NEW Install the Jupyter extension to Notebook 7 to learn about the new features and the actions to take (you are using extension). - Please note that updating to Notebook 7 might break some of your extensions.

Formative Assessment (Data Preprocessing)

Data Processing before building machine learning Model

Steps of Preprocessing of data

Step 1: Import necessary libraries Step 2: Read Dataset Step 3: Sanity Check of Data Step 4: Exploratory Data Analysis [EDA] Step 5: Missing Value Treatment Step 6: Outlier Treatments Step 7: Duplicate and Garbage Value treatment Step 8: Normalization Step 9: Encoding of Data

In [ ]:

Step 1: Import Necessary Library

In [1]:
```
# standard library imports
import warnings # for setting handling

# Data\analysis imports
import numpy as np # for data manipulation and analysis,the file IO
import numpy as np # for numerical operations and mathematical functions
import pandas as pd # for data manipulation and analysis
import numpy as np # for computing graphs
from sklearn.model_selection import train_test_split # for data splitting (training & testing) in machine learning
from sklearn.preprocessing import MinMaxScaler, StandardScaler # for feature standardization & normalization
from sklearn.metrics import accuracy_score, classification_report # for model evaluation

# plot setting\handling
warnings.filterwarnings('ignore') # for ignoring warnings
```

In [ ]:

In [ ]:

Step 2: Read DataSet

In [2]:
```
df.to_csv('C:\Users\rajan\Downloads\Employee.csv')
```

In [ ]:

In [3]:
```
df.head()
```

Out[3]:

	Company	Age	Salary	Place	Country	Gender
0	TCS	20	NaN	Chennai	India	0
1	Infosys	30	54000	Bangalore	India	0
2	TCS	35	25000	Calcutta	India	1
3	Infosys	40	35000	Delhi	India	0
4	TCS	35	40000	Mumbai	India	0

In [3]:
```
df.tail()
```

Out[3]:

	Company	Age	Salary	Place	Country	Gender
1995	TCS	30	30000	Calcutta	India	1
1996	Infosys	35	49000	Calcutta	India	1
1997	Infosys	40	65000	Mumbai	India	0
1998	TCS	30	30000	Calcutta	India	1
1999	Infosys	35	60000	Cochin	India	0

Step 3: Sanity Check of Data

In [3]:
```
#shape
print("The shape of the dataset
print("The shape",*,df.shape)

# dataset dimensions and statistics
print("Number of rows: {df.shape[0]}")
num_features = df.shape[1]
print("Number of features: {num_features}")

# print the information about the dataset
print("Number of rows: {df.shape[0]}")
print("Number of columns: {df.shape[1]}")
print("Number of features: {df.shape[1]}")
print("Number of Gender: {df.shape[1]}")


The shape * (1999, 6)
Number of rows: 1999
Number of columns: 6
Number of features: 6
Number of Gender: 6
```

In [4]:
```
#dtypes
df.info()

<class> pandas.core.frame.DataFrame>
RangeIndex: 1999 entries, 0 to 1999
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Company  1998 non-null   object
 1   Age      1998 non-null   float64
 2   Salary   1998 non-null   float64
 3   Place    1998 non-null   object
 4   Country  1998 non-null   object
 5   Gender   1998 non-null   int64
dtypes: float64(2), int64(1), object(3)
memory usage: 111.4 KB
```

In [4]:
```
# descriptive statistics of numeric variables only.
df.describe(include=['number'])
```

Out[4]:

	Age	Salary	Gender
count	1998.000000	1998.000000	1998.000000
mean	30.000000	50140.000000	0.500500
std	5.000000	15000.000000	0.500125
min	20.000000	20000.000000	0.000000
25%	25.000000	35000.000000	0.000000
50%	30.000000	50000.000000	0.000000
75%	35.000000	65000.000000	1.000000
max	40.000000	80000.000000	1.000000

In [4]:
```
df.describe(include=['object'])
```

Out[4]:

	Company	Place	Country
count	1998	1998	1998
unique	4	10	1
top	TCS	Mumbai	India
freq	501	207	1998

In [3]:
```
# calculate the average income
average_salary = df['Salary'].mean()
print("The average salary: {average_salary:.2f} ")

# calculate the average income for people above 30
above_average_salary = df[df['Salary'] > average_salary].sum()
below_average_count = df[df['Salary'] < average_salary].sum()

# calculate ratio and print the results
ratio = above_average_salary / below_average_count
print("The ratio of people with income above average to below average: {ratio:2f} ")
print("Number of people with income above average: {above_average_salary}")
print("Number of people income below the average: {below_average_count}")


# plot the ratio using seaborn
import seaborn as sns
sns.barplot(x=['Above Average', 'Below Average'], y=[above_average_count, below_average_count], palette="Viridis")
plt.title('Ratio of People with Income Above Average to Below Average')
plt.show()

The average Salary: 5014.00
Number of people income above average to below average: 77:144
Number of people income above the average: 30
Number of people income below the average: 30
```

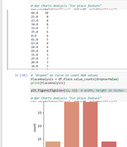
In [3]:
```
# bar chart visualization for numerical values "This enables us to see if there are outliers or not"

sns.set_theme(palette="Viridis")
```

In [4]:
```
#histogram to understand the distribution
import warnings as w
w.filterwarnings('ignore')

for i in df.select_dtypes(include=['number']):
    sns.histplot(data=df,x=i,bins=15,kde='True')
    plt.show()
```

In [4]:
```
# Exploratory Data Analysis

- Data Visualization
- Data Cleaning
- Data Preprocessing


- Data Visualization
  - Graphical Analysis
  - Univariate
  - Tri Charts
  - Bivariate
- Correlation Analysis
- Data Relationship Analysis
```

Graphical Analysis

In [5]:
```
# Company' is ratio to count and Gender is Number\ values
df.groupby('Gender').count()['Company']*100/df.shape[0]
print(genderanalysis)

# bar charts analysis "for gender feature"
sns.countplot(x='Gender', data=df, palette="Flare")
plt.show()

Gender
0   [1]
1   [1]
Name: count, dtype: int64
```



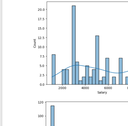
In [5]:
```
# "Company" is ratio to count and value\ values
age_analysis = df.age.value_counts(ratio=True)*100
print(age_analysis)

# create a pie chart "for Age\ratio feature"
plt.figure(figsize=(10, 12)) # figure in inches

# label by descending order
plt.pie(age_analysis[:10],
        labels=age_analysis[:10].index[:10],
        startangle=90,
        autopct='%1.1f%%',
        colors=sns.color_palette("Flare"),
        explode=[0.05,0.05,0.05,
                 ])

plt.axis('equal') # equal to set the aspect ratio of the plot to be equal.
plt.title('Age\ratio Drawn Distribution')
plt.show()

Age
40.0   10
35.0    9
30.0    9
20.0    8
21.0    8
22.0    8
23.0    8
24.0    8
25.0    8
26.0    8
27.0    8
28.0    8
29.0    8
31.0    8
32.0    8
33.0    8
34.0    8
36.0    8
37.0    8
38.0    8
39.0    8
Name: count, dtype: int64
```

In [5]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches
```

In [5]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Place feature"
sns.countplot(x='Place', data=df, palette="Flare")
plt.show()

Place
Chennai    [1]
Mumbai     [1]
Calcutta   [1]
Delhi      [1]
Bangalore  [1]
Cochin     [1]
Mysore     [1]
Hyderabad  [1]
Bhopal     [1]
Nagpur     [1]
Place
Name: count, dtype: int64
```

In [5]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Flare")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [5]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Flare")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```



In [5]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Flare")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [5]:
```
# descriptive statistics
df.describe(include=['object'])
```

Out[5]:

	Company	Place	Country
count	1998	1998	1998
unique	4	10	1
top	TCS	Mumbai	India
freq	501	207	1998

In [5]:
```
# calculate the average income
average_salary = df['Salary'].mean()
print("The average salary: {average_salary:.2f} ")

# calculate the average income for people above 30
above_average_count = df[df['Salary'] > average_salary].sum()
below_average_count = df[df['Salary'] < average_salary].sum()

# calculate ratio and print the results
ratio = above_average_count / below_average_count
print("The ratio of people with income above average to below average: {ratio:2f} ")
print("Number of people with income above average: {above_average_count}")
print("Number of people income below the average: {below_average_count}")


# plot the ratio using seaborn
import seaborn as sns
sns.barplot(x=['Above Average', 'Below Average'], y=[above_average_count, below_average_count], palette="Flare")
plt.title('Ratio of People with Income Above Average to Below Average')
plt.show()

The average Salary: 5014.00
Number of people income above average to below average: 77:144
Number of people income above the average: 30
Number of people income below the average: 30
```


In [5]:
```
# bar charts analysis "for numerical values "This enables us to see if there are outliers or not"

sns.set_theme(palette="Flare")
```

In [6]:
```
#histogram to understand the distribution
import warnings as w
w.filterwarnings('ignore')

for i in df.select_dtypes(include=['number']):
    sns.histplot(data=df,x=i,bins=15,kde='True')
    plt.show()
```


In [6]:
```
# Company' is ratio to count and Gender is Number\ values
df.groupby('Gender').count()['Company']*100/df.shape[0]
print(genderanalysis)

# bar charts analysis "for gender feature"
sns.countplot(x='Gender', data=df, palette="Flare")
plt.show()

Gender
0   [1]
1   [1]
Name: count, dtype: int64
```


In [6]:
```
# "Company" is ratio to count and value\ values
age_analysis = df.age.value_counts(ratio=True)*100
print(age_analysis)

# create a pie chart "for Age\ratio feature"
plt.figure(figsize=(10, 12)) # figure in inches

# label by descending order
plt.pie(age_analysis[:10],
        labels=age_analysis[:10].index[:10],
        startangle=90,
        autopct='%1.1f%%',
        colors=sns.color_palette("Flare"),
        explode=[0.05,0.05,0.05,
                 ])

plt.axis('equal') # equal to set the aspect ratio of the plot to be equal.
plt.title('Age\ratio Drawn Distribution')
plt.show()

Age
40.0   10
35.0    9
30.0    9
20.0    8
21.0    8
22.0    8
23.0    8
24.0    8
25.0    8
26.0    8
27.0    8
28.0    8
29.0    8
31.0    8
32.0    8
33.0    8
34.0    8
36.0    8
37.0    8
38.0    8
39.0    8
Name: count, dtype: int64
```

In [6]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches
```


In [6]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Place feature"
sns.countplot(x='Place', data=df, palette="Flare")
plt.show()

Place
Chennai    [1]
Mumbai     [1]
Calcutta   [1]
Delhi      [1]
Bangalore  [1]
Cochin     [1]
Mysore     [1]
Hyderabad  [1]
Bhopal     [1]
Nagpur     [1]
Place
Name: count, dtype: int64
```


In [6]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Flare")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [6]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Flare")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [6]:
```
# descriptive statistics
df.describe(include=['object'])
```

Out[6]:

	Company	Place	Country
count	1998	1998	1998
unique	4	10	1
top	TCS	Mumbai	India
freq	501	207	1998

In [6]:
```
# calculate the average income
average_salary = df['Salary'].mean()
print("The average salary: {average_salary:.2f} ")

# calculate the average income for people above 30
above_average_count = df[df['Salary'] > average_salary].sum()
below_average_count = df[df['Salary'] < average_salary].sum()

# calculate ratio and print the results
ratio = above_average_count / below_average_count
print("The ratio of people with income above average to below average: {ratio:2f} ")
print("Number of people with income above average: {above_average_count}")
print("Number of people income below the average: {below_average_count}")


# plot the ratio using seaborn
import seaborn as sns
sns.barplot(x=['Above Average', 'Below Average'], y=[above_average_count, below_average_count], palette="Flare")
plt.title('Ratio of People with Income Above Average to Below Average')
plt.show()

The average Salary: 5014.00
Number of people income above average to below average: 77:144
Number of people income above the average: 30
Number of people income below the average: 30
```


In [6]:
```
# bar charts analysis "for numerical values "This enables us to see if there are outliers or not"

sns.set_theme(palette="Viridis")
```

In [7]:
```
#histogram to understand the distribution
import warnings as w
w.filterwarnings('ignore')

for i in df.select_dtypes(include=['number']):
    sns.histplot(data=df,x=i,bins=15,kde='True')
    plt.show()
```


In [7]:
```
# Company' is ratio to count and Gender is Number\ values
df.groupby('Gender').count()['Company']*100/df.shape[0]
print(genderanalysis)

# bar charts analysis "for gender feature"
sns.countplot(x='Gender', data=df, palette="Viridis")
plt.show()

Gender
0   [1]
1   [1]
Name: count, dtype: int64
```


In [7]:
```
# "Company" is ratio to count and value\ values
age_analysis = df.age.value_counts(ratio=True)*100
print(age_analysis)

# create a pie chart "for Age\ratio feature"
plt.figure(figsize=(10, 12)) # figure in inches

# label by descending order
plt.pie(age_analysis[:10],
        labels=age_analysis[:10].index[:10],
        startangle=90,
        autopct='%1.1f%%',
        colors=sns.color_palette("Viridis"),
        explode=[0.05,0.05,0.05,
                 ])

plt.axis('equal') # equal to set the aspect ratio of the plot to be equal.
plt.title('Age\ratio Drawn Distribution')
plt.show()

Age
40.0   10
35.0    9
30.0    9
20.0    8
21.0    8
22.0    8
23.0    8
24.0    8
25.0    8
26.0    8
27.0    8
28.0    8
29.0    8
31.0    8
32.0    8
33.0    8
34.0    8
36.0    8
37.0    8
38.0    8
39.0    8
Name: count, dtype: int64
```

In [7]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches
```


In [7]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Place feature"
sns.countplot(x='Place', data=df, palette="Viridis")
plt.show()

Place
Chennai    [1]
Mumbai     [1]
Calcutta   [1]
Delhi      [1]
Bangalore  [1]
Cochin     [1]
Mysore     [1]
Hyderabad  [1]
Bhopal     [1]
Nagpur     [1]
Place
Name: count, dtype: int64
```


In [7]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Viridis")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [7]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Viridis")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [7]:
```
# descriptive statistics
df.describe(include=['object'])
```

Out[7]:

	Company	Place	Country
count	1998	1998	1998
unique	4	10	1
top	TCS	Mumbai	India
freq	501	207	1998

In [7]:
```
# calculate the average income
average_salary = df['Salary'].mean()
print("The average salary: {average_salary:.2f} ")

# calculate the average income for people above 30
above_average_count = df[df['Salary'] > average_salary].sum()
below_average_count = df[df['Salary'] < average_salary].sum()

# calculate ratio and print the results
ratio = above_average_count / below_average_count
print("The ratio of people with income above average to below average: {ratio:2f} ")
print("Number of people with income above average: {above_average_count}")
print("Number of people income below the average: {below_average_count}")


# plot the ratio using seaborn
import seaborn as sns
sns.barplot(x=['Above Average', 'Below Average'], y=[above_average_count, below_average_count], palette="Viridis")
plt.title('Ratio of People with Income Above Average to Below Average')
plt.show()

The average Salary: 5014.00
Number of people income above average to below average: 77:144
Number of people income above the average: 30
Number of people income below the average: 30
```


In [7]:
```
# bar charts analysis "for numerical values "This enables us to see if there are outliers or not"

sns.set_theme(palette="Pastel1")
```

In [8]:
```
#histogram to understand the distribution
import warnings as w
w.filterwarnings('ignore')

for i in df.select_dtypes(include=['number']):
    sns.histplot(data=df,x=i,bins=15,kde='True')
    plt.show()
```


In [8]:
```
# Company' is ratio to count and Gender is Number\ values
df.groupby('Gender').count()['Company']*100/df.shape[0]
print(genderanalysis)

# bar charts analysis "for gender feature"
sns.countplot(x='Gender', data=df, palette="Pastel1")
plt.show()

Gender
0   [1]
1   [1]
Name: count, dtype: int64
```


In [8]:
```
# "Company" is ratio to count and value\ values
age_analysis = df.age.value_counts(ratio=True)*100
print(age_analysis)

# create a pie chart "for Age\ratio feature"
plt.figure(figsize=(10, 12)) # figure in inches

# label by descending order
plt.pie(age_analysis[:10],
        labels=age_analysis[:10].index[:10],
        startangle=90,
        autopct='%1.1f%%',
        colors=sns.color_palette("Pastel1"),
        explode=[0.05,0.05,0.05,
                 ])

plt.axis('equal') # equal to set the aspect ratio of the plot to be equal.
plt.title('Age\ratio Drawn Distribution')
plt.show()

Age
40.0   10
35.0    9
30.0    9
20.0    8
21.0    8
22.0    8
23.0    8
24.0    8
25.0    8
26.0    8
27.0    8
28.0    8
29.0    8
31.0    8
32.0    8
33.0    8
34.0    8
36.0    8
37.0    8
38.0    8
39.0    8
Name: count, dtype: int64
```

In [8]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches
```


In [8]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Place feature"
sns.countplot(x='Place', data=df, palette="Pastel1")
plt.show()

Place
Chennai    [1]
Mumbai     [1]
Calcutta   [1]
Delhi      [1]
Bangalore  [1]
Cochin     [1]
Mysore     [1]
Hyderabad  [1]
Bhopal     [1]
Nagpur     [1]
Place
Name: count, dtype: int64
```


In [8]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Pastel1")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [8]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Company feature"
sns.countplot(x='Company', data=df, palette="Pastel1")
plt.show()

Company
TCS          [1]
Infosys      [1]
CTS          [1]
Tata Consultancy Services  [1]
Congizant    [1]
Infosys-Pvt Lmt  [1]
Name: count, dtype: int64
```


In [8]:
```
# descriptive statistics
df.describe(include=['object'])
```

Out[8]:

	Company	Place	Country
count	1998	1998	1998
unique	4	10	1
top	TCS	Mumbai	India
freq	501	207	1998

In [8]:
```
# calculate the average income
average_salary = df['Salary'].mean()
print("The average salary: {average_salary:.2f} ")

# calculate the average income for people above 30
above_average_count = df[df['Salary'] > average_salary].sum()
below_average_count = df[df['Salary'] < average_salary].sum()

# calculate ratio and print the results
ratio = above_average_count / below_average_count
print("The ratio of people with income above average to below average: {ratio:2f} ")
print("Number of people with income above average: {above_average_count}")
print("Number of people income below the average: {below_average_count}")


# plot the ratio using seaborn
import seaborn as sns
sns.barplot(x=['Above Average', 'Below Average'], y=[above_average_count, below_average_count], palette="Pastel1")
plt.title('Ratio of People with Income Above Average to Below Average')
plt.show()

The average Salary: 5014.00
Number of people income above average to below average: 77:144
Number of people income above the average: 30
Number of people income below the average: 30
```


In [8]:
```
# bar charts analysis "for numerical values "This enables us to see if there are outliers or not"

sns.set_theme(palette="Dark2")
```

In [9]:
```
#histogram to understand the distribution
import warnings as w
w.filterwarnings('ignore')

for i in df.select_dtypes(include=['number']):
    sns.histplot(data=df,x=i,bins=15,kde='True')
    plt.show()
```


In [9]:
```
# Company' is ratio to count and Gender is Number\ values
df.groupby('Gender').count()['Company']*100/df.shape[0]
print(genderanalysis)

# bar charts analysis "for gender feature"
sns.countplot(x='Gender', data=df, palette="Dark2")
plt.show()

Gender
0   [1]
1   [1]
Name: count, dtype: int64
```


In [9]:
```
# "Company" is ratio to count and value\ values
age_analysis = df.age.value_counts(ratio=True)*100
print(age_analysis)

# create a pie chart "for Age\ratio feature"
plt.figure(figsize=(10, 12)) # figure in inches

# label by descending order
plt.pie(age_analysis[:10],
        labels=age_analysis[:10].index[:10],
        startangle=90,
        autopct='%1.1f%%',
        colors=sns.color_palette("Dark2"),
        explode=[0.05,0.05,0.05,
                 ])

plt.axis('equal') # equal to set the aspect ratio of the plot to be equal.
plt.title('Age\ratio Drawn Distribution')
plt.show()

Age
40.0   10
35.0    9
30.0    9
20.0    8
21.0    8
22.0    8
23.0    8
24.0    8
25.0    8
26.0    8
27.0    8
28.0    8
29.0    8
31.0    8
32.0    8
33.0    8
34.0    8
36.0    8
37.0    8
38.0    8
39.0    8
Name: count, dtype: int64
```

In [9]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches
```


In [9]:
```
# "Company" is ratio to count and value\ values
Placeanalysis = df.Place.value_counts(ratio=True)*100
print(Placeanalysis)

plt.figure(figsize=(10, 12)) # width, height in inches

# bar charts analysis "for Place feature"
sns.countplot(x='Place', data=df, palette="Dark2")
plt.show()

Place
Chennai    [1]
Mumbai     [1]
Calcutta   [1]
Delhi      [1]
Bangalore  [1]
Cochin     [1]
Mysore     [1]
Hyderabad  [1]
Bhopal     [1]
Nagpur     [1]
Place
Name: count, dtype: int64
```

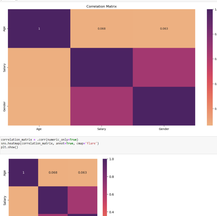

In [9]:
```
# "Company" is ratio to count and value\ values
Companyanalysis = df.Company.value_counts(ratio=True)*100
print(Companyanalysis
```

response of these curves are not in ballshape.

- Correlation Analysis

```
pds.displaytable(df)
```

In [20]:
```
# Correlation matrix using heatmap
# This shows correlation coefficients between all pairs of numerical variables in the dataset
correlation_matrix = df.corr(numeric_only=True)

# Create a heatmap figure with specified size (in inches)
pds.figure(figsize=(5, 5))

# Generate a heatmap to visualize the correlation matrix
# annot=True shows correlation values, cmap='viridis' uses a small-arrite the data value in each cell
sns.heatmap(correlation_matrix, annot=True, cmap='viridis')

# Add a title to the plot
pds.title('Correlation Matrix')

# Display the heatmap
pds.show()
```

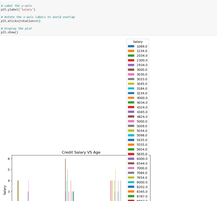


In [21]:
```
correlation_matrix = corr(numeric_only=True)
# Create a heatmap visualization with specified colormap
pds.show()
```

There is positive correlation between **"Salary"** and **"Age"**

Likely Relationship analysis

In [24]:
```
# Create a scatter plot to visualize relationship between "Salary" and "Company" variables
# This helps identify patterns in salary distribution across different companies
pds.figure(figsize=(12, 6))

# Add a title to the plot
pds.title('Credit Salary VS Age')

# Label the x-axis
pds.xlabel('Company')

# Label the y-axis
pds.ylabel('Salary')

# Rotate the x-axis labels to avoid overlap
pds.xticks(rotation=45)

# Display the plot
pds.show()
```



Infosis giving highest salary

In [26]:
```
# Create a scatter plot visualization of "Salary" and "Company" variables
# This helps identify patterns in salary distribution across different companies
pds.figure(figsize=(12, 6))

# Add a title to the plot
pds.title('Credit Salary VS Age')

# Label the x-axis
pds.xlabel('Salary')

# Label the y-axis
pds.ylabel('Age')

# Rotate the x-axis labels to avoid overlap
pds.xticks(rotation=45)

# Display the plot
pds.show()
```

Data Cleaning

Data cleaning refers to the process of **identifying**, **correcting**, and **removing errors**, **inconsistencies**, and **inaccuracies** from a dataset. It involves transforming raw data into a clean and structured format that is suitable for analysis, modeling, and other data-oriented tasks. Including:

- Remove Unneeded Features
- Handling Missing Values
- Encoding Categorical Variables
- Handling Outliers
- Feature Engineering

#1. Remove Unneeded Features

As Ones are completely unique and not correlated with any of the other values, So we will drop it using .drop() function. But before we do that we need to check:

- df['Name'].value_ counts ().sort_ ascending=False
- Check if the column is a key identifier in the original dataset(Yes, replace = False (default) means leaving the original dataset unchanged.

2. Handling Missing Values

"Missing values" are data points that are absent or not recorded for certain variables in a dataset. This can happen because of mistakes such as data entry errors, equipment failures, or simply because the information is not available.

"How to detect missing values??"
look for blank cells and use data analysis tools that can identify missing values like .isna()() function.

"Decision options:??"
- Drop Rows
  - Drop rows, if missing values can lead to biased analyses and inaccurate predictions.
  - Suitable when the missing data is random and represents a small portion (<5%).
  - Impractical if missing values are widespread or concentrated values based on the available data. This can be done using various methods such as:
    - **"Mean/Median Imputation:"** Replace with the average (mean) or middle value (median) for numerical data.
    - **"Mode Imputation:"** Replace with the most frequent value for categorical data.
    - **"Forward Fill/Backward Fill:"** Use the previous or subsequent value to fill gaps.
    - **"Predictive Modeling:"** Use other missing values can be filled with the last observed value (Forward Fill) or the next observed value (Backward Fill).
    - **"Interpolation:"** Estimate missing values based on the trend or pattern observed in the existing data points.
  - Suitable when missing values based on the trend or pattern observed in the existing data points.
  - Helps maintain data integrity, especially if missing values are extensive and cannot be reasonably imputed without introducing significant bias.)

In [29]:
```
# Check for missing values
null_counts = df.isna().sum()

# Display the number of null values
print(null_counts)
```

```
Company    0
Age        0
Salary     0
Place      0
Country    0
Gender     0
Year       0
dtype: int64
```

There are two types of Data:
- Numerical Data
- Categorical Data

In [31]:
```
# Fill the missing values for numerical data, filling = fill (NaN) values

df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
df['Age'] = df['Age'].fillna(df['Age'].mean())


# Mode Imputation for categorical data
# Find the most frequent value (mode) for each column
# If this returns NaN if there are multiple mode values, only the first one is selected
df['Company'] = df['Company'].fillna(df['Company'].mode()[0])
# Check if there are any NaN values left
# The first mode (if) is the case mode(s) in the data (for example,
# The first mode (if) is the
# The best mode fill is the credit Salary is the mode (most common value)


# Fill the missing values for categorical data
# Find the most frequent value (mode) for each column
df['Place'] = df['Place'].fillna(df['Place'].mode()[0])
```

In [34]:
```
# Check for missing values after handling
null_counts = df.isna().sum()

# Display the number of null values after handling
print(null_counts)
```

```
Company    0
Age        0
Salary     0
Place      0
Country    0
Gender     0
Year       0
dtype: int64
```

Now there is no null values in the data set.

3. Encoding Categorical Variables

In [36]:
```
# Convert categorical variables into dummy/indicator variables through a process called "one-hot" encoding
df_one_hot_encoded = pd.get_dummies(data = df, columns = ['Gender', 'Age'])

print(df_one_hot_encoded )

print("The shape of df_shape)                                                    ")
print(df.shape)
```

```
   Company  Age   Salary  Place Country  Gender_M  Gender_F  \
0  Infosys   20  5.52.00  Pune   India     True     False
1  Infosys   20  5.52.00Pune   Pune   India     True     False
2  Infosys   20  5.52.00Pune   Pune   India     True     False
3      TCS   21  2.00.00000   Pune   India     True     False
4      TCS   21  2.00.00000   Pune   India     True     False
5  Infosys   21  2.00.00000  Jaipur   India     False    True
6  Infosys   21  2.00.00000  Jaipur   India     False    True
7  Infosys   21  2.00.00000  Jaipur   India     False    True
8  Infosys   21  2.00.00000  Jaipur   India     False    True
9  Infosys   21  2.00.00000  Jaipur   India     False    True

   Age_19.0  Age_1.2E+3  Age_20.0  Age_21.0  Age_28.0  Age_29.0  \
0     False       False      True     False     False     False
1     False       False      False    False     False     False
2     False       False      False    False     False     False
3     False       False      False    False     False     False
4     False       False      False    False     False     False
5     False       False      False    False     False     False
6     False       False      False    False     False     False
7     False       False      False    False     False     False
8     False       False      False    False     False     False
9     False       False      False    False     False     False

   Age_31.0  Age_35.0  Age_36.0  Age_37.0  Age_38.0  Age_39.0  \
0     False     False     False     False     False     False
1     False     False     False     False     False     False
2     False     False     False     False     False     False
3     False     False     False     False     False     False
4     False     False     False     False     False     False
5     False     True      False     False     False     False
6     False     False     True      False     False     False
7     False     False     False     False     False     False
8     False     False     False     False     False     False
9     False     False     False     False     False     False

   Age_40.0
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     True

[10 rows x 37 columns]
The shape = (200, 7)
```

4. Handling Outliers

"Outliers" are data points that significantly differ from the rest of the observations in a dataset, often due to errors in measurement or unusual events.

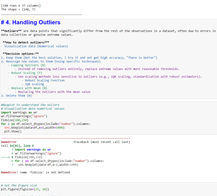
"How to detect outliers??"
- Visualization (box plot/scat values)

"Decision options:??"
- Remove Outliers: If they are impact validation, if they is and not get high accuracy, "There is better"?
- Replace the value (based on domain specific knowledge)
  - Capping Outliers:
    - Cap outliers at a threshold by setting values exceeding, replace extreme values with more reasonable thresholds.
    - Robust Scaling Methods: Less sensitive to outliers (e.g., IQR scaling, standardization with robust estimators).
      - Robust Scaling function
      - Log Transform
    - Replace with mean %
    - Replace outliers with the mean value
  - Delete them (%)

In [39]:
```
# Visualize to understand the outliers
# Create visualization data numerical values
# Select numerical columns
numerical_cols = df[['Salary']]
# Create figure with specified size
for i in df.select_dtypes(include=['number']).columns:
    sns.boxplot(data=df,x=i,palette='viridis')
    pds.show()
```

```
NameError                                 Traceback (most recent call last)
<ipynb_cell_name_line_5>
    1 for i in df.select_dtypes(include=['number']).columns:
    2     sns.boxplot(data=df,x=i,palette='viridis')
    3     pds.show()

NameError: name 'Salary' is not defined
```

In [40]:
```
# Set the figure size
pds.figure(figsize=(7, 5))

# Only variables that have outliers
outlierVariables = df.get('Salary', 'Age', 'Gender')

# Add outliers to the plot
sns.stripplot(data=outlierVariables, orient='h')

# Set the axis labels and title
pds.xlabel('Outliers')

# Show the plot
pds.show()
```

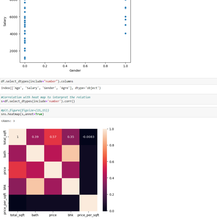


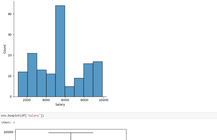
In [ ]:

In [ ]:

In [ ]:
```
# Analyze relationship between numerical variables
# Create correlation matrix
numerical_cols = df[['Salary', 'Age']]
correlation_matrix = numerical_cols.corr()

# Create figure with specified size
pds.figure(figsize=(8, 6))

# Create heatmap with annotations and viridis colormap
sns.heatmap(correlation_matrix, annot=True, cmap='viridis')

# Add title
pds.title('Correlation Matrix')

# Display the plot
pds.show()
```

In [ ]:
```
# Create scatter plot to visualize relationship
# Set figure size
pds.figure(figsize=(10, 6))

# Create scatter plot with salary on y-axis and age on x-axis
sns.scatterplot(data=df, x='Age', y='Salary')

# Add title
pds.title('Salary vs Age')

# Display the plot
pds.show()
```





In [20]: df.select_dtypes(include=['number'])columns

In [20]: [column] = df['Salary', 'Gender', 'Age'], dtype='object')

In [20]: dfcorrelation with Dual map to compare the correlation
ccaf.select_dtypes(include=['number'])corr()

In [20]: df[['Age']].describe()
df[['Salary']].describe()

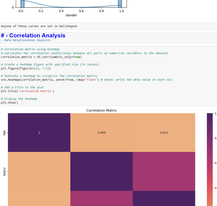
Out[20]: count 7

In [ ]:

Step : 6- Outliers treatment

In [20]: ins.boxplot(df['Salary'])

Out[20]: <seaborn.axisgrid.FacetGrid at 0x1b2d7f0>



In [20]: ins.boxplot(df['Salary'])

Out[20]: <seaborn.a

No major outliers detected

No need of outlier handling ...

In [ ]:

In [ ]: # IQR Methode, Percentile

In [36]: *Boolean whether to do outlier treatment or not , if do true*
 # Doing outlier treatment for continuous columns, Price and Supernfoot

```
def plotbox(df):
    for col in df.select_dtypes(np.number):
        plt.figure()
        sns.boxplot(data=df, x=col)
        plt.title(f'{col}')
        plt.show()
return True,df
```

In [ ]:

In [37]: plotbox(df[['Age']])

Out[37]: (-0.42499999999999, 64.575, 24.45)

In [38]: for i in df[['Age', 'Salary']]:
    lw,up=outlier(df[i],1)
    print("upper_limit",up,"lower_limit",lw)
    df[i]=np.where(df[i]>up,up,df[i])
    df[i]=np.where(df[i]<lw,lw,df[i])
    print(i,"done")
print("outlier treated")
cs.shape#outlier treated
cs.describe#outlier do 97% do.01
cs.5 2.5 1.0

In [39]: df.columns

Out[39]: Index(['Company', 'Age', 'Salary', 'Place', 'Country', 'Gender', 'Age'], dtype='object')

Z-Score Methode

In [39]: *#find the limits*
upper_limit=df['Age'].mean()+3*df['Age'].std()
lower_limit=df['Age'].mean()-3*df['Age'].std()
print("upper_limit",upper_limit,"lower_limit",lower_limit)

upper_limit 84.57500000000007
lower_limit 44.37499999999992

In [39]: *# find the outliers*
df.loc[df['Salary']>upper_limit] | df[df['Salary']<lower_limit]

Out[39]:

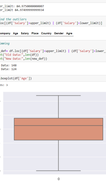
	Company	Age	Salary	Place	Country	Gender	Age

In [39]: *#Cleaning*

```
new_df=df.loc[df['Salary']<upper_limit] | df[df['Salary']>lower_limit]
print("New shape",cs[cs.Age_Out])

Old Data: 100
New Data: 100
```

In [39]: sns.boxplot(df['Age'])

Out[39]: <Axes: >



In [39]: *#Capping -Change the outlier values to upper lower limit values*
upper_limit,lower_limit=lw,up
new_df=np.where(df['Salary']>upper_limit,upper_limit,
new_df=np.where(df['Salary']>lower_limit,df['Salary'],lower_limit

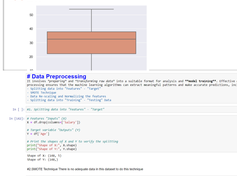
In [39]: sns.boxplot(new_df['Salary'])

Out[39]: <Axes: >

In [39]: *#Capping -Change the outlier values to upper lower limit values*
upper_limit,lower_limit=lw,up
new_df=np.where(df['Salary']>upper_limit,upper_limit
new_df=np.where(df['Salary']>lower_limit,df['Salary'],lower_limit

sns.boxplot(new_df['Age'])

Out[39]: <Axes: >



Data Preprocessing

After the initial exploration and cleaning the dataf into a suitable format for analysis and **"model training"**. Effective data preprocessing ensures that the machine learning algorithms can extract meaningful patterns and make accurate predictions, includes:
- Splitting data into training and testing sets using the 80/20 technique
- Encoding categorical and normalizing the features
- Splitting data into "training", "testing" data

In [ ]: *#x: splitting data into "features", "target"*

In [39]: *# features "Impact" (X)*
X = df[['Impact', 'Salary']]

 # Target variable "Outcome" (Y)
Y = df['Age']

 # print the shapes of X and Y to verify the splitting
print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)

Shape of X: (100, 7)
Shape of Y: (100,)

#2.INSPECTS Technique There is no ansupdo data in this dataset to do this technique

#we Normalize and normalize the features

StandardScaler,OneHotEncoder
#I will choose one of them in the future part "model selection" based on the highest accuracy

In [39]: ***standard_scaler = StandardScaler()***
X = standard_scaler.fit_transform(X)

#one-hot encoding (encoding)
 # encoder = OneHotEncoder()
 # X = encoder.fit_transform(X)

Out[40]: 'Standard_scaler = StandardScaler()\nX = standard_scaler.fit_transform(X)\nnfor-hot encoding (encoding)\nnnc_ohe_scaler = ncoder.fit_transform(X)'

#x: splitting data into "Training", "Testing"

In [39]: *# split the data into training and testing sets*
 # X: features, Y: target variable
 # test_size=0.20: 20% of the data will be used for testing and 80% for training
 # random_state=42: ensures the random seed for reproducibility
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)

 # print the shapes of the training and testing sets to verify the splitting
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of Y_train:", Y_train.shape)
print("Shape of Y_test:", Y_test.shape)

Shape of X_train: (80, 4)
Shape of X_test: (20, 4)
Shape of Y_train: (80,)
Shape of Y_test: (20,)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: