Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
data.describe() # computes summary values for continous column data
```

	age	blood_pressure	specific_gravity	albumin	sugar	blood glucose random	blood_urea	serum_creatinine	sodium
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

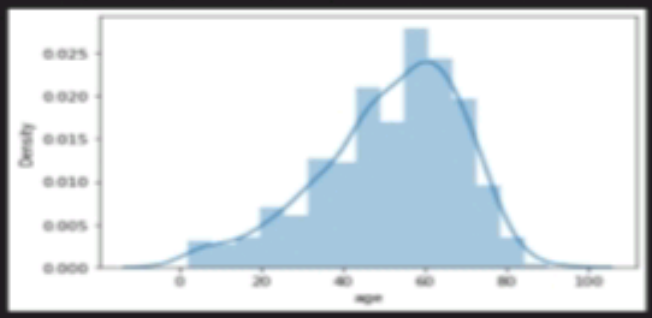
The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature.

Age distribution

```
sns.distplot(data.age)
```

```
C:\Users\Saumya\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWa
your code to use either `displot` (a figure-level function with similar flexibility
  warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='age', ylabel='Density'>
```

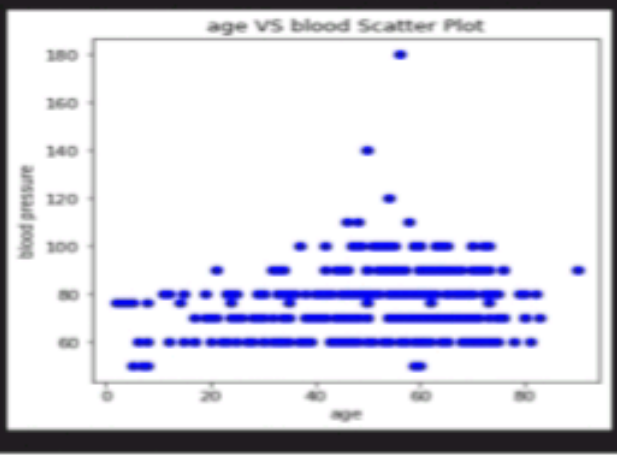


Activity 2.2: Bivariate analysis

Age vs Blood Pressure

