20th May 2025

In [1]:
```
import pandas as pd
```

In [2]:
```
pd.__version__
```

Out[2]:
```
'2.1.2'
```

In [3]:
```
pip install --upgrade openpyxl
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: openpyxl in c:\users\windows10 pro\appdata\roaming\py
thon\python312\site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in c:\programdata\anaconda3\lib\site-packa
ges (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
emp = pd.read_excel(r"C:\Users\Windows10 Pro\Downloads\DataScience_AI\2025\May2025\
```

In [5]:
```
emp
```

Out[5]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [6]:
```
id(emp)
```

Out[6]:
```
2039297417792
```

In [7]:
```
emp.shape
```

Out[7]:
```
(6, 6)
```

In [8]:
```
emp.dtypes
```

Out[8]:
```
Name        object
Domain      object
Age         object
Location    object
Salary      object
Exp         object
dtype: object
```

In [9]:
```
emp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       4 non-null      object
 3   Location  4 non-null      object
 4   Salary    6 non-null      object
 5   Exp       5 non-null      object
dtypes: object(6)
memory usage: 420.0+ bytes
```

In [10]:
```
emp.head()
```

Out[10]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year

In [11]:
```
emp.isnull()   # True means missing values, False means non-missing values
```

Out[11]:

	Name	Domain	Age	Location	Salary	Exp
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	True	True	False	False
3	False	False	True	False	False	True
4	False	False	False	True	False	False
5	False	False	False	False	False	False

In [12]:
```
emp.isnull().sum()
```

Out[12]:
```
Name        0
Domain      0
Age         2
Location    2
Salary      0
Exp         1
dtype: int64
```

In [13]:
```
emp.isna()   # True means missing values
```

Out[13]:

	Name	Domain	Age	Location	Salary	Exp
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	True	True	False	False
3	False	False	True	False	False	True
4	False	False	False	True	False	False
5	False	False	False	False	False	False

In [14]:
```
emp.columns
```

Out[14]:
```
Index(['Name', 'Domain', 'Age', 'Location', 'Salary', 'Exp'], dtype='object')
```

Data Cleaning or Data Cleansing

In [15]:
```
emp
```

Out[15]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [16]:
```
emp['Name']
```

Out[16]:
```
0      Mike
1    Teddy^
2     Uma#r
3      Jane
4    Uttam*
5       Kim
Name: Name, dtype: object
```

In [17]:
```
emp['Name'] = emp['Name'].str.replace(r'\W', '', regex=True)   # non word character
```

In [18]:
```
emp['Name']
```

Out[18]:
```
0     Mike
1    Teddy
2     Umar
3     Jane
4    Uttam
5      Kim
Name: Name, dtype: object
```

In [19]:
```
emp['Domain']
```

Out[19]:
```
0     Datascience#$
1           Testing
2    Dataanalyst^^#
3       Ana^^lytics
4        Statistics
5               NLP
Name: Domain, dtype: object
```

In [20]:
```
emp['Domain'] = emp['Domain'].str.replace(r'\W', '', regex=True)   # r for raw stri
```

In [21]:
```
emp['Domain']
```

Out[21]:
```
0    Datascience
1        Testing
2    Dataanalyst
3      Analytics
4     Statistics
5            NLP
Name: Domain, dtype: object
```

In [22]:
```
emp['Age']
```

Out[22]:
```
0    34 years
1      45' yr
2         NaN
3         NaN
4       67-yr
5        55yr
Name: Age, dtype: object
```

In [23]:
```
emp['Age'] = emp['Age'].str.replace(r'\W', '', regex =True)
```

In [24]:
```
emp['Age']
```

Out[24]:
```
0    34years
1       45yr
2        NaN
3        NaN
4       67yr
5       55yr
Name: Age, dtype: object
```

In [25]:
```
emp['Age'] = emp['Age'].str.extract('(\d+)')    # d means digit
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_16340\1639245287.py:1: SyntaxWar
ning: invalid escape sequence '\d'
  emp['Age'] = emp['Age'].str.extract('(\d+)')    # d means digit
```

In [26]:
```
emp['Age']
```

Out[26]:
```
0     34
1     45
2    NaN
3    NaN
4     67
5     55
Name: Age, dtype: object
```

Regular Expression

The expression str.replace(r'\W','',regex=True) is used in pandas to remove all non-word characters from a string column in a DataFrame. Here's how it works:

r'\W': This is a regular expression pattern where:

\W matches any non-word character (anything other than letters, digits, or underscores).

The r'' prefix makes it a raw string, ensuring that backslashes are treated literally.

'': This is the replacement string, meaning all non-word characters will be replaced with an empty string (effectively removed).

regex=True: This tells pandas to interpret the pattern as a regular expression rather than a literal string.

In [27]:
```
emp['Location']
```

Out[27]:
```
0       Mumbai
1    Bangalore
2          NaN
3     Hyderbad
4          NaN
5        Delhi
Name: Location, dtype: object
```

In [28]:
```
emp['Location']=emp['Location'].str.replace(r'\W','',regex=True)
emp['Location']
```

Out[28]:
```
0       Mumbai
1    Bangalore
2          NaN
3     Hyderbad
4          NaN
5        Delhi
Name: Location, dtype: object
```

In [29]:
```
emp
```

Out[29]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5^00#0	2+
1	Teddy	Testing	45	Bangalore	10%%000	<3
2	Umar	Dataanalyst	NaN	NaN	1$5%000	4> yrs
3	Jane	Analytics	NaN	Hyderbad	2000^0	NaN
4	Uttam	Statistics	67	NaN	30000-	5+ year
5	Kim	NLP	55	Delhi	6000^$0	10+

In [30]:
```
emp['Salary']=emp['Salary'].str.replace(r'\W','',regex=True)
emp['Salary']
```

Out[30]:
```
0     5000
1    10000
2    15000
3    20000
4    30000
5    60000
Name: Salary, dtype: object
```

In [31]:
```
emp['Exp']=emp['Exp'].str.extract('(\d+)')
# str.extract('(\d+)') function in pandas is used to extract numeric values from a
emp['Exp']
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_16340\749592179.py:1: SyntaxWarn
ing: invalid escape sequence '\d'
  emp['Exp']=emp['Exp'].str.extract('(\d+)')
```

Out[31]:
```
0      2
1      3
2      4
3    NaN
4      5
5     10
Name: Exp, dtype: object
```

In [32]:
```
clean_data=emp.copy()
```

In [33]:
```
clean_data
```

Out[33]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

Missing value Treatment

In [34]:
```
clean_data
```

Out[34]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [35]:
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       4 non-null      object
 3   Location  4 non-null      object
 4   Salary    6 non-null      object
 5   Exp       5 non-null      object
dtypes: object(6)
memory usage: 420.0+ bytes
```

In [36]:
```
import numpy as np
```

In [37]:
```
clean_data
```

Out[37]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [38]:
```
clean_data.head(1)
```

Out[38]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2

In [39]:
```
clean_data['Age']
```

Out[39]:
```
0     34
1     45
2    NaN
3    NaN
4     67
5     55
Name: Age, dtype: object
```

Note :- To remenber the below Concept

The expression .fillna(np.mean(pd.to_numeric(clean_data['Age']))) is used in pandas to replace missing values (NaN) in the 'Age' column with the mean of the column. Here's how it

works:

Breakdown:

pd.to_numeric(clean_data['Age']):--- Converts the 'Age' column to numeric format, ensuring all values are treated as numbers.

np.mean(...): ---- Computes the mean of the numeric values in the 'Age' column.

.fillna(...):--- Replaces all NaN values with the computed mean.

In [40]:
```
clean_data['Age']=clean_data['Age'].fillna(np.mean(pd.to_numeric(clean_data['Age'])
```

In [41]:
```
clean_data['Age']
```

Out[41]:
```
0       34
1       45
2    50.25
3    50.25
4       67
5       55
Name: Age, dtype: object
```

In [42]:
```
emp
```

Out[42]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [43]:
```
clean_data
```

Out[43]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50.25	NaN	15000	4
3	Jane	Analytics	50.25	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [44]:
```
clean_data['Exp']
```

Out[44]:
```
0      2
1      3
2      4
3    NaN
4      5
5     10
Name: Exp, dtype: object
```

In [45]:
```
clean_data['Exp']=clean_data['Exp'].fillna(np.mean(pd.to_numeric(clean_data['Exp'])
```

In [46]:
```
clean_data['Exp']
```

Out[46]:
```
0      2
1      3
2      4
3    4.8
4      5
5     10
Name: Exp, dtype: object
```

In [47]:
```
emp
```

Out[47]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [48]:
```
clean_data
```

Out[48]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50.25	NaN	15000	4
3	Jane	Analytics	50.25	Hyderbad	20000	4.8
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

Note:-

The expression .fillna(clean_data['Location'].mode()[0]) in pandas is used to replace missing values (NaN) in the 'Location' column with the most frequently occurring value (mode) of that column.

Breakdown:

clean_data['Location'].mode():----- Computes the mode (most common value) of the 'Location' column.

[0]:----- Since .mode() returns a Series (which may contain multiple modes), [0] selects the first mode.

.fillna(...):-------- Replaces all NaN values with the selected mode.

In [49]:
```
clean_data['Location'] = clean_data['Location'].fillna(clean_data['Location'].mode(
clean_data['Location']
```

Out[49]:
```
0       Mumbai
1    Bangalore
2    Bangalore
3     Hyderbad
4    Bangalore
5        Delhi
Name: Location, dtype: object
```

In [50]:
```
clean_data
```

Out[50]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50.25	Bangalore	15000	4
3	Jane	Analytics	50.25	Hyderbad	20000	4.8
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

Converting the Numerical Data into Integer by using astype(int)

In [51]:
```
clean_data['Age']=clean_data['Age'].astype(int)
```

In [52]:
```
clean_data['Salary']=clean_data['Salary'].astype(int)
```

In [53]:
```
clean_data['Exp']=clean_data['Exp'].astype(int)
```

In [54]:
```
clean_data
```

Out[54]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [55]:
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       6 non-null      int32 
 3   Location  6 non-null      object
 4   Salary    6 non-null      int32 
 5   Exp       6 non-null      int32 
dtypes: int32(3), object(3)
memory usage: 348.0+ bytes
```

