In [1]:

```
import pandas as pd
import os
os.getcwd()
```

Out[1]:

```
'C:\\Users\\Mahima Sharu'
```

In [2]:

```
os.listdir(os.getcwd())
```

Out[2]:

```
['.anaconda',
 '.android',
 '.AndroidStudio3.2',
 '.bash_history',
 '.conda',
 '.condarc',
 '.config',
 '.gitconfig',
 '.idlerc',
 '.ipynb_checkpoints',
 '.ipython',
 '.jupyter',
 '.matplotlib',
 '.node_repl_history',
 '.packettracer',
 '.PyCharmCE2019.1',
 '.VirtualBox',
 '.vscode',
 '3 - 1st and 2nd.ipynb',
 '3 - 3rd.ipynb',
 '3 - 4th.ipynb',
 '3D Objects',
 '4th - 1st.ipynb',
 'Amazon Reviews_ Unlocked Mobile Phones _ Kaggle.csv',
 'Amazon Reviews_ Unlocked Mobile Phones _ Kaggle_files',
 'AppData',
 'Application Data',
 'assignment 1.1.ipynb',
 'assignment 1.3.txt',
 'assignment1.4.ipynb',
 'Cisco Packet Tracer 7.2',
 'Contacts',
 'Cookies',
 'debug.log',
 'Desktop',
 'Documents',
 'Downloads',
 'electric motor temperature.ipynb',
 'Favorites',
 'file.txt',
 'Git-workshop',
 'Hotel booking demand _ Kaggle.csv',
 'Hotel booking demand _ Kaggle_files',
 'HP',
 'index.htmly',
 'input.txt',
 'IntelGraphicsProfiles',
 'iris.csv',
 'KCLT.csv',
 'Links',
 'Local Settings',
 'MicrosoftEdgeBackups',
 'Music',
 'My Documents',
 'NetHood',
 'NTUSER.DAT',
 'ntuser.dat.LOG1',
```

```
 'ntuser.dat.LOG1',
 'ntuser.dat.LOG2',
 'NTUSER.DAT{43757b42-e0ed-11e9-986f-e08e1ad91340}.TM.blf',
 'NTUSER.DAT{43757b42-e0ed-11e9-986f-e08e1ad91340}.TMContainer00000000000000000001.regtrans-ms',
 'NTUSER.DAT{43757b42-e0ed-11e9-986f-e08e1ad91340}.TMContainer00000000000000000002.regtrans-ms',
 'ntuser.ini',
 'OneDrive',
 'Oracle',
 'Pictures',
 'pmsm_temperature_data.csv',
 'PrintHood',
 'PycharmProjects',
 'Recent',
 'Saved Games',
 'ScStore',
 'Searches',
 'SendTo',
 'Sentiment Analysis on Movie Reviews _ Kaggle.csv',
 'Sentiment Analysis on Movie Reviews _ Kaggletest.csv',
 'Sentiment Analysis on Movie Reviews _ Kaggletest_files',
 'Sentiment Analysis on Movie Reviews _ Kaggletrain.csv',
 'Sentiment Analysis on Movie Reviews _ Kaggletrain_files',
 'Sentiment Analysis on Movie Reviews _ Kaggle_files',
 'Start Menu',
 'Templates',
 'Untitled.ipynb',
 'Untitled1.ipynb',
 'Untitled10.ipynb',
 'Untitled11.ipynb',
 'Untitled12.ipynb',
 'Untitled2.ipynb',
 'Untitled3.ipynb',
 'Untitled4.ipynb',
 'Untitled5.ipynb',
 'Untitled6.ipynb',
 'Untitled7.ipynb',
 'Untitled8.ipynb',
 'Untitled9.ipynb',
 'Videos',
 'VirtualBox VMs',
 'wekafiles',
 '_split.csv']
```

In [9]:

```
#Load File
df=pd.read_csv('C:\\Users\\Mahima Sharu\\pmsm_temperature_data.csv')
```