```
import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes
```

```
Text(0.5, 1.0, 'age VS blood Scatter Plot')
```



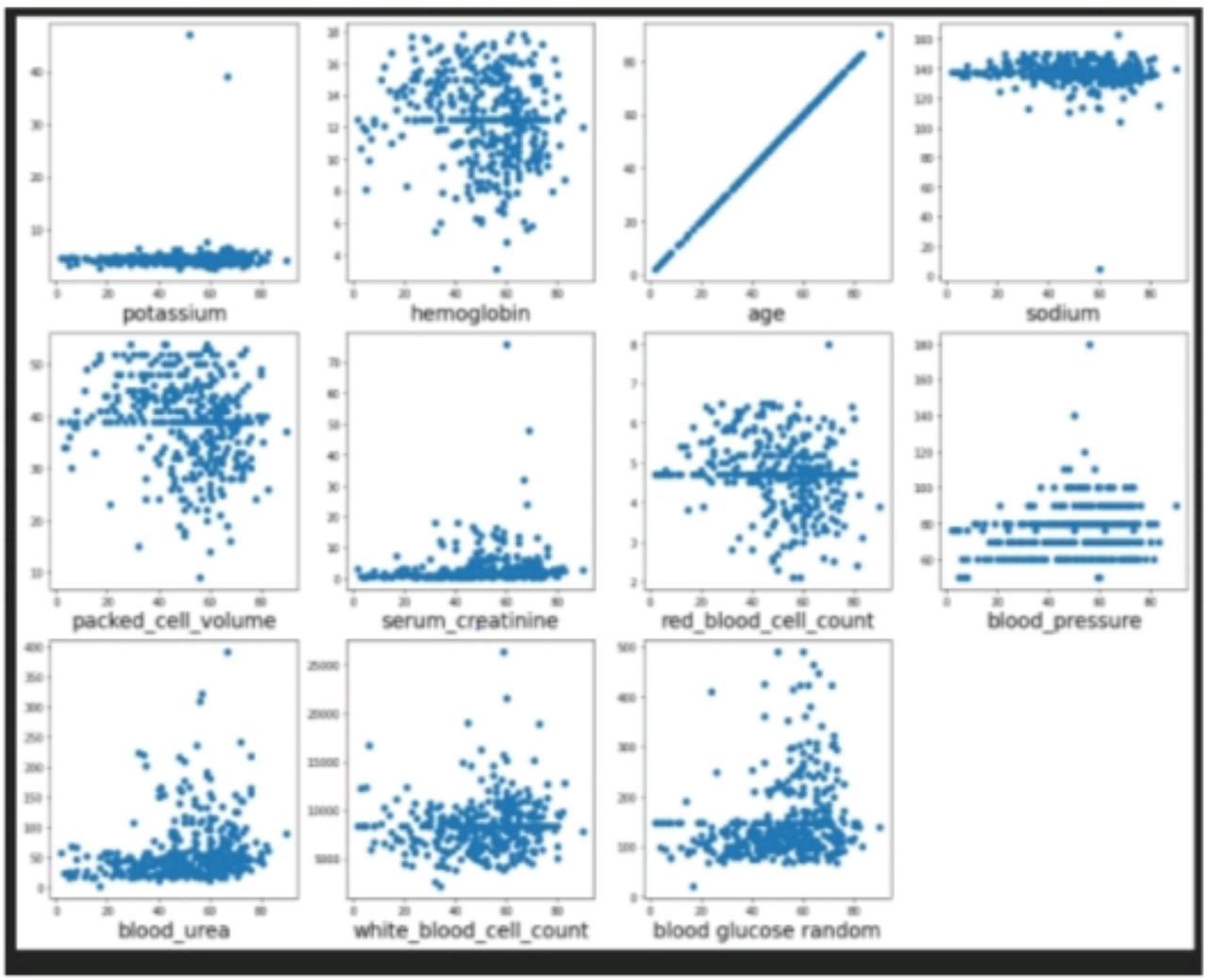
Activity 2.3: Multivariate analysis

Age vs all continuous columns

Age vs all continous columns ¶

```
plt.figure(figsize=(20,15), facecolor='white')
plotnumber = 1

for column in contcols:
    if plotnumber<=11 :     # as there are 11 continous columns in the data
        ax = plt.subplot(3,4,plotnumber) # 3,4 is refer to 3X4 matrix
        plt.scatter(data['age'],data[column]) #plotting scatter plot
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```



As you can observe with the scatter plot many of features are correlated with age.

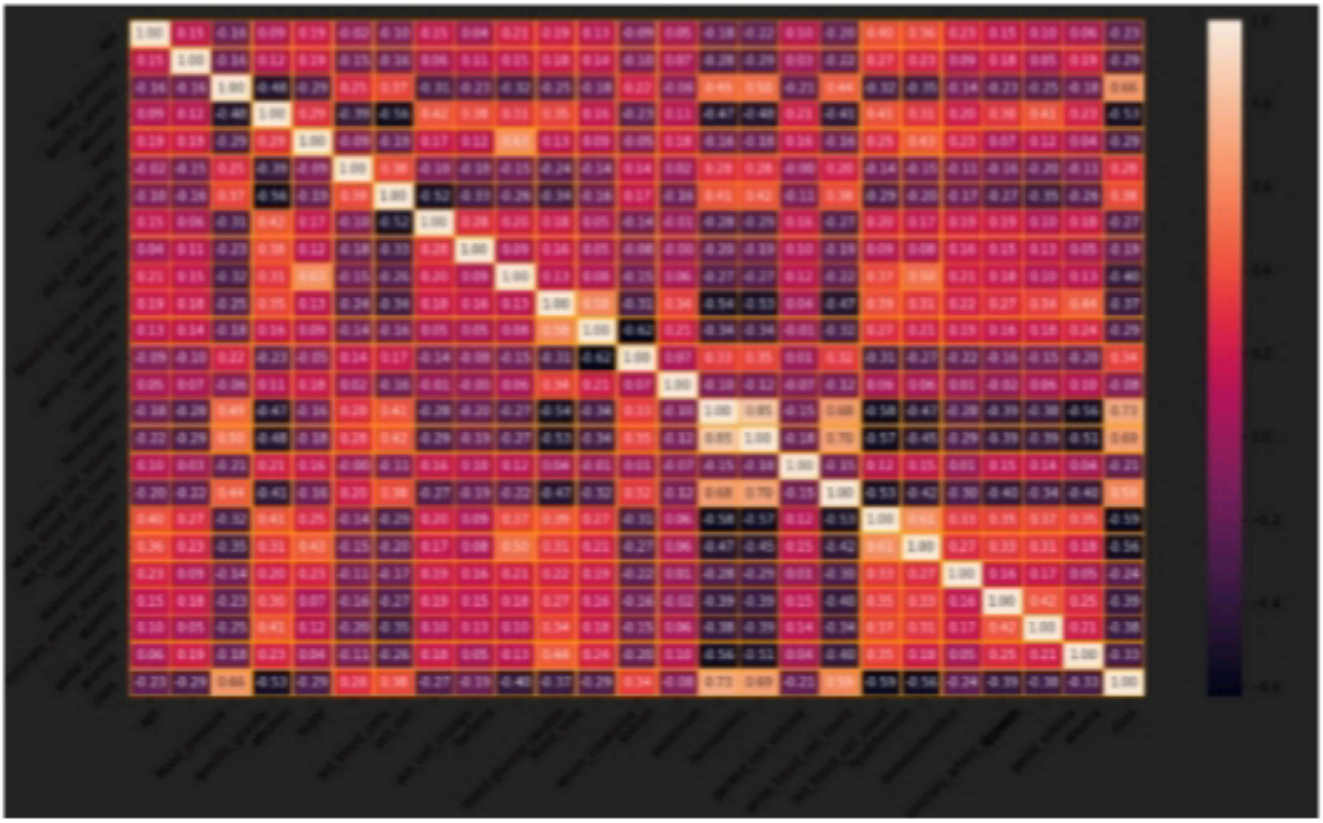
Finding correlation between the independent Columns

Correlation is a statistical relationship between two variables and it could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease.

With the help of seaborn heatmap we will be plotting the heatmap and for finding the correlation between variable we have **corr()** available.

Finding correlation between the independent Columns

```
#HEAT MAP #correlation of parameters
f,ax=plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linecolor="orange")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```



If you observe the heatmap, lighter the colour the correlation between that two variables will be high.

And correlation plays a very important role for extracting the correct features for build our model.

Now, let's observe the count of our target data classes, by using seaborn countplot

```
sns.countplot(data['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c1d390d30>
```

Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance-based method and Gradient Descent concept.

```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset because
# there different type of values in the columns
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

Separate independent and dependent variable

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

Creating Independent and Dependent

```
selcols=['red_blood_cells','pus_cell', 'blood glucose random','blood_urea',
            'pedal_edema', 'anemia','diabetesmellitus','coronary_artery_disease']
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['class'])
print(x.shape)
print(y.shape)
```

```
(400, 8)
(400, 1)
```

In the above code we are creating DataFrame of the independent variable **x** with our selected columns and for dependent variable **y** we are only taking the **class** column.

Where **DataFrame is used to** represents a table of data with rows and columns.

Splitting data into train and test

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

Splitting the data into train and test

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)#train test split
```