In [56]:
```
clean_data.to_csv('clean_data.csv') # converting the Dataset into CSV formate
```

Import OS

The os.getcwd() ------function in Python is used to get the current working directory of your script. It returns the absolute path of the directory where your Python script is running.

In [57]:
```
import os
os.getcwd()# Get the current Working Directory
```

Out[57]:
```
'C:\\Users\\Windows10 Pro'
```

Now We will Work with Matplotlib and Seaborn to get the better Visualization

After the Data Cleaning

In [58]:
```
import matplotlib.pyplot as plt # Visualization
```

In [59]:
```
import seaborn as sns # Advanced Visualization
```

To prevent errors, import the warnings module.

The warnings.filterwarnings('ignore') function in Python allows you to suppress all warnings in your code, ensuring a cleaner output.

In [60]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [61]:
```
clean_data
```

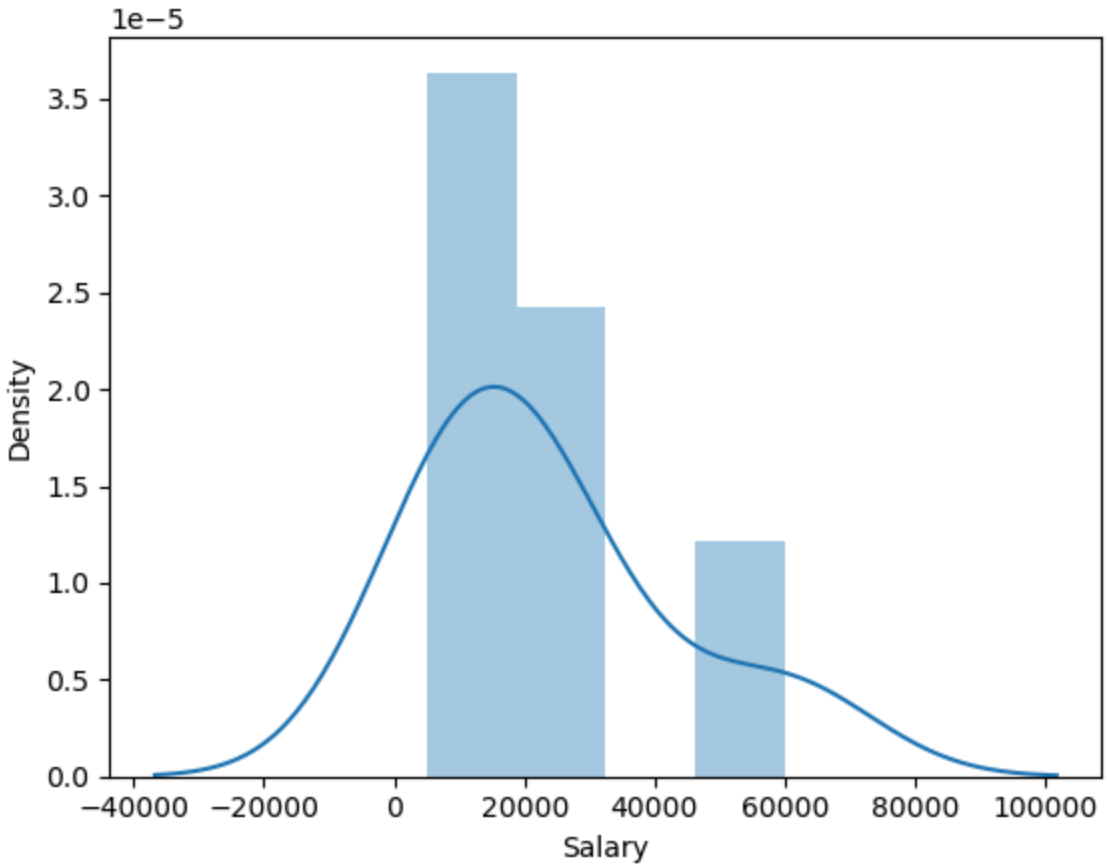
Out[61]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [62]:
```
clean_data['Salary']
```

Out[62]:
```
0     5000
1    10000
2    15000
3    20000
4    30000
5    60000
Name: Salary, dtype: int32
```

In [63]:
```
vis1=sns.distplot(clean_data['Salary'])
```



The plt.rcParams['figure.figsize'] = (10, 6) setting in Matplotlib is used to globally define the default figure size for plots. Here's how it works:

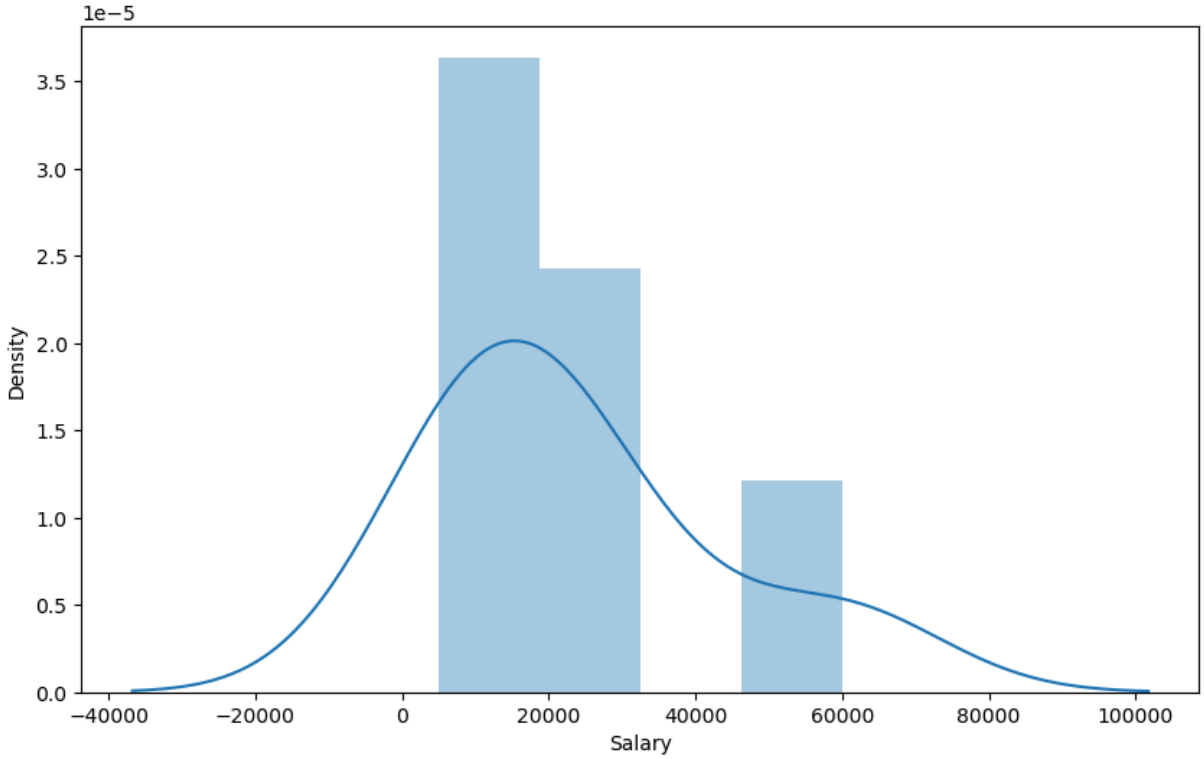
Breakdown:

plt.rcParams['figure.figsize']:--- This modifies the default figure size for all plots.

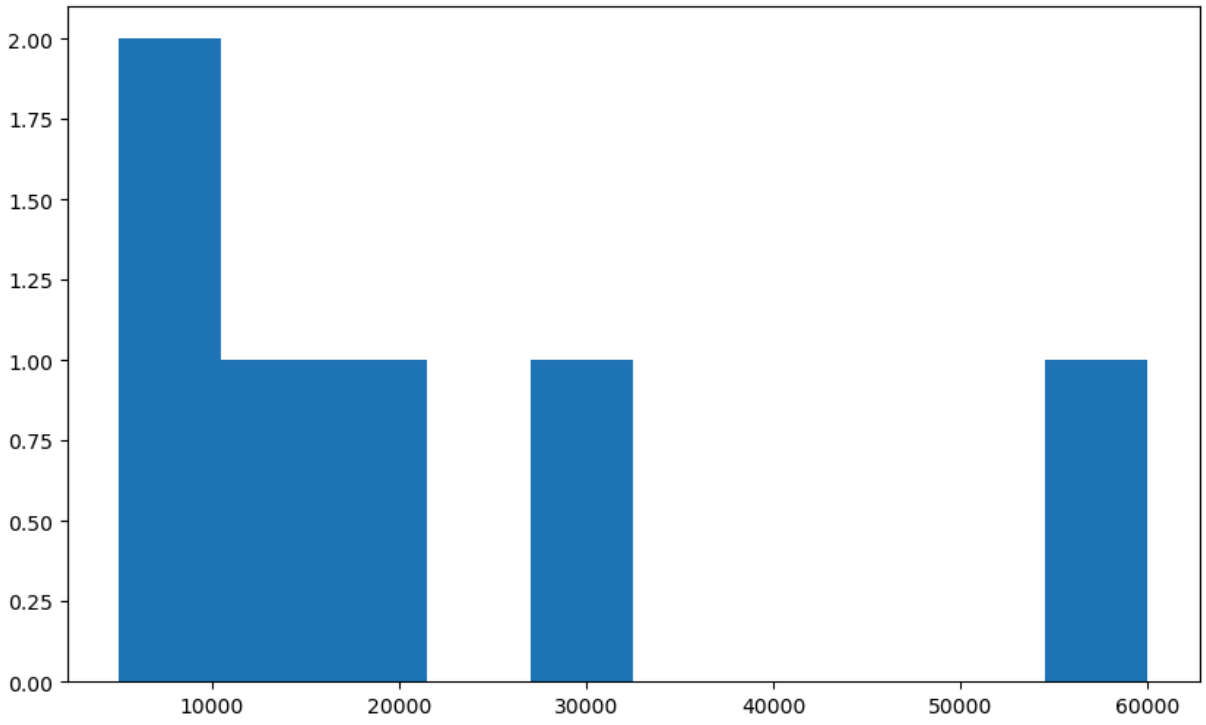
(10, 6): -----Specifies the width (10 inches) and height (6 inches) of the figure.

In [64]:
```
plt.rcParams['figure.figsize']=10,6
```

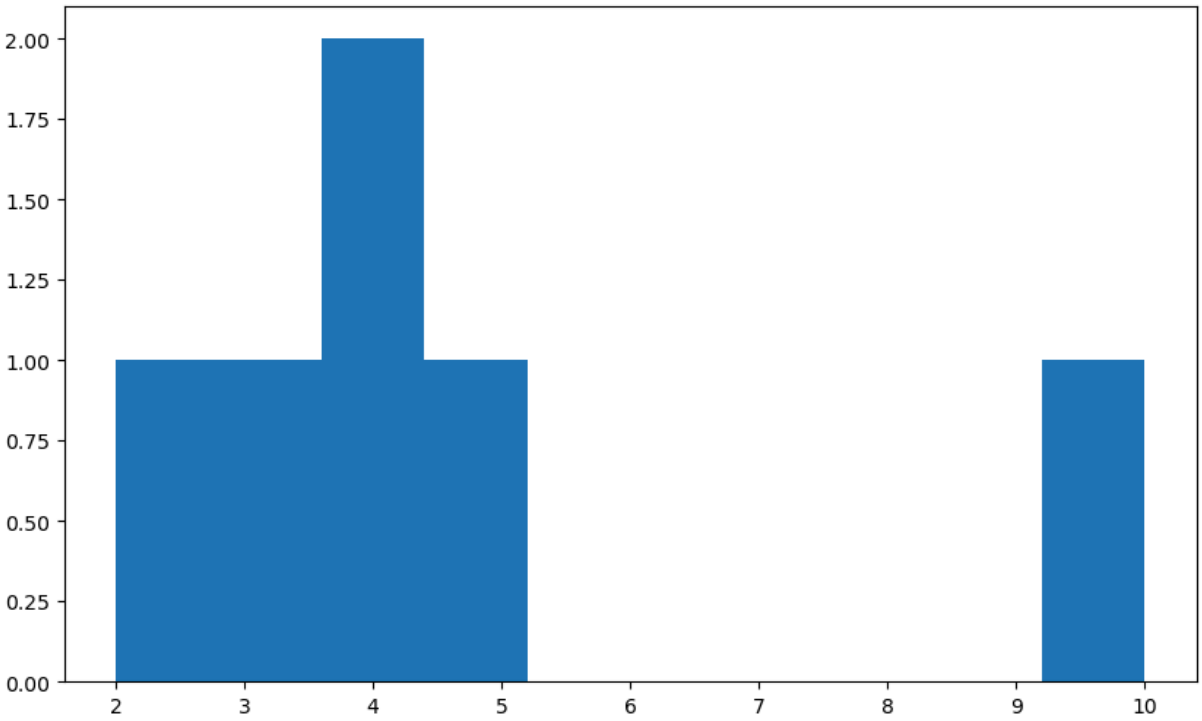
In [65]:
```
vis1=sns.distplot(clean_data['Salary'])
```



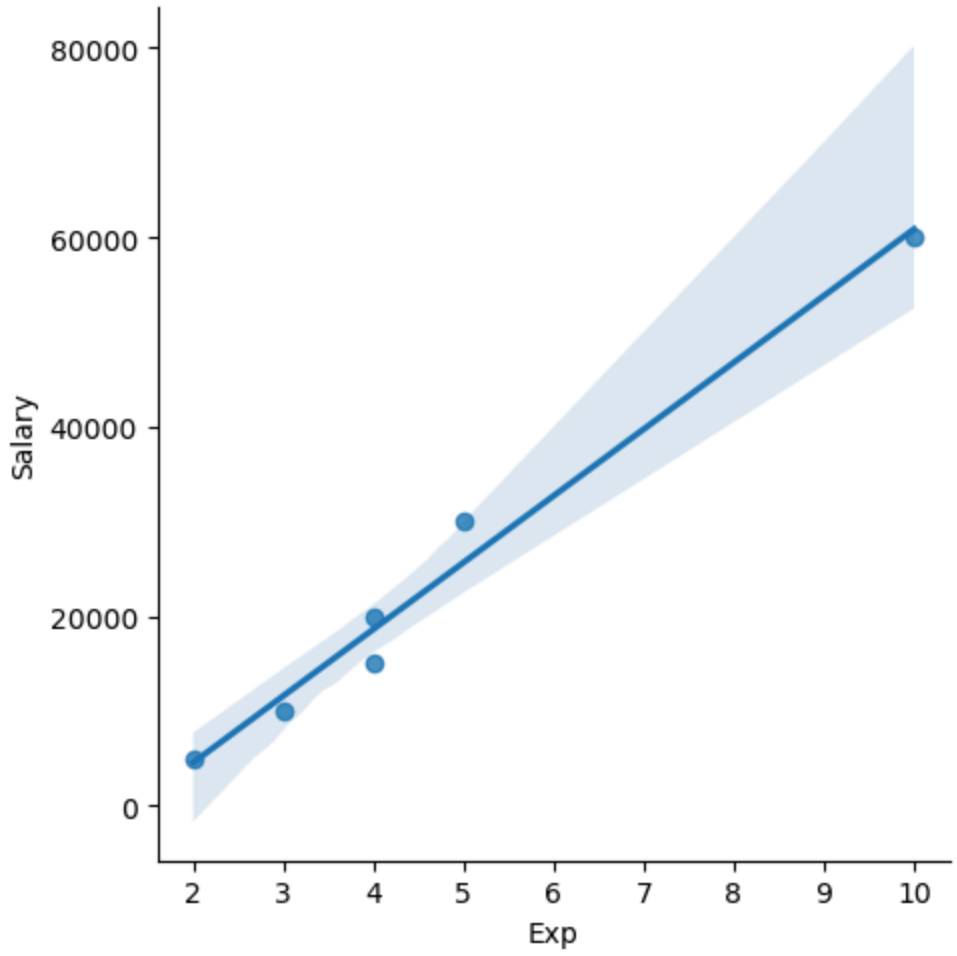
In [66]:
```
vis2 = plt.hist(clean_data['Salary'])# Hist is used for plot a Histogram
```



