Data Science

Univariate Analysis

Observation and Explanation

1. Finding if there is null values (empty cells) in the dataset

```
dataset.isnull().sum()
```

```
sl_no             0
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
salary           67
dtype: int64
```

- .isnull() function of pandas shows Nan(null values) of the dataset
- Here the salary column has 67 null values

2. Replacing the Nan with zero or mean/median/mode respective to the dataset and removing data.

```
1]: dataset.fillna(0,inplace=True)
```

```
2]: dataset
```

2]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	0.0

- .fillna() function helps replacing the NaN cells with 0, it can also be replaced with mean/median/mode.

Python: dataset(["column name here"].fillna(["column name here"].mean(),inplace=true)

- But this function can replace values at only one column at a time

- .drop(0,inplace=True), this function deletes row or column with null values where 0 refers to row and 1 refers to column

Measure of Central Tendency:

```
[21]: dataset.describe()
```

[21]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
count	215.000000	215.000000	215.000000	215.000000	215.000000	215.000000	148.000000
mean	108.000000	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
std	62.209324	10.827205	10.897509	7.358743	13.275956	5.833385	93457.452420
min	1.000000	40.890000	37.000000	50.000000	50.000000	51.210000	200000.000000
25%	54.500000	60.600000	60.900000	61.000000	60.000000	57.945000	240000.000000
50%	108.000000	67.000000	65.000000	66.000000	71.000000	62.000000	265000.000000
75%	161.500000	75.700000	73.000000	72.000000	83.500000	66.255000	300000.000000
max	215.000000	89.400000	97.700000	91.000000	98.000000	77.890000	940000.000000

- .describe() a function under pandas that gives mean (MCT) , count, Standard Deviation(MOS),percentile(Q1,Q2,Q3,Q4)(MLD), IQR, Outliers, min and max value of the data
- Here the average (mean) “of ssc_p” is 67.3
- But we need mean, median,mode of the dataset , so we create a dataframe using pandas and for loop to calculate these values from the dataset and bring it under the dataframe

```
def dest(dataset):
    import numpy as np
    descriptive=pd.DataFrame(index=["Mean","Median","Mode","Q1:25%","Q2:50%","Q3:75%","99%","Q4:100%","Min","Max","IQR","1.5_Rule","Less_Outlier_Ra
    for columnName in quan:
        descriptive[columnName]["Mean"]=dataset[columnName].mean()
        descriptive[columnName]["Median"]=dataset[columnName].median()
        descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
        descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
        descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
        descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
        descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
        descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
        descriptive[columnName]["Min"]=dataset[columnName].min()
        descriptive[columnName]["Max"]=dataset[columnName].max()
        descriptive[columnName]["IQR"]=descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
        descriptive[columnName]["1.5_Rule"]=1.5*descriptive[columnName]["IQR"]
        descriptive[columnName]["Less_Outlier_Range"]=descriptive[columnName]["Q1:25%"]-descriptive[columnName]["1.5_Rule"]
        descriptive[columnName]["Great_Outlier_range"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5_Rule"]
        descriptive[columnName]["Variance"]=dataset[columnName].var()
        descriptive[columnName]["Standard_Deviation"]=dataset[columnName].std()
        return descriptive
```

- ❖ Formulas:
  IQR=Q3-Q1
  1.5 Rule= 1.5*IQR
  Less Outlier Range=Q1-1.5 Rule
  Great Outlier Range=Q3+1.5 Rule

Out Lier Detection

```
def outliers(quan,dataset):
    lesser=[]
    greater=[]
    for columnName in quan:
        if(descriptive[columnName]["Min"]<descriptive[columnName]["Less_Outlier_Range"]):
            lesser.append(columnName)
        if(descriptive[columnName]["Max"]>descriptive[columnName]["Great_Outlier_range"]):
            greater.append(columnName)
            return lesser,greater
```

Less outlier exist if:

- Minimun value of a column is less than the lesser outlier range
- Means that a certain data point is far away from the mean in leftside

Greater outlier exist if:

- Maximun value of a column id greater than greater outlier range
- Means that a certain data point is far away from the mean in rightside.

Outlier Replacement

```
[151]: def outlier_replace(dataset):
           replace=dataset
           for columnName in dataset[lesser]:
               replace[columnName][dataset[columnName].min()<descriptive[columnName]["Less_Outlier_Range"]]=
           for columnName in dataset[greater]:
               replace[columnName][dataset[columnName].max()>descriptive[columnName]["Great_Outlier_range"]]
               return replace
```

- Replacing outlier with less and greater outlier range calculated from IQR and 1.5 rule

Measure of Location of Data

```
def freqTable(columnName):
    freqTable=pd.DataFrame(columns=["Unique_Values","Frequency","Relative_Frequency","CumSum"])
    freqTable["Unique_Values"]=replace[columnName].value_counts().index
    freqTable["Frequency"]=replace[columnName].value_counts().values
    length=len(freqTable["Frequency"])
    freqTable["Relative_Frequency"]=freqTable["Frequency"]/length
    freqTable["CumSum"]=replace["ssc_p"].cumsum()
    print(f"The Length of the column {columnName} is",length)
    return freqTable
```

- Python to get frequency, Relative frequency, cumulative frequency
- .value_counts gives the repeated values (frequency) and no of repetations.
- .cumsum gives cumulative density.

```
[164]: freqTable("hsc_p")
```

```
The Length of the column hsc_p is 91
```

[164]:

	Unique_Values	Frequency	Relative_Frequency	CumSum
0	63.00	14	0.153846	0.153846
1	62.00	12	0.131868	0.285714
2	60.00	9	0.098901	0.384615
3	67.00	9	0.098901	0.483516
4	70.00	8	0.087912	0.571429
...	...	...	...	...
86	70.29	1	0.010989	2.318681
87	83.83	1	0.010989	2.329670
88	70.40	1	0.010989	2.340659
89	90.90	1	0.010989	2.351648
90	61.33	1	0.010989	2.362637

91 rows × 4 columns

- Frequency table for column “hsc_p”
- Here 63 repeated 14 times and it’s the highest repeated value.

- Relative Frequency: how often a value has occurred 14/91=0.153845, where 14 occurrence of 63 and 91 total no of values
- Cum Sum : sum of values under the range

Normal Distribution

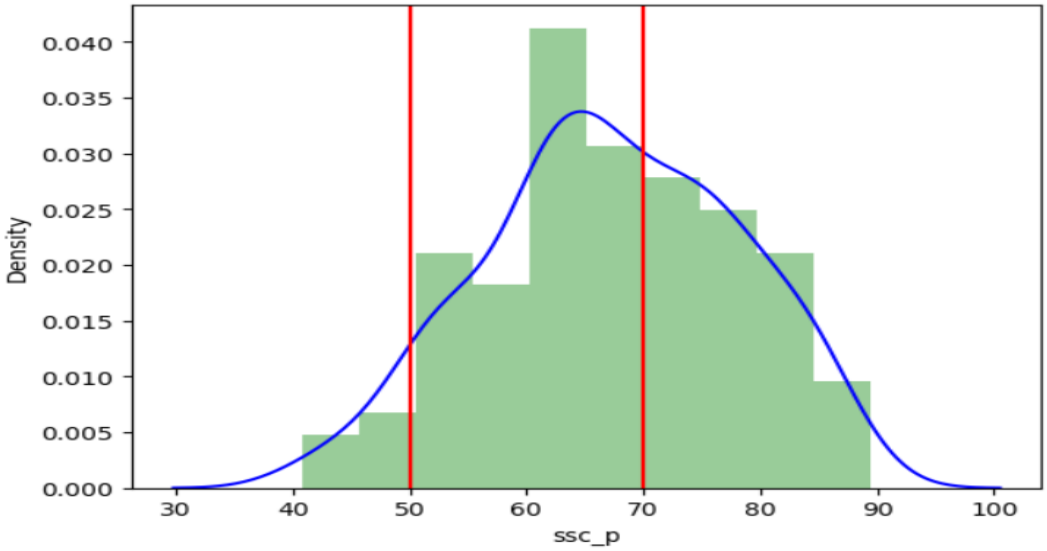
[19]:
```
def pdf_probability(dataset,startrange,endrange):
    import seaborn as sns
    import matplotlib.pyplot as plt
    from scipy.stats import norm
    sns.distplot(dataset,kde=True,kde_kws={'color':'blue'},color='Green')
    plt.axvline(startrange,color='Red')
    plt.axvline(endrange,color='Red')

    sample=dataset
    sample_mean=sample.mean()
    sample_std=sample.std()

    dist=norm(sample_mean,sample_std)
    values=[value for value in range(startrange,endrange)]
    probability=[dist.pdf(value) for value in values]
    prob=sum(probability)
    print('Mean=%.3f,Standard_Deviation=%.3f'%(sample_mean,sample_std))
    #print(f"The values between the range ({startrange},{endrange}) is",values)
    #print("The probability of the values are", probability)
    print(f"The area between the range ({startrange},{endrange}):{prob}")
    return prob
```

- Seaborn to graph histogram with curve using KDE (Kernel Density Estimation) with given blue color to the curve and green color to the histogram(plotting of frequency).
- Matplotlib.pyplot to bring start and end range line with given red color.
- Since Normal Distribution is characterised by mean and standard deviation, so calculated using .mean() and .std() function
- Then probability of values between the range and cumsum is calculated and brought inside the list using one line for loop.

[20]: 0.5304184324400784




```
Mean=67.303,Standard_Deviation=10.827
The area between the range (50,70):0.5304184324400784
```

Empirical Rule

[25]:
```
from statsmodels.distributions.empirical_distribution import ECDF
ecdf=ECDF(dataset["hsc_p"])
```

[26]:
```
ecdf(80)
```

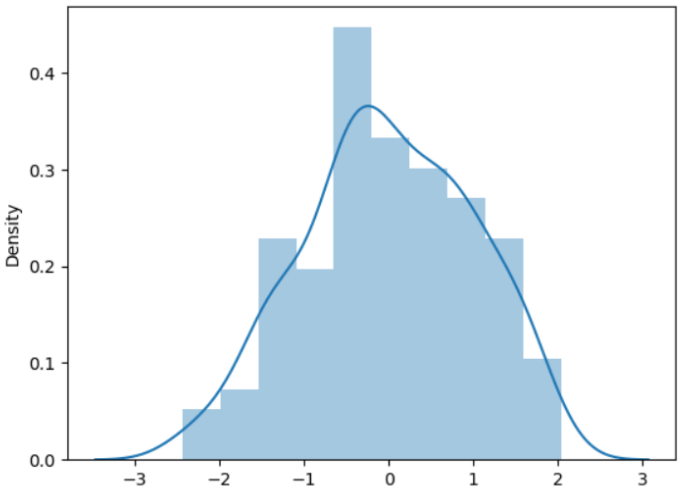
[26]: 0.9162790697674418

- Percentage of occurrence data at a range

Standard Normal Distribution

[27]:
```
def stdNBgraph(dataset):
    import seaborn as sns
    mean=dataset.mean()
    std=dataset.std()

    values=[value for value in dataset]
    z_score=[(j-mean)/std for j in values]
    sns.distplot(z_score,kde=True)
    sum(z_score)/len(z_score)
```



- stdNB is measured using Z-Score and helpful to compare two different type of data (eg. Height in cm and feet).
- Z=(X-mean)/Standard Deviation.
- And plots the x-axis using standard deviation and the y-axis has the density.