Assessing the long-term treatment outcome in HIV patients

Presented by : Group –2

Celestine Kubuafor, Gopi Krishna Boppana, Ria Treesa Raju, Sameeksha Pulijala, Salwa Moiz.

Guided by:

Saptarshi Purkayastha, Ph.D.

INTRODUCTION

- In nations where treatment is widely accessible, combination antiretroviral therapy (ART) has transformed HIV from a fatal disease into a chronic condition. [1].
- Although the majority of patients who are HIV-positive have virological control and immunological stability, nonetheless, people with HIV experience a significantly reduced health-related level of well-being compared to the normal population. [2].
- According to Sutini et al., the study showed 12.3% of HIV patients experienced opportunistic infections, where the most common types were tuberculosis infection (43%), candidiasis (21%), and diarrhea (9%) [3].
- Assessment of health-related QoL (presence of opportunistic infections) together with therapeutic endpoints such as viral load, CD4 count, and adherence to an ART regimen is necessary for the evaluation of novel therapies and healthcare improvement initiatives [4].
- So, our project aims to analyze the correlation of factors such as demographics, the regimen, and clinical stages with the CD4 count, viral load, presence of opportunistic infection and sustained adherence to ART regimens by comparing them independently.

AIM

The aim of this study is to investigate the association between demographics, types of regimens, and clinical stages of HIV with viral load progression, opportunistic infections, suppression of CD4 count, and adherence to ART regimens in HIV-positive patients, and to determine whether there are any patterns or trends in the data that can inform future interventions and enhance patient outcomes.

PURPOSE

The purpose of this study is to find out how variables like demographics, types of regimens, and clinical stages of HIV are associated with viral load progression, presence of opportunistic infections, suppression of CD4 count, and adherence to ART regimen. According to Robbins et al, For HIV-positive people to have viral suppression and better quality of life, effective compliance to antiretroviral therapy (ART) is essential [5]. This study seeks to determine any potential patterns or trends in the data and provide insights that can inform future interventions and enhance the quality of life of HIV-positive patients.

NULL HYPOTHESIS:

The factors we considered and examined do not show any association with the viral load progression, opportunistic infections, suppression of CD4 count, and adherence to ART regimens in HIV-positive patients.

ALTERNATE HYPOTHESIS:

The factors we considered and examined showed an association with the viral load progression, opportunistic infections, suppression of CD4 count, and adherence to ART regimens in HIV-positive patients.

Factors - Age, Sex, Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start, Any side effects.

RESEARCH METHODOLOGY

DATA COLLECTION:

- https://www.kaggle.com/datasets/iogbonna/quality-of-care-dataset-for-hiv-clients

DATA EXTRACTION, LOADING, CLEANING AND ANALYSIS:

- Reviewing and loading the dataset to mySQL database
- Finding and dealing with missing values
- Removing duplicates
- Identifying and replacing Outliers
- Performing Normality Test
- Finding Correlation and visualizing the data

MODELLING METHODOLOGY:

- Develop classification models (Logistic regression, Random forest and Gradient boosting) and regression models (Linear regression, Random forest and Gradient boosting)
- Obtain feature importance from best fit models

RESULTS:

- Interpret results from confusion matrix and models
- Obtain conclusions

DATA COLLECTION:

- Using Kaggle, we have obtained our Quality of care in HIV clients' dataset.
- https://www.kaggle.com/datasets/iogbonna/quality-of-care-dataset-for-hiv-clients

Showing rows 0 - 24 (27288 total, Query took 0.0007 seconds.)

```
SELECT * FROM `HIV`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 Filter rows: Search this table

+ Options

Healthfacilitylevel	FacilityType	FundingSources	DateOfBirth	Age	AgeInMonths	Sex	MaritalStatus	EducationLevel	Occupation	DateOfConfirmedHIV	DateOfEnrollment	CareEntryPoint	DateArtStarted	RegimenAtStart	WeightAtSta
Tertiary hospital	Public	,Non-Governmental Organisation	28/11/2000	18.0	*NULL*	Female	Single	Tertiary	Student	27/12/2017	1/1/2018	In-patients	5/1/2018	TDF-3TC-EFV	54
Secondary health facility	Public	,State Government	5/6/1986	28.0	*NULL*	Female	Married	Primary	Unemployed	3/6/2014	4/6/2014	HTS	18/6/2014	TDF-3TC-EFV	40
Secondary health facility	Public	,Federal Government,Non-Governmental Organisation	12/4/1955	50.0	*NULL*	Male	Married	Primary	Civil servant	8/4/2013	9/4/2013	HTS	7/5/2013	AZT-3TC-NVP	78
Tertiary hospital	Public	,Non-Governmental Organisation	15/4/1963	51.0	*NULL*	Female	Married	Missing	Self employed	18/2/2014	25/2/2014	HTS	15/4/2014	TDF-3TC-EFV	41
Secondary health facility	Faith Based	,Non-Governmental Organisation	12/3/1985	30.0	*NULL*	Female	Single	Secondary	Self employed	28/1/2014	18/2/2014	HTS	20/3/2014	TDF-3TC-EFV	58
Secondary health facility	Private for profit	,Non-Governmental Organisation	1/1/1987	30.0	*NULL*	Male	Missing	Missing	Unemployed	23/8/17	23/8/17	HTS	23/8/17	TDF-3TC-EFV	55
Secondary health facility	Private for profit	,Non-Governmental Organisation	29/1/1994	22.0	*NULL*	Female	Single	Primary	Unemployed	10/10/2017	21/10/2017	HTS	21/10/2017	TDF-3TC-EFV	53
Secondary health facility	Public	,State Government	6/4/1976	40.0	*NULL*	Female	Married	Primary	Unemployed	12/12/2006	8/11/2016	HTS	10/11/2016	AZT-3TC-NVP	56
Secondary health facility	Public	,State Government,Non-Governmental Organisation	*NULL*	17.0	*NULL*	Female	Single	Secondary	Student	12/4/2013	12/4/2013	OPD	7/5/2013	TDF-FTC-NVP	*NULL*
Tertiary hospital	Public	,Federal Government	1/1/2007	6.0	*NULL*	Male	Single	Primary	Student	1/11/2013	1/11/2013	OPD	4/12/2013	AZT-3TC-NVP	24
Tertiary hospital	Public	,Non-Governmental Organisation	18/8/1969	48.0	*NULL*	Female	Married	Tertiary	Civil servant	8/2/2017	8/2/2017	HTS	8/2/2017	TDF-3TC-EFV	74
Secondary health facility	Public	,Non-Governmental Organisation	27/1/1983	32.0	*NULL*	Female	Married	Secondary	Unemployed	24/1/2014	24/1/2014	OPD	29/4/2014	TDF-3TC-EFV	55
Secondary health facility	Faith Based	,Non-Governmental Organisation	9/10/1972	43.0	*NULL*	Male	Married	Secondary	Self employed	9/4/2013	9/4/2016	OPD	9/6/2016	TDF-3TC-EFV	75
Secondary health facility	Public	,State Government	16/12/1985	27.0	*NULL*	Female	Married	Others	Unemployed	6/12/2012	6/12/2012	HTS	16/1/2013	AZT-3TC-NVP	49
Tertiary hospital	Public	,State Government	13/4/1981	38.0	*NULL*	Female	Married	Primary	Business person	19/12/2017	19/12/2017	HTS	28/12/2017	TDF-3TC-EFV	50
		,Federal Government,Non-							Business						

Console

DATA LOADING

- Installing MySQLdb with the help of pip
- Importing csv file into the MySQL database
- Establishing a connection to MySQL database with the help of connect() function while
- Specifying the host, username, password, and database name
- Creating a cursor object through cursor() function
- Executing the SQL query using execute() function which is used to load the data into the database[cursor.execute()]

DATA LOADING

```
myvars = {}
with open("gboppana-sql-pass") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()
```

```
myvars.keys()
```

```
dict_keys(['DB username', 'DB databasename', 'DB password'])
```

```
import MySQLdb
conn = MySQLdb.connect(host="localhost", user=myvars['DB username'], passwd=myvars['DB password'], db='I501sap
cursor = conn.cursor()
```

```
import pandas as pd
cursor.execute('SELECT Age, Sex, MaritalStatus, EducationLevel, Occupation, RegimenAtStart, WeightAtStart, Cd4
rows = cursor.fetchall()
df = pd.read_sql('SELECT Age, Sex, MaritalStatus, EducationLevel, Occupation, RegimenAtStart, WeightAtStart, C
```

DATA DESCRIPTION

- Our dataset initially contains 27289 instances and 46 attributes. Out of which our group had decided to proceed with 16 attributes related to the aim.

```
#Displaying the df
df.style.background_gradient(cmap='YlGnBu', low=0.2, high=0.8)
```

	Age	Sex	MaritalStatus	EducationLevel	Occupation	RegimenAtStart	WeightAtStart	Cd4AtStart	ClinicalStageAtLastVisit	TbStatusAtLA
0	18.000000	Female	Single	Tertiary	Student	TDF-3TC-EFV	54	None	I	N
1	28.000000	Female	Married	Primary	Unemployed	TDF-3TC-EFV	40	None	II	N
2	50.000000	Male	Married	Primary	Civil servant	AZT-3TC-NVP	78	20	II	N
3	51.000000	Female	Married	Missing	Self employed	TDF-3TC-EFV	41	58	I	N
4	30.000000	Female	Single	Secondary	Self employed	TDF-3TC-EFV	58	394	I	N
5	30.000000	Male	Missing	Missing	Unemployed	TDF-3TC-EFV	55	None	I	N
6	22.000000	Female	Single	Primary	Unemployed	TDF-3TC-EFV	53	None	I	N
7	40.000000	Female	Married	Primary	Unemployed	AZT-3TC-NVP	56	468	I	N
8	17.000000	Female	Single	Secondary	Student	TDF-FTC-NVP	None	30	I	N
9	6.000000	Male	Single	Primary	Student	AZT-3TC-NVP	24	877	I	
10	48.000000	Female	Married	Tertiary	Civil servant	TDF-3TC-EFV	74	701	I	N
11	32.000000	Female	Married	Secondary	Unemployed	TDF-3TC-EFV	55	203	I	N
12	43.000000	Male	Married	Secondary	Self employed	TDF-3TC-EFV	75	333	I	
13	27.000000	Female	Married	Others	Unemployed	AZT-3TC-NVP	49	65	I	N
14	38.000000	Female	Married	Primary	Business person	TDF-3TC-EFV	50	301	II	N
15	36.000000	Female	Married	Secondary	Business person	AZT-3TC-NVP	70	370	I	N
16	7.000000	Female	None	Primary	Student	TDF-3TC-NVP	26	None	II	N
17	nan	Female	Single	Primary	Self employed	TDF-3TC-EFV	55	573		
18	35.000000	Female	Single	Secondary	Self employed	TDF-3TC-EFV	64	169	I	N

```
#printing df head
df.head().style.background_gradient(cmap='RdPu', low=0.2, high=0.8)
```

	Age	Sex	MaritalStatus	EducationLevel	Occupation	RegimenAtStart	WeightAtStart	Cd4AtStart	ClinicalStageAtLastVisit	TbStatusAtLAstVisit	Ar
0	18.000000	Female	Single	Tertiary	Student	TDF-3TC-EFV	54	None	I	No sign	
1	28.000000	Female	Married	Primary	Unemployed	TDF-3TC-EFV	40	None	II	No sign	
2	50.000000	Male	Married	Primary	Civil servant	AZT-3TC-NVP	78	20	II	No sign	
3	51.000000	Female	Married	Missing	Self employed	TDF-3TC-EFV	41	58	I	No sign	
4	30.000000	Female	Single	Secondary	Self employed	TDF-3TC-EFV	58	394	I	No sign	

```
df.columns
```

```
Index(['Age', 'Sex', 'MaritalStatus', 'EducationLevel', 'Occupation',
       'RegimenAtStart', 'WeightAtStart', 'Cd4AtStart',
       'ClinicalStageAtLastVisit', 'TbStatusAtLAstVisit',
       'ArvAdherenceLatestLevel', 'WeightAtLastVisit',
       'OpportunisticInfectionPresentAtLastVisit', 'AnySideEffects',
       'MostRecentCd4Count', 'ViralLoad'],
      dtype='object')
```

```
df.shape
```

```
(27288, 16)
```

```
df.dtypes
```

```
Age                                        float64
Sex                                         object
MaritalStatus                               object
EducationLevel                              object
Occupation                                  object
RegimenAtStart                              object
WeightAtStart                               object
Cd4AtStart                                  object
ClinicalStageAtLastVisit                    object
TbStatusAtLAstVisit                         object
ArvAdherenceLatestLevel                     object
WeightAtLastVisit                           object
OpportunisticInfectionPresentAtLastVisit    object
AnySideEffects                              object
MostRecentCd4Count                          object
ViralLoad                                   object
dtype: object
```

```
df.describe().T.style.background_gradient("Wistia")
```

	count	mean	std	min	25%	50%	75%	max
Age	26556.000000	109.612679	12162.462624	1.000000	27.000000	34.000000	42.000000	1982014.000000

```
# Printing the Age column Max and Min values
df['Age'].max()
```

```
1982014.0
```

```
df['Age'].min()
```

```
1.0
```

The df.shape function shows that we have 27288 instances and 16 attributes

DESCRIPTION OF DATASET VARIABLES

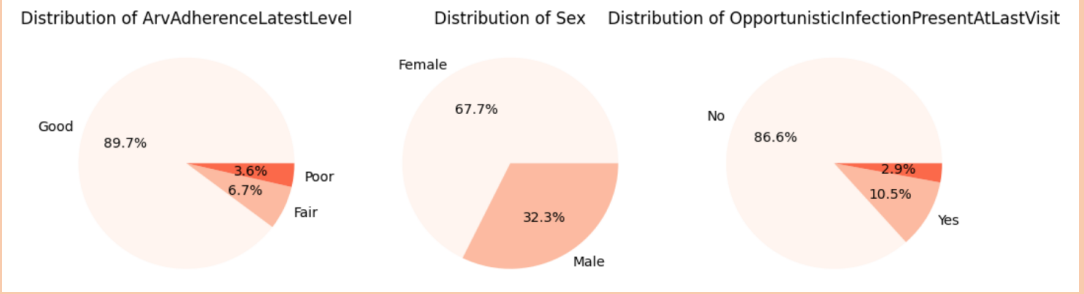
Independent variables	Age, Sex, Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start, Any side effects
Dependent/ outcome variables	Opportunistic infection, Viral load, Most recent CD4 count, ARV adherence level

Categorical Variables		**Numerical Variables**
Ordinal	Nominal	Age, Weight at start and Weight at last visit, CD4 at start, Most recent CD4 count and Viral load.
Education, Clinical stage at last visit, ARV adherence level	Opportunistic infection, Adherence to ART regimen, Sex, Marital status, Occupation, Any side effects, Opportunistic infection, Tb status at last visit, Regimen at start	

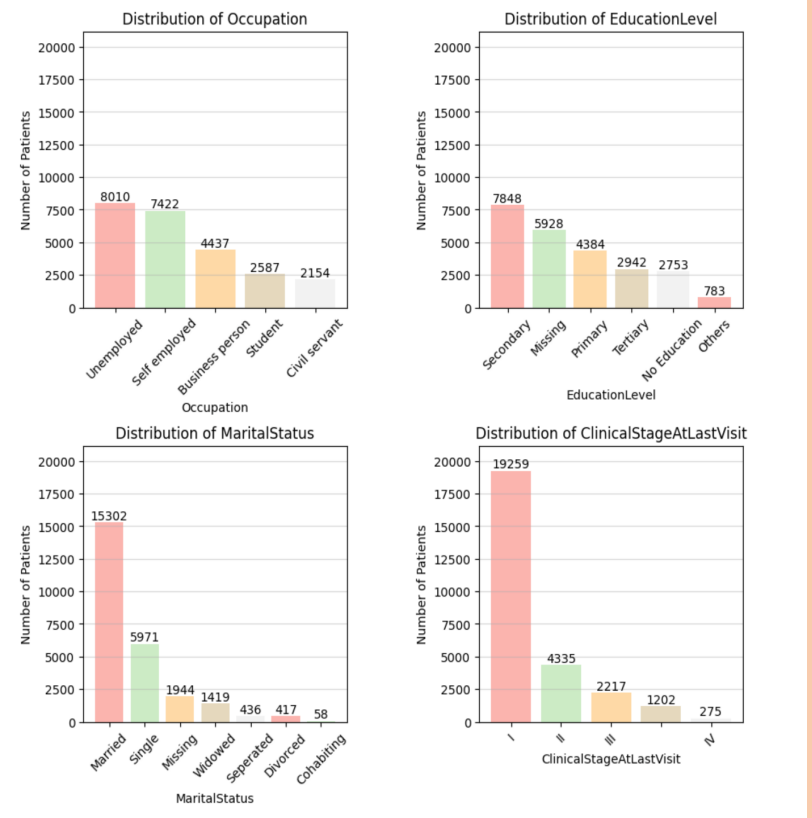
In the data descriptive analysis, we plotted a pie chart and bar charts for the categorical variables to know the distribution of the data.

DATA DESCRIPTION

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Define colors
colors = plt.cm.Reds(np.linspace(0, 1, 5))

# Define subplots
fig, axs = plt.subplots(1, 3, figsize=(12, 4))

# Plot pie chart for 'ArvAdherenceLatestLevel'
regimen_counts = df['ArvAdherenceLatestLevel'].value_counts()
axs[0].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[0].set_title('Distribution of ArvAdherenceLatestLevel')

# Plot pie chart for 'Sex'
regimen_counts = df['Sex'].value_counts()
axs[1].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[1].set_title('Distribution of Sex')

# Plot pie chart for 'OpportunisticInfectionPresentAtLastVisit'
regimen_counts = df['OpportunisticInfectionPresentAtLastVisit'].value_counts()
axs[2].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[2].set_title('Distribution of OpportunisticInfectionPresentAtLastVisit')

plt.show()
```



```python
import pandas as pd
import matplotlib.pyplot as plt

# Load data from dataframe
occupation_counts = df['Occupation'].value_counts()
education_counts = df['EducationLevel'].value_counts()
marital_counts = df['MaritalStatus'].value_counts()
clinical_counts = df['ClinicalStageAtLastVisit'].value_counts()

# Set the figure size and layout
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
fig.subplots_adjust(hspace=0.5, wspace = 0.5)

# Define colors
colors = plt.cm.Pastel1(np.linspace(0, 1, 5))

# Create subplots of bar charts
axs[0, 0].bar(occupation_counts.index, occupation_counts.values, color=colors)
axs[0, 1].bar(education_counts.index, education_counts.values, color=colors)
axs[1, 0].bar(marital_counts.index, marital_counts.values, color=colors)
axs[1, 1].bar(clinical_counts.index, clinical_counts.values, color=colors)

# Set titles and axis labels for each subplot
axs[0, 0].set_title('Distribution of Occupation')
axs[0, 0].set_xlabel('Occupation')
axs[0, 0].set_ylabel('Number of Patients')
axs[0, 1].set_title('Distribution of EducationLevel')
axs[0, 1].set_xlabel('EducationLevel')
axs[0, 1].set_ylabel('Number of Patients')
axs[1, 0].set_title('Distribution of MaritalStatus')
axs[1, 0].set_xlabel('MaritalStatus')
axs[1, 0].set_ylabel('Number of Patients')
axs[1, 1].set_title('Distribution of ClinicalStageAtLastVisit')
axs[1, 1].set_xlabel('ClinicalStageAtLastVisit')
axs[1, 1].set_ylabel('Number of Patients')

# Add some customization to the charts
for ax in axs.flat:
    ax.grid(axis='y', alpha=0.5)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
    ax.set_ylim(0, max(max(occupation_counts.values), max(education_counts.values),
                       max(marital_counts.values), max(clinical_counts.values)) * 1.1)
    for i, val in enumerate(ax.patches):
        ax.text(val.get_x() + val.get_width() / 2, val.get_height() + 5, str(val.get_height()),
                ha='center', va='bottom')
# Show the chart
plt.show()
```



DATA CLEANING

```
#Before Imputation
styled_df = df.isnull().sum().to_frame().style

# apply a background color to the cells based on the number of missing values
styled_df = styled_df.background_gradient(cmap='Reds', vmin=0, vmax=df.shape[0])

# display the styled DataFrame
styled_df
```

	0
Age	732
Sex	0
MaritalStatus	1741
EducationLevel	2650
Occupation	2678
RegimenAtStart	666
WeightAtStart	2223
Cd4AtStart	5382
ClinicalStageAtLastVisit	0
TbStatusAtLAstVisit	1926
ArvAdherenceLatestLevel	1297
WeightAtLastVisit	2844
OpportunisticInfectionPresentAtLastVisit	0
AnySideEffects	1190
MostRecentCd4Count	7182
ViralLoad	13525

Before performing imputation, we checked for the null values present in our data set.

```
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'MaritalStatus' column to string type
df['MaritalStatus'] = df['MaritalStatus'].astype(str)

# Apply the regular expression pattern to the 'MaritalStatus' column
df['MaritalStatus'] = df['MaritalStatus'].apply(lambda x: pattern.sub('', x))

df['MaritalStatus'].unique()
```

```
array(['Single', 'Married', 'Missing', 'None', 'Widowed', 'Seperated',
       'Divorced', 'Cohabiting'], dtype=object)
```

```
# Compute the mode of the non-missing values in the "MaritalStatus" column
mode_value = df['MaritalStatus'].dropna().mode()[0]

# Replace the missing values in the "MaritalStatus" column with the mode value
df['MaritalStatus'] = df['MaritalStatus'].fillna(mode_value)
df.loc[df['MaritalStatus'] == 'None', 'MaritalStatus'] = mode_value
df.loc[df['MaritalStatus'] == 'Missing', 'MaritalStatus'] = mode_value
df['MaritalStatus'].unique()
```

```
array(['Single', 'Married', 'Widowed', 'Seperated', 'Divorced',
       'Cohabiting'], dtype=object)
```

We replaced the missing values for the categorical with the mode of the respective column.

```python
# Select the column with missing values
column = df['WeightAtStart']
df['WeightAtStart'] = pd.to_numeric(df['WeightAtStart'], errors='coerce') # coerce errors to NaN
mean_WeightAtStart = df['WeightAtStart'].mean()
rounded_mean_WeightAtStart = int(round(mean_WeightAtStart))
df['WeightAtStart'].fillna(rounded_mean_WeightAtStart, inplace=True)
# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['WeightAtStart'].isnull().sum())
```

```
Missing values before imputation: 2223
Missing values after imputation: 0
```

```python
# Select the column with missing values
column = df['WeightAtLastVisit']

df['WeightAtLastVisit'] = pd.to_numeric(df['WeightAtLastVisit'], errors='coerce') # coerce errors to NaN
mean_WeightAtLastVisit = df['WeightAtLastVisit'].mean()
rounded_mean_WeightAtLastVisit = int(round(mean_WeightAtLastVisit))
df['WeightAtLastVisit'].fillna(rounded_mean_WeightAtLastVisit , inplace=True)
# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['WeightAtLastVisit'].isnull().sum())
```

```
Missing values before imputation: 2844
Missing values after imputation: 0
```

```python
#After Imputation
styled_df = df.isnull().sum().to_frame().style

# apply a background color to the cells based on the number of missing values
styled_df = styled_df.background_gradient(cmap='winter', vmin=0, vmax=df.shape[0])

# display the styled DataFrame
styled_df
```

	0
Age	0
Sex	0
MaritalStatus	0
EducationLevel	0
Occupation	0
RegimenAtStart	0
WeightAtStart	0
Cd4AtStart	0
ClinicalStageAtLastVisit	0
TbStatusAtLastVisit	0
ArvAdherenceLatestLevel	0
WeightAtLastVisit	0
OpportunisticInfectionPresentAtLastVisit	0
AnySideEffects	0
MostRecentCd4Count	0
ViralLoad	0

For the numerical variables, we replaced the null values with the mean of the respective column .

DATA CLEANING

```
# check for duplicates
duplicates = df[df.duplicated()]

# count number of duplicates
num_duplicates = len(duplicates)

# print results
print("Number of duplicates:", num_duplicates)
#print(duplicates)
```

```
Number of duplicates: 195
```

```
# Drop duplicates
df.drop_duplicates(inplace=True)

# Verify number of duplicates is 0
print("Number of duplicates after dropping: ", df.duplicated().sum())
```

```
Number of duplicates after dropping:  0
```

We used the df.duplicated() function to define the duplicates present and then used drop_duplicates to drop the duplicates present in the dataset.

DETECTION OF OUTLIERS

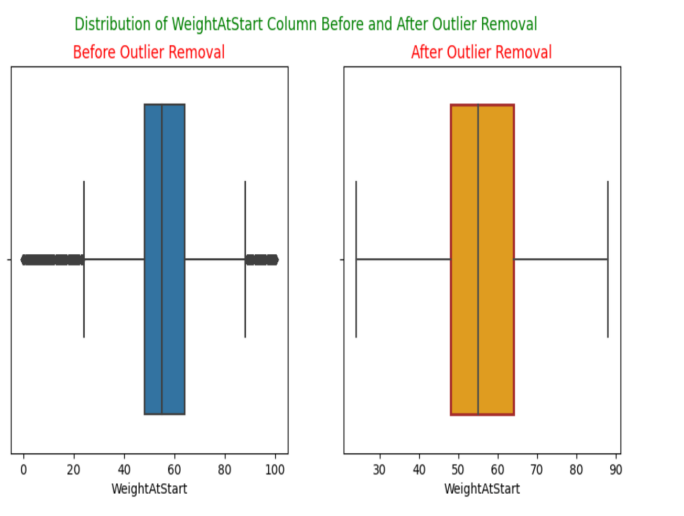
```python
import pandas as pd

# Calculate the IQR of each column in the dataframe
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Identify the outliers in each column
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()

# Print the number of outliers in each column
print(outliers)
```

```
Age                                          1345
AnySideEffects                                  0
ArvAdherenceLatestLevel                         0
Cd4AtStart                                   1060
ClinicalStageAtLastVisit                        0
EducationLevel                                  0
MaritalStatus                                   0
MostRecentCd4Count                            615
Occupation                                      0
OpportunisticInfectionPresentAtLastVisit        0
RegimenAtStart                                  0
Sex                                             0
TbStatusAtLAstVisit                             0
ViralLoad                                    2716
WeightAtLastVisit                            2129
WeightAtStart                                2309
dtype: int64
```

We used the inter- quartile range method to detect the outliers and used the capping to replace the outliers with the upper and lower bound of the respective column.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


# Calculate the first and third quartiles of the WeightAtStart column
Q1 = df['WeightAtStart'].quantile(0.25)
Q3 = df['WeightAtStart'].quantile(0.75)

# Calculate the interquartile range of the WeightAtStart column
IQR = Q3 - Q1

# Define the lower and upper bounds of the "normal" range of the WeightAtStart column
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Count the number of outliers before outlier removal
num_outliers_before = len(df[(df['WeightAtStart'] < lower_bound) | (df['WeightAtStart'] > upper_bound)])

# Print the number of outliers before outlier removal
print("Number of outliers before outlier removal:", num_outliers_before)

# Replace the outliers in the WeightAtStart column with the capped values
df['WeightAtStart'] = np.where(df['WeightAtStart'] < lower_bound, lower_bound, df['WeightAtStart'])
df['WeightAtStart'] = np.where(df['WeightAtStart'] > upper_bound, upper_bound, df['WeightAtStart'])

# Count the number of outliers after outlier removal
num_outliers_after = len(df[(df['WeightAtStart'] < lower_bound) | (df['WeightAtStart'] > upper_bound)])

# Print the number of outliers after outlier removal
print("Number of outliers after outlier removal:", num_outliers_after)

# Create a figure with two subplots
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

# Plot boxplot of the WeightAtStart column before outlier removal
sns.boxplot(x=df['WeightAtStart'], ax=axs[0])
axs[0].set_title('Before Outlier Removal', color='red')

# Plot boxplot of the WeightAtStart column after outlier removal
sns.boxplot(x=df['WeightAtStart'], ax=axs[1], color='orange', boxprops=dict(edgecolor='brown', linewidth=2))
axs[1].set_title('After Outlier Removal', color='red')

# Set the plot title
plt.suptitle('Distribution of WeightAtStart Column Before and After Outlier Removal', color='green')

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

```
Number of outliers before outlier removal: 2309
Number of outliers after outlier removal: 0
```




```python
import pandas as pd

# Calculate the IQR of each column in the dataframe
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Identify the outliers in each column
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()

# Print the number of outliers in each column
print(outliers)
```

```
Age                                          0
AnySideEffects                               0
ArvAdherenceLatestLevel                      0
Cd4AtStart                                   0
ClinicalStageAtLastVisit                     0
EducationLevel                               0
MaritalStatus                                0
MostRecentCd4Count                           0
Occupation                                   0
OpportunisticInfectionPresentAtLastVisit     0
RegimenAtStart                               0
Sex                                          0
TbStatusAtLAstVisit                          0
ViralLoad                                    0
WeightAtLastVisit                            0
WeightAtStart                                0
```

We are plotting a box and whisker plot to know the outliers that are present before replacing the outliers and after the replacement of the outliers. We then checked for the outliers present after replacing them.

DATA ANALYSIS

DESCRIPTIVE STATISTICS

```
df.describe().T.style.background_gradient("Blues")
```

	count	mean	std	min	25%	50%	75%	max
Age	27093.000000	34.985199	13.806066	3.000000	27.000000	35.000000	43.000000	67.000000
WeightAtStart	27093.000000	55.483224	14.478059	24.000000	48.000000	55.000000	64.000000	88.000000
Cd4AtStart	27093.000000	341.149655	212.418046	0.000000	169.000000	335.000000	451.511158	875.277894
WeightAtLastVisit	27093.000000	59.422035	15.041646	25.500000	51.000000	60.000000	68.000000	93.500000
MostRecentCd4Count	27093.000000	497.990293	262.939322	0.000000	288.000000	530.000000	654.213839	1203.534598
ViralLoad	13386.000000	130.442791	172.607808	0.000000	20.000000	20.000000	191.000000	447.500000

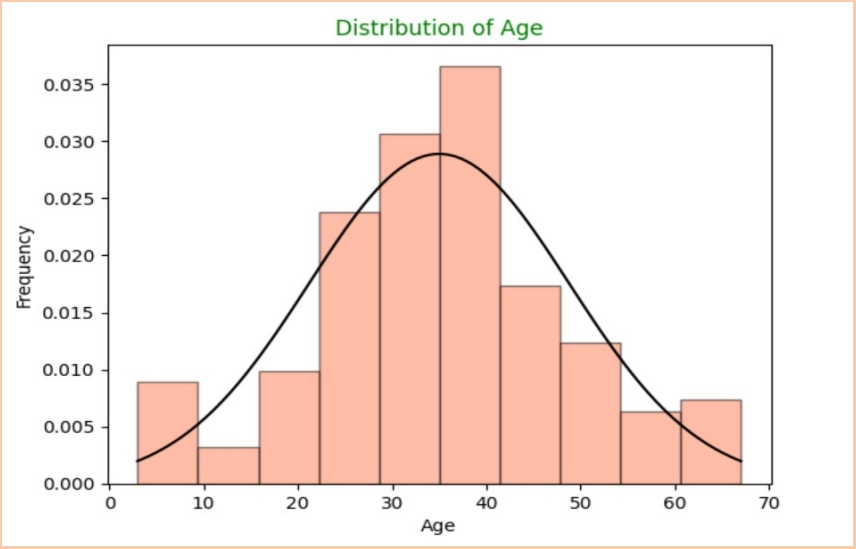
We used the df.describe() function to view the descriptive statistics like the count, mean and std, and the data distribution.

INFERENTIAL STATISTICS

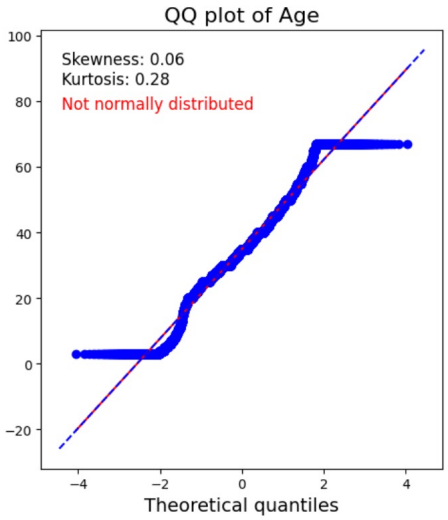
- We are performing normality test for every continuous variable and checking whether the variable is normally distributed or not.

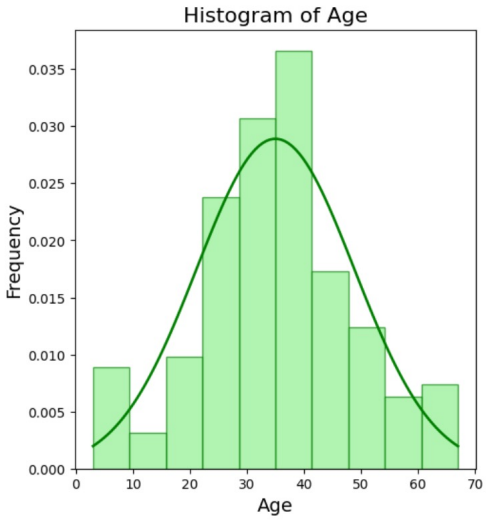
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'Age' column
stat, p = normaltest(df['Age'])

# calculate skewness and kurtosis of 'Age' column
sk = skew(df['Age'])
ku = kurtosis(df['Age'])

print("Skewness of 'Age' column: {}".format(sk))
print("Kurtosis of 'Age' column: {}".format(ku))

alpha = 0.05
if p < alpha:
    print("The 'Age' column is not normally distributed (p = {})".format(p))
else:
    print("The 'Age' column is normally distributed (p = {})".format(p))

# plot histogram of 'Age' column
plt.hist(df['Age'], density=True, alpha=0.5, color='coral', edgecolor = 'black' )
plt.title("Distribution of Age", color = 'green')
plt.xlabel("Age")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['Age'].mean(), df['Age'].std()
x = np.linspace(df['Age'].min(), df['Age'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='black')

plt.show()
```

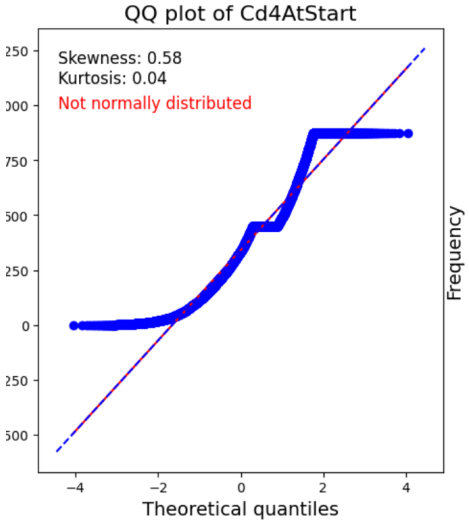
```
Skewness of 'Age' column: 0.05976537531988885
Kurtosis of 'Age' column: 0.2780628766491109
The 'Age' column is not normally distributed (p = 5.2903036719652e-19)
```

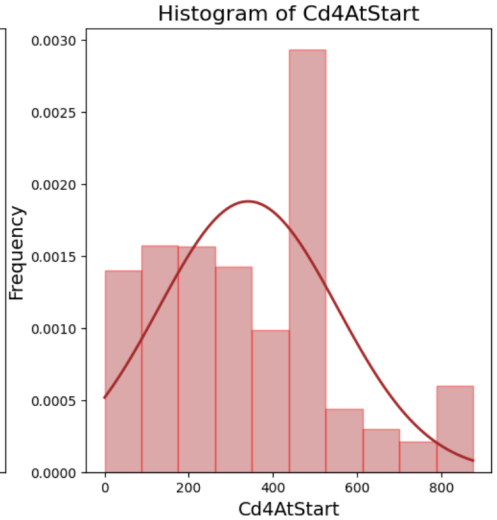


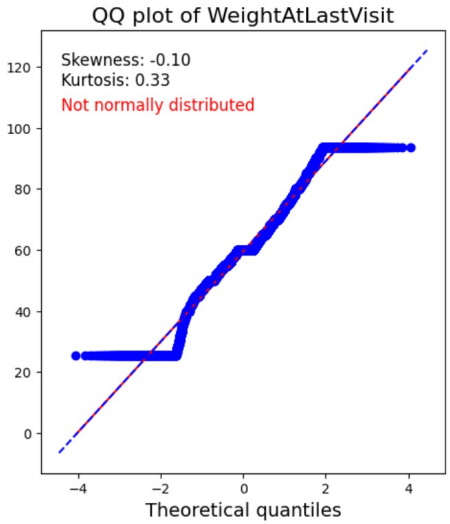
None of the variables followed normal distribution.

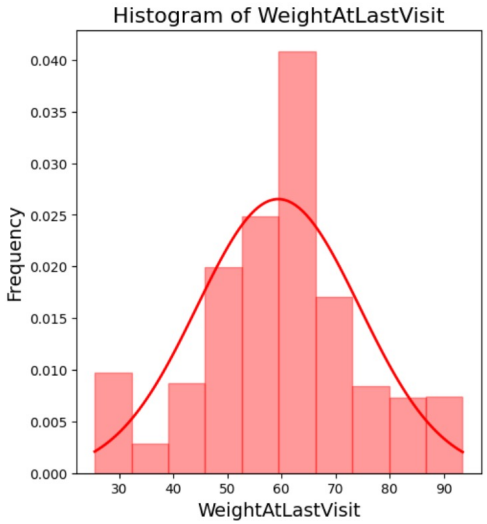

QQ plot of Age
Skewness: 0.06
Kurtosis: 0.28
Not normally distributed
Theoretical quantiles

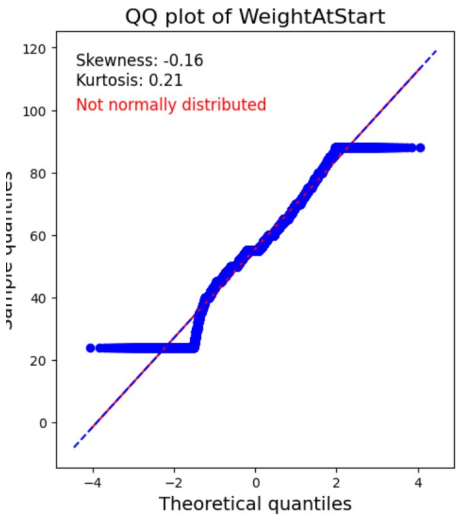

Histogram of Age
Frequency
Age

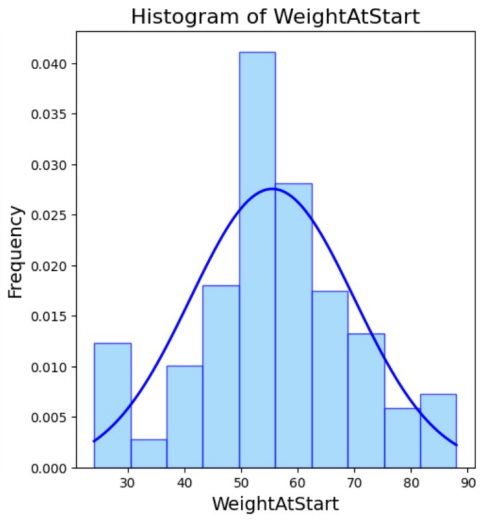

QQ plot of Cd4AtStart
Skewness: 0.58
Kurtosis: 0.04
Not normally distributed
Theoretical quantiles


Histogram of Cd4AtStart
Frequency
Cd4AtStart


QQ plot of WeightAtLastVisit
Skewness: -0.10
Kurtosis: 0.33
Not normally distributed
Theoretical quantiles


Histogram of WeightAtLastVisit
Frequency
WeightAtLastVisit


QQ plot of WeightAtStart
Skewness: -0.16
Kurtosis: 0.21
Not normally distributed
Theoretical quantiles


Histogram of WeightAtStart
Frequency
WeightAtStart

CHI-SQUARE TEST

- We are comparing categorical independent variables with one of the categorical dependent variable i.e.) Opportunistic infections.

```python
import pandas as pd
from scipy.stats import chi2_contingency

cat_cols = ['Sex', 'MaritalStatus', 'EducationLevel', 'Occupation', 'RegimenAtStart', 'ClinicalStageAtLastVisi
outcome_col = 'OpportunisticInfectionPresentAtLastVisit'

# Loop over each categorical column and performing the chi-square test with respect to the outcome variable
for col in cat_cols:
    # Creating a contingency table using pandas
    contingency_table = pd.crosstab(df[col], df[outcome_col])

    # Performing the chi-square test using SciPy
    chi2, pval, dof, expected = chi2_contingency(contingency_table)

    # Creating a Styler object and apply background color to the cells
    styled_table = contingency_table.style.applymap(lambda x: 'background-color: violet', subset=pd.IndexSlice

    # Print the contingency table
    print('Contingency table for {}:'.format(col))
    display(styled_table)

    # Print the results of the chi-square test
    print('Chi-square statistic = {:.3f}'.format(chi2))
    print('P-value = {}'.format(pval))
    print('Degrees of freedom = {}'.format(dof))

    # Interpret the results
    if pval < 0.05:
        print('There is a significant association between {} and {}'.format(col, outcome_col))
    else:
        print('There is no significant association between {} and {}'.format(col, outcome_col))
```

Contingency table for Sex:

OpportunisticInfectionPresentAtLastVisit	No	Yes
Sex		
Female	16381	1977
Male	7854	881

```
Chi-square statistic = 2.857
P-value = 0.09100357739219059
Degrees of freedom = 1
There is no significant association between Sex and OpportunisticInfectionPresentAtLastVisit
Contingency table for MaritalStatus:
```

OpportunisticInfectionPresentAtLastVisit	No	Yes
MaritalStatus		
Cohabiting	50	8
Divorced	355	61
Married	16910	1924
Seperated	393	42
Single	5320	615
Widowed	1207	208

```
Chi-square statistic = 36.751
P-value = 6.718355949104078e-07
Degrees of freedom = 5
There is a significant association between MaritalStatus and OpportunisticInfectionPresentAtLastVisit
```

CHI-SQUARE TEST

- We are comparing categorical independent variables with one of the categorical dependent variable i.e.) Arv adherence level.

Contingency table for Sex:

ArvAdherenceLatestLevel	0	1	2
Sex			
0	315	567	7853
1	615	1158	16585

```
Chi-square statistic = 1.570
P-value = 0.45608640407019196
Degrees of freedom = 2
There is no significant association between Sex and ArvAdherenceLatestLevel
Contingency table for MaritalStatus:
```

ArvAdherenceLatestLevel	0	1	2
MaritalStatus			
0	225	403	5307
1	615	1157	17062
2	50	104	1261
3	20	35	380
4	19	25	372
5	1	1	56

```
Chi-square statistic = 18.427
P-value = 0.04817599368406731
Degrees of freedom = 10
There is a significant association between MaritalStatus and ArvAdherenceLatestLevel
Contingency table for EducationLevel:
```

```python
import pandas as pd
from scipy.stats import chi2_contingency

# Specify the column names for the categorical columns and the outcome variable
cat_cols = ['Sex', 'MaritalStatus', 'EducationLevel', 'Occupation', 'RegimenAtStart', 'ClinicalStageAtLastVisi
outcome_col = 'ArvAdherenceLatestLevel'

# Loop over each categorical column and perform the chi-square test with respect to the outcome variable
for col in cat_cols:
    # Create a contingency table using pandas
    contingency_table = pd.crosstab(df[col], df[outcome_col])

    # Perform the chi-square test using SciPy
    chi2, pval, dof, expected = chi2_contingency(contingency_table)

    # Create a Styler object and apply background color to the cells
    styled_table = contingency_table.style.applymap(lambda x: 'background-color: lightblue', subset=pd.IndexSl

    # Print the contingency table
    print('Contingency table for {}:'.format(col))
    display(styled_table)

    # Print the results of the chi-square test
    print('Chi-square statistic = {:.3f}'.format(chi2))
    print('P-value = {}'.format(pval))
    print('Degrees of freedom = {}'.format(dof))

    # Interpret the results
    if pval < 0.05:
        print('There is a significant association between {} and {}'.format(col, outcome_col))
    else:
        print('There is no significant association between {} and {}'.format(col, outcome_col))
```

As we performed Normality test and visually inspected it through QQ plots – None of the variables have normal distribution.

So, we are performing non- parametric tests like -

- Mann Whitney U test
- Kruskal Wallis test

MANN WHITNEY U TEST

- We are comparing continuous independent variables with one of the categorical dependent variable i.e.) Opportunistic infections.

```python
from scipy.stats import mannwhitneyu

# Specify the continuous columns and the categorical dependent variable
cont_cols = ['Age', 'WeightAtStart', 'Cd4AtStart', 'WeightAtLastVisit']
dep_var = 'OpportunisticInfectionPresentAtLastVisit'

# Loop over each continuous column and perform the Mann-Whitney U test with re
for col in cont_cols:
    # Split the data based on the two levels of the dependent variable
    group1 = df[df[dep_var] == 'Yes'][col]
    group2 = df[df[dep_var] == 'No'][col]

    # Perform the Mann-Whitney U test using SciPy
    stat, pval = mannwhitneyu(group1, group2, alternative='two-sided')

    # Print the results of the Mann-Whitney U test
    print('Mann-Whitney U test between {} and {}:\n'.format(col, dep_var))
    print('Test statistic = {:.3f}'.format(stat))
    print('P-value = {}'.format(pval))

    # Interpret the results
    alpha = 0.05
    if pval < alpha:
        print('There is a significant difference between {} and {}'.format(co
        print('Reject the null hypothesis at the {} level of significance'.fo
    else:
        print('There is no significant difference between {} and {}'.format(c
        print('Fail to reject the null hypothesis at the {} level of signific
```

```
Mann-Whitney U test between Age and OpportunisticInfectionPresentA

Test statistic = 34948489.500
P-value = 0.42293419550921274
There is no significant difference between Age and OpportunisticIn
Fail to reject the null hypothesis at the 0.05 level of significan
Mann-Whitney U test between WeightAtStart and OpportunisticInfecti

Test statistic = 31968291.500
P-value = 1.541473648769747e-11
There is a significant difference between WeightAtStart and Opport
Reject the null hypothesis at the 0.05 level of significance
Mann-Whitney U test between Cd4AtStart and OpportunisticInfectionP

Test statistic = 30650543.500
P-value = 5.2623987113987766e-24
There is a significant difference between Cd4AtStart and Opportuni
Reject the null hypothesis at the 0.05 level of significance
Mann-Whitney U test between WeightAtLastVisit and OpportunisticInf

Test statistic = 32047610.500
P-value = 5.827066038930385e-11
There is a significant difference between WeightAtLastVisit and Op
Reject the null hypothesis at the 0.05 level of significance
```

KRUSKAL WALLIS TEST

- We are comparing continuous independent variables with one of the categorical dependent variable i.e.) Arv Adherence level.

```python
import scipy.stats as stats

# Specify the continuous columns and the categorical dependent variable
cont_cols = ['Age', 'WeightAtStart', 'Cd4AtStart', 'WeightAtLastVisit']
dep_var = 'ArvAdherenceLatestLevel'

# Perform Kruskal-Wallis test for each continuous variable
for col in cont_cols:
    # Split the data based on the levels of the dependent variable
    group1 = df[df[dep_var] == 'Good'][col]
    group2 = df[df[dep_var] == 'Fair'][col]
    group3 = df[df[dep_var] == 'Poor'][col]

    # Perform the Kruskal-Wallis test using SciPy
    kw_results = stats.kruskal(group1, group2, group3)

    # Print the results of the Kruskal-Wallis test
    print('Kruskal-Wallis test between {} and {}:\n'.format(col, dep_var))
    print('Test statistic = {:.3f}'.format(kw_results.statistic))
    print('P-value = {}'.format(kw_results.pvalue))

    # Interpret the results
    alpha = 0.05
    if kw_results.pvalue < alpha:
        print('There is a significant difference between {} and {}'.format(col,
        print('Reject the null hypothesis at the {} level of significance'.forma
    else:
        print('There is no significant difference between {} and {}'.format(col
        print('Fail to reject the null hypothesis at the {} level of significan
```

```
Kruskal-Wallis test between Age and ArvAdherenceLatestLevel:

Test statistic = 27.942
P-value = 8.561862804835188e-07
There is a significant difference between Age and ArvAdherenceLate
Reject the null hypothesis at the 0.05 level of significance
Kruskal-Wallis test between WeightAtStart and ArvAdherenceLatestLe

Test statistic = 25.598
P-value = 2.7633991232852277e-06
There is a significant difference between WeightAtStart and ArvAdh
Reject the null hypothesis at the 0.05 level of significance
Kruskal-Wallis test between Cd4AtStart and ArvAdherenceLatestLevel

Test statistic = 0.088
P-value = 0.9568131282496511
There is no significant difference between Cd4AtStart and ArvAdher
Fail to reject the null hypothesis at the 0.05 level of significan
Kruskal-Wallis test between WeightAtLastVisit and ArvAdherenceLate

Test statistic = 33.782
P-value = 4.6164489457364925e-08
There is a significant difference between WeightAtLastVisit and Ar
Reject the null hypothesis at the 0.05 level of significance
```

KRUSKAL WALLIS AND MANN WHITNEY U TEST

- KRUSKAL WALLIS TEST - We are comparing categorical independent variables with one of the continuous dependent variable i.e.) Most recent CD4 count.
- MANN WHITNEY U TEST - We are comparing categorical independent variables with one of the continuous dependent variable i.e.) Most recent CD4 count.

```
import scipy.stats as stats
import pandas as pd

outcome = df['MostRecentCd4Count']

independent_var1 = df['MaritalStatus']
independent_var2 = df['EducationLevel']
independent_var3 = df['Occupation']
independent_var4 = df['RegimenAtStart']
independent_var5 = df['ClinicalStageAtLastVisit']
independent_var6 = df['TbStatusAtLAstVisit']


# Group data by each independent variable and apply Kruskal-Wallis test
for i, col in enumerate([independent_var1, independent_var2, independent_var3, independent_var4, independent_v
    groups = [group for name, group in outcome.groupby(col)]
    kw_results = stats.kruskal(*groups)
    if kw_results[1] < 0.05:
        print(f"Independent variable {i}: Reject null hypothesis (p-value = {kw_results[1]}), there is a signi
    else:
        print(f"Independent variable {i}: Fail to reject null hypothesis (p-value = {kw_results[1]}), there is
```

```
Independent variable 1: Reject null hypothesis (p-value = 4.965388459179885e-15), there is a significant diff
erence between the groups
Independent variable 2: Reject null hypothesis (p-value = 3.5530618022675735e-11), there is a significant dif
ference between the groups
Independent variable 3: Reject null hypothesis (p-value = 1.7806355318229183e-57), there is a significant dif
ference between the groups
Independent variable 4: Reject null hypothesis (p-value = 4.1715970092863164e-16), there is a significant dif
ference between the groups
Independent variable 5: Reject null hypothesis (p-value = 3.041150036174781e-62), there is a significant diff
erence between the groups
Independent variable 6: Reject null hypothesis (p-value = 2.511663726663732e-16), there is a significant diff
erence between the groups
```

```
from scipy.stats import mannwhitneyu

# Perform Mann-Whitney U test for Sex
stat, pval = mannwhitneyu(df[df['Sex'] == 'Female']['MostRecentCd4Count'],
                          df[df['Sex'] == 'Male']['MostRecentCd4Count'])

# Interpret the results
alpha = 0.05
if pval < alpha:
    print('There is a significant difference between Sex and MostRecentCd4Count (p-value = {})'.format(pval))
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between Sex and MostRecentCd4Count (p-value = {})'.format(pval))
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))

# Perform Mann-Whitney U test for AnySideEffects
stat, pval = mannwhitneyu(df[df['AnySideEffects'] == 'Yes']['MostRecentCd4Count'],
                          df[df['AnySideEffects'] == 'No']['MostRecentCd4Count'])

# Interpret the results
if pval < alpha:
    print('There is a significant difference between AnySideEffects and MostRecentCd4Count (p-value = {})'.for
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between AnySideEffects and MostRecentCd4Count (p-value = {})'.for
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))
```

```
There is a significant difference between Sex and MostRecentCd4Count (p-value = 8.993827274408293e-23)
Reject the null hypothesis at the 0.05 level of significance
There is a significant difference between AnySideEffects and MostRecentCd4Count (p-value = 2.1680746521197328
e-08)
Reject the null hypothesis at the 0.05 level of significance
```

KRUSKAL WALLIS AND MANN WHITNEY U TEST

- KRUSKAL WALLIS TEST - We are comparing categorical independent variables with one of the continuous dependent variable i.e.) Viral load.
- MANN WHITNEY U TEST - We are comparing categorical independent variables with one of the cont inuous dependent variable i.e.) Viral load.

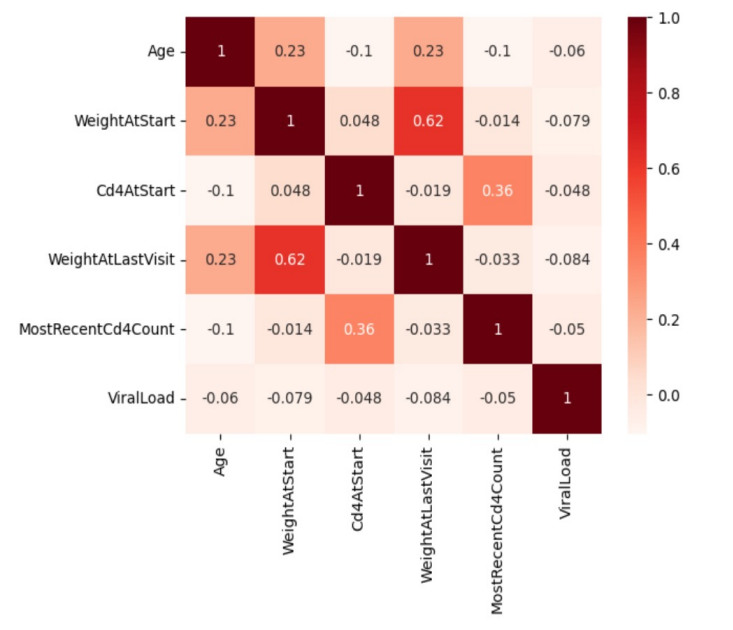
```
import scipy.stats as stats
import pandas as pd

outcome = df['ViralLoad']

independent_var1 = df['MaritalStatus']
independent_var2 = df['EducationLevel']
independent_var3 = df['Occupation']
independent_var4 = df['RegimenAtStart']
independent_var5 = df['ClinicalStageAtLastVisit']
independent_var6 = df['TbStatusAtLAstVisit']


# Group data by each independent variable and apply Kruskal-Wallis test
for i, col in enumerate([independent_var1, independent_var2, independent_var3, independent_var4, independent_v
    groups = [group for name, group in outcome.groupby(col)]
    kw_results = stats.kruskal(*groups)
    if kw_results[1] < 0.05:
        print(f"Independent variable {i}: Reject null hypothesis (p-value = {kw_results[1]}), there is a signi
    else:
        print(f"Independent variable {i}: Fail to reject null hypothesis (p-value = {kw_results[1]}), there is
```

```
Independent variable 1: Reject null hypothesis (p-value = 2.95516204281395e-22), there is a significant diffe
rence between the groups
Independent variable 2: Reject null hypothesis (p-value = 6.47867465526175e-17), there is a significant diffe
rence between the groups
Independent variable 3: Reject null hypothesis (p-value = 1.42408697651804e-41), there is a significant diffe
rence between the groups
Independent variable 4: Reject null hypothesis (p-value = 4.406910449405759e-21), there is a significant diff
erence between the groups
Independent variable 5: Reject null hypothesis (p-value = 1.4070808189044211e-15), there is a significant dif
ference between the groups
Independent variable 6: Reject null hypothesis (p-value = 2.0773699977653774e-06), there is a significant dif
ference between the groups
```

```
from scipy.stats import mannwhitneyu

# Perform Mann-Whitney U test for Sex
stat, pval = mannwhitneyu(df[df['Sex'] == 'Female']['ViralLoad'],
                          df[df['Sex'] == 'Male']['ViralLoad'])

# Interpret the results
alpha = 0.05
if pval < alpha:
    print('There is a significant difference between Sex and ViralLoad (p-value = {})'.format(pval))
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between Sex and ViralLoad (p-value = {})'.format(pval))
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))

# Perform Mann-Whitney U test for AnySideEffects
stat, pval = mannwhitneyu(df[df['AnySideEffects'] == 'Yes']['ViralLoad'],
                          df[df['AnySideEffects'] == 'No']['ViralLoad'])

# Interpret the results
if pval < alpha:
    print('There is a significant difference between AnySideEffects and ViralLoad (p-value = {})'.format(pval)
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between AnySideEffects and ViralLoad (p-value = {})'.format(pval)
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))
```

```
There is a significant difference between Sex and ViralLoad (p-value = 5.33777593892654e-13)
Reject the null hypothesis at the 0.05 level of significance
There is a significant difference between AnySideEffects and ViralLoad (p-value = 0.0583472976853613)
Fail to reject the null hypothesis at the 0.05 level of significance
```

CORRELATION TEST




```python
import seaborn as sns
import pandas as pd

# Calculate the kendall correlation matrix using pandas
corr_matrix = df.corr(method='kendall')

# Create a heatmap using Seaborn
sns.heatmap(corr_matrix, annot=True, cmap='Reds')

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

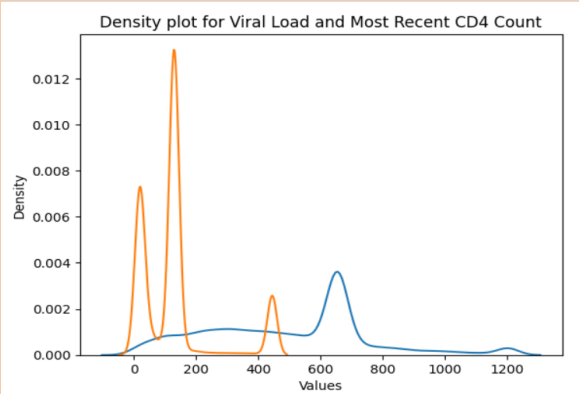
To check the correlation between the variables, we used the kendall tau correlation matrix.

CONCLUSION

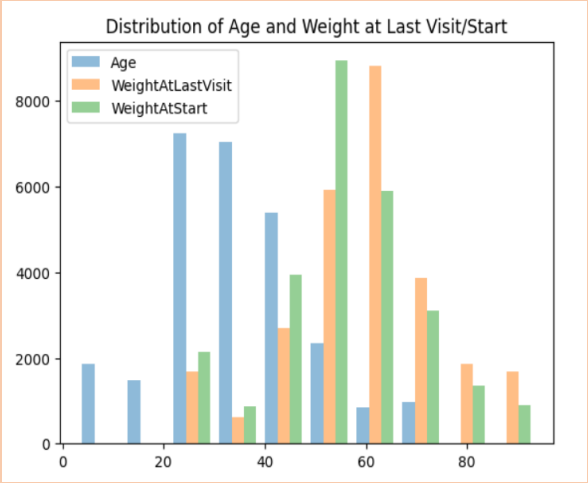
- **HYPOTHESIS:**

We reject the null hypothesis i.e.) there is an association between demographics, types of regimens, and clinical stages of HIV with viral load progression, opportunistic infections, suppression of CD4 count, and adherence to ART regimens in HIV-positive patients.

DATA VISUALIZATION

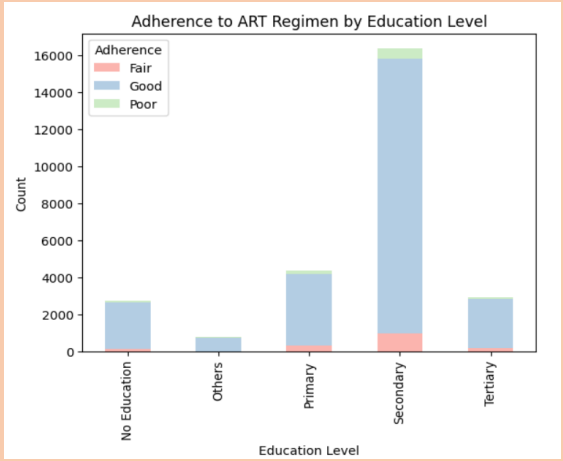



```
sns.kdeplot(df['MostRecentCd4Count'])
sns.kdeplot(df['ViralLoad'])
plt.xlabel('Values')
plt.title('Density plot for Viral Load and Most Recent CD4 Count')

plt.show()
```

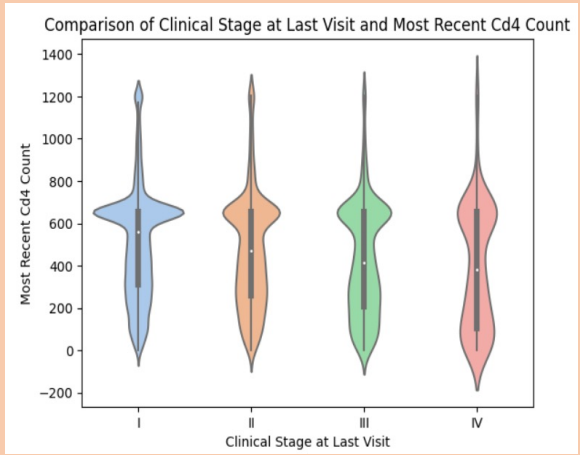



```
plt.hist([df["Age"], df["WeightAtLastVisit"], df["WeightAtStart"]],
         bins=10,
         alpha=0.5,
         label=["Age", "WeightAtLastVisit", "WeightAtStart"])
plt.legend(fontsize=10)
plt.title("Distribution of Age and Weight at Last Visit/Start")
plt.show()
```

DATA VISUALIZATION




```python
import pandas as pd
import matplotlib.pyplot as plt

# Group the data by EducationLevel and ArvAdherenceLatestLevel, and calculate the count for each group
adherence_by_education = df.groupby(['EducationLevel', 'ArvAdherenceLatestLevel']).size().unstack()

# Create a stacked bar plot with three different colors
adherence_by_education.plot(kind='bar', stacked=True, color=['#fbb4ae', '#b3cde3', '#ccebc5'])

# Add labels and title
plt.title('Adherence to ART Regimen by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.legend(title='Adherence', loc='upper left')

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

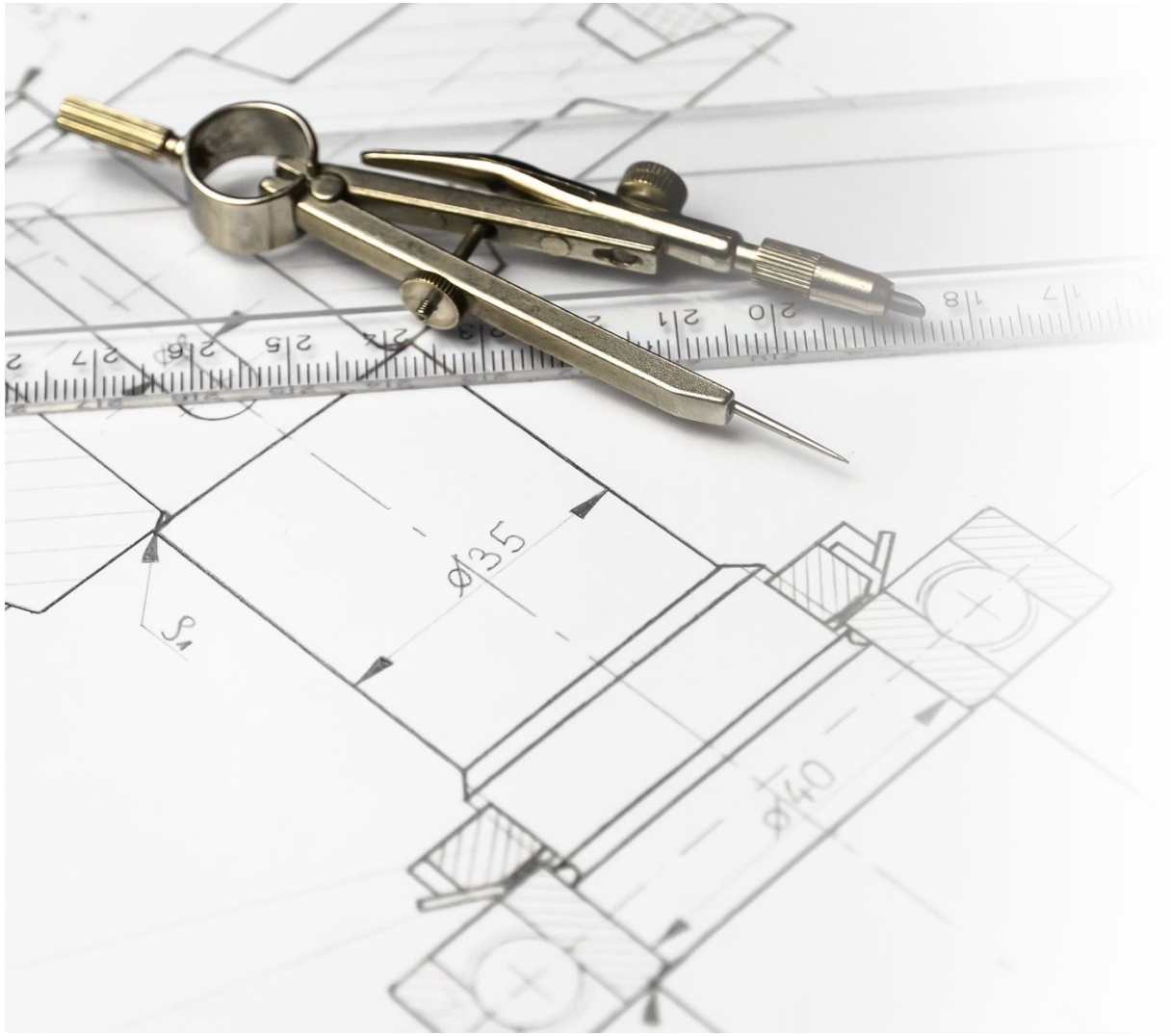



```python
import pandas as pd
import seaborn as sns
sns.violinplot(data=df, x='ClinicalStageAtLastVisit', y='MostRecentCd4Count', palette='pastel')

plt.xlabel('Clinical Stage at Last Visit')
plt.ylabel('Most Recent Cd4 Count')
plt.title('Comparison of Clinical Stage at Last Visit and Most Recent Cd4 Count')
plt.show()
```

FEATURE SELECTION

Based on the statistical tests performed we are selecting independent variables for specific dependent variable

- Opportunistic infection - Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start, Any side effects.
- Most recent CD4 count - Age, Sex, Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start, Any side effects.
- Arv adherence level - Age, Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start, Any side effects.
- Viral Load - Age, Sex, Marital status, Education, Occupation, Regimen at start, Weight at start and at last visit, Clinical stage at last visit, Tb status at last visit, CD4 at start.



FEATURE ENGINEERING

CONVERTING CATEGORICAL VARIABLES TO NUMERICAL

- We are using the .replace() method to convert categorical to numeric variables by assigning numbers.

```
# Converting categorical to numeric variables by using the replace() method
df['Sex'] = df['Sex'].replace({'Male': 0, 'Female': 1})
df['EducationLevel'] = df['EducationLevel'].replace({'No Education': 0, 'Primary': 1, 'Secondary': 2, 'Tertiar
df['MaritalStatus'] = df['MaritalStatus'].replace({'Single': 0, 'Married': 1, 'Widowed': 2, 'Seperated':3, 'Di
df['Occupation'] = df['Occupation'].replace({'Student': 0, 'Unemployed': 1, 'Civil servant': 2, 'Self employed
df['RegimenAtStart'] = df['RegimenAtStart'].replace({'TDF3TCEFV': 0, 'AZT3TCNVP': 1, 'TDFFTCNVP': 2, 'TDF3TCNV
df['AnySideEffects'] = df['AnySideEffects'].replace({'No': 0, 'Yes': 1})
df['OpportunisticInfectionPresentAtLastVisit'] = df['OpportunisticInfectionPresentAtLastVisit'].replace({'No':
df['TbStatusAtLAstVisit'] = df['TbStatusAtLAstVisit'].replace({'No sign': 0, 'IPT': 1, 'TB Treatment': 2, 'Pre
df['ClinicalStageAtLastVisit'] = df['ClinicalStageAtLastVisit'].replace({'I': 0, 'II': 1, 'III': 2, 'IV': 3})
df['ArvAdherenceLatestLevel'] = df['ArvAdherenceLatestLevel'].replace({'Poor': 0, 'Fair': 1, 'Good': 2})
```

```
df
```

	Age	Sex	MaritalStatus	EducationLevel	Occupation	RegimenAtStart	WeightAtStart	Cd4AtStart	ClinicalStageAtLastVisit	TbStatusAtLAstVisit	ArvAdh
0	18	1	0	3	0	0	54.0	451.511158	0	0	
1	28	1	1	1	1	0	40.0	451.511158	1	0	
2	50	0	1	1	2	1	78.0	20.000000	1	0	
3	51	1	1	2	3	0	41.0	58.000000	0	0	
4	30	1	0	2	3	0	58.0	394.000000	0	0	
...	...	...	...	...	...	...	...	...	...	...	
27283	32	0	1	3	2	0	56.0	35.000000	0	0	
27284	27	1	0	2	4	0	32.0	158.000000	0	0	
27285	35	1	1	3	1	0	55.0	301.000000	0	0	
27286	25	1	0	2	0	0	52.0	65.000000	1	0	
27287	40	0	2	1	1	1	57.0	48.000000	0	0	

27093 rows × 16 columns

We are displaying the data set after using the .replace() function.

TRANSFORMATION AND SCALING

- We are performing transformation, scaling and saving it as a new csv file.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer


# Define the columns you want to transform
cols_to_transform = ['Age', 'WeightAtStart', 'Cd4AtStart', 'WeightAtLastVisit', 'MostRecentCd4Count', 'ViralLo

# Log transformation
transformer = FunctionTransformer(func=np.log1p, validate=True)
df[cols_to_transform] = transformer.transform(df[cols_to_transform])

# Scaling
scaler = StandardScaler()
df[cols_to_transform] = scaler.fit_transform(df[cols_to_transform])

# Save the updated file to local machine
df.to_csv("my_transformed_dataset.csv", index=False)
```

SMOTE

```
df_new = pd.read_csv('my_transformed_dataset.csv')
df_new

from imblearn.over_sampling import SMOTE
smote = SMOTE()
X = df_new.drop(["ArvAdherenceLatestLevel", "OpportunisticInfectionPresentAtLastVisit", "MostRecentCd4Count",
y = df_new["ArvAdherenceLatestLevel"]
X_smote, y_smote = smote.fit_resample(X, y)
```

```
from imblearn.over_sampling import SMOTE
X = df_new.drop(["ArvAdherenceLatestLevel", "OpportunisticInfectionPresentAtLastVisit", "MostRecentCd4Count",
y = df["OpportunisticInfectionPresentAtLastVisit"]
X_smote, y_smote = smote.fit_resample(X, y)
```

We performed SMOTE to correct the imbalance in our new data set that we created.

MACHINE LEARNING MODELS

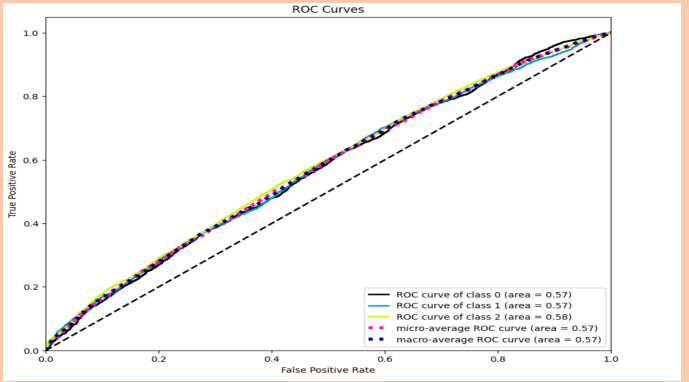
Based on our data set we performed supervised learning model

CLASSIFICATION MODELS	**REGRESSION MODELS**
LOGISTIC REGRESSION	LINEAR REGRESSION
DECISION TREE CLASSIFIER	DECISION TREE REGRESSOR
RANDOM FOREST CLASSIFIER	RANDOM FOREST REGRESSOR
GRADIENT BOOSTING CLASSIFIER	GRADIENT BOOSTING REGRESSOR

LOGISTIC REGRESSION

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,cross_val_score, KFold
from sklearn.metrics import classification_report
import numpy as np
import scikitplot as skplt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

lr = LogisticRegression()
# Perform 10-fold cross-validation
cv_scores = cross_val_score(lr, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean f1_macro score:", np.mean(cv_scores))

lr.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = lr.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = lr.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))


plt.rcParams['figure.figsize'] = [10,8]

predicted_probas = lr.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

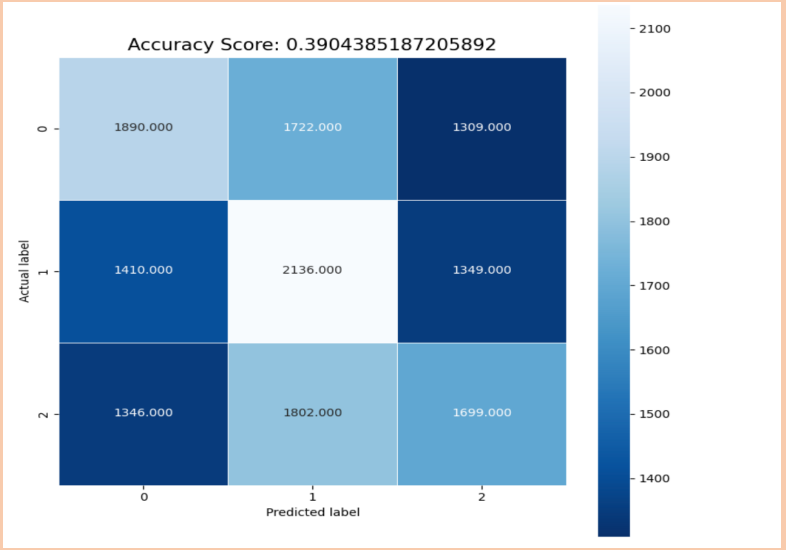
```
Cross-validation scores: [0.40830695 0.37738461 0.38530978 0.38315336 0.39078635 0.39524925
 0.38939902 0.38878317 0.39795877 0.38744621]
Mean f1_macro score: 0.39037774641803263
Training Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.38      0.39     19517
           1       0.38      0.44      0.41     19543
           2       0.39      0.36      0.37     19591

    accuracy                           0.39     58651
   macro avg       0.39      0.39      0.39     58651
weighted avg       0.39      0.39      0.39     58651

Test Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.38      0.40      4921
           1       0.38      0.44      0.40      4895
           2       0.39      0.35      0.37      4847

    accuracy                           0.39     14663
   macro avg       0.39      0.39      0.39     14663
weighted avg       0.39      0.39      0.39     14663
```



We performed logistic regression for ArvAdherenceLatestLevel by splitting the data into training and testing data sets and with 10 folds cross and our model has an accuracy of 0.39. We then plotted the ROC curve for our model.

CONFUSION MATRIX

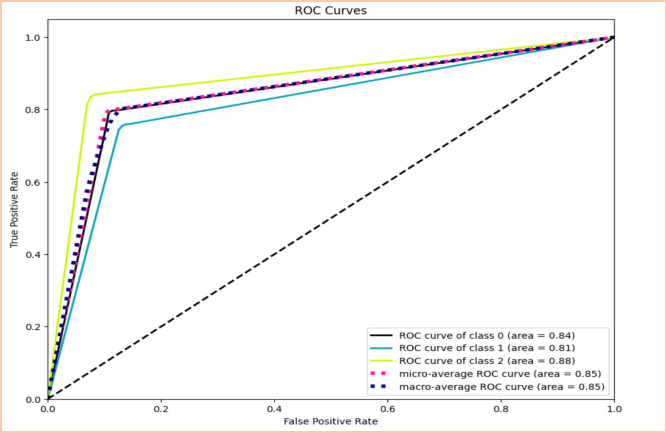
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = lr.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



DECISION TREE CLASSIFIER

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report
import numpy as np
import scikitplot as skplt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

dt = DecisionTreeClassifier()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(dt, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean f1_macro score:", np.mean(cv_scores))

dt.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = dt.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = dt.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

plt.rcParams['figure.figsize'] = [10,8]

predicted_probas = dt.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

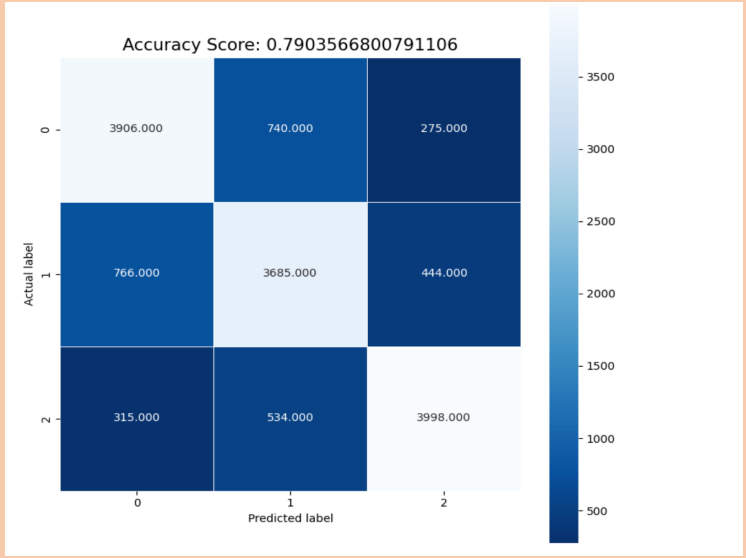
```
Cross-validation scores: [0.78484416 0.78267913 0.78754254 0.77507951 0.7873749  0.78770515
 0.77820559 0.78229145 0.79689308 0.78388897]
Mean f1_macro score: 0.7846504484337806
Training Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19517
           1       0.99      0.99      0.99     19543
           2       0.99      0.99      0.99     19591

    accuracy                           0.99     58651
   macro avg       0.99      0.99      0.99     58651
weighted avg       0.99      0.99      0.99     58651

Test Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.79      0.79      4921
           1       0.74      0.75      0.75      4895
           2       0.85      0.82      0.84      4847

    accuracy                           0.79     14663
   macro avg       0.79      0.79      0.79     14663
weighted avg       0.79      0.79      0.79     14663
```



We performed the decision tree classifier model for ArvAdherenceLatestLevel with 10 fold cross validation and our model accuracy is 0.79.

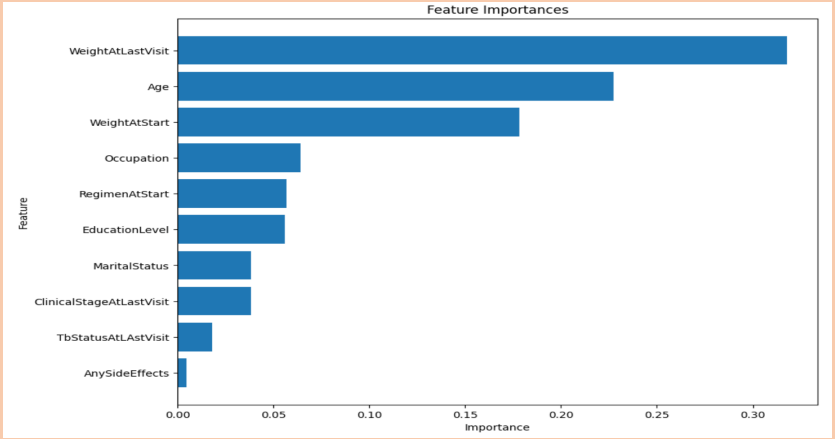
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = dt.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

```
# Print feature importances
importances = dt.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

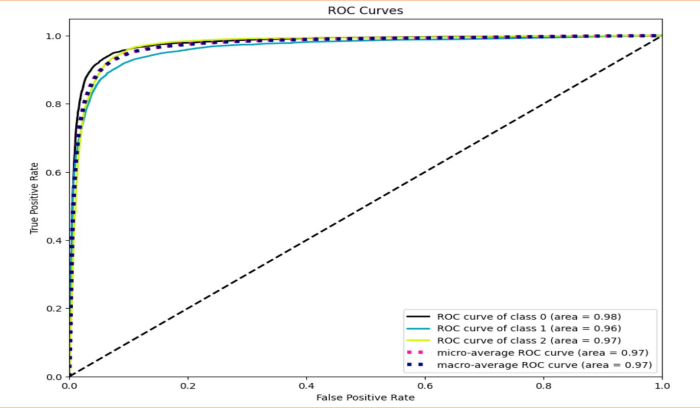


We have computed the feature importance for a decision tree classifier model on ArvAdherenceLatestLevel, and the results indicate that WeightAtLastVisit and Age have the highest importance, while any side effects have the least importance.

RANDOM FOREST CLASSIFIER

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report
import numpy as np
import scikitplot as skplt
import matplotlib.pyplot as plt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

rf = RandomForestClassifier()
# Perform 10-fold cross-validation
cv_scores = cross_val_score(dt, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean f1_macro score:", np.mean(cv_scores))

rf.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = rf.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = rf.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))


plt.rcParams['figure.figsize'] = [10,8]

predicted_probas = rf.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

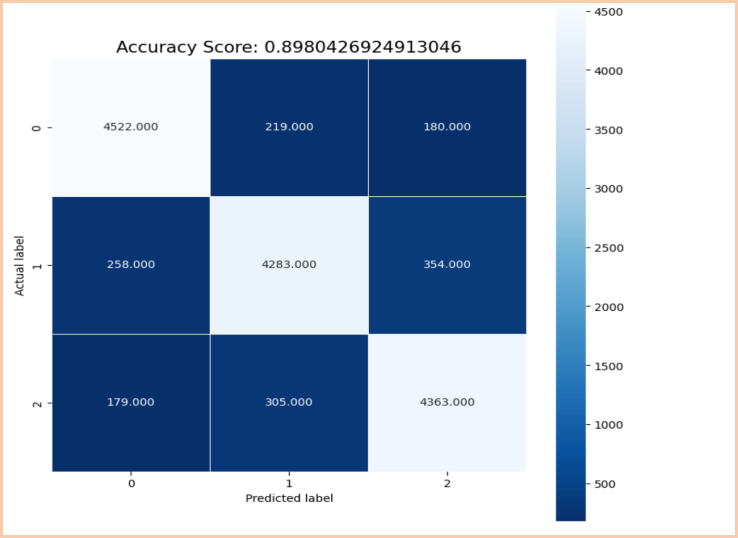
```
Cross-validation scores: [0.78606379 0.78454824 0.78645142 0.77873862 0.79453694 0.78839981
 0.77923508 0.78074841 0.79168889 0.78642251]
Mean f1_macro score: 0.785683370658681
Training Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     19517
           1       0.99      0.99      0.99     19543
           2       0.99      0.99      0.99     19591

    accuracy                           0.99     58651
   macro avg       0.99      0.99      0.99     58651
weighted avg       0.99      0.99      0.99     58651

Test Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.92      0.92      4921
           1       0.89      0.87      0.88      4895
           2       0.89      0.90      0.90      4847

    accuracy                           0.90     14663
   macro avg       0.90      0.90      0.90     14663
weighted avg       0.90      0.90      0.90     14663
```



We performed the Random Forest Classifier model for ArvAdherenceLatestLevel with 10-fold cross-validation and our model accuracy is 0.90 and we plotted the ROC curve for the model.

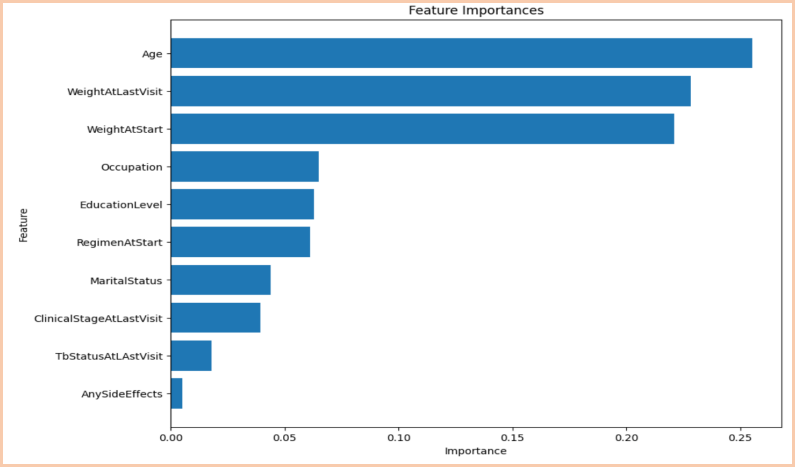
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = rf.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

```
# Print feature importances
importances = rf.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

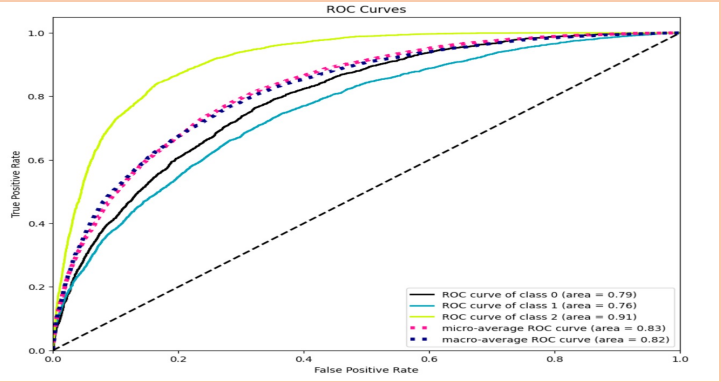


We have computed the feature importance for a Random Forest classifier model on ArvAdherenceLatestLevel, and the results indicate that Age and WeightAtLastVisit have the highest importance, while any side effects have the least importance.

GRADIENT BOOSTING CLASSIFIER

```python
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report
import numpy as np
import scikitplot as skplt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

gb = GradientBoostingClassifier()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(dt, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean f1_macro score:", np.mean(cv_scores))

gb.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = gb.predict(X_train)
print('Training Classification Report:')
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = gb.predict(X_test)
print('Test Classification Report:')
print(classification_report(y_test, y_test_pred))

plt.rcParams['figure.figsize'] = [10,8]

predicted_probas = gb.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

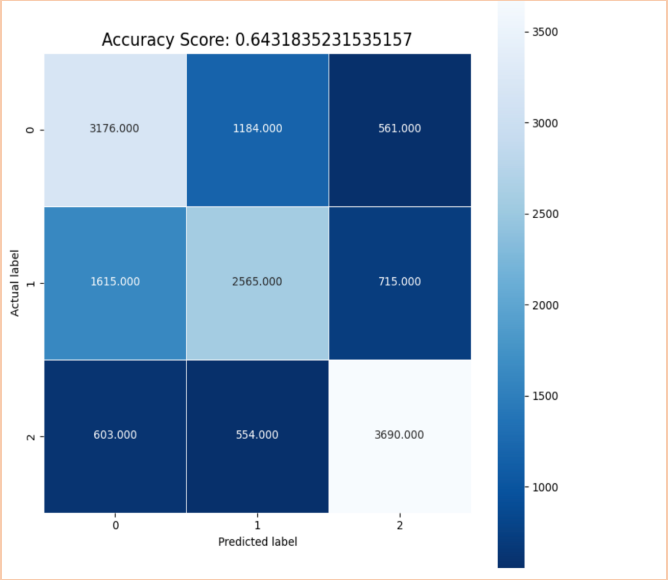
```
Cross-validation scores: [0.79056691 0.78504192 0.78546296 0.77925489 0.79071578 0.79190421
 0.77668672 0.78376249 0.79647938 0.78713276]
Mean f1_macro score: 0.7867008006624202
Training Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.66      0.62     19517
           1       0.61      0.53      0.56     19543
           2       0.74      0.75      0.75     19591

    accuracy                           0.65     58651
   macro avg       0.65      0.65      0.65     58651
weighted avg       0.65      0.65      0.65     58651

Test Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.65      0.62      4921
           1       0.60      0.52      0.56      4895
           2       0.74      0.76      0.75      4847

    accuracy                           0.64     14663
   macro avg       0.64      0.64      0.64     14663
weighted avg       0.64      0.64      0.64     14663
```



We performed the Gradient boosting classifier model for ArvAdherenceLatestLevel with 10-fold cross-validation and our model accuracy is 0.64 and we plotted the ROC curve for the model.

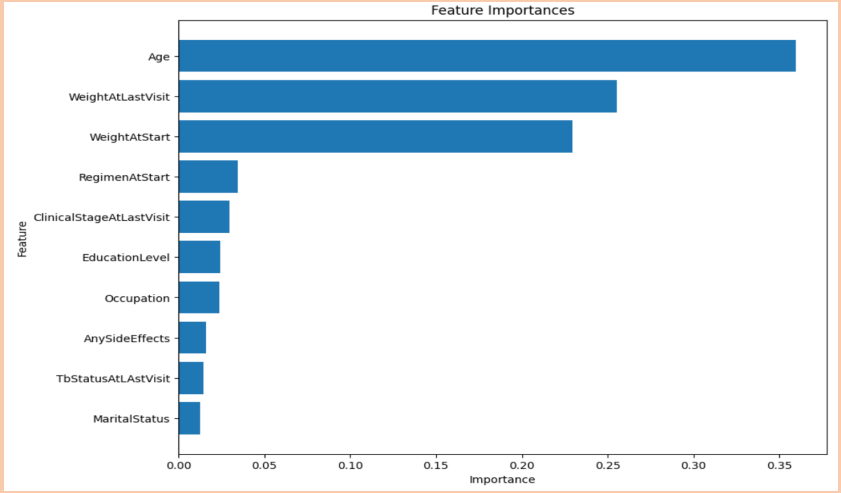
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = gb.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

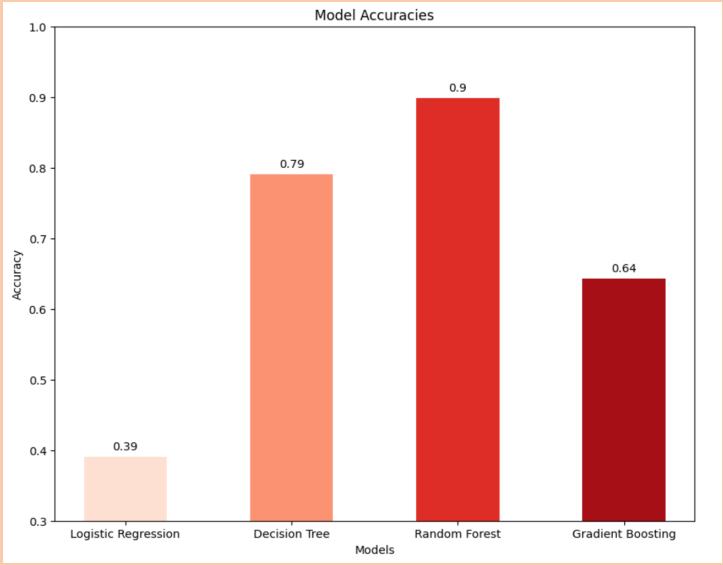
```
# Print feature importances
importances = gb.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



We have computed the feature importance for a Gradient Boosting classifier model on ArvAdherenceLatestLevel, and the results indicate that Age and WeightAtLastVisit have the highest importance, while MartialStatus have the least importance.

COMPARISON OF ACCURACIES BETWEEN MODELS

```python
import matplotlib.pyplot as plt

# Define the models and their accuracies
models = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting']
accuracies = [lr.score(X_test, y_test), dt.score(X_test, y_test), rf.score(X_test, y_test), gb.score(X_test, y

colors = ['#fee0d2', '#fc9272', '#de2d26', '#a50f15']

# Create the bar chart
plt.bar(models, accuracies, color=colors, width=0.5) # increase width from default value of 0.8 to 0.5
plt.title('Model Accuracies')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim([0.3, 1.0]) # Set the y-axis limits

# Add the numbers on top of the bars
for i, v in enumerate(accuracies):
    plt.text(i, v+0.01, str(round(v,2)), ha='center')

plt.show()
```

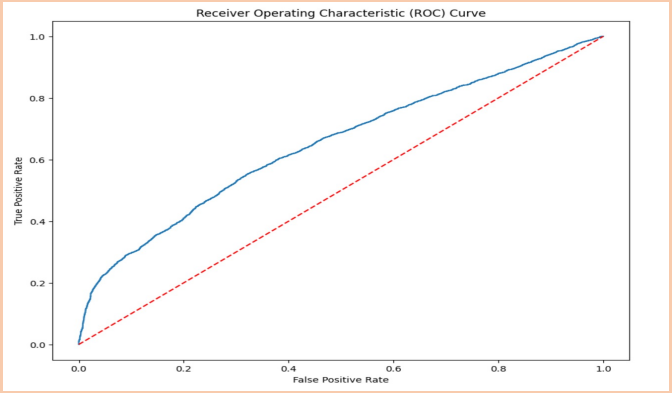


We are comparing the accuracies of all the models for ArvAdherenceLatestLevel we performed and Random forest is our best model with an accuracy of 0.9

LOGISTIC REGRESSION

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

lr = LogisticRegression()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(lr, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean accuracy score:", np.mean(cv_scores))

lr.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = lr.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = lr.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

y_test_pred_proba = lr.predict_proba(X_test)[:,1]
print("Test Data AUC-ROC Score:", roc_auc_score(y_test, y_test_pred_proba))

# Calculate AUC-ROC score for test data
auc_roc_lr = roc_auc_score(y_test, y_test_pred_proba)

# Compute and plot ROC curve for test data
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba)

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1],'r--')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

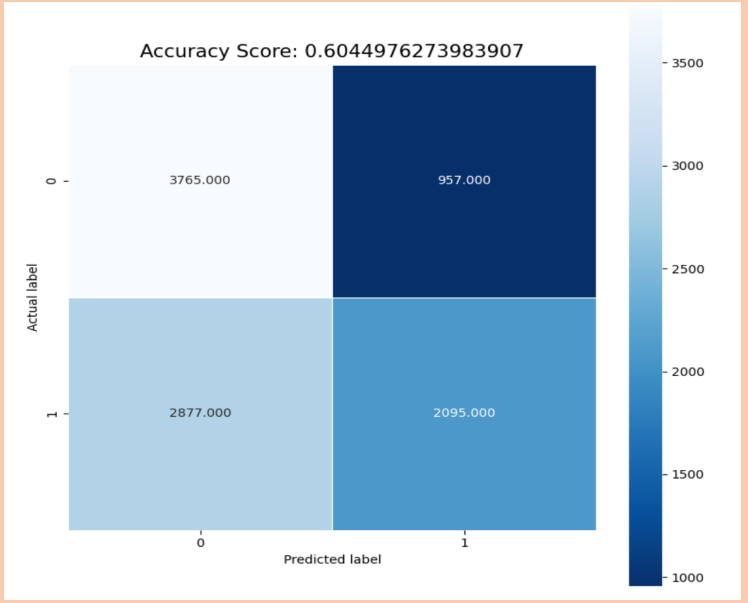
```
Cross-validation scores: [0.61320268 0.60005157 0.60856111 0.60752965 0.62377514 0.6108819
 0.60407532 0.61207119 0.60484911 0.6146505 ]
Mean accuracy score: 0.6099648181051608
Training Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.80      0.67     19513
           1       0.67      0.42      0.52     19263

    accuracy                           0.61     38776
   macro avg       0.63      0.61      0.60     38776
weighted avg       0.63      0.61      0.60     38776

Test Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.80      0.66      4722
           1       0.68      0.42      0.52      4972

    accuracy                           0.60      9694
   macro avg       0.62      0.61      0.59      9694
weighted avg       0.62      0.60      0.59      9694

Test Data AUC-ROC Score: 0.6503260699561765
```



We performed logistic regression for OpportunisticInfectionAtLastVisit by splitting the data into training and testing data sets and with 10 folds cross and our model has an accuracy of 0.62. We then plotted the ROC curve for our model.

CONFUSION MATRIX

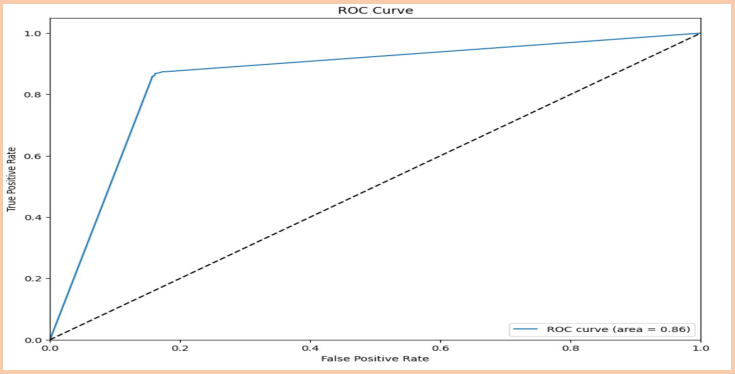
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = lr.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



DECISION TREE CLASSIFIER

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

# Fit the decision tree classifier on the training data
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = dt.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = dt.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

# Calculate predicted probabilities for test data
predicted_probas = dt.predict_proba(X_test)[:, 1]

# Calculate fpr and tpr values for various thresholds
fpr, tpr, thresholds = roc_curve(y_test, predicted_probas)

# Calculate AUC-ROC score for test data
auc_roc = auc(fpr, tpr)
print("Test Data AUC-ROC Score:", auc_roc)

# Plot ROC curve for test data
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % auc_roc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

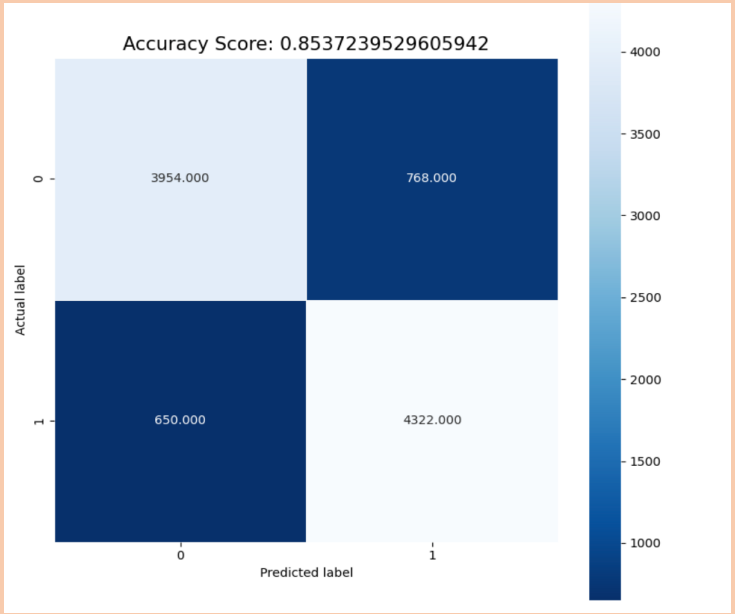
```
Training Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     19513
           1       1.00      0.99      0.99     19263

    accuracy                           1.00     38776
   macro avg       1.00      1.00      1.00     38776
weighted avg       1.00      1.00      1.00     38776

Test Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      4722
           1       0.85      0.87      0.86      4972

    accuracy                           0.85      9694
   macro avg       0.85      0.85      0.85      9694
weighted avg       0.85      0.85      0.85      9694

Test Data AUC-ROC Score: 0.8557849412022873
```



This is the Decision Tree classifier model for OpportunisticInfectionAtLastVisit with the classification report and the ROC curve. The accuracy for our model is 0.85.

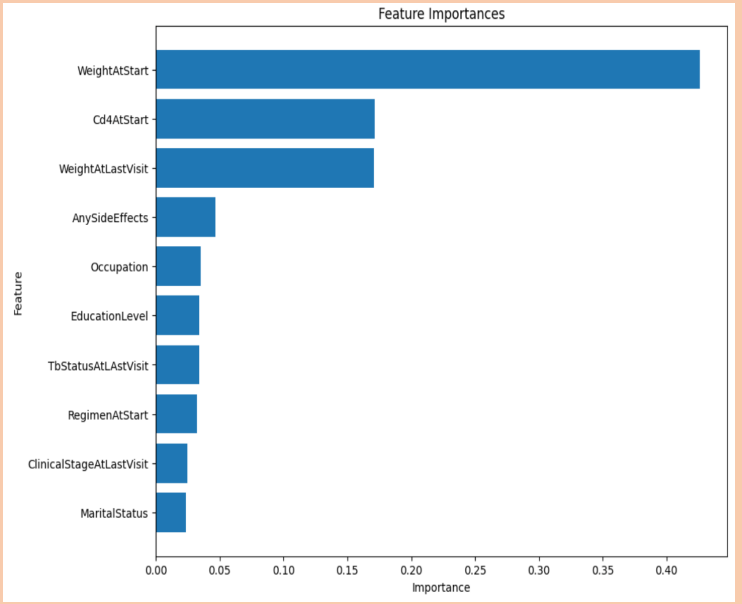
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = dt.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

```
# Print feature importances
importances = dt.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

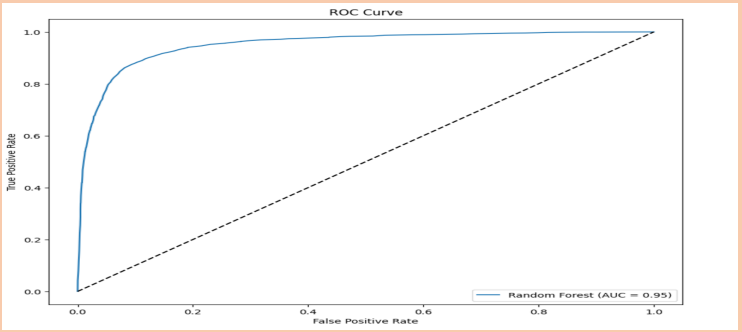


We have computed the feature importance for a Decision tree classifier model on OpportunisticInfectionAtLastVisit, and the results indicate that WeightAtStart and CD4AtStart have the highest importance, while MartialStatus have the least importance.

RANDOM FOREST CLASSIFIER

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

rfc = RandomForestClassifier()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scori

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean accuracy score:", np.mean(cv_scores))

rfc.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = rfc.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = rfc.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

y_test_pred_proba = rfc.predict_proba(X_test)[:,1]
print("Test Data AUC-ROC Score:", roc_auc_score(y_test, y_test_pred_proba))

# Calculate AUC-ROC score for test data
auc_roc_rfc = roc_auc_score(y_test, y_test_pred_proba)

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba)
plt.plot(fpr, tpr, label='Random Forest (AUC = {:.2f})'.format(auc_roc_rfc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

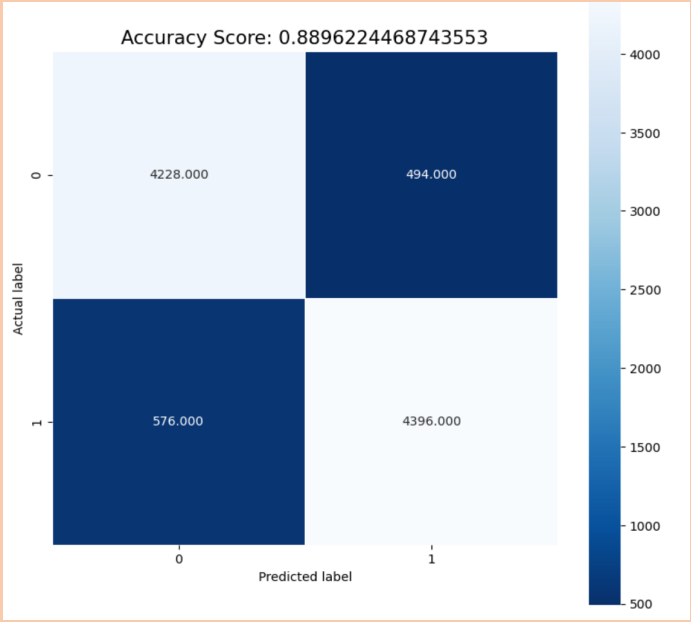
```
Cross-validation scores: [0.88731305 0.88009283 0.89092316 0.88421867 0.88886024 0.8891181
 0.88264122 0.87748259 0.89656951 0.89089502]
Mean accuracy score: 0.886811438585392
Training Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     19513
           1       1.00      0.99      0.99     19263

    accuracy                           0.99     38776
   macro avg       0.99      0.99      0.99     38776
weighted avg       0.99      0.99      0.99     38776

Test Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89      4722
           1       0.90      0.88      0.89      4972

    accuracy                           0.89      9694
   macro avg       0.89      0.89      0.89      9694
weighted avg       0.89      0.89      0.89      9694

Test Data AUC-ROC Score: 0.9512993645396857
```



We performed the Random Forest Classifier model for OpportunisticInfectionAtLastVisit with 10-fold cross-validation and our model accuracy is 0.89 and we plotted the ROC curve for the model.

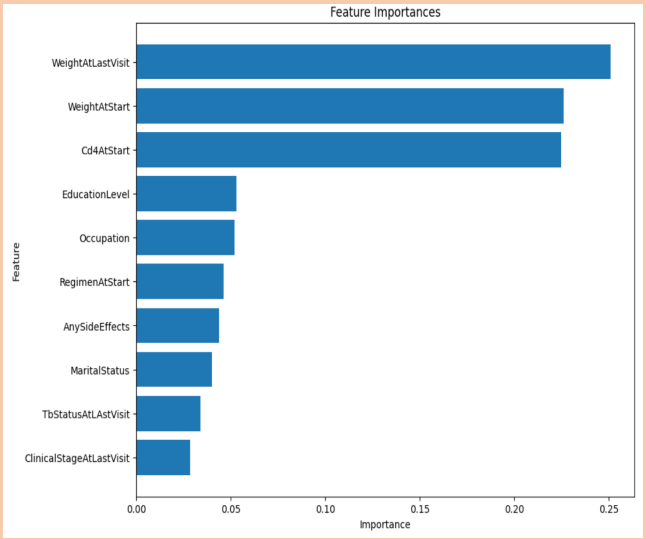
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = rfc.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

```
# Print feature importances
importances = rfc.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

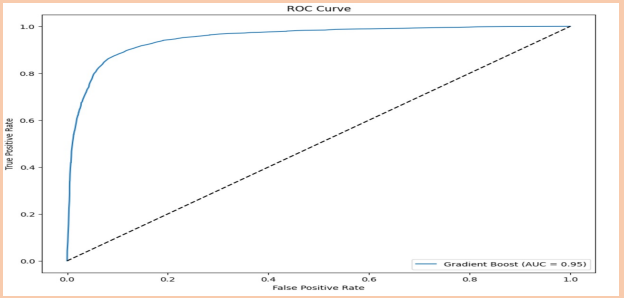


We have computed the feature importance for a Random forest classifier model on OpportunisticInfectionAtLastVisit, and the results indicate that WeightAtLastVisit and WeightAtstart have the highest importance, while ClinicalstagesLastVisit have the least importance.

GRADIENT BOOSTING CLASSIFIER

```python
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report, roc_auc_score
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

gb = GradientBoostingClassifier()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(gb, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean accuracy score:", np.mean(cv_scores))

gb.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = gb.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = gb.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

y_test_pred_proba = rfc.predict_proba(X_test)[:,1]
print("Test Data AUC-ROC Score:", roc_auc_score(y_test, y_test_pred_proba))

# Calculate AUC-ROC score for test data
auc_roc_rfc = roc_auc_score(y_test, y_test_pred_proba)

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba)
plt.plot(fpr, tpr, label='Gradient Boost (AUC = {:.2f})'.format(auc_roc_rfc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

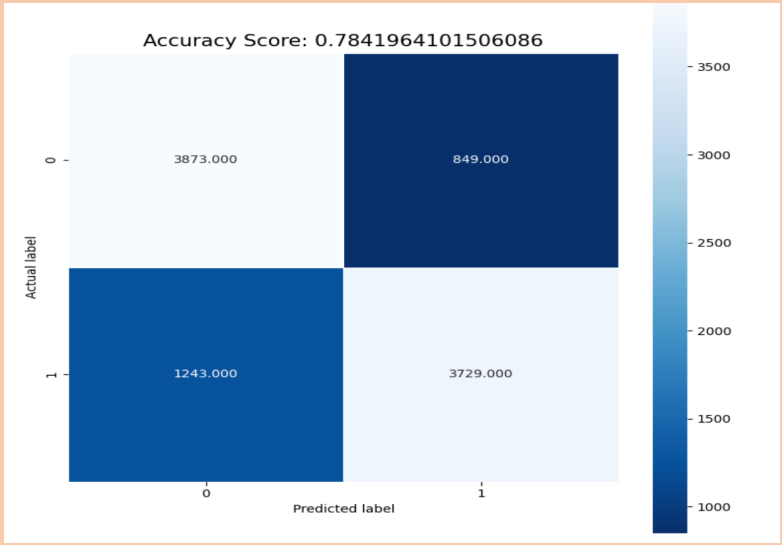
```
Cross-validation scores: [0.78932439 0.76843734 0.77101599 0.77772047 0.78726147 0.79009799
 0.77611555 0.76889347 0.7789528  0.78204798]
Mean accuracy score: 0.7789867459979729
Training Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.82      0.79     19513
           1       0.80      0.75      0.77     19263

    accuracy                           0.78     38776
   macro avg       0.79      0.78      0.78     38776
weighted avg       0.79      0.78      0.78     38776

Test Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.82      0.79      4722
           1       0.81      0.75      0.78      4972

    accuracy                           0.78      9694
   macro avg       0.79      0.79      0.78      9694
weighted avg       0.79      0.78      0.78      9694

Test Data AUC-ROC Score: 0.9512993645396857
```



We performed the Gradient Boosting Classifier model for OpportunisticInfectionAtLastVisit by splitting the data set into test and train and a cross-validation fold of 10 folds and we found the accuracy for our model as 0.79. We Plotted the ROC curve and the AUC – ROC score is 0.95 .

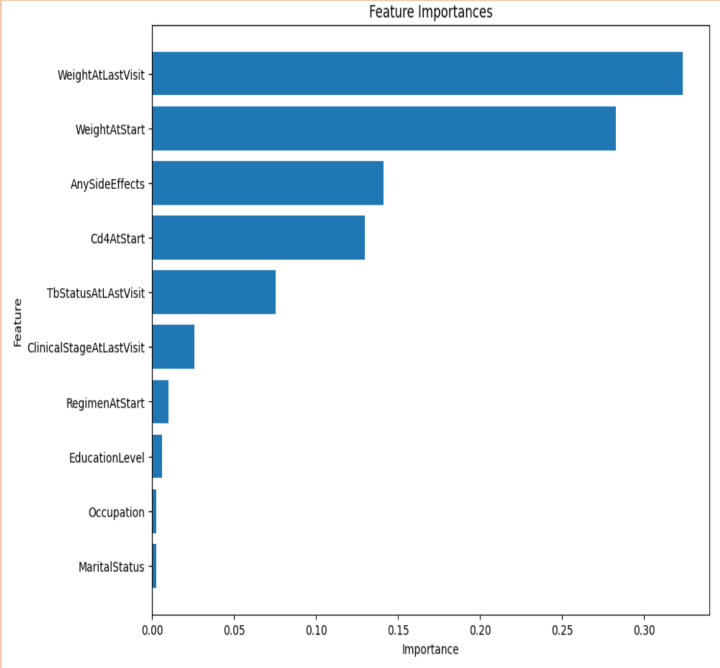
CONFUSION MATRIX

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = gb.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```



FEATURE IMPORTANCE

```
# Print feature importances
importances = gb.feature_importances_
feature_importances = pd.DataFrame({"feature": X_train.columns, "importance": importances})
feature_importances = feature_importances.sort_values(by="importance", ascending=True)

# Create a bar graph of feature importances
plt.barh(feature_importances["feature"], feature_importances["importance"])
plt.title("Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



We have computed the feature importance for a Gradient Boosting classifier model on OpportunisticInfectionAtLastVisit, and the results indicate that WeightAtLastVisit and WeightAtstart have the highest importance, while MartialStatus have the least importance.

COMPARISION OF ACCURACIES BETWEEN MODELS

MODEL CONSIDERED	ACCURACY OF THE MODEL
LOGISTIC REGRESSION	0.62
DECISION TREE CLASSIFIER	0.85
RANDOM FOREST CLASSIFIER	0.89
GRADIENT BOOSTING CLASSIFIER	0.79

Based on Accuracy, Random forest performed well.

REGRESSION MODELS

LINEAR REGRESSION

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from skopt import gp_minimize
from skopt.space import Real, Categorical
from skopt.utils import use_named_args
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["MostRecentCd4Count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [Real(0.01, 100.0, name='alpha'),
                Categorical([True, False], name='fit_intercept')]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = LinearRegression(fit_intercept=params['fit_intercept'])
    scores = cross_val_score(model, X_train, y_train, cv=10, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best hyperparameters: alpha=%.4f, fit_intercept=%s" % (result.x[0], result.x[1]))

# Train the model using the best hyperparameters
model = LinearRegression(fit_intercept=result.x[1])
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best hyperparameters: alpha=26.8910, fit_intercept=True
Mean Squared Error: 0.7539649785683739
R-squared: 0.2798695665187695
Mean Absolute Percentage Error: 2.3130864888572495
Root Mean Squared Error: 0.8683115676808492
```

DECISION TREE REGRESSOR

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["MostRecentCd4Count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(1, 10, name='max_depth'),
    Integer(2, 50, name='min_samples_split'),
    Real(0.01, 0.5, name='min_samples_leaf')
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = DecisionTreeRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=10, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: max_depth=%d, min_samples_split=%d, min_samples_leaf=%.4f" % tuple(result.x))

# Train the model using the best hyperparameters
best_params = {'max_depth': result.x[0], 'min_samples_split': result.x[1], 'min_samples_leaf': result.x[2]}
model = DecisionTreeRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best parameters: max_depth=9, min_samples_split=27, min_samples_leaf=0.0224
Mean Squared Error: 0.7696243724839089
R-squared: 0.2649128955207875 6
Mean Absolute Percentage Error: 2.246259464940518
Root Mean Squared Error: 0.8772823789886064
```

RANDOM FOREST REGRESSOR

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd
import numpy as np

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["MostRecentCd4Count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(10, 500, name='n_estimators'),
    Real(0.01, 1.0, name='max_features'),
    Integer(1, 10, name='max_depth'),
    Real(0.01, 0.5, name='min_samples_split')
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = RandomForestRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: n_estimators=%d, max_features=%.4f, max_depth=%d, min_samples_split=%.4f" % tuple(resu

# Train the model using the best hyperparameters
best_params = {'n_estimators': result.x[0], 'max_features': result.x[1], 'max_depth': result.x[2], 'min_sample
model = RandomForestRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best parameters: n_estimators=500, max_features=0.6134, max_depth=10, min_samples_split=0.0100
Mean Squared Error: 0.7482718574645667
R-squared: 0.285307206044322
Mean Absolute Percentage Error: 2.313119580150672
Root Mean Squared Error: 0.8650270848155951
```

GRADIENT BOOSTING REGRESSOR

```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["MostRecentCd4Count"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(1, 10, name='max_depth'),
    Integer(2, 50, name='min_samples_split'),
    Real(0.01, 0.5, name='learning_rate'),
    Real(0.01, 1.0, name='subsample'),
    Real(0.01, 0.99, name='min_samples_leaf'),
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = GradientBoostingRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: max_depth=%d, min_samples_split=%d, learning_rate=%.4f, subsample=%.4f, min_samples_le

# Train the model using the best hyperparameters
best_params = {'max_depth': result.x[0], 'min_samples_split': result.x[1], 'learning_rate': result.x[2], 'subs
model = GradientBoostingRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best parameters: max_depth=2, min_samples_split=6, learning_rate=0.1177, subsample=0.7139, min_samples_leaf=
0.0100
Mean Squared Error: 0.7524534996457536
R-squared: 0.2813132170896634
Mean Absolute Percentage Error: 2.2954084180397927
Root Mean Squared Error: 0.8674407758721938
```

COMPARISION BETWEEN MODELS

MODEL	R-SQUARED	MAPE
LINEAR REGRESSION	0.27	2.31
DECISION TREE REGRESSOR	0.26	2.24
RANDOM FOREST REGRESSOR	0.28	2.31
GRADIENT BOOSTING REGRESSOR	0.28	2.29

Based on R-square and MAPE, Random forest and Gradient boosting performed well.

LINEAR REGRESSION

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from skopt import gp_minimize
from skopt.space import Real, Categorical
from skopt.utils import use_named_args

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["ViralLoad"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [Real(0.01, 100.0, name='alpha'),
                Categorical([True, False], name='fit_intercept')]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = LinearRegression(fit_intercept=params['fit_intercept'])
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best hyperparameters: alpha=%.4f, fit_intercept=%s" % (result.x[0], result.x[1]))

# Train the model using the best hyperparameters
model = LinearRegression(fit_intercept=result.x[1])
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best hyperparameters: alpha=76.6075, fit_intercept=True
Mean Squared Error: 0.9775759085601476
R-squared: 0.02311727176452605
Mean Absolute Percentage Error: 1.119406063042112
Root Mean Squared Error: 0.9887243845279369
```

DECISION TREE REGRESSOR

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["ViralLoad"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(1, 10, name='max_depth'),
    Integer(2, 50, name='min_samples_split'),
    Real(0.01, 0.5, name='min_samples_leaf')
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = DecisionTreeRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: max_depth=%d, min_samples_split=%d, min_samples_leaf=%.4f" % tuple(result.x))

# Train the model using the best hyperparameters
best_params = {'max_depth': result.x[0], 'min_samples_split': result.x[1], 'min_samples_leaf': result.x[2]}
model = DecisionTreeRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best parameters: max_depth=7, min_samples_split=10, min_samples_leaf=0.0493
Mean Squared Error: 1.0051489307796757
R-squared: 0.020836096885720634
Mean Absolute Percentage Error: 1.2133109890897824
Root Mean Squared Error: 1.0025711599580729
```

RANDOM FOREST REGRESSOR

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["ViralLoad"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(10, 500, name='n_estimators'),
    Real(0.01, 1.0, name='max_features'),
    Integer(1, 10, name='max_depth'),
    Real(0.01, 0.5, name='min_samples_split')
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = RandomForestRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: n_estimators=%d, max_features=%.4f, max_depth=%d, min_samples_split=%.4f" % tuple(resu

# Train the model using the best hyperparameters
best_params = {'n_estimators': result.x[0], 'max_features': result.x[1], 'max_depth': result.x[2], 'min_sample
model = RandomForestRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
```

```
Best score: 0.9575
Best parameters: n_estimators=500, max_features=0.4361, max_depth=10, min_samples_split=0.0100
Mean Squared Error: 0.9898491838779941
R-squared: 0.035740315986158455
Mean Absolute Percentage Error: 1.1494673879729143
```

GRADIENT BOOSTING REGRESSOR

```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, train_test_split
from skopt import gp_minimize
from skopt.space import Real, Integer
from skopt.utils import use_named_args
import pandas as pd

# Split data into training and testing sets
X = df_new.drop(["ArvAdherenceLatestLevel", "ViralLoad", "MostRecentCd4Count", "OpportunisticInfectionPresentA
y = df_new["ViralLoad"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=47)

# Define the search space for hyperparameters
search_space = [
    Integer(1, 10, name='max_depth'),
    Integer(2, 50, name='min_samples_split'),
    Real(0.01, 0.5, name='learning_rate'),
    Real(0.01, 1.0, name='subsample'),
    Real(0.01, 0.99, name='min_samples_leaf'),
]

# Define the objective function to minimize
@use_named_args(search_space)
def objective_function(**params):
    model = GradientBoostingRegressor(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    return -1.0 * scores.mean()

# Run Bayesian optimization to minimize the objective function
result = gp_minimize(objective_function, search_space, n_calls=50)

# Print the best set of hyperparameters found
print("Best parameters: max_depth=%d, min_samples_split=%d, learning_rate=%.4f, subsample=%.4f, min_samples_le

# Train the model using the best hyperparameters
best_params = {'max_depth': result.x[0], 'min_samples_split': result.x[1], 'learning_rate': result.x[2], 'subs
model = GradientBoostingRegressor(**best_params)
model.fit(X_train, y_train)

# Make predictions using the trained model
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE), R-squared, and MAPE of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics of the model
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Percentage Error:', mape)
print('Root Mean Squared Error:', rmse)
```

```
Best parameters: max_depth=1, min_samples_split=2, learning_rate=0.2856, subsample=0.7472, min_samples_leaf=
0.0100
Mean Squared Error: 0.9637620735013834
R-squared: 0.036921311697830306
Mean Absolute Percentage Error: 1.1923304052170787
Root Mean Squared Error: 0.9817138450186915
```

COMPARISION BETWEEN MODELS

MODEL	R-SQUARED	MAPE
LINEAR REGRESSION	0.023	1.11
DECISION TREE REGRESSOR	0.020	1.21
RANDOM FOREST REGRESSOR	0.035	1.14
GRADIENT BOOSTING REGRESSOR	0.036	1.19

Based on R-square and MAPE, Random forest and Gradient boosting performed well.

LIMITATIONS

- Difficulty in Data Cleaning: The presence of extra characters can make it challenging to clean the data. The process of data cleaning may require manual intervention to remove the extra characters, which can be time-consuming.
- Increased Computing Time: A large dataset with many null values and extra characters can take longer to process. This can increase the computing time required to analyze the data, leading to delays in obtaining results.
- Reduced Quality of Insights: With incomplete or inaccurate data, the insights obtained from analysis may not be of high quality. This can limit the usefulness of the analysis in making informed decisions.

REFERENCES

1) Nakagawa, F., Lodwick, R. K., Smith, C. J., Smith, R., Cambiano, V., Lundgren, J. D., Delpech, V., & Phillips, A. N. (2012). Projected life expectancy of people with HIV according to timing of diagnosis. AIDS (London, England), 26(3), 335–343. https://doi.org/10.1097/QAD.0b013e32834dcec9

2) Miners, A., Phillips, A., Kreif, N., Rodger, A., Speakman, A., Fisher, M., Anderson, J., Collins, S., Hart, G., Sherr, L., Lampe, F. C., & ASTRA (Antiretrovirals, Sexual Transmission and Attitudes) Study (2014). Health-related quality-of-life of people with HIV in the era of combination antiretroviral treatment: a cross-sectional comparison with the general population. The lancet. HIV, 1(1), e32–e40. https://doi.org/10.1016/S2352-3018(14)70018-9

3) Sutini, Rahayu, S. R., Saefurrohim, M. Z., Al Ayubi, M. T. A., Wijayanti, H., Wandastuti, A. D., Miarso, D., & Susilastuti, M. S. (2022). Prevalence and Determinants of Opportunistic Infections in HIV Patients: A Cross-Sectional Study in the City of Semarang. Ethiopian journal of health sciences, 32(4), 809–816. https://doi.org/10.4314/ejhs.v32i4.18

4) Cooper, V., Clatworthy, J., Harding, R., Whetham, J., & Emerge Consortium (2017). Measuring quality of life among people living with HIV: a systematic review of reviews. Health and quality of life outcomes, 15(1), 220. https://doi.org/10.1186/s12955-017-0778-6

5) Robbins, R. N., Spector, A. Y., Mellins, C. A., & Remien, R. H. (2014). Optimizing ART adherence: update for HIV treatment and prevention. Current HIV/AIDS reports, 11(4), 423–433. https://doi.org/10.1007/s11904-014-0229-5

APPENDIX

SQL CONNECTION

```
myvars = {}
with open("gboppana-sql-pass") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()
```

```
myvars.keys()
```

```
import MySQLdb
conn = MySQLdb.connect(host="localhost", user=myvars['DB username'], passwd=myvars['DB password'], db='I501sap
cursor = conn.cursor()
```

```
import pandas as pd
cursor.execute('SELECT Age, Sex, MaritalStatus, EducationLevel, Occupation, RegimenAtStart, WeightAtStart, Cd4
rows = cursor.fetchall()
df = pd.read_sql('SELECT Age, Sex, MaritalStatus, EducationLevel, Occupation, RegimenAtStart, WeightAtStart, C
```

DATA DESCRIPTION

```
#Displaying the df
df
```

```
#printing df head
df.head()
```

```
df.describe()
```

```
df.columns
```

```
df.shape
```

```
df.dtypes
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Define colors
colors = plt.cm.Reds(np.linspace(0, 1, 5))

# Define subplots
fig, axs = plt.subplots(1, 3, figsize=(12, 4))

# Plot pie chart for 'ArvAdherenceLatestLevel'
regimen_counts = df['ArvAdherenceLatestLevel'].value_counts()
axs[0].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[0].set_title('Distribution of ArvAdherenceLatestLevel')

# Plot pie chart for 'Sex'
regimen_counts = df['Sex'].value_counts()
axs[1].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[1].set_title('Distribution of Sex')

# Plot pie chart for 'OpportunisticInfectionPresentAtLastVisit'
regimen_counts = df['OpportunisticInfectionPresentAtLastVisit'].value_counts()
axs[2].pie(regimen_counts.values, labels=regimen_counts.index, autopct='%1.1f%%', colors=colors)
axs[2].set_title('Distribution of OpportunisticInfectionPresentAtLastVisit')

plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load data from dataframe
occupation_counts = df['Occupation'].value_counts()
education_counts = df['EducationLevel'].value_counts()
marital_counts = df['MaritalStatus'].value_counts()
clinical_counts = df['ClinicalStageAtLastVisit'].value_counts()

# Set the figure size and layout
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
fig.subplots_adjust(hspace=0.5, wspace = 0.5)

# Define colors
colors = plt.cm.Pastel1(np.linspace(0, 1, 5))

# Create subplots of bar charts
axs[0, 0].bar(occupation_counts.index, occupation_counts.values, color=colors)
axs[0, 1].bar(education_counts.index, education_counts.values, color=colors)
axs[1, 0].bar(marital_counts.index, marital_counts.values, color=colors)
axs[1, 1].bar(clinical_counts.index, clinical_counts.values, color=colors)

# Set titles and axis labels for each subplot
axs[0, 0].set_title('Distribution of Occupation')
axs[0, 0].set_xlabel('Occupation')
axs[0, 0].set_ylabel('Number of Patients')
axs[0, 1].set_title('Distribution of EducationLevel')
axs[0, 1].set_xlabel('EducationLevel')
axs[0, 1].set_ylabel('Number of Patients')
axs[1, 0].set_title('Distribution of MaritalStatus')
axs[1, 0].set_xlabel('MaritalStatus')
axs[1, 0].set_ylabel('Number of Patients')
axs[1, 1].set_title('Distribution of ClinicalStageAtLastVisit')
axs[1, 1].set_xlabel('ClinicalStageAtLastVisit')
axs[1, 1].set_ylabel('Number of Patients')

# Add some customization to the charts
for ax in axs.flat:
    ax.grid(axis='y', alpha=0.5)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
    ax.set_ylim(0, max(max(occupation_counts.values), max(education_counts.values),
                       max(marital_counts.values), max(clinical_counts.values)) * 1.1)
    for i, val in enumerate(ax.patches):
        ax.text(val.get_x() + val.get_width() / 2, val.get_height() + 5, str(val.get_height()),
                ha='center', va='bottom')
# Show the chart
plt.show()
```

```python
# Printing the Age column Max and Min values
df['Age'].max()
```

```python
df['Age'].min()
```

CHECKING NULL VALUES AND DEALING WITH IT - DATA CLEANING - IMPUTATION

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

```
df['Age'] = pd.to_numeric(df['Age'], errors='coerce') # coerce errors to NaN
mean_age = df['Age'].mean()
rounded_mean_age = int(round(mean_age))
df['Age'].fillna(rounded_mean_age, inplace=True) # fill NaN with rounded mean of non-NaN values
df['Age'] = df['Age'].astype(int)
```

```
# Convert the 'Cd4AtStart' column to numeric
df['Cd4AtStart'] = pd.to_numeric(df['Cd4AtStart'], errors='coerce')
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import pandas as pd


# Select the column with missing values
column = df['Cd4AtStart']

# Create the imputer object
imputer = IterativeImputer(max_iter=10, random_state=0)

# Fit the imputer on the column
imputer.fit(column.values.reshape(-1, 1))

# Transform the column with imputed values
column_imputed = pd.DataFrame(imputer.transform(column.values.reshape(-1, 1)))

# Replace the missing values in the original DataFrame with the imputed values
df['Cd4AtStart'] = column_imputed.values

# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['Cd4AtStart'].isnull().sum())
```

```
# Select the column with missing values
column = df['WeightAtStart']
df['WeightAtStart'] = pd.to_numeric(df['WeightAtStart'], errors='coerce') # coerce errors to NaN
mean_WeightAtStart = df['WeightAtStart'].mean()
rounded_mean_WeightAtStart = int(round(mean_WeightAtStart))
df['WeightAtStart'].fillna(rounded_mean_WeightAtStart, inplace=True)
# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['WeightAtStart'].isnull().sum())
```

```python
# Select the column with missing values
column = df['WeightAtLastVisit']

df['WeightAtLastVisit'] = pd.to_numeric(df['WeightAtLastVisit'], errors='coerce') # coerce errors to NaN
mean_WeightAtLastVisit = df['WeightAtLastVisit'].mean()
rounded_mean_WeightAtLastVisit = int(round(mean_WeightAtLastVisit))
df['WeightAtLastVisit'].fillna(rounded_mean_WeightAtLastVisit , inplace=True)
# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['WeightAtLastVisit'].isnull().sum())
```

```python
df['MostRecentCd4Count'] = pd.to_numeric(df['MostRecentCd4Count'], errors='coerce')
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import pandas as pd


# Select the column with missing values
column = df['MostRecentCd4Count']

# Create the imputer object
imputer = IterativeImputer(max_iter=10, random_state=0)

# Fit the imputer on the column
imputer.fit(column.values.reshape(-1, 1))

# Transform the column with imputed values
column_imputed = pd.DataFrame(imputer.transform(column.values.reshape(-1, 1)))

# Replace the missing values in the original DataFrame with the imputed values
df['MostRecentCd4Count'] = column_imputed.values

# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['MostRecentCd4Count'].isnull().sum())
```

```python
import pandas as pd
import re

# Define a regular expression pattern to extract numbers
pattern = r'(\d+\.?\d*)'

# Convert the 'ViralLoad' column to string data type
df['ViralLoad'] = df['ViralLoad'].astype(str)

# Apply the pattern to the 'ViralLoad' column and extract only the matched values
df['ViralLoad'] = df['ViralLoad'].str.extract(pattern)

# Convert the 'ViralLoad' column to a float data type
df['ViralLoad'] = df['ViralLoad'].astype(float)
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'Sex' column to string type
df['Sex'] = df['Sex'].astype(str)

# Apply the regular expression pattern to the 'Sex' column
df['Sex'] = df['Sex'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'Sex' column
df['Sex'].unique()
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'MaritalStatus' column to string type
df['MaritalStatus'] = df['MaritalStatus'].astype(str)

# Apply the regular expression pattern to the 'MaritalStatus' column
df['MaritalStatus'] = df['MaritalStatus'].apply(lambda x: pattern.sub('', x))

df['MaritalStatus'].unique()
```

```python
# Compute the mode of the non-missing values in the "MaritalStatus" column
mode_value = df['MaritalStatus'].dropna().mode()[0]

# Replace the missing values in the "MaritalStatus" column with the mode value
df['MaritalStatus'] = df['MaritalStatus'].fillna(mode_value)
df.loc[df['MaritalStatus'] == 'None', 'MaritalStatus'] = mode_value
df.loc[df['MaritalStatus'] == 'Missing', 'MaritalStatus'] = mode_value
df['MaritalStatus'].unique()
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'EducationLevel' column to string type
df['EducationLevel'] = df['EducationLevel'].astype(str)

# Apply the regular expression pattern to the 'EducationLevel' column
df['EducationLevel'] = df['EducationLevel'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'EducationLevel' column
df['EducationLevel'].unique()
```

```python
mode_value = df['EducationLevel'].dropna().mode()[0]

df['EducationLevel'] = df['EducationLevel'].fillna(mode_value)
df.loc[df['EducationLevel'] == 'None', 'EducationLevel'] = mode_value
df.loc[df['EducationLevel'] == 'Missing', 'EducationLevel'] = mode_value
df['EducationLevel'].unique()
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'Occupation' column to string type
df['Occupation'] = df['Occupation'].astype(str)

# Apply the regular expression pattern to the 'Occupation' column
df['Occupation'] = df['Occupation'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'Occupation' column
df['Occupation'].unique()
```

```python
mode_value = df['Occupation'].dropna().mode()[0]

df['Occupation'] = df['Occupation'].fillna(mode_value)
df.loc[df['Occupation'] == 'None', 'Occupation'] = mode_value
df['Occupation'].unique()
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'RegimenAtStart' column to string type
df['RegimenAtStart'] = df['RegimenAtStart'].astype(str)

# Apply the regular expression pattern to the 'RegimenAtStart' column
df['RegimenAtStart'] = df['RegimenAtStart'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'RegimenAtStart' column
df['RegimenAtStart'].unique()
```

```python
mode_value = df['RegimenAtStart'].dropna().mode()[0]

df['RegimenAtStart'] = df['RegimenAtStart'].fillna(mode_value)
df.loc[df['RegimenAtStart'] == 'None', 'RegimenAtStart'] = mode_value
df['RegimenAtStart'].unique()
```

```python
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'ClinicalStageAtLastVisit' column to string type
df['ClinicalStageAtLastVisit'] = df['ClinicalStageAtLastVisit'].astype(str)

# Apply the regular expression pattern to the 'ClinicalStageAtLastVisit' column
df['ClinicalStageAtLastVisit'] = df['ClinicalStageAtLastVisit'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'ClinicalStageAtLastVisit' column
df['ClinicalStageAtLastVisit'].unique()
```

```
mode_value = df['TbStatusAtLAstVisit'].dropna().mode()[0]

df['TbStatusAtLAstVisit'] = df['TbStatusAtLAstVisit'].fillna(mode_value)
df.loc[df['TbStatusAtLAstVisit'] == 'None', 'TbStatusAtLAstVisit'] = mode_value
df['TbStatusAtLAstVisit'].unique()
```

```
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'TbStatusAtLAstVisit' column to string type
df['TbStatusAtLAstVisit'] = df['TbStatusAtLAstVisit'].astype(str)

# Apply the regular expression pattern to the 'TbStatusAtLAstVisit' column
df['TbStatusAtLAstVisit'] = df['TbStatusAtLAstVisit'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'TbStatusAtLAstVisit' column
df['TbStatusAtLAstVisit'].unique()
```

```
mode_value = df['ArvAdherenceLatestLevel'].dropna().mode()[0]

df['ArvAdherenceLatestLevel'] = df['ArvAdherenceLatestLevel'].fillna(mode_value)
# Replace all 'None' values with the mode
df.loc[df['ArvAdherenceLatestLevel'] == 'None', 'ArvAdherenceLatestLevel'] = mode_value
df['ArvAdherenceLatestLevel'].unique()
```

```
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'ArvAdherenceLatestLevel' column to string type
df['ArvAdherenceLatestLevel'] = df['ArvAdherenceLatestLevel'].astype(str)

# Apply the regular expression pattern to the 'ArvAdherenceLatestLevel' column
df['ArvAdherenceLatestLevel'] = df['ArvAdherenceLatestLevel'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'ArvAdherenceLatestLevel' column
df['ArvAdherenceLatestLevel'].unique()
```

```
import pandas as pd
from scipy.stats import mode


# get the mode of the column
mode_val = mode(df['ClinicalStageAtLastVisit'])[0][0]

# replace empty string with mode
df['ClinicalStageAtLastVisit'] = df['ClinicalStageAtLastVisit'].replace('', mode_val)

# check the values of the column after replacement
print(df['ClinicalStageAtLastVisit'].unique())
```

```
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'OpportunisticInfectionPresentAtLastVisit' column to string type
df['OpportunisticInfectionPresentAtLastVisit'] = df['OpportunisticInfectionPresentAtLastVisit'].astype(str)

# Apply the regular expression pattern to the 'OpportunisticInfectionPresentAtLastVisit' column
df['OpportunisticInfectionPresentAtLastVisit'] = df['OpportunisticInfectionPresentAtLastVisit'].apply(lambda x

# View the unique values in the cleaned 'OpportunisticInfectionPresentAtLastVisit' column
df['OpportunisticInfectionPresentAtLastVisit'].unique()
```

```
import pandas as pd
from scipy.stats import mode


# get the mode of the column
mode_val = mode(df['OpportunisticInfectionPresentAtLastVisit'])[0][0]

# replace empty string with mode
df['OpportunisticInfectionPresentAtLastVisit'] = df['OpportunisticInfectionPresentAtLastVisit'].replace('', mo

# check the values of the column after replacement
print(df['OpportunisticInfectionPresentAtLastVisit'].unique())
```

```
import re
# Create a regular expression pattern to remove any non-alphanumeric and non-whitespace characters
pattern = re.compile(r'[^a-zA-Z0-9\s]+')

# Convert the 'AnySideEffects' column to string type
df['AnySideEffects'] = df['AnySideEffects'].astype(str)

# Apply the regular expression pattern to the 'AnySideEffects' column
df['AnySideEffects'] = df['AnySideEffects'].apply(lambda x: pattern.sub('', x))

# View the unique values in the cleaned 'AnySideEffects' column
df['AnySideEffects'].unique()
```

```
mode_value = df['AnySideEffects'].dropna().mode()[0]

df['AnySideEffects'] = df['AnySideEffects'].fillna(mode_value)
df.loc[df['AnySideEffects'] == 'None', 'AnySideEffects'] = mode_value
df['AnySideEffects'].unique()
```

```
#After Imputation
styled_df = df.isnull().sum().to_frame().style

# apply a background color to the cells based on the number of missing values
styled_df = styled_df.background_gradient(cmap='winter', vmin=0, vmax=df.shape[0])

# display the styled DataFrame
styled_df
```

CHECKING AND REMOVING DUPLICATES

```
# check for duplicates
duplicates = df[df.duplicated()]

# count number of duplicates
num_duplicates = len(duplicates)

# print results
print("Number of duplicates:", num_duplicates)
#print(duplicates)
```

```
# Drop duplicates
df.drop_duplicates(inplace=True)

# Verify number of duplicates is 0
print("Number of duplicates after dropping: ", df.duplicated().sum())
```

DEALING WITH OUTLIERS

```
import pandas as pd

# Calculate the IQR of each column in the dataframe
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Identify the outliers in each column
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()

# Print the number of outliers in each column
print(outliers)
```

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Convert the 'Cd4AtStart' column to numeric
df['Cd4AtStart'] = pd.to_numeric(df['Cd4AtStart'], errors='coerce')

# Create a figure with two subplots
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

# Plot boxplot of the Cd4AtStart column before outlier removal
sns.boxplot(x=df['Cd4AtStart'], ax=axs[0])
axs[0].set_title('Before Outlier Removal')

# Calculate the first and third quartiles of the Cd4AtStart column
Q1 = df['Cd4AtStart'].quantile(0.25)
Q3 = df['Cd4AtStart'].quantile(0.75)

# Calculate the interquartile range of the Cd4AtStart column
IQR = Q3 - Q1

# Define the lower and upper bounds of the "normal" range of the Cd4AtStart column
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace the outliers in the Cd4AtStart column with the capped values
df['Cd4AtStart'] = np.where(df['Cd4AtStart'] < lower_bound, lower_bound, df['Cd4AtStart'])
df['Cd4AtStart'] = np.where(df['Cd4AtStart'] > upper_bound, upper_bound, df['Cd4AtStart'])


# Plot boxplot of the Cd4AtStart column after outlier removal
sns.boxplot(x=df['Cd4AtStart'], ax=axs[1])
axs[1].set_title('After Outlier Removal')

# Set the plot title
plt.suptitle('Distribution of Cd4AtStart Column Before and After Outlier Removal')

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

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


# Calculate the first and third quartiles of the WeightAtStart column
Q1 = df['WeightAtStart'].quantile(0.25)
Q3 = df['WeightAtStart'].quantile(0.75)

# Calculate the interquartile range of the WeightAtStart column
IQR = Q3 - Q1

# Define the lower and upper bounds of the "normal" range of the WeightAtStart column
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Count the number of outliers before outlier removal
num_outliers_before = len(df[(df['WeightAtStart'] < lower_bound) | (df['WeightAtStart'] > upper_bound)])

# Print the number of outliers before outlier removal
print("Number of outliers before outlier removal:", num_outliers_before)

# Replace the outliers in the WeightAtStart column with the capped values
df['WeightAtStart'] = np.where(df['WeightAtStart'] < lower_bound, lower_bound, df['WeightAtStart'])
df['WeightAtStart'] = np.where(df['WeightAtStart'] > upper_bound, upper_bound, df['WeightAtStart'])

# Count the number of outliers after outlier removal
num_outliers_after = len(df[(df['WeightAtStart'] < lower_bound) | (df['WeightAtStart'] > upper_bound)])

# Print the number of outliers after outlier removal
print("Number of outliers after outlier removal:", num_outliers_after)

# Create a figure with two subplots
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

# Plot boxplot of the WeightAtStart column before outlier removal
sns.boxplot(x=df['WeightAtStart'], ax=axs[0])
axs[0].set_title('Before Outlier Removal', color='red')

# Plot boxplot of the WeightAtStart column after outlier removal
sns.boxplot(x=df['WeightAtStart'], ax=axs[1], color='orange', boxprops=dict(edgecolor='brown', linewidth=2))
axs[1].set_title('After Outlier Removal', color='red')

# Set the plot title
plt.suptitle('Distribution of WeightAtStart Column Before and After Outlier Removal', color='green')

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

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


# Create a figure with two subplots
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

# Plot boxplot of the MostRecentCd4Count column before outlier removal
sns.boxplot(x=df['MostRecentCd4Count'], ax=axs[0])
axs[0].set_title('Before Outlier Removal')

# Calculate the first and third quartiles of the MostRecentCd4Count column
Q1 = df['MostRecentCd4Count'].quantile(0.25)
Q3 = df['MostRecentCd4Count'].quantile(0.75)

# Calculate the interquartile range of the MostRecentCd4Count column
IQR = Q3 - Q1

# Define the lower and upper bounds of the "normal" range of the MostRecentCd4Count column
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace the outliers in the MostRecentCd4Count column with the capped values
df['MostRecentCd4Count'] = np.where(df['MostRecentCd4Count'] < lower_bound, lower_bound, df['MostRecentCd4Coun
df['MostRecentCd4Count'] = np.where(df['MostRecentCd4Count'] > upper_bound, upper_bound, df['MostRecentCd4Coun


# Plot boxplot of the MostRecentCd4Count column after outlier removal
sns.boxplot(x=df['MostRecentCd4Count'], ax=axs[1])
axs[1].set_title('After Outlier Removal')

# Set the plot title
plt.suptitle('Distribution of MostRecentCd4Count Column Before and After Outlier Removal')

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

```python
# Print number of outliers before removing them
Q1 = df['ViralLoad'].quantile(0.25)
Q3 = df['ViralLoad'].quantile(0.75)
IQR = Q3 - Q1
lower_cap = Q1 - 1.5 * IQR
upper_cap = Q3 + 1.5 * IQR
outliers_before = df[(df['ViralLoad'] < lower_cap) | (df['ViralLoad'] > upper_cap)].shape[0]
print("Number of outliers before removing them:", outliers_before)

sns.boxplot(x='ViralLoad', data=df)
plt.show()

df['ViralLoad'] = df['ViralLoad'].clip(lower=lower_cap, upper=upper_cap)

# Print number of outliers after removing them
outliers_after = df[(df['ViralLoad'] < lower_cap) | (df['ViralLoad'] > upper_cap)].shape[0]
print("Number of outliers after removing them:", outliers_after)

sns.boxplot(x='ViralLoad', data=df)
plt.show()
df['ViralLoad'].hist()
```

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import pandas as pd


# Select the column with missing values
column = df['ViralLoad']

# Create the imputer object
imputer = IterativeImputer(max_iter=10, random_state=0)

# Fit the imputer on the column
imputer.fit(column.values.reshape(-1, 1))

# Transform the column with imputed values
column_imputed = pd.DataFrame(imputer.transform(column.values.reshape(-1, 1)))

# Replace the missing values in the original DataFrame with the imputed values
df['ViralLoad'] = column_imputed.values

# Print the number of missing values before and after imputation
print('Missing values before imputation:', column.isnull().sum())
print('Missing values after imputation:', df['ViralLoad'].isnull().sum())
```

DECSRIPTIVE STATISTICS - DATA ANALYSIS

```
df.describe()
```

QQ PLOT - FOR CHECKING NORMALITY

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt


# Extract the columns as NumPy arrays
age_data = df["Age"].to_numpy()
weight_start_data = df["WeightAtStart"].to_numpy()
weight_last_data = df["WeightAtLastVisit"].to_numpy()
cd4_start_data = df["Cd4AtStart"].to_numpy()

# Create a 2 x 2 grid of subplots
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))

# Plot the QQ plot for age data
sm.qqplot(age_data, line='s', ax=axs[0, 0])
axs[0, 0].set_title("QQ Plot for Age Data")

# Plot the QQ plot for weight at start data
sm.qqplot(weight_start_data, line='s', ax=axs[0, 1])
axs[0, 1].set_title("QQ Plot for WeightAtStart Data")

# Plot the QQ plot for weight at last visit data
sm.qqplot(weight_last_data, line='s', ax=axs[1, 0])
axs[1, 0].set_title("QQ Plot for WeightAtLastVisit Data")

# Plot the QQ plot for CD4 at start data
sm.qqplot(cd4_start_data, line='s', ax=axs[1, 1])
axs[1, 1].set_title("QQ Plot for Cd4AtStart Data")

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```

NORMALITY TEST

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'Age' column
stat, p = normaltest(df['Age'])

# calculate skewness and kurtosis of 'Age' column
sk = skew(df['Age'])
ku = kurtosis(df['Age'])

# plot histogram of 'Age' column
plt.hist(df['Age'], density=True, alpha=0.5, color='lightgreen', edgecolor = 'green')
plt.title("Distribution of Age")
plt.xlabel("Age")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['Age'].mean(), df['Age'].std()
x = np.linspace(df['Age'].min(), df['Age'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='green')

plt.show()

alpha = 0.05
if p < alpha:
    print("The 'Age' column is not normally distributed (p = {})".format(p))
else:
    print("The 'Age' column is normally distributed (p = {})".format(p))

print("Skewness of 'Age' column: {}".format(sk))
print("Kurtosis of 'Age' column: {}".format(ku))
```

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'Age' column
stat, p = normaltest(df['WeightAtStart'])

# plot histogram of 'Age' column
plt.hist(df['WeightAtStart'], density=True, alpha=0.5, color='green')
plt.title("Distribution of WeightAtStart")
plt.xlabel("WeightAtStart")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['WeightAtStart'].mean(), df['WeightAtStart'].std()
x = np.linspace(df['WeightAtStart'].min(), df['WeightAtStart'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='red')

plt.show()

alpha = 0.05
if p < alpha:
    print("The 'WeightAtStart' column is not normally distributed (p = {})".format(p))
else:
    print("The 'WeightAtStart' column is normally distributed (p = {})".format(p))

print("Skewness of 'WeightAtStart' column: {}".format(sk))
print("Kurtosis of 'WeightAtStart' column: {}".format(ku))
```

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'Cd4AtStart' column
stat, p = normaltest(df['Cd4AtStart'])

# plot histogram of 'Cd4AtStart' column
plt.hist(df['Cd4AtStart'], density=True, alpha=0.5, color='green')
plt.title("Distribution of Cd4AtStart")
plt.xlabel("Cd4AtStart")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['Cd4AtStart'].mean(), df['Cd4AtStart'].std()
x = np.linspace(df['Cd4AtStart'].min(), df['Cd4AtStart'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='red')

plt.show()

alpha = 0.05
if p < alpha:
    print("The 'Cd4AtStart' column is not normally distributed (p = {})".format(p))
else:
    print("The 'Cd4AtStart' column is normally distributed (p = {})".format(p))

print("Skewness of 'Cd4AtStart' column: {}".format(sk))
print("Kurtosis of 'Cd4AtStart' column: {}".format(ku))
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'WeightAtLastVisit' column
stat, p = normaltest(df['WeightAtLastVisit'])

# plot histogram of 'WeightAtLastVisit' column
plt.hist(df['WeightAtLastVisit'], density=True, alpha=0.5, color='green')
plt.title("Distribution of WeightAtLastVisit")
plt.xlabel("WeightAtLastVisit")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['WeightAtLastVisit'].mean(), df['WeightAtLastVisit'].std()
x = np.linspace(df['WeightAtLastVisit'].min(), df['WeightAtLastVisit'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='red')

plt.show()

alpha = 0.05
if p < alpha:
    print("The 'WeightAtLastVisit' column is not normally distributed (p = {})".format(p))
else:
    print("The 'WeightAtLastVisit' column is normally distributed (p = {})".format(p))

print("Skewness of 'WeightAtLastVisit' column: {}".format(sk))
print("Kurtosis of 'WeightAtLastVisit' column: {}".format(ku))
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import normaltest, skew, kurtosis

# perform normality test on 'MostRecentCd4Count' column
stat, p = normaltest(df['MostRecentCd4Count'])

# plot histogram of 'MostRecentCd4Count' column
plt.hist(df['MostRecentCd4Count'], density=True, alpha=0.5, color='green')
plt.title("Distribution of MostRecentCd4Count")
plt.xlabel("MostRecentCd4Count")
plt.ylabel("Frequency")

# add a normal distribution curve to the plot for comparison
mu, sigma = df['MostRecentCd4Count'].mean(), df['MostRecentCd4Count'].std()
x = np.linspace(df['MostRecentCd4Count'].min(), df['MostRecentCd4Count'].max(), 100)
y = np.exp(-(x-mu)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
plt.plot(x, y, color='red')

plt.show()

alpha = 0.05
if p < alpha:
    print("The 'MostRecentCd4Count' column is not normally distributed (p = {})".format(p))
else:
    print("The 'MostRecentCd4Count' column is normally distributed (p = {})".format(p))

print("Skewness of 'MostRecentCd4Count' column: {}".format(sk))
print("Kurtosis of 'MostRecentCd4Count' column: {}".format(ku))
```

STATISTICS - CORRELATION, MANN WHITNEY U TEST AND KRUSKAL WALLIS

```
import seaborn as sns
import pandas as pd

# Calculate the Spearman's correlation matrix using pandas
corr_matrix = df.corr(method='kendall')

# Create a heatmap using Seaborn
sns.heatmap(corr_matrix, annot=True)

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

```
import pandas as pd
from scipy.stats import chi2_contingency

# Specify the column names for the categorical columns and the outcome variable
cat_cols = ['Sex', 'MaritalStatus', 'EducationLevel', 'Occupation', 'RegimenAtStart', 'ClinicalStageAtLastVisi
outcome_col = 'OpportunisticInfectionPresentAtLastVisit'

# Loop over each categorical column and perform the chi-square test with respect to the outcome variable
for col in cat_cols:
    # Create a contingency table using pandas
    contingency_table = pd.crosstab(df[col], df[outcome_col])

    # Perform the chi-square test using SciPy
    chi2, pval, dof, expected = chi2_contingency(contingency_table)

    # Create a Styler object and apply background color to the cells
    styled_table = contingency_table.style.applymap(lambda x: 'background-color: lightblue', subset=pd.IndexSl

    # Print the contingency table
    print('Contingency table for {}:'.format(col))
    display(styled_table)

    # Print the results of the chi-square test
    print('Chi-square statistic = {:.3f}'.format(chi2))
    print('P-value = {}'.format(pval))
    print('Degrees of freedom = {}'.format(dof))

    # Interpret the results
    if pval < 0.05:
        print('There is a significant association between {} and {}'.format(col, outcome_col))
    else:
        print('There is no significant association between {} and {}'.format(col, outcome_col))
```

```python
import scipy.stats as stats
import pandas as pd

outcome = df['ViralLoad']

independent_var1 = df['MaritalStatus']
independent_var2 = df['EducationLevel']
independent_var3 = df['Occupation']
independent_var4 = df['RegimenAtStart']
independent_var5 = df['ClinicalStageAtLastVisit']
independent_var6 = df['TbStatusAtLAstVisit']


# Group data by each independent variable and apply Kruskal-Wallis test
for i, col in enumerate([independent_var1, independent_var2, independent_var3, independent_var4, independent_v
    groups = [group for name, group in outcome.groupby(col)]
    kw_results = stats.kruskal(*groups)
    if kw_results[1] < 0.05:
        print(f"Independent variable {i}: Reject null hypothesis (p-value = {kw_results[1]}), there is a signi
    else:
        print(f"Independent variable {i}: Fail to reject null hypothesis (p-value = {kw_results[1]}), there is
```

```python
from scipy.stats import mannwhitneyu

# Perform Mann-Whitney U test for Sex
stat, pval = mannwhitneyu(df[df['Sex'] == 'Female']['ViralLoad'],
                          df[df['Sex'] == 'Male']['ViralLoad'])

# Interpret the results
alpha = 0.05
if pval < alpha:
    print('There is a significant difference between Sex and ViralLoad (p-value = {})'.format(pval))
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between Sex and ViralLoad (p-value = {})'.format(pval))
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))

# Perform Mann-Whitney U test for AnySideEffects
stat, pval = mannwhitneyu(df[df['AnySideEffects'] == 'Yes']['ViralLoad'],
                          df[df['AnySideEffects'] == 'No']['ViralLoad'])

# Interpret the results
if pval < alpha:
    print('There is a significant difference between AnySideEffects and ViralLoad (p-value = {})'.format(pval)
    print('Reject the null hypothesis at the {} level of significance'.format(alpha))
else:
    print('There is a significant difference between AnySideEffects and ViralLoad (p-value = {})'.format(pval)
    print('Fail to reject the null hypothesis at the {} level of significance'.format(alpha))
```

DATA VISUALIZATION

```
import pandas as pd
import seaborn as sns
sns.kdeplot(df['MostRecentCd4Count'])
sns.kdeplot(df['ViralLoad'])
plt.xlabel('Values')
plt.title('Density plot for Viral Load and Most Recent CD4 Count')

plt.show()
```

```
sns.violinplot(data=df, x='ClinicalStageAtLastVisit', y='MostRecentCd4Count')

plt.xlabel('Clinical stage at last visit')
plt.ylabel('MostRecentCd4Count')
plt.title('Comparison of Clinical stage at last visit and MostRecentCd4Count')
plt.show()
```

```
plt.hist([df["Age"], df["WeightAtLastVisit"], df["WeightAtStart"]],
         bins=10,
         alpha=0.5,
         label=["Age", "WeightAtLastVisit", "WeightAtStart"])
plt.legend(fontsize=10)
plt.title("Distribution of Age and Weight at Last Visit/Start")
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Group the data by EducationLevel and ArvAdherenceLatestLevel, and calculate the count for each group
adherence_by_education = df.groupby(['EducationLevel', 'ArvAdherenceLatestLevel']).size().unstack()

# Create a stacked bar plot with three different colors
adherence_by_education.plot(kind='bar', stacked=True, color=['#fbb4ae', '#b3cde3', '#ccebc5'])

# Add labels and title
plt.title('Adherence to ART Regimen by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.legend(title='Adherence', loc='upper left')

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

FEATURE SELECTION - BASED ON THE STATISTICS PERFORMED - RESPECTIVE FEATURES ARE SELECTED FOR VARIOUS OUTCOME VARIABLES

FEATURE ENGINEERING

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer


# Define the columns you want to transform
cols_to_transform = ['Age', 'WeightAtStart', 'Cd4AtStart', 'WeightAtLastVisit', 'MostRecentCd4Count', 'ViralLo

# Log transformation
transformer = FunctionTransformer(func=np.log1p, validate=True)
df[cols_to_transform] = transformer.transform(df[cols_to_transform])

# Scaling
scaler = StandardScaler()
df[cols_to_transform] = scaler.fit_transform(df[cols_to_transform])

# Save the updated file to local machine
df.to_csv("my_transformed_dataset.csv", index=False)
```

MACHINE LEARNING

SMOTE - OUTCOME VARIABLE - Arv adherence latest level

```
df_new = pd.read_csv('my_transformed_dataset.csv')
df_new

from imblearn.over_sampling import SMOTE
smote = SMOTE()
X = df_new.drop(["ArvAdherenceLatestLevel", "OpportunisticInfectionPresentAtLastVisit", "MostRecentCd4Count",
y = df_new["ArvAdherenceLatestLevel"]
X_smote, y_smote = smote.fit_resample(X, y)
```

RANDOM FOREST CLASSIFIER

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import classification_report
import numpy as np
import scikitplot as skplt
import matplotlib.pyplot as plt

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)

rf = RandomForestClassifier()
# Perform 10-fold cross-validation
cv_scores = cross_val_score(dt, X_train, y_train, cv=KFold(n_splits=10, shuffle=True, random_state=42), scorin

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean f1_macro score:", np.mean(cv_scores))

rf.fit(X_train, y_train)

# Print classification report for training set
y_train_pred = rf.predict(X_train)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print classification report for test set
y_test_pred = rf.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))


plt.rcParams['figure.figsize'] = [10,8]

predicted_probas = rf.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
score = rf.score(X_test, y_test)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
plt.show()
```