BUISNESS REPORT – Linear Regression:

The data is taken from the comp-activ databases which is a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. The data was collected continuously every 5 seconds. Using the various system programs which are running in the background for every task being performed by the user, Predict the percentage portion of time (out of 100), that cpu runs in user mode, and how does each system program affect the same.

Tasks to be performed:

1. Load data and describe data

a. Import necessary libraries & packages

b. Load dataset

c. Check necessary details about data like shape, data types of the variable, missing values etc.

```
import pandas as pd
import numpy as np
import seaborn as sns
import seaborn as sb
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
import math
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

```
df = pd.read_csv('compactiv.csv')
df.head()
```

From the table we can see all the required variables that we require excel sheet

	0	1	2	3	4
lread	1	0	15	0	5
lwrite	0	0	3	0	1
scall	2147	170	2162	160	330
sread	79	18	159	12	39
swrite	68	21	119	16	38
fork	0.2	0.2	2.0	0.2	0.4
exec	0.2	0.2	2.4	0.2	0.4
rchar	40671.0	448.0	NaN	NaN	NaN
wchar	53995.0	8385.0	31950.0	8670.0	12185.0
pgout	0.0	0.0	0.0	0.0	0.0
ppgout	0.0	0.0	0.0	0.0	0.0
pgfree	0.0	0.0	0.0	0.0	0.0
pgscan	0.0	0.0	0.0	0.0	0.0
atch	0.0	0.0	1.2	0.0	0.0
pgin	1.6	0.0	6.0	0.2	1.0
ppgin	2.6	0.0	9.4	0.2	1.2
pflt	16.0	15.63	150.2	15.6	37.8
vflt	26.4	16.83	220.2	16.8	47.6
runqsz	CPU_Bound	Not_CPU_Bound	Not_CPU_Bound	Not_CPU_Bound	Not_CPU_Bound
freemem	4670	7278	702	7248	633
freeswap	1730946	1869002	1021237	1863704	1760253
usr	95	97	87	98	90

Check the necessary detail about data is given by its info, shape, data types:

```
df.shape
```

```
(8192, 22)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8192 entries, 0 to 8191
Data columns (total 23 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   lread                    8192 non-null   float64
 1   lwrite                   8192 non-null   float64
 2   scall                    8192 non-null   float64
 3   sread                    8192 non-null   float64
 4   swrite                   8192 non-null   float64
 5   fork                     8192 non-null   float64
 6   exec                     8192 non-null   float64
 7   rchar                    8192 non-null   float64
 8   wchar                    8192 non-null   float64
 9   pgout                    8192 non-null   float64
 10  ppgout                   8192 non-null   float64
 11  pgfree                   8192 non-null   float64
 12  pgscan                   8192 non-null   float64
 13  atch                     8192 non-null   float64
 14  pgin                     8192 non-null   float64
 15  ppgin                    8192 non-null   float64
 16  pflt                     8192 non-null   float64
 17  vflt                     8192 non-null   float64
 18  freemem                  8192 non-null   float64
 19  freeswap                 8192 non-null   float64
 20  usr                      8192 non-null   float64
 21  CPU_Bound_CPU_Bound      8192 non-null   uint8  
 22  CPU_Bound_Not_CPU_Bound  8192 non-null   uint8  
dtypes: float64(21), uint8(2)
memory usage: 1.4 MB
```

df.dtypes

```
lread          int64
lwrite         int64
scall          int64
sread          int64
swrite         int64
fork         float64
exec         float64
rchar        float64
wchar        float64
pgout        float64
ppgout       float64
pgfree       float64
pgscan       float64
atch         float64
pgin         float64
ppgin        float64
pflt         float64
vflt         float64
runqsz        object
freemem        int64
freeswap       int64
usr            int64
dtype: object
```

