*Exploratory Data Analysis (EDA) Using Python Libraries *

Dataset contains 8 columns namely – First Name, Gender, Start Date, Last Login, Salary, Bonus%, Senior Management, and Team.

```
import pandas as pd
df=pd.read_csv('/content/employees.csv')
df.head()
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

```
df.shape               #  shape of the data
```

```
(1000, 8)
```

The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc.

Note we can also get the description of categorical columns of the dataset if we specify

include ='all' in the describe function.

```
df.describe()
```

	Salary	Bonus %
count	1000.000000	1000.000000
mean	90662.181000	10.207555
std	32923.693342	5.528481
min	35013.000000	1.015000
25%	62613.000000	5.401750
50%	90428.000000	9.838500
75%	118740.250000	14.838000
max	149908.000000	19.944000

```
#the columns and their data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   First Name         933 non-null    object 
 1   Gender             855 non-null    object 
 2   Start Date         1000 non-null   object 
 3   Last Login Time    1000 non-null   object 
 4   Salary             1000 non-null   int64  
 5   Bonus %            1000 non-null   float64
 6   Senior Management  933 non-null    object 
 7   Team               957 non-null    object 
dtypes: float64(1), int64(1), object(6)
memory usage: 62.6+ KB
```

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

```
First Name           67
Gender              145
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
```

```
Team                43
dtype: int64
```

```
# Changing Dtype from Object to Datetime
# convert "Start Date" column to datetime data type
df['Start Date'] = pd.to_datetime(df['Start Date'])
df.head()
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	4:47 PM	101004	1.389	True	Client Services

```
#the number of unique elements
df.nunique()
```

```
First Name          200
Gender                2
Start Date          972
Last Login Time     720
Salary              995
Bonus %             971
Senior Management     2
Team                 10
dtype: int64
```

Handling Missing Values

```
df.isnull().sum()          #check if there are any missing values in our dataset or not
```

```
First Name           67
Gender              145
Start Date            0
```

```
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management     67
Team                  43
dtype: int64
```

```
#missing values of gender with the string “No Gender”
df["Gender"].fillna("No Gender", inplace = True)
df.isnull().sum()
```

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

```
##fill the senior management with the mode value.

import numpy as np
mode = df['Senior Management'].mode().values[0]
df['Senior Management']= df['Senior Management'].replace(np.nan, mode)
df.isnull().sum()
```

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management     0
Team                 43
dtype: int64
```

```
# for the first name and team, we cannot fill the missing values with arbitrary data, # so, let’s drop all the rows containing these missing values.
df = df.dropna(axis = 0, how ='any')
print(df.isnull().sum())
df.shape
```

```
First Name           0
Gender               0
Start Date           0
Last Login Time      0
Salary               0
Bonus %              0
Senior Management    0
Team                 0
dtype: int64
(899, 8)
```

Data Encoding

methods for encoding:Label encoding or One-hot encoding

```
from sklearn.preprocessing import LabelEncoder
# create an instance of LabelEncoder
le = LabelEncoder()

# fit and transform the "Senior Management" column with LabelEncoder
df['Gender'] = le.fit_transform(df['Gender'])
df.head()
```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	1	1993-08-06	12:42 PM	97308	6.945	True	Marketing
2	Maria	0	1993-04-23	11:17 AM	130590	11.858	False	Finance
3	Jerry	1	2005-03-04	1:00 PM	138705	9.340	True	Finance
4	Larry	1	1998-01-24	4:47 PM	101004	1.389	True	Client Services
5	Dennis	1	1987-04-18	1:35 AM	115163	10.125	False	Legal

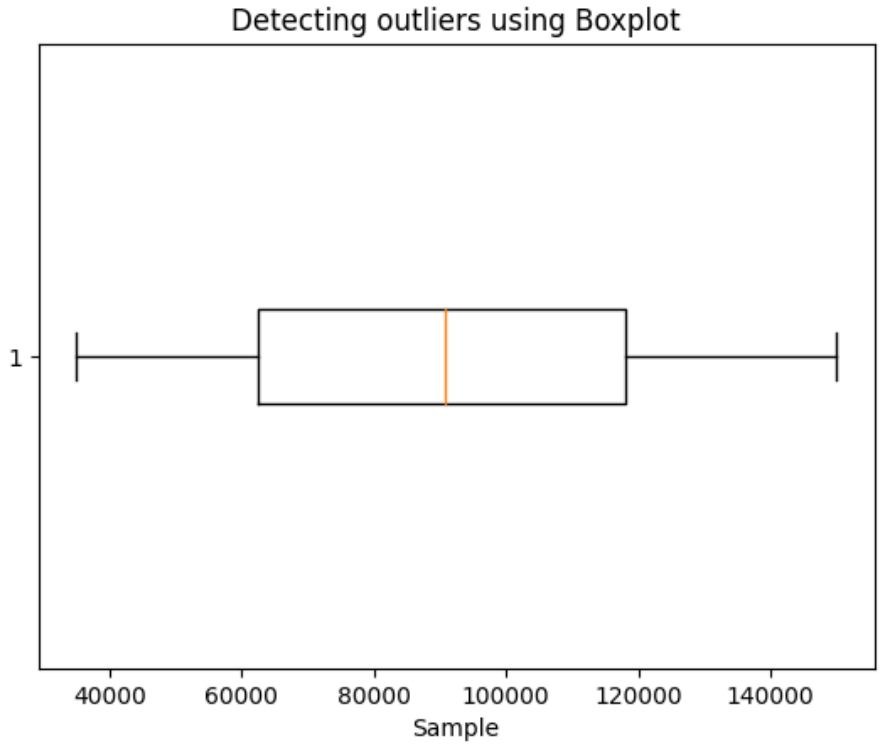
Data visualization

```
# Histogram
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(x='Salary', data=df, )
plt.show()
```

```
# Boxplot

sns.boxplot( x="Salary", y='Team', data=df)
plt.show()
```

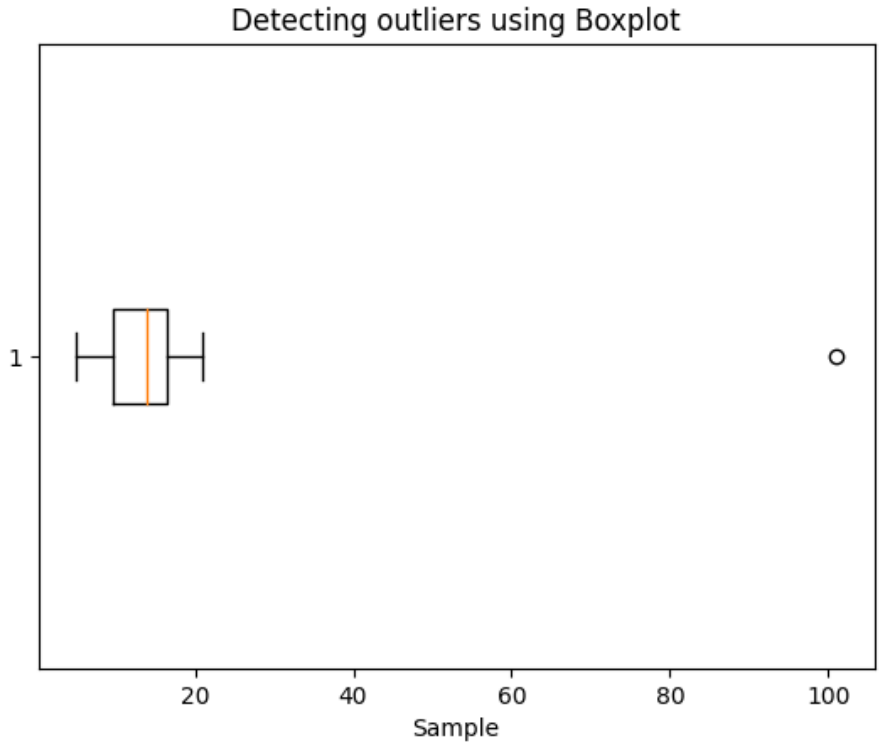
```
#pairplot
sns.pairplot(df, hue='Gender', height=2)
```

```
import matplotlib.pyplot as plt

#sample= [15, 101, 18, 7, 13, 16, 11, 21, 5, 15, 10, 9]
plt.boxplot(df['Salary'], vert=False)
plt.title("Detecting outliers using Boxplot")
plt.xlabel('Sample')
plt.show()
```



Handling Outliers

```
import matplotlib.pyplot as plt

sample= [15, 101, 18, 7, 13, 16, 11, 21, 5, 15, 10, 9]
plt.boxplot(sample, vert=False)
plt.title("Detecting outliers using Boxplot")
plt.xlabel('Sample')
plt.show()
```



Detecting Outliers using the Z-scores

```
import numpy as np
outliers = []
def detect_outliers_zscore(data):
    thres = 3
    mean = np.mean(data)
    std = np.std(data)
    # print(mean, std)
    for i in data:
        z_score = (i-mean)/std
        print(i,"\t",z_score)
        if (np.abs(z_score) > thres):
            outliers.append(i)
    return outliers# Driver code
sample_outliers = detect_outliers_zscore(sample)
print("\n Outliers from Z-scores method: ", sample_outliers)
```

```
15       -0.20502261723677698
101      3.2635567432280412
18       -0.08402566280195775
7        -0.5276811623962949
13       -0.28568725352665647
16       -0.16469029909183724
11       -0.366351889816536
21       0.03697129163286148
5        -0.6083457986861744
15       -0.20502261723677698
10       -0.4066842079614757
9        -0.44701652610641546

 Outliers from Z-scores method:  [101]
```

Detecting Outliers using the Inter Quantile Range(IQR)

```
outliers = []
def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    print("1st quratile=",q1,"\t","2nd quartile=" ,q3)
    IQR = q3-q1
    print("IQR=",IQR)
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    print("Lower bound=",lwr_bound, "\t Upper bound=",upr_bound)
    for i in data:
        if (i<lwr_bound or i>upr_bound):
            outliers.append(i)
    return outliers# Driver code
sample_outliers = detect_outliers_iqr(sample)
print("Outliers from IQR method: ", sample_outliers)
```

```
 1st quratile= 9.75       2nd quartile= 16.5
 IQR= 6.75
 Lower bound= -0.375      Upper bound= 26.625
 Outliers from IQR method:  [101]
```

How to Handle Outliers?

Step 1: Trimming/Remove the outliers

```
#outliers_removed = [x for x in data if x >= lower and x <= upper]

# Trimming
print(sample)
for i in sample_outliers:
  a=[x for x in sample if x==i]
  sample.remove(a)
  #a = np.delete(sample, np.where(i==sample),axis=None)
print(a)
       # print(len(sample), len(a))
```

2. Quantile Based Flooring and Capping

```
# Computing 10th, 90th percentiles and replacing the outliers
tenth_percentile = np.percentile(sample, 10)
ninetieth_percentile = np.percentile(sample, 90)
print(tenth_percentile, ninetieth_percentile)
b = np.where(sample<tenth_percentile, tenth_percentile, sample)
b = np.where(b>ninetieth_percentile, ninetieth_percentile, b)
# print("Sample:", sample)
print("New array:",b)
```

3.Mean/Median Imputation

```
    median = np.median(sample)
```