilize `RigidRegression` from the



`sklearn` library.

source : https://satishgunjal.com/multivariate_lr_scikit/

In [21]:
```
from sklearn import linear_model
model_r = linear_model.Ridge(alpha=0.3)
```

```
X_train = pd.concat([ds_train["A"],ds_train["N"],ds_train["F"]], axis = 1, ignore_index=True)
y_train = ds_train["TTF"]
model_r.fit(X_train,y_train)
print('coeff= ' , model_r.coef_)
print('intercept= ' , model_r.intercept_)
```

```
coeff=  [ 0.30394867 -0.35431821 -0.97441096]
intercept=  0.004933062789906276
```

In [22]:

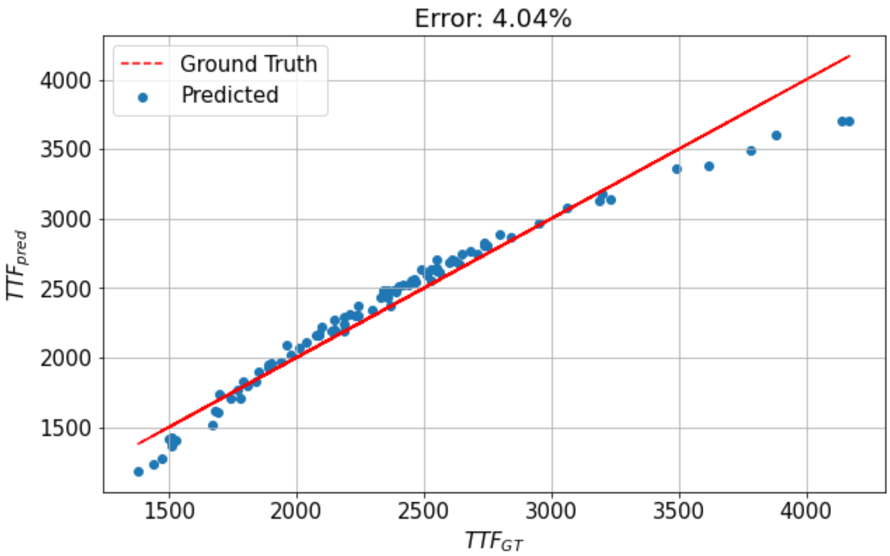
```
X_test = pd.concat([ds_test["A"],ds_test["N"],ds_test["F"]], axis = 1, ignore_index=True)
y_test = ds_test["TTF"]
y_pred = model_r.predict(X_test)

y_test_denorm = y_test * std[-1] + mean[-1]
y_pred_denorm = y_pred * std[-1] + mean[-1]

def MAPE(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

error = MAPE(y_test_denorm,y_pred_denorm)

fig= plt.figure(figsize=(10,6))
plt.title('Error: {:.2%}'.format(error/100))
plt.xlabel(r'$TTF_{GT}$')
plt.ylabel(r'$TTF_{pred}$')
plt.grid()
plt.scatter(y_test_denorm,y_pred_denorm,label='Predicted')
plt.plot(y_test_denorm,y_test_denorm,'r--',label='Ground Truth')
plt.legend()
plt.show()
```



The linear regression model succeed to achieve good predictions on the test-set.

In [23]:

```
print('The error of prediction on testset is: {:.2%}'.format(error/100))
print('The success of prediction on testset is: {:.2%}'.format(1-error/100))
```

```
The error of prediction on testset is: 4.04%
The success of prediction on testset is: 95.96%
```

The regression model succeeded to achieve an acceptable predictions on the test-set. This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. It can be used as a multivariate regression model.

8. (8 points) Fully-connected deep neural network

In [25]:

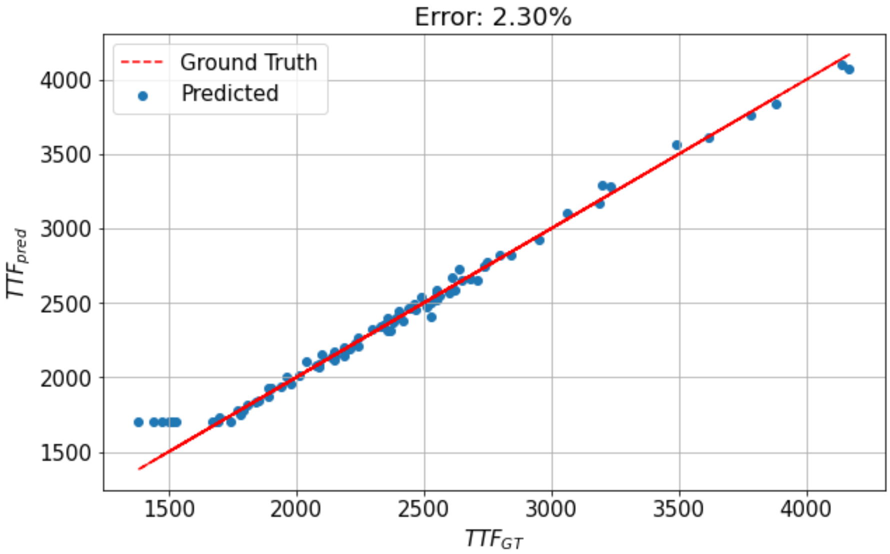
```
from sklearn.neural_network import MLPRegressor
MLP = MLPRegressor(hidden_layer_sizes=(64,64,1),
                   max_iter=250,
                   solver='adam',
                   alpha=0.001,
                   random_state=420,
                   n_iter_no_change=100,
                   verbose=False)

y_pred_MLP = MLP.fit(X_train, y_train).predict(X_test)
y_pred_MLP_denorm = y_pred_MLP * std[-1] + mean[-1]

error_MLP = MAPE(y_test_denorm,y_pred_MLP_denorm)

fig= plt.figure(figsize=(10,6))
plt.title('Error: {:.2%}'.format(error_MLP/100))
plt.xlabel(r'$TTF_{GT}$')
plt.ylabel(r'$TTF_{pred}$')
plt.grid()
plt.scatter(y_test_denorm,y_pred_MLP_denorm,label='Predicted')
plt.plot(y_test_denorm,y_test_denorm,'r--',label='Ground Truth')
plt.legend()
```

```
plt.show()
print('The success of prediction on testset is: {:.2%}'.format(1-error_MLP/100))
```




```
The success of prediction on testset is: 97.70%
```

The FC-DNN model obtained better prediction results on the test-set in comparison to the linear regression model. In our case, success rate of prediction using linear regression model is 95.80% and using FC-DNN (hidden layer: 64-64-1) is 97.49%. We even get better result (98.95%) when using a model whose hidden layer is 64-32-16. This is because the non-linear characteristic of our data as we can inspect from the beginning. Neural networks outperform linear regression because they deal with non linearities automatically, whereas in linear regression you need to mention explicitly.