Out[9]:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stato
0	-0.752143	-1.118446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	
1	-0.771263	-1.117021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	
2	-0.782892	-1.116681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	
3	-0.780935	-1.116764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	
4	-0.774043	-1.116775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	
...	...	...	...	...	...	...	...	...	...	...	...	
998065	-0.047497	0.341638	0.331475	-1.246114	-1.222428	-0.255640	1.029142	-0.245722	0.429853	1.018568	0.836084	
998066	-0.048839	0.320022	0.331701	-1.250655	-1.222437	-0.255640	1.029148	-0.245736	0.429751	1.013417	0.834438	
998067	-0.042350	0.307415	0.330946	-1.246852	-1.222430	-0.255640	1.029191	-0.245701	0.429439	1.002906	0.833936	
998068	-0.039433	0.302082	0.330987	-1.249505	-1.222432	-0.255640	1.029147	-0.245727	0.429558	0.999157	0.830504	
998069	[illegible]	0.312666	0.330830	[illegible]	-1.222431	[illegible]	1.029141	[illegible]	0.429166	0.987163	0.828046	

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator
	0.043803			1.246591		0.255640		0.245722				

998070 rows × 13 columns

In [4]:

```
from sklearn.preprocessing import Imputer
from sklearn.model_selection import KFold          #Provides train/test indices to split data in
train/test sets
from sklearn import linear_model
from sklearn.metrics import make_scorer

from sklearn import svm           #Support Vector Machine are a set of supervised learning methods
# linear algebra
import numpy as np
# data processing, CSV file I/O (e.g. pd.read_csv)
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns        #Visualizing dataset structure that can be used to make visualizations
with multiple plots
from sklearn.linear_model import LinearRegression

from sklearn import neighbors
from math import sqrt
```

In [10]:

```
#************************************************Basic Linear
Regression*************************************************

import statsmodels.api as sm          #provides classes and functions for the estimation of many
different statistical models

#Defining dependent and independent variable
X = df['i_d']
X=sm.add_constant(X)

y = df['motor_speed']

lm=sm.OLS(y,X)              #Leastsquare Minimization using Ordinary Least Square value along the
x and y axes
model=lm.fit()               #Data Fitting

model.summary()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .
ptp is deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Out[10]:

OLS Regression Results

Dep. Variable:	motor_speed	**R-squared:**	0.523
Model:	OLS	**Adj. R-squared:**	0.523
Method:	Least Squares	**F-statistic:**	1.093e+06
Date:	Fri, 03 Apr 2020	**Prob (F-statistic):**	0.00
Time:	12:17:00	**Log-Likelihood:**	-1.0484e+06
No. Observations:	998070	**AIC:**	2.097e+06
Df Residuals:	998068	**BIC:**	2.097e+06
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0020	0.001	-2.826	0.005	-0.003	-0.001
i_d	-0.7245	0.001	-1045.267	0.000	-0.726	-0.723

Omnibus:	7561.278	**Durbin-Watson:**	0.003
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5931.452
Skew:	-0.109	**Prob(JB):**	0.00
Kurtosis:	2.691	**Cond. No.**	1.01

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [11]:

```
model.params
```

Out[11]:

```
const   -0.001957
i_d     -0.724531
dtype: float64
```

In [12]:

```
print("f_pvalue:", "%.4f" % model.f_pvalue)
```

```
f_pvalue: 0.0000
```

In [13]:

```
#mean square value
model.mse_model
```

Out[13]:

```
522878.40071765514
```

In [14]:

```
model.rsquared
```

Out[14]:

```
0.5226043734324983
```

In [15]:

```
model.rsquared_adj
```

Out[15]:

```
0.522603895112758
```

In [16]:

```
#Predicted values
model.fittedvalues[0:5]
```

Out[16]:

```
0   -0.747914
1   -0.747869
2   -0.747824
3   -0.750285
4   -0.749534
dtype: float64
```

In [17]:

```
#Real values
y[0:5]
```

Out[17]:

```
0   -1.222428
1   -1.222429
2   -1.222428
3   -1.222430
4   -1.222429
Name: motor_speed, dtype: float64
```

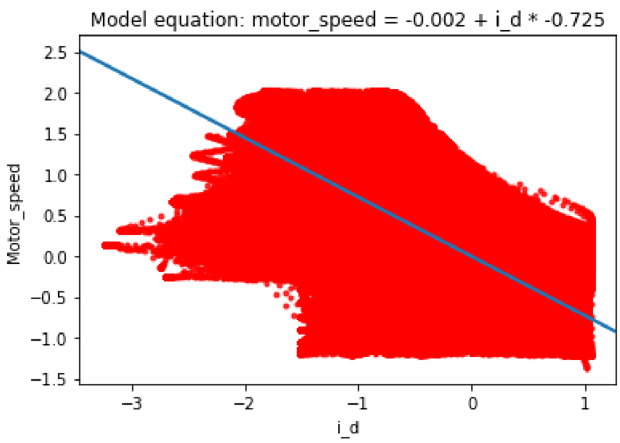
In [18]:

```
#Model equation
print("Motor speed = " +
      str("%.3f" % model.params[0]) + ' + i_d' + "*" +
      str("%.3f" % model.params[1]))
```

```
Motor speed = -0.002 + i_d*-0.725
```

In [19]:

```
#Regression Model Visualization
g=sns.regplot(df['i_d'] , df['motor_speed'],
              ci=None, scatter_kws={'color': 'r', 's':9})
g.set_title('Model equation: motor_speed = -0.002 + i_d * -0.725')
g.set_ylabel('Motor_speed')
g.set_xlabel('i_d');
```



In [20]:

```
from sklearn.metrics import r2_score,mean_squared_error
mse=mean_squared_error(y, model.fittedvalues)
rmse=np.sqrt(mse)
rmse
```

Out[20]:

```
0.6917872418443057
```

In [21]:

```
k_t=pd.DataFrame({'Real_values':y[0:50],
                  'Predicted_values' :model.fittedvalues[0:50]})
k_t['error']=k_t['Real_values']-k_t['Predicted_values']
k_t.head()
```

Out[21]:

	Real_values	Predicted_values	error

	Real_values	Predicted_values	error
0	-1.222428	-0.747914	-0.474514
1	-1.222429	-0.747869	-0.474561
2	-1.222428	-0.747824	-0.474604
3	-1.222430	-0.750285	-0.472145
4	-1.222429	-0.749534	-0.472895

In [22]:

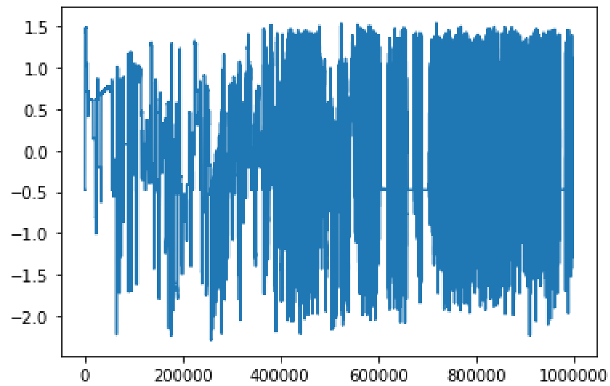
```
#Easiest way to learn residual
model.resid[0:10]
```

Out[22]:

```
0    -0.474514
1    -0.474561
2    -0.474604
3    -0.472145
4    -0.472895
5    -0.473457
6    -0.473848
7    -0.474130
8    -0.474315
9    -0.474471
dtype: float64
```

In [23]:

```
plt.plot(model.resid);
```



In [24]:

```
#***************************************Multiple Linear
Regression**************************************************
X=df.drop("motor_speed", axis=1)
y=df["motor_speed"]
```

In [25]:

```
from sklearn.model_selection import train_test_split,cross_val_score,cross_val_predict

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2, random_state=42)

training=df.copy()
```

In [26]:

```
lm=sm.OLS(y_train, X_train)

model=lm.fit()
#All coefficients are significant for the model by looking at the p-value. ( P>|t| )
model.summary()
```

Out[26]:

OLS Regression Results

Dep. Variable:	motor_speed	**R-squared (uncentered):**	0.928
Model:	OLS	**Adj. R-squared (uncentered):**	0.928
Method:	Least Squares	**F-statistic:**	8.582e+05
Date:	Fri, 03 Apr 2020	**Prob (F-statistic):**	0.00
Time:	12:35:22	**Log-Likelihood:**	-83308.
No. Observations:	798456	**AIC:**	1.666e+05
Df Residuals:	798444	**BIC:**	1.668e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
ambient	-0.0503	0.000	-131.429	0.000	-0.051	-0.050
coolant	0.4091	0.002	218.100	0.000	0.405	0.413
u_d	-0.1657	0.001	-254.611	0.000	-0.167	-0.164
u_q	0.5394	0.000	1469.606	0.000	0.539	0.540
torque	-0.3411	0.005	-70.287	0.000	-0.351	-0.332
i_d	-0.6580	0.001	-1268.145	0.000	-0.659	-0.657
i_q	0.1352	0.005	29.630	0.000	0.126	0.144
pm	0.1061	0.001	170.646	0.000	0.105	0.107
stator_yoke	-1.6278	0.006	-282.304	0.000	-1.639	-1.617
stator_tooth	2.3219	0.008	304.592	0.000	2.307	2.337
stator_winding	-1.1714	0.004	-310.271	0.000	-1.179	-1.164
profile_id	-1.117e-05	5.6e-06	-1.995	0.046	-2.22e-05	-1.96e-07

Omnibus:	43472.517	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	169112.669
Skew:	-0.111	**Prob(JB):**	0.00
Kurtosis:	5.244	**Cond. No.**	1.82e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.82e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [27]:

```
#Root Mean Squared Error for Train
rmse1=np.sqrt(mean_squared_error(y_train,model.predict(X_train)))
rmse1
```

Out[27]:

0.2685809987166384

In [28]:

```
#Root Mean Squared Error for Test
rmse2=np.sqrt(mean_squared_error(y_test,model.predict(X_test)))
rmse2
```

Out[28]:

0.26823759191828583

In [29]:

```
#Model Tuning for Multiple Linear Regression
model = LinearRegression().fit(X_train,y_train)
cross_val_score1=cross_val_score(model, X_train, y_train, cv=10, scoring='r2').mean() #verified
score value for train model
print('Verified R2 value for Training model: ' + str(cross_val_score1))

cross_val_score2=cross_val_score(model, X_test, y_test, cv=10, scoring='r2').mean() #verified score
value for test model
print('Verified R2 value for Testing Model: ' + str(cross_val_score2))
```

```
Verified R2 value for Training model: 0.9280570425519296
Verified R2 value for Testing Model: 0.9281973950061891
```

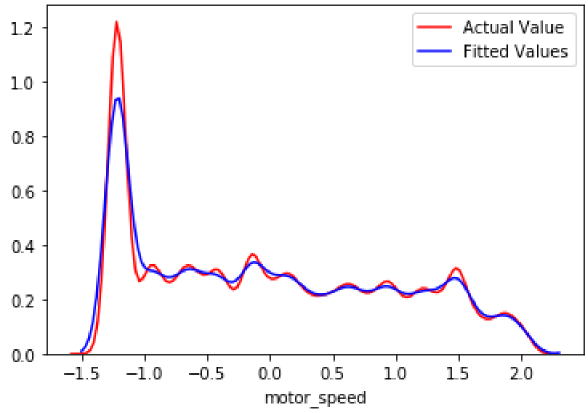
In [30]:

```
#For Root Mean square value
RMSE1=np.sqrt(-cross_val_score(model, X_train, y_train, cv=10,
                               scoring='neg_mean_squared_error')).mean() #verified RMSE score value
for train model
print('Verified RMSE value for Training model: ' + str(RMSE1))

RMSE2=np.sqrt(-cross_val_score(model, X_test, y_test, cv=10,
                               scoring='neg_mean_squared_error')).mean() #verified RMSE score value
for test model
print('Verified RMSE value for Testing Model: ' + str(RMSE2))
```

```
Verified RMSE value for Training model: 0.2685584755640468
Verified RMSE value for Testing Model: 0.26822947422677207
```

In [31]:

```
#Visualizing for Multiple Linear Regression y values

import seaborn as sns
ax1 = sns.distplot(y_train, hist=False, color="r", label="Actual Value")
sns.distplot(y_test, hist=False, color="b", label="Fitted Values" , ax=ax1);
```



In [33]:

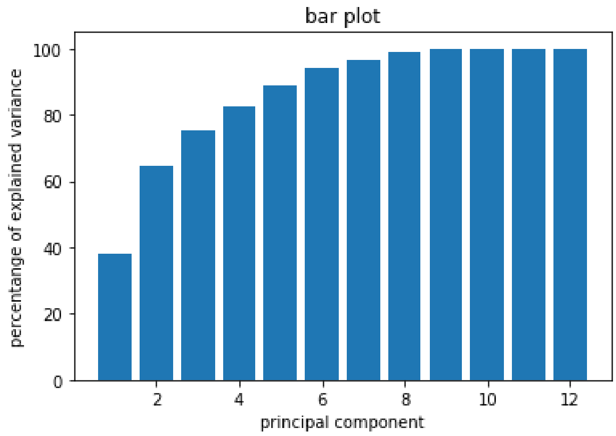
```
#**************************************Principal Component
Regression*********************************************************
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

pca=PCA()
X_reduced_train=pca.fit_transform(scale(X_train))
```

In [34]:

```
explained_variance_ratio=np.cumsum(np.round(pca.explained_variance_ratio_ , decimals=4)* 100)[0:20]
```

In [35]:

```
plt.bar(x=range(1, len(explained_variance_ratio)+1), height=explained_variance_ratio)
plt.ylabel('percentange of explained variance')
plt.xlabel('principal component')
plt.title('bar plot')
plt.show()
# 7 component is enough for model.
```



In [36]:

```
lm=LinearRegression()
pcr_model=lm.fit(X_reduced_train,y_train)
print('Intercept: ' + str(pcr_model.intercept_))
print('Coefficients: ' + str(pcr_model.coef_))
```

```
Intercept: -0.005461278328146827
Coefficients: [-0.13248389  0.18798958  0.64978531 -0.24235563 -0.038966    0.34814071
 -0.01512341 -0.38662935 -0.30625748  0.14768885  0.59449755 -3.03336011]
```

In [37]:

```
#Prediction
y_pred=pcr_model.predict(X_reduced_train)
np.sqrt(mean_squared_error(y_train,y_pred))
```

Out[37]:

```
0.268553922694838
```

In [38]:

```
df['motor_speed'].mean()
```

Out[38]:

```
-0.006335507987812318
```

In [39]:

```
#R squared
r2_score(y_train,y_pred)
```

Out[39]:

```
0.9280615197697045
```

In [40]:

```
# Prediction For testing error
pca2=PCA()

X_reduced_test=pca2.fit_transform(scale(X_test))
```

```
pcr_model2=lm.fit(X_test,y_test)

y_pred=pcr_model2.predict(X_reduced_test)

print('RMSE for test model : ' +str(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
RMSE for test model : 1.657960891802902
```

In [41]:

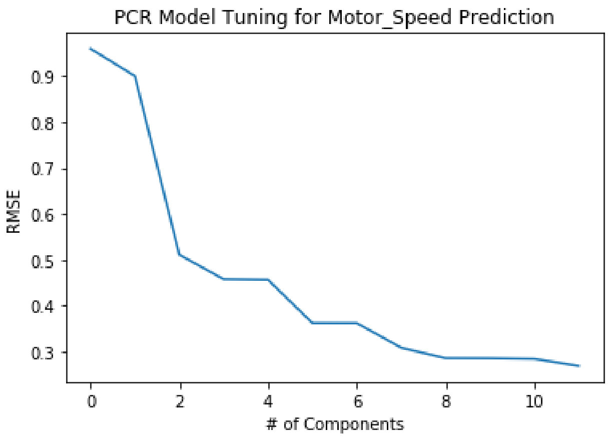
```
#Model Tuning for PCR

lm=LinearRegression()
pcr_model=lm.fit(X_reduced_train[:,0:10],y_train)
y_pred=pcr_model.predict(X_reduced_test[:,0:10])

from sklearn import model_selection

cv_10=model_selection.KFold(n_splits=10,
                            shuffle=True,
                            random_state=1)
```

In [44]:

```
lm=LinearRegression()
RMSE=[]

for i in np.arange(1,X_reduced_train.shape[1] + 1):
    score=np.sqrt(-1*model_selection.cross_val_score(lm,
                                                       X_reduced_train[:,:i],
                                                       y_train.ravel(),
                                                       cv=cv_10,
                                                       scoring='neg_mean_squared_error').mean())
    RMSE.append(score)
```

In [43]:

```
plt.plot(RMSE)
plt.xlabel('# of Components')
plt.ylabel('RMSE')
plt.title('PCR Model Tuning for Motor_Speed Prediction');
```



In [45]:

```
##10 component is good for the model because RMSE value is the smallest for this component number.

