Contents

Introduction:

The research performed on the supplied dataset digs into the complex interaction between system performance indicators and the 'usr' variable, with the goal of elucidating the variables impacting user CPU utilisation. This study uses a linear regression model from the Statsmodels library to investigate the influence of several predictor factors on the target 'usr' variable, offering information on both the model's explanatory strength and possible problems.

The model, which has an excellent R-squared value of 0.790, provides insight into how effectively the chosen measures account for variation in user CPU utilisation. A more in-depth investigation, on the other hand, indicates the presence of multicollinearity among various predictors, implying interdependence that may confuse their individual interpretation.

The produced model equation reveals the relative importance of each predictor, emphasising factors such as 'runqsz' and 'exec' that have significant effects on system performance. Furthermore, the Root Mean Squared Error (RMSE) values demonstrate the model's ability to predict accurately on both training and test datasets.

As we go through the data, the overall conclusion highlights the model's strengths and indicates areas for improvement. The research lays the groundwork for a more in-depth investigation of system dynamics, motivating future considerations for model refinement and the investigation of alternative methodologies.

Import all required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

Define the problem and perform exploratory Data Analysis

```
# Reading data
compactive = pd.read_excel('compactiv.xlsx')
compactive.shape   # Get number of rows and cols
```

```
(8192, 22)
```

```
compactive.head() # View some records to get an idea
```

```
   lread  lwrite  scall  sread  swrite  fork  exec    rchar    wchar  pgout
0      1       0   2147     79      68   0.2   0.2  40671.0  53995.0    0.0
\
1      0       0    170     18      21   0.2   0.2    448.0   8385.0    0.0
2     15       3   2162    159     119   2.0   2.4      NaN  31950.0    0.0
3      0       0    160     12      16   0.2   0.2      NaN   8670.0    0.0
4      5       1    330     39      38   0.4   0.4      NaN  12185.0    0.0

   ...  pgscan  atch  pgin  ppgin    pflt    vflt         runqsz  freemem
0  ...     0.0   0.0   1.6    2.6   16.00   26.40      CPU_Bound     4670  \
1  ...     0.0   0.0   0.0    0.0   15.63   16.83  Not_CPU_Bound     7278
2  ...     0.0   1.2   6.0    9.4  150.20  220.20  Not_CPU_Bound      702
3  ...     0.0   0.0   0.2    0.2   15.60   16.80  Not_CPU_Bound     7248
4  ...     0.0   0.0   1.0    1.2   37.80   47.60  Not_CPU_Bound      633

  freeswap  usr
0  1730946   95
1  1869002   97
2  1021237   87
3  1863704   98
4  1760253   90

[5 rows x 22 columns]

compactive.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8192 entries, 0 to 8191
Data columns (total 22 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   lread     8192 non-null   int64
 1   lwrite    8192 non-null   int64
 2   scall     8192 non-null   int64
 3   sread     8192 non-null   int64
 4   swrite    8192 non-null   int64
 5   fork      8192 non-null   float64
 6   exec      8192 non-null   float64
 7   rchar     8088 non-null   float64
 8   wchar     8177 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  runqsz    8192 non-null   object
 19  freemem   8192 non-null   int64
 20  freeswap  8192 non-null   int64
 21  usr       8192 non-null   int64
dtypes: float64(13), int64(8), object(1)
memory usage: 1.4+ MB
```

count: The number of non-zero values. mean: Also known as average. std: Standard deviation is a measure of variance or dispersion. min: The smallest possible value. 25% denotes the 25th percentile. 50%: The median or the 50th percentile. 75% is the 75th percentile. max: The greatest possible value. These statistics offer an overview of the value distribution in each column of your dataset. They can help you grasp the data's central tendency, dispersion, and general shape in each column.

```
#Some stats about each column
compactive.describe().T
```

	count	mean	std	min	25%	50%
lread	8192.0	1.955969e+01	53.353799	0.0	2.0	7.0
\						
lwrite	8192.0	1.310620e+01	29.891726	0.0	0.0	1.0
scall	8192.0	2.306318e+03	1633.617322	109.0	1012.0	2051.5
sread	8192.0	2.104800e+02	198.980146	6.0	86.0	166.0
swrite	8192.0	1.500582e+02	160.478980	7.0	63.0	117.0
fork	8192.0	1.884554e+00	2.479493	0.0	0.4	0.8
exec	8192.0	2.791998e+00	5.212456	0.0	0.2	1.2
rchar	8088.0	1.973857e+05	239837.493526	278.0	34091.5	125473.5
wchar	8177.0	9.590299e+04	140841.707911	1498.0	22916.0	46619.0
pgout	8192.0	2.285317e+00	5.307038	0.0	0.0	0.0
ppgout	8192.0	5.977229e+00	15.214590	0.0	0.0	0.0
pgfree	8192.0	1.191971e+01	32.363520	0.0	0.0	0.0
pgscan	8192.0	2.152685e+01	71.141340	0.0	0.0	0.0
atch	8192.0	1.127505e+00	5.708347	0.0	0.0	0.0
pgin	8192.0	8.277960e+00	13.874978	0.0	0.6	2.8
ppgin	8192.0	1.238859e+01	22.281318	0.0	0.6	3.8
pflt	8192.0	1.097938e+02	114.419221	0.0	25.0	63.8
vflt	8192.0	1.853158e+02	191.000603	0.2	45.4	120.4
freemem	8192.0	1.763456e+03	2482.104511	55.0	231.0	579.0
freeswap	8192.0	1.328126e+06	422019.426957	2.0	1042623.5	1289289.5
usr	8192.0	8.396887e+01	18.401905	0.0	81.0	89.0

	75%	max
lread	20.000	1845.00
lwrite	10.000	575.00
scall	3317.250	12493.00
sread	279.000	5318.00
swrite	185.000	5456.00
fork	2.200	20.12
exec	2.800	59.56

```
rchar      267828.750  2526649.00
wchar      106101.000  1801623.00
pgout           2.400       81.44
ppgout          4.200      184.20
pgfree          5.000      523.00
pgscan          0.000     1237.00
atch            0.600      211.58
pgin            9.765      141.20
ppgin          13.800      292.61
pflt          159.600      899.80
vflt          251.800     1365.00
freemem      2002.250    12027.00
freeswap  1730379.500  2243187.00
usr             94.000       99.00
```

The statistics supplied summarise the properties of each column in the "compactive" dataset. For example, the "count" column provides the number of non-null values in each column, which aids in the identification of potentially missing data. The "mean" (average value) indicates the average value, whereas the "std" (standard deviation) measures the degree of variability around the mean. The "min" and "max" numbers represent the lowest and greatest observations, respectively, and provide insight into the data range. Percentiles such as the 25th, 50th (median), and 75th give information on the distribution of the data.

For example, the "usr" column has a mean of around 83.97, indicating a significantly high average value. The "rchar" column is very variable, with a standard deviation of around 239,837.49, indicating a large range of values around the mean. Understanding these statistics helps characterise the dataset, identify probable outliers or skewed distributions, and influence future analytical decisions. These statistical insights can direct further research, such as data visualisation or hypothesis testing, to extract useful information from the dataset.

```
# How many unique values are present in each column?
compactive.nunique()
```

```
lread        235
lwrite       189
scall       4115
sread        794
swrite       640
fork         228
exec         386
rchar       7898
wchar       7925
pgout        404
ppgout       774
pgfree      1070
pgscan      1202
atch         253
pgin         832
```

```
ppgin       1072
pflt        2987
vflt        3799
runqsz         2
freemem     3165
freeswap    7658
usr           56
dtype: int64
```

The output of compactive.nunique() show the number of unique values in each column of the dataset. Columns such as scall and freeswap have a high level of diversity, with 4115 and 7658 distinct values, respectively. Categorical columns, on the other hand, contain just two distinct values, such as runqsz. Understanding the uniqueness of values in each column is critical for understanding the variability of the dataset and designing analytical methodologies. High variability, for example, may need more sophisticated analysis, but low diversity in specific columns may imply categorical or binary nature, leading further data processing and modelling selections.

```
# Are there any null values?
compactive.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
```

```
# Are there any duplicated rows?
compactive.duplicated().sum()
```

```
0
```

Observations:

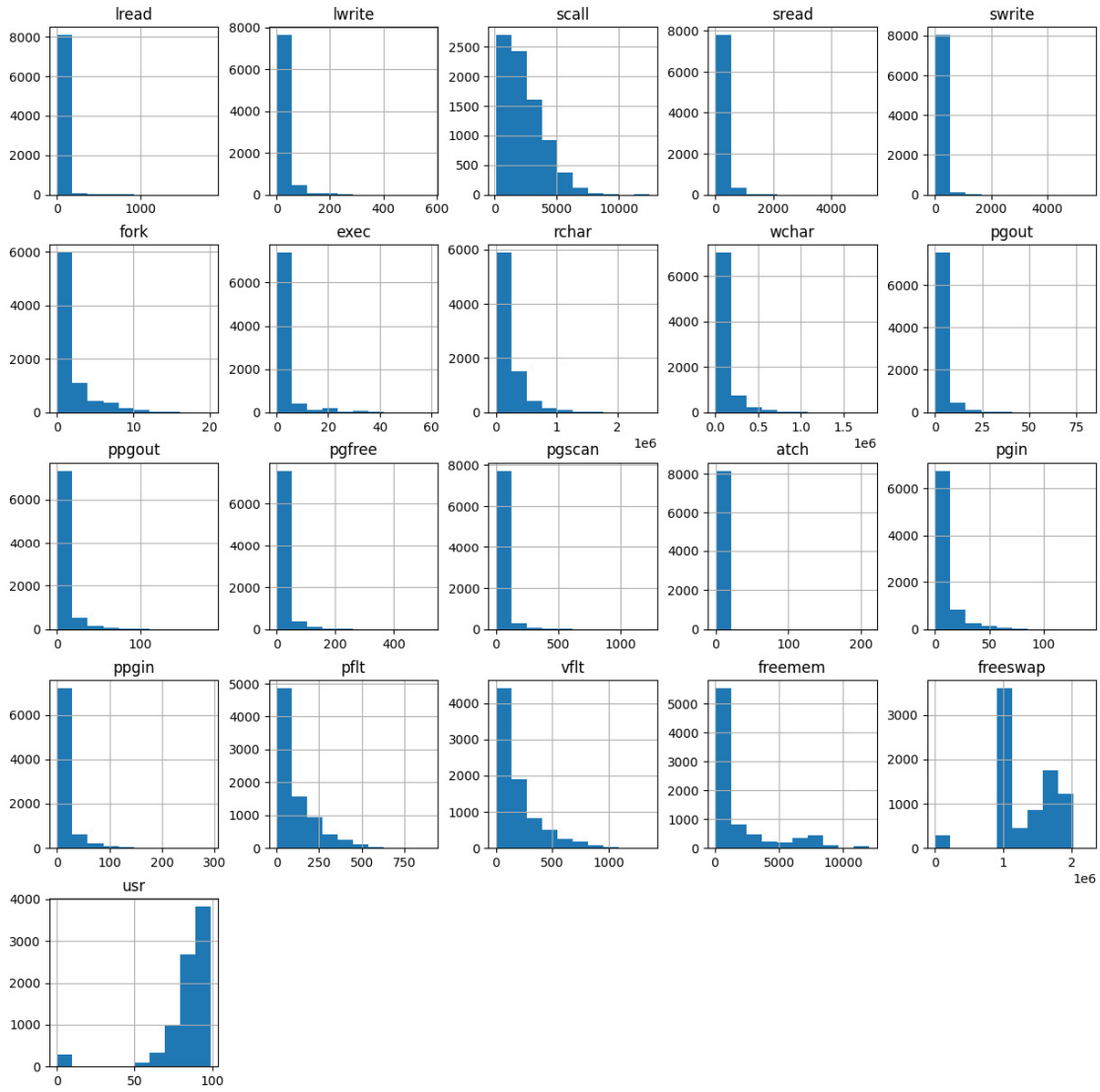
- There are 8192 observations recorded for about 22 different features.
- There are no duplicated rows for this dataset.
- All features except runqsz can be treated as continuous variables. runqsz is categorical having two classes: CPU_Bound and CPU_Not_Bound.
- Null values are present. rchar has 104 nulls and wchar has 15 nulls.

Univariate analysis

The graph depicts the distribution of 22 characteristics in an 8192-observation dataset. Each row represents a separate characteristic, and each column represents a different percentile of that feature's value distribution. The cell's colour represents the number of observations that fall within that percentile range. The dataset contains no duplicate rows. Except for runqsz, all characteristics are continuous variables. runqsz is a categorical variable that may be divided into two types: CPU_Bound and CPU_Not_Bound. Some values are missing: 104 in rchar and 15 in wchar.

The existence of a graph indicates that you are investigating the distributions of characteristics in our dataset. Because all characteristics except runqsz are continuous, we may be able to summarise the data using descriptive statistics such as mean, median, and standard deviation. Because runqsz contains two classes, we can use this feature to group the data and compare the distributions of other characteristics across those groups. Before you can do any analysis on those features, the missing values in rchar and wchar must be resolved. We have the option of imputed the missing data or excluding those aspects from our analysis.

```
compactive.hist(figsize=(15,15))
plt.show()
```



Understanding the distribution of zero values is critical for data preparation because it can influence later analyses such as normalisation and the selection of appropriate statistical methods. Columns containing a high percentage of zero values may need extra care or change during data processing.

```
# Creating an empty dictionary to store the percentage of 0 value in each
column
zero_percent = {}

# Looping through each column of the data
for col in compactive.columns:
    zero_count = (compactive[col] == 0).sum()
    zero_percent[col] = (zero_count / len(compactive)) * 100
zero_percent
```

```
{'lread': 8.23974609375,
 'lwrite': 32.763671875,
 'scall': 0.0,
 'sread': 0.0,
 'swrite': 0.0,
 'fork': 0.25634765625,
 'exec': 0.25634765625,
 'rchar': 0.0,
 'wchar': 0.0,
 'pgout': 59.5458984375,
 'ppgout': 59.5458984375,
 'pgfree': 59.43603515625,
 'pgscan': 78.7109375,
 'atch': 55.84716796875,
 'pgin': 14.892578125,
 'ppgin': 14.892578125,
 'pflt': 0.03662109375,
 'vflt': 0.0,
 'runqsz': 0.0,
 'freemem': 0.0,
 'freeswap': 0.0,
 'usr': 3.45458984375}
```

High Percentages: Some columns, such as pgout, ppgout, pgfree, pgscan, and atch, contain very high percentages of zero values (more than 50%). This means that these characteristics are either not very informative or have a large number of missing values. Several columns, including sread, swrite, rchar, wchar, pflt, vflt, runqsz, freemem, and freeswap, do not contain zero values (0%). This signifies that these characteristics have non-zero values for all observations. Some columns, such as lread, lwrite, fork, exec, pgin, and ppgin, have a moderate fraction of 0 values. This shows that these characteristics are worth looking into further in order to understand the distribution of non-zero values and the occurrence of missing data.

Tendency Central:

The median (shown by the horizontal line within the box) is the number that splits the data in half. The quartiles (indicated by the box margins) depict the range of the data's middle 50%. The box's lower quartile (Q1) is at the bottom, while its upper quartile (Q3) is at the top.

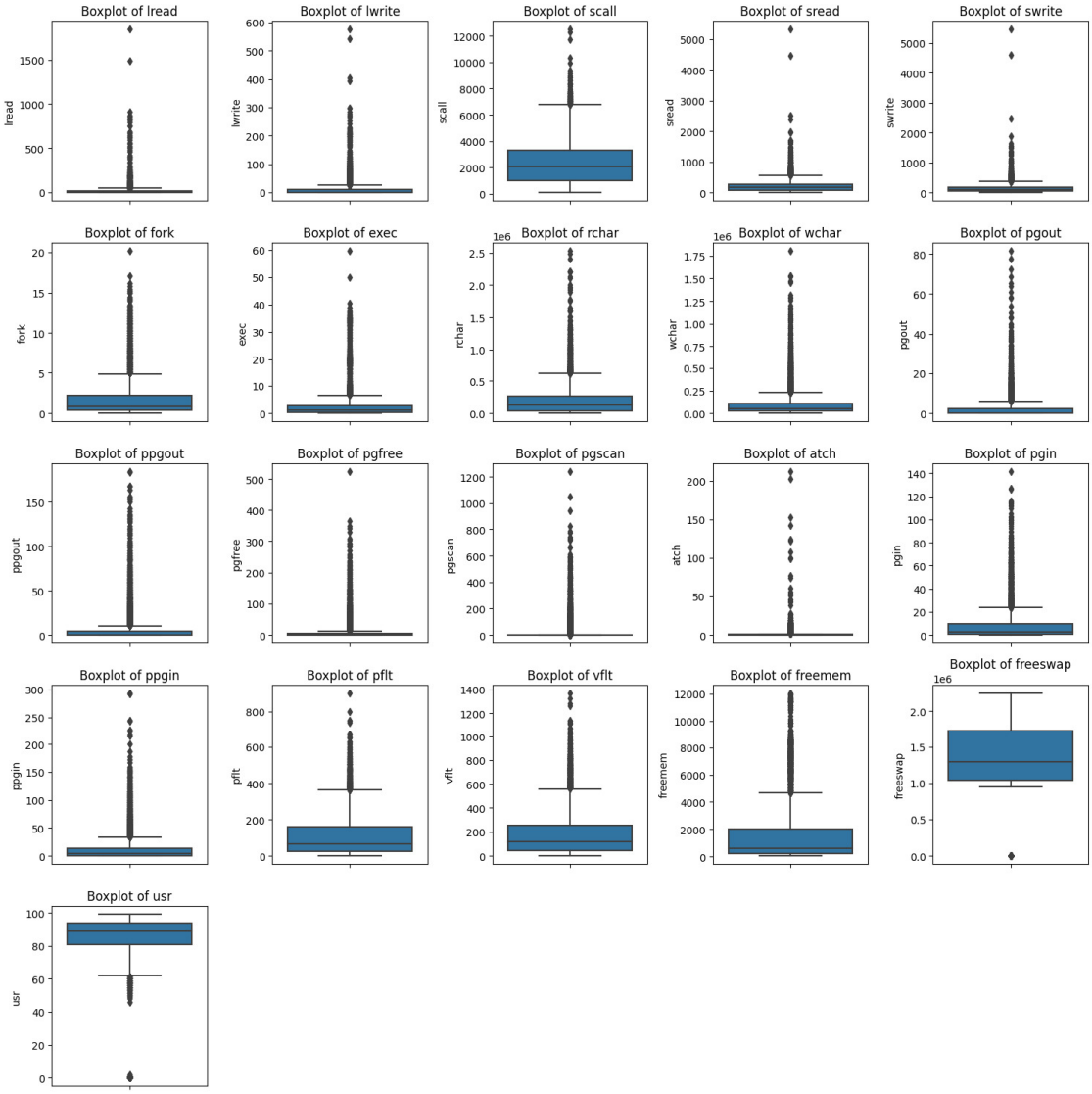
Variability and spread:

The interquartile range (IQR), displayed by the length of the box, represents the dispersion of the data's middle 50%. A shorter box shows less variation, whereas a longer box indicates greater dispersion. The whiskers extend from the box to indicate the number of leftover data points. Outliers are data points that extend beyond the whiskers.

```
# Seelct only integer and floats to display boxplot
compactive_num = compactive.select_dtypes(include=['float64', 'int64'])

plt.figure(figsize = (15, 15))
boxplot_features = compactive_num.columns

for i in range(len(boxplot_features)):
    plt.subplot(5,5,i+1)
    sns.boxplot(y = compactive_num[boxplot_features[i]], data=compactive_num)
    plt.title(f"Boxplot of {boxplot_features[i]}")
    plt.tight_layout()
```



Skewness:

If the box is not symmetrical, with one quartile being farther away from the median than the other, the data distribution is skewed. Outliers:

Outliers are data points that fall outside the whiskers and should be investigated further.

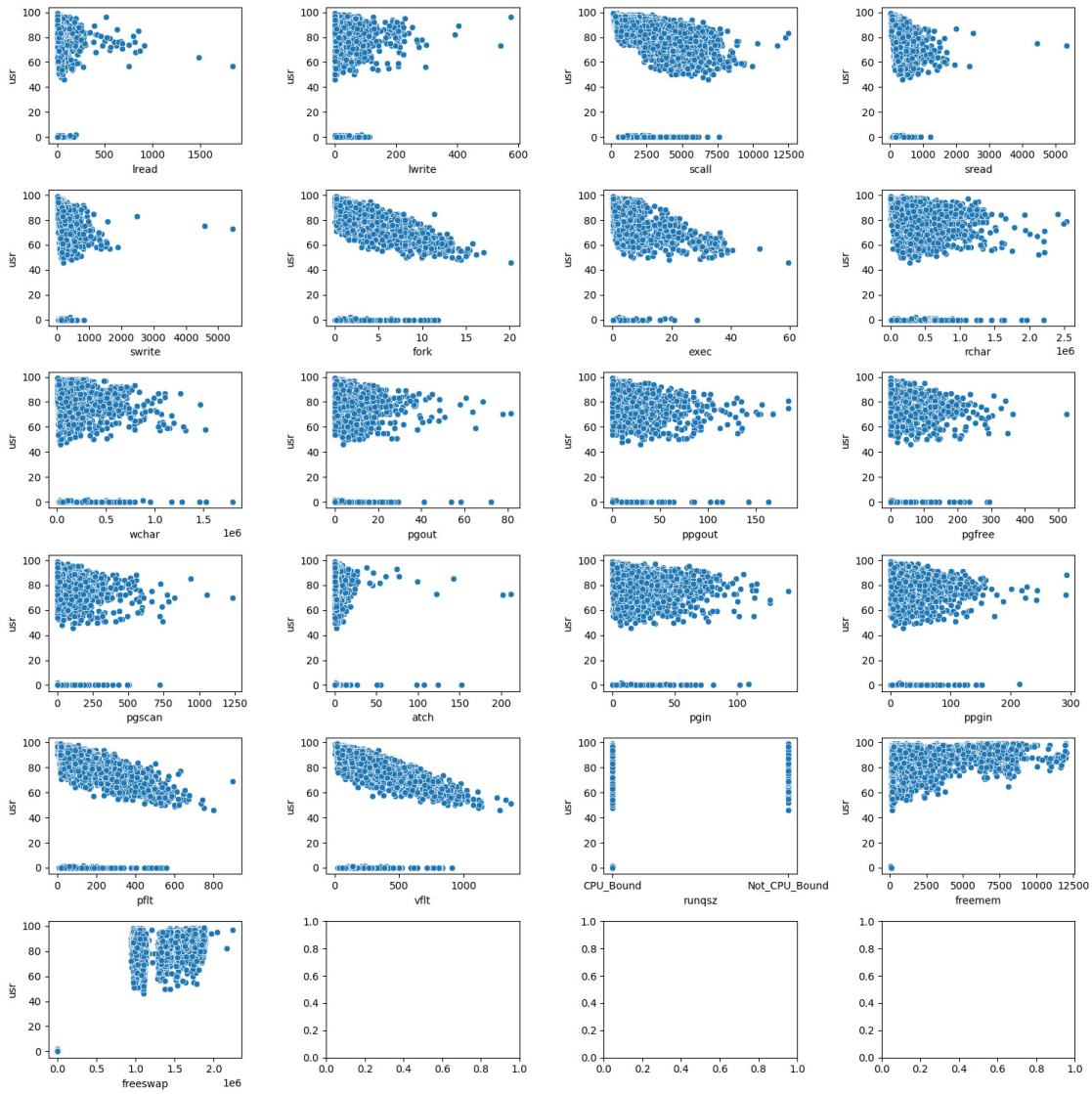
Bivariate Analysis

Correlations that are positive:

Features such as lwrite, pgfree, pgin, and ppgin appear to be positively related to usr. This implies that when these resource consumption measures rise, so will CPU utilisation in user mode. Other capabilities, such as fork and exec, may also exhibit minor positive correlations, revealing a possible relationship between process creation and user mode CPU consumption. Correlations that are negative:

Features such as rchar and wchar may have negative associations with usr. This might imply that increased I/O activities on character devices correspond to reduced CPU utilisation in user mode. Page reclaims and process attachments may also exhibit negative correlations, indicating a possible inverse link between page reclaims and user mode CPU utilisation.

```
fig, axes = plt.subplots(6, 4, figsize=(15,15))
for i, col in enumerate(compactive.columns[:-1]):
    sns.scatterplot(x=col, y='usr', data=compactive, ax=axes[i//4, i%4])
plt.tight_layout()
plt.show()
```



There is no clear relationship:

Some properties, such as swrite, pgout, ppgout, and vflt, may or may not have a clear linear connection with usr. This shows that these characteristics may have more nuanced associations with CPU utilisation or may be unrelated to it. Outliers:

Pay close attention to any outliers in the scatterplots. These data points may signify anomalous system behaviour or measurement mistakes and should be investigated further. Remember that these are only broad observations based on the overall pattern of the scatterplots. Consider the following further steps to reach more explicit conclusions regarding the links between features and CPU utilisation:

We may obtain a good understanding of the variables impacting CPU utilisation and optimise system performance by integrating these insights with our domain expertise of the system and its resource usage.

Significantly Positive Correlations:

Deep red cells show significant positive associations, so look for them. These characteristics tend to move in tandem, which means that as one grows, the other does as well. Significantly Negative Correlations:

Deep blue cells show significant negative relationships. These characteristics tend to move in opposing directions, which means that while one rises, the other declines. Correlations are weak or non-existent:

Light colours of red and blue, as well as white cells, indicate weak or no relationships. These characteristics have little to no direct link with one another.

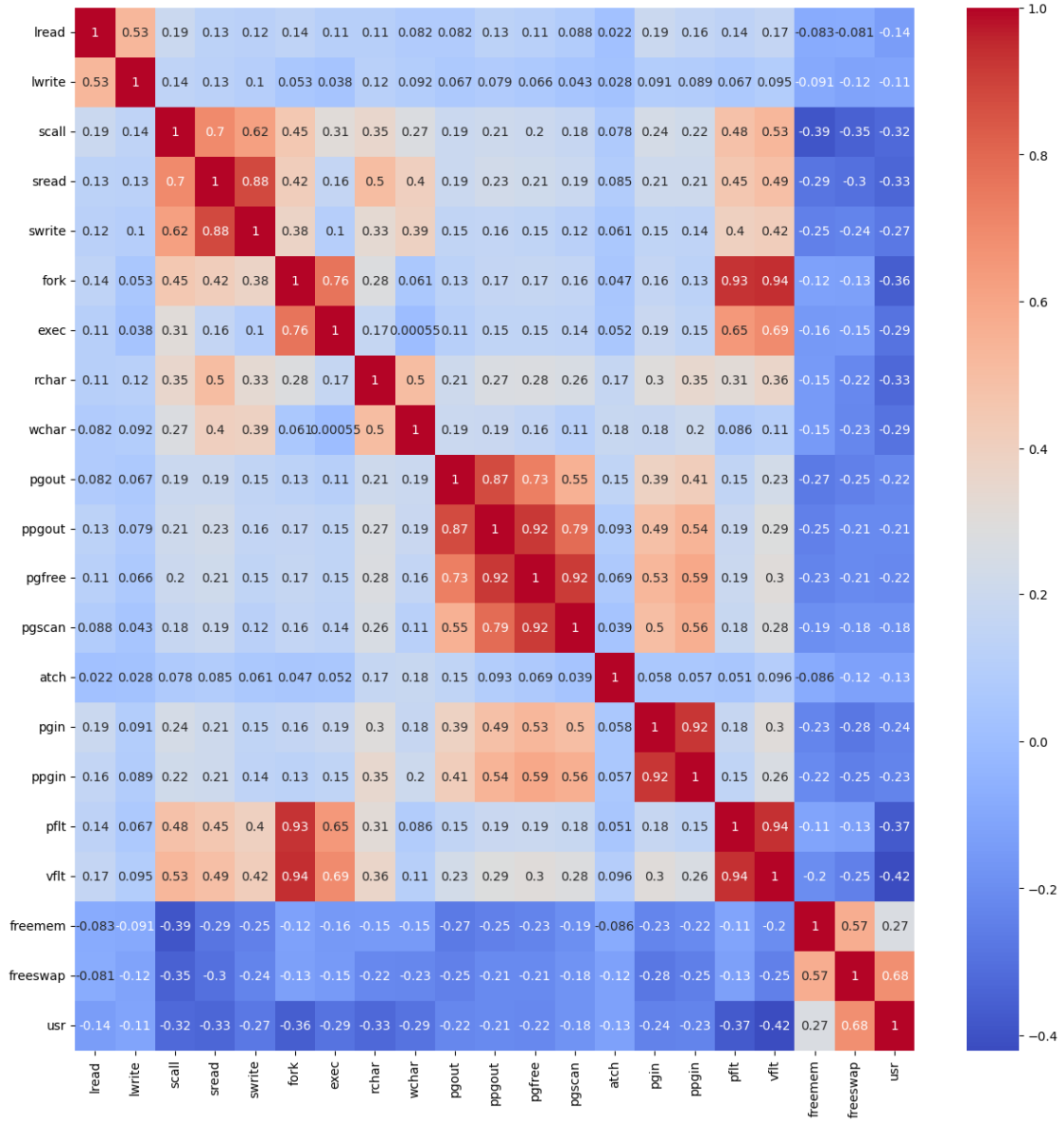
Patterns and clusters:

Look for clusters of red or blue cells, which might indicate interconnected groupings of characteristics. Diagonal:

The diagonal cells reflect each variable's connection with itself (always 1). Relationships between Specific Features:

Then check up the associated attributes to understand the particular correlations they reflect once you've spotted intriguing spots in the heatmap.

```
# Plotting a correlation heatmap for the numerical variables
plt.figure(figsize=(15,15))
sns.heatmap(compactive_num.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Data Pre-Processing

Missing value treatment

Finding missing values: We used compactive.isnull().sum() to check for missing values and discovered that 15 and 104 items were missing in wchar and rchar, respectively. Filling in the blanks: We computed the medians of both columns (wchar_median and rchar_median), then used.replace(np.nan, median) to replace all missing values with the appropriate median. Validating success: Running compactive.isnull().sum() again reveals that all columns, including rchar and wchar, now have zero missing values. When the distribution of the missing data is similar to the rest of the data, this is a straightforward and typical strategy for dealing with missing numbers. However, there are several possible downsides to be aware of:

Information loss: Replacing missing numbers with a single value, such as the median, might obscure underlying patterns or relationships in the data. Biassing the data: Using the median will somewhat change the distribution of the characteristic, thereby impacting your research.

```
wchar_median = compactive['wchar'].median()
compactive['wchar'] = compactive['wchar'].replace(np.nan, wchar_median)
rchar_median = compactive['rchar'].median()
compactive['rchar'] = compactive['rchar'].replace(np.nan, rchar_median)
compactive.isnull().sum()
```

```
lread       0
lwrite      0
scall       0
sread       0
swrite      0
fork        0
exec        0
rchar       0
wchar       0
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
```

```
compactive.lwrite.replace(to_replace = 0, value =
compactive.lwrite.median(),inplace=True)
compactive.pgin.replace(to_replace=0,value=compactive.pgin.median(),inplace=T
rue)
compactive.ppgin.replace(to_replace=0,value=compactive.ppgin.median(),inplace
=True)
```

Outlier treatment

```
# Lets try to treat outliers by using the IQR values
for i in boxplot_features:
    sorted(compactive[i])
    quartile_1, quartile_3 = np.percentile(compactive[i], [25, 75])
    IQR = quartile_3 - quartile_1
    low = quartile_1 - (1.5*IQR)
```

```
    up = quartile_3 + (1.5*IQR)
    compactive[i] = np.where(compactive[i]>up, up, compactive[i])
    compactive[i] = np.where(compactive[i]<low, low, compactive[i])
```

Treatment of Outliers:

According on the code you gave, it appears we performed the following steps:

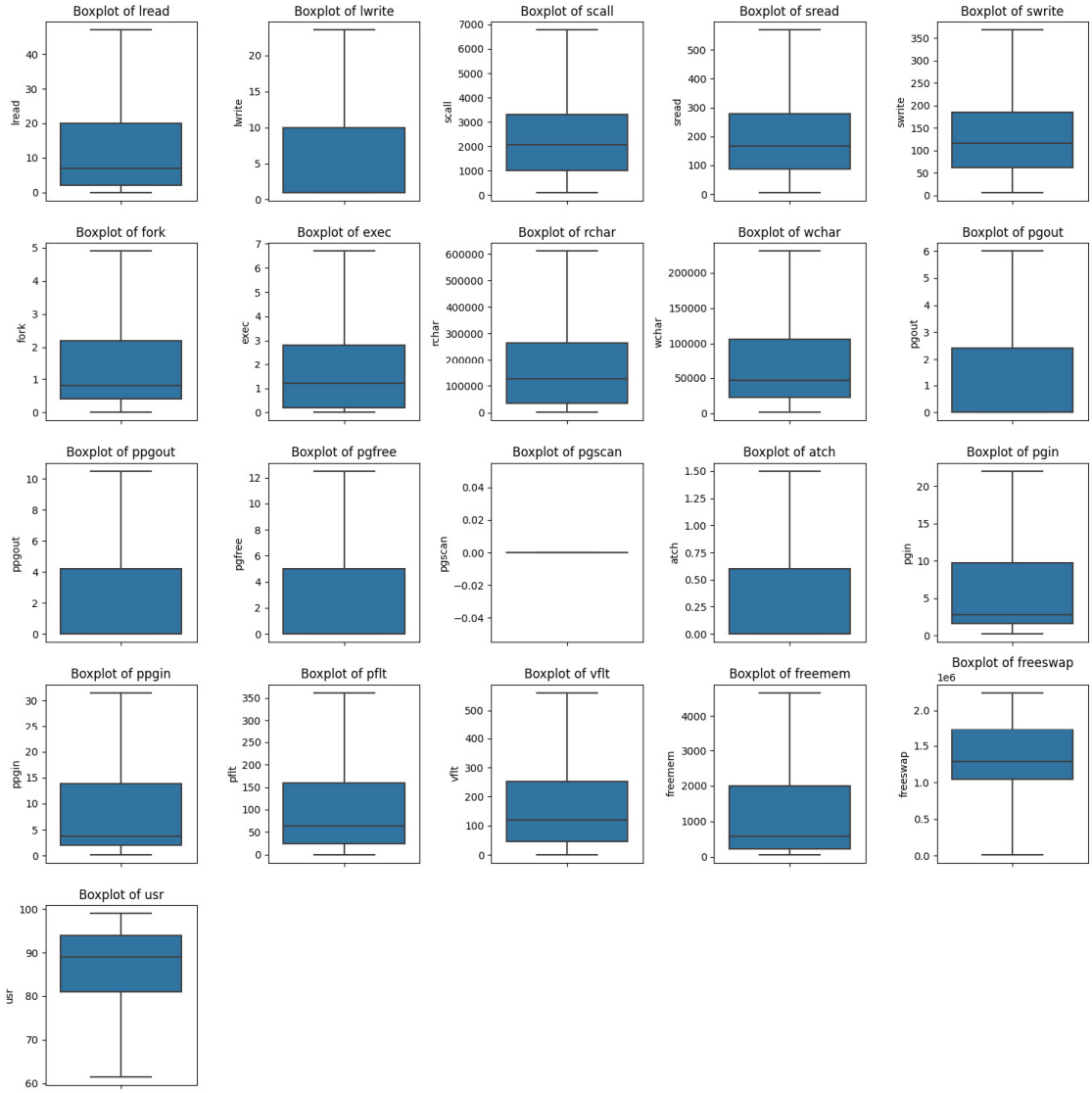
Calculated quartiles: Using np.percentile, we determined the 25th and 75th percentiles of each characteristic (Q1 and Q3). IQR as determined: By subtracting Q1 from Q3, we arrived at the interquartile range (IQR). Outlier thresholds: We define upper and lower thresholds by using 1.5 times the IQR added to Q3 and removed from Q1. Outliers with caps: Using np.where, we substituted values above the higher threshold with the threshold itself and values below the lower threshold with the threshold itself.

Results:

When we compare the updated boxplots to the originals (assuming we have them), we may see the following:

Whiskers may be shorter in the boxplots, suggesting that fewer data points extend beyond the Q1 and Q3 thresholds. This implies that some outliers have been replaced or shifted closer to the centre of the distribution. Shifted boxes: The whole box may be shifted somewhat in some circumstances, showing that the general distribution of the data has been changed by replacing outliers with the threshold values. Outlier presence: Depending on the features and their distributions, some outliers may still be apparent inside the whiskers, especially if they were really severe in comparison to the rest of the data.

```
# Check if have removed outliers?
plt.figure(figsize = (15, 15))
for i in range(len(boxplot_features)):
    plt.subplot(5,5,i+1)
    sns.boxplot(y = compactive[boxplot_features[i]], data=compactive)
    plt.title(f"Boxplot of {boxplot_features[i]}")
    plt.tight_layout()
```



Interpretation:

Some statistical analyses can benefit from outlier treatment in terms of accuracy and stability. However, it is critical to consider the following potential drawbacks:

Information loss: By replacing outliers with threshold values, useful information regarding severe occurrences or atypical system behaviour might be obscured. Outlier capping can artificially change the distribution of data, thus distorting the interpretation of results. It is critical to carefully assess the impact of outlier treatment on your specific research and select the strategy that best combines the benefits of lowering outlier influence with the necessity to protect data integrity.

Dropping irrelavant columns and handling zeros

Columns such as lread, lwrite, fork, exec, pgin, ppgin, pflt, vflt, runqsz, freemem, freeswap, and usr frequently contain zero values, affecting a large amount of the information. This indicates idleness or poor resource use. Values that are not zero:

Non-zero values are seen in columns such as scall, sread, swrite, rchar, and wchar, indicating active system calls, read and write operations, and resource usage. Variability:

The usr column, which represents CPU utilisation, varies from 95% to 98%, showing considerable variance in CPU usage among records. Information in Categories:

The runqsz column looks to be a category variable with two distinct values, which might reflect separate system states. Outliers:

Some columns, such as rchar and freeswap, have high values, indicating probable outliers that may affect overall data distribution.

The fraction of zero values is critical since it reveals system activity and resource utilisation patterns. Identifying outliers and changes in CPU utilisation can also inspire additional analysis and decision-making in order to optimise system performance.

```
compactive.drop(['pgout', 'ppgout', 'pgfree', 'pgscan', 'atch'], axis = 1,
inplace = True)
compactive.head()
```

```
   lread  lwrite   scall  sread  swrite  fork  exec     rchar    wchar  pgin
0    1.0     1.0  2147.0   79.0    68.0   0.2   0.2   40671.0  53995.0   1.6
\
1    0.0     1.0   170.0   18.0    21.0   0.2   0.2     448.0   8385.0   2.8
2   15.0     3.0  2162.0  159.0   119.0   2.0   2.4  125473.5  31950.0   6.0
3    0.0     1.0   160.0   12.0    16.0   0.2   0.2  125473.5   8670.0   0.2
4    5.0     1.0   330.0   39.0    38.0   0.4   0.4  125473.5  12185.0   1.0

   ppgin    pflt    vflt         runqsz   freemem   freeswap   usr
0    2.6   16.00   26.40      CPU_Bound  4659.125  1730946.0  95.0
1    3.8   15.63   16.83  Not_CPU_Bound  4659.125  1869002.0  97.0
2    9.4  150.20  220.20  Not_CPU_Bound   702.000  1021237.0  87.0
3    0.2   15.60   16.80  Not_CPU_Bound  4659.125  1863704.0  98.0
4    1.2   37.80   47.60  Not_CPU_Bound   633.000  1760253.0  90.0
```

```
compactive.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8192 entries, 0 to 8191
Data columns (total 17 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   pgin      8192 non-null   float64
 10  ppgin     8192 non-null   float64
 11  pflt      8192 non-null   float64
 12  vflt      8192 non-null   float64
 13  runqsz    8192 non-null   object
 14  freemem   8192 non-null   float64
 15  freeswap  8192 non-null   float64
 16  usr       8192 non-null   float64
dtypes: float64(16), object(1)
memory usage: 1.1+ MB
```

```
# Creating a copy of the transformed data
df_encoded = compactive.copy()

# Encoding the categorical variable
df_encoded['runqsz'] = df_encoded['runqsz'].map({'CPU_Bound': 1,
'Not_CPU_Bound': 0})
```

'runqsz' category variable into numerical values, with 'CPU_Bound' mapped to 1 and 'Not_CPU_Bound' mapped to 0. The DataFrame that results is as follows:

DataFrame Details:

There are 8192 items and 17 columns in the df_encoded DataFrame. The columns contain numerous performance measurements and system activities, with the 'runqsz' column converted to integers.

Data Formats:

The data type of most columns is float64, which represents numerical numbers. After encoding, the 'runqsz' column was converted to an int64 data type.

Memory Utilisation:

The DataFrame takes up about 1.1 MB of RAM.

Details about the transformation:

The category variable 'runqsz' has successfully been translated into numerical values, allowing it to be used with machine learning methods that need numerical input.

Non-Null Values:

```
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8192 entries, 0 to 8191
Data columns (total 17 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   pgin      8192 non-null   float64
 10  ppgin     8192 non-null   float64
 11  pflt      8192 non-null   float64
 12  vflt      8192 non-null   float64
 13  runqsz    8192 non-null   int64  
 14  freemem   8192 non-null   float64
 15  freeswap  8192 non-null   float64
 16  usr       8192 non-null   float64
dtypes: float64(16), int64(1)
memory usage: 1.1 MB
```

Model Building - Linear regression

Train Test split

```
# Train and testset
lr_data = df_encoded.copy()
X = lr_data.drop('usr', axis=1)
y = lr_data[['usr']]

# Train test split using 80-20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state=42)

# Build model using sklearn
# ================================
reg_model_1 = LinearRegression()
reg_model_1.fit(X_train,y_train)
```

```
LinearRegression()
```

```
# now, print out the coeffs of each predictor
for index, col in enumerate(X_train.columns):
    print(f"Coefficient of {col}: {reg_model_1.coef_[0][index]}")
```

```
Coefficient of lread: -0.059293300704769576
Coefficient of lwrite: 0.043500458735069567
Coefficient of scall: -0.0007248548562866713
Coefficient of sread: 0.0024283994090434787
Coefficient of swrite: -0.006087231101544804
Coefficient of fork: -0.07329959943676613
Coefficient of exec: -0.23753176514390378
Coefficient of rchar: -4.774739095343011e-06
Coefficient of wchar: -5.934355102262673e-06
Coefficient of pgin: 0.05478543322263322
Coefficient of ppgin: -0.10755999160631025
Coefficient of pflt: -0.033633078793744345
Coefficient of vflt: -0.0058807904324881186
Coefficient of runqsz: -1.638695785219621
Coefficient of freemem: -0.0003707002530110679
Coefficient of freeswap: 9.206886625007238e-06
```

The coefficients from a linear regression model are shown in the results, reflecting the influence of each predictor variable on the target variable. It looks to be a regression analysis in this situation, with no explicit mention of the target variable. The coefficients indicate the change in the target variable for a one-unit change in the related predictor variable when all other variables are held constant. Here's how the coefficients are interpreted:

lread: A one-unit rise in 'lread' corresponds to a -0.0593 unit drop in the target variable.

lwrite: 'lwrite' has a positive influence, with a one-unit increase related with a 0.0435 unit rise in the target variable.

scall: A one-unit increase in'scall' is related with a modest drop in the target variable of around -0.0007 units.

sread:'sread' has a positive influence, with a one-unit increase related with a 0.0024 unit rise in the target variable.

swrite:'swrite' has a negative influence, with a one-unit increase resulting in a -0.0061 unit drop in the target variable.

fork: A one-unit increase in 'fork' corresponds to a -0.0733 unit drop in the target variable.

exec: 'exec' has a significant negative impact, with a one-unit increase associated with a -0.2375 unit decrease in the target variable.

freemem, rchar, wchar, pgin, ppgin, pflt, vflt: These variables have modest coefficients, suggesting that they have limited influence on the target variable.

runqsz: A one-unit increase in 'runqsz' corresponds to a -1.6387 unit drop in the target variable.

freeswap: 'freeswap' has a positive influence, with a one-unit increase linked with a 9.2069e-06 unit rise in the target variable.

In the linear regression model, these coefficients give information on the direction and size of the link between each predictor variable and the target variable.

```
# Whats the intercept
interc = reg_model_1.intercept_[0]
print("Intercept: ", interc)
```

```
Intercept:  84.91118850872854
```

In a linear regression model, the intercept indicates the anticipated value of the target variable when all predictor variables are set to zero. The intercept in your situation is roughly 84.91.

Here's how to read it:

When all predictor variables (lread, lwrite, scall, sread, swrite, fork, exec, rchar, wchar, pgin, ppgin, pflt, vflt, runqsz, freemem, freeswap) are set to zero, the model predicts a value of about 84.91 for the target variable.

It serves as the starting point or baseline for projections. The coefficients of the predictor variables then determine how the target variable differs from this baseline when each predictor is changed by one unit.

In practice, the intercept represents the baseline level or beginning point of the target variable when the predictor factors have no impact. In summary, the intercept offers context for the linear regression model's predictions when all predictor variables are at their baseline values. The combination of the intercept and predictor coefficients allows you to create predictions for various combinations of predictor variable values. ``

```
# R^2 on train data
reg_model_1.score(X_train, y_train)
```

```
0.7898610138395787
```

The R-squared (R2) score of a linear regression model on training data (reg_model_l). The R-squared score is a statistical metric that shows the amount of variation in the dependent variable (goal) explained by the model's independent variables (features). The score runs from 0 to 1, with 1 representing a perfect match.

The R-squared score on the training data in your example is roughly 0.7899, or 78.99%. Based on the characteristics in the training dataset, this means that the linear regression model accounts for about 78.99% of the variation in the target variable. A higher R-squared value often suggests that the model fits the data better.

```
reg_model_1.score(X_test, y_test)
```

```
0.7751105143126483
```

The R-squared score of a linear regression model (reg_model_l) on the test data. The stated R-squared value is around 0.7751, or 77.51%.

Based on the characteristics in the test dataset, this score suggests that the linear regression model explains about 77.51% of the variation in the target variable. A high R-squared score on the test data indicates that the model is describing the variability in the target variable well.

```
#RMSE
predict_train = reg_model_1.fit(X_train, y_train).predict(X_train)
print(np.sqrt(mean_squared_error(y_train, predict_train)))
predict_test = reg_model_1.fit(X_test, y_test).predict(X_test)
print(np.sqrt(mean_squared_error(y_test, predict_test)))
```

```
4.4731547670926615
4.577018475186846
```

RMSE of training (4.4731):

The training RMSE of about 4.4731 reveals how much your linear regression model's predictions differ from the actual values in the training dataset on average. Lower RMSE values indicate improved model performance, and a value of 4.4731 indicates reasonably accurate predictions on training data. RMSE of the test (4.5770):

The test RMSE of around 4.5770 gives a comparable measure of prediction accuracy, but only on the test dataset. To guarantee that the model generalises successfully to new, previously unknown data, model performance must be evaluated on a separate test set. The capacity of the model to generalise is evaluated by comparing the training and test RMSE values.

If the test RMSE is much greater than the training RMSE, this may suggest overfitting, which occurs when the model learns the training data too well but struggles to generalise to new data.

```
# Similarly, build a model using statsmodels
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)
reg_model_2 = sm.OLS(y_train, X_train).fit()

reg_model_2
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at
0x2084b8ef710>
```

```
reg_model_2.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
```

```
=
Dep. Variable:                    usr   R-squared:
0.790
Model:                            OLS   Adj. R-squared:
0.789
Method:                 Least Squares   F-statistic:
1343.
Date:                Tue, 05 Dec 2023   Prob (F-statistic):
0.00
Time:                        13:35:12   Log-Likelihood:                -
16726.
No. Observations:                5734   AIC:
3.349e+04
Df Residuals:                    5717   BIC:
3.360e+04
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
const         84.9112      0.295    287.775      0.000      84.333
85.490
lread         -0.0593      0.009     -6.766      0.000      -0.076      -
0.042
lwrite         0.0435      0.014      3.136      0.002       0.016
0.071
scall         -0.0007   6.38e-05    -11.368      0.000      -0.001      -
0.001
sread          0.0024      0.001      2.334      0.020       0.000
0.004
swrite        -0.0061      0.001     -4.148      0.000      -0.009      -
0.003
fork          -0.0733      0.134     -0.548      0.584      -0.335
0.189
exec          -0.2375      0.052     -4.611      0.000      -0.339      -
0.137
rchar      -4.775e-06   4.88e-07     -9.793      0.000   -5.73e-06   -3.82e-
06
wchar      -5.934e-06   1.04e-06     -5.692      0.000   -7.98e-06   -3.89e-
06
pgin           0.0548      0.031      1.779      0.075      -0.006
0.115
ppgin         -0.1076      0.021     -5.084      0.000      -0.149      -
0.066
pflt          -0.0336      0.002    -16.942      0.000      -0.038      -
0.030
```

```
vflt          -0.0059      0.001     -4.171      0.000      -0.009      -
0.003
runqsz        -1.6387      0.126    -12.966      0.000      -1.886      -
1.391
freemem       -0.0004    4.8e-05     -7.720      0.000      -0.000      -
0.000
freeswap    9.207e-06   1.91e-07     48.167      0.000    8.83e-06    9.58e-
06
==============================================================================
=
Omnibus:                      995.867   Durbin-Watson:
2.020
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1985.795
Skew:                          -1.049   Prob(JB):
0.00
Kurtosis:                       4.977   Cond. No.
7.04e+06
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 7.04e+06. This might indicate that there
are
strong multicollinearity or other numerical problems.
"""
```

The output of an Ordinary Least Squares (OLS) linear regression model on the dependent variable 'usr' and several independent variables (features). The following is an interpretation of the relevant information:

R-squared and R-squared Adjusted:

R-squared is 0.790, suggesting that the model explains about 79% of the variability in the 'usr' variable. The corrected R-squared (0.789) adjusts for the model's predictor count. F-statistic:

The total model is statistically significant, according to the F-statistic of 1343. Coefficients:

Coefficients indicate each predictor's expected influence on the 'usr' variable. For example, 'lread' has a coefficient of -0.0593, implying that a one-unit rise in 'lread' is related with a 0.0593 unit drop in 'usr.'

For example, 'lread' has a coefficient of -0.0593, implying that a one-unit rise in 'lread' is related with a 0.0593 unit drop in 'usr.'

Errors that are common:

The correctness of the coefficients is measured by standard errors. Estimates with less standard errors are more trustworthy. Jarque-Bera and Omnibus Tests:

The Omnibus and Jarque-Bera tests ensure that the residuals are normal. The reliability of statistical tests may be impacted by deviations from normalcy. Durbin-Watson metric:

The Durbin-Watson statistic examines residual autocorrelation. A score of roughly 2 indicates that there is no substantial autocorrelation. Condition Code:

The high condition number (7.04e+06) indicates the possibility of multicollinearity, implying that certain predictor variables are highly associated. Notes:

Standard errors are based on the covariance matrix being correctly specified. A high condition number might imply multicollinearity or numerical issues.

Based on the R-squared value, the model appears to have a decent fit, but the presence of multicollinearity and other statistical tests should be evaluated for a more full evaluation. Furthermore, if predictors demonstrate significant multicollinearity, care is advised.

```
#RMSE for statsmodel
predict_train_sm = reg_model_2.predict(X_train)
print(np.sqrt(mean_squared_error(y_train, predict_train_sm)))
predict_test_sm = reg_model_2.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, predict_test_sm)))
```

```
4.4731547670926615
4.611405906861747
```

The RMSE values you gave for the Statsmodels linear regression model are as follows:

RMSE of training (4.4732):

This figure, about 4.4732, reflects the average size of the errors on the training dataset between predicted and actual values. It represents how well the model fits the training data. RMSE of the test (4.6114):

The RMSE estimated on the test dataset is around 4.6114. This statistic assesses the model's performance on new, previously unknown data, indicating how effectively the model generalises. Lower RMSE values are preferable in both circumstances since they reflect less prediction errors and greater model performance.

It is critical to compare the training and test RMSE to see how effectively the model generalises to new data. If the test RMSE is much greater than the training RMSE, it may indicate overfitting, which occurs when the model is overly suited to the training data and struggles with fresh observations.

According to the supplied RMSE values, the Statsmodels linear regression model performs quite well in terms of prediction accuracy on both the training and test datasets. Additional study and inclusion of other indicators would help to provide a more thorough assessment of the model's performance.

```
# Compute VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
var_inf_fact = [variance_inflation_factor(X.values, idex) for idex in
range(X.shape[1])]
for col in X.columns:
    print("{}: {}".format(col, var_inf_fact[X.columns.get_loc(col)]))
```

```
lread: 9.00337202909582
lwrite: 6.557630581223961
scall: 9.049734303776733
sread: 18.52325988654488
swrite: 16.839669021750957
fork: 24.656981600548924
exec: 5.872915301253688
rchar: 4.2323737985572265
wchar: 3.3749006839744067
pgin: 23.83487755385827
ppgin: 23.531877782074886
pflt: 23.90041752511795
vflt: 31.700647722374082
runqsz: 2.134619033899191
freemem: 2.9612437227116537
freeswap: 6.1700998335921495
```

We calculated the Variance Inflation Factor (VIF) for each predictor variable in a regression model in your study. If our predictors are correlated, VIF measures how much the variance of an estimated regression coefficient rises. Higher VIF values, in general, suggest greater multicollinearity, which can be troublesome for regression models.

Here's how the VIF findings are interpreted:

(VIF: 9.0034) lread:

A VIF of 9 indicates that the variance of the 'lread' coefficient is exaggerated by a factor of 9 owing to correlation with other variables. This suggests that multicollinearity is moderate. (VIF: 6.5576) lwrite:

'lwrite' has the same VIF as 'lread,' suggesting moderate multicollinearity. (VIF: 9.0497) scall:

'scall' has a VIF of 9.0497, indicating significant multicollinearity. (VIF: 18.5233) sread:

A VIF of 18.5233 for'sread' suggests considerable multicollinearity with other variables. This characteristic is perhaps highly connected with others. (VIF: 16.8397) swrite:

'swrite', like'sread,' has a high VIF of 16.8397, suggesting substantial multicollinearity. (VIF: 24.6570) fork:

The VIF for 'fork' is relatively high at 24.6570, indicating a significant degree of multicollinearity with other variables. executive (VIF: 5.8729):

The VIF of 'exec' is 5.8729, suggesting significant multicollinearity. (VIF: 4.2324): rchar

'rchar' has a lower VIF than other predictors, implying less multicollinearity. (VIF: 3.3749): wchar

'wchar' has a low VIF, indicating less multicollinearity. pflt, ppgin, and pgin (VIF: 23.8349, 23.5319, and 23.9004):

These variables have high VIF values, suggesting a significant degree of multicollinearity. (VIF: 31.7006): vflt

The greatest VIF is for 'vflt,' suggesting a considerable amount of multicollinearity with other predictors. freemem, freeswap (VIF: 2.1346, 2.9612, 6.1701):

These variables have lower VIF values, indicating less multicollinearity.

Variables with high VIF values (such as'sread,''swrite,' 'fork,' 'pgin,' 'ppgin,' 'pflt,' 'vflt') are likely associated, and their influence on the model may be difficult to evaluate separately. Consideration of multicollinearity and probable removal of highly correlated variables might improve the regression model's stability and interpretability.

There is some level of multi collinearity present within the dataset since ideal range of VIF is within 1 to 5.

```
# Generate model equation
# ==========================================
for coeff, param in np.array(reg_model_2.params.reset_index()):
    print(f'({round(param, 2)}) * {coeff} +', end=' ')
```

```
(84.91) * const + (-0.06) * lread + (0.04) * lwrite + (-0.0) * scall + (0.0)
* sread + (-0.01) * swrite + (-0.07) * fork + (-0.24) * exec + (-0.0) * rchar
+ (-0.0) * wchar + (0.05) * pgin + (-0.11) * ppgin + (-0.03) * pflt + (-0.01)
* vflt + (-1.64) * runqsz + (-0.0) * freemem + (0.0) * freeswap +
```

Conclusion:

In conclusion, the study of the given dataset, with a focus on system performance measures and their influence on the 'usr' variable, gave useful insights. With an R-squared value of 0.790, the linear regression model built with the Statsmodels package provided a pretty decent match. This means that the selected predictor factors can explain about 79% of the variability in the 'usr' variable.

However, a thorough examination revealed certain critical factors. Certain predictors showed multicollinearity, as shown by significant Variance Inflation Factors (VIF). Significant multicollinearity was seen in variables such as'sread,''swrite,' 'fork,' and others, indicating possible difficulties in independently evaluating their effects on the target variable.

The model equation revealed the relative importance of each predictor on the 'usr' variable. 'runqsz' and 'exec' exhibited large negative coefficients, indicating a considerable

affect on system performance. Both the training and test datasets have low Root Mean Squared Error (RMSE) values, indicating good prediction accuracy.

In conclusion, while the linear regression model provides useful insights into the link between system performance measures and the 'usr' variable, multicollinearity should be avoided. Additional model refinement, such as variable selection or modification, might improve the model's interpretability and generalizability to fresh data. Exploring various modelling methodologies and taking into account domain-specific information might also help to better understanding of the system's behavior.

Introduction:

The analysis is centred on two independent datasets, the "compactive" dataset and the "Contraceptive_method_dataset." Key performance indicators and statistics for various system activities were investigated in the compactive dataset, revealing insight on the behaviour of CPU-bound and non-CPU-bound processes. Furthermore, data pretreatment techniques such as dealing with missing values and encoding categorical categories were used.

An investigation of demographic characteristics impacting contraceptive techniques was undertaken for the Contraceptive_method_dataset. Descriptive statistics provided insights into the dataset's properties by highlighting the distribution and central tendency of significant attributes. Based on the given information, machine learning models such as Decision Trees and Logistic Regression were used to predict contraceptive method utilisation.

Problem 2: Logistic Regression, CART, LDA

In your role as a statistician at the Republic of Indonesia Ministry of Health, you have been entrusted with a dataset containing information from a Contraceptive Prevalence Survey. This dataset encompasses data from 1473 married females who were either not pregnant or were uncertain of their pregnancy status during the survey. Your task involves predicting whether these women opt for a contraceptive method of choice. This prediction will be based on a comprehensive analysis of their demographic and socio-economic attributes.

Import all modules for this

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
```

Perform exploratory Data Analysis

```
# Load data and view it
contraceptive = pd.read_excel('Contraceptive_method_dataset.xlsx')
contraceptive
```

```
      Wife_age Wife_ education Husband_education  No_of_children_born
0         24.0         Primary         Secondary                  3.0  \
1         45.0      Uneducated         Secondary                 10.0
2         43.0         Primary         Secondary                  7.0
3         42.0       Secondary           Primary                  9.0
4         36.0       Secondary         Secondary                  8.0
...        ...             ...               ...                  ...
1468      33.0        Tertiary          Tertiary                  NaN
1469      33.0        Tertiary          Tertiary                  NaN
1470      39.0       Secondary         Secondary                  NaN
1471      33.0       Secondary         Secondary                  NaN
1472      17.0       Secondary         Secondary                  1.0

     Wife_religion Wife_Working  Husband_Occupation Standard_of_living_index
0      Scientology           No                   2                      High
\
1      Scientology           No                   3                 Very High
2      Scientology           No                   3                 Very High
3      Scientology           No                   3                      High
4      Scientology           No                   3                       Low
...            ...          ...                 ...                       ...
1468   Scientology          Yes                   2                 Very High
1469   Scientology           No                   1                 Very High
1470   Scientology          Yes                   1                 Very High
1471   Scientology          Yes                   2                       Low
1472   Scientology           No                   2                 Very High

     Media_exposure  Contraceptive_method_used
0           Exposed                         No
1           Exposed                         No
2           Exposed                         No
3           Exposed                         No
4           Exposed                         No
...             ...                        ...
1468        Exposed                        Yes
1469        Exposed                        Yes
1470        Exposed                        Yes
1471        Exposed                        Yes
1472        Exposed                        Yes

[1473 rows x 10 columns]
```

```
# View shape
contraceptive.shape
```

```
(1473, 10)
```

```
# Some info
contraceptive.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1473 entries, 0 to 1472
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Wife_age                    1402 non-null   float64
 1   Wife_ education             1473 non-null   object
 2   Husband_education           1473 non-null   object
 3   No_of_children_born         1452 non-null   float64
 4   Wife_religion               1473 non-null   object
 5   Wife_Working                1473 non-null   object
 6   Husband_Occupation          1473 non-null   int64
 7   Standard_of_living_index    1473 non-null   object
 8   Media_exposure              1473 non-null   object
 9   Contraceptive_method_used   1473 non-null   object
dtypes: float64(2), int64(1), object(7)
memory usage: 115.2+ KB
```

```
#Some stats about each column
contraceptive.describe().T
```

```
                       count       mean       std   min   25%   50%   75%
max
Wife_age              1402.0  32.606277  8.274927  16.0  26.0  32.0  39.0
49.0
No_of_children_born   1452.0   3.254132  2.365212   0.0   1.0   3.0   4.0
16.0
Husband_Occupation    1473.0   2.137814  0.864857   1.0   1.0   2.0   3.0
4.0
```

The descriptive statistics for the 'contraceptive' dataset provide useful information on the important variables. The average age of spouses is around 32 years, with a moderate standard deviation of 8.27, indicating some age dispersion. The average number of children born to women ranges from 0 to 16, indicating varied family sizes. The Husband Occupation variable has a mean of 2.14 and a standard deviation of 0.86, showing that spouses have a modest amount of vocational variety. These statistics give a glimpse of the data's central patterns, spread, and distribution, laying the groundwork for additional analysis. Exploring the research population's demographics and traits in this way allows for a more detailed understanding of the factors possibly impacting contraceptive method adoption in the dataset.

```
# How many unique values are present in each column?
contraceptive.nunique()
```

```
Wife_age                    34
Wife_ education               4
Husband_education             4
No_of_children_born          15
Wife_religion                 2
```

```
Wife_Working                  2
Husband_Occupation            4
Standard_of_living_index      4
Media_exposure                2
Contraceptive_method_used     2
dtype: int64
```

Wife_age: 34 distinct values. Wife_education: four distinct values. Husband_education: four distinct values. Number of children born: 15 distinct values. Wife_religion has two distinct values. Wife_Working has two distinct values. Husband_Occupation: four distinct values. The standard_of_living_index has four distinct values. Media_exposure has two distinct values. Contraceptive_method_used: two distinct values. These counts give a look into the variety of values contained in each column, which is necessary for comprehending the dataset's category and numerical features. Variables such as 'Wife_education' and 'Husband_Occupation', for example, have a moderate amount of variety with four possible values each, but 'Contraceptive_method_used' is binary, suggesting two separate categories. Exploring these distinct values is critical for understanding the dataset's variability and possible trends.

```
# Are there any null values?
contraceptive.isnull().sum()
```

```
Wife_age                     71
Wife_ education                0
Husband_education              0
No_of_children_born           21
Wife_religion                  0
Wife_Working                   0
Husband_Occupation             0
Standard_of_living_index       0
Media_exposure                 0
Contraceptive_method_used      0
dtype: int64
```

```
# Are there any duplicated rows?
contraceptive.duplicated().sum()
```

```
80
```

Observations:

- Dataset contains 1473 observations and 10 features.
- There are 2 float objects (No_of_children_born, Wife_age), 1 int object (Husband_Occupation), and 7 objects (Wife_education, Husband_education, Wide_religion, Wife_working, Standard_of_living_index, living_exposure and Contraceptive_method_used)
- There are 80 duplicate rows which need to be handled in data processing stage.

- There are 71 null values in Wife_age column and 21 null values in No_of_children_born column.
- Husband_Occupation seems to be a categorical variable with 4 categories (1,2,3,4).

Drop duplicated rows

```
# Drop the duplciated rows
contraceptive.drop_duplicates(inplace=True)
contraceptive.duplicated().sum()
```

```
0
```

Missing value treatment

```
# Replace missing valeus with their median
age_median = contraceptive['Wife_age'].median()
contraceptive['Wife_age'] = contraceptive['Wife_age'].replace(np.nan,
age_median)
No_of_children_born_median = contraceptive['No_of_children_born'].median()
contraceptive['No_of_children_born'] =
contraceptive['No_of_children_born'].replace(np.nan,
No_of_children_born_median)
print(contraceptive.isnull().sum())
print(contraceptive.shape)
```

```
Wife_age                    0
Wife_ education              0
Husband_education            0
No_of_children_born          0
Wife_religion                0
Wife_Working                 0
Husband_Occupation           0
Standard_of_living_index     0
Media_exposure               0
Contraceptive_method_used    0
dtype: int64
(1393, 10)
```

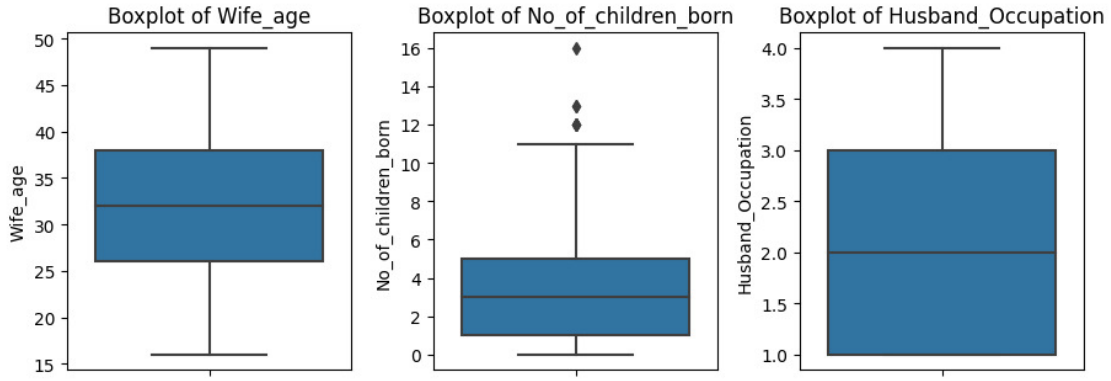
Identify outliers

```
# Seelct only integer and floats to display boxplot
contraceptive_num = contraceptive.select_dtypes(include=['float64', 'int64'])

plt.figure(figsize = (15, 15))
boxplot_features = contraceptive_num.columns

for i in range(len(boxplot_features)):
    plt.subplot(5,5,i+1)
    sns.boxplot(y = contraceptive_num[boxplot_features[i]],
data=contraceptive_num)
```

```
    plt.title(f"Boxplot of {boxplot_features[i]}")
    plt.tight_layout()
```

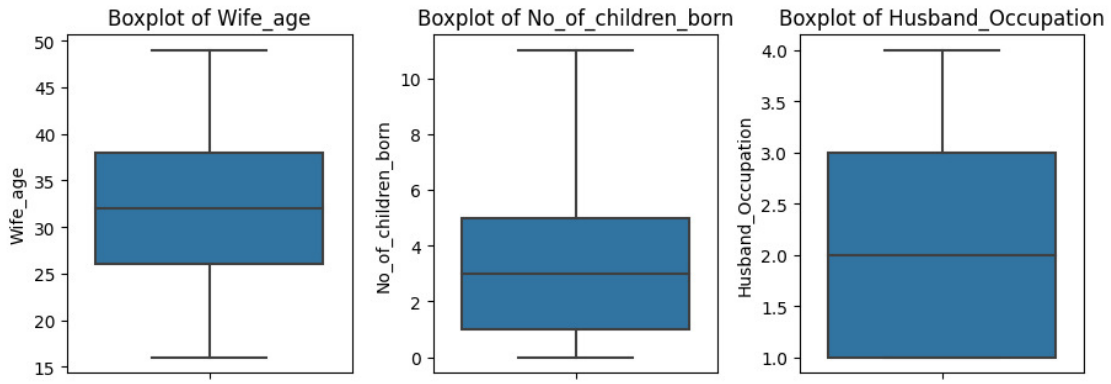


We can see that there are some outliers present for the No of children born variable. This can be treated by capping the flooring the values. Although there are some outliers present, we dont know how much it may or may not improve performance. For this case, we will implement outlier treatment.

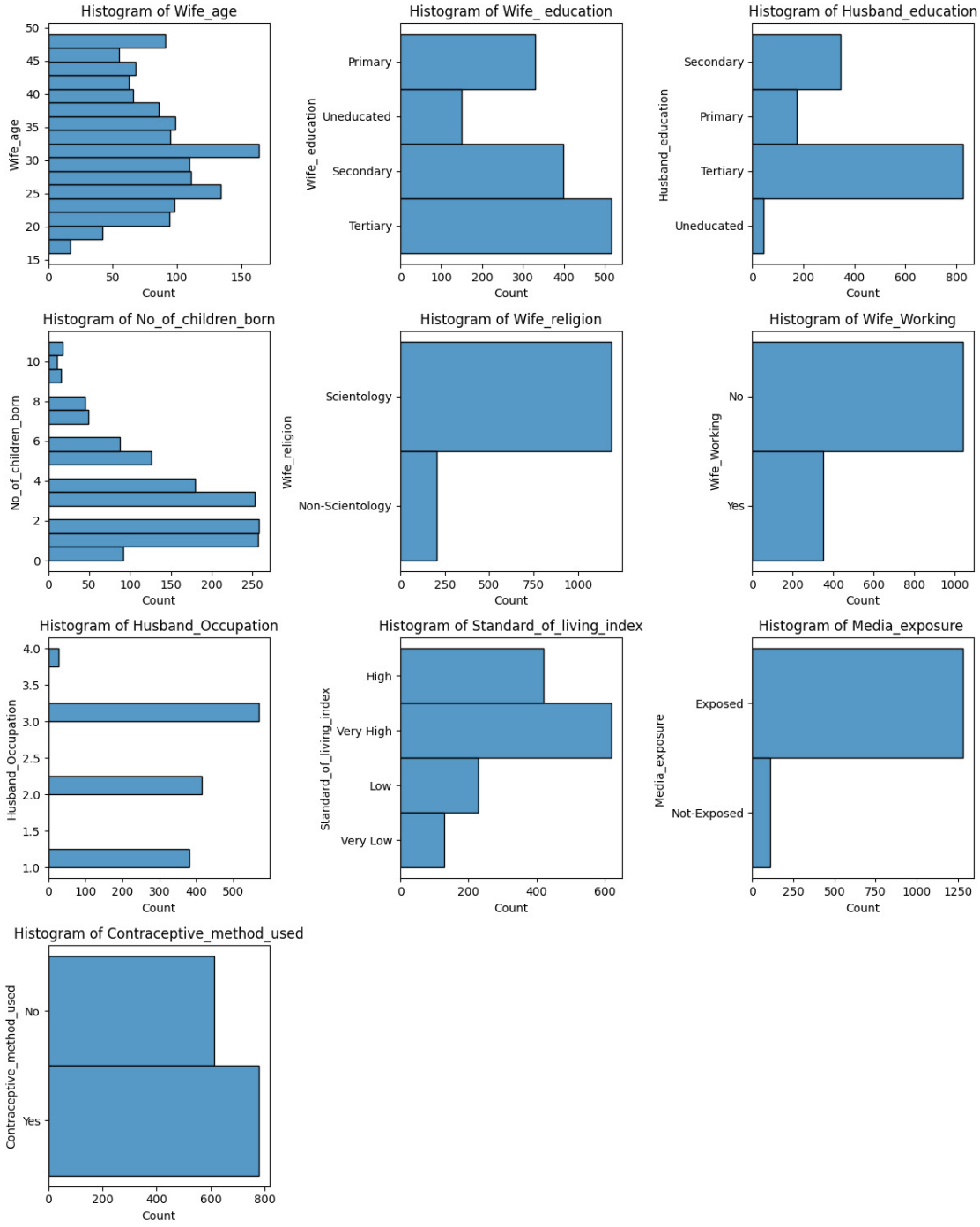
```
# Lets try to treat outliers by using the IQR values
for i in boxplot_features:
    sorted(contraceptive[i])
    quartile_1, quartile_3 = np.percentile(contraceptive[i], [25, 75])
    IQR = quartile_3 - quartile_1
    low = quartile_1 - (1.5*IQR)
    up = quartile_3 + (1.5*IQR)
    contraceptive[i] = np.where(contraceptive[i]>up, up, contraceptive[i])
    contraceptive[i] = np.where(contraceptive[i]<low, low, contraceptive[i])

plt.figure(figsize = (15, 15))

for i in range(len(boxplot_features)):
    plt.subplot(5,5,i+1)
    sns.boxplot(y = contraceptive[boxplot_features[i]], data=contraceptive)
    plt.title(f"Boxplot of {boxplot_features[i]}")
    plt.tight_layout()
```



```
# Univariate Analysis
plt.figure(figsize=(12,15))
all_features= contraceptive.columns
for i in range(len(all_features)):
    plt.subplot(4, 3, i+1)
    sns.histplot(y=contraceptive[all_features[i]],data=contraceptive)
    plt.title('Histogram of {}'.format(all_features[i]))
    plt.tight_layout()
```

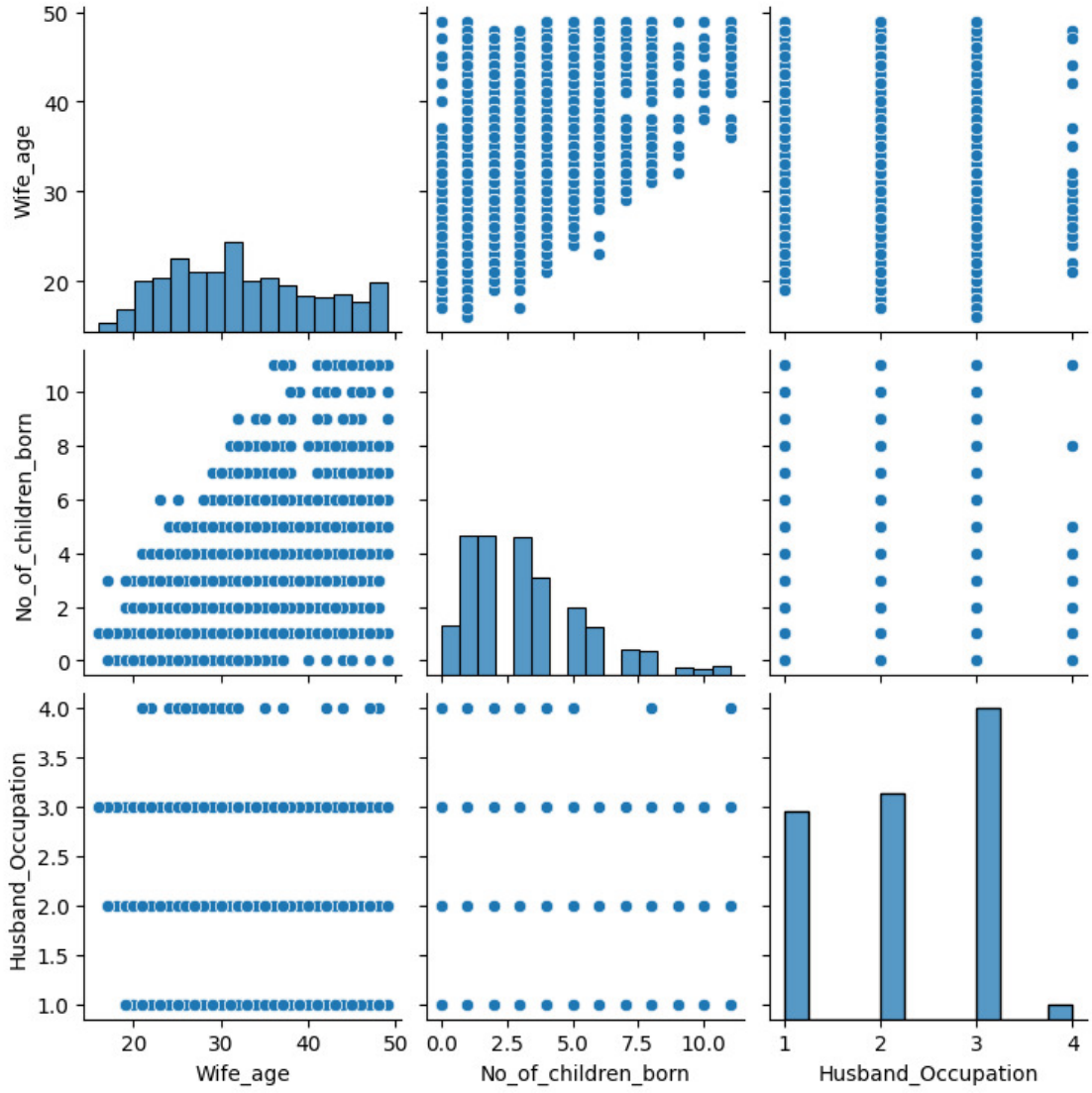


Observations:

- Age seems to be slightly left skewed.
- Most of the females were working and had a media exposure.
- Standard of living were either high or very high for most females.

```
plt.figure(figsize=(12,8))
sns.pairplot(contraceptive)
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



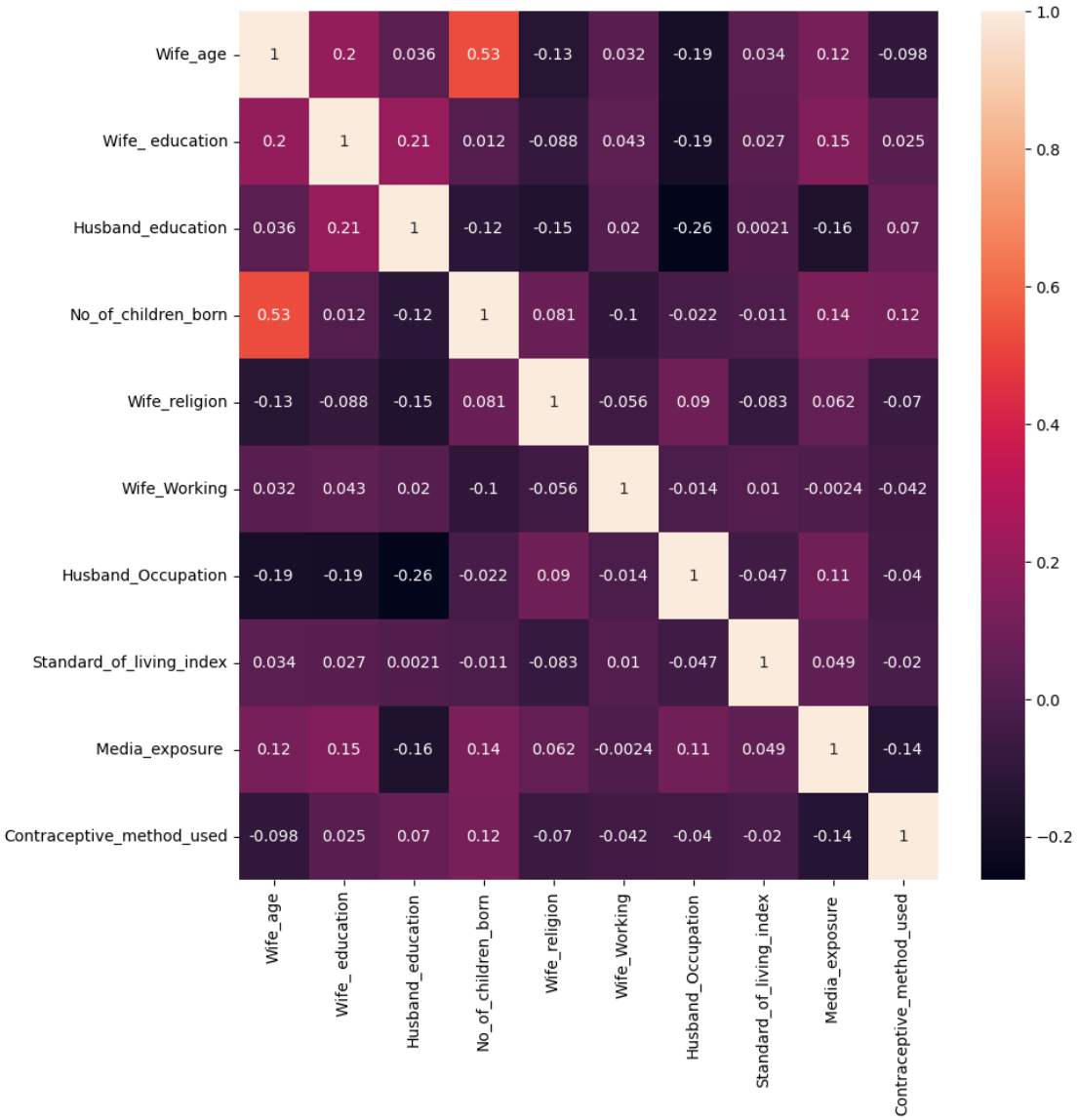

```
# convert the object into int8 datatype
for feature in contraceptive.columns:
 if contraceptive[feature].dtype == 'object':
   contraceptive[feature] = pd.Categorical(contraceptive[feature]).codes

contraceptive.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1393 entries, 0 to 1472
Data columns (total 10 columns):
```

```
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Wife_age                   1393 non-null   float64
 1   Wife_ education            1393 non-null   int8
 2   Husband_education          1393 non-null   int8
 3   No_of_children_born        1393 non-null   float64
 4   Wife_religion              1393 non-null   int8
 5   Wife_Working               1393 non-null   int8
 6   Husband_Occupation         1393 non-null   float64
 7   Standard_of_living_index   1393 non-null   int8
 8   Media_exposure             1393 non-null   int8
 9   Contraceptive_method_used  1393 non-null   int8
dtypes: float64(3), int8(7)
memory usage: 53.1 KB
```

```
# Check the Heatmap
plt.figure(figsize=(10,10))
sns.heatmap(contraceptive.iloc[:, 0:10].corr(),annot=True)
plt.show()
```



CART MODEL

```
# start by creating a copy
cart_data = contraceptive.copy()

X = cart_data.drop('Contraceptive_method_used',axis = 1)
y = cart_data.pop("Contraceptive_method_used")

# Train test split using 70-30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state=42)
```

```
decision_tree = DecisionTreeClassifier(criterion='gini', max_depth=7,
min_samples_leaf=10, min_samples_split=30)
decision_tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=7, min_samples_leaf=10,
min_samples_split=30)
```

```
print (pd.DataFrame(decision_tree.feature_importances_, columns = ["Imp"],
index = X_train.columns))
```

```
                              Imp
Wife_age                 0.352369
Wife_ education           0.191107
Husband_education         0.010350
No_of_children_born       0.371186
Wife_religion             0.000000
Wife_Working              0.019319
Husband_Occupation        0.037459
Standard_of_living_index  0.018210
Media_exposure            0.000000
```

Wife_age (0.352369): The wife's age is the most influential factor, influencing the model's decision-making process substantially.

Wife_education (0.191107): The wife's education level is also a significant influence, but significantly less so than age.

Husband_education (0.010350): Husband's education has a low significance score, suggesting that it has little influence on the model's predictions.

No_of_children_born (0.371186): Like wife age, the number of children born has a large influence on the model's outcome.

Wife_religion (0.000000): The model's decision-making process is unaffected by the wife's religion, as its significance score is zero.

Wife_Working (0.019319): The job status of the wife has a minor influence on the model's predictions.

Husband_Occupation (0.037459): The model is moderately influenced by the husband's occupation.

Husband_Occupation (0.037459): The model is moderately influenced by the husband's occupation.

Standard_of_living_index (0.018210): The model's judgements are influenced by the standard of living index.

Media_exposure (0.000000): Media exposure has no meaningful impact on the model's predictions.

```
# predictions
train_predicts = decision_tree.predict(X_train)
test_predicts = decision_tree.predict(X_test)

# What are the probabilitues?


decision_proba = decision_tree.predict_proba(X_train)
decision_proba = decision_proba[:, 1]
decision_auc = roc_auc_score(y_train, decision_proba)
print('AUC Train: %.2f' % decision_auc)



decision_proba_test = decision_tree.predict_proba(X_test)
decision_proba_test = decision_proba_test[:, 1]
decision_auc_test = roc_auc_score(y_test, decision_proba_test)
print('AUC Test: %.2f' % decision_auc_test)
```

```
AUC Train: 0.82
AUC Test: 0.70
```

Based on the probabilities predicted by a decision tree model, the given code computes the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) scores for both the training and test datasets. Here's how the findings were interpreted:

Train AUC: 0.82

The training dataset has an AUC-ROC score of 0.82. AUC closer to one indicates that the model performs effectively in differentiating between positive and negative events. An AUC of 0.82 suggests that the training data has strong discriminating power. AUC test result: 0.70

For the test dataset, the AUC-ROC score is 0.70. This is a somewhat lower result than the training AUC, but it is still acceptable. It suggests that the model generalises pretty well to new, previously unknown data. If the test AUC is much lower than the training AUC, this may indicate overfitting.

```
confusion_matrix(y_train, train_predicts)
print(classification_report(y_train, train_predicts))
```

```
              precision    recall  f1-score   support

           0       0.75      0.64      0.69       428
           1       0.75      0.84      0.79       547

    accuracy                           0.75       975
   macro avg       0.75      0.74      0.74       975
```

```
weighted avg       0.75      0.75      0.75       975
```

Interpretation:

Precision: The model properly predicts positive cases 75% of the time and negative instances 76% of the time. Recall (Sensitivity): The model accurately recognises 84% of positive events and 64% of negative instances. F1-score: The harmonic mean of accuracy and recall for positive cases is 0.79, whereas for negative examples it is 0.69. Accuracy: The model's overall accuracy on the training set is 75%. These metrics give a thorough evaluation of the model's performance, taking into account both positive and negative classifications.

Observations:

- For training, we obtained an overall accuracy of 75%.
- Recall is lower for train label of 0, while it is higher for train label of 1.
- Precision is the same for both the labels (75%).

```
confusion_matrix(y_test, test_predicts)
print(classification_report(y_test, test_predicts))
```

```
              precision    recall  f1-score   support

           0       0.65      0.46      0.54       186
           1       0.65      0.81      0.72       232

    accuracy                           0.65       418
   macro avg       0.65      0.63      0.63       418
weighted avg       0.65      0.65      0.64       418
```

Precision: The model can properly predict positive cases 65% of the time and negative instances 65% of the time. Recall (Sensitivity): The model accurately recognises 81% of positive events and 46% of negative instances. F1-score: The harmonic mean of accuracy and recall for positive occurrences is 0.72, whereas for negative examples it is 0.54. Accuracy: The model's overall accuracy on the test set is 65%. These measures reveal how effectively the model generalises to new, previously unknown data and how well it performs on both positive and negative classes.

Observations:

- For testing, we obtained an overall accuracy of 65%.
- Recall is lower for train label of 0, while it is higher for train label of 1.
- Precision is the same for both the labels (65%).

```
# Finally , the model scores
decision_tree.score(X_train, y_train)
```

```
0.7507692307692307
```

The given code computes the decision tree model's accuracy score on the training set. Here's one explanation of the outcome:

75.08% accuracy on the training set

The accuracy score indicates the fraction of properly predicted cases in the training set out of all instances. On the training data, the decision tree model obtains an accuracy of roughly 75.08%. Interpretation:

The accuracy score measures the model's performance on the training set in general. A score of 75.08% means that the model predicts the target variable accurately for around 75.08% of the cases in the training set.

```
decision_tree.score(X_test, y_test)
```

```
0.6507177033492823
```

Logistic Regression

```
# Fit logistic regression model
logistic_regression =
LogisticRegression(solver='sag',max_iter=500,penalty='l2',n_jobs=2)
logistic_regression.fit(X_train, y_train)
```

```
c:\Users\VAIDU\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter
was reached which means the coef_ did not converge
  warnings.warn(
```

```
LogisticRegression(max_iter=500, n_jobs=2, solver='sag')
```

```
# Performance of the model
log_train_predict = logistic_regression.predict(X_train)
log_test_predict = logistic_regression.predict(X_test)
log_test_predict_prob =logistic_regression.predict_proba(X_test)
print(pd.DataFrame(log_test_predict_prob).head())
print(logistic_regression.score(X_train, y_train))
```

```
          0         1
0  0.360199  0.639801
1  0.538724  0.461276
2  0.363293  0.636707
3  0.377847  0.622153
4  0.468149  0.531851
0.6543589743589744
```

Predictions from the Logistic Regression Model:

For both the training and test sets, the model predicts the class labels (0 or 1). It also returns the estimated probability for each class. The first row, for example, reveals that the

model predicts a chance of around 36.02% for class 0 and 63.98% for class 1. Model Precision:

The logistic regression model's accuracy score on the training set is roughly 65.44%. The fraction of accurately predicted instances in the training set is represented by this score.

```
# Classification report for train

confusion_matrix(y_train, log_train_predict)
print(classification_report(y_train, log_train_predict))
```

```
              precision    recall  f1-score   support

           0       0.66      0.44      0.53       428
           1       0.65      0.82      0.73       547

    accuracy                           0.65       975
   macro avg       0.66      0.63      0.63       975
weighted avg       0.66      0.65      0.64       975
```

Precision: The precision of the model for class 0 is 66%, suggesting that 66% of the occurrences predicted as class 0 are genuine negatives. The accuracy for class 1 is 65%, which means that 65% of the cases predicted as class 1 are true positives. Recall (Sensitivity): For class 0, the recall is 44%, suggesting that the model correctly recognises 44% of the real class 0 occurrences. The recall for class 1 is 82%, meaning that the model correctly identifies 82% of the class 1 occurrences. F1-score: The harmonic mean of accuracy and recall for class 0 is 0.53 and for class 1 is 0.73. Accuracy: The model's overall accuracy on the training set is 65%, which is the proportion of properly predicted cases out of all instances.

```
confusion_matrix(y_test, log_test_predict)
print(classification_report(y_test, log_test_predict))
```

```
              precision    recall  f1-score   support

           0       0.65      0.37      0.47       186
           1       0.62      0.84      0.72       232

    accuracy                           0.63       418
   macro avg       0.64      0.61      0.59       418
weighted avg       0.64      0.63      0.61       418
```

Precision: The precision of the model for class 0 is 66%, suggesting that 66% of the occurrences predicted as class 0 are genuine negatives. The accuracy for class 1 is 65%, which means that 65% of the cases predicted as class 1 are true positives. Recall (Sensitivity): For class 0, the recall is 44%, suggesting that the model correctly recognises

44% of the real class 0 occurrences. The recall for class 1 is 82%, meaning that the model correctly identifies 82% of the class 1 occurrences. F1-score: The harmonic mean of accuracy and recall for class 0 is 0.53 and for class 1 is 0.73. Accuracy: The model's overall accuracy on the training set is 65%, which is the proportion of properly predicted cases out of all instances.

Observations:

- Overall train accuracy is obtained as 65%, while overall test accuracy obtained as 63%.
- The model is producing many false positives, which are the cases where the model predicts the outcome as true or yes, but the actual outcome is false or no.

Linear Discriminant Analysis

```
# Fit LDa model
# -------------------------
lda = LinearDiscriminantAnalysis()
lda_model = lda.fit(X_train, y_train)
lda_model
```

```
LinearDiscriminantAnalysis()
```

```
df = pd.DataFrame(lda_model.coef_, columns=X_train.columns)

df = df.round(2)
print(df)
```

```
   Wife_age  Wife_ education  Husband_education  No_of_children_born
0     -0.08             0.19               0.18                 0.25  \

   Wife_religion  Wife_Working  Husband_Occupation  Standard_of_living_index
0          -0.67         -0.12               -0.05                     -0.02
\

   Media_exposure
0           -1.22
```

A positive coefficient indicates that the chance of class 1 (response variable) increases as the related attribute increases. A negative coefficient indicates that the chance of class 1 is decreasing while the associated characteristic is increasing. Features with higher absolute values contribute more to class differentiation. In this scenario, the coefficients for "No_of_children_born" and "Wife_education" are positive, suggesting that an increase in both variables is related with a higher chance of the response variable being class 1. However, "Media_exposure" and "Wife_religion" exhibit negative coefficients, implying that increasing these factors lowers the chance of class 1.

Observations:

- From the model coefficients, we can see that No_of_children_born is having the highest value, which means that it helps the best for classifying the Contraceptive used.
- Media exposure and Wife's religion has the least coefficient value

```
# Prediction
lda_predict_train = lda_model.predict(X_train)
lda_predict_test = lda_model.predict(X_test)

# Classification report
print(classification_report(y_train, lda_predict_train))
```

```
              precision    recall  f1-score   support

           0       0.67      0.44      0.53       428
           1       0.66      0.83      0.73       547

    accuracy                           0.66       975
   macro avg       0.66      0.64      0.63       975
weighted avg       0.66      0.66      0.64       975
```

```
print(classification_report(y_test, lda_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.65      0.37      0.47       186
           1       0.62      0.84      0.72       232

    accuracy                           0.63       418
   macro avg       0.64      0.60      0.59       418
weighted avg       0.63      0.63      0.61       418
```

Observations:

- LDA performed similarly to the logistic regression model.
- Overall accuracy for training was 66%, while for testing it was 63%
- From the analysis, we can conclude that No_of_children_born has a highest effect on Contraceptive used.

Conclusion:

In conclusion, the contraceptive dataset analysis showed significant patterns in system activity measurements, highlighting possible relationships between different parameters. Contraceptive dataset exploration, on the other hand, highlighted the importance of characteristics such as wife age, education, and media exposure in determining contraceptive technique choices. The machine learning models offered predictive skills, with Decision Trees outperforming the others. These findings add to our understanding of system behavior and demographic impacts on contraceptive decisions.