# Sem1 Mini Project
Submitted By Om Nimmalwar

## Importing Required Packages

```
#Importing Required Packages
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

## Importing Data

```
#Importinh Data Set
df = pd.read_csv("All Data Sets/employee_survey.csv")
df.head()
```

## Getting Information About Data
Now we check the null values in data set if any present

```
#Checking Number Of row and column in dataframe
df.shape
```

```
df.info()
```

## Checking For Null Values And Clean Them

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

As per Above Information There Is No Null Values In DataFrame So We Move Further Toward Our Next Step That Is finding Outliers In Important Columns by using boxplot if any outlier is present then we will replace the but before that we will check the statastical info about all numeric olumns using describe function

## Finding And Replacing The Outliers

```
df.describe()
```

```
plt.figure(figsize=(12,10))
# Outlier for Age Variable
plt.subplot(3, 3, 1)
sns.boxplot(y=df['Age'])
plt.title("Outlier in Age")

# Outlier For Sleephours Variable
plt.subplot(3, 3, 2)
sns.boxplot(y=df['SleepHours'])
plt.title("Outlier in SleepHours")

# Outlier For Experience Variable
plt.subplot(3, 3, 3)
sns.boxplot(y=df['Experience'])
plt.title("Outlier in Experience")

# Outlier For PhysicalActivityHours Variable
plt.subplot(3, 3, 4)
sns.boxplot(y=df['PhysicalActivityHours'])
plt.title("Outlier in PhysicalActivityHours")

# Outlier For CommuteDistance Variable
plt.subplot(3, 3, 5)
sns.boxplot(y=df['CommuteDistance'])
plt.title("Outlier in CommuteDistance")
```

```
# Outlier For TrainingHoursPerYear Variable
plt.subplot(3, 3, 6)
sns.boxplot(y=df['TrainingHoursPerYear'])
plt.title("Outlier in TrainingHoursPerYear")

# Outlier For NumCompanies Variable
plt.subplot(3, 3, 7)
sns.boxplot(y=df['NumCompanies'])
plt.title("Outlier in NumCompanies")

# Outlier For NumReports Variable
plt.subplot(3, 3, 8)
sns.boxplot(y=df['NumReports'])
plt.title("Outlier in NumReports")

plt.tight_layout()
plt.show()
```

As we seen in above outlier box we find that **PhysicalActivityhours** And **SleepHours** Coloum Has Outliers Then Now We Will **Chage** Or **Modify** Them by changing the value to top or lowe values if Outlier Is below the lower limit then chage or replace it values with Lower limit values and vice versa but for that we need all **statastical values of PhysicalActivityHours And SleepHours** For that we will use **describe()** function on this columns

```
df[['PhysicalActivityHours','SleepHours']].describe()
```

As We get value of 25% and 75% which are First Qurtile And Third Qurtile Simulteniously now we can find InterQurtileRange Using Below Formula
InterQurtileRange(IQR) = 3rd Qurtile - 1st Qurtile
And To Found Higher And Lower Limit Of Cell We Use Below Formulas
Lower_Limit = Q1 - 1.5 * IQR
Upper_Limit = Q3 + 1.5 * IQR

```
#As per Above Formula
IQR_Of_SleepHours = 7.7 - 6.3 # 25% = 6.3 and 75% = 7.7
IQR_Of_PhysicalActivityHours = 2.70 - 1.3 # 25% = 1.3 and 75% = 2.7

sleepHoursOutliers = []
lower_SH = 6.3 - 1.5 * IQR_Of_SleepHours #Lower Limit Of Column Below This If Any
Item Found It Will Be Outlier
upper_SH = 7.7 + 1.5 * IQR_Of_SleepHours #Higher Limit Of Column Above This If any
Item Found That Will Be Outlier

sleepHoursOutliers = df[(df['SleepHours'] < lower_SH) | (df['SleepHours'] >
upper_SH)]['SleepHours'].tolist()
sleepHoursOutliers.sort()
print("***************** Outliers in Sleephours : ******************")
print(sleepHoursOutliers)

outliers_p = []
lower_PAH = 1.3 - 1.5 * IQR_Of_PhysicalActivityHours
upper_PAH = 2.7 + 1.5 * IQR_Of_PhysicalActivityHours

outliers_p = df[(df['PhysicalActivityHours'] < lower_PAH) |
(df['PhysicalActivityHours'] > upper_PAH)]['PhysicalActivityHours'].tolist()
outliers_p.sort()
print("***************** Outliers in PhysicalActivityHours : ******************")
print(outliers_p)

#Now We Got All The Outliers In Both Cloumns Now We Will Replace Them By Lower And
Upper Limit Of respective Columns
#For SleepHours
for i in range(len(df)):
if df.loc[i, 'SleepHours'] < lower_SH:
df.loc[i, 'SleepHours'] = lower_SH
elif df.loc[i, 'SleepHours'] > upper_SH:
df.loc[i, 'SleepHours'] = upper_SH
for j in range(len(df)):
if df.loc[j, 'PhysicalActivityHours'] < lower_PAH:
```

```
df.loc[j, 'PhysicalActivityHours'] = lower_PAH
elif df.loc[j, 'PhysicalActivityHours'] > upper_PAH:
df.loc[j, 'PhysicalActivityHours'] = upper_PAH
plt.subplot(1,2,1)
sns.boxplot(y='SleepHours',data=df)
plt.title("Outliers In Sleep Hours")
plt.subplot(1,2,2)
sns.boxplot(y='PhysicalActivityHours',data = df)
plt.title("Outliers In PhysicalActivityHours")
```

So we have removed all outliers from the table now we will check is their any spelling mistakes or any duplicate values in important colums

# Finding And Cleaning The Spell Mistakes In Nominal Data Columns
# Important Columns Are
* Gender
* Department
* MaritalStatus
* JobLevel
* EmpType
* CommuteMode
* EduLevel
* haveOT

```
df[['Gender','Dept','MaritalStatus','JobLevel','EmpType','CommuteMode','EduLevel','h
aveOT']].nunique()
```

```
Impoetant_Columns = ['Gender','Dept','MaritalStatus','JobLevel','EmpType','CommuteMo
de','EduLevel','haveOT']
for i in Impoetant_Columns:
print(f"Unique Values In {i} : {df[i].unique()}")
```

**We got value error in gender in which we have three labels 1.male , 2.female , 3.other so we dont know what exactly the others are so we will check if all others
maritalstatus is 'Single' . if all are single then it may possible other show Third gender in that case we will not change any values . if not then we will drop all the values**

```
grouped = df.groupby('Gender')
df_A = grouped.get_group('Other')
df_A['MaritalStatus'].value_counts()
```

So as we seen all others are labbaled as Single so no need to change anything in Gendeers Column

**Our data is now cleaned now we will move to next step that is comparitive analysis in which we will compare all catogries like
gender , dept ,etc.**

```
# Saving Cleaned Data Set
df.to_csv("Cleaned DataSet.csv")
```

# Statastical Analysis

```
aask = df.select_dtypes(include='number').corr()
print(aask)
```

```
plt.figure(figsize=(12,7))
sns.heatmap(aask)
```

* This Shows That The Workload and Stress Has Negative Correletion With JobSatifaction
* Age Has Positive Reletion With Experience
* Experiance Also Has Positive Reletion With NumCompanies

```
# Department Wise Avg Age
Age_group = df.groupby(['Dept'])['Age'].mean().round()
print(Age_group)

Experiance_Group = df.groupby(['Dept'])['Experience'].mean().round()
print(Experiance_Group)

WLB_Group = df.groupby(['Dept'])['WLB'].mean()
print(Experiance_Group)

Stress_Group = df.groupby(['Dept'])['Stress'].mean()
print(Stress_Group)

JobSat_Group = df.groupby(['Dept'])['JobSatisfaction'].mean()
print(JobSat_Group)

df['EduLevel'].value_counts(normalize=True) * 100

df['Gender'].value_counts(normalize=True) * 100

df.groupby('MaritalStatus')['Stress'].mean()
```

# Comparitive Analysis

```
plt.figure(figsize=(12,7))
ax = sns.countplot(x="Dept", hue="Gender", data=df)
for container in ax.containers:
ax.bar_label(container)

plt.title("Department-wise Gender Distribution")
plt.xlabel("Department")
plt.ylabel("Count")
plt.show()

joblevel_gender_dept = df.groupby(['JobLevel', 'Gender',
'Dept']).size().reset_index(name='Count')
joblevel_gender_dept

plt.figure(figsize=(14,7))

sns.barplot(
data=joblevel_gender_dept,
x="JobLevel",
y="Count",
hue="Gender"
)

plt.title("Job Level vs Gender (Count)")
plt.xlabel("Job Level")
plt.ylabel("Employee Count")
plt.show()

pivot_dept = joblevel_gender_dept.pivot_table(
index='JobLevel',
columns='Dept',
values='Count',
fill_value=0
)

pivot_dept.plot(kind='bar', figsize=(12,7))

plt.xlabel("Job Level")
plt.ylabel("Employee Count")
plt.title("Job Level vs Department Count")
plt.xticks(rotation=0)

for container in plt.gca().containers:
plt.bar_label(container, fmt='%.0f')

plt.show()


Edulevel_gender_dept = df.groupby(['EduLevel', 'Gender',
'Dept']).size().reset_index(name='Count')
Edulevel_gender_dept


plt.figure(figsize=(14,7))

sns.barplot(
data=Edulevel_gender_dept,
x="EduLevel",
y="Count",
hue="Gender"
)

plt.title("Education Level vs Gender (Count)")
plt.xlabel("Education Level")
plt.ylabel("Employee Count")
plt.show()


pivot_dept = Edulevel_gender_dept.pivot_table(
index='EduLevel',
columns='Dept',
values='Count',
fill_value=0
)

pivot_dept.plot(kind='bar', figsize=(12,7))

plt.xlabel("Education Level")
plt.ylabel("Employee Count")
plt.title("Education Level vs Department Count")
plt.xticks(rotation=0)

for container in plt.gca().containers:
plt.bar_label(container, fmt='%.0f')

plt.show()


data = df.groupby('Gender')['JobSatisfaction'].mean()

plt.figure()
data.plot(kind='bar')
plt.xlabel("Gender")
plt.ylabel("Avg Job Satisfaction")
plt.title("Gender vs Job Satisfaction")
plt.show()


dept_vs_js = df.groupby('Dept')['JobSatisfaction'].mean()

plt.figure()
dept_vs_js.plot(kind='bar')
plt.xlabel("Dept")
plt.ylabel("Avg Job Satisfaction")
plt.title("Dept vs Job Satisfaction")
plt.xticks(rotation = 45)
plt.show()



df.groupby('JobLevel')['Workload'].mean()


df.groupby('WLB')['Stress'].mean()


ComuteDist_vs_Stress = df.groupby('CommuteDistance')['Stress'].mean().reset_index()
sns.lineplot(x='CommuteDistance',y='Stress',data=ComuteDist_vs_Stress)


df.groupby('haveOT')['Stress'].mean()
```

```
Team_vs_Workload = df.groupby('TeamSize')['Workload'].mean().reset_index()
sns.lineplot(x='TeamSize',y='Workload',data=Team_vs_Workload)
```

# Advance Analysis

```
plt.figure(figsize=(8,6))
sns.lineplot(x="CommuteDistance", y="SleepHours", data=df)
plt.title("Commute Distance vs Sleep Hours")
plt.show()



pivot = df.pivot_table(
index="JobLevel",
columns="Dept",
values="Stress",
aggfunc="mean"
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot, annot=True, cmap="YlOrRd")
plt.title("Average Stress by Job Level & Department")
plt.show()

pivot_exp = df.pivot_table(
index="JobLevel",
columns="Dept",
values="Experience",
aggfunc="mean"
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_exp, annot=True, cmap="Blues")
plt.title("Average Experience by Job Level & Department")
plt.show()

pivot_workload = df.pivot_table(
index="JobLevel",
columns="Dept",
values="Workload",
aggfunc="mean"
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_workload, annot=True, cmap="Oranges")
plt.title("Average Workload by Job Level & Department")
plt.show()

pivot_wlb = df.pivot_table(
index="JobLevel",
columns="Dept",
values="WLB",
aggfunc="mean"
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_wlb, annot=True, cmap="Greens")
plt.title("Average WLB by Job Level & Department")
plt.show()

pivot_ot = df.pivot_table(
index="JobLevel",
columns="Dept",
values="haveOT",
aggfunc="mean"
) * 100 # convert to percentage

plt.figure(figsize=(10,6))
sns.heatmap(pivot_ot, annot=True, fmt=".1f", cmap="Reds")
plt.title("Overtime % by Job Level & Department")
plt.show()
```

```
pivot_marital_workload = df.pivot_table(
index="MaritalStatus",
columns="Dept",
values="Workload",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_marital_workload, annot=True, cmap="Blues")
plt.title("Average Workload by Marital Status & Department")
plt.xlabel("Department")
plt.ylabel("Marital Status")
plt.show()

pivot_marital_stress = df.pivot_table(
index="MaritalStatus",
columns="Dept",
values="Stress",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_marital_stress, annot=True, cmap="YlOrRd")
plt.title("Average Stress by Marital Status & Department")
plt.xlabel("Department")
plt.ylabel("Marital Status")
plt.show()

pivot_wlb = df.pivot_table(
index="MaritalStatus",
columns="Dept",
values="WLB",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_wlb, annot=True, cmap="Greens")
plt.title("Average Work-Life Balance by Marital Status & Department")
plt.xlabel("Department")
plt.ylabel("Marital Status")
plt.show()


pivot_wlb_gender = df.pivot_table(
index="Gender",
columns="Dept",
values="WLB",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_wlb_gender, annot=True, cmap="Greens")
plt.title("Gender-wise Average WLB by Department")
plt.xlabel("Department")
plt.ylabel("Gender")
plt.show()

pivot_workload_gender = df.pivot_table(
index="Gender",
columns="Dept",
values="Workload",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_workload_gender, annot=True, cmap="Oranges")
plt.title("Gender-wise Average Workload by Department")
plt.xlabel("Department")
plt.ylabel("Gender")
plt.show()
```

```
metrics = df.groupby(["Gender", "Dept"])[["WLB", "Workload",
"Stress"]].mean().reset_index()

plt.figure(figsize=(14,6))
sns.barplot(data=metrics, x="Dept", y="WLB", hue="Gender")
plt.title("Gender-wise WLB Across Departments")
plt.show()

plt.figure(figsize=(14,6))
sns.barplot(data=metrics, x="Dept", y="Workload", hue="Gender")
plt.title("Gender-wise Workload Across Departments")
plt.show()

plt.figure(figsize=(14,6))
sns.barplot(data=metrics, x="Dept", y="Stress", hue="Gender")
plt.title("Gender-wise Stress Across Departments")
plt.show()


pivot_stress_gender = df.pivot_table(
index="Gender",
columns="Dept",
values="Stress",
aggfunc="mean",
fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot_stress_gender, annot=True, cmap="Reds")
plt.title("Gender-wise Average Stress by Department")
plt.xlabel("Department")
plt.ylabel("Gender")
plt.show()
```