##That's why there is no need to tune the model.
```

In [6]:

```
#*********************Polynomial Regression****************************************
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score,mean_squared_error
import pandas as pd
```

In [7]:

```
df=pd.read_csv('C:\\Users\\Mahima Sharu\\pmsm_temperature_data.csv')
X=df.drop("motor_speed", axis=1)
y=df["motor_speed"]
from sklearn.model_selection import train_test_split,cross_val_score,cross_val_predict
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2, random_state=42)
training=df.copy()

quad = PolynomialFeatures (degree = 2)
x_quad = quad.fit_transform(X_train)

X_train,X_test,y_train,y_test = train_test_split(x_quad,y_train, random_state = 0)

plr = LinearRegression().fit(X_train,y_train)

Y_train_pred = plr.predict(X_train)
Y_test_pred = plr.predict(X_test)

print('Polynomial Linear Regression:' ,plr.score(X_test,y_test))
```

```
Polynomial Linear Regression: 0.9952573854207651
```

In [10]:

```
#Plotting Residual in Linear Regression

import matplotlib.pyplot as plt
from sklearn import linear_model,metrics
#Create linear regression object
reg=linear_model.LinearRegression()

#train the model using the train data sets
reg.fit(X_train,y_train)

#regression coefficients
print("Coefficients: \n", reg.coef_)

#Variance score
print("Variance score: {}".format(reg.score(X_test,y_test)))

plt.style.use('fivethirtyeight')

#plotting residual errors in training data
plt.scatter(reg.predict(X_train),reg.predict(X_train)-y_train,
            color="green", s=10, label="train data")

#plotting residual errors in test data
plt.scatter(reg.predict(X_test),reg.predict(X_test)-y_test,
            color="blue", s=10, label="test data")

#plot line for zero residual error
plt.hlines(y=0,xmin=-2, xmax=2, linewidth=2)

#plot legend
plt.legend(loc='upper right')

#plot title
plt.title("residual error")

plt.show()
```

```
Coefficients: 
 [-1.19652298e-12 -1.94751304e-03  8.86767668e-02 -1.51002720e-01
  8.46456963e-01 -4.33513690e-01 -4.11436417e-01  2.67644301e-01
  1.87790064e-02 -1.25988463e-01 -1.71853359e-01  2.54897914e-01
  7.40823232e-04 -4.34110557e-03 -3.24485408e-02  1.34249919e-02
 -1.05690117e-04  1.67560244e-01  8.52062538e-03 -1.56724820e-01
 -1.89292501e-03  5.46155535e-02 -9.53354640e-03 -1.36632243e-02
  2.28461480e-05  6.35750499e-02 -1.63781265e-01 -5.32017845e-02
```

```
 -9.07557261e-01 -5.67804823e-02  7.42197288e-01  1.03560119e-01
 -4.73717311e-01  3.83839811e-01 -4.06039694e-02 -6.25699752e-04
  1.38771773e-01  3.68937154e-02 -8.83271855e-01  2.50808967e-02
  1.04897761e+00 -3.74045176e-02  6.23406019e-01 -6.54857489e-01
  1.79516489e-01 -1.55441855e-03 -4.59404937e-05 -3.55526844e-01
 -3.64804458e-01  2.34531508e-01 -3.94524138e-03  2.36891662e-01
 -2.88885594e-01  1.08053335e-01 -3.56503666e-04  7.56339301e-01
  7.49643580e-01 -3.33692091e+00 -3.06318572e-01  4.23305718e+00
 -5.39802183e+00  2.10899507e+00 -3.84090475e-03  3.82247855e-01
 -6.62559679e-01 -3.21686761e-02  2.04955803e-01 -2.70200980e-01
  1.16040888e-01 -3.21988930e-04  2.61534726e+00  2.63818883e-01
 -3.58802305e+00  4.68164431e+00 -1.89010204e+00  2.42748089e-03
  8.90538096e-03 -3.41181602e-01  3.53620032e-01 -1.06780573e-01
  2.51351405e-04  8.80366838e-01 -1.46367360e+00  2.09986581e-01
  6.14259740e-04  5.86532523e-01 -1.59061514e-01  3.16715435e-03
  9.53273810e-03 -3.66373716e-03 -5.86982787e-06]
Variance score: 0.9952573854207651
```

In [ ]: