Assignment 16 (Neural_Network)(Gas_turbines)

In [1]:
```
import pandas as pd
import numpy as npd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings(action='ignore')
from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split


#Load data
df = pd.read_csv('gas_turbines.csv')
df.head()
```

Out[1]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	114.70	10.605	3.1547	82.722
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	114.72	10.598	3.2363	82.776
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	114.71	10.601	3.2012	82.468
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	114.72	10.606	3.1923	82.670
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	114.72	10.612	3.2484	82.311

In [2]:
```
df.shape
```

Out[2]:
```
(15039, 11)
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP', 'CO',
       'NOX'],
      dtype='object')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15039 entries, 0 to 15038
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AT      15039 non-null  float64
 1   AP      15039 non-null  float64
 2   AH      15039 non-null  float64
 3   AFDP    15039 non-null  float64
 4   GTEP    15039 non-null  float64
 5   TIT     15039 non-null  float64
 6   TAT     15039 non-null  float64
 7   TEY     15039 non-null  float64
 8   CDP     15039 non-null  float64
 9   CO      15039 non-null  float64
 10  NOX     15039 non-null  float64
dtypes: float64(11)
memory usage: 1.3 MB
```

In [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
AT	15039.0	17.764381	7.574323	0.522300	11.408000	18.1860	23.8625	34.9290
AP	15039.0	1013.199240	6.410760	985.850000	1008.900000	1012.8000	1016.9000	1034.2000
AH	15039.0	79.124174	13.793439	30.344000	69.750000	82.2660	90.0435	100.2000
AFDP	15039.0	4.200294	0.760197	2.087400	3.723900	4.1862	4.5509	7.6106
GTEP	15039.0	25.419061	4.173916	17.878000	23.294000	25.0820	27.1840	37.4020
TIT	15039.0	1083.798770	16.527806	1000.800000	1079.600000	1088.7000	1096.0000	1100.8000
TAT	15039.0	545.396183	7.866803	512.450000	542.170000	549.8900	550.0600	550.6100
TEY	15039.0	134.188464	15.829717	100.170000	127.985000	133.7800	140.8950	174.6100
CDP	15039.0	12.102353	1.103196	9.904400	11.622000	12.0250	12.5780	15.0810
CO	15039.0	1.972499	2.222206	0.000388	0.858055	1.3902	2.1604	44.1030
NOX	15039.0	68.190934	10.470586	27.765000	61.303500	66.6010	73.9355	119.8900

EDA & Feature Engineering

In [6]:
```
# Check for missing values
df.isna().sum()
```

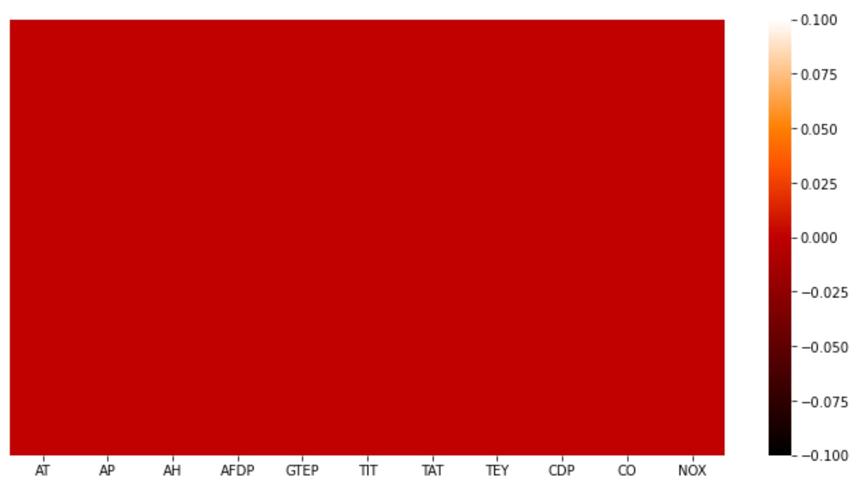
Out[6]:
```
AT      0
AP      0
AH      0
AFDP    0
GTEP    0
TIT     0
TAT     0
TEY     0
CDP     0
CO      0
NOX     0
dtype: int64
```

In [7]:
```
df.isna().any()
```

Out[7]:
```
AT      False
AP      False
AH      False
AFDP    False
GTEP    False
TIT     False
TAT     False
TEY     False
CDP     False
CO      False
NOX     False
dtype: bool
```

In [8]:
```
plt.rcParams['figure.figsize']=(12,6)
sns.heatmap(df.isna(), cmap =('gist_heat'), yticklabels=False)
```

Out[8]: <AxesSubplot:>



In [9]:
```
# check for duplicate values
df[df.duplicated()].shape
```

Out[9]: (0, 11)

In [10]:
```
df[df.duplicated()]
```

Out[10]:

AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX

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

Out[11]:
```
AT      float64
AP      float64
AH      float64
AFDP    float64
GTEP    float64
TIT     float64
TAT     float64
TEY     float64
CDP     float64
CO      float64
NOX     float64
dtype: object
```

In [12]:
```
df.nunique()
```

Out[12]:
```
AT      12086
AP        540
AH      12637
AFDP    11314
GTEP     8234
TIT       706
TAT      2340
TEY      4207
CDP      3611
CO      13096
NOX     11996
dtype: int64
```

Observation:

1.No missing values

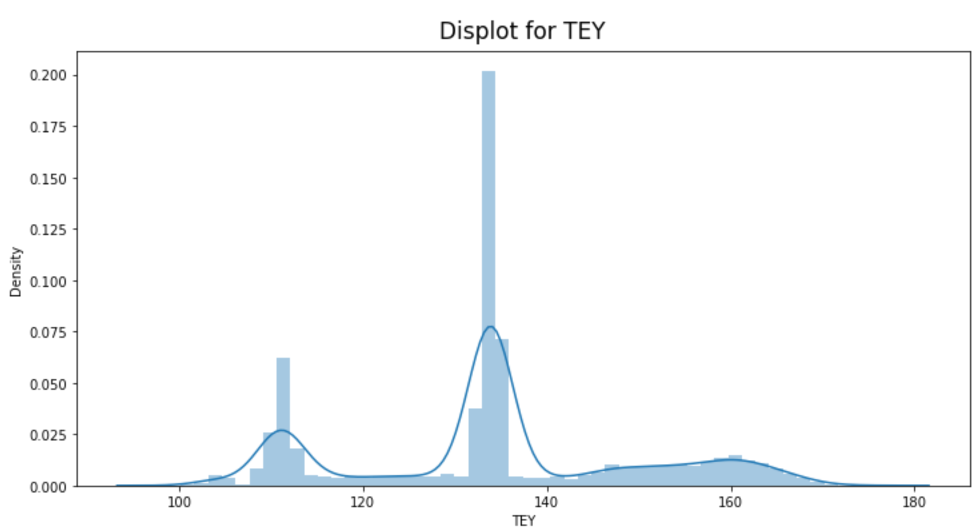
2.No duplicated values

3.All dtypes are correct

Data Visualisation

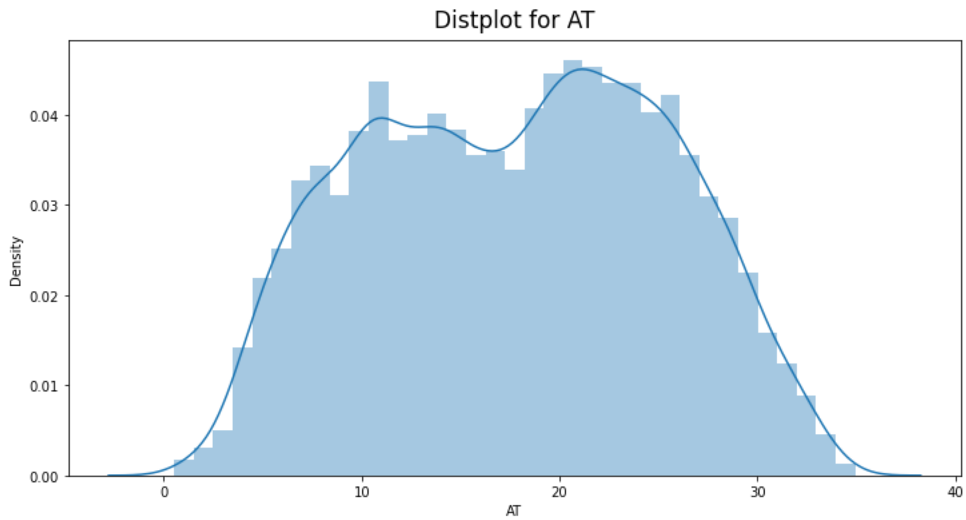
In [13]:
```
# Target variable
plt.title('Displot for TEY', fontsize=17, y = 1.01)
sns.distplot(df['TEY'])
```

Out[13]: <AxesSubplot:title={'center':'Displot for TEY'}, xlabel='TEY', ylabel='Density'>



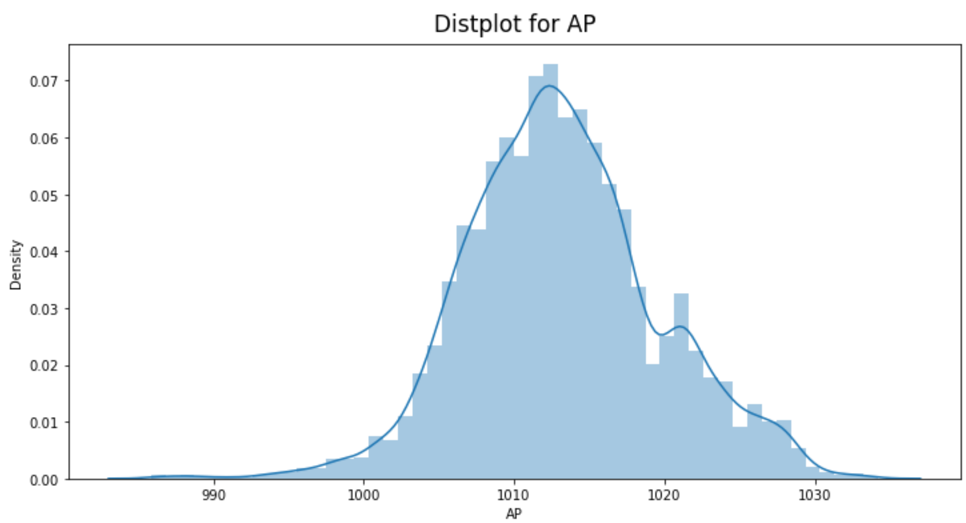
In [14]:
```
plt.title('Distplot for AT', fontsize=17, y = 1.01)
sns.distplot(df['AT'])
```

Out[14]: <AxesSubplot:title={'center':'Distplot for AT'}, xlabel='AT', ylabel='Density'>



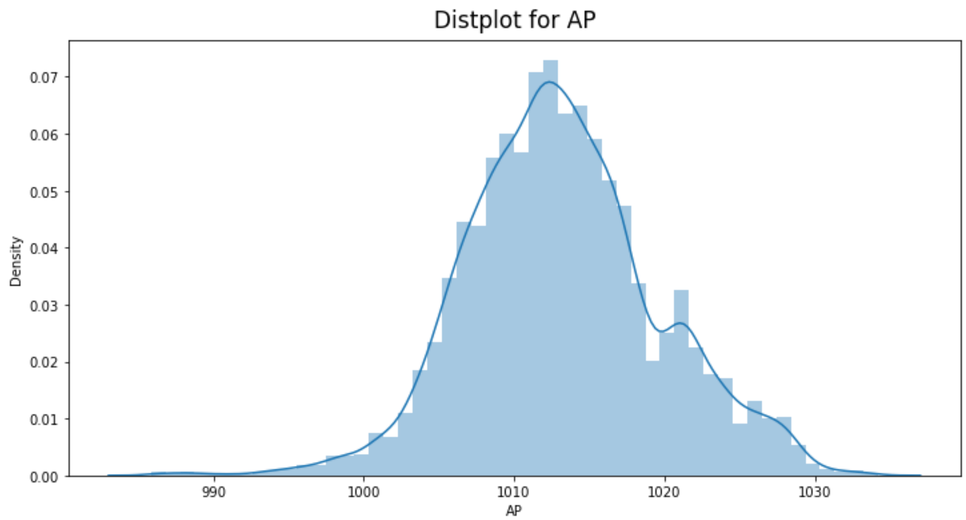
In [15]:
```
plt.title('Distplot for AP', fontsize=17, y = 1.01)
sns.distplot(df['AP'])
```

Out[15]: <AxesSubplot:title={'center':'Distplot for AP'}, xlabel='AP', ylabel='Density'>



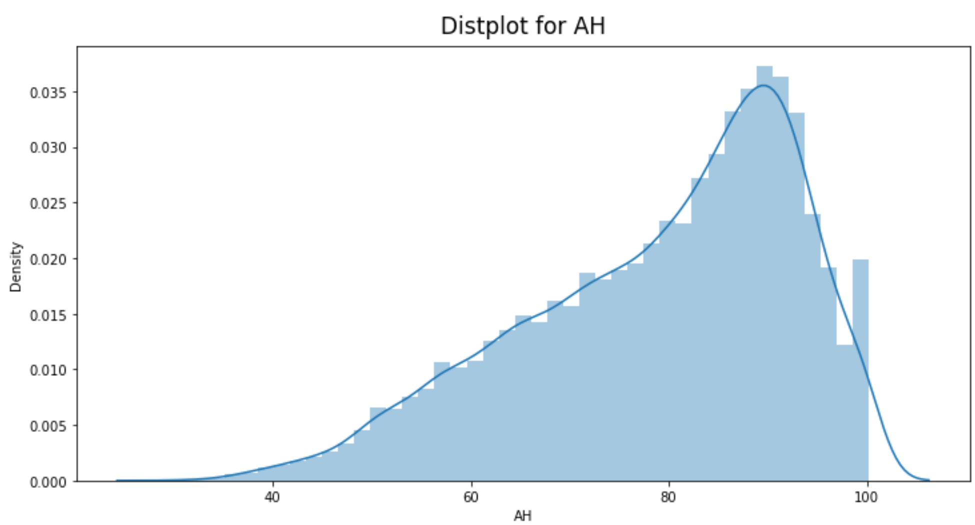
In [16]:
```
plt.title('Distplot for AP', fontsize=17, y = 1.01)
sns.distplot(df['AP'])
```

Out[16]: <AxesSubplot:title={'center':'Distplot for AP'}, xlabel='AP', ylabel='Density'>



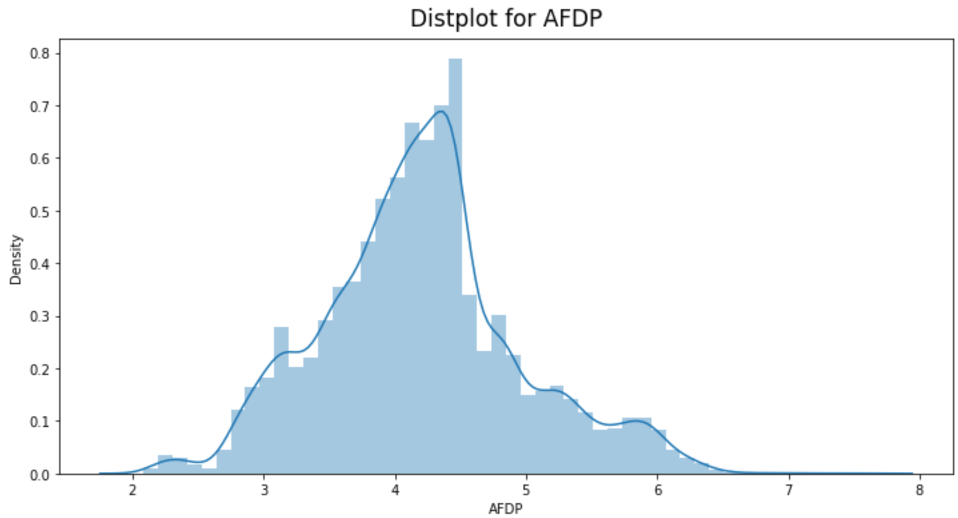
In [17]:
```
plt.title('Distplot for AH', fontsize=17, y = 1.01)
sns.distplot(df['AH'])
```

Out[17]: <AxesSubplot:title={'center':'Distplot for AH'}, xlabel='AH', ylabel='Density'>



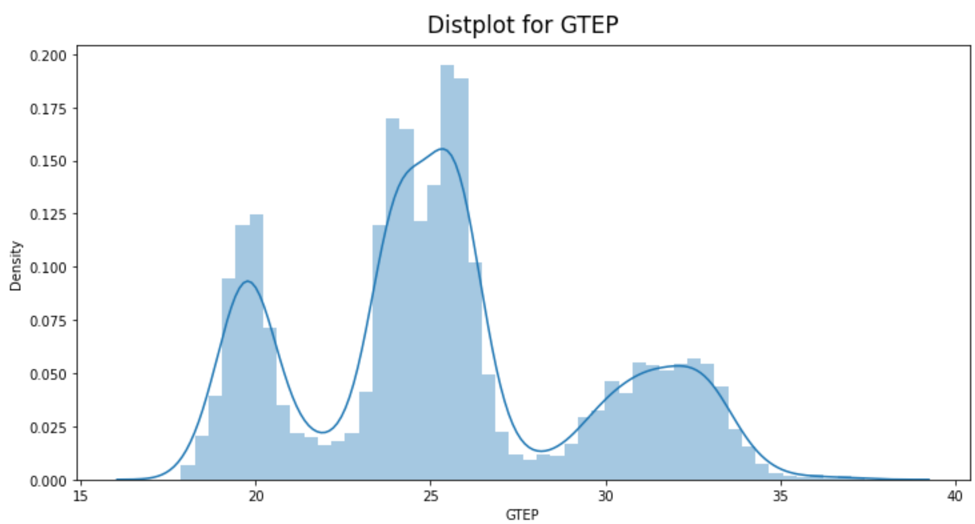
In [18]:
```
plt.title('Distplot for AFDP', fontsize=17, y = 1.01)
sns.distplot(df['AFDP'])
```

Out[18]: <AxesSubplot:title={'center':'Distplot for AFDP'}, xlabel='AFDP', ylabel='Density'>



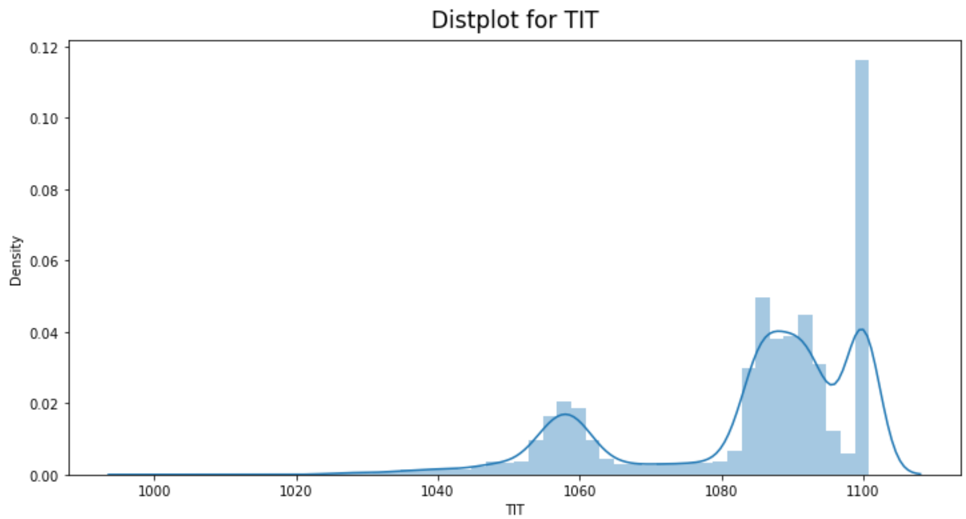
In [19]:
```
plt.title('Distplot for GTEP', fontsize=17, y = 1.01)
sns.distplot(df['GTEP'])
```

Out[19]: <AxesSubplot:title={'center':'Distplot for GTEP'}, xlabel='GTEP', ylabel='Density'>



