In [2]:
```
import pandas as pd
import numpy as np
from  math import *
from scipy.interpolate import interp1d

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import cross_val_score
import scipy.stats as stat
import pylab
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, roc_auc_score, recall_score, precision_s
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import classification_report,confusion_matrix,plot_confusion_ma
from sklearn.model_selection import cross_val_score
```

In [4]:
```
df  = pd.read_csv("staggered_RAW_DATA.csv")
```

In [5]:
```
df  = df.convert_dtypes()
```

In [6]:
```
df
```

Out[6]:

	filling	PPI	Porosity	P/L	h_wall	Nu_wall	DeltaT
0	1.0	5	0.8	457.891233	26.224101	8.190942	8.643194
1	1.0	10	0.8	1379.951933	30.962416	9.670926	7.320488
2	1.0	15	0.8	2739.221533	33.169437	10.360276	6.833399
3	1.0	20	0.8	4526.8568	34.415382	10.749439	6.586008
4	1.0	25	0.8	6738.152	35.198431	10.99402	6.439492
...	...	...	...	...	...	...	...
75	0.25	5	0.95	34.908639	17.638211	21.123863	12.850509
76	0.25	10	0.95	100.303273	20.67384	24.75939	10.963614
77	0.25	15	0.95	194.66478	22.303387	26.710967	10.162582
78	0.25	20	0.95	317.425527	23.306586	27.912418	9.725148
79	0.25	25	0.95	468.253393	23.974496	28.71232	9.454213

80 rows × 7 columns

PROGRAM FOR INTERPOLATION

In [252]:
```
x = df['h_wall']
```

In [253]:
```
x
```

Out[253]:
```
0     20.955562
1     26.142631
2     29.034629
3     30.885388
4     32.165965
        ...
75    14.789964
76    17.660218
77    19.447801
78    20.695163
79    21.616499
Name: h_wall, Length: 80, dtype: Float64
```

In [254]:
```
y1  = df['P/L']
```

In [255]:
```
f = interp1d(x, y1)
```

In [256]:
```
def interploate(y ,x1 , y1 , x2 , y2):
    x = (x1*(y-y2) - x2*(y-y1))/(y1-y2)
    return x
```

In [257]:
```
y1 = 0.80
y2 = 0.85
y = 0.845


nu1 = []
nu2 =[]
h1=[]
h2=[]
P1 =[]
```

```
P2 =[]
for  i in  range(len(df['Porosity'])):
    if df['Porosity'][i]==y1:
        nu1.append(df['Nu_wall'][i])
        h1.append(df['h_wall'][i])
        P1.append(df['P/L'][i])
    if df['Porosity'][i]==y2:
        nu2.append(df['Nu_wall'][i])
        h2.append(df['h_wall'][i])
        P2.append(df['P/L'][i])
arr = []
for i in  range(len(nu1)):
    arr.append(interploate(y ,float(nu1[i]) ,y1 ,float(nu2[i]), y2))
arr
```

Out[257]:

```
[10.281820829299999,
 12.8249504333,
 14.226808028999999,
 15.116707276899996,
 15.728706487999998,
 9.7070674709,
 12.3245863582,
 13.833127636699999,
 14.8201103264,
 15.5132299042,
 8.181987605,
 10.0137848398,
 11.106065522299998,
 11.8415112306,
 12.369509673799998,
 6.6818351799,
 7.922918501,
 8.695771355599998,
 9.237874408699998,
 9.6397308253]
```

In [ ]:

DATA PREPROCESSING

In [258]:

```
df.head()
```

Out[258]:

	filling	PPI	Porosity	P/L	h_wall	Nu_wall	DeltaT
0	1.0	5	0.8	123.196607	20.955562	6.545345	10.816222
1	1.0	10	0.8	328.563047	26.142631	8.165495	8.67013
2	1.0	15	0.8	608.814787	29.034629	9.068793	7.80654
3	1.0	20	0.8	961.523733	30.885388	9.646867	7.338745
4	1.0	25	0.8	1385.303867	32.165965	10.046847	7.046579

In [259]:

```
df.describe()
```

Out[259]:

	filling	PPI	Porosity	P/L	h_wall	Nu_wall	DeltaT
count	80.000000	80.000000	80.000000	80.000000	80.000000	80.000000	80.000000
mean	0.625000	15.000000	0.875000	275.604029	25.049549	17.684996	9.639715
std	0.281272	7.115681	0.056254	273.210239	6.140894	10.564759	2.524360
min	0.250000	5.000000	0.800000	13.523349	13.918894	4.347484	6.148549
25%	0.437500	10.000000	0.837500	80.741098	20.097634	9.358972	7.486743
50%	0.625000	15.000000	0.875000	186.743760	24.807700	15.099619	9.138325
75%	0.812500	20.000000	0.912500	364.109502	30.274846	24.406253	11.278009
max	1.000000	25.000000	0.950000	1385.303867	36.863983	44.149019	16.284340

In [260]:
```
df.shape
```

Out[260]:
```
(80, 7)
```

In [261]:
```
df.info
```

Out[261]:
```
<bound method DataFrame.info of     filling  PPI  Porosity          P/L     h_wall   
Nu_wall     DeltaT
0       1.0    5       0.8   123.196607  20.955562   6.545345  10.816222
1       1.0   10       0.8   328.563047  26.142631   8.165495   8.67013
2       1.0   15       0.8   608.814787  29.034629   9.068793   7.80654
3       1.0   20       0.8   961.523733  30.885388   9.646867   7.338745
4       1.0   25       0.8  1385.303867  32.165965  10.046847   7.046579
..      ...  ...       ...          ...        ...        ...        ...
75     0.25    5      0.95    13.523349  14.789964  17.712746  15.325257
76     0.25   10      0.95    35.163144  17.660218  21.150218  12.834496
77     0.25   15      0.95    64.426884  19.447801  23.291062  11.654788
78     0.25   20      0.95   101.108913  20.695163  24.784926  10.952318
79     0.25   25      0.95    145.09478  21.616499  25.888337  10.485509

[80 rows x 7 columns]>
```

In [262]:
```
df.isnull().sum()
```

Out[262]:
```
filling     0
PPI         0
Porosity    0
P/L         0
h_wall      0
Nu_wall     0
DeltaT      0
dtype: int64
```

In [263]:
```
testing = pd.read_csv("features.csv")
```

In [264]:
```
testing
```

Out[264]:

	filling	PPI	Porosity
0	1.00	5	0.80
1	1.00	10	0.80
2	1.00	15	0.80
3	1.00	20	0.80
4	1.00	25	0.80
...	...	...	...
595	0.25	5	0.95
596	0.25	10	0.95
597	0.25	15	0.95
598	0.25	20	0.95
599	0.25	25	0.95

600 rows × 3 columns

In [265]:
```
testing  = np.array(testing)
```

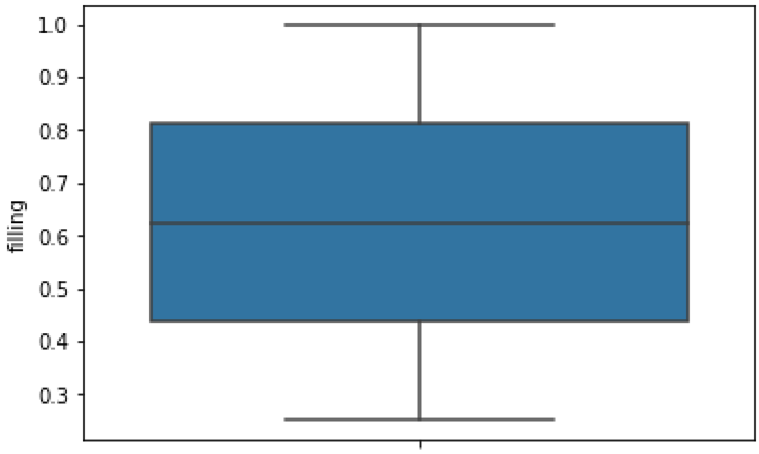
In [ ]:

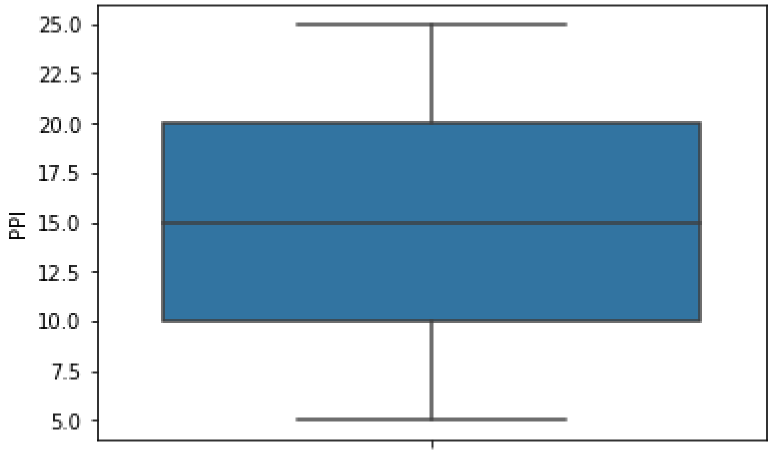
In [ ]:

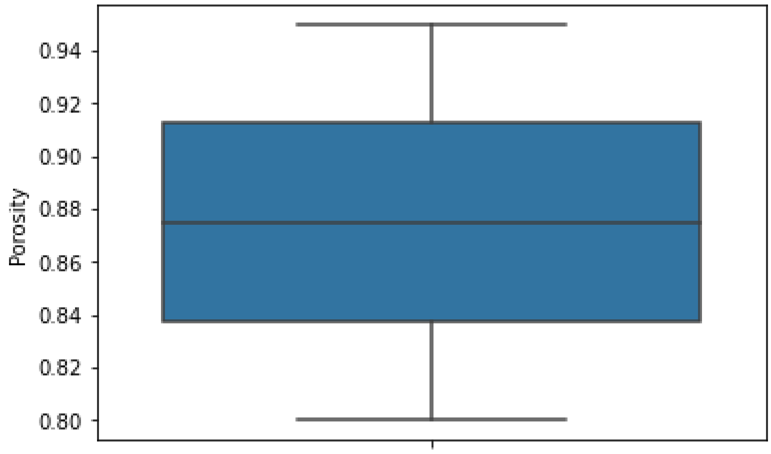
BOX PLOT TO CHECK IF THERE IS ANY OUTLIERS

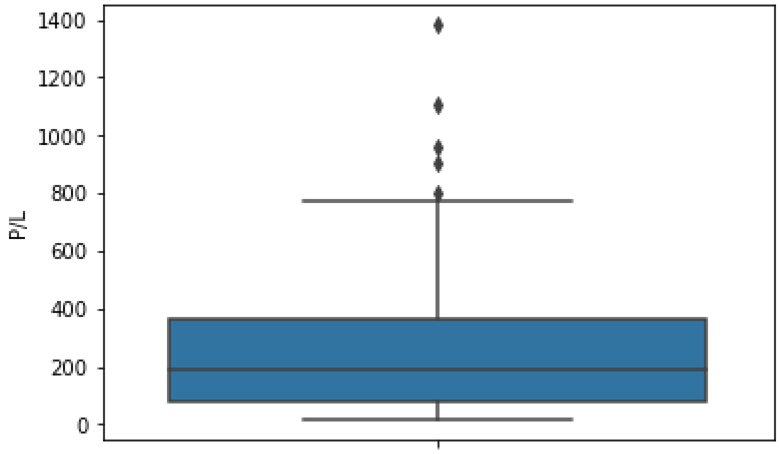
In [266]:
```
feature = []
for  i in  df.columns :
    if df.dtypes[i]!='object':
        feature.append(i)
```

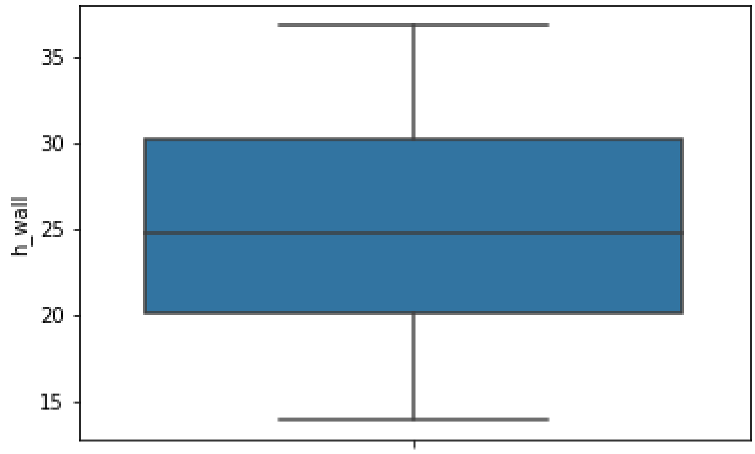
In [267]:
```
for i in  feature:
    plt.figsize=(10,5)
    sns.boxplot(y=df[i])
    plt.show()
```

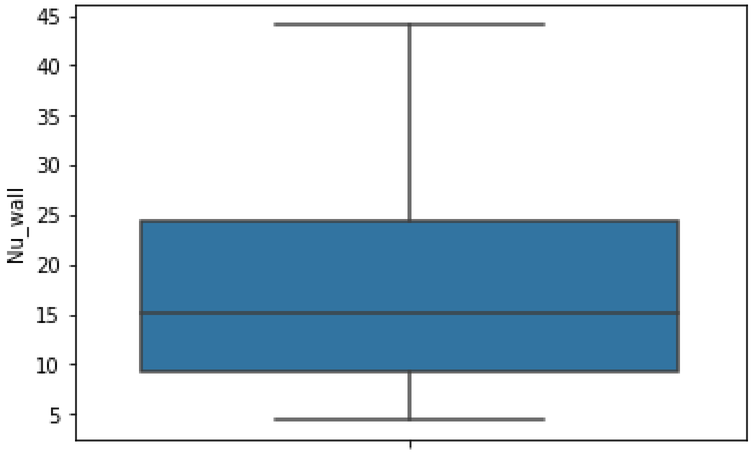


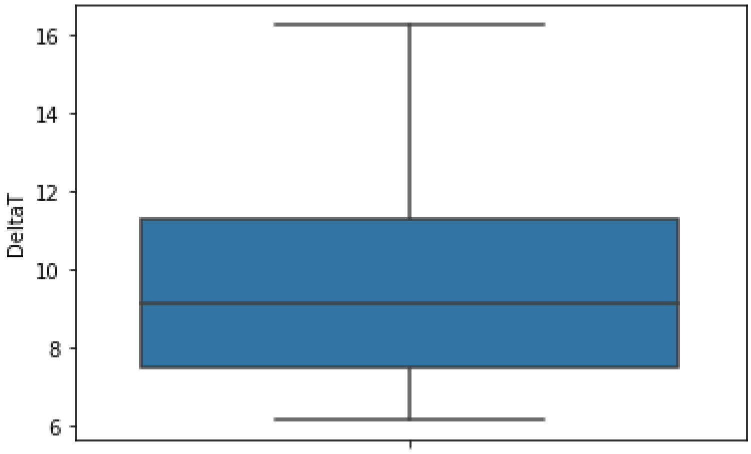

PPI
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0


Porosity
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80


P/L
1400
1200
1000
800
600
400
200
0

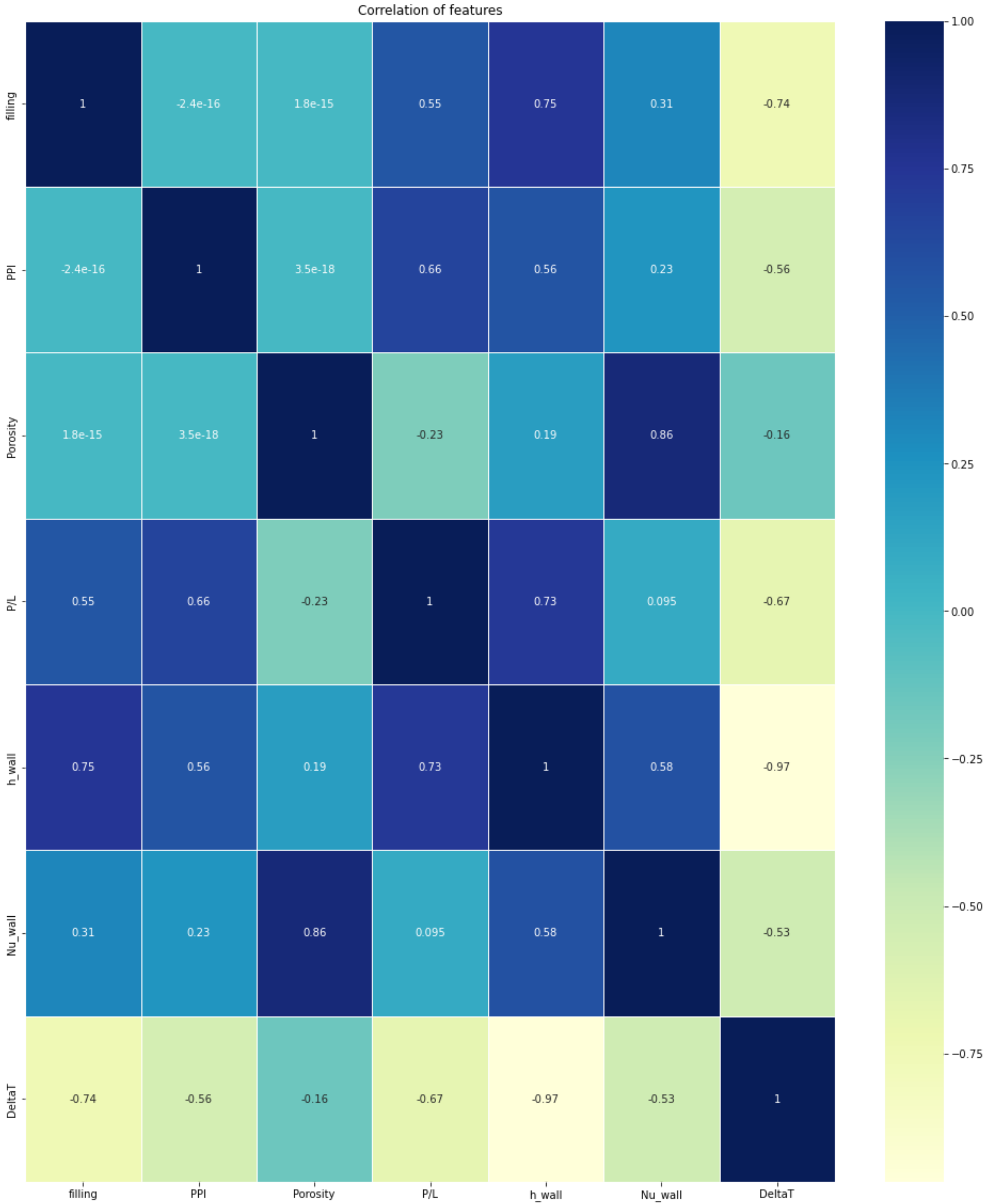

h_wall
35
30
25
20
15





In [268]:
```
plt.figure(figsize=(17, 20))
plt.title('Correlation of features')
sns.heatmap(df.corr(), annot=True, linewidths=0.5, cmap="YlGnBu")
```

Out[268]:
```
<AxesSubplot:title={'center':'Correlation of features'}>
```



In [18]:
```
def curve_plot(df , feature):
    print(feature+ ':')
    plt.figure(figsize=(10,6))
   # plt.subplot(1,2,1)
   # df[feature].hist()
    #plt.plot(1,2,2)
    stat.probplot(df[feature],dist='norm',plot=pylab)
    plt.show()
```

In [11]:
```
df.dtypes
```

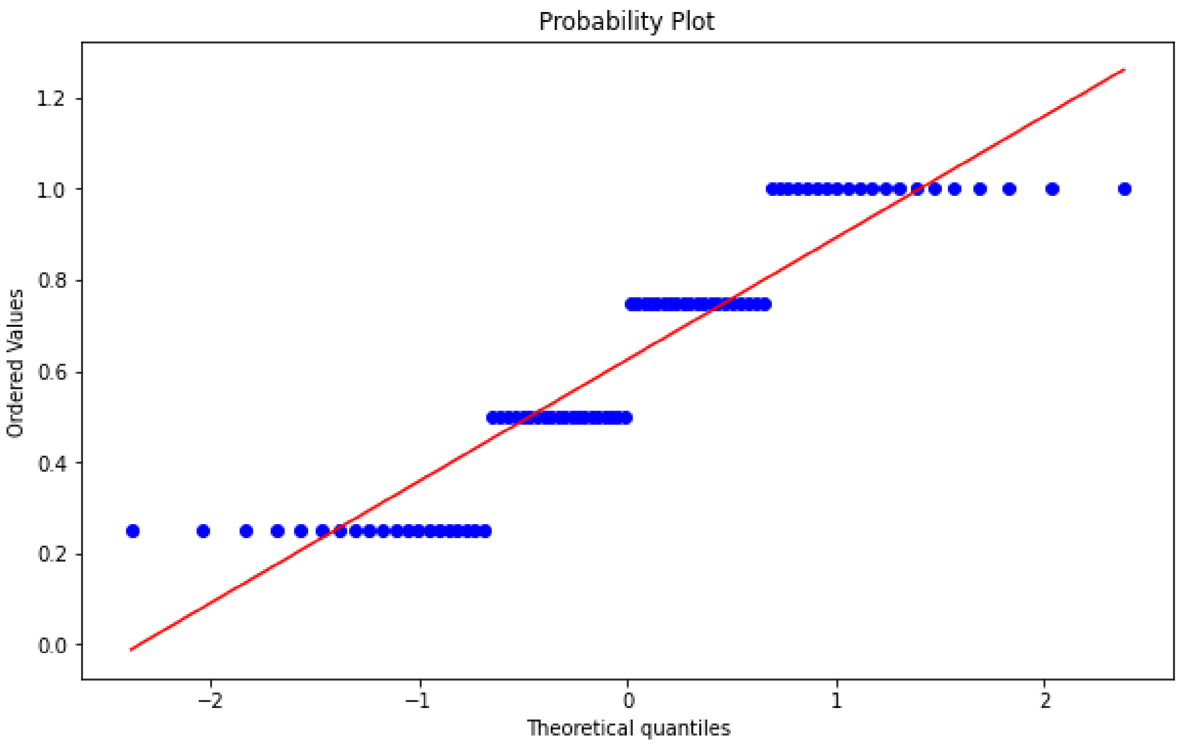
Out[11]:
```
filling      float64
PPI          float64
Porosity     float64
P/L          float64
h_wall       float64
Nu_wall      float64
DeltaT       float64
dtype: object
```

In [12]:
```
df['PPI'] = df['PPI'].astype(float)
df['Porosity'] = df['Porosity'].astype(float)
df['Nu_wall'] = df['Nu_wall'].astype(float)
df['h_wall'] = df['h_wall'].astype(float)
df['DeltaT'] = df['DeltaT'].astype(float)
df['P/L'] = df['P/L'].astype(float)
df['filling'] = df['filling'].astype(float)
```

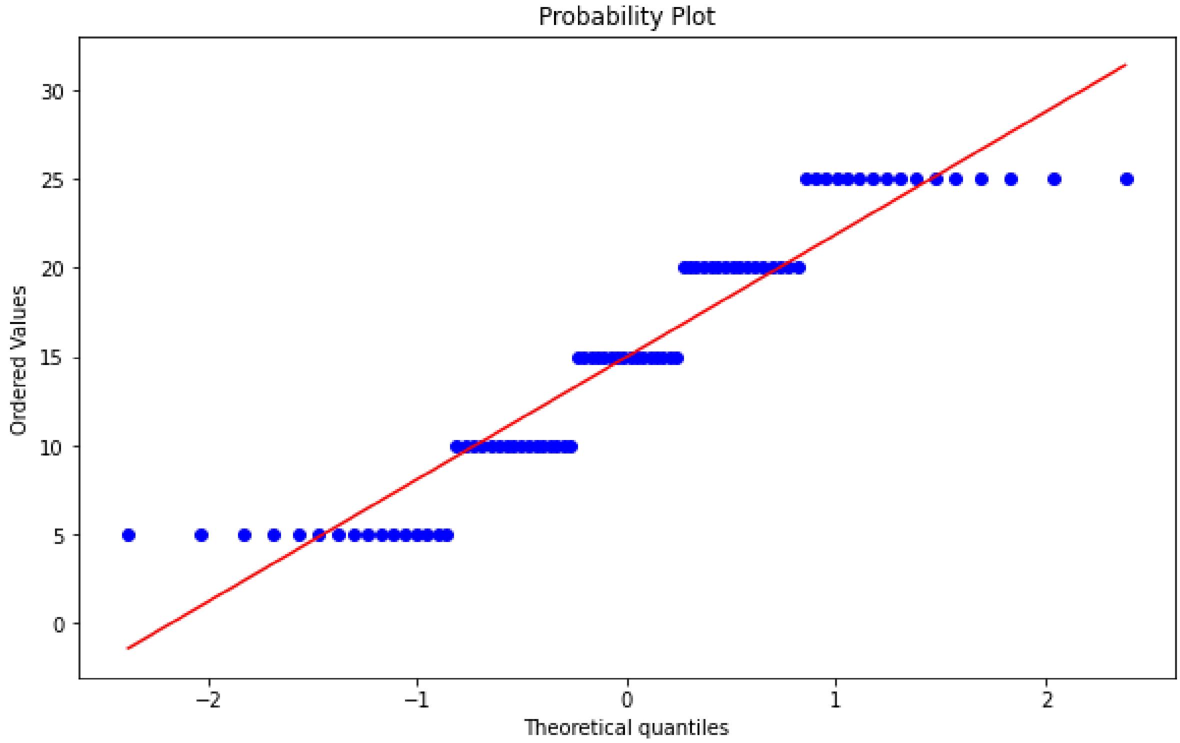
In [22]:
```
df['DeltaT'] = np.log(df['DeltaT'])
```

In [23]:
```
for i in  df.columns:
    curve_plot(df ,i)
```

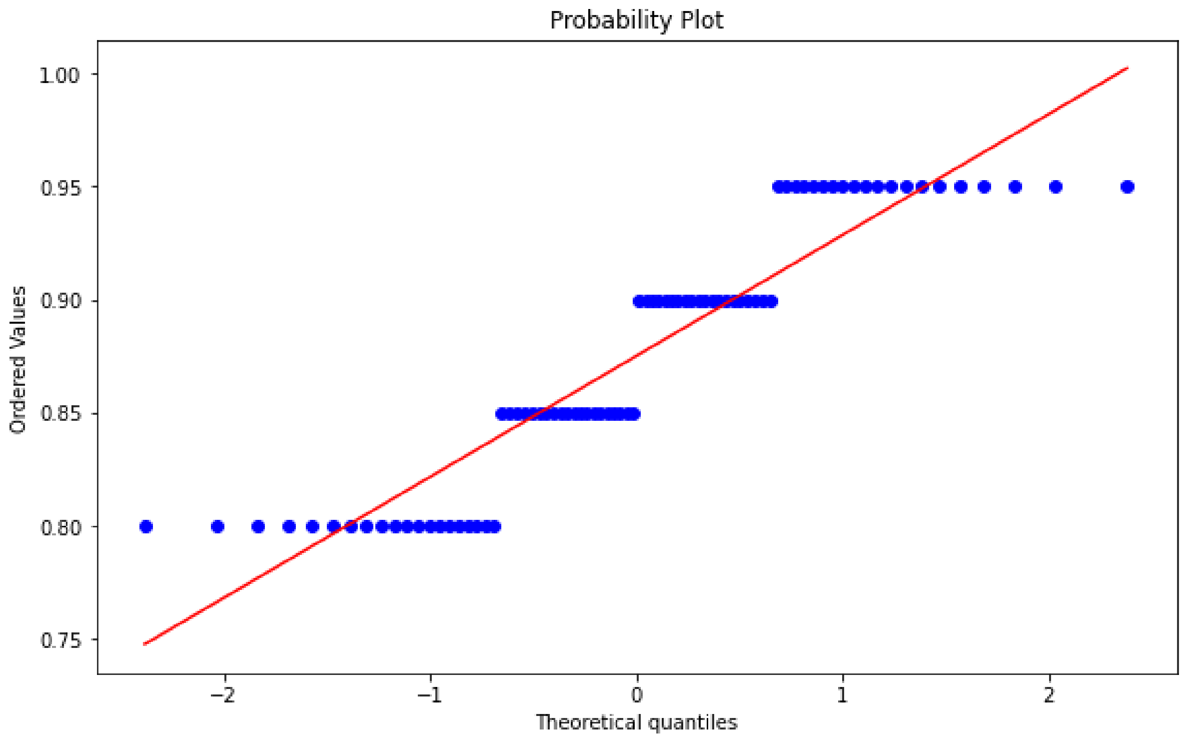
filling:



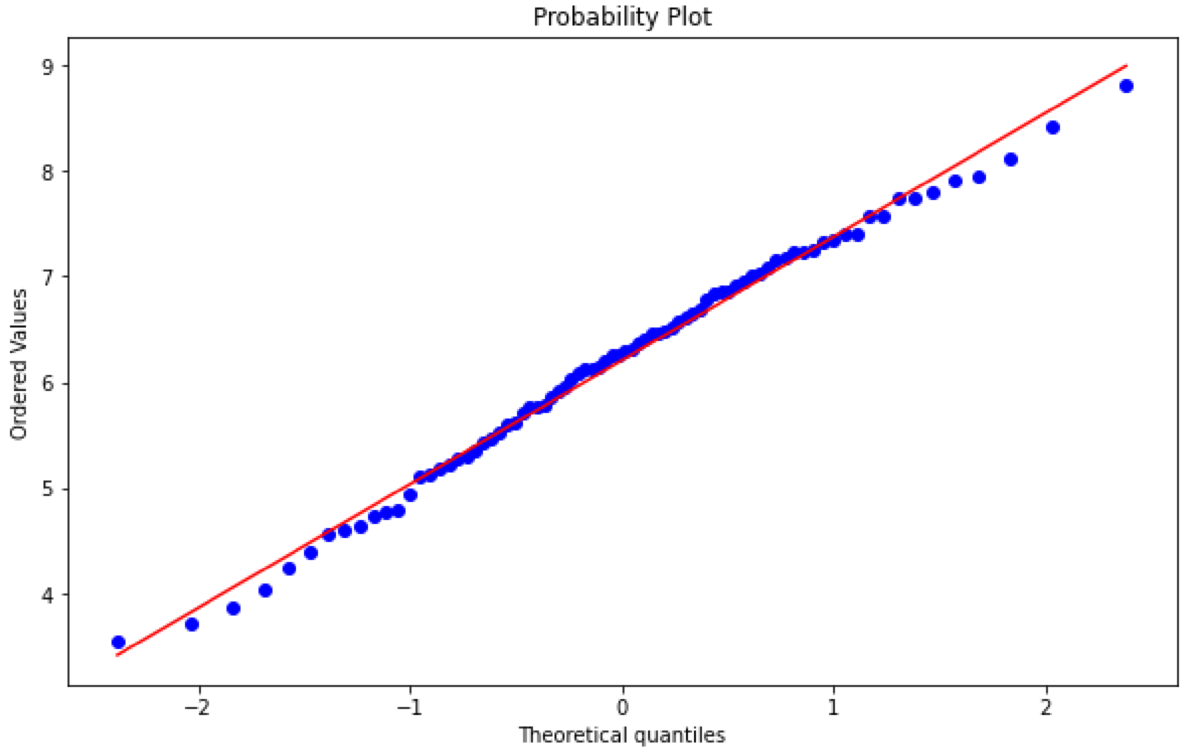
PPI:


Probability Plot
Ordered Values
Theoretical quantiles
30
25
20
15
10
5
0
−2
−1
0
1
2

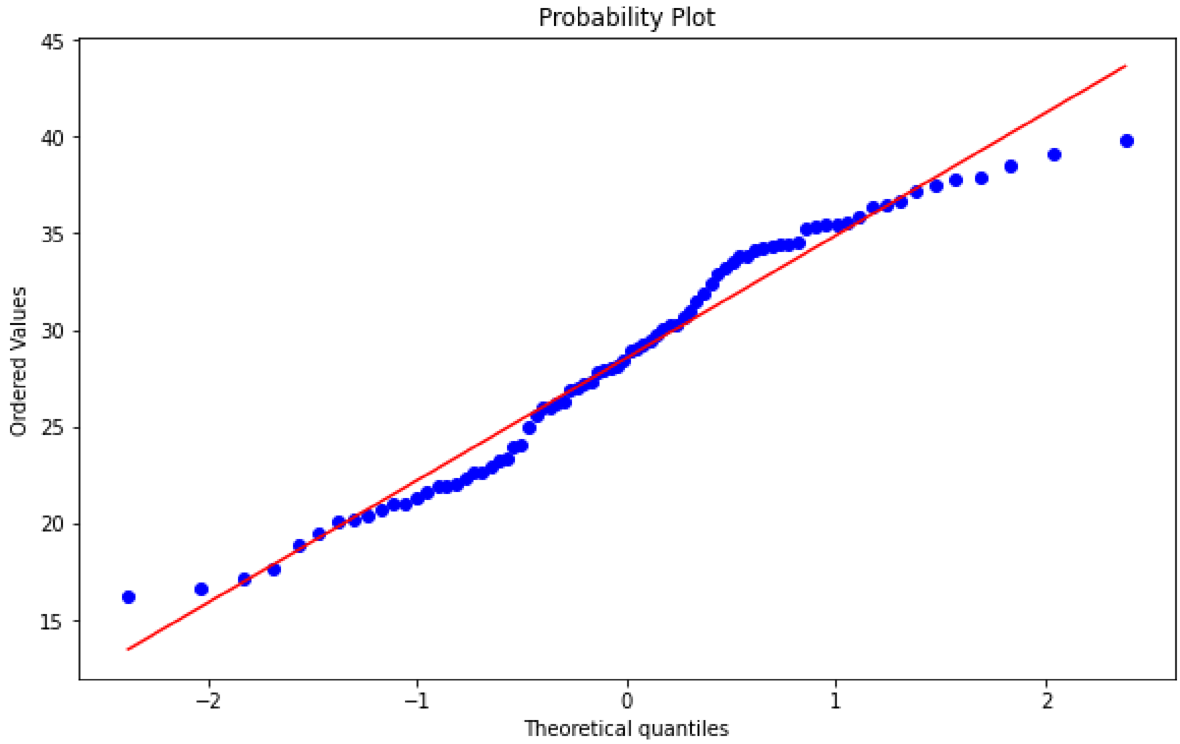
```
Porosity:
```


Probability Plot
Ordered Values
Theoretical quantiles
1.00
0.95
0.90
0.85
0.80
0.75
−2
−1
0
1
2

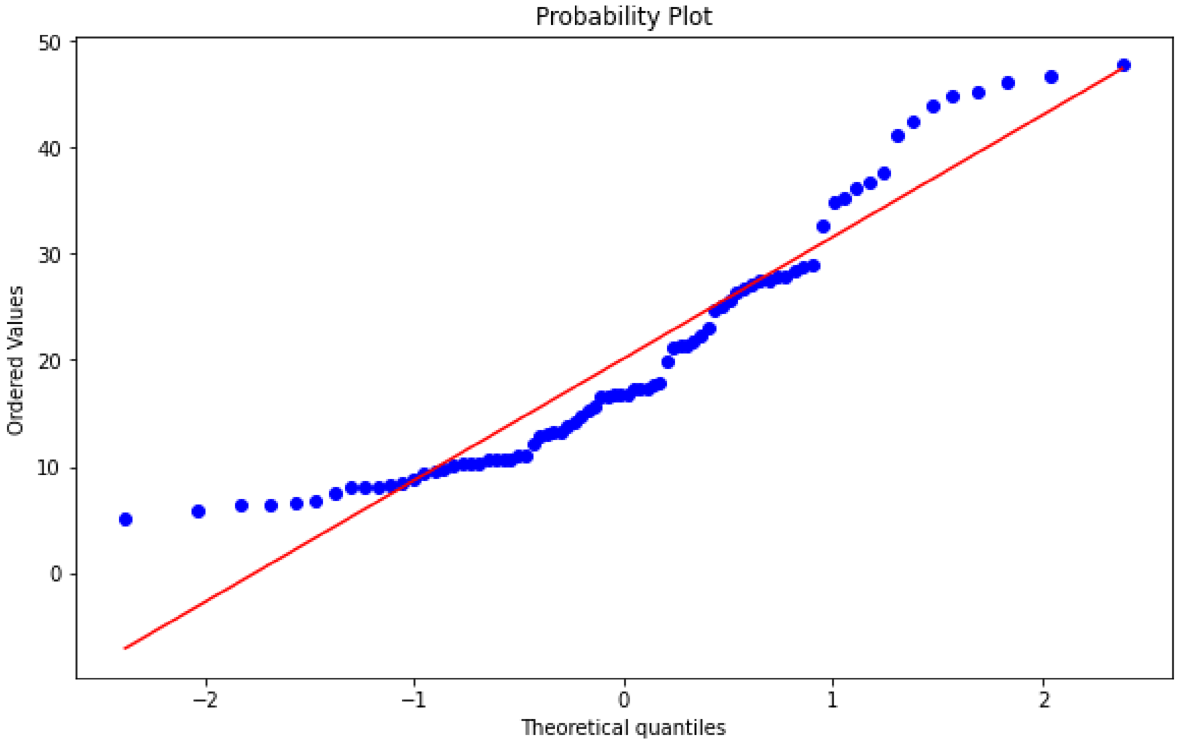
```
P/L:
```



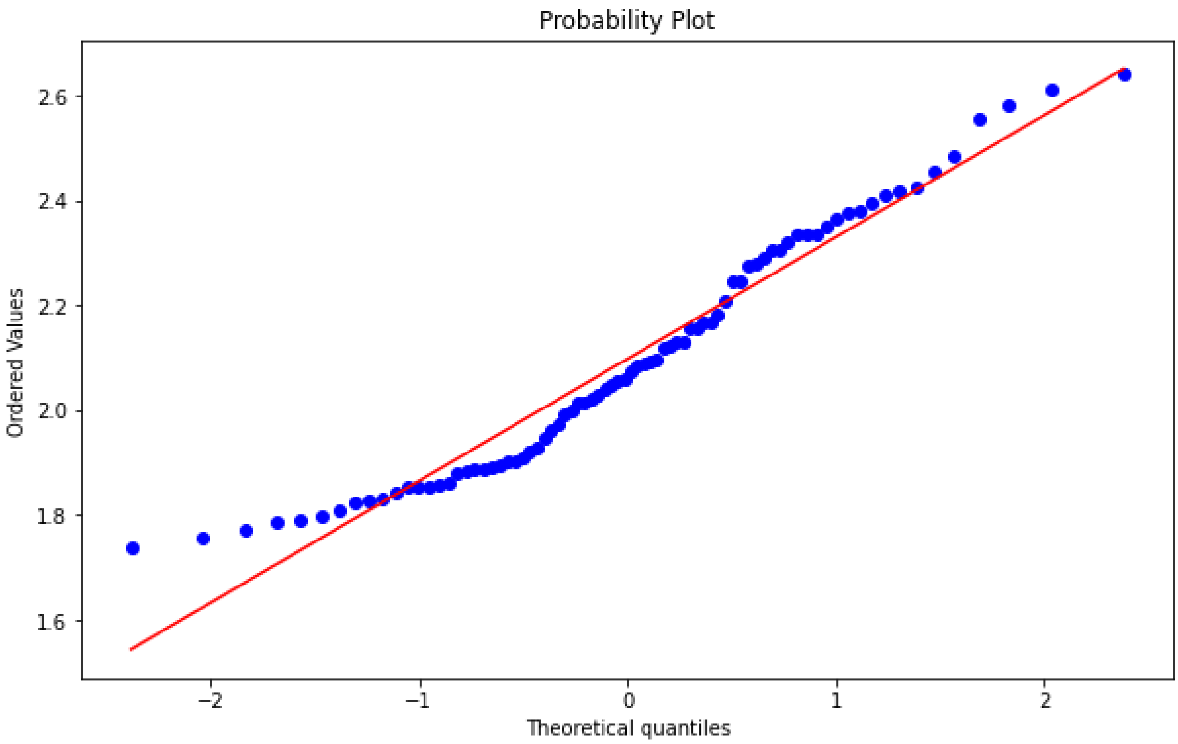
h_wall:



Nu_wall:



DeltaT:



In [27]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
```

In [ ]:
```
```

In [ ]:
```
```

In [270]:
```
y = df['DeltaT']
X = df.drop(labels = ['DeltaT' , 'h_wall','P/L' ,'Nu_wall'] , axis = 1)
```

In [271]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [272]:
```
num_bins = 5
bin_edges = np.linspace(np.min(X), np.max(X), num_bins + 1)
bin_indices = np.digitize(X, bin_edges)
```

In [273]:
```
y
```

Out[273]:
```
0     10.816222
1      8.670130
2      7.806540
3      7.338745
4      7.046579
        ...
75    15.325257
76    12.834496
77    11.654788
78    10.952318
79    10.485509
Name: DeltaT, Length: 80, dtype: float64
```

Scaling and Modelling

In [274]:
```
#test_size = int(0.2*(len(y)))
#train_size = int(0.8*(len(y)))

#X_train = X[:-test_size]
#X_test = X[train_size:]
#y_train = y[:-test_size]
#y_test = y[train_size:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2 ,  random_
X_test_new , y_test_new , X_valid , y_valid =  train_test_split(X_test, y_test, test
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [37]:
```
import os

# Tensorflow and Keras are two packages for creating neural network models.
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, BatchNormalization, Dropout
from tensorflow.keras import optimizers


from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
```

In [38]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score
```

In [21]:
```
X_te = X_test
```

In [22]:
```
def build_model(alpha , neurons):
    model=keras.Sequential([layers.Dense(5, activation='relu'),layers.Dense(neurons,
    optimizer=tf.keras.optimizers.RMSprop(alpha)
    model.compile(loss='mse', optimizer=optimizer, metrics=['mae','mse'])
    return model
```

In [ ]:

In [23]:
```
X = np.array(X_train)
Y= np.array(y_train)
```

In [24]:
```
alps = [0.1,0.01, 0.001 , 0.0001 , 0.00001]
neurons = np.array(list(range(1, 101)))
```

In [25]:
```
neurons
```

Out[25]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
        92,  93,  94,  95,  96,  97,  98,  99, 100])
```

In [26]:
```
R = [ ]
```

In [27]:
```
c = [{1,2,3}]
```

In [ ]:

for i in range(len(alps)): for j in range(len(neurons)) : model=build_model(alps[i] , neurons[j]) history = model.fit(X,Y, epochs=1000) X1 = np.array(X_test) y12 = np.array(y_test) tdp = model.predict(X1) loss,mae,mse=model.evaluate(X1,y12,verbose=0) r2 = r2_score(y12, tdp) R.append([r2,alps[i], neurons[j]]);

In [36]:
```
sorted_data = sorted(R, key=lambda x: x[1], reverse=True)
```

In [ ]:

In [37]:
```
len(final)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [37], in <cell line: 1>()
----> 1 len(final)

NameError: name 'final' is not defined
```

In [38]:
```
sorted_data = sorted(R, key=lambda x: x[0], reverse=True)
```

In [39]:
```
len(sorted_data)
```

Out[39]:
```
76
```

In [40]:
```
ans = pd.DataFrame(sorted_data)
```

In [41]:
```
ans.columns = ['R2', 'alpha', 'no_of_neurons_in_hidden_layer']
```

In [42]:
```
ans
```

Out[42]:

	R2	alpha	no_of_neurons_in_hidden_layer
0	0.969791	0.1	40
1	0.967030	0.1	44
2	0.963580	0.1	58
3	0.962834	0.1	73
4	0.960507	0.1	37
...	...	...	...
71	0.790820	0.1	18
72	0.773861	0.1	55
73	0.772997	0.1	22
74	0.729160	0.1	28
75	0.719375	0.1	15

76 rows × 3 columns

In [ ]:
```
ans.to_excel('ann_hypertuning.xlsx', index=False)
```

In [ ]:
```
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```

In [ ]:
```
(90+95+83+94+99)/5
```

In [ ]:

In [ ]:
```
tdp
```

In [ ]:

In [37]:

In [38]:
```
print("the mean squared error is: " , mse)
```
```
the mean squared error is:  0.008656417950987816
```

In [39]:
```
print("the loss   is: " , mae)
```
```
the loss   is:  0.07009359449148178
```

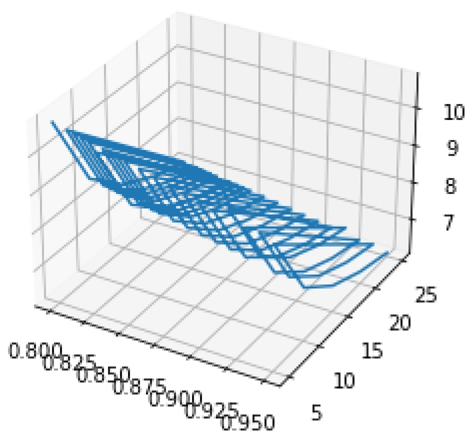
In [40]:
```
print('the root mean  squared error' , np.sqrt(mse))
```
```
the root mean  squared error 0.09303987290934901
```

In [41]:

In [42]:
```
print("the value of R2 score is" , r2)
```
```
the value of R2 score is 0.99412132509296
```

In [43]:
```
# importing mplot3d toolkits, numpy and matplotlib
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [44]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(df['Porosity'] , df['PPI']  , df['DeltaT'])
```
Out[44]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x2283f697f10>]
```



In [45]:
```
PPI_g = df['PPI']
Porosity = df['Porosity']
hwallg = df['h_wall']
Nussletg = df['Nu_wall']
```

cross validation

In [257]:
```
accuracy_scores = []
loss_scores = []
k = 15
```

In [258]:
```
kfold = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
```

In [259]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [76]:
```
testing = scaler.fit_transform(testing)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [260]:
```
kfold = KFold(n_splits=k, shuffle=True, random_state=42)
for train_index, test_index in kfold.split(X_scaled):
    # Split data into training and testing sets
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Create a new ANN model
    model = Sequential()
    # Add layers to the model
    model.add(Dense(16, input_dim=X.shape[1], activation='relu'))

    model.add(Dense(31, activation='relu'))
    model.add(Dense(1))
    optimizer=tf.keras.optimizers.RMSprop(0.001)
    # Compile the model
    model.compile(loss='mean_squared_error', optimizer=optimizer)

    # Train the model
    model.fit(X_train, y_train, epochs=1000, batch_size=32, verbose=0)

    # Evaluate the model on the test set
    scores = model.evaluate(X_test, y_test, verbose=0)
    ypred= model.predict(X_test)
    r = r2_score(y_test , ypred)
    accuracy_scores.append(r)
    # Store the evaluation metric
    loss_scores.append(scores)

# Print the average evaluation metric
print('Average Loss:', np.mean(loss_scores))
```

```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
Average Loss: 0.006948375383702418
```

In [283]:
```
accuracy_scores
```

Out[283]:
```
[0.9973522443091238,
 0.9931688029414855,
 0.9977383414711862,
 0.9972431539117199,
 0.9963878344605445,
 0.9941842567860442,
 0.9906377025416628,
 0.9865373632466382,
 0.9972846337173806,
 0.9926335117769705,
 0.9904871183084505,
 0.9950791991457503,
 0.9909127559419251,
 0.9542404417031518,
 0.9986046375492541]
```

In [311]:
```
res1 = np.array(accuracy_scores)
res2 = np.array(loss_scores)
res3 = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

In [306]:
```
res2.shape
```

Out[306]:
```
(15,)
```

In [316]:
```
res = np.concatenate((np.array(res3).reshape(-1,1) ,np.array(res1).reshape(-1,1) , r
```

In [319]:
```
res
```

Out[319]:
```
array([[1.00000000e+00, 9.97352244e-01, 5.49482508e-03],
       [2.00000000e+00, 9.93168803e-01, 2.05052793e-02],
       [3.00000000e+00, 9.97738341e-01, 2.82076909e-03],
       [4.00000000e+00, 9.97243154e-01, 6.83047296e-03],
       [5.00000000e+00, 9.96387834e-01, 3.52651975e-03],
       [6.00000000e+00, 9.94184257e-01, 1.36244046e-02],
       [7.00000000e+00, 9.90637703e-01, 1.23421084e-02],
       [8.00000000e+00, 9.86537363e-01, 1.22557429e-03],
       [9.00000000e+00, 9.97284634e-01, 2.62121647e-03],
       [1.00000000e+01, 9.92633512e-01, 9.82567761e-03],
       [1.10000000e+01, 9.90487118e-01, 4.15957067e-03],
       [1.20000000e+01, 9.95079199e-01, 6.95184106e-03],
       [1.30000000e+01, 9.90912756e-01, 3.76818608e-03],
       [1.40000000e+01, 9.54240442e-01, 7.71960057e-03],
       [1.50000000e+01, 9.98604638e-01, 2.80958484e-03]])
```

In [320]:
```
ans2 = pd.DataFrame(res)
```

In [321]:
```
ans2.columns = ['k' , 'R2' , 'loss_scores']
```

In [322]:
```
ans2
```

Out[322]:

	k	R2	loss_scores
0	1.0	0.997352	0.005495
1	2.0	0.993169	0.020505
2	3.0	0.997738	0.002821
3	4.0	0.997243	0.006830
4	5.0	0.996388	0.003527
5	6.0	0.994184	0.013624
6	7.0	0.990638	0.012342
7	8.0	0.986537	0.001226
8	9.0	0.997285	0.002621
9	10.0	0.992634	0.009826
10	11.0	0.990487	0.004160
11	12.0	0.995079	0.006952
12	13.0	0.990913	0.003768
13	14.0	0.954240	0.007720
14	15.0	0.998605	0.002810

In [323]:

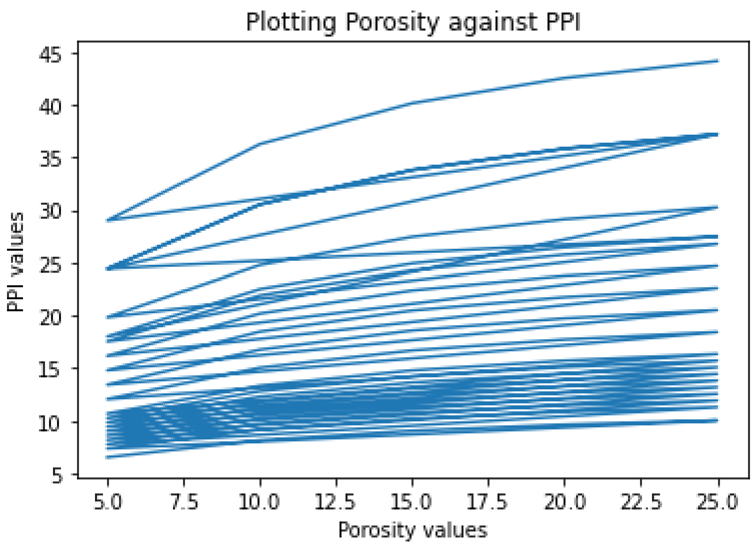
```
ans2.to_excel('cross_validation.xlsx', index=False)
```

In [46]:

```
import matplotlib.pyplot as plt
```

In [47]:

```
plt.plot(PPI_g,Nussletg)
plt.xlabel('Porosity values')
plt.ylabel('PPI values')
plt.title('Plotting Porosity against PPI')
plt.show()
```



In [ ]:

other way for ann model

In [48]:
```
X_train
```

Out[48]:
```
array([[-0.58843894,  1.53398003],
       [ 0.82381452, -0.01474981],
       [ 0.11768779,  1.09148579],
       [-1.29456567, -0.89973829],
       [ 0.82381452,  1.31273291],
       [-0.58843894, -0.34662049],
       [-1.29456567, -0.23599693],
       [-0.58843894,  1.75522715],
       [-0.58843894, -0.67849117],
       [ 0.82381452,  1.97647428],
       [ 0.82381452, -0.34662049],
       [-1.29456567,  0.20649731],
       [ 1.52994125, -0.78911473],
       [ 1.52994125, -1.34223253],
       [ 0.82381452,  0.20649731],
       [ 1.52994125, -0.23599693],
       [-0.58843894, -0.78911473],
       [ 0.11768779,  1.53398003],
       [ 0.82381452, -1.34223253],
       [ 1.52994125,  0.87023867],
       [-1.29456567, -0.56786761],
       [-0.58843894,  0.64899155],
       [-1.29456567,  0.87023867],
       [ 0.11768779, -0.34662049],
       [ 0.11768779, -0.23599693],
       [ 1.52994125,  1.53398003],
       [ 0.82381452, -1.12098541],
       [-1.29456567,  0.42774443],
       [-1.29456567, -1.34223253],
       [-1.29456567,  1.75522715],
       [ 0.11768779,  0.20649731],
       [ 0.11768779, -0.78911473],
       [-0.58843894,  0.42774443],
       [ 0.82381452,  0.42774443],
       [ 0.11768779, -1.12098541],
       [-0.58843894,  1.97647428],
       [ 0.82381452, -1.01036185],
       [ 0.82381452,  0.64899155],
       [-1.29456567,  1.53398003],
       [ 1.52994125, -1.01036185],
       [ 1.52994125, -0.67849117],
       [ 0.82381452, -0.67849117],
       [-0.58843894, -1.01036185],
       [ 0.82381452, -0.89973829],
       [-1.29456567, -0.78911473],
       [-1.29456567, -0.01474981],
       [-1.29456567, -0.67849117],
       [-0.58843894, -1.12098541],
       [-0.58843894,  0.87023867],
       [-0.58843894,  1.09148579],
       [-0.58843894, -1.34223253],
       [-0.58843894, -0.89973829],
       [ 1.52994125,  0.42774443],
       [ 1.52994125,  1.09148579],
       [-1.29456567, -1.12098541],
       [-1.29456567,  1.09148579],
       [ 1.52994125, -1.12098541],
       [ 0.11768779,  0.87023867],
       [ 0.11768779, -1.01036185],
       [ 0.11768779, -0.45724405]])
```

In [108]:
```
X_train = X_train.astype('float32')
y_train = y_train.astype('float32')
X_test =  X_test.astype('float32')
y_test = y_test.astype('float32')
```

In [109]:
```
X_train.shape
```

Out[109]:
```
(64, 3)
```

In [110]:
```
from  keras.models import Sequential
from  keras.layers import Dense


model = Sequential()

### deefining thr input layer
model.add(Dense(units=256 , input_dim =3 , kernel_initializer='normal' , activation=
## Defining the seond layer of the model
model.add(Dense(128 , kernel_initializer='normal' , activation='sigmoid'))
#model.add(Dense(256 , kernel_initializer='normal' , activation='sigmoid'))
model.add(Dense(256 , kernel_initializer='normal' , activation='sigmoid'))
model.add(Dense(1 , kernel_initializer='normal') )

model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(learning
```

In [135]:
```
testing = scaler.fit_transform(testing)
```

model.fit(np.array(X_train) , np.array(y_train) , batch_size =64 , epochs=5000 , verbose =2)

In [ ]:
```
```

In [114]:
```
pred = model.predict(X_test)
```

```
1/1 [==============================] - 0s 35ms/step
```

In [115]:
```
rt = r2_score(y_test , pred)
```

In [ ]:
```
```

In [116]:
```
rt
```

Out[116]:
```
0.954908889251611
```

In [123]:
```
np.exp(y_test)
```

Out[123]:
```
30     121.225365
0      457.891296
22     955.391418
31     351.126465
18    1648.028198
28    1313.986572
10     210.314545
70      40.833828
4     6738.153809
12    1183.942627
49    1292.609985
33    1114.630615
67     268.977661
35     103.488205
68     438.513367
45      95.589211
Name: P/L, dtype: float32
```

In [574]:
```
pred = np.exp(pred)
```

pred

In [577]:
```
max(pred)
```

Out[577]:
```
array([14.124857], dtype=float32)
```

In [583]:
```
import xgboost as xgb
```

In [584]:
```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

In [585]:
```
params = {
    'objective': 'reg:squarederror',  # Objective for regression
    'eval_metric': 'rmse'  # Evaluation metric for regression
}
```

In [586]:
```
num_rounds = 100  # Number of boosting rounds
model = xgb.train(params, dtrain, num_rounds)
```

In [587]:
```
y_pred = model.predict(dtest)
```

In [588]:
```
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.7684008607422053
```

In [589]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [594]:
```
# Create the Random Forest regressor
model = RandomForestRegressor(n_estimators=1000, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.7685827493464679
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [16]:
```
import numpy as np
from sklearn.metrics import r2_score
from keras.layers import Input, Dense
from keras.models import Model
```

In [45]:
```
X_train
```

Out[45]:
```
array([[5., 3., 2.],
       [2., 1., 4.],
       [2., 4., 4.],
       [5., 6., 5.],
       [1., 2., 2.],
       [4., 1., 2.],
       [2., 6., 1.],
       [5., 4., 4.],
       [1., 2., 5.],
       [5., 4., 1.],
       [5., 3., 5.],
       [4., 4., 5.],
       [5., 4., 2.],
       [1., 1., 2.],
       [5., 2., 2.],
       [4., 2., 5.],
       [1., 3., 4.],
       [2., 4., 5.],
       [2., 2., 2.],
       [1., 4., 5.],
       [5., 1., 5.],
       [4., 3., 2.],
       [2., 2., 1.],
       [4., 2., 2.],
       [2., 4., 2.],
       [4., 6., 1.],
       [2., 4., 1.],
       [1., 3., 5.],
       [2., 3., 5.],
       [5., 2., 4.],
       [4., 3., 4.],
       [1., 1., 5.],
       [2., 6., 5.],
       [1., 4., 1.],
       [1., 6., 2.],
       [4., 3., 5.],
       [4., 6., 2.],
       [5., 2., 1.],
       [2., 3., 4.],
       [4., 2., 1.],
       [5., 3., 1.],
       [4., 4., 1.],
       [1., 6., 4.],
       [4., 1., 1.],
       [1., 1., 1.],
       [1., 2., 4.],
       [5., 6., 4.],
       [2., 2., 4.]], dtype=float32)
```

In [86]:
```
input_dim = 3  # Number of input features (X)
output_dim = 1
```

In [87]:
```
input_layer = Input(shape=(input_dim,))
hidden_1 = Dense(16, activation='relu')(input_layer)
hidden_2 = Dense(8, activation='relu')(hidden_1)
encoded = Dense(32, activation='relu')(hidden_2)  # Encoding layer
hidden_3 = Dense(8, activation='relu')(encoded)
hidden_4 = Dense(16, activation='relu')(hidden_3)
decoded = Dense(output_dim, activation='relu')(hidden_4)  # Decoding layer
```

In [88]:
```
autoencoder = Model(input_layer, decoded)
```

In [89]:
```
autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='me
```

autoencoder.fit(X_train, y_train, epochs=1000, batch_size=32, shuffle=True)

In [108]:
```
reconstructed_output = autoencoder.predict(testing)
```

```
19/19 [==============================] - 0s 501us/step
```

np.exp(reconstructed_output)

In [105]:
```
y_test
```

Out[105]:
```
30    4.797652
0     6.126632
22    6.862121
31    5.861147
18    7.407335
28    7.180821
10    5.348604
70    3.709511
4     8.815541
12    7.076605
49    7.164419
33    7.016278
67    5.594628
35    4.639458
68    6.083390
45    4.560060
73    5.919050
61    5.101876
55    4.238428
40    4.730799
9     8.119660
64    6.641474
5     5.512925
47    6.286549
34    7.405366
62    5.765123
42    6.457226
54    7.000236
16    6.252354
39    7.246518
56    5.298303
79    6.149010
Name: P/L, dtype: float64
```

In [106]:
```
r2 = r2_score(y_test, reconstructed_output)
```

In [107]:
```
r2
```

Out[107]:
```
0.9550677883895686
```

In [41]:
```
testing
```

Out[41]:

	filling	PPI	Porosity
0	1.00	5	0.80
1	1.00	10	0.80
2	1.00	15	0.80
3	1.00	20	0.80
4	1.00	25	0.80
...	...	...	...
595	0.25	5	0.95
596	0.25	10	0.95
597	0.25	15	0.95
598	0.25	20	0.95
599	0.25	25	0.95

600 rows × 3 columns

In [209]:
```
pred = autoencoder.predict(testing)
```

```
19/19 [==============================] - 0s 1ms/step
```

pred

In [126]:
```
y_test
```

Out[126]:
```
63    185.393326
27    281.629730
31    124.676781
69    213.576782
46    101.888268
47    188.094193
53    241.746170
74    174.059204
39    433.814514
73    121.184875
34    520.685059
62    117.544907
36    104.250183
40     47.588596
58    201.607880
10     81.950943
38    301.900543
2     608.814758
35     39.544945
33    362.100128
45     38.505684
15     68.825745
66     51.225105
56     69.807091
19    752.051392
61     63.612995
77     64.426888
67     94.352486
26    152.348450
78    101.108910
65     19.482712
44    533.141235
Name: P/L, dtype: float32
```

In [125]:
```
pred
```

Out[125]:
```
array([[184.57568 ],
       [288.36835 ],
       [133.56613 ],
       [204.1521  ],
       [101.18329 ],
       [188.43918 ],
       [234.4572  ],
       [167.30417 ],
       [469.57068 ],
       [120.58119 ],
       [565.0937  ],
       [128.53323 ],
       [112.813324],
       [ 44.83865 ],
       [193.55737 ],
       [ 86.95609 ],
       [313.33514 ],
       [598.62286 ],
       [ 41.768444],
       [373.90936 ],
       [ 35.672924],
       [ 80.25492 ],
       [ 51.60412 ],
       [ 68.42386 ],
       [726.39886 ],
       [ 67.45118 ],
       [ 65.44147 ],
       [ 97.1205  ],
       [155.68872 ],
       [101.631775],
       [ 18.67115 ],
       [501.79944 ]], dtype=float32)
```

In [67]:
```
r2
```

Out[67]:
```
0.99637752124919
```

In [578]:
```
pred =  pd.DataFrame(pred)
```

In [581]:
```
pred.to_excel("temp_1_inline-a.xlsx", index=False, header=False)
```

In [51]:
```

```

Out[51]:
```
0.99637752124919
```

In [120]:
```
X13 = np.array(X_test)
y13 = np.array(y_test)
```

In [121]:
```
y13.shape
```

Out[121]:
```
(40,)
```

In [126]:
```
tdp1 = model.predict(X13)
```

```
2/2 [==============================] - 0s 1ms/step
```

In [127]:
```
tdp1.shape
```

Out[127]:
```
(40, 1)
```

In [128]:
```
r3 = r2_score(y13, tdp1)
```

In [129]:
```
r3
```

Out[129]:
```
0.9958438839851078
```

In [57]:
```
#def  hypertuning(X_tr , y_tr , X_te , y_te):
#    batch =[5,10,15,20 , 25 , 30 , 35]
#    ep = [5,10,50,100 ,200 , 500 , 1000]
#    SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', 'Accuracy'
#    trial = 0
#    for  b in  batch :
#        for e in  ep :
#            trial+=1
#            model = Sequential()
#            model.add(Dense(units=5 , input_dim=X_tr.shape[1] , kernel_initializer=
#            model.add(Dense(units=5 , kernel_initializer='normal' , activation='rel
#            model.add(Dense(1, kernel_initializer='normal'))
#            model.compile(loss='mean_squared_error' , optimizer=tf.keras.optimizers
#            model.fit(X_tr , y_tr , batch_size= b , epochs = e , verbose = 0)
#            y_test = np.array(y_te)
#            X_test = np.array(X_te)
#            MAPE =np.mean(100 * (np.abs(y_te-model.predict(X_te))/y_te))
#            print(trial, 'Parameters:','batch_size:', b,'-', 'epochs:',e, 'Accuracy
#            SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[trial, s
#    return SearchResultsData
```

In [58]:
```
#results = hypertuning(X_train , y_train , X_test , y_test)
```

In [59]:
```
print(y_test.shape)
```

```
(40,)
```

In [60]:
```
%matplotlib inline

#results.plot(x='Parameters' , y='Accuracy' , figsize=(10,4) , kind='line')
```

In [61]:
```
#results
```

In [62]:
```
model.fit(X_train , y_train , batch_size = 25 , epochs = 1000 , verbose = 0)
```

Out[62]:
```
<keras.callbacks.History at 0x2283fe27a30>
```

In [63]:
```
Pred = model.predict(X_te)
```

```
2/2 [==============================] - 0s 999us/step
```

In [64]:
```
X_te
```

Out[64]:
```
array([[-1.29456567,  1.31273291],
       [ 1.52994125,  1.31273291],
       [ 0.82381452, -0.56786761],
       [-0.58843894,  1.31273291],
       [ 0.82381452,  1.75522715],
       [ 0.11768779, -0.89973829],
       [-0.58843894, -0.45724405],
       [ 0.11768779,  1.31273291],
       [ 1.52994125,  0.64899155],
       [-1.29456567,  0.64899155],
       [ 0.11768779,  1.75522715],
       [ 1.52994125, -0.45724405],
       [-0.58843894,  0.20649731],
       [ 0.11768779, -0.01474981],
       [-0.58843894, -0.01474981],
       [ 0.11768779, -0.56786761],
       [-0.58843894, -0.56786761],
       [ 1.52994125, -0.34662049],
       [ 0.82381452,  1.09148579],
       [-1.29456567, -1.01036185],
       [ 0.11768779, -1.34223253],
       [ 0.82381452,  0.87023867],
       [ 0.11768779,  1.97647428],
       [ 0.11768779,  0.42774443],
       [ 1.52994125, -0.89973829],
       [-1.29456567, -0.45724405],
       [ 1.52994125,  1.75522715],
       [ 0.11768779, -0.67849117],
       [-0.58843894, -0.23599693],
       [ 0.82381452, -0.45724405],
       [ 0.11768779,  0.64899155],
       [ 1.52994125,  1.97647428],
       [ 1.52994125, -0.01474981],
       [-1.29456567,  1.97647428],
       [ 0.82381452,  1.53398003],
       [-1.29456567, -0.34662049],
       [ 0.82381452, -0.23599693],
       [ 1.52994125,  0.20649731],
       [ 0.82381452, -0.78911473],
       [ 1.52994125, -0.56786761]])
```

In [65]:
```
Pred
```

Out[65]:
```
array([[ 9.918861 ],
       [ 6.3921175],
       [ 7.1707315],
       [ 7.675173 ],
       [ 6.504658 ],
       [ 7.6233974],
       [ 8.409271 ],
       [ 7.012352 ],
       [ 6.4424596],
       [10.082568 ],
       [ 6.88628  ],
       [ 6.76416  ],
       [ 8.143364 ],
       [ 7.383209 ],
       [ 8.23331  ],
       [ 7.5340447],
       [ 8.452425 ],
       [ 6.732232 ],
       [ 6.6977305],
       [10.440893 ],
       [ 7.741125 ],
       [ 6.7616534],
       [ 6.8228397],
       [ 7.260912 ],
       [ 6.8912535],
       [10.329222 ],
       [ 6.3921175],
       [ 7.563927 ],
       [ 8.321954 ],
       [ 7.1397033],
       [ 7.19925  ],
       [ 6.3921175],
       [ 6.6361055],
       [ 9.742823 ],
       [ 6.569219 ],
       [10.305977 ],
       [ 7.0774117],
       [ 6.5717516],
       [ 7.232545 ],
       [ 6.7960277]], dtype=float32)
```

Comparision with other models

1 Linear Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn import preprocessing
from sklearn import utils
```

In [282]:
```
y = df['P/L']
X = df.drop(labels = ['DeltaT' , 'h_wall','P/L' ,'Nu_wall'] , axis = 1)
```

In [283]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
```

In [284]:
```
scaler=StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [285]:
```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test)
```

Out[285]:
```
0.7931896435262877
```

In [329]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), X, y, cv=cv)
```

Out[329]:
```
array([ 0.66198538,  0.52709167, -0.22703334,  0.5273235 ,  0.51177606])
```

In [330]:
```
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor

def find_best_model_using_gridsearchcv(X,y):
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_sc
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

find_best_model_using_gridsearchcv(X,y)
```

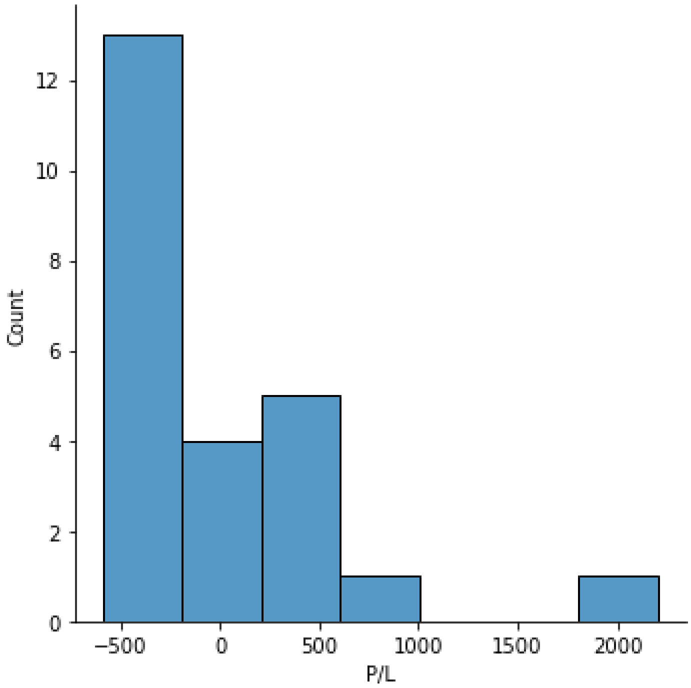
Out[330]:

	model	best_score	best_params
0	linear_regression	0.400229	{'normalize': True}
1	lasso	0.423747	{'alpha': 2, 'selection': 'random'}
2	decision_tree	0.818735	{'criterion': 'mse', 'splitter': 'random'}

In [331]:
```
pred= lr_clf.predict(X_test)
```

In [332]:
```
import seaborn as sns
sns.displot(y_test-pred)
```

Out[332]: <seaborn.axisgrid.FacetGrid at 0x24019fb2790>



In [ ]:

Polynomial Regression

In [275]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import PolynomialFeatures
```

In [286]:
```
max_degree = 1000

poly = PolynomialFeatures(degree=2 , include_bias=False)
x_poly = poly.fit_transform(X_train)
xt = poly.fit_transform(X_test)
```

In [287]:
```
model = LinearRegression()
model.fit(x_poly, y_train)
y_pred = model.predict(xt)
```

In [288]:
```
y_train
```

Out[288]:
```
55      26.632139
17     332.824593
60      24.037297
62     117.544907
6      264.242033
56      69.807093
73     121.184873
4     1385.303867
33     362.100133
53     241.746173
28     444.119720
11     216.294647
57     128.277433
23     554.672200
10      81.950940
31     124.676780
43     370.137607
61      63.612994
1      328.563047
32     229.942753
75      13.523349
14     902.617067
54     347.476693
19     752.051400
29     639.216187
49     426.501633
24     799.170000
35      39.544943
18     523.420707
0      123.196607
78     101.108913
15      68.825747
5       99.624040
59     289.558700
16     180.904140
51      83.432387
20      70.940700
74     174.059207
8      769.911000
13     627.755400
25      75.101100
37     191.924033
46     101.888267
39     433.814500
65      19.482712
58     201.607880
12     398.710240
70      16.064561
36     104.250180
21     189.466700
9     1108.060800
76      35.163144
67      94.352487
64     266.901147
47     188.094200
44     533.141260
Name: P/L, dtype: float64
```

In [289]:
```
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.9775846331724232
```

In [280]:
```
y_test
```

Out[280]:
```
30    10.486114
0     10.816222
22     8.027379
31     8.240966
18     6.382000
28     7.187765
10     9.879654
70    15.643892
4      7.046579
12     7.143033
49     8.617024
33     6.860035
67    12.264212
35     9.951711
68    11.544225
45    13.029163
Name: DeltaT, dtype: float64
```

In [281]:
```
y_pred
```

Out[281]:
```
array([10.4715031 , 10.62839894,  8.13498078,  8.63671801,  6.03064402,
        7.14368007,  9.75414641, 15.49104413,  7.23412784,  6.9245741 ,
        8.53637991,  6.7964836 , 12.27592773, 10.04428243, 11.40870044,
       12.76440768])
```

In [290]:
```
testing = pd.read_csv("features.csv")
print(testing)
testing = np.array(testing)
testing = scaler.fit_transform(testing)
testing  = poly.fit_transform(testing)
yp = model.predict(testing)
```

```
     filling  PPI  Porosity
0       1.00    5      0.80
1       1.00   10      0.80
2       1.00   15      0.80
3       1.00   20      0.80
4       1.00   25      0.80
..       ...  ...       ...
595     0.25    5      0.95
596     0.25   10      0.95
597     0.25   15      0.95
598     0.25   20      0.95
599     0.25   25      0.95

[600 rows x 3 columns]
```

yp

In [193]:
```
yp =  pd.DataFrame(yp)
```

In [194]:
```
yp.to_excel('temp_1_inline-a.xlsx', index=False)
```

np.exp(yp)

In [269]:
```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from statsmodels.api import OLS



# Fit polynomial regression models and calculate BIC
```

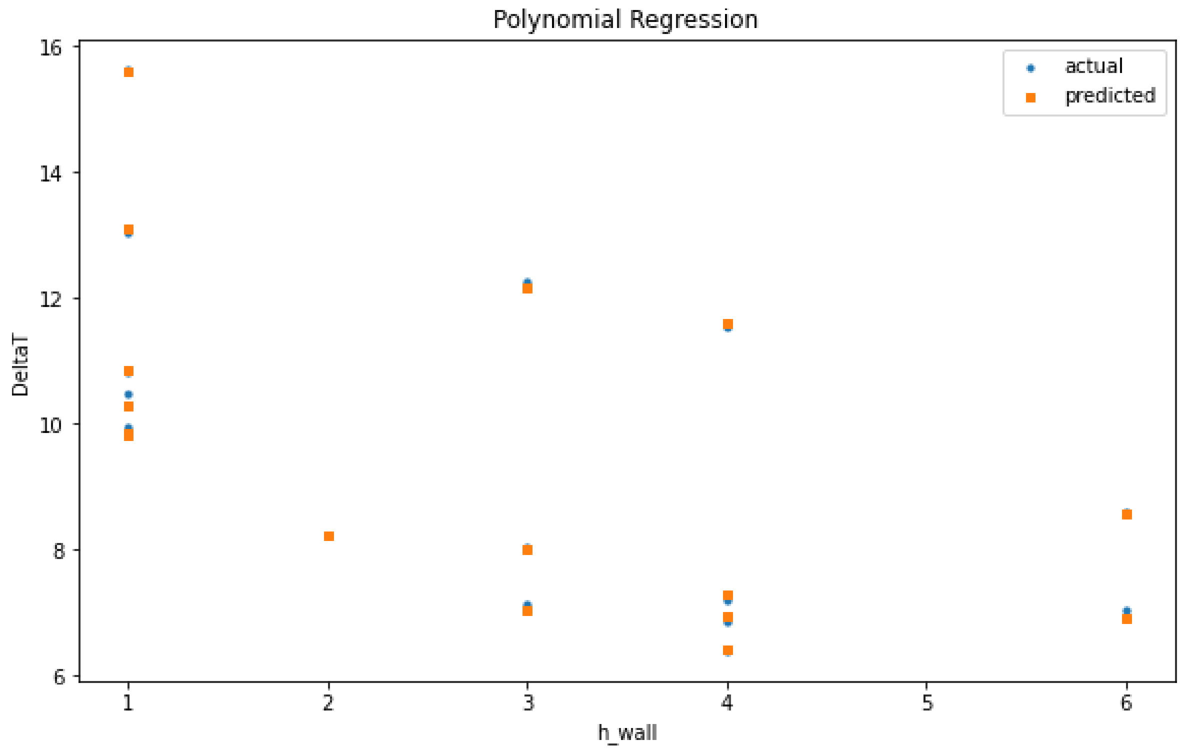
```
bic_values = []
for degree in range(1, max_degree + 1):
    X_poly_subset = x_poly[:, :degree]  # Select polynomial features up to the curre

    # Fit the model using scikit-learn
    model_sklearn = LinearRegression()
    model_sklearn.fit(X_poly_subset, y_train)

    # Calculate the residual sum of squares (RSS)
    rss = np.sum((model_sklearn.predict(X_poly_subset) - y_train) ** 2)

    # Calculate the number of parameters (including the intercept term)
    num_params = degree + 1  # degree + 1 for the intercept term

    # Calculate the BIC
    bic = len(X) * np.log(rss / len(X)) + num_params * np.log(len(X))
    bic_values.append(bic)

# Find the degree with the lowest BIC
best_degree = np.argmin(bic_values) + 1
best_bic = bic_values[best_degree - 1]

print("Best degree:", best_degree)
print("BIC:", best_bic)
```

```
Best degree: 198
BIC: -3413.949035014941
```

In [65]:
```
np.array(y_test).reshape(-1,1).shape
```

Out[65]:
```
(16, 1)
```

In [66]:
```
# Create a new plot
plt.figure(figsize=(10
                    , 6))

# Plot the data points
plt.scatter(np.array(X_test)[:,1], np.array(y_test).reshape(-1,1), s=10 , marker='o'
plt.scatter(np.array(X_test)[:,1], np.array(y_pred).reshape(-1,1), s=10 ,marker='s'
# Plot the regression line
#plt.plot((sorted_X_test), (sorted_y_pred), color='r')
plt.legend()
# Set the plot title and axis labels
plt.title('Polynomial Regression')
plt.xlabel('h_wall')
plt.ylabel('DeltaT')

# Show the plot
plt.show()
```



In [85]:
```
model = LinearRegression()
model.fit(X_test, y_test)
y_pred = model.predict(X_test)
```

In [86]:
```
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.8612348142970546
```

In [87]:
```
X_scale =[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28
```

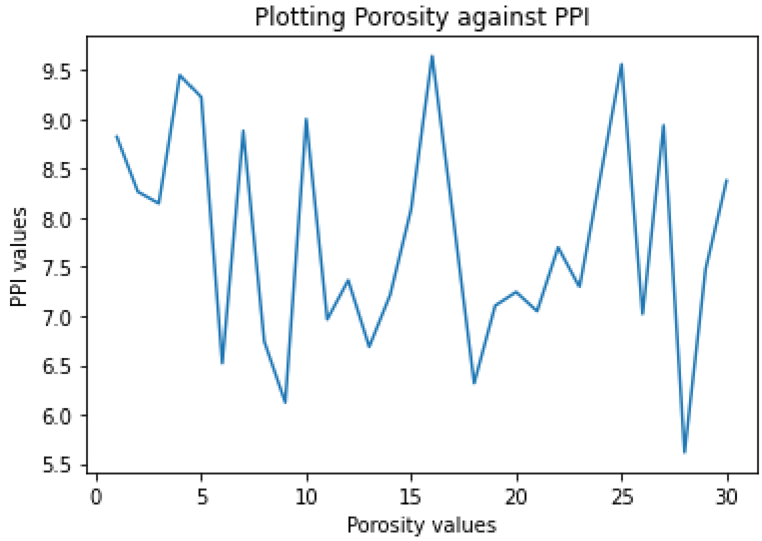
In [88]:
```
y_test.shape
```

Out[88]:
```
(30,)
```

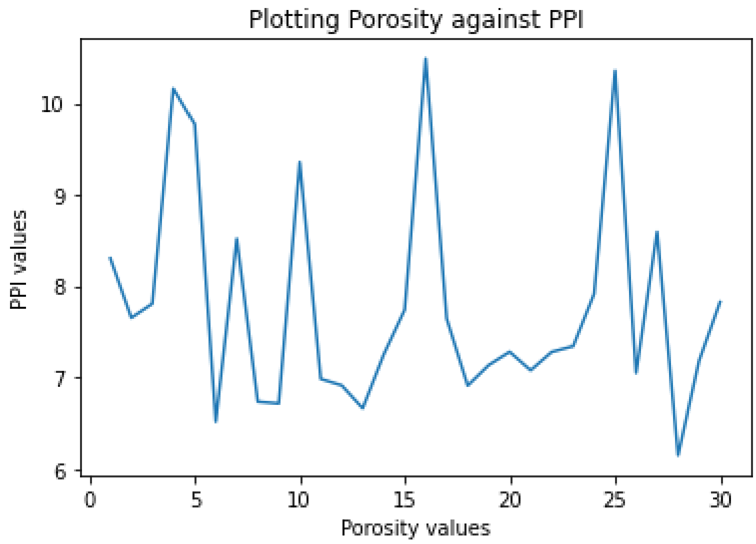
In [89]:
```
y_pred
```

Out[89]:
```
array([8.82615226, 8.26709085, 8.14707022, 9.44932816, 9.22570359,
       6.51758378, 8.8820584 , 6.74120835, 6.11803245, 9.00207903,
       6.96483291, 7.36438424, 6.6853022 , 7.21641055, 8.09116408,
       9.64499965, 8.00730487, 6.31370394, 7.10459827, 7.24436362,
       7.04869212, 7.69982109, 7.30026976, 8.43480927, 9.56114044,
       7.02073905, 8.93796454, 5.61487718, 7.47619653, 8.37890313])
```

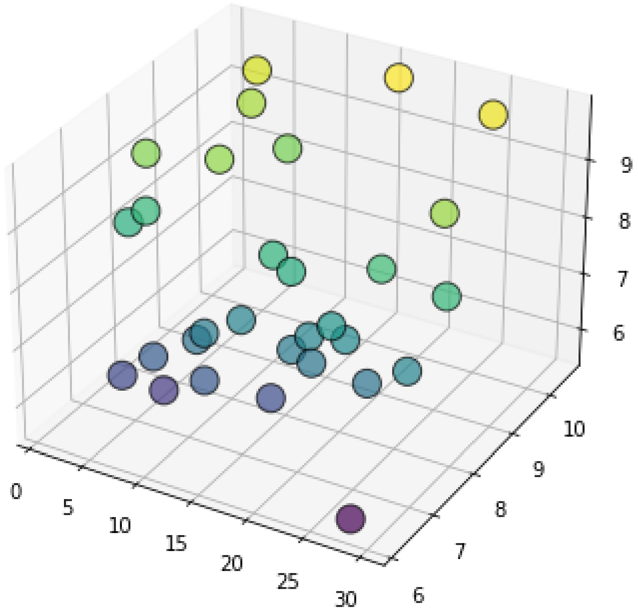
In [90]:
```
plt.plot(X_scale , y_pred)
plt.xlabel('Porosity values')
plt.ylabel('PPI values')
plt.title('Plotting Porosity against PPI')
plt.show()
```



In [97]:
```
plt.plot(X_scale , y_test)
plt.xlabel('Porosity values')
plt.ylabel('PPI values')
plt.title('Plotting Porosity against PPI')
plt.show()
```



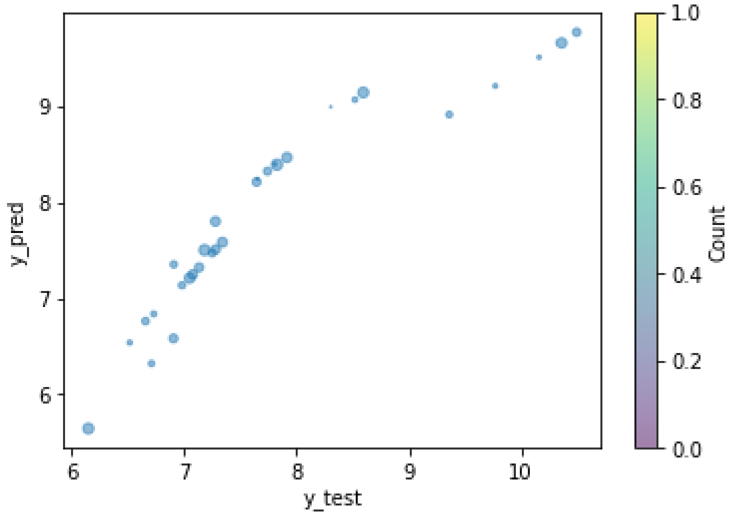
In [113]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_scale, y_test, y_pred,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=y_pred)
plt.show()
```



In [114]:
```
np.array(y_pred[-10]).reshape(-1,1).shape
```

Out[114]:
```
(1, 1)
```

In [119]:
```
import matplotlib.pyplot as plt
import numpy as np

# Generate some random data


# Create a scatter plot
plt.scatter(np.array(y_test).reshape(-1,1), np.array(y_pred).reshape(-1,1), s=np.arr

# Add labels and a colorbar
plt.xlabel('y_test')
plt.ylabel('y_pred')
plt.colorbar(label='Count')

# Show the plot
plt.show()
```

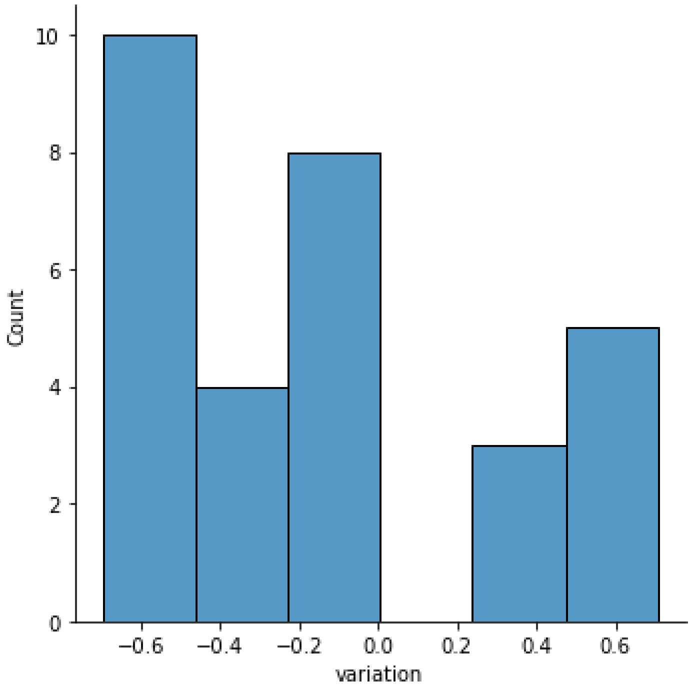


In [120]:
```
import seaborn as sns
sns.displot(y_test-y_pred)
```

```
plt.xlabel('variation')
plt.ylabel('Count')
```

Out[120]:

```
Text(10.049999999999997, 0.5, 'Count')
```



Ridge Regression

In [50]:

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
```

In [57]:

```
rd = Ridge(alpha=0.01)
```

In [58]:

```
rd.fit(X_train, y_train)
```

Out[58]:

```
Ridge(alpha=0.01)
```

In [59]:

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

In [60]:

```
mse = mean_squared_error(y_test, y_pred)
```

In [61]:

```
print('accuracy is' ,100-mse)
```

```
accuracy is -8233.734375
```

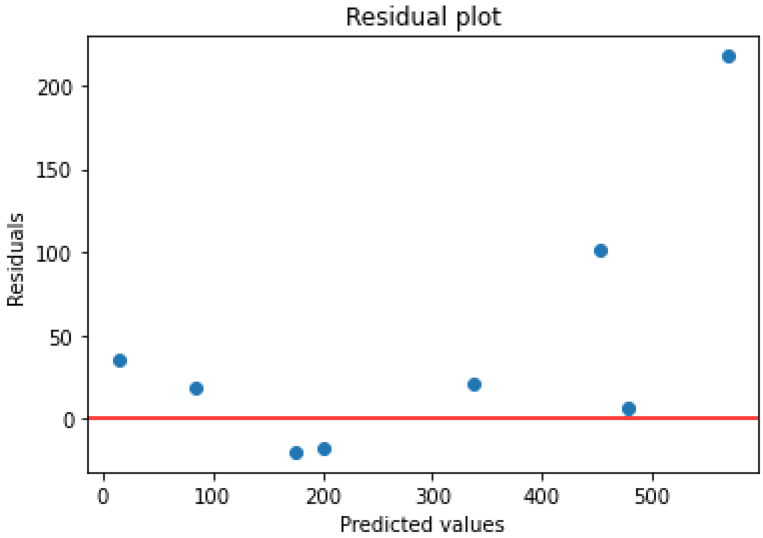
In [62]:

```
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.7489097486841676
```

In [257]:

```
residuals = y_test - y_pred

# plot residuals against predicted values
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.title('Residual plot')
plt.show()
```



curve fitting practise

In [129]:

```
import numpy as np
from scipy.optimize import curve_fit

# Define the function to fit
def func(x, y , z ,a, b, c, d):
    return a * (b**x) * (c**y) * (d**z)

# Define the data points
xdata = np.array([1, 2, 3, 4, 5,6,7,8])
ydata = np.array([1.2, 2.4, 4.8, 9.6, 19.2,38.4,76.8,153.6])

# Perform the curve fit
popt, pcov = curve_fit(func, xdata, ydata)

# Print the optimized parameters
print('a =', popt[0])
print('b =', popt[1])
print('c =', popt[2])
print('d =', popt[3])
```

```
a = 1.0043568694166436
b = 1.034511260382111
c = 0.65206477482931
d = 2.0
```

In [ ]:

In [130]:

```
import numpy as np
from scipy.optimize import curve_fit

def ln_func(x, a, x1, x2, x3):
    Re, Rib, wc, eps = x[:,0], x[:,1], x[:,2], x[:,3]
    return np.log(a * (Re ** x1) * ((Rib * wc) ** x2) * (eps ** x3))



data = np.array([[1000, 100, 0.5, 0.75, 3.2],
                 [2000, 100, 0.5, 0.75, 6.4],
                 [4000, 100, 0.5, 0.75, 12.8],
                 [8000, 100, 0.5, 0.75, 25.6],
                 [16000, 100, 0.5, 0.75, 51.2],
                 [32000, 100, 0.5, 0.75, 102.4],
```

```
                 [64000, 100, 0.5, 0.75, 204.8],
                 [128000, 100, 0.5, 0.75, 409.6]])

# Separate the inputs and outputs
xdata, ydata = data[:,:-1], data[:,-1]
print(ydata)

# Perform the curve fit
popt, pcov = curve_fit(ln_func, xdata, np.log(ydata))

# Extract the optimized values of the parameters
a_opt, x1_opt, x2_opt, x3_opt = np.exp(popt)

# Print the optimized values
print('a =', a_opt)
print('x =', x1_opt)
print('y =', x2_opt)
print('z =', x3_opt)
```

```
[  3.2   6.4  12.8  25.6  51.2 102.4 204.8 409.6]
a = 85.89099624375832
x = 2.718281828459045
y = 2.9059781978217907
z = 1.6868459529373136e+17
```

In [131]:

```
x  = X_train
y = y_train
```

In [132]:

```
y = np.array(y).reshape(-1,1)
```

In [ ]:

In [ ]:

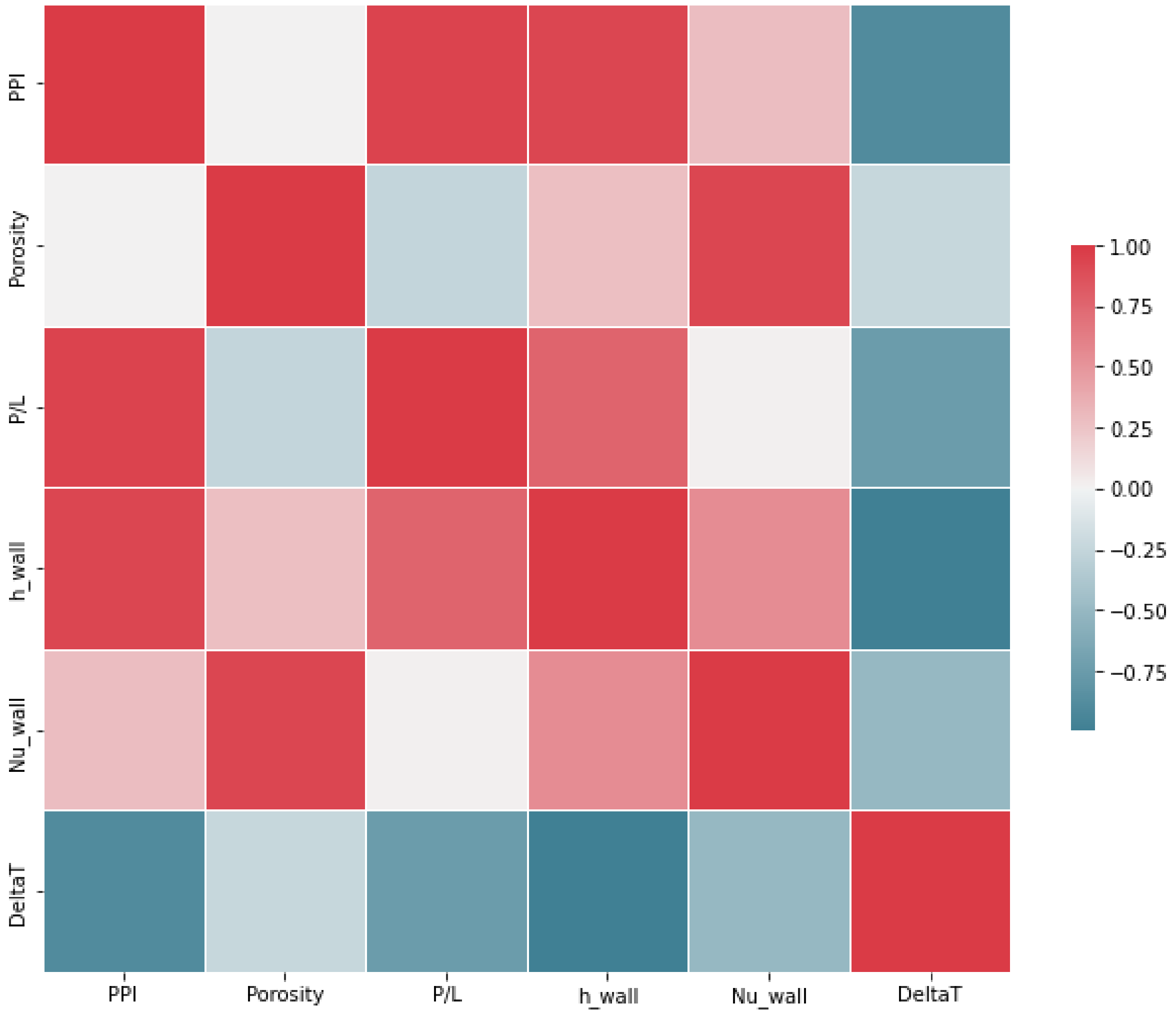
In [ ]:

In [ ]:

In [133]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression

# Load the Boston Housing dataset


# Convert to a pandas dataframe


# Compute the correlation matrix
corr_matrix = df.corr()

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, cmap=cmap, center=0,
```

```
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: