Study of correlation between a patient's risk factors for bone fractures and employing machine learning models to predict bone fractures.

Appala Naidu Bandaru[1], Dulcie Moses Kancherla[1], Suneetha Naidu Mekala[1], Thrinithi Polsani[1], and Vaishali Vijay Shinde[1]

[1]Indiana University-Purdue University, Indianapolis, IN46202, USA
appband@iu.edu, sumekala@iu.edu, dukanch@iu.edu, tpolsani@iu.edu, vshinde@iu.edu

Abstract: The prevalence of fractures is a growing concern worldwide, and identifying the factors that increase the risk of fractures is critical in preventing their occurrence. This project aims to investigate the relationship between various patient risk factors and the likelihood of experiencing bone fractures. Through our statistical analysis, we aimed to provide the best predictive model using machine learning techniques that can assist physicians in identifying patients at high risk for fractures, thus enabling early intervention and prevention. To achieve our goals, we utilized quantitative research design and conducted our study in six stages, data collection, data extraction, cleaning and storage, data analysis and machine learning model development, and data visualization. Our findings revealed that various factors such as age, gender, BMD, BMI, diabetes, hypertension, CAD, CKD, and COPD influence the likelihood of fractures. Our research provides valuable insights that physicians can use to counsel and educate patients on ways to reduce their risk of fractures. Ultimately, our predictive models could enable healthcare professionals to identify high-risk patients early and provide targeted interventions to prevent fractures.

Keywords: bone fractures, risk factors, machine learning, predictive model, statistical analysis, prevention, age, gender, BMD, BMI, diabetes, hypertension, CAD, CKD, COPD, OP.

1. Project Scope:

1.1. Introduction:

Fractures impose a substantial economic burden on society, with direct and indirect medical expenses expected to reach $25.3 billion in the United States by 2025. Globally, there were 178 million new fractures in 2019, up 34% from 1990, with 455 million cases of prevalent acute or chronic fracture symptoms since 1990, up 71%, and 25.8 million YLDs since 1990, up 65.3% (GBD 2019 Fracture Collaborators, 2021) [1]. Diabetes may be associated with a decrease in bone mineral density (BMD) and insulin and insulin-like growth factor-1 (IGF-1) deficiency, increasing the risk of fracture in people with Type 2 diabetes (Dufour et al., 2021) [2]. Hypertensive-related factors, such as increased sympathetic output, cytokines, angiotensin II, oxidative stress, and vascular disease, can affect bone metabolism and the homeostasis between bone production and resorption (Carmo et al., 2020) [3]. Minerals, including iron (Fe), zinc (Zn), copper (Cu), calcium (Ca), phosphorus (P), and magnesium (Mg), play a vital role in maintaining bone health and proper mineralization. Deficiencies or excesses of these minerals can decrease bone density and increase the risk of fractures (Ciosek, 2021) [4]. While a higher body mass index (BMI) is generally associated with increased BMD, the relationship between BMI and fracture risk is not straightforward and varies by bone location (Johansson et al., 2014) [5]. Smoking and alcohol consumption have also been linked to an increased risk of fractures and complications (Yang et al., 2021) [6]. Bone fractures are a common complication of osteoporosis. In fact, the risk of fractures increases significantly in individuals with osteoporosis (OP) as the bones become weak and brittle, making them more prone to fractures. Patients with osteoporosis lose density and become fragile, increasing the risk of fractures, especially in the hip, spine, and wrist. (Sözen,er., et al 2017). [7]

1.2. Aim:

a) To investigate the relationship between patient risk factors for bone fractures and the likelihood of experiencing a bone fracture, using machine learning models.

b) To identify which risk factors are most strongly associated with bone fractures and to develop accurate predictive models that can help healthcare professionals identify individuals at high risk of bone fractures and develop targeted interventions to prevent or mitigate fracture risk.

1.3. Research Questions:

a) **Null Hypothesis**: Factors such as age, gender, bone mineral density, BMI, and comorbidities like diabetes, coronary artery diseases, chronic kidney disease, and chronic obstructive pulmonary disease do not influence bone fractures.

b) **Alternative Hypothesis**: The factors such as age, gender, bone mineral density, and comorbidities like diabetes, coronary artery diseases, chronic kidney disease, and chronic obstructive pulmonary disease influence bone fractures.

1.4. Purpose:

The purpose of this project is to utilize machine learning models to thoroughly analyze patient data and accurately predict the likelihood of bone fractures based on multiple risk factors. These risk factors include critical elements such as body mass index (BMI), bone mineral density (BMD), age (specifically post-menopausal women), gender, social habits such as smoking and drinking, and comorbidities like diabetes, coronary artery disease, chronic kidney disease, and chronic obstructive pulmonary disease. Despite previous fracture occurrences, many healthcare professionals fail to recognize and treat high-risk patients for osteoporosis or other bone-related diseases. Early recognition and implementation of appropriate preventive and therapeutic measures can potentially alleviate the burden of bone disease. Furthermore, this research has the potential to enhance our comprehension of the causes and mechanisms of bone fractures. Such knowledge may inspire the development of novel therapies and interventions that will aid in the prevention of bone fractures and enhance patient outcomes.

2. Methodology:

2.1. Steps of the project

- **Stages**: Our project has four stages, which are stated below and will be described in more depth later in the report:
  i. Data Collection
  ii. Data extraction, cleaning, and storage
  iii. Data Analysis
  iv. Data Visualization
  v. Machine Learning Models
- **Database management system**: MySQL
- **Programming Language**: Python
- **Tools**: Python Jupyter Notebook and phpMyAdmin.

2.2. Original Team Members and Responsibilities:

Our team consisted of five members, and we all came from diverse backgrounds in healthcare and with a basic understanding of MySQL and Python. The team members and the duties they were each given before we began working on the project are listed below:

Name	Background	Responsibilities
Vaishali Vijay Shinde	Doctor of Pharmacy (Post Baccalaureate)	Data Collection, Data Extraction, and Data Cleaning. Statistical Analysis. Project proposal.
Suneetha Naidu Mekala	Bachelor of Medicine Bachelor of Surgery	Data Collection. Statistical Analysis. Project proposal.

Appala Naidu Bandaru	Doctor of Pharmacy	SQL and Python coding. Machine Learning Models. Final Project Report.
Thrinithi Polsani	Bachelors In Dentistry	Data Collection. Statistical Analysis. Project proposal.
Dulcie Moses Kancherla	Doctor of Pharmacy	Data Visualization and Final Project Report.

Fig. 1. Original Project Team Members' Responsibilities

2.3. Actual Contributions from Individual Team Members

Name	Background	Responsibilities
Vaishali Vijay Shinde	Doctor of Pharmacy (Post Baccalaureate)	Data Collection and Data Cleaning. Statistical Analysis. Literature review. Proofreading. Project proposal. Project Presentation. Final Project Report.
Suneetha Naidu Mekala	Bachelor of Medicine Bachelor of Surgery	Data Cleaning. Statistical Analysis. Project proposal. Project Presentation. Final Project Report.
Appala Naidu Bandaru	Doctor of Pharmacy	SQL Table creation. Data Extraction. Machine Learning Models. Data Visualization. Final Project Report.
Thrinithi Polsani	Bachelors In Dentistry	Data Cleaning. Statistical Analysis. Project proposal. Project Presentation. Final Project Report.
Dulcie Moses Kancherla	Doctor of Pharmacy	Literature review. Project proposal. Data Visualization. Final Project Report.

Fig. 2. Team Members' Updated Responsibilities for This Project

2.4. Project Challenge

- Our project was not without its challenges, the most significant of which was the diverse professional backgrounds of the team members. As healthcare professionals with little to no experience in programming languages, the initial stages of the project presented an obstacle as we were unsure of how to proceed. However, through determination and perseverance, we familiarized ourselves with the fundamentals of MySQL and Python and applied this knowledge to the project. With the guidance of our esteemed mentor, Professor Saptarshi Purkayastha, and the teaching staff's assistance, we completed our project within the designated time frame.
- Secondly, we encountered a challenge related to the data types of columns in the CSV file that we uploaded to MySQL. The data types of the columns were automatically inferred by MySQL, which sometimes resulted in inconsistencies with the original data types of the CSV file. This issue could have compromised the integrity of the data and hindered effective analysis and querying. To address this issue, we employed the MySQL update function to modify the data types of the columns in the table. The process involved modifying the table structure by executing the ALTER TABLE statement and then using the UPDATE statement to update the column with the new data type. Through this approach, we ensured that the data was consistent and reliable, and could be used to derive valuable insights. This process was executed under the guidance of

Professor Saptarshi Purkayastha and the Teaching Assistants (TA) and allowed us to overcome the challenge and progress with the project in a professional and efficient manner.

- Finally, despite our efforts to mitigate overfitting using the Synthetic Minority Over-sampling Technique (SMOTE), the Random Forest Regression Model still produced an overfitting model with a high accuracy of 0.95. Therefore, we made the decision to discard the model due to its lack of generalizability and potential to produce inaccurate predictions when applied to new, unseen data.

3. Data Collection:

The dataset was taken from a website named Kaggle, version- V1, 2022. It can be accessed using the link: https://www.kaggle.com/datasets/jehanbhathena/bone-mineral-density

The dataset under consideration contains pertinent information pertaining to risk factors that contribute to bone fractures. These factors include but are not limited to age, gender, comorbidities, laboratory parameters, as well as social habits such as smoking and drinking. The dataset file, namely UA.csv, was uploaded in December of the year 2022 and encompasses a total of 1537 rows and 40 columns.

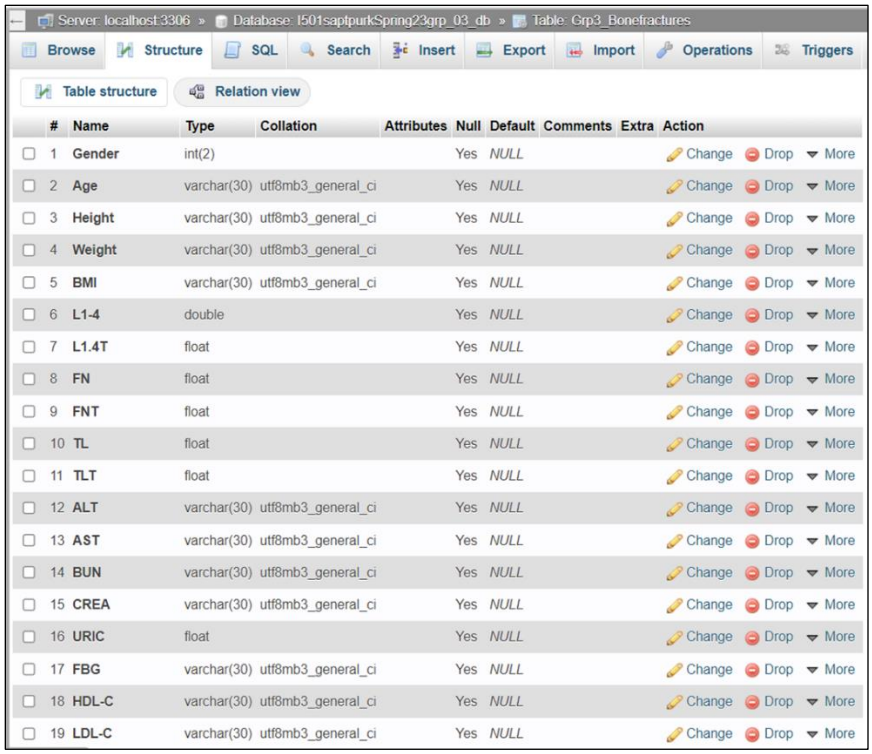
4. Data Extraction, Cleaning, and Storage:

4.1 Data Import and Pre-processing:

We were given shared database access in phpMyAdmin (MySQL) by the professor, named **"I501saptpurkSpring23grp_03_db"**. We created a table in this shared database and imported our dataset named "UA.csv".

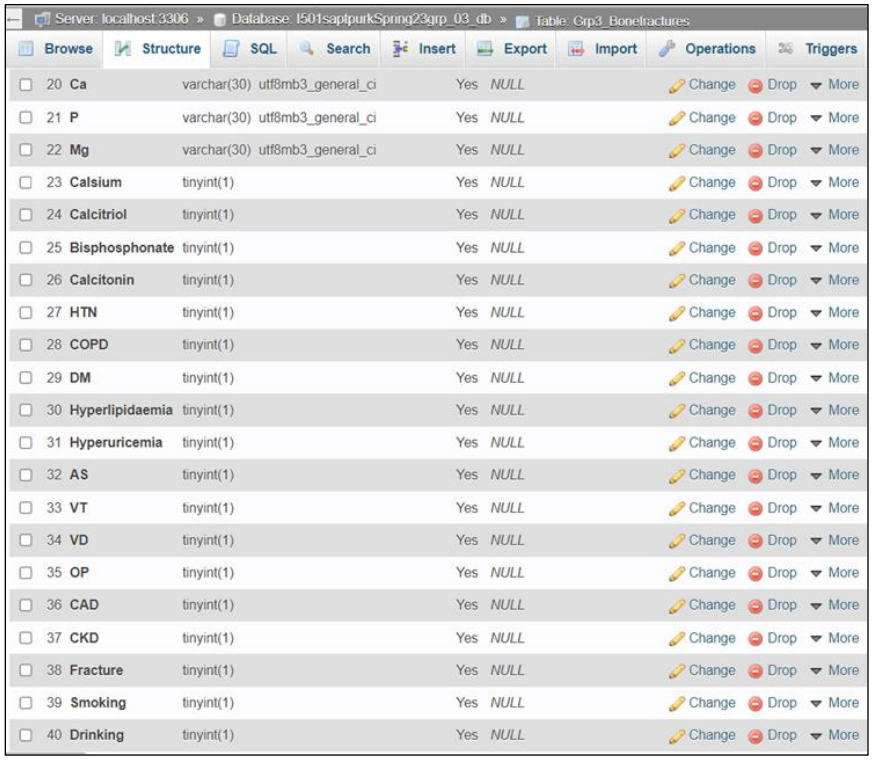
Fig.3. MySQL code for Creating a table in phpMyAdmin in the shared database: I501saptpurkSpring23grp_03_db

We checked for the data types under the "structure" of the table in MySQL. The data types of the columns were automatically inferred by MySQL. However, in some cases, the inferred data types were not the same as the original data types of the CSV file.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Gender	int(2)			Yes	NULL			Change Drop More
2	Age	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
3	Height	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
4	Weight	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
5	BMI	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
6	L1-4	double			Yes	NULL			Change Drop More
7	L1.4T	float			Yes	NULL			Change Drop More
8	FN	float			Yes	NULL			Change Drop More
9	FNT	float			Yes	NULL			Change Drop More
10	TL	float			Yes	NULL			Change Drop More
11	TLT	float			Yes	NULL			Change Drop More
12	ALT	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
13	AST	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
14	BUN	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
15	CREA	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
16	URIC	float			Yes	NULL			Change Drop More
17	FBG	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
18	HDL-C	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
19	LDL-C	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More

Fig. 4.a. Data types of the columns after importing the CSV file in MySQL database.



20	Ca	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
21	P	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
22	Mg	varchar(30)	utf8mb3_general_ci		Yes	NULL			Change Drop More
23	Calsium	tinyint(1)			Yes	NULL			Change Drop More
24	Calcitriol	tinyint(1)			Yes	NULL			Change Drop More
25	Bisphosphonate	tinyint(1)			Yes	NULL			Change Drop More
26	Calcitonin	tinyint(1)			Yes	NULL			Change Drop More
27	HTN	tinyint(1)			Yes	NULL			Change Drop More
28	COPD	tinyint(1)			Yes	NULL			Change Drop More
29	DM	tinyint(1)			Yes	NULL			Change Drop More
30	Hyperlipidaemia	tinyint(1)			Yes	NULL			Change Drop More
31	Hyperuricemia	tinyint(1)			Yes	NULL			Change Drop More
32	AS	tinyint(1)			Yes	NULL			Change Drop More
33	VT	tinyint(1)			Yes	NULL			Change Drop More
34	VD	tinyint(1)			Yes	NULL			Change Drop More
35	OP	tinyint(1)			Yes	NULL			Change Drop More
36	CAD	tinyint(1)			Yes	NULL			Change Drop More
37	CKD	tinyint(1)			Yes	NULL			Change Drop More
38	Fracture	tinyint(1)			Yes	NULL			Change Drop More
39	Smoking	tinyint(1)			Yes	NULL			Change Drop More
40	Drinking	tinyint(1)			Yes	NULL			Change Drop More

Fig. 4.b. Data types of the columns after importing the CSV file in MySQL database.

We used the Update function in MySQL to update the columns that showed Varchar as datatypes. The detailed codes are listed in the Appendix section.

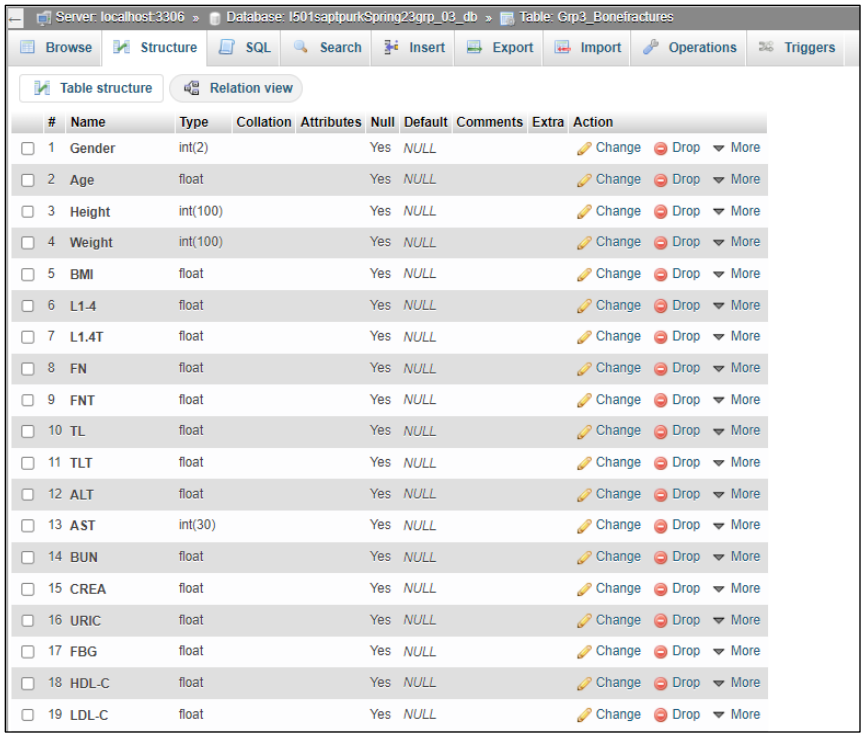


Fig. 4.c. Data types after updating in MySQL.

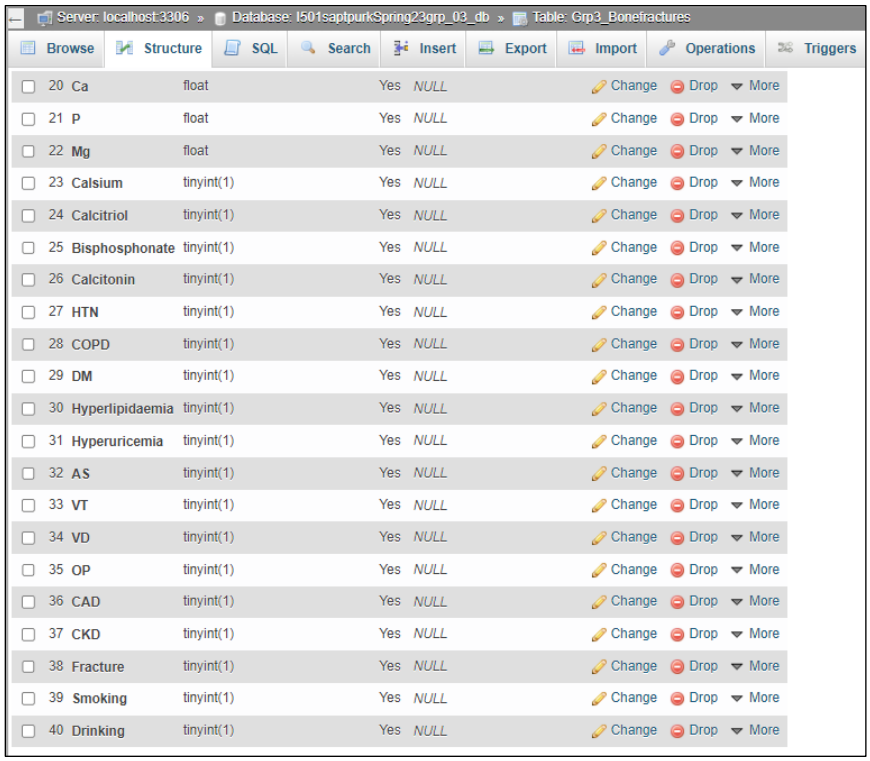


Fig. 4.d. Data types after updating in MySQL.

4.2. Connection between SQL and Python:

The below Python code makes a connection to the MySQL database in phpMyAdmin using the Python MySQLdb package, providing the hostname, username, password, and database name

(**I501saptpurkSpring23grp_03_db**). To run SQL queries on the database, it also constructs a cursor object.

```
In [6]: myvars={}
        with open('vshinde-mysql-password') as myfile:
            for line in myfile:
                name, var=line.partition(':')[::2]
                myvars[name.strip()]=var.strip()
        myvars.keys()

Out[6]: dict_keys(['DB username', 'DB databasename', 'DB password'])

In [7]: import MySQLdb
        conn=MySQLdb.connect(host='localhost', user=myvars['DB username'], password=myvars['DB password'],
                             db='I501saptpurkSpring23grp_03_db')
        cursor = conn.cursor()
        conn

Out[7]: <_mysql.connection open to 'localhost' at 0x55cb86515320>
```

Fig. 5. Connecting the Python and SQL databases to fetch data.

```
In [8]: import pandas as pd
        cursor.execute('select* from Grp3_Bonefractures');
        rows=cursor.fetchall()
        df=pd.read_sql('select* from Grp3_Bonefractures', conn)
```

Fig. 6. Reading the data and retrieving all the rows in Python from MySQL

After establishing the connection, we used the Python codes below to show the data retrieved from MySQL. The following figure confirms that the connection was established successfully.

```
In [9]: print(len(rows))

1538

In [10]: df.shape

Out[10]: (1538, 40)

In [11]: df

Out[11]:
```

	Gender	Age	Height	Weight	BMI	L1-4	L1.4T	FN	FNT	TL	...	Hyperuricemia	AS	VT	VD	OP	CAD	CKD	Fracture	Smoking	Drinkin
0	2.0	61.9	164.0	47.0	17.0	0.894	-2.4	0.6895	-2.95	0.7130	...	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.
1	2.0	55.0	162.0	54.0	21.0	1.333	1.3	0.9130	-1.30	1.0675	...	1.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.
2	2.0	44.0	160.0	54.0	21.0	1.157	-0.2	0.5190	-3.85	0.5770	...	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.
3	1.0	64.7	158.0	59.0	24.0	0.948	-2.3	0.7920	-2.15	0.9050	...	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	1.
4	1.0	88.5	167.0	60.0	22.0	1.114	-0.9	0.8250	-1.90	0.9385	...	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
1533	1.0	51.0	167.0	65.0	23.0	1.048	-1.4	0.9755	-0.05	1.1630	...	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.
1534	1.0	46.6	168.0	69.0	24.0	1.411	1.6	0.9450	-0.95	1.0750	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1535	1.0	55.0	175.0	70.0	23.0	1.075	-1.2	1.0815	0.85	1.1705	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1536	1.0	45.0	172.0	87.0	29.0	1.409	1.6	0.8850	-1.45	0.9215	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1537	NaN	25.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal

1538 rows × 40 columns

Fig.7. Data frame in Python after successful connection establishment

4.3 Data Extraction:

4.3.1. Drop Columns:

The initial phase of data extraction involves the critical task of selecting pertinent attributes or variables for the intended analysis. In accordance with our research objective, which is to explore the correlation between patient risk

factors for bone fractures and the probability of experiencing a bone fracture using machine learning models, we have picked 16 relevant attributes or variables that consisted of 15 independent variables (Gender, Age, BMI, FNT, TLT, HTN, COPD, DM, VT, VD, OP, CAD, CKD, Smoking, Drinking) and one dependent variable (Fracture). Since we have the BMI column, we decided to drop the Height and Weight.

Dropping the columns

In [14]:
```
import pandas as pd
columns_to_drop = ['Height','Weight','L1-4','L1.4T', 'FN','ALT','AST','BUN','CREA','URIC','FBG',
                   'HDL-C','LDL-C','Ca','P','Mg', 'TL', 'Calsium', 'Calcitriol', 'Bisphosphonate',
                   'Calcitonin','Hyperlipidaemia','Hyperuricemia','AS',]
df.drop(columns=columns_to_drop, inplace=True)
```

In [15]:
```
df
```

Out[15]:

	Gender	Age	BMI	FNT	TLT	HTN	COPD	DM	VT	VD	OP	CAD	CKD	Fracture	Smoking	Drinking
0	2.0	61.9	17.0	-2.95	-2.90	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0
1	2.0	55.0	21.0	-1.30	-0.15	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0
2	2.0	44.0	21.0	-3.85	-3.55	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
3	1.0	64.7	24.0	-2.15	-1.40	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0
4	1.0	88.5	22.0	-1.90	-1.15	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1533	1.0	51.0	23.0	-0.05	1.35	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1534	1.0	46.6	24.0	-0.95	-0.10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1535	1.0	55.0	23.0	0.85	1.40	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1536	1.0	45.0	29.0	-1.45	-1.30	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1537	NaN	25.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1538 rows × 16 columns

Fig.8. Python code to drop the undesired columns.

The data description of our dataset is as follows:

S. No.	Variables	Description
1.	Gender	1=male, 2=female
2.	Age	(years)
3.	BMI	(kg/m^2) body mass index
4.	FNT	(T-score) the average of the right and left femoral neck
5.	TLT	(T-score) the average of the right and left total proximal femur
	Clinical diagnosis	
6.	HTN	0=No, 1=Yes Hypertension
7.	COPD	0=No, 1=Yes Chronic obstructive pulmonary disease
8.	DM	0=No, 1=Yes Diabetes
9.	VT	0=No, 1=Yes Venous thromboembolism
10.	VD	0=No, 1=Yes Valve degeneration
11.	OP	0=No, 1=Yes Osteoporosis
12.	CAD	0=No, 1=Yes Chronic artery disease
13.	CKD	0=No, 1=Yes Chronic kidney disease
14.	Fracture	0=No, 1=Yes
	Personal history	
15.	Smoking	0=No, 1=Yes
16.	Drinking	0=No, 1=Yes

Fig. 9. Data description of the desired attributes as per hypothesis

This data was divided into categorical (nominal) data and quantitative (continuous) data:

Categorical		Quantitative	
Nominal Data	**Ordinal Data**	**Discrete Data**	**Continuous Data**
Gender, HTN, COPD, DM, VT, VD, OP, CAD, CKD, Fracture, Smoking, Drinking.			Age, BMI, FNT, and TLT.

Fig.10. Types of data

4.4 Data Cleaning:

4.4.1. Check the null values in Python:

We checked for the null values in Python after dropping the undesired columns. Age (36) and BMI (35) showed the highest number of null values.

Checking for null values

```
In [16]: null_counts = df.isnull().sum()
         print(null_counts)

Gender       1
Age         36
BMI         35
FNT          1
TLT          1
HTN          1
COPD         1
DM           1
VT           1
VD           1
OP           1
CAD          1
CKD          1
Fracture     1
Smoking      1
Drinking     1
dtype: int64
```

Fig.11. Null Values of the desired attributes

4.4.2 Replacing null values in Python:

We replaced the null values using the mean function in Python.

Replacing Null Values with mean

```
In [17]: mean_value = df[['Gender','Age','BMI','FNT','TLT','HTN','COPD','DM','VT','VD','OP','CAD',
                          'CKD','Fracture','Smoking','Drinking']].mean()

         df.fillna(mean_value, inplace=True)
```

Fig.12. Python code for replacing the null values.

4.4.3 Check the data types and correct them in Python:

```
In [14]: print(df.dtypes)

Gender      float64
Age         float64
BMI         float64
FNT         float64
TLT         float64
HTN         float64
COPD        float64
DM          float64
VT          float64
VD          float64
OP          float64
CAD         float64
CKD         float64
Fracture    float64
Smoking     float64
Drinking    float64
dtype: object
```

Fig.13. Data types before cleaning

We then changed the data types of the columns Gender, HTN, COPD, DM, VT, VD, OP, CAD, CKD, Fracture, Smoking, and Drinking to integer in Python using the "**.astype()"** function.

Changing the data types using .astype function

```
In [18]: df['Gender'] = df['Gender'].astype(int)

In [19]: import numpy as np

         df[['HTN','COPD','DM','VT','VD','OP','CAD','CKD','Fracture',
             'Smoking','Drinking']] = df[['HTN','COPD','DM','VT','VD','OP','CAD','CKD',
                                          'Fracture','Smoking','Drinking']].astype(int)

         print(df)
```

```
      Gender   Age        BMI       FNT       TLT  HTN  COPD  DM  VT  VD  OP  \
0          2  61.9  17.000000 -2.950000 -2.900000    1     1   0   0   1   1
1          2  55.0  21.000000 -1.300000 -0.150000    1     0   1   0   1   1
2          2  44.0  21.000000 -3.850000 -3.550000    1     1   1   0   1   1
3          1  64.7  24.000000 -2.150000 -1.400000    1     1   1   0   1   1
4          1  88.5  22.000000 -1.900000 -1.150000    1     0   1   0   1   1
...      ...   ...        ...       ...       ...  ...   ...  ..  ..  ..  ..
1533       1  51.0  23.000000 -0.050000  1.350000    1     0   0   0   0   0
1534       1  46.6  24.000000 -0.950000 -0.100000    0     0   0   0   0   0
1535       1  55.0  23.000000  0.850000  1.400000    0     0   0   0   0   0
1536       1  45.0  29.000000 -1.450000 -1.300000    1     0   0   0   0   0
1537       1  25.0  24.304059 -1.300755 -0.931835    0     0   0   0   0   0

      CAD  CKD  Fracture  Smoking  Drinking
0       1    0         0        1         0
1       0    1         0        0         0
2       1    0         0        0         0
3       1    0         0        1         1
4       1    0         0        1         0
...   ...  ...       ...      ...       ...
1533    1    0         0        0         0
1534    0    0         0        0         0
1535    0    0         0        0         0
1536    0    0         0        0         0
1537    0    0         0        0         0

[1538 rows x 16 columns]
```

Fig.14. Python code to update the data types using .astype() function

```
In [20]: print(df.dtypes)
```

```
Gender        int64
Age         float64
BMI         float64
FNT         float64
TLT         float64
HTN           int64
COPD          int64
DM            int64
VT            int64
VD            int64
OP            int64
CAD           int64
CKD           int64
Fracture      int64
Smoking       int64
Drinking      int64
dtype: object
```

Fig.15. Updated data types

After correcting the data types, we rechecked for null values to ensure the data is completely clean to proceed with further steps in data cleaning.

Recheck for Null Values:

```
In [22]: null_counts = df.isnull().sum()
         print(null_counts)
```

```
Gender      0
Age         0
BMI         0
FNT         0
TLT         0
HTN         0
COPD        0
DM          0
VT          0
VD          0
OP          0
CAD         0
CKD         0
Fracture    0
Smoking     0
Drinking    0
dtype: int64
```

Fig.16. Rechecked for Null Values

4.4.4 Check for Outliers:

On a subset of columns in a Pandas DataFrame, we carried out outlier identification. Outliers are data points in a dataset that deviate considerably from other observations. To prevent inaccurate statistical analysis or modeling outcomes, it is crucial to recognize and handle outliers while analyzing data. We employed the Box plot method to detect any outliers for continuous data in columns Age, BMI, FNT, and TLT.

Outliers

```
In [23]: import matplotlib.pyplot as plt

         # create a list of column names to plot
         columns_to_plot = ['Age','BMI','FNT' ,'TLT']

         # create a boxplot for each column
         for column in columns_to_plot:
             plt.boxplot(df[column], vert=True)
             plt.title('Detecting outliers for ' + column + ' using Boxplot')
             plt.xlabel(column)
             plt.show()
```

Fig.16.a. Python code to check for outliers using the boxplot method.

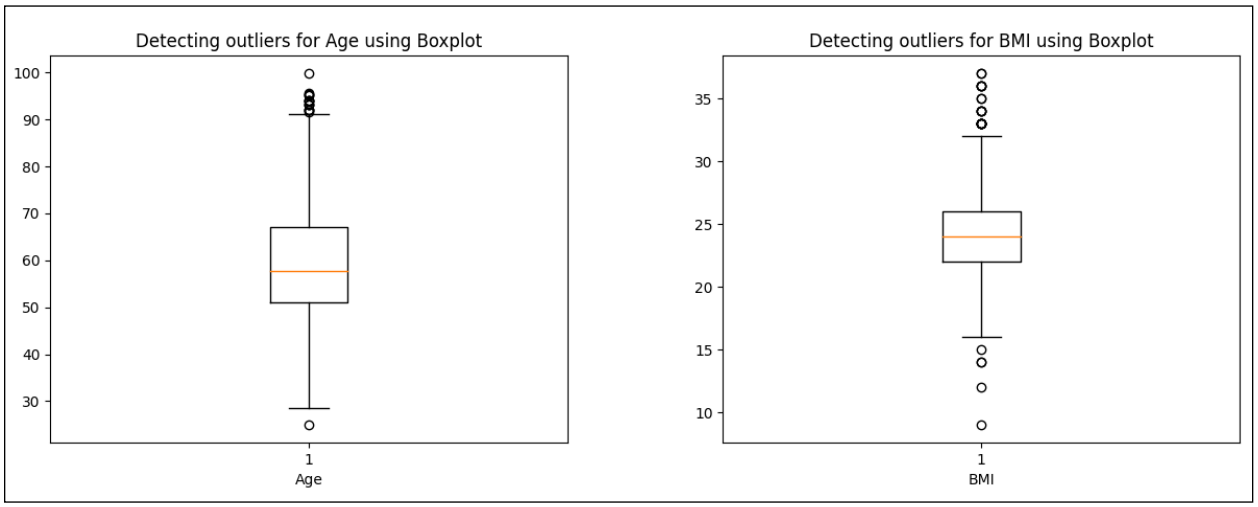


Fig. 16.b. Checked for Outliers using the Boxplot method.

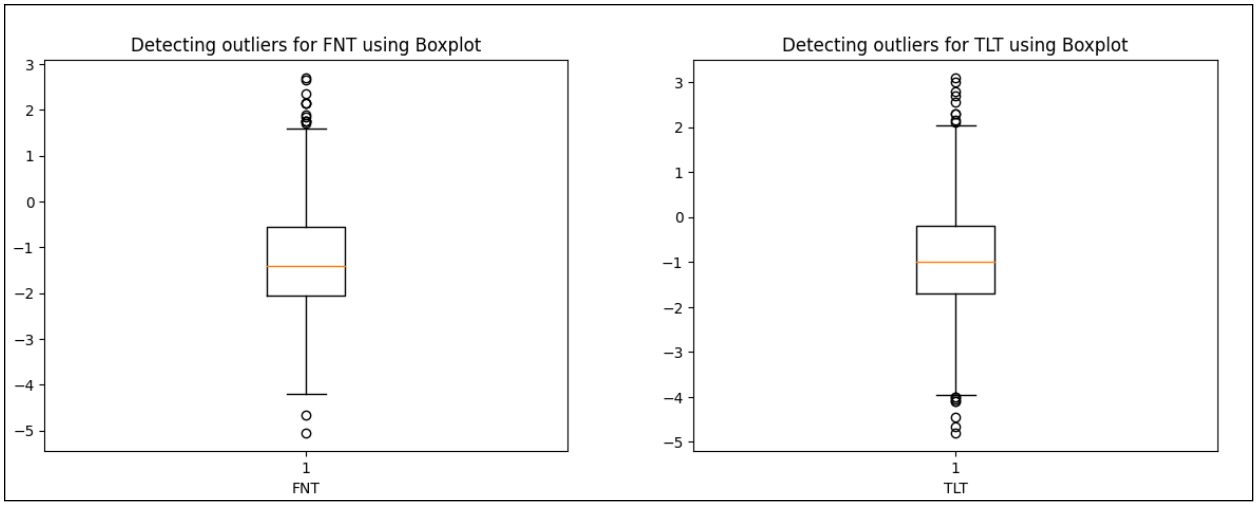


Fig. 16.c. Checked for Outliers using the Boxplot method.

For each column, we calculated the interquartile range (IQR), which is the distance between the first and third quartiles of the distribution for that column. Then, we established 1.5 times the IQR below Q1 and above Q3, respectively, as the lower and upper boundaries for outlier detection.
The below code counts the number of rows with at least one outlier across all four columns after applying boolean logic to check for outliers in each column. This makes it possible to quickly and effectively find outliers in a sample of columns from a bigger dataset.

Outliers treatment (IQR)

In [24]:
```
# create a subset of columns to plot
columns_to_plot = df[['Age','BMI','FNT' ,'TLT']]

# calculate the IQR for each column
Q1 = columns_to_plot.quantile(0.25)
Q3 = columns_to_plot.quantile(0.75)
IQR = Q3 - Q1

# define the upper and lower bounds for outlier detection
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# find outliers for each column
outliers = ((columns_to_plot < lower_bound) | (columns_to_plot > upper_bound)).any(axis=1)

# print the number of outliers for each column
print(outliers.sum())
```

```
67
```

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

# calculate the mean for each column
mean = columns_to_plot.mean()

# replace outliers with mean
for column in columns_to_plot.columns:
    outliers = ((columns_to_plot[column] < lower_bound[column]) | (columns_to_plot[column] > upper_bound[column]))
    columns_to_plot.loc[outliers, column] = mean[column]

# verify that there are no more outliers
outliers = ((columns_to_plot < lower_bound) | (columns_to_plot > upper_bound)).any(axis=1)
print(outliers.sum())
```

```
0
```

Fig. 17. Python code to treat the outliers.

5. Data Analysis:

5.1 Descriptive Analysis:

To obtain a statistical overview of the data frame, we utilized the df.describe() function since it offers a quick and simple approach to learning more about how the data are distributed. Important statistical indicators including the mean, standard deviation, minimum and maximum values, and quartile values are included in this summary. These indicators can aid us in understanding the data's central tendency, distribution, and form.

Descriptive Statistics

```
df.describe()
```

	Gender	Age	BMI	FNT	TLT	HTN	COPD	DM	VT	VD	OP	CAD	CKD	Fracture	Smoking	Drinking
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	1.384915	59.829028	24.304059	-1.300755	-0.931835	0.547464	0.246424	0.328999	0.018856	0.072822	0.369311	0.202861	0.038362	0.020156	0.257477	0.227568
std	0.486734	12.814708	3.274678	1.111250	1.151761	0.497904	0.431068	0.470002	0.136059	0.259928	0.482775	0.402261	0.192130	0.140580	0.437387	0.419399
min	1.000000	25.000000	9.000000	-5.050000	-4.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	51.000000	22.000000	-2.050000	-1.700000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	57.750000	24.000000	-1.400000	-1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	2.000000	67.000000	26.000000	-0.550000	-0.200000	1.000000	0.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000
max	2.000000	99.800000	37.000000	2.700000	3.100000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Fig. 18. Descriptive Statistics of the data frame.

```
In [27]: median = df.median()
         print(median)

Gender       1.00
Age         57.75
BMI         24.00
FNT         -1.40
TLT         -1.00
HTN          1.00
COPD         0.00
DM           0.00
VT           0.00
VD           0.00
OP           0.00
CAD          0.00
CKD          0.00
Fracture     0.00
Smoking      0.00
Drinking     0.00
dtype: float64
```

```
In [28]: range_stat = df.max() - df.min()
         print(range_stat)

Gender       1.00
Age         74.80
BMI         28.00
FNT          7.75
TLT          7.90
HTN          1.00
COPD         1.00
DM           1.00
VT           1.00
VD           1.00
OP           1.00
CAD          1.00
CKD          1.00
Fracture     1.00
Smoking      1.00
Drinking     1.00
dtype: float64
```

```
In [29]: variance = df.var()
         print(variance)

Gender       0.236910
Age        164.216743
BMI         10.723517
FNT          1.234878
TLT          1.326553
HTN          0.247908
COPD         0.185820
DM           0.220902
VT           0.018512
VD           0.067563
OP           0.233072
CAD          0.161814
CKD          0.036914
Fracture     0.019763
Smoking      0.191307
Drinking     0.175895
dtype: float64
```

Fig. 19. Python code for descriptive statistics: Median, Range, and Variance respectively.

5.2 Test for Normality and Skewness of Data:

To handle Python packages and their dependencies, we installed Pip, the standard package manager for Python. Later imported the required packages for statistical analysis:

Statistical tests

```
In [87]: pip install factor-analyzer
```

```
In [88]: from scipy import stats, linalg
         from scipy.stats import shapiro
         from scipy.stats import chi2_contingency
         from scipy.stats import mannwhitneyu
```

Fig. 20. Python codes to import required packages for statistical analysis.

We chose to utilize the Shapiro-Wilk test to determine the normal distribution of our data due to the small size of our dataset. This test entails generating a test statistic W based on the observed values and predicted values, then contrasting it with a critical value from the Shapiro-Wilk distribution. The outcome is a probability value (p-value) connected to the test statistic. We fail to reject the null hypothesis and determine that the data are normally distributed

if the p-value is higher than the significance level, which is often set at 0.05. On the other hand, if the p-value is smaller than the significance level, we reject the null hypothesis and come to the conclusion that the data are not regularly distributed.

In our case, the Shapiro-Wilk test resulted in a p-value less than 0.05, thus, we reject the null hypothesis and conclude that our data is not normally distributed.

Normality test Using Shapiro

```
In [38]: from scipy.stats import shapiro
         import pandas as pd
         cols_of_interest = ['Age', 'BMI', 'FNT', 'TLT']
         for col in cols_of_interest:
             col_data = df[col]
             stat, p = shapiro(col_data)
             print(f"Column '{col}': Statistic= {stat:.4f}, p = {p:.20f}")
             alpha = 0.05
             if p > alpha:
                 print("Sample looks Gaussian (fail to reject H0)\n")
             else:
                 print("Sample does not look Gaussian (reject H0)\n")

Column 'Age': Statistic= 0.9769, p = 0.00000000000000519622
Sample does not look Gaussian (reject H0)

Column 'BMI': Statistic= 0.9844, p = 0.00000000000757454342
Sample does not look Gaussian (reject H0)

Column 'FNT': Statistic= 0.9943, p = 0.00001269864787900588
Sample does not look Gaussian (reject H0)

Column 'TLT': Statistic= 0.9967, p = 0.00212097517214715481
Sample does not look Gaussian (reject H0)
```

Fig. 21. Shapiro-Wilk test to check for normality of the data.

We used a density plot to illustrate the distribution of data.

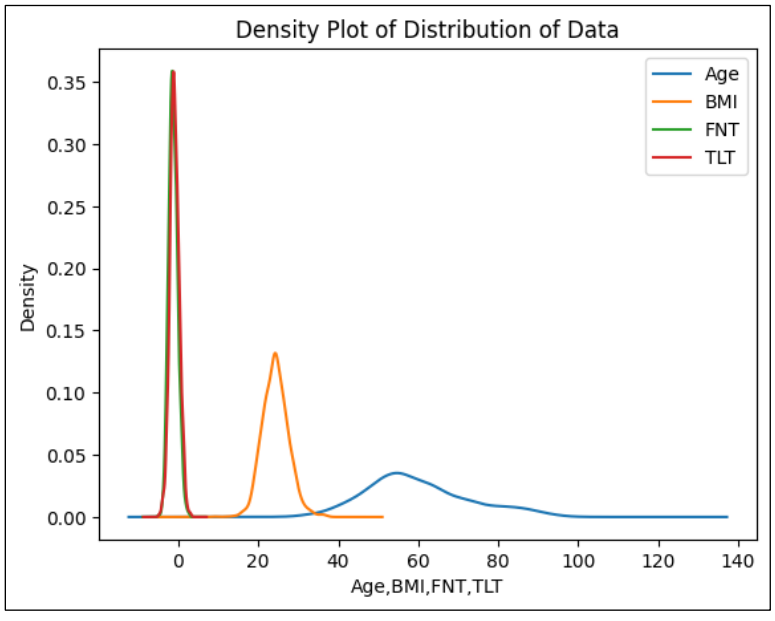


Fig. 22. Normality distribution using density plot.

Skewness of data:
Finding whether variables have non-normal distributions and gaining insight into the data may be accomplished by calculating the skewness of a distribution. Since the value of p for our data for skewness was closer to 0.2, we conclude that the data is positively skewed.

```
In [40]: from scipy.stats import skew
         for col in cols_of_interest:
             col_skewness = skew(df[col])
             print(f"Skewness of {col}: {col_skewness:.2f}")

         Skewness of Age: 0.50
         Skewness of BMI: 0.22
         Skewness of FNT: 0.29
         Skewness of TLT: 0.14
```

Fig. 23. Python code to check for the skewness of data.

5.3 Non-Parametric tests:
After conducting a normality test, we found that our data is not normally distributed. As a result, we utilized non-parametric tests, such as the chi-square and Mann-Whitney U test, moving forward.

5.3.1 Chi-square test:
To evaluate if there is a significant difference between the observed and predicted frequencies of categorical data, the chi-square test is a statistical test that compares the two.

We installed tabulate package to obtain the outcome of the chi-square test in tabular form.

Data preparation for Chi-square teat

```
In [41]: df_chi = df[['Gender','HTN', 'COPD', 'DM', 'VT', 'VD', 'OP', 'CAD', 'CKD','Fracture','Smoking','Drinking']]
         df_chi.head(2)
```

Out[41]:

	Gender	HTN	COPD	DM	VT	VD	OP	CAD	CKD	Fracture	Smoking	Drinking
0	2	1	1	0	0	1	1	1	0	0	1	0
1	2	1	0	1	0	1	1	0	1	0	0	0

```
In [42]: pip install tabulate

         Defaulting to user installation because normal site-packages is not writeable
         Collecting tabulate
           Downloading tabulate-0.9.0-py3-none-any.whl (35 kB)
         Installing collected packages: tabulate
           WARNING: The script tabulate is installed in '/home/students/vshinde/.local/bin' which is not on PATH.
           Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
         Successfully installed tabulate-0.9.0

         [notice] A new release of pip is available: 23.0.1 -> 23.1.2
         [notice] To update, run: python3 -m pip install --upgrade pip
         Note: you may need to restart the kernel to use updated packages.
```

Fig. 24. Data preparation for Chi-square test.

We performed a chi-square test between gender and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between gender and fracture.

```
from tabulate import tabulate
from scipy.stats import chi2_contingency

# create the contingency table using pd.crosstab
chi_Gender = pd.crosstab(df_chi.Gender, df_chi.Fracture, margins=True)

# extract the values from the contingency table
values = chi_Gender.iloc[0:2, 0:5].values

# conduct chi-square test
chi2, p, dof, expected = chi2_contingency(values)

# print the contingency table
print("Contingency table:")
table = chi_Gender.reset_index().values.tolist()
headers = ['Gender', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))

print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

# statistical significance
alpha = 0.05
if p < alpha:
    print("There is a significant association between gender and fracture occurrence (reject H0)")
else:
    print("There is no significant association between gender and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+----------+---------------+------------+---------+
| Gender   |   No Fracture |   Fracture |   Total |
+==========+===============+============+=========+
| 1        |           928 |         18 |     946 |
+----------+---------------+------------+---------+
| 2        |           579 |         13 |     592 |
+----------+---------------+------------+---------+
| All      |          1507 |         31 |    1538 |
+----------+---------------+------------+---------+
Chi-square statistic: 0.1584947392674621
p-value: 0.9238113732968963
Degrees of freedom: 2

There is no significant association between gender and fracture occurrence (fail to reject H0)
```

Fig. 25. Chi-square test between Gender and Fracture

We performed a chi-square test between HTN and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between HTN and fracture.

```
chi_HTN = pd.crosstab(df_chi.HTN, df_chi.Fracture, margins=True)

values = chi_HTN.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_HTN.reset_index().values.tolist()
headers = ['HTN', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between HTN and fracture occurrence (reject H0)")
else:
    print("There is no significant association between HTN and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+-------+---------------+------------+---------+
| HTN   |   No Fracture |   Fracture |   Total |
+=======+===============+============+=========+
| 0     |           686 |         10 |     696 |
+-------+---------------+------------+---------+
| 1     |           821 |         21 |     842 |
+-------+---------------+------------+---------+
| All   |          1507 |         31 |    1538 |
+-------+---------------+------------+---------+
Chi-square statistic: 2.1566659239237778
p-value: 0.3401621164367488
Degrees of freedom: 2

There is no significant association between HTN and fracture occurrence (fail to reject H0)
```

Fig. 26. Chi-square test between HTN and fracture.

We performed a chi-square test between COPD and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between COPD and fracture.

```
chi_COPD = pd.crosstab(df_chi.COPD, df_chi.Fracture, margins=True)

values = chi_COPD.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_COPD.reset_index().values.tolist()
headers = ['COPD', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between COPD and fracture occurrence (reject H0)")
else:
    print("There is no significant association between COPD and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+--------+---------------+------------+---------+
| COPD   |   No Fracture |   Fracture |   Total |
+========+===============+============+=========+
| 0      |          1141 |         18 |    1159 |
+--------+---------------+------------+---------+
| 1      |           366 |         13 |     379 |
+--------+---------------+------------+---------+
| All    |          1507 |         31 |    1538 |
+--------+---------------+------------+---------+
Chi-square statistic: 5.094951607127447
p-value: 0.07827900842495153
Degrees of freedom: 2

There is no significant association between COPD and fracture occurrence (fail to reject H0)
```

Fig. 27. Chi-square test between COPD and fracture.

We performed a chi-square test between DM and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between DM and fracture.

```
chi_DM = pd.crosstab(df_chi.DM, df_chi.Fracture, margins=True)

values = chi_DM.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_DM.reset_index().values.tolist()
headers = ['DM', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between DM and fracture occurrence (reject H0)")
else:
    print("There is no significant association between DM and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+------+---------------+------------+---------+
| DM   |   No Fracture |   Fracture |   Total |
+======+===============+============+=========+
| 0    |          1010 |         22 |    1032 |
+------+---------------+------------+---------+
| 1    |           497 |          9 |     506 |
+------+---------------+------------+---------+
| All  |          1507 |         31 |    1538 |
+------+---------------+------------+---------+
Chi-square statistic: 0.21437444282294582
p-value: 0.8983574654920776
Degrees of freedom: 2

There is no significant association between DM and fracture occurrence (fail to reject H0)
```

Fig. 28. Chi-square test between DM and fracture.

We performed a chi-square test between VT and fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between VT and fracture.

```
chi_VT = pd.crosstab(df_chi.VT, df_chi.Fracture, margins=True)

values = chi_VT.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_VT.reset_index().values.tolist()
headers = ['VT', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between VT and fracture occurrence (reject H0)")
else:
    print("There is no significant association between VT and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+------+---------------+------------+---------+
| VT   |   No Fracture |   Fracture |   Total |
+======+===============+============+=========+
| 0    |          1485 |         24 |    1509 |
+------+---------------+------------+---------+
| 1    |            22 |          7 |      29 |
+------+---------------+------------+---------+
| All  |          1507 |         31 |    1538 |
+------+---------------+------------+---------+
Chi-square statistic: 73.24271052395953
p-value: 1.246084488591757e-16
Degrees of freedom: 2

There is a significant association between VT and fracture occurrence (reject H0)
```

Fig. 29. Chi-square test between VT and fracture.

We performed a chi-square test between VD and fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between VD and fracture.

```
chi_VD = pd.crosstab(df_chi.VD, df_chi.Fracture, margins=True)

values = chi_VD.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_VD.reset_index().values.tolist()
headers = ['VD', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between VD and fracture occurrence (reject H0)")
else:
    print("There is no significant association between VD and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+------+---------------+------------+---------+
| VD   |   No Fracture |   Fracture |   Total |
+======+===============+============+=========+
| 0    |          1404 |         22 |    1426 |
+------+---------------+------------+---------+
| 1    |           103 |          9 |     112 |
+------+---------------+------------+---------+
| All  |          1507 |         31 |    1538 |
+------+---------------+------------+---------+
Chi-square statistic: 22.166715835448176
p-value: 1.5365928760235158e-05
Degrees of freedom: 2

There is a significant association between VD and fracture occurrence (reject H0)
```

Fig. 30. Chi-square test between VD and fracture.

We performed a chi-square test between OP and fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between OP and fracture.

```
chi_OP = pd.crosstab(df_chi.OP, df_chi.Fracture, margins=True)

values = chi_OP.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_OP.reset_index().values.tolist()
headers = ['OP', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between OP and fracture occurrence (reject H0)")
else:
    print("There is no significant association between OP and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+------+---------------+------------+---------+
| OP   |   No Fracture |   Fracture |   Total |
+======+===============+============+=========+
| 0    |           959 |         11 |     970 |
+------+---------------+------------+---------+
| 1    |           548 |         20 |     568 |
+------+---------------+------------+---------+
| All  |          1507 |         31 |    1538 |
+------+---------------+------------+---------+
Chi-square statistic: 10.335819579325578
p-value: 0.00569646319026625
Degrees of freedom: 2

There is a significant association between OP and fracture occurrence (reject H0)
```

Fig. 31. Chi-square test between OP and fracture.

We performed a chi-square test between CAD and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between CAD and fracture.

```
chi_CAD = pd.crosstab(df_chi.CAD, df_chi.Fracture, margins=True)

values = chi_CAD.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_CAD.reset_index().values.tolist()
headers = ['CAD', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between CAD and fracture occurrence (reject H0)")
else:
    print("There is no significant association between CAD and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+-------+--------------+------------+---------+
| CAD   |   No Fracture |   Fracture |   Total |
+=======+==============+============+=========+
| 0     |          1204 |         22 |    1226 |
+-------+--------------+------------+---------+
| 1     |           303 |          9 |     312 |
+-------+--------------+------------+---------+
| All   |          1507 |         31 |    1538 |
+-------+--------------+------------+---------+
Chi-square statistic: 1.4966092842914818
p-value: 0.4731680623170935
Degrees of freedom: 2

There is no significant association between CAD and fracture occurrence (fail to reject H0)
```

Fig. 32. Chi-square test between CAD and fracture.

We performed a chi-square test between CKD and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between CKD and fracture.

```
chi_CKD = pd.crosstab(df_chi.CKD, df_chi.Fracture, margins=True)

values = chi_CKD.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_CKD.reset_index().values.tolist()
headers = ['CKD', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between CKD and fracture occurrence (reject H0)")
else:
    print("There is no significant association between CKD and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+-------+---------------+------------+---------+
| CKD   |   No Fracture |   Fracture |   Total |
+=======+===============+============+=========+
| 0     |          1450 |         29 |    1479 |
+-------+---------------+------------+---------+
| 1     |            57 |          2 |      59 |
+-------+---------------+------------+---------+
| All   |          1507 |         31 |    1538 |
+-------+---------------+------------+---------+
Chi-square statistic: 0.5866703106497732
p-value: 0.7457721492995977
Degrees of freedom: 2

There is no significant association between CKD and fracture occurrence (fail to reject H0)
```

Fig. 33. Chi-square test between CKD and fracture.

We performed a chi-square test between Smoking and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between Smoking and fracture.

```
chi_Smoking = pd.crosstab(df_chi.Smoking, df_chi.Fracture, margins=True)

values = chi_Smoking.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_Smoking.reset_index().values.tolist()
headers = ['Smoking', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between Smoking and fracture occurrence (reject H0)")
else:
    print("There is no significant association between Smoking and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+-----------+---------------+------------+---------+
| Smoking   |   No Fracture |   Fracture |   Total |
+===========+===============+============+=========+
| 0         |          1114 |         28 |    1142 |
+-----------+---------------+------------+---------+
| 1         |           393 |          3 |     396 |
+-----------+---------------+------------+---------+
| All       |          1507 |         31 |    1538 |
+-----------+---------------+------------+---------+
Chi-square statistic: 4.273703561827226
p-value: 0.11802582991177724
Degrees of freedom: 2

There is no significant association between Smoking and fracture occurrence (fail to reject H0)
```

Fig. 34. Chi-square test between Smoking and fracture.

We performed a chi-square test between Drinking and fracture and got a p-value of >0.05, thus, we fail to reject the null hypothesis (HO) and conclude that there is no significant association between Drinking and fracture.

```
chi_Drinking = pd.crosstab(df_chi.Drinking, df_chi.Fracture, margins=True)

values = chi_Drinking.iloc[0:2, 0:5].values

chi2, p, dof, expected = chi2_contingency(values)

print("Contingency table:")
table = chi_Drinking.reset_index().values.tolist()
headers = ['Drinking', 'No Fracture', 'Fracture', 'Total']
print(tabulate(table, headers=headers, tablefmt='grid'))
print("Chi-square statistic:", chi2)
print("p-value:", p)
print("Degrees of freedom:", dof)
print("")

alpha = 0.05
if p < alpha:
    print("There is a significant association between Drinking and fracture occurrence (reject H0)")
else:
    print("There is no significant association between Drinking and fracture occurrence (fail to reject H0)")
```

```
Contingency table:
+------------+---------------+------------+---------+
| Drinking   |   No Fracture |   Fracture |   Total |
+============+===============+============+=========+
| 0          |          1160 |         28 |    1188 |
+------------+---------------+------------+---------+
| 1          |           347 |          3 |     350 |
+------------+---------------+------------+---------+
| All        |          1507 |         31 |    1538 |
+------------+---------------+------------+---------+
Chi-square statistic: 3.078995947918181
p-value: 0.21448875334234457
Degrees of freedom: 2

There is no significant association between Drinking and fracture occurrence (fail to reject H0)
```

Fig. 35. Chi-square test between Drinking and fracture.

5.3.2. Mann-Whitney U Test:

The Mann-Whitney U test is a statistical test used to compare the differences between two independent groups. This test helps to determine whether two groups of data are significantly different from each other or not by comparing the ranks of the data in each group and calculating a U statistic.

```
In [54]: df_mwu = df[['Age','BMI', 'FNT', 'TLT','Fracture']]
         df_mwu.head(2)
```

Out[54]:

	Age	BMI	FNT	TLT	Fracture
0	61.9	17.0	-2.95	-2.90	0
1	55.0	21.0	-1.30	-0.15	0

Fig. 36. Data preparation for Mann-Whitney U Test.

We performed a Mann-Whitney U test between Age with fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between OP and fracture.

```
from scipy.stats import mannwhitneyu
U, p_value = mannwhitneyu(df_mwu.Age, df_mwu.Fracture)
print("U-statistic:", U)
print("p-value:", p_value)

if p_value < 0.05:
    print('there is a significant association between the Age and fracture occurance(Reject H0)')
else:
    print('there is no significant association between the Age and fracture occurance(Fail to reject H0)')
```

```
U-statistic: 2365444.0
p-value: 0.0
there is a significant association between the Age and fracture occurance(Reject H0)
```

Fig. 37. Mann-Whitney U Test between Age and Fracture.

We performed a Mann-Whitney U test between BMI with fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between OP and fracture.

```
U, p_value = mannwhitneyu(df_mwu.BMI, df_mwu.Fracture)
print("U-statistic:", U)
print("p-value:", p_value)

if p_value < 0.05:
    print('there is a significant association between the BMI and fracture occurance(Reject H0)')
else:
    print('there is no significant association between the BMI and fracture occurance(Fail to reject H0)')
```

```
U-statistic: 2365444.0
p-value: 0.0
there is a significant association between the BMI and fracture occurance(Reject H0)
```

Fig. 38. Mann-Whitney U Test between BMI and Fracture.

We performed a Mann-Whitney U test between FNT with fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between OP and fracture.

```
U, p_value = mannwhitneyu(df_mwu.FNT, df_mwu.Fracture)
print("U-statistic:", U)
print("p-value:", p_value)

if p_value < 0.05:
    print('there is a significant association between the FNT and fracture occurance(Reject H0)')
else:
    print('there is no significant association between the FNT and fracture occurance(Fail to reject H0)')
```

```
U-statistic: 289799.5
p-value: 0.0
there is a significant association between the FNT and fracture occurance(Reject H0)
```

Fig. 39. Mann-Whitney U Test between FNT and Fracture.

We performed a Mann-Whitney U test between TLT with fracture and got a p-value of <0.05, thus, we reject the null hypothesis (HO) and conclude that there is a significant association between OP and fracture.

```
U, p_value = mannwhitneyu(df_mwu.TLT, df_mwu.Fracture)
print("U-statistic:", U)
print("p-value:", p_value)

if p_value < 0.05:
    print('there is a significant association between the TLT and fracture occurance(Reject H0)')
else:
    print('there is no significant association between the TLT and fracture occurance(Fail to reject H0)')
```

```
U-statistic: 489396.0
p-value: 2.7581395011572007e-198
there is a significant association between the TLT and fracture occurance(Reject H0)
```

Fig. 40. Mann-Whitney U Test between TLT and Fracture.

6. Data Visualization:

Data visualization is the process of presenting data in a graphical or visual format to help understand and identify patterns, trends, and relationships that might have significance on the study conclusions. We illustrated the data using histograms, pie chart, and bar charts.

6.1. Histogram representation of the continuous data:

```
df_hist = df[['Age','BMI','FNT','TLT']]
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,10))
for ax, col in zip(axes.flatten(), df_hist.columns):
    ax.hist(df_hist[col], bins=8, color='lightpink', edgecolor='black', lw=1)
    ax.set_xlabel(col)
    ax.set_ylabel('Frequency')

plt.show()
```

Fig. 41. Python code for plotting Histogram of the continuous data

The plot is generated using the matplotlib library in Python. The results show that the Age variable has a right-tailed distribution with a peak around the 60s. The BMI variable is right tailed indicating more patients are overweight. The FNT and TLT variable have a right-tailed distribution.

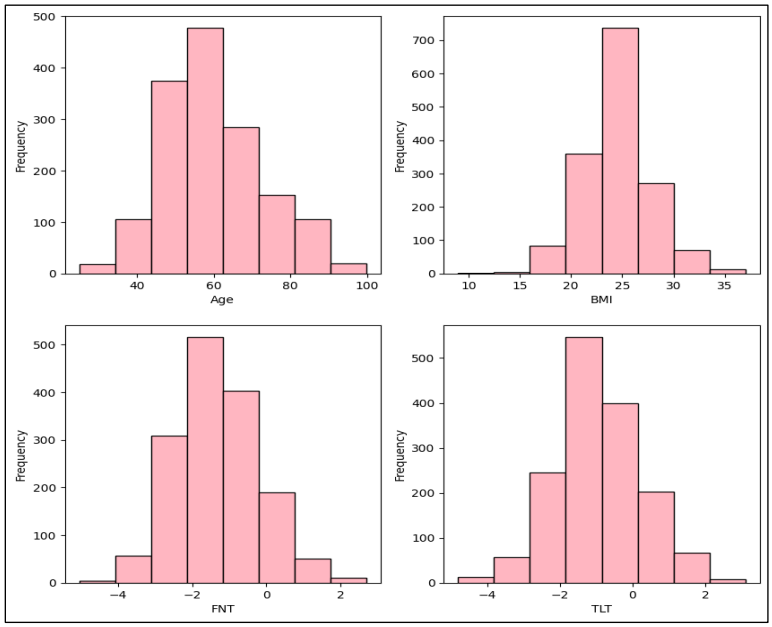


Fig.42. Histograms for Age, BMI, FNT, and TLT

6.2. Pie chart representation of Gender with Fracture:

The pie() function from the matplotlib library is then used to create the pie chart. It shows the distribution of genders for patients who are prone to fractures. Most of the patients are males, representing around 61.5% of the total patients. This analysis suggests that males may be more susceptible to fractures than females.

```
# create a new DataFrame with only male and female patients with fractures
df_pie = df[df['Fracture'] == 1]
df_pie = df[df['Gender'].isin([1, 2])]

# count the number of males and females
counts = df_pie['Gender'].value_counts()

# create a pie chart with the counts
labels = ['Male', 'Female']
colors = ['orange', 'purple']
plt.pie(counts, colors=colors, labels=labels, autopct='%1.1f%%', startangle=90)

# add a title to the chart
plt.title('Gender Distribution of Patients with Fractures')

# display the chart
plt.show()
```

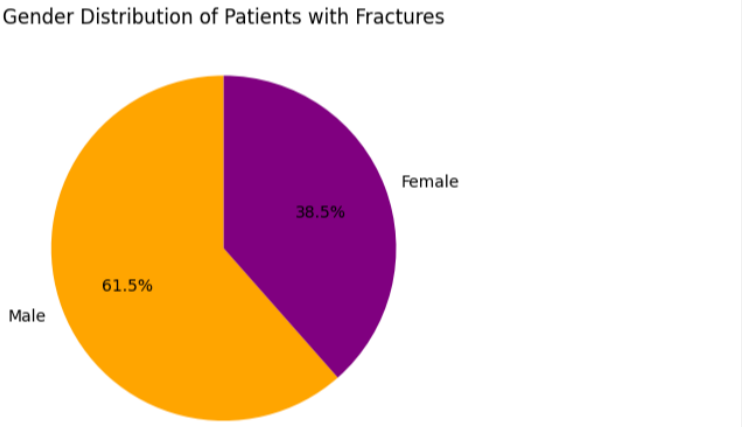


Fig. 43. Python code and Pie chart for Gender distribution.

6.3. Bar chart representation of comorbid conditions and lifestyle changes (smoking and drinking) with fracture:

The bar() function from the matplotlib library is used to create a bar chart with the comorbid conditions as the x-axis and the number of fractures as the y-axis. The resulting bar chart shows the distribution of fractures among patients with different comorbid conditions. The chart suggests that patients with hypertension (HTN), osteoporosis (OP), and chronic obstructive pulmonary disease (COPD) have the highest number of fractures among the comorbid conditions studied.

```
# subset data to only include rows with a fracture
fractures_Data = df[df['Fracture'] == 1]

# count the number of fractures in each medical condition column
fracture_counts = fractures_Data[['HTN','COPD','DM','VT','VD','OP','CAD','CKD','Smoking','Drinking']].sum()

# create the bar chart
plt.bar(fracture_counts.index, fracture_counts.values, color='brown')

# rotate x-axis labels by 45 degrees
plt.xticks(rotation=45, ha='right')

# add axis labels and title
plt.xlabel('Comorbidity')
plt.ylabel('Number of Fractures')
plt.title('Distribution of Fractures Among Comorbid Conditions')

# show the chart
plt.show()
```

Fig.44.a. Python code for Bar chart to illustrate the distribution of fracture with different comorbid conditions.

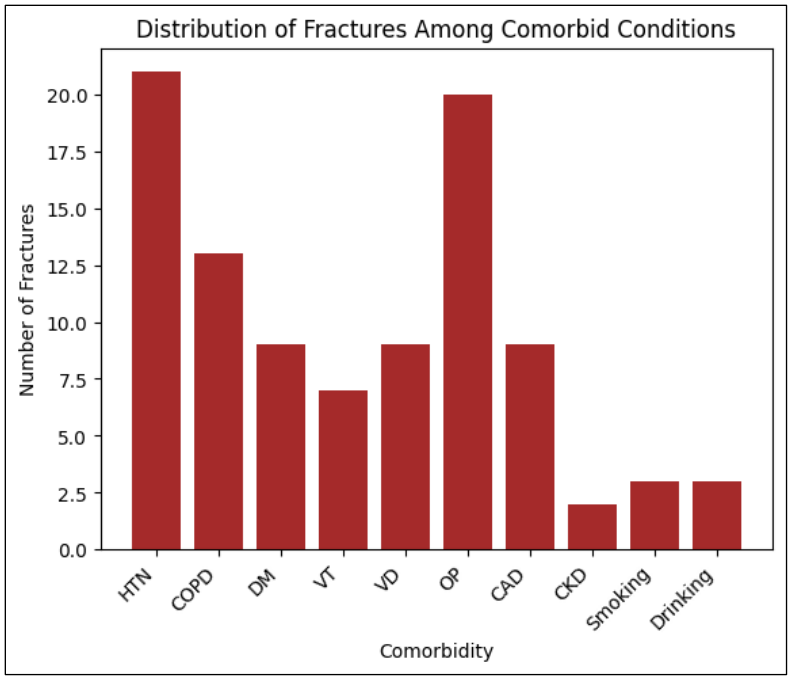


Fig.44.b. Bar chart to illustrate the distribution of fracture with different comorbid conditions.

6.4. Overall Histogram distribution of the data frame:

```
df_hist = df[['Gender', 'Age', 'BMI', 'FNT', 'TLT', 'HTN', 'COPD', 'DM', 'VT', 'VD', 'OP', 'CAD', 'CKD',
              'Fracture', 'Smoking', 'Drinking']]
df_hist.hist(figsize= [12,12], bins=6, grid=False, color='yellow', edgecolor='black', lw=1)
plt.show()
```

Fig.45.a. Python code to plot histograms for all the desired attributes in the data set.

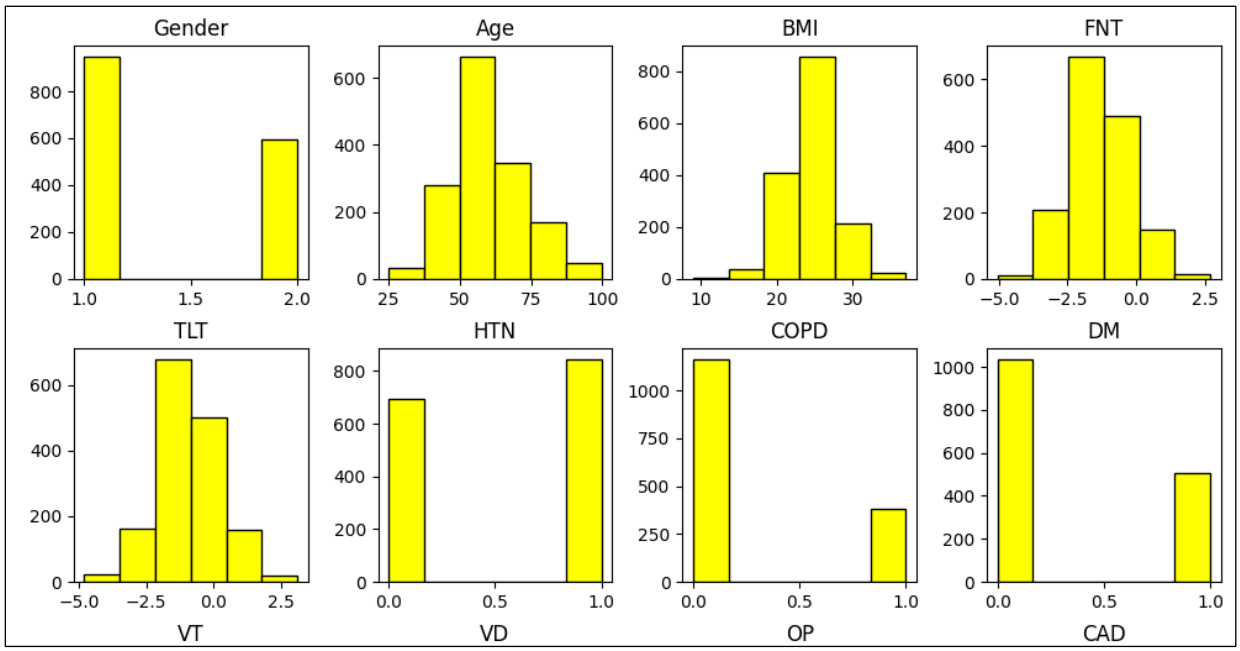


Fig. 45.b. Histograms for all the desired attributes.

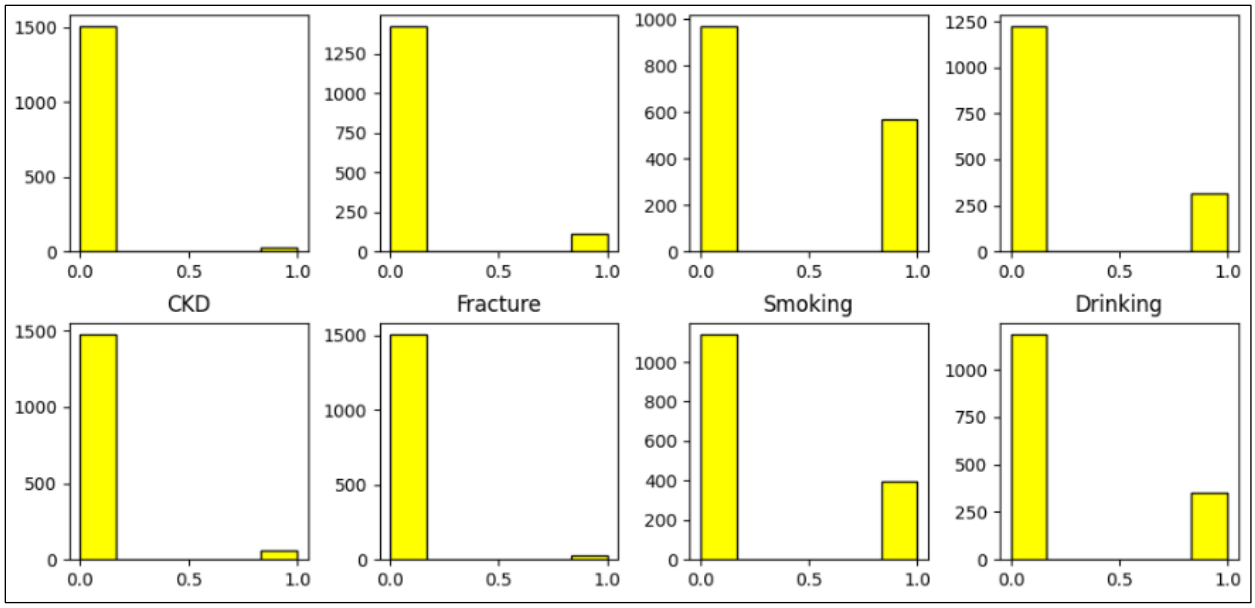


Fig. 45.c. Histograms for all the desired attributes.

6.5. Spearman's Correlation Heat Map:

Spearman's correlation coefficient is a non-parametric measure of the correlation between two variables, which means that it does not rely on assumptions about the underlying distribution of the data. This makes it suitable for use with non-normal data distributions, including those that are positively skewed data as well and it can help to identify patterns and relationships that might not be apparent from looking at the data. Spearman's correlation map uses a color scale to represent the strength and direction of the correlation between two variables. Positive correlations are represented by warm colors (such as Red), while negative correlations are represented by cool colors (such as Purple). A neutral or weak correlation is represented by white or light colors.

Correlation using heat map

```
corr_matrix = df.corr('spearman')
fig, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(corr_matrix, annot=True, cmap='PuRd', ax=ax)
plt.show()
```

Fig. 46. Python code for plotting heatmap using Spearman correlation test.

As per Spearman's correlation heatmap, we conclude that VT (0.22), VD (0.12), and OP (0.082) have a weak positive correlation with Fracture.

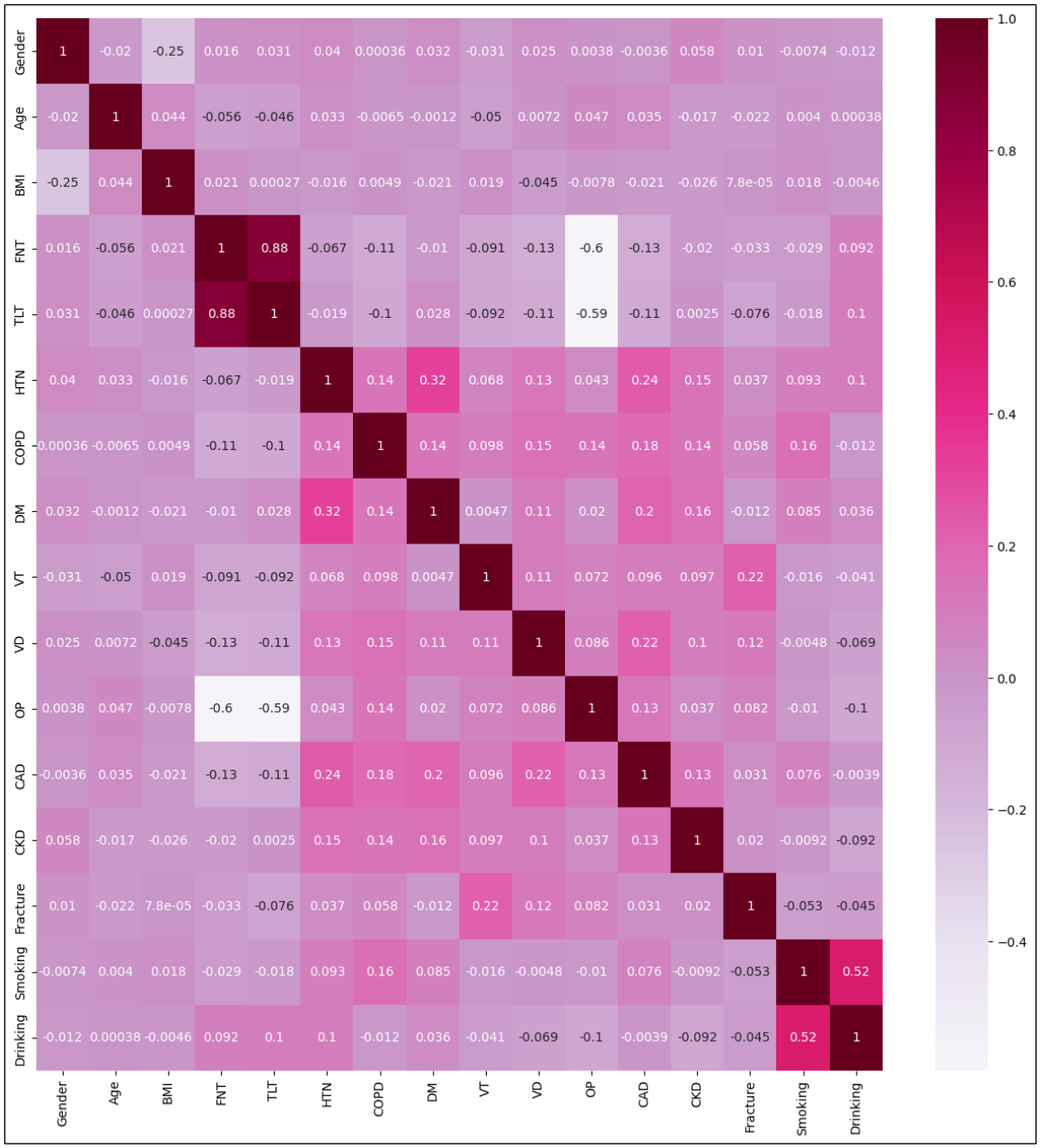


Fig.47. Correlation Heatmap using Spearman correlation test.

7. Machine Learning Models:

Machine learning models are algorithms that enable machines to learn and make predictions or decisions based on data. These models are designed to find patterns in data and use those patterns to make accurate predictions or decisions. Creating visualizations for machine learning in Python is made simple with the help of the Scikit-plot Python package. It is designed on top of Matplotlib and offers tools for producing many kinds of graphs, including ROC curves and confusion matrices.

```
In [59]: pip install scikit-plot
```

Fig.48. Python code to install scikit-plot package.

The Python module imblearn offers resources for addressing unbalanced datasets in machine learning. It includes several methods, like oversampling, under-sampling, and the creation of synthetic samples, for dealing with unbalanced datasets. Since our dataset was imbalanced, we used this module to address this concern. This helped us in dividing the data equally between test and train data sets respectively, for building machine learning models.

```
In [60]: !pip install imblearn
```

Fig.49. Python code to install imblearn package.

For building the machine learning models, we installed the desired packages in Python from the Python library.

```
In [61]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn import preprocessing
         from imblearn.over_sampling import SMOTENC
         from sklearn.naive_bayes import GaussianNB
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.linear_model import LinearRegression
         from sklearn.linear_model import LogisticRegression
         from xgboost import XGBClassifier
         from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
         from sklearn import metrics
         import scikitplot as skplt
         from sklearn.metrics import plot_roc_curve
         from sklearn.metrics import roc_curve, auc
```

Fig. 50. Python codes to import relevant machine learning model packages.

We conducted a feature importance analysis using the Random Forest Regressor from Scikit-Learn. The data was divided into features and target variables. A model with 100 trees and a random state of 42 was then developed, and the model was fitted to the data. Each feature's significance value and associated column name from the original data frame were included in the ranking, which was done in descending order of importance.

The output shows that the top features that contribute most to the prediction of 'Fracture' are Age, FNT, TLT, BMI, VT and HTN. This indicates that these three features are most useful in predicting if a patient is likely to experience a fracture.

Preparing data For Machine learning models

```
In [62]: from sklearn.ensemble import RandomForestRegressor
         import pandas as pd

         df_m = df

         # split the data into features (X) and target (y)
         X = df.drop('Fracture', axis=1)
         y = df['Fracture']

         # create the model
         model = RandomForestRegressor(n_estimators=100, random_state=42)

         # fit the model to the data
         model.fit(X, y)

         # calculate feature importance
         importances = model.feature_importances_

         # sort the features by importance
         indices = importances.argsort()[::-1]

         # print the feature ranking
         print("Feature ranking:")

         for f in range(X.shape[1]):
             print(f"{f+1}. {X.columns[indices[f]]}: {importances[indices[f]]}")

         Feature ranking:
         1. Age: 0.21483042010422845
         2. FNT: 0.20222490183730066
         3. TLT: 0.19812367556275293
         4. BMI: 0.1250170662900543
         5. VT: 0.04961737501567083
         6. HTN: 0.04079313745918612
         7. OP: 0.03148135109993465
         8. DM: 0.02614371184049996
         9. VD: 0.021477952582778168
         10. Gender: 0.021205161645851754
         11. COPD: 0.02084826016220691
         12. CAD: 0.01687541179804779
         13. Smoking: 0.011876591084005982
         14. Drinking: 0.010502412937516544
         15. CKD: 0.008982570579965264
```

Fig.51. Python code to prepare data for machine learning models.

We then plot the bar chart to visualize the prior feature importance determined with a random forest regressor. The results of the plot show the relative importance of each feature in the model, with the most important feature (Age) having an importance of about 0.21 and the least important feature (CKD) having an importance of about 0.01.

This information can be used to identify the most important predictors of fractures in the dataset, which can be useful for improving the accuracy of the model or for making predictions about future cases.

```
import matplotlib.pyplot as plt

plt.bar(range(X.shape[1]), importances[indices])

plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)

plt.title("Features Importances")
plt.xlabel("Features")
plt.ylabel("Importance")

plt.show()
```

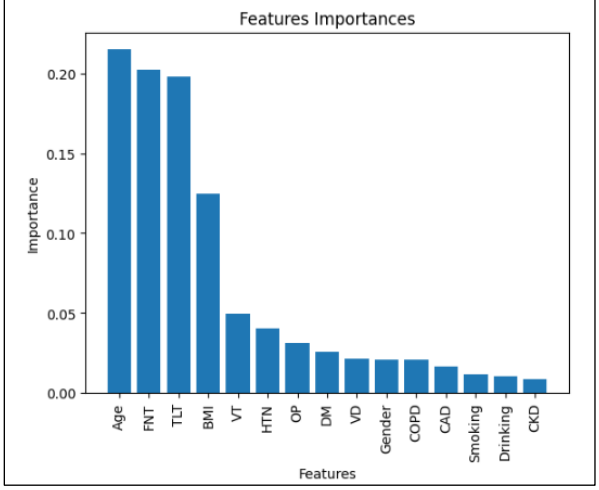


Fig.52. Bar chart to illustrate the feature importance.

The values of a few columns (Age, BMI, FNT, TLT, VT, and HTN) in the input data frame (df) are normalized to a range of 0–1 using the MinMaxScaler object from sklearn.preprocessing. A new data frame (Model_df) that contains the target variable (Fracture) is created and contains the normalized values. Each column in the data is transformed using the fit_transform() technique.

```
from sklearn.preprocessing import MinMaxScaler
Model_df = df[['Age','BMI', 'FNT', 'TLT','VT','HTN','Fracture']]
# create a scaler object
scaler = MinMaxScaler()

# Fiting age column from 0-1
Model_df['Age'] = scaler.fit_transform(Model_df['Age'].values.reshape(-1, 1))
Model_df['BMI'] = scaler.fit_transform(Model_df['BMI'].values.reshape(-1, 1))
Model_df['FNT'] = scaler.fit_transform(Model_df['FNT'].values.reshape(-1, 1))
Model_df['TLT'] = scaler.fit_transform(Model_df['TLT'].values.reshape(-1, 1))
Model_df['VT'] = scaler.fit_transform(Model_df['VT'].values.reshape(-1, 1))
Model_df['HTN'] = scaler.fit_transform(Model_df['HTN'].values.reshape(-1, 1))


# Resulting dataframe
print(Model_df)
```

```
          Age       BMI       FNT       TLT   VT  HTN  Fracture
0     0.493316  0.285714  0.270968  0.240506  0.0  1.0         0
1     0.401070  0.428571  0.483871  0.588608  0.0  1.0         0
2     0.254011  0.428571  0.154839  0.158228  0.0  1.0         0
3     0.530749  0.535714  0.374194  0.430380  0.0  1.0         0
4     0.848930  0.464286  0.406452  0.462025  0.0  1.0         0
...        ...       ...       ...       ...  ...  ...       ...
1533  0.347594  0.500000  0.645161  0.778481  0.0  1.0         0
1534  0.288770  0.535714  0.529032  0.594937  0.0  0.0         0
1535  0.401070  0.500000  0.761290  0.784810  0.0  0.0         0
1536  0.267380  0.714286  0.464516  0.443038  0.0  1.0         0
1537  0.000000  0.546574  0.483774  0.489641  0.0  0.0         0

[1538 rows x 7 columns]
```

Fig.53. Python code for MinMaxScaler

As per the feature importance ranking, we selected the first six attributes to build our machine-learning models. “y” is the target variable (Fracture) and “x” contains the selected features for the model, which are Age, BMI, FNT, TLT, HTN, and VT. “train_test_split” function from scikit-learn is used to split the data into training and testing sets. “x_train” and “y_train” represent the training data, which is used to train the model, while “x_test” and “y_test” represent the testing data, which is used to evaluate the performance of the model. The data is split into 80-20 training and test respectively.

```
#Defining dependent and independent varaibles
y = Model_df['Fracture']
x = Model_df.filter(['Age','BMI', 'FNT', 'TLT','HTN', 'VT'])

 #Training data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=20)
```

Fig.54. Python code to define the dependent and independent variables.

The below code is used to perform oversampling using SMOTE algorithm to balance the class distribution in the training data, the code is used to address the class imbalance in the original dataset by oversampling the minority class using SMOTE, which helps to improve the performance of machine learning models by reducing bias towards the majority class and improving the overall accuracy and predictive power of the model.

```
from imblearn.over_sampling import SMOTE

x_train_resampled, y_train_resampled = SMOTE().fit_resample(x_train, y_train)
x_train, x_test, y_train, y_test = train_test_split(x_train_resampled, y_train_resampled, test_size=0.20, random_state=30)
```

Fig. 55. Python code to balance the samples in test and training data respectively.

Overall, this code confirms that the SMOTE technique was successful in balancing the data by oversampling the minority class (1 - Fracture) to match the majority class (0 - No Fracture)

```
print(y_train_resampled.shape)
y_train_resampled.value_counts()
```

```
(2406,)

0    1203
1    1203
Name: Fracture, dtype: int64
```

Fig.56. Python code to confirm the balanced sampling

We utilized a classification-based machine learning model since the goal variable in classification issues is categorical.

7.1. Logistic Regression:

Logistic regression is a statistical model used to predict the likelihood or probability of a binary outcome that helps to understand the relationship between a set of independent variables and a dependent variable. The test works by calculating the probability of the binary outcome based on the values of the independent variables.

The logistic regression model is created using the LogisticRegression() function. Then, 5-fold cross-validation is performed on the training data using the cross_val_score() function. The accuracy score of the logistic regression model is then calculated using the accuracy_score(). Overall, the model achieves an accuracy of 67.01%, with precision and recall values of around 63-72% for one class and 57-78% for the other. The f1-score, which is the mean of precision and recall, is around 0.74 and 0.64 for 0 and 1 respectively.

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression


# logistic regression model
logisticRegr = LogisticRegression()

# 5-fold cross-validation
cv_scores = cross_val_score(logisticRegr, x_train, y_train, cv=5)

# cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())

# Fit the logistic regression model on the training data
logisticRegr.fit(x_train, y_train)

# Predict on the test data
lg_pred = logisticRegr.predict(x_test)
from sklearn.metrics import accuracy_score
lg_accuracy = accuracy_score(y_test, lg_pred) * 100
print('Accuracy: ', lg_accuracy)


from sklearn.metrics import classification_report
print(classification_report(y_test, lg_pred))
```

```
Cross-validation scores: [0.66493506 0.66753247 0.62077922 0.6        0.65104167]
Mean cross-validation score: 0.6408576839826841
Accuracy:  67.01244813278008
              precision    recall  f1-score   support

           0       0.63      0.78      0.70       237
           1       0.72      0.57      0.64       245

    accuracy                           0.67       482
   macro avg       0.68      0.67      0.67       482
weighted avg       0.68      0.67      0.67       482
```

Fig.57. Python code for Logistic Regression

```python
#Constructing confusion matrix
plt.figure(figsize=(9,9))
conf_mat = metrics.confusion_matrix(y_test, lg_pred)
sns.heatmap(conf_mat, annot=True, fmt=".5f", linewidths=.7, square=True);
plt.xlabel('Predicted value');
plt.ylabel('Given values');
plt.title('Confusion matrix - Logistic Regression')
```

```
Text(0.5, 1.0, 'Confusion matrix - Logistic Regression')
```

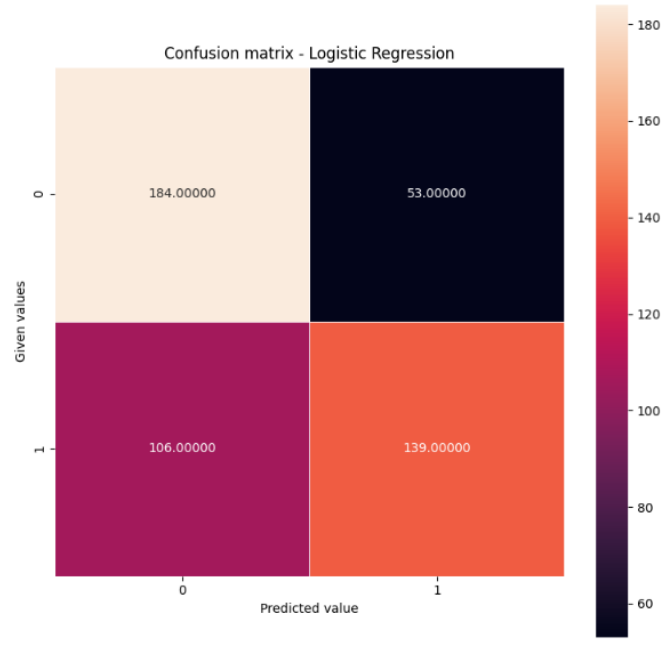


Fig. 58. Confusion Matrix for Logistic Regression

7.2. Decision Tree:

A decision tree is a graphical representation of the decision-making process, which involves a number of options and potential outcomes. Each internal node represents a choice or decision, each branch indicates an effect or result, and each leaf node represents the ultimate choice or result. Decision trees are frequently used in machine learning as a classification algorithm, with the objective of predicting the class or category of an object based on a set of features. To perform this, the data is divided into subsets based on the values of the features, until each subset only contains objects belonging to the same class.

The accuracy of the decision tree classifier model is 91.49%. The model's accuracy, recall, and f1-score are all high for both the positive and negative classes, showing that the model effectively classified the data. Cross-validation results reveal that the model's mean accuracy is 91%, which is fairly satisfactory, indicating that the model is stable and not overfitting the training data.

```
#Decision tree
from sklearn.tree import DecisionTreeClassifier

#fit the model
dtc = DecisionTreeClassifier()

dtc_scores = cross_val_score(dtc, x_train, y_train, cv=5)
print("DTC Classification Model")
print("CV scores: ", dtc_scores)
print("Mean CV accuracy: {:.2f}".format(dtc_scores.mean()))
print(" ")

dtc.fit(x_train, y_train)
dtc_pred = dtc.predict(x_test)

#Model performance
print("Decision Tree Classification Model")
#print('Mean squared error: %.2f' % mean_squared_error(y_test, dtc_pred))
#print('Coefficient of determination: %.2f' % r2_score(y_test, dtc_pred))

from sklearn.metrics import accuracy_score
dtc_accuracy = accuracy_score(y_test, dtc_pred) * 100
print('Accuracy: ', dtc_accuracy)

#Bar plot of the model performance
from sklearn.metrics import classification_report
print(classification_report(y_test, dtc_pred))
print("")
```

```
DTC Classification Model
CV scores:  [0.8987013  0.93506494 0.90909091 0.92727273 0.8671875 ]
Mean CV accuracy: 0.91

Decision Tree Classification Model
Accuracy:  91.49377593360995
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       237
           1       0.91      0.93      0.92       245

    accuracy                           0.91       482
   macro avg       0.92      0.91      0.91       482
weighted avg       0.92      0.91      0.91       482
```

Fig. 59. Python code for Decision Tree

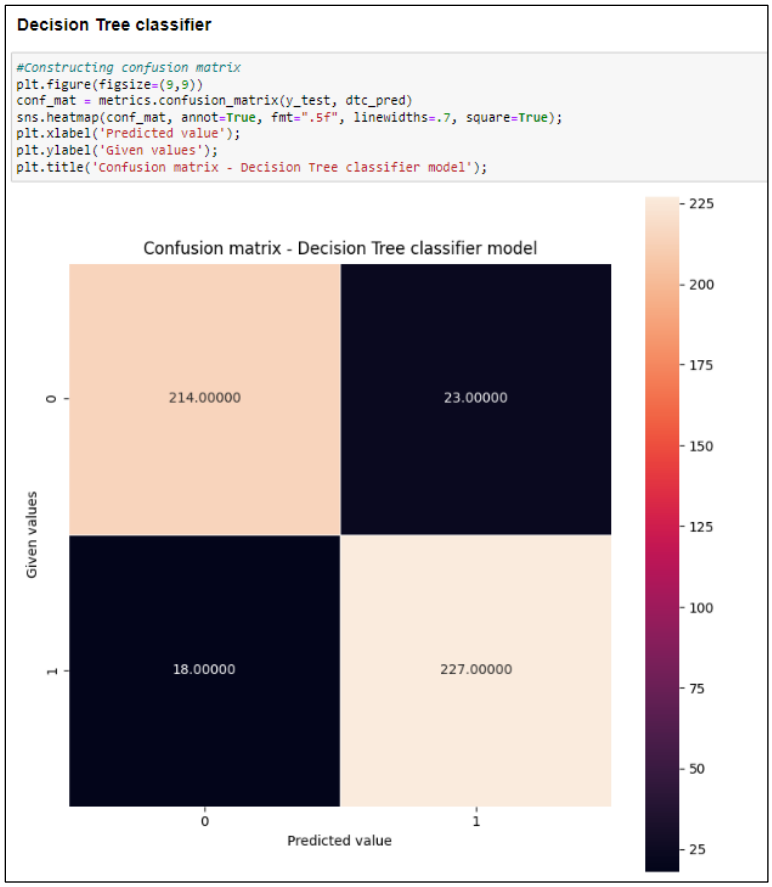


Fig.60. Confusion Matrix for Decision Tree

7.3. Gaussian Naïve Bayes:

Gaussian Naive Bayes is a machine learning algorithm that is commonly used for classification tasks. It's based on Bayes' theorem, which describes the probability of an event based on prior knowledge or evidence and helps to classify data into different categories by calculating the probability of each category based on the features of the data by assuming that the features are independent of each other and that they follow a normal (Gaussian) distribution.

Using 6-fold cross-validation, the Gaussian Naive Bayes classifier model was trained and assessed. The mean cross-validation accuracy was shown to be 0.57, which is poor in comparison to the other models. This suggests that the model could have difficulty estimating the target variable.

The classification accuracy of the Gaussian Naive Bayes classifier model was 56.64%. For classes 0 and 1, the accuracy and recall were 0.53 and 0.98 and 0.89 and 0.17, respectively. Class 0 and 1 had f1 scores of 0.69 and 0.28, respectively. The model's overall performance is not sufficient, as indicated by the macro-average f1-score of 0.49 and the weighted-average f1-score of 0.48.

Gaussian Naive Bayes Classifier

```
#Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB


GNB = GaussianNB()
GNB.fit(x_train, y_train)


GNB_scores = cross_val_score(GNB, x_train, y_train, cv=6)
print("GradientBoostingClassifier Model")
print("CV scores: ", GNB_scores)
print("Mean CV accuracy: {:.2f}".format(GNB_scores.mean()))
print(" ")
```

```
GradientBoostingClassifier Model
CV scores:  [0.58255452 0.55140187 0.54517134 0.59501558 0.559375   0.584375  ]
Mean CV accuracy: 0.57
```

Gaussian Naive Bayes Classifier Model

```
GNB.fit(x_train, y_train)
GNB_pred = GNB.predict(x_test)

#Model performance
print("Gaussian Naive Bayes classifier Model")
from sklearn.metrics import accuracy_score
GNB_accuracy = accuracy_score(y_test, GNB_pred) * 100
print('Accuracy: ', GNB_accuracy)

#Bar plot of the model performance
from sklearn.metrics import classification_report
print(classification_report(y_test, GNB_pred))
print("")
```

```
Gaussian Naive Bayes classifier Model
Accuracy:  56.63900414937759
              precision    recall  f1-score   support

           0       0.53      0.98      0.69       237
           1       0.89      0.17      0.28       245

    accuracy                           0.57       482
   macro avg       0.71      0.57      0.49       482
weighted avg       0.71      0.57      0.48       482
```

Fig. 61. Python code for Gaussian Naïve Bayes Classifier Model

Naive Bayes classifier

```
#Constructing confusion matrix
plt.figure(figsize=(9,9))
conf_mat = metrics.confusion_matrix(y_test, GNB_pred)
sns.heatmap(conf_mat, annot=True, fmt=".5f", linewidths=.7, square=True);
plt.xlabel('Predicted value');
plt.ylabel('Given values');
plt.title('Confusion matrix - Naive Bayes classifier')
```

```
Text(0.5, 1.0, 'Confusion matrix - Naive Bayes classifier')
```

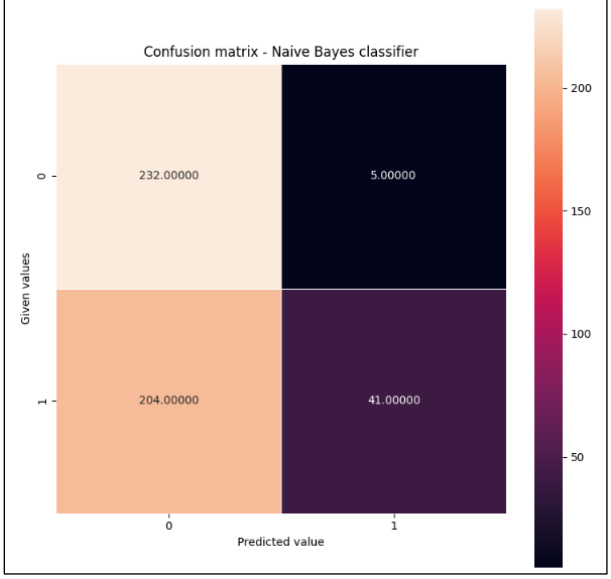


Fig. 62. Confusion Matrix for Gaussian Naïve Bayers Classifier model

7.4. Conclusion for Machine Learning Models:

We plotted a bar chart to depict which machine learning model gave higher accuracy. Depending on the below bar chart we conclude that Decision Tree Classification model was the best based on the accuracy followed by Logistic Regression and Gaussian Naïve Bayers respectively.

```
#Bar plot of to compare the model performance
models = ['Logistic Regression', 'GaussianNB','Decision Tree Classification']
accuracy = [lg_accuracy, GNB_accuracy,dtc_accuracy]
plt.bar(models, accuracy)
plt.title('Model Performance')
plt.xlabel('Models')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.show()
```

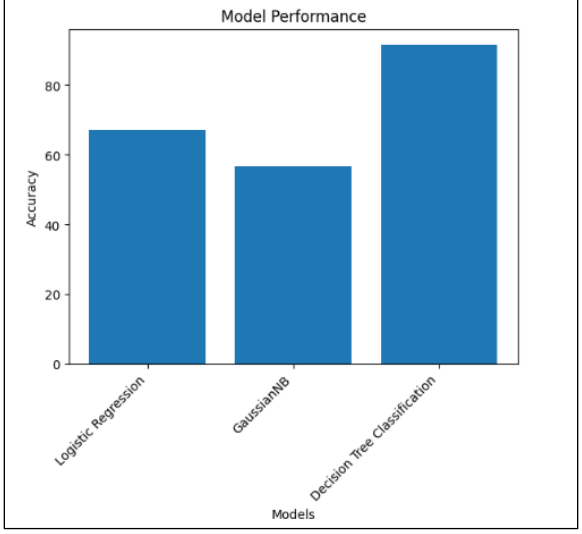


Fig. 63. Graph showing the accuracy of the models.

Conclusion of Machine Learning Models:

Depending on the accuracy, precision, recall, F1 score, and support matrix, we conclude that the Decision tree model was the best among the three models.

```
The accuracy of the logistic regression model is:  67.01244813278008
The accuracy of the Decision tree model is:  91.49377593360995
The accuracy of the Gaussian Naive Bayes Classifier model is:  56.63900414937759
```

Fig. 64. Accuracies of the models. The decision tree model has the highest accuracy.

model	class	precision	recall	f1-score	support
Logistic Regression	0	0.634483	0.776371	0.698292	237.000000
	1	0.723958	0.567347	0.636156	245.000000
	accuracy	0.670124	0.670124	0.670124	0.670124
	macro avg	0.679221	0.671859	0.667224	482.000000
	weighted avg	0.679963	0.670124	0.666708	482.000000
Decision Tree	0	0.922414	0.902954	0.912580	237.000000
	1	0.908000	0.926531	0.917172	245.000000
	accuracy	0.914938	0.914938	0.914938	0.914938
	macro avg	0.915207	0.914742	0.914876	482.000000
	weighted avg	0.915087	0.914938	0.914914	482.000000
Gaussian Naive Bayes	0	0.532110	0.978903	0.689450	237.000000
	1	0.891304	0.167347	0.281787	245.000000
	accuracy	0.566390	0.566390	0.566390	0.566390
	macro avg	0.711707	0.573125	0.485619	482.000000
	weighted avg	0.714688	0.566390	0.482235	482.000000

Fig. 65. Detailed scores of precision, recall, F1 score, and support matrix for all the models.

We then plotted receiver operating characteristics (ROC) curves for all the above models to check for the area under the curve (AUC). Based on the below figure we conclude that the decision tree model was the best among the three models.

```
# Probabilities for positive class
lg_probs = logisticRegr.predict_proba(x_test)[:, 1]

# Calculate false positive rate (FPR), true positive rate (TPR), and thresholds
fpr_lg, tpr_lg, thresholds_lg = roc_curve(y_test, lg_probs)

# Predicted probabilities for test data
GNB_probs = GNB.predict_proba(x_test)[:, 1]

# False positive rate and true positive rate
fpr_GNB, tpr_GNB, thresholds_svm = roc_curve(y_test, GNB_probs)

# Area under the curve
auc_GNB= roc_auc_score(y_test, GNB_probs)

# Predict probabilities for positive class
dtc_probs = dtc.predict_proba(x_test)[:, 1]

# Compute false positive rate (FPR), true positive rate (TPR), and thresholds
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_test, dtc_probs)

# Compute Area Under the ROC Curve (AUC)
auc_dtc = roc_auc_score(y_test, dtc_probs)

# Plot ROC curve for all models
plt.plot(fpr_lg, tpr_lg, label=f'Logistic Regression (AUC = {auc_score:.2f})')
#plt.plot(fpr_rfc, tpr_rfc, label=f'Random Forest (AUC = {auc_rfc:.2f})')
plt.plot(fpr_GNB, tpr_GNB, label=f'Gaussian Naive Bayes (AUC = {auc_GNB:.2f})')
plt.plot(fpr_dtc, tpr_dtc, label=f'Decision Tree (AUC = {auc_dtc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

Fig.66. Python code to plot ROC for all the models.

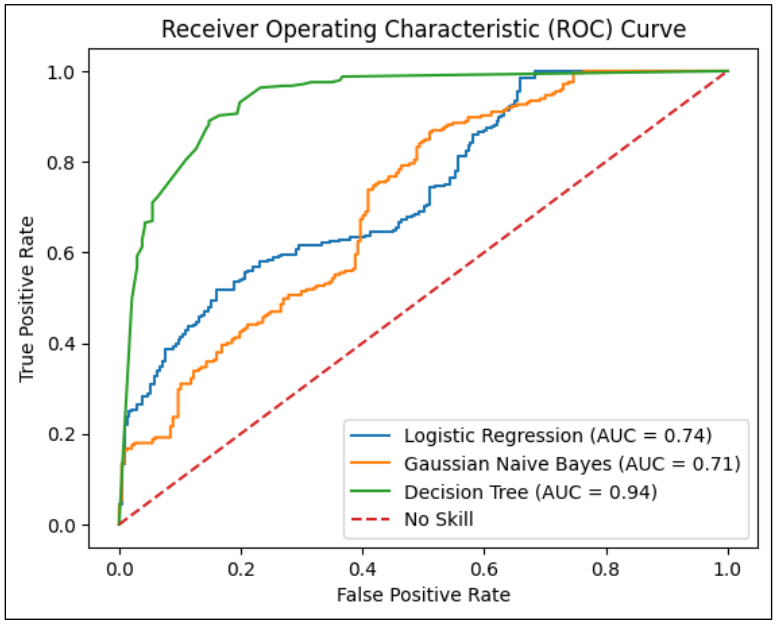


Fig.67. ROC Curves for Logistic Regression, Gaussian Naïve Bayes and Decision tree model

8. Limitations:

1. While we endeavored to conduct a rigorous analysis of the dataset, we must acknowledge certain limitations in our study. Specifically, despite the recent vintage of the dataset, we found that many of the available columns were not fitting appropriate to our proposed hypotheses, thus we were compelled to utilize only 16 out of the 40 columns. Regrettably, this limited scope for conducting statistical analysis may have impacted the robustness of our conclusions. Nonetheless, we maintain confidence in the results we have obtained to date.
2. Although we conducted extensive statistical analyses, the input and valuable feedback from a data science expert could have provided more insight and improved the robustness of the study. We recognize this as a potential area for improvement in future studies, and we hope to seek expert consultation to address this limitation in our future projects.

9. Appendix

9.1 Appendix for SQL Query:

Update the datatypes in MySQL phpMyAdmin:

```
UPDATE Grp3_Bonefractures SET Age = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET Height = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET Weight = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET BMI = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET ALT = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET AST = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET BUN = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET CREA = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET FBG = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET HDL-C = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET LDL-C = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET Ca = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET P = CAST (Age AS FLOAT);

UPDATE Grp3_Bonefractures SET Mg = CAST (Age AS FLOAT);
```

9.2. Appendix for Python:

```
#Random Forest Classification
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=10)

rfc_scores = cross_val_score(rfc, x_train, y_train, cv=6)
print("Random Forest Classification Model")
print("CV scores: ", rfc_scores)
print("Mean CV accuracy: {:.2f}".format(rfc_scores.mean()))
print(" ")

rfc.fit(x_train, y_train)
rfc_pred = rfc.predict(x_test)

from sklearn.metrics import accuracy_score
rfc_accuracy = accuracy_score(y_test, rfc_pred) * 100
print('Accuracy: ', rfc_accuracy)

#Bar plot of the model performance
from sklearn.metrics import classification_report
print(classification_report(y_test, rfc_pred))
print("")
```

```
Random Forest Classification Model
CV scores:  [0.96884735 0.9470405  0.95638629 0.9376947  0.934375   0.896875  ]
Mean CV accuracy: 0.94

Accuracy:  95.22821576763485
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       237
           1       0.98      0.93      0.95       245

    accuracy                           0.95       482
   macro avg       0.95      0.95      0.95       482
weighted avg       0.95      0.95      0.95       482
```

Fig. 68. Python code for Random Forest Classifier model

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Probabilities for positive class
rfc_probs = rfc.predict_proba(x_test)[:, 1]

# Calculate false positive rate (FPR), true positive rate (TPR), and thresholds
fpr_rfc, tpr_rfc, thresholds_rfc = roc_curve(y_test, rfc_probs)

# AUC score
auc_rfc = roc_auc_score(y_test, rfc_probs)

# Plot ROC curve
plt.plot(fpr_rfc, tpr_rfc, label='Random Forest Classification (AUC = {:.2f})'.format(auc_rfc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Random Forest Classification')
plt.legend()
plt.show()
```

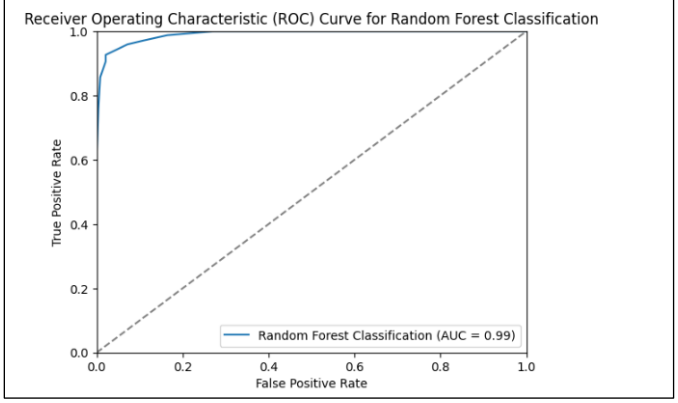


Fig. 69. Python code and ROC curve for Random Forest Classifier model.

10. Summary of the findings:

Through the study's methodology of data cleaning, analysis, and visualizations we came up with the following conclusions.

1. Based on the statistical analysis like Mann-Whitney test and Chi-square tests we found that the factors such as Age, Gender, Bone Mineral Density, BMI, and comorbidities like Diabetes, Coronary artery diseases, Coronary kidney disease, Osteoporosis, chronic kidney disease, and chronic obstructive pulmonary disease do have influence bone fractures.
2. Since one of our aims was to identify the best predictive model using machine learning techniques and the Precision, Recall, F-1 and Support matrix we found that the Decision Tree Classification fits best to our expectations.

11. Abbreviations:

- CKD- chronic kidney disease
- CAD- coronary artery disease
- VT- Venous Thromboembolism
- VD- Valve Degeneration
- OP- Osteoporosis
- AS- Atherosclerosis
- HTN- Hypertension
- DM- Diabetes Mellitus
- COPD- Chronic Obstructive Lung Disease
- ML-Machine Learning
- BMI-Body Mass index
- BMD-Bone Mineral density
- FNT- T score for Femoral Neck
- TLT- T score for Total Proximal Femur
- AST- Alanine Transferase
- ALT-Alanine Aminotransferase
- HDL-C-Hight Density lipoprotein cholesterol
- LDL-C-Low Density lipoprotein cholesterol
- CREA-Creatinine
- URIC-Uric acid
- BUN-Blood Urea Nitrogen
- Ca-Calcium
- P-Phosphorous
- Mg-Magnesium
- Zn- Zinc
- Cu-copper
- Fe-Iron

12. References:

1. GBD 2019 Fracture Collaborators. (2021). Global burden of 369 diseases and injuries in 204 countries and territories, 1990-2019: A systematic analysis for the Global Burden of Disease Study 2019. The Lancet, 398(10298), 1789-1858. https://doi.org/10.1016/S0140-6736(21)00198-4
2. Dufour, A. B., Hannan, M. T., Murabito, J. M., & McLean, R. R. (2021). Diabetes, bone mineral density, and fracture risk: A systematic review and meta-analysis. Journal of Bone and Mineral Research, 36(3), 424-436. https://doi.org/10.1002/jbmr.4209
3. Carmo, L. S. D., Moreira, L. D. P., de Oliveira, M. L. B., de Souza, A. L. A., & da Veiga, G. L. (2020). Hypertension and bone metabolism: A review. Clinical Rheumatology, 39(9), 2599-2607. https://doi.org/10.1007/s10067-020-05077-5

4. Ciosek, J. (2021). Bone metabolism and mineralization. Advances in Clinical Chemistry, 101, 131-158. https://doi.org/10.1016/bs.acc.2020.10.002
5. Johansson, H., Kanis, J. A., Oden, A., McCloskey, E., & Lorentzon, M. (2014). Body mass index, obesity and fracture risk: A meta-analysis. Osteoporosis International, 25(1), 49-58. https://doi.org/10.1007
6. Yang, C. Y., Cheng-Yen Lai, J., Huang, W. L., Hsu, C. L., & Chen, S. J. (2021). Effects of sex, tobacco smoking, and alcohol consumption osteoporosis development: Evidence from Taiwan biobank participants. *Tobacco induced diseases*, *19*, 52. https://doi.org/10.18332/tid/136419
7. Sözen, T., Özışık, L., & Başaran, N. Ç. (2017). An overview and management of osteoporosis. *European journal of rheumatology*, *4*(1), 46–56. https://doi.org/10.5152/eurjrheum.2016.048