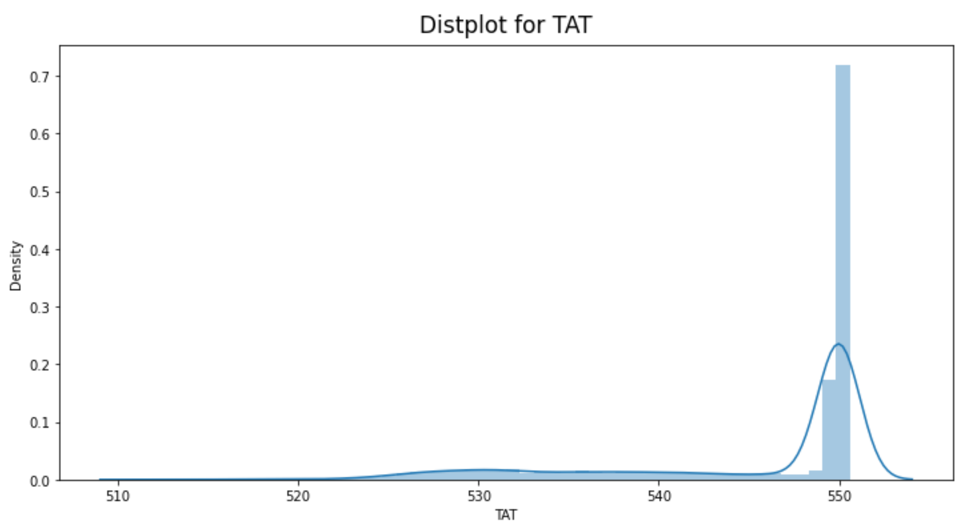
In [20]:
```
plt.title('Distplot for TIT', fontsize=17, y = 1.01)
sns.distplot(df['TIT'])
```

Out[20]: <AxesSubplot:title={'center':'Distplot for TIT'}, xlabel='TIT', ylabel='Density'>



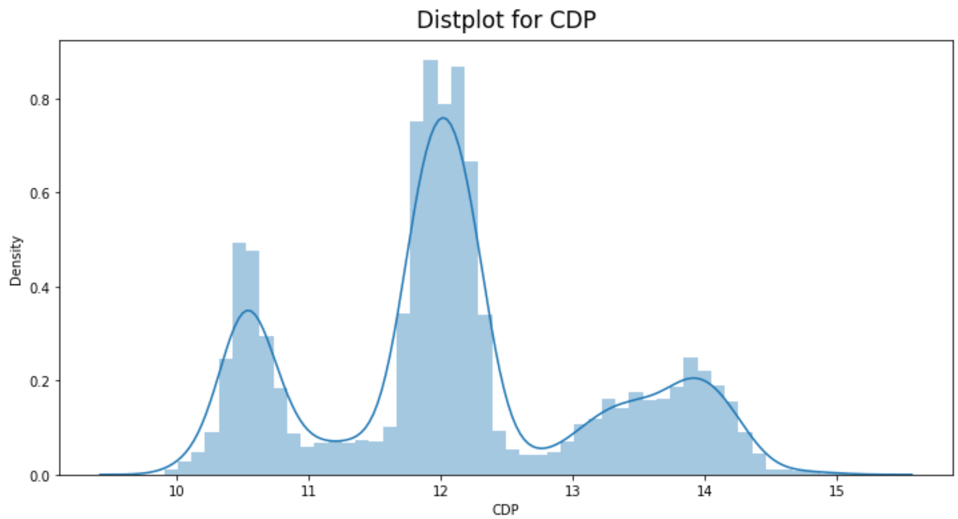
In [21]:
```
plt.title('Distplot for TAT', fontsize=17, y = 1.01)
sns.distplot(df['TAT'])
```

Out[21]: <AxesSubplot:title={'center':'Distplot for TAT'}, xlabel='TAT', ylabel='Density'>



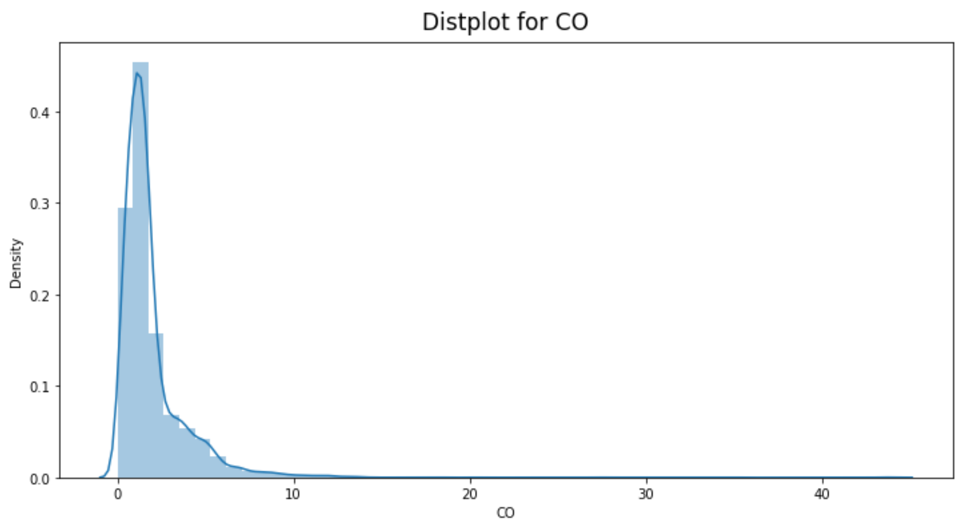
In [22]:
```
plt.title('Distplot for CDP', fontsize=17, y = 1.01)
sns.distplot(df['CDP'])
```

Out[22]: <AxesSubplot:title={'center':'Distplot for CDP'}, xlabel='CDP', ylabel='Density'>



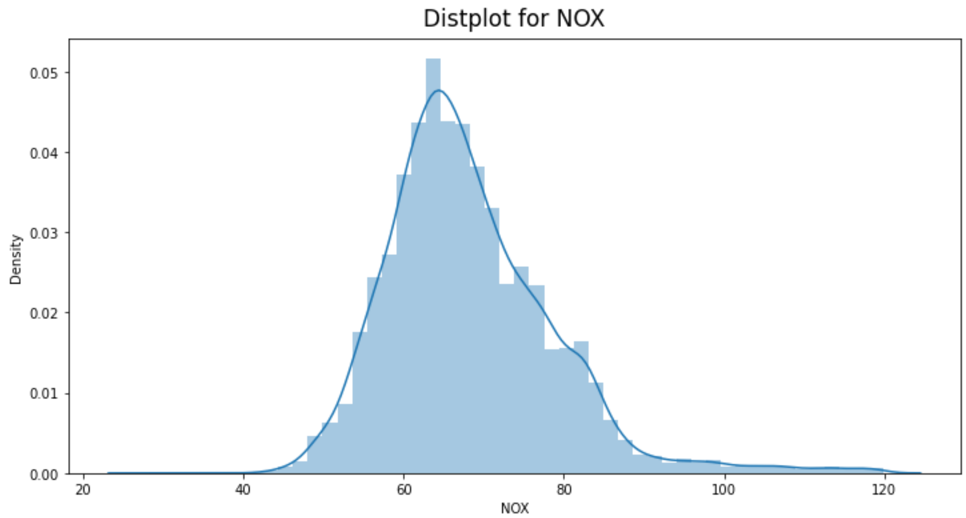
In [23]:
```
plt.title('Distplot for CO', fontsize=17, y = 1.01)
sns.distplot(df['CO'])
```

Out[23]: <AxesSubplot:title={'center':'Distplot for CO'}, xlabel='CO', ylabel='Density'>



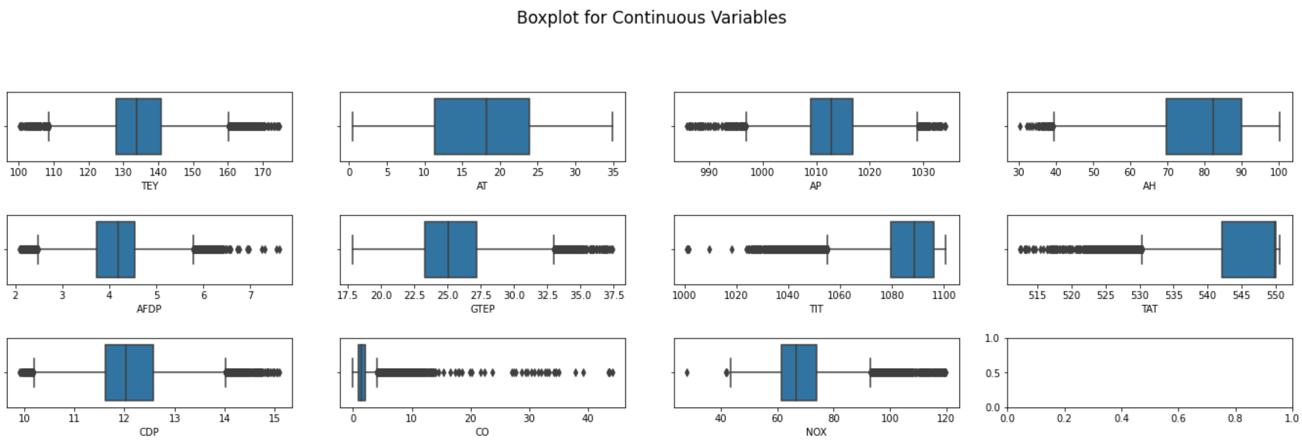
In [24]:
```
plt.title('Distplot for NOX', fontsize=17, y = 1.01)
sns.distplot(df['NOX'])
```

Out[24]: <AxesSubplot:title={'center':'Distplot for NOX'}, xlabel='NOX', ylabel='Density'>



In [25]:
```
import numpy as np
```

In [26]:
```
#check for outliers
fig, ax=plt.subplots(3,4, figsize=(19,6), sharex= False, sharey = False)
sns.boxplot(df.TEY, ax=ax[0,0])
sns.boxplot(df.AT, ax=ax[0,1])
sns.boxplot(df.AP, ax=ax[0,2])
sns.boxplot(df.AH, ax=ax[0,3])
sns.boxplot(df.AFDP, ax=ax[1,0])
sns.boxplot(df.GTEP, ax=ax[1,1])
sns.boxplot(df.TIT, ax=ax[1,2])
sns.boxplot(df.TAT, ax=ax[1,3])
sns.boxplot(df.CDP, ax=ax[2,0])
sns.boxplot(df.CO, ax=ax[2,1])
sns.boxplot(df.NOX, ax=ax[2,2])
plt.suptitle("Boxplot for Continuous Variables", fontsize= 17, y = 1.06)
plt.tight_layout(pad=2.0)

```

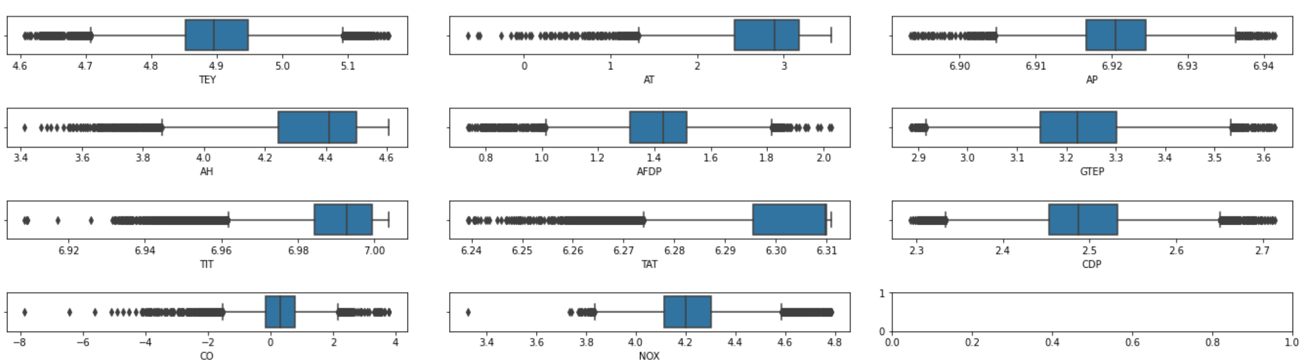


1.We have a noisy data

In [27]:

```
# check for outliers
fig, ax=plt.subplots(4,3, figsize=(19,6), sharex = False, sharey = False)
sns.boxplot(np.log(df.TEY), ax=ax[0,0])
sns.boxplot(np.log(df.AT), ax=ax[0,1])
sns.boxplot(np.log(df.AP), ax=ax[0,2])
sns.boxplot(np.log(df.AH), ax=ax[1,0])
sns.boxplot(np.log(df.AFDP), ax=ax[1,1])
sns.boxplot(np.log(df.GTEP), ax=ax[1,2])
sns.boxplot(np.log(df.TIT), ax=ax[2,0])
sns.boxplot(np.log(df.TAT), ax=ax[2,1])
sns.boxplot(np.log(df.CDP), ax=ax[2,2])
sns.boxplot(np.log(df.CO), ax=ax[3,0])
sns.boxplot(np.log(df.NOX), ax=ax[3,1])
plt.suptitle("Log Transformation for Continous Variables", fontsize=17, y = 1.06)
plt.tight_layout(pad=2.0)
```

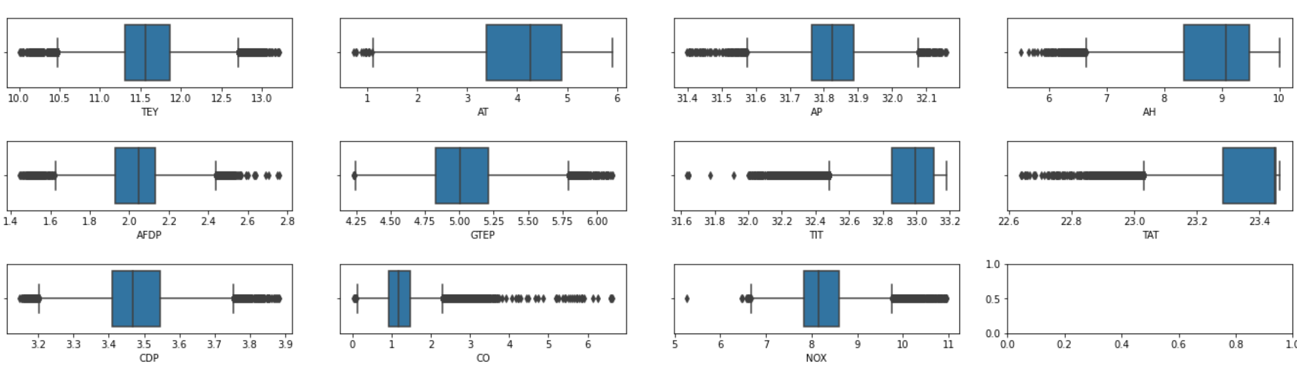
Log Transformation for Continous Variables



In [28]:

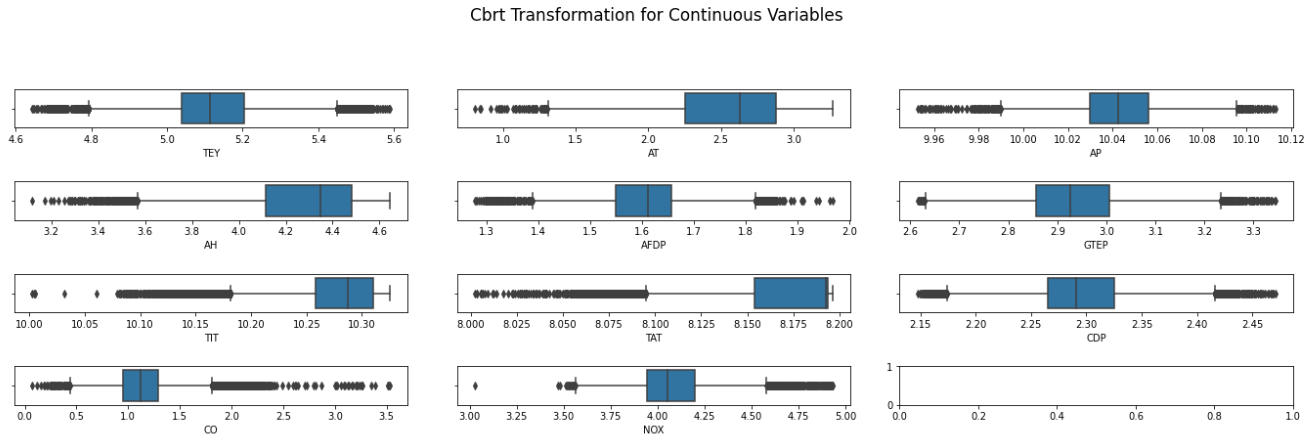
```
fig, ax=plt.subplots(3,4, figsize=(19,6), sharex= False, sharey = False)
sns.boxplot(np.sqrt(df.TEY), ax=ax[0,0])
sns.boxplot(np.sqrt(df.AT), ax=ax[0,1])
sns.boxplot(np.sqrt(df.AP), ax=ax[0,2])
sns.boxplot(np.sqrt(df.AH), ax=ax[0,3])
sns.boxplot(np.sqrt(df.AFDP), ax=ax[1,0])
sns.boxplot(np.sqrt(df.GTEP), ax=ax[1,1])
sns.boxplot(np.sqrt(df.TIT), ax=ax[1,2])
sns.boxplot(np.sqrt(df.TAT), ax=ax[1,3])
sns.boxplot(np.sqrt(df.CDP), ax=ax[2,0])
sns.boxplot(np.sqrt(df.CO), ax=ax[2,1])
sns.boxplot(np.sqrt(df.NOX), ax=ax[2,2])
plt.suptitle("SQRT Transformation for Continuous Variables", fontsize= 17, y = 1.06)
plt.tight_layout(pad=2.0)

```

SQRT Transformation for Continuous Variables



In [29]:

```python
fig, ax=plt.subplots(4,3, figsize=(19,6), sharex= False, sharey = False)
sns.boxplot(np.cbrt(df.TEY), ax=ax[0,0])
sns.boxplot(np.cbrt(df.AT), ax=ax[0,1])
sns.boxplot(np.cbrt(df.AP), ax=ax[0,2])
sns.boxplot(np.cbrt(df.AH), ax=ax[1,0])
sns.boxplot(np.cbrt(df.AFDP), ax=ax[1,1])
sns.boxplot(np.cbrt(df.GTEP), ax=ax[1,2])
sns.boxplot(np.cbrt(df.TIT), ax=ax[2,0])
sns.boxplot(np.cbrt(df.TAT), ax=ax[2,1])
sns.boxplot(np.cbrt(df.CDP), ax=ax[2,2])
sns.boxplot(np.cbrt(df.CO), ax=ax[3,0])
sns.boxplot(np.cbrt(df.NOX), ax=ax[3,1])
plt.suptitle("Cbrt Transformation for Continuous Variables", fontsize= 17, y = 1.06)
plt.tight_layout(pad=2.0)
```

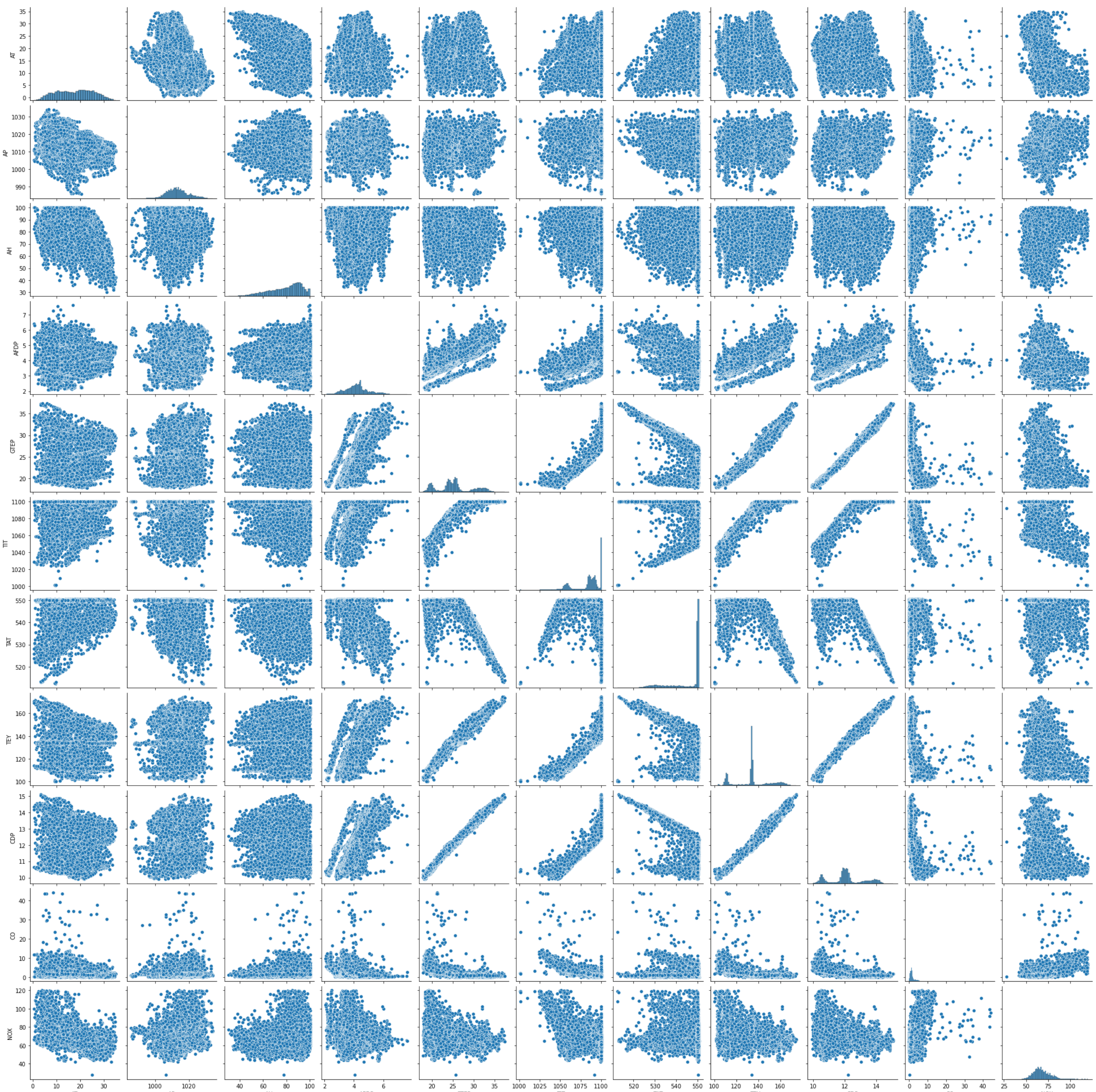


1.None of the transformations are helpful to treat the outliers.

Dependency of Target variable on diff Features.

In [30]:
```
sns.pairplot(df)
```

Out[30]: <seaborn.axisgrid.PairGrid at 0x141556a4a60>



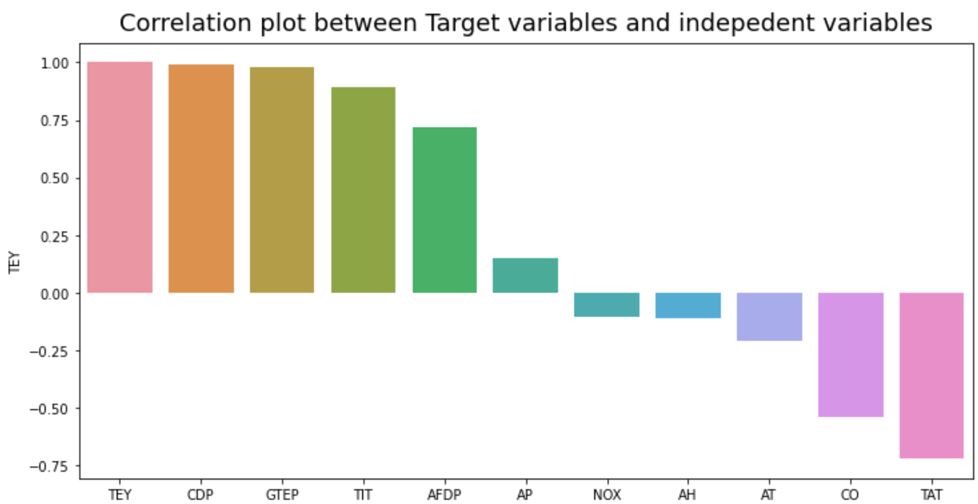
In [31]:
```
corr = pd.DataFrame(data = df.corr().iloc[:,7], index=df.columns)
corr = corr.sort_values(by='TEY', ascending=False)
corr
```

Out[31]:

	TEY
TEY	1.000000
CDP	0.988473
GTEP	0.977042
TIT	0.891587
AFDP	0.717995
AP	0.146939
NOX	-0.102631
AH	-0.110272
AT	-0.207495
CO	-0.541751
TAT	-0.720356

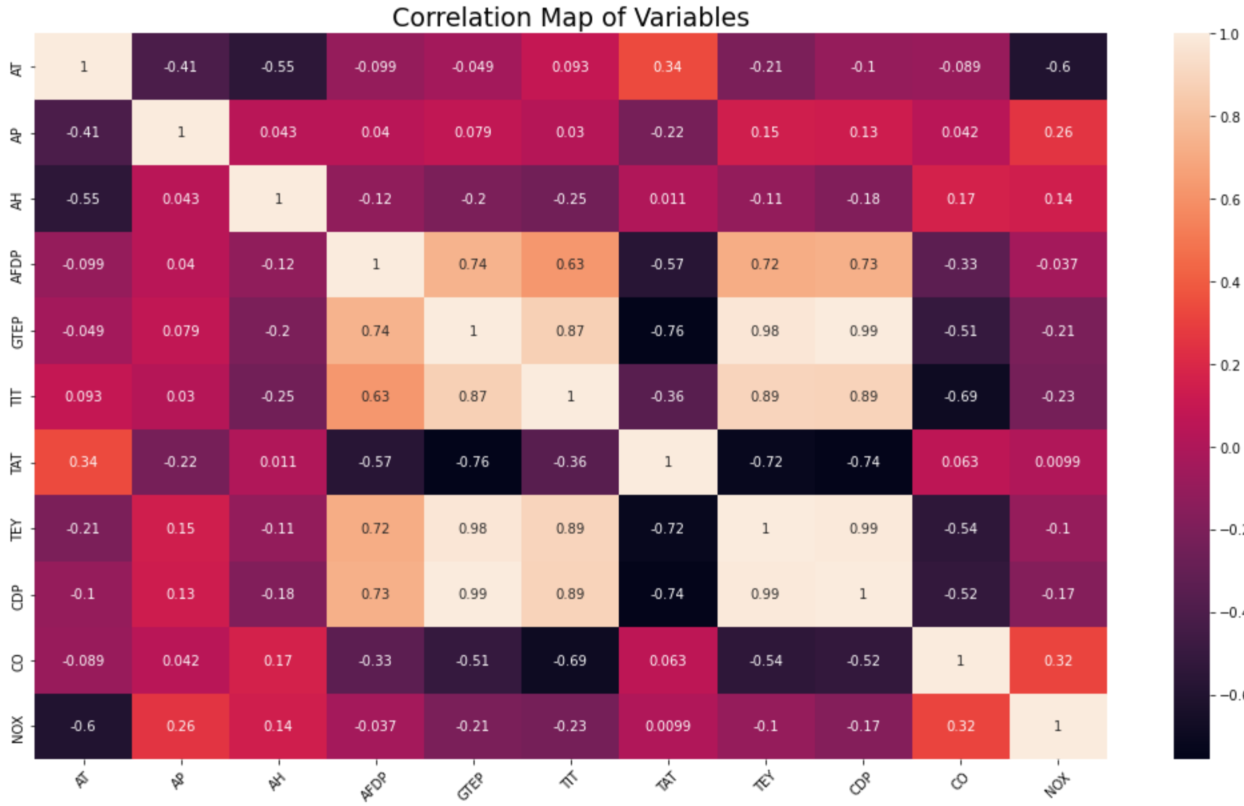
In [32]:
```
plt.title("Correlation plot between Target variables and indepedent variables",y=1.01, fontsize=18)
sns.barplot(x = corr.index, y = corr.TEY)
```

Out[32]: <AxesSubplot:title={'center':'Correlation plot between Target variables and indepedent variables'}, ylabel='TEY'>



In [33]:
```
fig= plt.figure(figsize=(18, 10))
sns.heatmap(df.corr(), annot=True);
plt.xticks(rotation=45)
plt.title("Correlation Map of Variables", fontsize=19)
```

Out[33]: Text(0.5, 1.0, 'Correlation Map of Variables')



In [34]:
```
!pip install ppscore
```

```
Requirement already satisfied: ppscore in c:\users\admin\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: pandas<2.0.0,>=1.0.0 in c:\users\admin\anaconda3\lib\site-packages (from ppscore) (1.3.4)
Requirement already satisfied: scikit-learn<2.0.0,>=0.20.2 in c:\users\admin\anaconda3\lib\site-packages (from ppscore) (1.1.3)
Requirement already satisfied: pytz>=2017.3 in c:\users\admin\anaconda3\lib\site-packages (from pandas<2.0.0,>=1.0.0->ppscore)
(2021.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\lib\site-packages (from pandas<2.0.0,>=1.0.0->ppscore)
(1.23.5)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\admin\anaconda3\lib\site-packages (from pandas<2.0.0,>=1.0.0-
>ppscore) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas<2.0.
0,>=1.0.0->ppscore) (1.16.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn<2.0.0,>=0.20.2->pp
score) (1.9.3)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn<2.0.0,>=0.20.2->p
pscore) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn<2.0.0,>=0.
20.2->ppscore) (2.2.0)
```

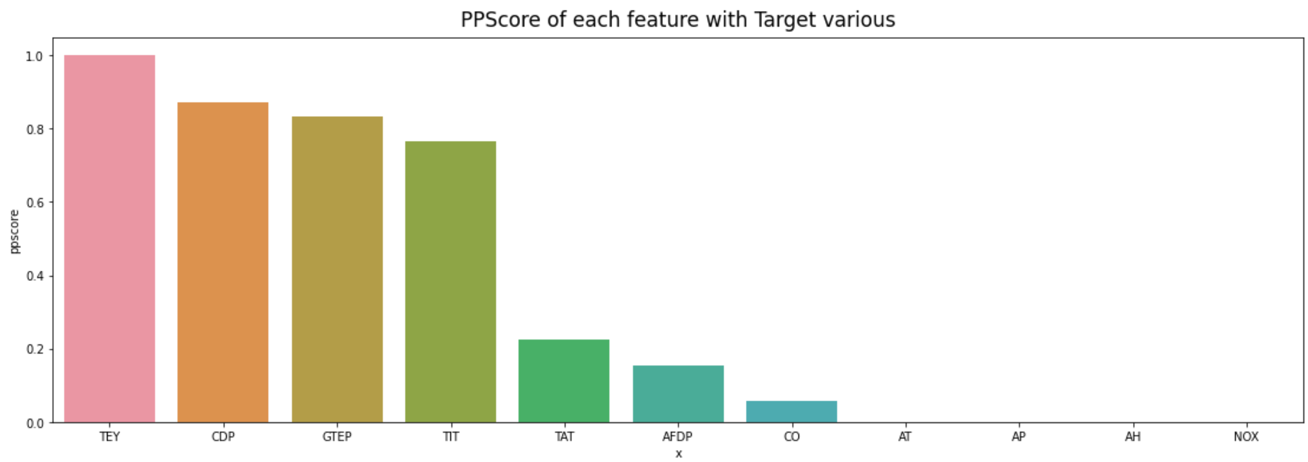
In [35]:
```
import ppscore as PPS
score = PPS.matrix(df)
score_s = score[score['y']=='TEY']
score_s.sort_values(by="ppscore", ascending=False)
```

Out[35]:

	x	y	ppscore	case	is_valid_score	metric	baseline_score	model_score	model
84	TEY	TEY	1.000000	predict_itself	True	None	0.000000	1.000000	None
95	CDP	TEY	0.872285	regression	True	mean absolute error	11.172076	1.426840	DecisionTreeRegressor()
51	GTEP	TEY	0.832336	regression	True	mean absolute error	11.172076	1.873154	DecisionTreeRegressor()
62	TIT	TEY	0.766040	regression	True	mean absolute error	11.172076	2.613821	DecisionTreeRegressor()
73	TAT	TEY	0.226050	regression	True	mean absolute error	11.172076	8.646631	DecisionTreeRegressor()
40	AFDP	TEY	0.152509	regression	True	mean absolute error	11.172076	9.468234	DecisionTreeRegressor()
106	CO	TEY	0.055869	regression	True	mean absolute error	11.172076	10.547906	DecisionTreeRegressor()
7	AT	TEY	0.000000	regression	True	mean absolute error	11.172076	16.007470	DecisionTreeRegressor()
18	AP	TEY	0.000000	regression	True	mean absolute error	11.172076	12.475617	DecisionTreeRegressor()
29	AH	TEY	0.000000	regression	True	mean absolute error	11.172076	16.950976	DecisionTreeRegressor()
117	NOX	TEY	0.000000	regression	True	mean absolute error	11.172076	14.537337	DecisionTreeRegressor()

In [36]:
```
plt.rcParams['figure.figsize']=(19,6)
sns.barplot(x='x', y='ppscore', data=score_s.sort_values(by='ppscore', ascending=False))
plt.title("PPScore of each feature with Target various", fontsize=17, y=1.01)
```

Out[36]: `Text(0.5, 1.01, 'PPScore of each feature with Target various')`



Observation:

1. From correlation martix as well as ppscore we can clearly see that TEY is highly dependent on 'CDP', 'GTEP', 'TIT'.
2. We can drop 'AT', 'AP', 'AH', 'NOX', as they have very less impact on dependent variables.

Check for outliers.

In [37]:
```
# Check for Outliers
from sklearn.ensemble import IsolationForest
data1=df.copy()

#training the model
clf = IsolationForest(random_state=10, contamination=.001)
clf.fit(data1)
data1['anamoly'] = clf.predict(data1.iloc[:,0:11])
outliers = data1[data1['anamoly']==-1]
```

In [38]:
```
outliers
```

Out[38]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX	anamoly
261	5.66020	1018.30	86.968	3.8404	21.079	1028.5	523.86	112.02	10.963	43.4280	99.237	-1
553	3.55320	1027.30	90.871	4.2162	21.464	1041.2	531.68	117.76	10.984	8.8254	106.840	-1
763	1.81300	1007.20	74.980	3.6967	19.958	1026.4	528.18	111.72	10.553	12.0900	114.940	-1
764	1.49880	1006.30	76.734	3.7063	20.041	1027.6	528.79	112.28	10.585	11.6520	112.830	-1
765	0.97877	1005.70	78.978	3.7379	20.084	1027.9	528.52	112.71	10.628	11.6910	108.880	-1
993	4.36570	1021.60	85.528	3.9574	20.263	1025.6	525.72	111.35	10.652	12.7860	112.270	-1
6896	17.13200	1010.80	80.503	2.2148	18.484	1034.1	539.98	102.07	10.182	11.5150	110.760	-1
7019	7.02760	997.23	97.761	2.0992	19.227	1037.2	538.53	109.63	10.338	11.0440	105.060	-1
7470	7.04730	1019.60	96.885	2.4558	19.501	1032.0	532.32	109.21	10.567	11.3740	112.230	-1
9920	15.17900	1017.60	71.630	2.7816	18.435	1027.8	533.45	103.64	10.143	12.1440	113.800	-1
13820	14.18300	1023.10	78.110	3.1557	18.869	1025.0	530.16	103.80	10.340	13.3130	116.340	-1
13921	11.58500	1018.20	92.751	3.2518	18.784	1009.5	519.71	100.83	10.253	39.0500	111.780	-1
14100	9.40970	1027.90	82.224	3.3003	18.987	1001.4	512.60	100.32	10.495	23.6290	107.890	-1
14278	9.90780	1026.10	65.923	3.3126	19.366	1024.5	527.21	108.08	10.506	20.2710	105.660	-1
14317	3.93850	1021.30	90.536	3.4765	20.031	1026.6	526.30	111.70	10.683	14.0350	114.700	-1
14320	3.49070	1020.80	91.519	3.5309	20.098	1025.8	525.35	111.91	10.761	11.9210	113.900	-1

1. These are the outlires in our data.

Data Preprocessing

In [39]:
```
df.shape
```

Out[39]: (15039, 11)

In [40]:
```
# drop the outliers
df = df.drop(outliers.index)
df.shape
```

Out[40]: (15023, 11)

In [41]:
```
#reset index after dropping outlires
df = df.reset_index()
df = df.drop('index', axis = 1)
df
```

Out[41]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	114.70	10.605	3.1547	82.722
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	114.72	10.598	3.2363	82.776
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	114.71	10.601	3.2012	82.468
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	114.72	10.606	3.1923	82.670
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	114.72	10.612	3.2484	82.311
...	...	...	...	...	...	...	...	...	...	...	...
15018	9.0301	1005.6	98.460	3.5421	19.164	1049.7	546.21	111.61	10.400	4.5186	79.559
15019	7.8879	1005.9	99.093	3.5059	19.414	1046.3	543.22	111.78	10.433	4.8470	79.917
15020	7.2647	1006.3	99.496	3.4770	19.530	1037.7	537.32	110.19	10.483	7.9632	90.912
15021	7.0060	1006.8	99.008	3.4486	19.377	1043.2	541.24	110.74	10.533	6.2494	93.227
15022	6.9279	1007.2	97.533	3.4275	19.306	1049.9	545.85	111.58	10.583	4.9816	92.498

15023 rows × 11 columns

In [42]:
```
df = df.drop(['AT', 'AP', 'AH', 'NOX'], axis=1)
```

In [43]:
```
df.shape
```

Out[43]: (15023, 7)

Converting independent feature into normalised and standardized data

In [44]:
```
# Standardize & Normalize the data
norm = MinMaxScaler()
std = StandardScaler()

df_norm = pd.DataFrame(norm.fit_transform(df), columns=df.columns)  # data between -3 to +3
df_std = pd.DataFrame(std.fit_transform(df), columns=df.columns)    # data betweem -1 to +1
```

Take a smaller sample to build a model.

In [45]:
```
# We will take a small model as this is a large data and will take huge ammount of time to
# build model to randomly shuffle and select a % of data
temp = df_std.sample(frac=1)          # shuffle all the data
temp_s = df_std.sample(frac=0.1)      # shuffle and select only 10% of the data randomly to train
```

In [46]:
```
temp_s
```

Out[46]:

	AFDP	GTEP	TIT	TAT	TEY	CDP	CO
10473	0.351488	1.290082	0.976474	-1.174103	1.191558	1.313317	-0.509066
3911	-0.811922	-0.997964	-1.196849	0.554989	-1.320629	-1.038528	1.564372
12188	-0.463255	0.081457	0.489455	0.633989	-0.014266	0.184105	-0.774782
6408	-0.638576	-1.318458	-1.568201	0.594489	-1.491986	-1.435792	1.329205
4776	0.448757	0.260282	0.629473	0.602134	0.025569	0.116987	-0.612006
...	...	...	...	...	...	...	...
392	0.471659	-0.247427	0.081576	0.575376	0.001541	-0.169624	-0.014359
1909	0.040333	-1.037516	-1.135972	0.580473	-1.019015	-1.050318	0.158078
11709	0.365572	1.104546	0.982562	-0.686084	0.862754	1.008565	-0.857748
4850	-0.366907	-0.560251	-0.423706	0.546069	-0.712974	-0.522446	-0.262947
9785	0.415062	1.608898	0.976474	-1.882560	1.651251	1.626231	-0.492619

1502 rows × 7 columns

Splitting data into target variable and independent variables.

In [47]:
```
x = temp_s.drop('TEY', axis=1)
y = temp_s['TEY']
x
```

Out[47]:

	AFDP	GTEP	TIT	TAT	CDP	CO
10473	0.351488	1.290082	0.976474	-1.174103	1.313317	-0.509066
3911	-0.811922	-0.997964	-1.196849	0.554989	-1.038528	1.564372
12188	-0.463255	0.081457	0.489455	0.633989	0.184105	-0.774782
6408	-0.638576	-1.318458	-1.568201	0.594489	-1.435792	1.329205
4776	0.448757	0.260282	0.629473	0.602134	0.116987	-0.612006
...	...	...	...	...	...	...
392	0.471659	-0.247427	0.081576	0.575376	-0.169624	-0.014359
1909	0.040333	-1.037516	-1.135972	0.580473	-1.050318	0.158078
11709	0.365572	1.104546	0.982562	-0.686084	1.008565	-0.857748
4850	-0.366907	-0.560251	-0.423706	0.546069	-0.522446	-0.262947
9785	0.415062	1.608898	0.976474	-1.882560	1.626231	-0.492619

1502 rows × 6 columns

Creating train and test data for model validation

In [48]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```

In [49]:
```
x_train.shape, y_test.shape, y_train.shape, y_test.shape
```

Out[49]:
```
((1126, 6), (376,), (1126,), (376,))
```

Build a model.

In [50]:
```
# Importing the neccessary packages
import tensorflow as tf
import keras
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from tensorflow.keras.optimizers import Adam
from keras.layers import Dropout
tf.config.experimental.list_logical_devices('GPU')    # to use GPU for faster processing of model
```

Out[50]:
```
[]
```

In [51]:
```
# create model with 2 hidden layers
def create_model_two_hidden_layers():
    model = Sequential()
    model.add(Dense(5, input_dim=6, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(6, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(10, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(1))

    adam=Adam(lr=0.001)
    model.compile(loss='mse', optimizer=adam, metrics=['mse', 'mae', 'mape'])
    return model
```

In [52]:
```
model1 = create_model_two_hidden_layers()
print("Here is the summary of the model:")
model1.summary()
```

```
Here is the summary of the model:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 35

 dense_1 (Dense)             (None, 6)                 36

 dense_2 (Dense)             (None, 10)                70

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 152
Trainable params: 152
Non-trainable params: 0
_________________________________________________________________
```

In [53]:
```
# create model with 3 hidden layers
def create_model_three_hidden_layers():
    model = Sequential()
    model.add(Dense(32, input_dim=6, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(32, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(64, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(128, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(1))

    adam=Adam(lr=0.01)
    model.compile(loss='mse', optimizer=adam, metrics=['mse', 'mae', 'mape'])
    return model
```

In [54]:
```
model2 = create_model_three_hidden_layers()
print("Here is the summary of the model:")
model2.summary()
```

```
Here is the summary of the model:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 32)                224       
                                                                 
 dense_5 (Dense)             (None, 32)                1056      
                                                                 
 dense_6 (Dense)             (None, 64)                2112      
                                                                 
 dense_7 (Dense)             (None, 128)               8320      
                                                                 
 dense_8 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 11,841
Trainable params: 11,841
Non-trainable params: 0
_________________________________________________________________
```

In [55]:
```
%%time
epochs=500
batch_size=50

print("Here is the summary of this model:")
model2.summary()

with tf.device('/GPU:0'):
  model2.fit(x_train,y_train, verbose = 0,batch_size = batch_size,epochs = epochs, shuffle=True)
```

```
Here is the summary of this model:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 32)                224       
                                                                 
 dense_5 (Dense)             (None, 32)                1056      
                                                                 
 dense_6 (Dense)             (None, 64)                2112      
                                                                 
 dense_7 (Dense)             (None, 128)               8320      
                                                                 
 dense_8 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 11,841
Trainable params: 11,841
Non-trainable params: 0
_________________________________________________________________
Wall time: 22.2 s
```

In [56]:
```
print("Predicted Values:")
model2.predict(x_test[:10])
```

```
Predicted Values:
1/1 [==============================] - 0s 129ms/step
```

Out[56]:
```
array([[-1.833806  ],
       [-1.4910991 ],
       [ 0.00822169],
       [-0.00512612],
       [-1.605835  ],
       [ 0.9588574 ],
       [-1.4259658 ],
       [ 0.01884493],
       [-0.01247218],
       [-0.04152176]], dtype=float32)
```

In [57]:
```
print('Actual Values')
y_test[:10]
```

```
Actual Values
```

Out[57]:
```
2317    -1.931445
6485    -1.474914
9091    -0.004782
5443    -0.028810
10027   -1.484398
11794    0.800787
10773   -1.402830
8915     0.098286
6588    -0.035133
1384    -0.095203
Name: TEY, dtype: float64
```

In [58]:
```
loss, mae, mse, mape = model2.evaluate(x_train, y_train)
print('\n', "Results for model 2:", '\n', "Training Loss:", loss, '\n', "Training Mean Absolute Error:" , mae, '\n', "Traini
```

```
36/36 [==============================] - 0s 2ms/step - loss: 0.0049 - mse: 0.0049 - mae: 0.0501 - mape: 76.3284

 Results for model 2:
 Training Loss: 0.0049224658869206905
 Training Mean Absolute Error: 0.0049224658869206905
 Training Mean Squared Error: 0.050090059638023376
```

In [59]:
```
loss, mae, mse, mape = model2.evaluate(x_train, y_train)
print('\n', "Results for model 2:", '\n', "Test Loss:", loss, '\n', "Test Mean Absolute Error:" , mae, '\n', "Test Mean Squa
```

```
36/36 [==============================] - 0s 2ms/step - loss: 0.0049 - mse: 0.0049 - mae: 0.0501 - mape: 76.3284

 Results for model 2:
 Test Loss: 0.0049224658869206905
 Test Mean Absolute Error: 0.0049224658869206905
 Test Mean Squared Error: 0.050090059638023376
```

Observations

1. We got pretty good results for their model.
2. Train and test errors are also quiet similar, which means our model is not overfitted or underfitted.
3. Still we will try to get best results by doing hyperparameter tuning.

Hyperparameter Tuning to get best options for:

1. batchsize
2. epochs
3. neurons
4. learning rate
5. dropout
6. kernel initializer
7. activation function

In [60]:
```
from numpy import array
from sklearn.model_selection import KFold
```

In [61]:

```python
# Create the model
#get best value for batch size and epochs by hyperparameter tuning
model = KerasRegressor(build_fn = create_model_three_hidden_layers,verbose = 0)
# Define the grid search parameters
batch_size = [30,50,70]
epochs = [300,500,800]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size = batch_size,epochs = epochs)
# Build and fit the GridSearchCV
grid = GridSearchCV(estimator = model,param_grid = param_grid,cv = KFold(),verbose = 10)
grid_result = grid.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START batch_size=30, epochs=300...................................
[CV 1/5; 1/9] END ...batch_size=30, epochs=300;, score=-0.007 total time=  16.3s
[CV 2/5; 1/9] START batch_size=30, epochs=300...................................
[CV 2/5; 1/9] END ...batch_size=30, epochs=300;, score=-0.012 total time=  17.3s
[CV 3/5; 1/9] START batch_size=30, epochs=300...................................
[CV 3/5; 1/9] END ...batch_size=30, epochs=300;, score=-0.008 total time=  17.1s
[CV 4/5; 1/9] START batch_size=30, epochs=300...................................
[CV 4/5; 1/9] END ...batch_size=30, epochs=300;, score=-0.007 total time=  17.2s
[CV 5/5; 1/9] START batch_size=30, epochs=300...................................
[CV 5/5; 1/9] END ...batch_size=30, epochs=300;, score=-0.006 total time=  17.9s
[CV 1/5; 2/9] START batch_size=30, epochs=500...................................
[CV 1/5; 2/9] END ...batch_size=30, epochs=500;, score=-0.015 total time=  26.5s
[CV 2/5; 2/9] START batch_size=30, epochs=500...................................
[CV 2/5; 2/9] END ...batch_size=30, epochs=500;, score=-0.010 total time=  28.6s
[CV 3/5; 2/9] START batch_size=30, epochs=500...................................
[CV 3/5; 2/9] END ...batch_size=30, epochs=500;, score=-0.010 total time=  28.7s
[CV 4/5; 2/9] START batch_size=30, epochs=500...................................
[CV 4/5; 2/9] END ...batch_size=30, epochs=500;, score=-0.008 total time=  28.7s
```

In [62]:

```python
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for means, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(means, stdev, param))
```

```
Best : -0.007391261588782072, using {'batch_size': 70, 'epochs': 300}
-0.007894857414066791,0.001991391197786599 with: {'batch_size': 30, 'epochs': 300}
-0.009753851965069772,0.0030524625775304425 with: {'batch_size': 30, 'epochs': 500}
-0.007557791378349066,0.002037569500462883 with: {'batch_size': 30, 'epochs': 800}
-0.010061297938227654,0.0044086401658283494 with: {'batch_size': 50, 'epochs': 300}
-0.009114649146795273,0.0015327226797154107 with: {'batch_size': 50, 'epochs': 500}
-0.009040820598602294,0.0027362051480239943 with: {'batch_size': 50, 'epochs': 800}
-0.007391261588782072,0.0012853548730909179 with: {'batch_size': 70, 'epochs': 300}
-0.009143414068967104,0.0022328322340198397 with: {'batch_size': 70, 'epochs': 500}
-0.008697202522307634,0.0010586759703553783 with: {'batch_size': 70, 'epochs': 800}
```

In [63]:
```
#get best value for learning rate and dropuout by hyperparameter tuning

# Defining the model
%time
def create_model_three_hidden_layers(learning_rate,dropout_rate):
    model = Sequential()
    model.add(Dense(32,input_dim = 6,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(64,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(128,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))

    adam = Adam(lr = learning_rate)
    model.compile(loss = 'mse', optimizer = adam,metrics = ['mse', 'mae', 'mape'])
    return model

# Create the model

model = KerasRegressor(build_fn = create_model_three_hidden_layers,verbose = 0,
                       batch_size = 70,epochs = 300)

# Define the grid search parameters

learning_rate = [0.001,0.01,0.1]
dropout_rate = [0.0,0.1,0.2]

# Make a dictionary of the grid search parameters

param_grids = dict(learning_rate = learning_rate,dropout_rate = dropout_rate)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 0)
grid_result = grid.fit(x_train,y_train)
```

```
Wall time: 0 ns
```

In [64]:
```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : -0.007342452276498079, using {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.007342452276498079,0.0019959824416733475 with: {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.008561298809945583,0.001991463301998141 with: {'dropout_rate': 0.0, 'learning_rate': 0.01}
-0.42658794578164816,0.48784841583609684 with: {'dropout_rate': 0.0, 'learning_rate': 0.1}
-0.010010520182549953,0.0011587275738232285 with: {'dropout_rate': 0.1, 'learning_rate': 0.001}
-0.010904456581920385,0.0023298427190362866 with: {'dropout_rate': 0.1, 'learning_rate': 0.01}
-0.5191736936569213,0.36482949841031104 with: {'dropout_rate': 0.1, 'learning_rate': 0.1}
-0.011447767540812493,0.0033028640505105367 with: {'dropout_rate': 0.2, 'learning_rate': 0.001}
-0.017722824588418007,0.004725459057919966 with: {'dropout_rate': 0.2, 'learning_rate': 0.01}
-0.7296398758888245,0.28865086924640976 with: {'dropout_rate': 0.2, 'learning_rate': 0.1}
```

In [89]:

```
# Defining the model
#get best value for kernel initializer and activation func by hyperparameter tuning

%%time
def create_model_three_hidden_layers(activation_function,init):
    model = Sequential()
    model.add(Dense(32,input_dim = 6,kernel_initializer = init,activation = activation_function))

    model.add(Dense(32,kernel_initializer = init,activation = activation_function))

    model.add(Dense(64,kernel_initializer = init,activation = activation_function))

    model.add(Dense(128,kernel_initializer = init,activation = activation_function))

    model.add(Dense(1))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'mse',optimizer = adam,metrics = ['mse', 'mae', 'mape'])
    return model

# Create the model

model = KerasRegressor(build_fn = create_model_three_hidden_layers,verbose = 0,batch_size = 70,epochs = 300)

# Define the grid search parameters
activation_function = ['softmax','relu','tanh','linear']
init = ['uniform','normal','zero']

# Make a dictionary of the grid search parameters
param_grids = dict(activation_function = activation_function,init = init)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 0)
grid_result = grid.fit(x_train,y_train)
```

```
UsageError: Line magic function `%%time` not found.
```

In [66]:

```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : -0.007342452276498079, using {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.007342452276498079,0.0019959824416733475 with: {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.008561298809945583,0.001991463301998141 with: {'dropout_rate': 0.0, 'learning_rate': 0.01}
-0.42658794578164816,0.48784841583609684 with: {'dropout_rate': 0.0, 'learning_rate': 0.1}
-0.010010520182549953,0.0011587275738232285 with: {'dropout_rate': 0.1, 'learning_rate': 0.001}
-0.010904456581920385,0.0023298427190362866 with: {'dropout_rate': 0.1, 'learning_rate': 0.01}
-0.5191736936569213,0.36482949841031104 with: {'dropout_rate': 0.1, 'learning_rate': 0.1}
-0.011447767540812493,0.0033028640505105367 with: {'dropout_rate': 0.2, 'learning_rate': 0.001}
-0.017722824588418007,0.004725459057919966 with: {'dropout_rate': 0.2, 'learning_rate': 0.01}
-0.7296398758888245,0.28865086924640976 with: {'dropout_rate': 0.2, 'learning_rate': 0.1}
```

In [67]:
```
# Defining the model
#get best value for neuron by hyperparameter tuning
%%time
def create_model_three_hidden_layers(neuron1,neuron2,neuron3,neuron4):
    model = Sequential()
    model.add(Dense(neuron1,input_dim = 6,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dense(neuron2,input_dim = neuron1,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dense(neuron3,input_dim = neuron2,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dense(neuron4,input_dim = neuron3,kernel_initializer = 'uniform',activation = 'relu'))
    model.add(Dense(1))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'mse',optimizer = adam,metrics = ['mse', 'mae', 'mape'])
    return model

# Create the model

model = KerasRegressor(build_fn = create_model_three_hidden_layers,verbose = 0,batch_size = 70,epochs = 300)

# Define the grid search parameters

neuron1 = [8,16,32]
neuron2 = [32,64,128]
neuron3 = [32,64,128]
neuron4 = [32,64,128]

# Make a dictionary of the grid search parameters

param_grids = dict(neuron1 = neuron1,neuron2 = neuron2, neuron3 = neuron3, neuron4 = neuron4)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 0)
grid_result = grid.fit(x_train,y_train)
```

```
UsageError: Line magic function `%%time` not found.
```

In [68]:
```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : -0.007342452276498079, using {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.007342452276498079,0.0019959824416733475 with: {'dropout_rate': 0.0, 'learning_rate': 0.001}
-0.008561298809945583,0.001991463301998141 with: {'dropout_rate': 0.0, 'learning_rate': 0.01}
-0.42658794578164816,0.48784841583609684 with: {'dropout_rate': 0.0, 'learning_rate': 0.1}
-0.010010520182549953,0.0011587275738232285 with: {'dropout_rate': 0.1, 'learning_rate': 0.001}
-0.010904456581920385,0.0023298427190362866 with: {'dropout_rate': 0.1, 'learning_rate': 0.01}
-0.5191736936569213,0.36482949841031104 with: {'dropout_rate': 0.1, 'learning_rate': 0.1}
-0.011447767540812493,0.0033028640505105367 with: {'dropout_rate': 0.2, 'learning_rate': 0.001}
-0.017722824588418007,0.004725459057919966 with: {'dropout_rate': 0.2, 'learning_rate': 0.01}
-0.7296398758888245,0.28865086924640976 with: {'dropout_rate': 0.2, 'learning_rate': 0.1}
```

In [69]:
```
#create a model with 3 hidden layers with best hyperparameters
def create_model_three_hidden_layers():
    model = Sequential()
    model.add(Dense(8, input_dim=6, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(128, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(64, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(128, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(1))

    adam=Adam(lr=0.001)
    model.compile(loss='mse', optimizer=adam, metrics=['mse', 'mae', 'mape'])
    return model
```

In [70]:
```
%%time
epochs=300
batch_size=70

final_model=create_model_three_hidden_layers()

print("Here is the summary of our final model:")
final_model.summary()

with tf.device('/GPU:0'):
  final_model.fit(x_train,y_train, verbose = 0,batch_size = batch_size,epochs = epochs, shuffle=True)
```

```
Here is the summary of our final model:
Model: "sequential_94"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_469 (Dense)           (None, 8)                 56        
                                                                 
 dense_470 (Dense)           (None, 128)               1152      
                                                                 
 dense_471 (Dense)           (None, 64)                8256      
                                                                 
 dense_472 (Dense)           (None, 128)               8320      
                                                                 
 dense_473 (Dense)           (None, 1)                 129       
                                                                 
=================================================================
Total params: 17,913
Trainable params: 17,913
Non-trainable params: 0
_________________________________________________________________
Wall time: 12.4 s
```

In [77]:
```
loss, mae, mse, mape = final_model.evaluate(x_train, y_train)
print('\n', "Results for final model :", '\n', "Training Loss:", loss, '\n',
      "Training Mean Absolute Error:" , mae, '\n',"Training Mean Squared Error:", mse)
```

```
36/36 [==============================] - 0s 2ms/step - loss: 0.0059 - mse: 0.0059 - mae: 0.0552 - mape: 128.9680

 Results for final model :
 Training Loss: 0.005904133897274733
 Training Mean Absolute Error: 0.005904133897274733
 Training Mean Squared Error: 0.055208414793014526
```

In [78]:
```
loss_t, mae_t, mse_t, mape_t = final_model.evaluate(x_test, y_test)
print('\n', "Results for final model :", '\n', "Test Loss:", loss_t, '\n',
      "Test Mean Absolute Error:" , mae_t, '\n', "Test Mean Squared Error:", mse_t)
```

```
12/12 [==============================] - 0s 3ms/step - loss: 0.0072 - mse: 0.0072 - mae: 0.0613 - mape: 101.6804

 Results for final model :
 Test Loss: 0.007246457971632481
 Test Mean Absolute Error: 0.007246457971632481
 Test Mean Squared Error: 0.06132793426513672
```

Predicting values from Model using same dataset

In [85]:
```
from keras.wrappers.scikit_learn import KerasRegressor
```

In [87]:

```
# generating predictions for test data
y_predict_test = model1.predict(x_test)

# creating table with test price & predicted price for test
predictions_df = pd.DataFrame(x_test)
predictions_df['Actual'] = y_test
predictions_df['Predicted'] = y_predict_test
print(predictions_df.shape)
predictions_df.head(10)
```

```
12/12 [==============================] - 0s 2ms/step
(376, 8)
```

Out[87]:

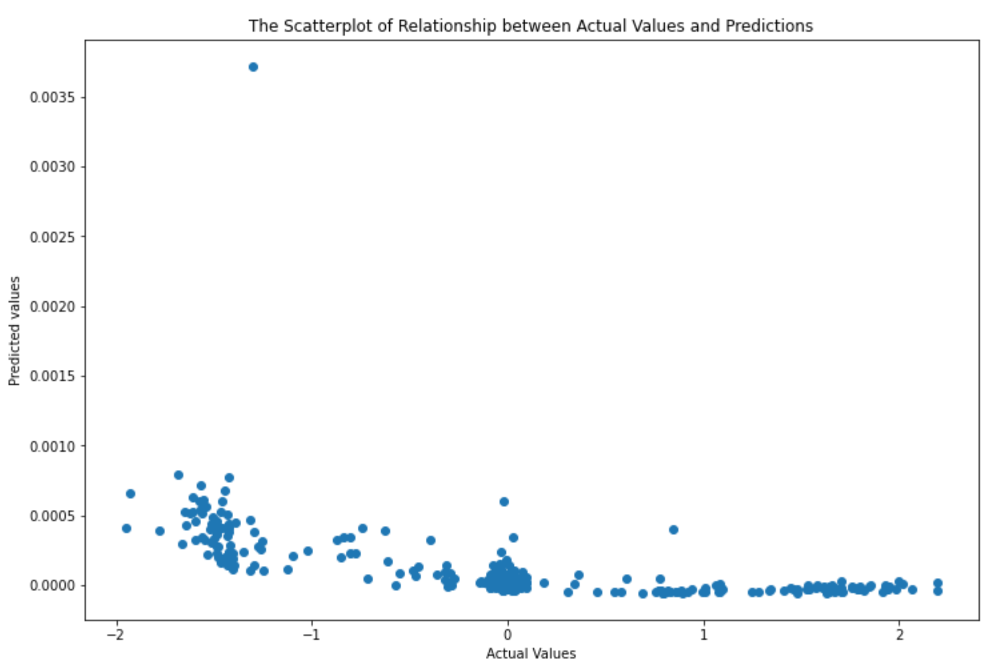
	AFDP	GTEP	TIT	TAT	CDP	CO	Actual	Predicted
2317	-0.066281	-1.644945	-2.146537	0.544795	-1.667984	1.906030	-1.931445	6.653878e-04
6485	-0.843512	-1.345066	-1.598640	0.589392	-1.416745	1.034617	-1.474914	4.127419e-04
9091	0.119701	-0.266125	0.178980	0.583021	-0.033574	-0.446970	-0.004782	-6.646042e-07
5443	0.368204	0.127002	0.501631	0.613602	0.076172	-0.364359	-0.028810	-1.562580e-05
10027	-1.553088	-1.524370	-1.793448	0.586844	-1.649844	0.790223	-1.484398	3.629955e-04
11794	0.426776	1.158241	0.866895	-0.979151	1.101986	-0.635490	0.800787	-4.597998e-05
10773	-1.538083	-1.344586	-1.556026	0.586844	-1.370489	0.211813	-1.402830	1.968982e-04
8915	-0.049171	-0.369920	0.227682	0.583021	-0.084366	-0.308247	0.098286	2.172192e-05
6588	0.166427	-0.263248	0.185068	0.571554	-0.091622	-0.183720	-0.035133	2.059035e-05
1384	0.707132	-0.252461	0.075489	0.594489	-0.251254	-0.339798	-0.095203	3.739018e-05

Visualizing the Relationship between the Actual and Predicted Values Model Validation

In [88]:

```
plt.figure(figsize=(12,8))
plt.xlabel("Actual Values")
plt.ylabel("Predicted values")
plt.title("The Scatterplot of Relationship between Actual Values and Predictions")
plt.scatter(predictions_df['Actual'], predictions_df['Predicted'])
```

Out[88]: <matplotlib.collections.PathCollection at 0x1417f9b7d00>



In [ ]:

```

```