```
: df.isnull().sum()
```

```
: lread         0
  lwrite        0
  scall         0
  sread         0
  swrite        0
  fork          0
  exec          0
  rchar       104
  wchar        15
  pgout         0
  ppgout        0
  pgfree        0
  pgscan        0
  atch          0
  pgin          0
  ppgin         0
  pflt          0
  vflt          0
  runqsz        0
  freemem       0
  freeswap      0
  usr           0
  dtype: int64
```

From the above code and result we can figure out if there any null values, missing values, datatype of all the variables. Rchar has 104 missing values and wchar has 15 missing values.

2. Perform EDA and data cleaning

a. Generate the summary statistics for each of the variables and write comments on your observations

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
lread	8192.0	1.955969e+01	53.353799	0.0	2.0	7.0	20.000	1845.00
lwrite	8192.0	1.310620e+01	29.891726	0.0	0.0	1.0	10.000	575.00
scall	8192.0	2.306318e+03	1633.617322	109.0	1012.0	2051.5	3317.250	12493.00
sread	8192.0	2.104800e+02	198.980146	6.0	86.0	166.0	279.000	5318.00
swrite	8192.0	1.500582e+02	160.478980	7.0	63.0	117.0	185.000	5456.00
fork	8192.0	1.884554e+00	2.479493	0.0	0.4	0.8	2.200	20.12
exec	8192.0	2.791998e+00	5.212456	0.0	0.2	1.2	2.800	59.56
rchar	8088.0	1.973857e+05	239837.493526	278.0	34091.5	125473.5	267828.750	2526649.00
wchar	8177.0	9.590299e+04	140841.707911	1498.0	22916.0	46619.0	106101.000	1801623.00
pgout	8192.0	2.285317e+00	5.307038	0.0	0.0	0.0	2.400	81.44
ppgout	8192.0	5.977229e+00	15.214590	0.0	0.0	0.0	4.200	184.20
pgfree	8192.0	1.191971e+01	32.363520	0.0	0.0	0.0	5.000	523.00
pgscan	8192.0	2.152685e+01	71.141340	0.0	0.0	0.0	0.000	1237.00
atch	8192.0	1.127505e+00	5.708347	0.0	0.0	0.0	0.600	211.58
pgin	8192.0	8.277960e+00	13.874978	0.0	0.6	2.8	9.765	141.20
ppgin	8192.0	1.238859e+01	22.281318	0.0	0.6	3.8	13.800	292.61
pflt	8192.0	1.097938e+02	114.419221	0.0	25.0	63.8	159.600	899.80
vflt	8192.0	1.853158e+02	191.000603	0.2	45.4	120.4	251.800	1365.00
freemem	8192.0	1.763456e+03	2482.104511	55.0	231.0	579.0	2002.250	12027.00
freeswap	8192.0	1.328126e+06	422019.426957	2.0	1042623.5	1289289.5	1730379.500	2243187.00
usr	8192.0	8.396887e+01	18.401905	0.0	81.0	89.0	94.000	99.00

We can see from the above summary statistics the mean of - Number of characters transferred per second by system write calls is the maximum. Most of the variables have minimum values from 0. The maximum Number of characters transferred per second by system read calls is 2526649 which is the highest frequency reached among all other variables.

b. Working with Null/Missing values

i. Check for Missing values and perform the necessary steps for data imputation and provide reasoning for your approach.

As we saw in the previous question there are missing values that must be replaced. Since both are numerical in nature we must replace them by the median.

```
df['rchar'].fillna(df.rchar.median(), inplace = True)
df
```

	lread	lwrite	scall	sread	swrite	fork	exec	rchar	wchar	pgout	...	pgscan	atch	pgin	ppgin	pflt	vflt	runqsz	freemem	freesw
0	1	0	2147	79	68	0.2	0.20	40671.0	53995.0	0.00	...	0.00	0.0	1.60	2.60	16.00	26.40	CPU_Bound	4670	1730
1	0	0	170	18	21	0.2	0.20	448.0	8385.0	0.00	...	0.00	0.0	0.00	0.00	15.63	16.83	Not_CPU_Bound	7278	1869
2	15	3	2162	159	119	2.0	2.40	125473.5	31950.0	0.00	...	0.00	1.2	6.00	9.40	150.20	220.20	Not_CPU_Bound	702	1021
3	0	0	160	12	16	0.2	0.20	125473.5	8670.0	0.00	...	0.00	0.0	0.20	0.20	15.60	16.80	Not_CPU_Bound	7248	1863
4	5	1	330	39	38	0.4	0.40	125473.5	12185.0	0.00	...	0.00	0.0	1.00	1.20	37.80	47.60	Not_CPU_Bound	633	1760
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8187	16	12	3009	360	244	1.6	5.81	405250.0	85282.0	8.02	...	55.11	0.6	35.87	47.90	139.28	270.74	CPU_Bound	387	986
8188	4	0	1596	170	146	2.4	1.80	89489.0	41764.0	3.80	...	0.20	0.8	3.80	4.40	122.40	212.60	Not_CPU_Bound	263	1055
8189	16	5	3116	289	190	0.6	0.60	325948.0	52640.0	0.40	...	0.00	0.4	28.40	45.20	60.20	219.80	Not_CPU_Bound	400	969
8190	32	45	5180	254	179	1.2	1.20	62571.0	29505.0	1.40	...	18.04	0.4	23.05	24.25	93.19	202.81	CPU_Bound	141	1022
8191	2	0	985	55	46	1.6	4.80	111111.0	22256.0	0.00	...	0.00	0.2	3.40	6.20	91.80	110.00	CPU_Bound	659	1756

8192 rows × 22 columns

```python
df['wchar'].fillna(df.wchar.median(), inplace = True)
df
```

	lread	lwrite	scall	sread	swrite	fork	exec	rchar	wchar	pgout	...	pgscan	atch	pgin	ppgin	pflt	vflt	runqsz	freemem	freeswa
0	1	0	2147	79	68	0.2	0.20	40671.0	53995.0	0.00	...	0.00	0.0	1.60	2.60	16.00	26.40	CPU_Bound	4670	173094
1	0	0	170	18	21	0.2	0.20	448.0	8385.0	0.00	...	0.00	0.0	0.00	0.00	15.63	16.83	Not_CPU_Bound	7278	18690
2	15	3	2162	159	119	2.0	2.40	125473.5	31950.0	0.00	...	0.00	1.2	6.00	9.40	150.20	220.20	Not_CPU_Bound	702	10212
3	0	0	160	12	16	0.2	0.20	125473.5	8670.0	0.00	...	0.00	0.0	0.20	0.20	15.60	16.80	Not_CPU_Bound	7248	18637
4	5	1	330	39	38	0.4	0.40	125473.5	12185.0	0.00	...	0.00	0.0	1.00	1.20	37.80	47.60	Not_CPU_Bound	633	17602
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8187	16	12	3009	360	244	1.6	5.81	405250.0	85282.0	8.02	...	55.11	0.6	35.87	47.90	139.28	270.74	CPU_Bound	387	98664
8188	4	0	1596	170	146	2.4	1.80	89489.0	41764.0	3.80	...	0.20	0.8	3.80	4.40	122.40	212.60	Not_CPU_Bound	263	105574
8189	16	5	3116	289	190	0.6	0.60	325948.0	52640.0	0.40	...	0.00	0.4	28.40	45.20	60.20	219.80	Not_CPU_Bound	400	9691
8190	32	45	5180	254	179	1.2	1.20	62571.0	29505.0	1.40	...	18.04	0.4	23.05	24.25	93.19	202.81	CPU_Bound	141	10224
8191	2	0	985	55	46	1.6	4.80	111111.0	22256.0	0.00	...	0.00	0.2	3.40	6.20	91.80	110.00	CPU_Bound	659	175651

192 rows × 22 columns

ii. Check for the Zero values and understand the importance of that data point. Please provide your comments on if we need to change them (impute) or drop them.

```python
df.isin([0]).sum()
```

```
lread        675
lwrite      2684
scall          0
sread          0
swrite         0
fork          21
exec          21
rchar          0
wchar          0
pgout       4878
ppgout      4878
pgfree      4869
pgscan      6448
atch        4575
pgin        1220
ppgin       1220
pflt           3
vflt           0
runqsz         0
freemem        0
freeswap       0
usr          283
dtype: int64
```

We can see from the above result we can see how many of the variables have the 0 values.

c. Working with Outliers

i. Check for outliers and provide comments

Check Outliers in the HTML file.

On performing our code we can see that almost every variable has an outlier(refer from HTML file.)

'lread', 'sread', 'swrite', rchar, wchar,exec, atch 'pgout', 'ppgout', 'pgfree', 'pgin', 'ppgin', 'pflt', 'vflt', freemem, freeswap are all right skwed and have the mode is often less than the median, which is less than the mean.

Pg scan has no distribution hence no skewness determined.

Freeswap and usr have a left skewed.

ii. Perform outlier treatment (only if required)

```
def remove_outlier(column):
sorted(column)
   q1=df[column].quantile(0.25)
   q3=df[column].quantile(0.75)
   iqr=q3-q1
   lower=q1-1.5*iqr
   upper=q3+1.5*iqr
   return lower,upper
or i in nums:
   lower,upper=remove_outlier(i)
   df[i]=np.where(df[i]>upper,upper,df[i])
   df[i]=np.where(df[i]<lower,lower,df[i])
for i in nums:
   sns.boxplot(df[i],showmeans=True)
   plt.show()
```

We create a function to remove the outlier and use quartiles to treat the inconsistency if any. From the above code we can treat the outlier.(Can see the corrected outlier treatment in the HTML File)

d. Scaling the data

i. Only if scaling is necessary, please perform the same and provide your reasoning

```
scaler = MinMaxScaler()

columns=['lread','lwrite','scall','sread','swrite','fork','exec','rchar','wchar','pgout','ppgout','pgscan','pg
free','atch','pgin','ppgin','pflt','vflt','freemem','freeswap','usr']


df_scaled = scaler.fit_transform(df[columns].to_numpy())

df_scaled = pd.DataFrame(df_scaled,columns=columns)

print("Scaled Dataset Using MinMaxScaler")

df_scaled['runqsz']=df['runqsz']

df_scaled.head().T
```

	0	1	2	3	4
lread	0.021277	0.0	0.319149	0.0	0.106383
lwrite	0.0	0.0	0.12	0.0	0.04
scall	0.305725	0.009151	0.307925	0.007651	0.033153
sread	0.129778	0.021333	0.272	0.010667	0.058667
swrite	0.168975	0.038781	0.310249	0.024931	0.085873
fork	0.040816	0.040816	0.408163	0.040816	0.081633
exec	0.029851	0.029851	0.358209	0.029851	0.059701
rchar	0.066119	0.000278	0.20493	0.20493	0.20493
wchar	0.226117	0.030057	0.132904	0.031301	0.049642
pgout	0.0	0.0	0.0	0.0	0.0
ppgout	0.0	0.0	0.0	0.0	0.0
pgscan	0.0	0.0	0.0	0.0	0.0
pgfree	0.0	0.0	0.0	0.0	0.0
atch	0.0	0.0	0.8	0.0	0.0
pgin	0.098049	0.0	0.255183	0.008506	0.042531
ppgin	0.077381	0.0	0.279762	0.005952	0.035714
pflt	0.04426	0.043237	0.415491	0.043154	0.104564
vflt	0.049686	0.029633	0.392017	0.029579	0.084492
freemem	1.0	1.0	0.140526	1.0	0.12554
freeswap	0.770522	0.832369	0.45258	0.829666	0.783651
usr	0.883333	0.946667	0.68	0.973333	0.76
runqsz	CPU_Bound	Not_CPU_Bound	Not_CPU_Bound	Not_CPU_Bound	Not_CPU_Bound

In this method, we convert variables with different scales of measurements into a single scale. First we check what are the values that needs to be changed. Standard Scaler normalizes the data using the formula (x-mean)/standard deviation.

e. Working with Categorical variables

i. Identify the categorical data given as part of the data set?

```
df_scaled.runqsz.unique()
```

```
array(['CPU_Bound', 'Not_CPU_Bound'], dtype=object)
```

```
df= pd.get_dummies(df, prefix='CPU_Bound', columns=['runqsz'])
```

```
df.head()
```

lread	lwrite	scall	sread	swrite	fork	exec	rchar	wchar	pgout	...	pgscan	atch	pgin	ppgin	pflt	vflt	runqsz	freemem	freeswap	us
1	0	2147	79	68	0.2	0.2	40671.0	53995.0	0.0	...	0.0	0.0	1.6	2.6	16.00	26.40	CPU_Bound	4670	1730946	95
0	0	170	18	21	0.2	0.2	448.0	8385.0	0.0	...	0.0	0.0	0.0	0.0	15.63	16.83	Not_CPU_Bound	7278	1869002	97
15	3	2162	159	119	2.0	2.4	125473.5	31950.0	0.0	...	0.0	1.2	6.0	9.4	150.20	220.20	Not_CPU_Bound	702	1021237	87
0	0	160	12	16	0.2	0.2	125473.5	8670.0	0.0	...	0.0	0.0	0.2	0.2	15.60	16.80	Not_CPU_Bound	7248	1863704	98
5	1	330	39	38	0.4	0.4	125473.5	12185.0	0.0	...	0.0	0.0	1.0	1.2	37.80	47.60	Not_CPU_Bound	633	1760253	90

On performing the given code we can see that , runqsz has the categorical values that needs to be changed. Hence perform get dummies function to replace these values with one and 0.(Given in the HTML file)

ii.) Perform encoding and provide detailed comments and reasoning for the encoding approach.

Most of the machine learning models are designed to work on numeric data. Hence, we need to convert categorical text data into numerical data for model building

One-Hot-Encoding is used to create dummy variables to replace the categories in a categorical variable into features of each category and represent it using 1 or 0 based on the presence or absence of the categorical value in the record

3. Perform univariate, bivariate & Multivariate analysis

a. Perform Univariate analysis for each variable and write comments

We have performed univariate analysis for each variable(in the HTML file.) For lread its is positively skewed and greater frequency. For lwrite its similar to lread but its less frequent. For all of the graphs itself we can see an unique peak in the beginning and then a drop which is constant and then a sudden rise as well. For pg scan we can see there is no data to be distributed.

b. Perform bivariate analysis and make necessary inference about the relation between the variables.

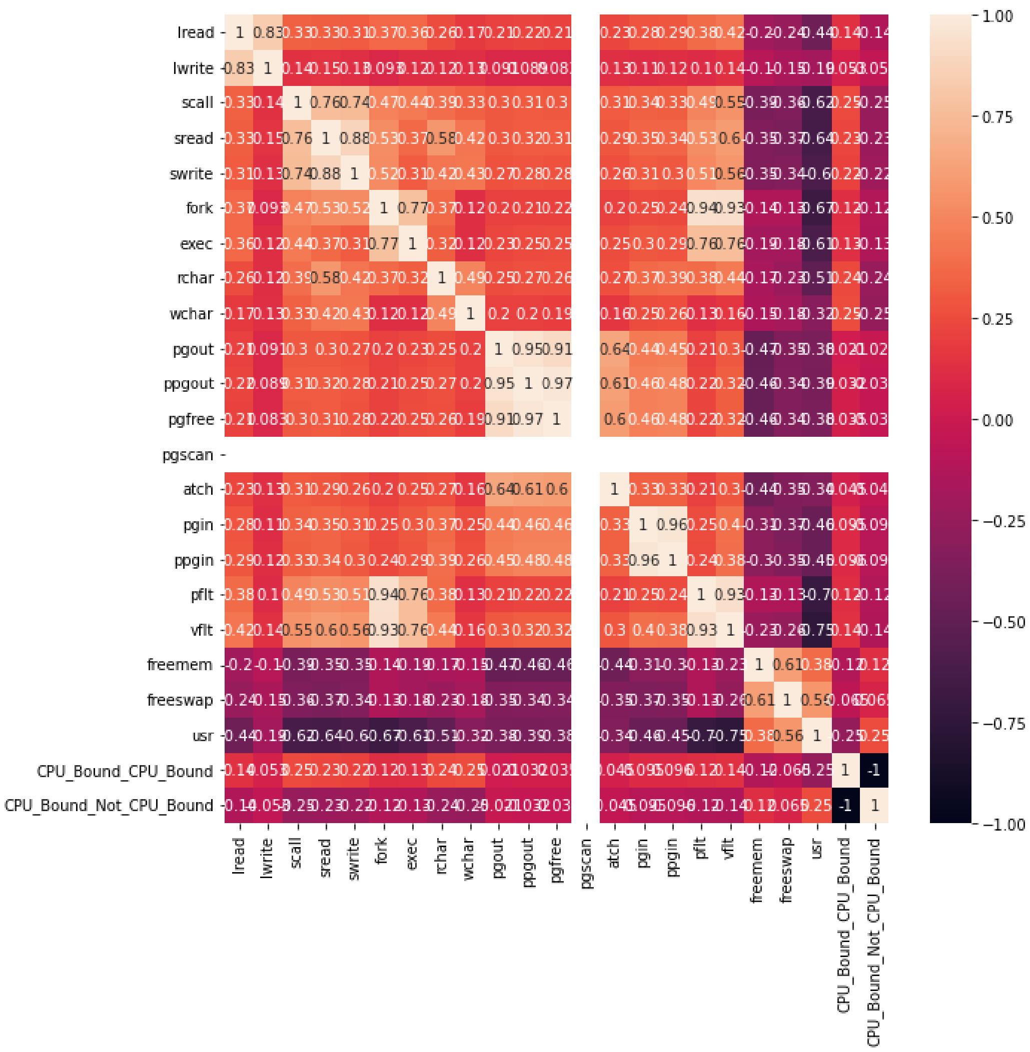
Refer html file for the plot.

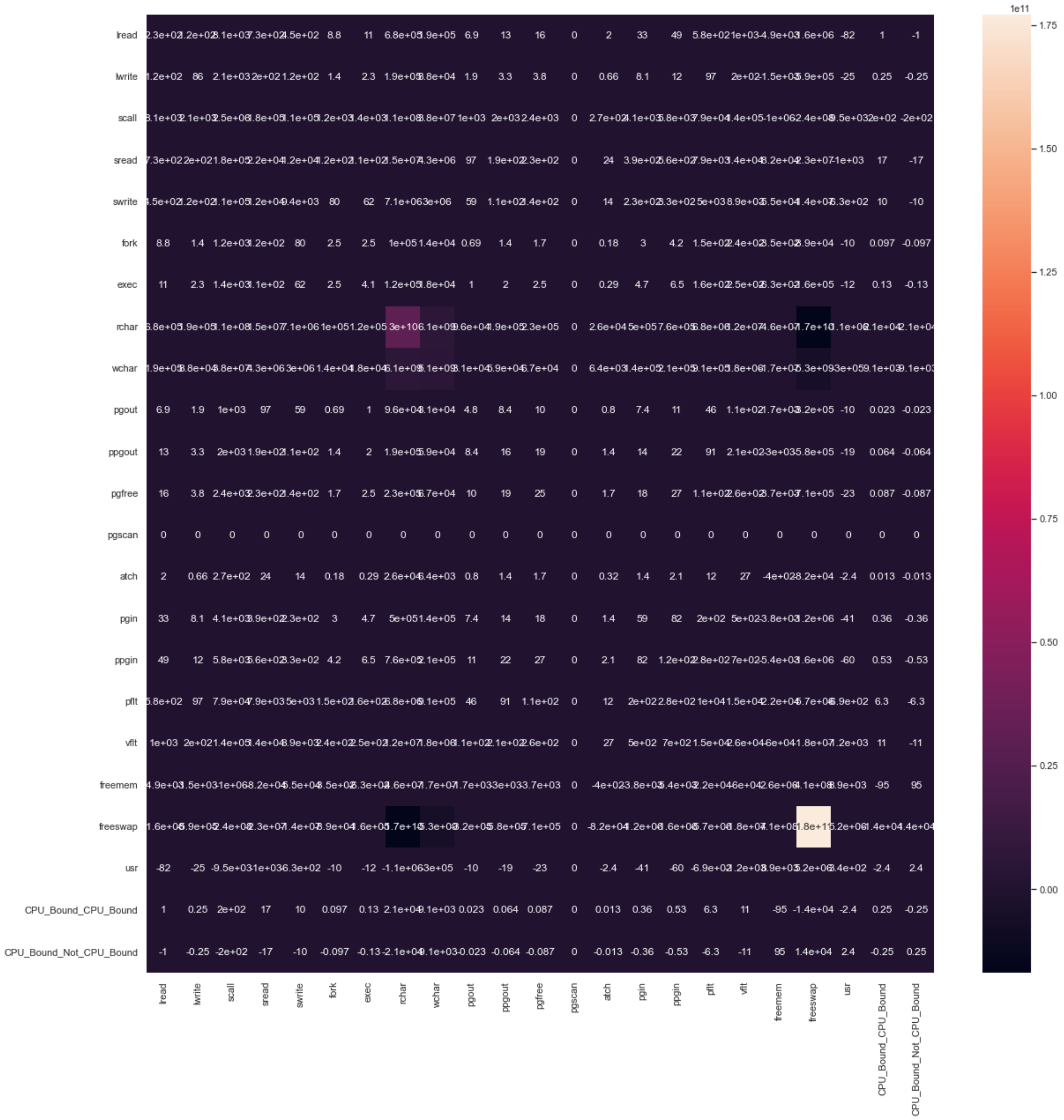
```
sns.pairplot(df_scaled, diag_kind='kde',size = 3)
plt.show()
```

c. Perform Multivariate analysis and make necessary inferences about the relation between variables.

d. Check Covariance and Correlation and identify positively and negatively correlated variables.

```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True)
```





e. Identify the variables which has multicollinearity. Check for multi collinearity and drop the variables.

]:
```
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]
cols_to_remove=list(vif_data.loc[vif_data['VIF']>=5]['feature'])
print(cols_to_remove)

vif_data.sort_values('VIF', ascending=False)
print(vif_data)
```

```
['lread', 'sread', 'swrite', 'fork', 'pgout', 'ppgout', 'pgfree', 'pgin', 'ppgin', 'pflt', 'vflt', 'CPU_Bound_CPU_Bound', 'CPU_
Bound_Not_CPU_Bound']
                    feature        VIF
0                     lread   5.232535
1                    lwrite   4.260737
2                     scall   2.989088
3                     sread   6.517014
4                    swrite   5.639798
5                      fork  12.947438
6                      exec   3.137525
7                     rchar   2.087446
8                     wchar   1.605667
9                     pgout  11.474544
10                   ppgout  30.685098
11                   pgfree  17.175122
12                   pgscan        NaN
13                     atch   1.859658
14                     pgin  13.733882
15                    ppgin  13.962094
16                     pflt  11.536918
17                     vflt  15.204685
18                  freemem   1.969949
19                 freeswap   1.825789
20      CPU_Bound_CPU_Bound  13.213383
21  CPU_Bound_Not_CPU_Bound  13.225624
```

:
```
new_df=X.drop(['lread', 'sread', 'swrite', 'fork', 'pgout', 'ppgout', 'pgfree', 'pgin', 'ppgin', 'pflt', 'vflt', 'CPU_Bound_CPU_E
new_df
```

:

	lwrite	scall	exec	rchar	wchar	pgscan	atch	freemem	freeswap
0	0.0	2147.0	0.20	40671.0	53995.0	0.0	0.0	4659.125	1730946.0
1	0.0	170.0	0.20	448.0	8385.0	0.0	0.0	4659.125	1869002.0
2	3.0	2162.0	2.40	125473.5	31950.0	0.0	1.2	702.000	1021237.0
3	0.0	160.0	0.20	125473.5	8670.0	0.0	0.0	4659.125	1863704.0
4	1.0	330.0	0.40	125473.5	12185.0	0.0	0.0	633.000	1760253.0
...	...	...	...	...	...	...	...	...	...
8187	12.0	3009.0	5.81	405250.0	85282.0	0.0	0.6	387.000	986647.0
8188	0.0	1596.0	1.80	89489.0	41764.0	0.0	0.8	263.000	1055742.0
8189	5.0	3116.0	0.60	325948.0	52640.0	0.0	0.4	400.000	969106.0
8190	25.0	5180.0	1.20	62571.0	29505.0	0.0	0.4	141.000	1022458.0
8191	0.0	985.0	4.80	111111.0	22256.0	0.0	0.2	659.000	1756514.0

8192 rows × 9 columns

First we check for multicollinearity and remove all the features that are multicollinear using variance inflation factor. Mathematically, the VIF for a regression model variable is equal to the ratio of the overall model variance to the variance of a model that includes only that single independent variable.

4. Building a Linear Regression Model

a. Prepare data for model building

Preparing the data for model building, we drop the values of usr and copy it in another dataset so that we can use this data frame to train and test our model.

X = df.drop('usr', axis=1)

y = df[['usr']]

We get the table that we need to test and train.(refer to HTML file.)

b. Build linear regression model.

```
]: X = new_df
   y = df[['usr']]
```

```
]: from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25 , random_state=1)
```

```
]: model.predict(X_train)
```

```
]: 5745     68.397776
   1930     53.316204
   5622     60.481046
   5340    102.002734
   2255     98.826406
              ...
   7935     74.811305
   5192     96.452824
   3980     69.561082
   235      70.027195
   5157     94.588283
   Length: 6144, dtype: float64
```

```
]: regression_model = LinearRegression()
   regression_model.fit(X_train, y_train)
```

Following all the required codes to achieve the model to further find the required measures to check accuracy of our model.

c. Find the features that add value to the model. Identify the list of Variables which highly impact the prediction based on the correlation Metrix given for regression (target variable)

```
vif = [variance_inflation_factor(X.values, ix) for ix in range(X.shape[1])]
```

```
i=0
for column in X.columns:
    if i < 15:
        print (column ,"--->",  vif[i])
        i = i+1
```

```
lread ---> 5.232534882470838
lwrite ---> 4.260737491684919
scall ---> 2.9890877339622937
sread ---> 6.517014292159949
swrite ---> 5.639797517443181
fork ---> 12.947438478657318
exec ---> 3.1375249704246078
rchar ---> 2.087445737239059
wchar ---> 1.6056667145387191
pgout ---> 11.474543882811679
ppgout ---> 30.685097541787272
pgfree ---> 17.17512180642015
pgscan ---> nan
atch ---> 1.8596581052295806
pgin ---> 13.733882306297474
```

Linear Regression Model

Again we can find the best correlated values that suit for our model. First we check for multicollinearity and remove all the features that are multicollinear using variance inflation factor. Mathematically, the VIF for a regression model variable is equal to the ratio of the overall model variance to the variance of a model that includes only that single independent variable.

d. Print test and train results with all variables and best fit line.

```
sns.set(rc = {'figure.figsize':(7,7)})
ax=sns.regplot(x=y_test,y=y_pred,ci=None,color ='blue');
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```

Heavily clustered but in a consistent manner, but the growth seems to be very consistent.

5. Model Performance :

a. Check for the performance measures for linear regression (Hint: RMSE, R square, etc.)

```
# Let us check the intercept for the model
intercept = regression_model.intercept_[0]

print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is 52.703408616461346
```

```
# R square on training data
regression_model.score(X_train, y_train)
```

```
0.5513661898690928
```

```
#R square on testing data
regression_model.score(X_test, y_test)
```

```
0.5447312405803018
```

```
#RMSE on training data
predicted_train=regression_model.fit(X_train, y_train).predict(X_train)
np.sqrt(mean_squared_error(y_train,predicted_train))
```

```
12.16989876561295
```

```
##RMSE on testing data
predicted_test=regression_model.fit(X_train, y_train).predict(X_test)
np.sqrt(mean_squared_error(y_test,predicted_test))
```

```
12.870554763335232
```

```
# concatenate X and y into a single dataframe
data_train = pd.concat([X_train, y_train], axis=1)
data_test=pd.concat([X_test,y_test],axis=1)
data_train.head()
```

	lwrite	scall	exec	rchar	wchar	pgscan	atch	freemem	freeswap	usr
5745	10.0	2007.0	0.60	32665.0	49643.0	0.0	0.6	329.0	989029.0	94
1930	4.0	837.0	0.80	6255.0	23670.0	0.0	0.0	3052.0	1013758.0	96
5622	2.0	2227.0	6.70	108370.0	28568.0	0.0	0.2	314.0	1108418.0	77
5340	0.0	2132.0	0.20	90813.0	28590.0	0.0	0.0	488.0	1742493.0	93
2255	0.0	3517.0	2.99	310439.0	214462.0	0.0	0.2	420.0	1547657.0	86

From above relations we can see that my RMSE value is 12.16 for test and train almost. For R square the value is 0.544.

b. Experiment with data transformation and suggest if we can improve the model performance.

```
100 - (rmse/y_train.mean())*100
```

```
usr    85.523778
dtype: float64
```

To improve the performance of this model we can decrease the RMSE values and increase the R*2 mean value.