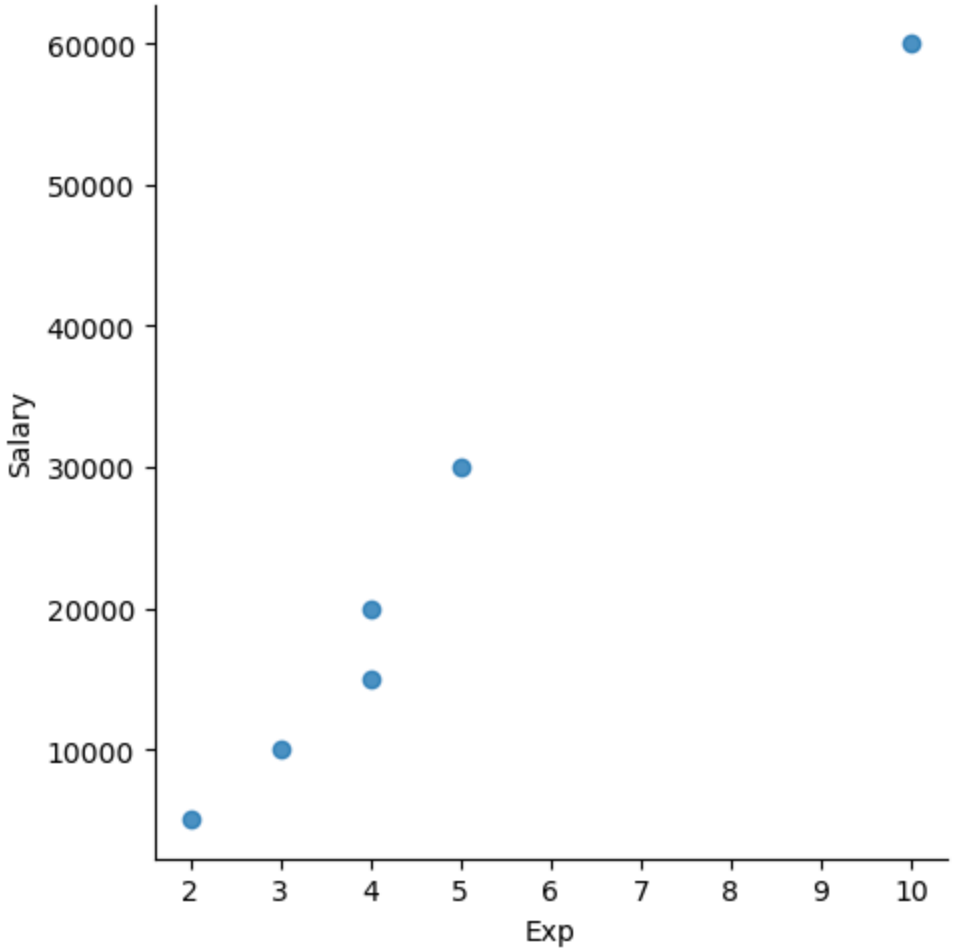
In [67]:
```
vis3 = plt.hist(clean_data['Exp'])
```



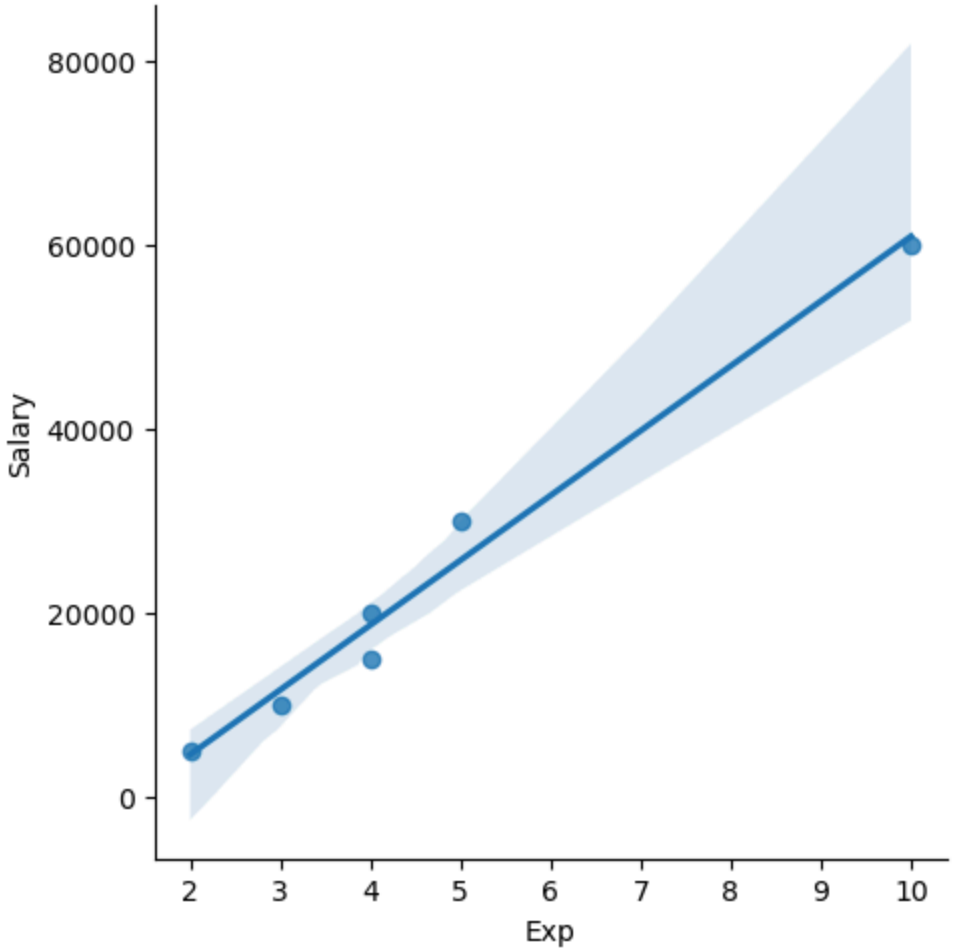
In [68]:
```
vis4=sns.lmplot(data=clean_data,x='Exp',y='Salary')
```



In [69]:
```
vis5=sns.lmplot(data=clean_data,x='Exp',y='Salary',fit_reg=False)#Fit_reg will remo
```



In [70]:
```
vis6 = sns.lmplot(data=clean_data,x = 'Exp', y='Salary', fit_reg = True)
```



In [71]:
```
clean_data
```

Out[71]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [72]:
```
clean_data[:]
```

Out[72]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [73]:
```
clean_data[:2]
```

Out[73]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3

In [74]:
```
clean_data[2:]
```

Out[74]:

	Name	Domain	Age	Location	Salary	Exp
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [75]:
```
clean_data[:]
```

Out[75]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [76]:
```
clean_data[0:1]
```

Out[76]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2

In [77]:
```
clean_data[0,3]

#It looks like you're encountering a KeyError in pandas when trying to access clean
#This happens because pandas does not support indexing with a tuple like (0,3).
#Instead, you should use .iloc[] for positional indexing.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\indexes\base.py:3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: (0, 3)

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[77], line 1
----> 1 clean_data[0,3]
      3 #It looks like you're encountering a KeyError in pandas when trying to access clean_data[0,3].
      4 #This happens because pandas does not support indexing with a tuple like (0, 3).
      5 #Instead, you should use .iloc[] for positional indexing.

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\frame.py:3893, in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\AppData\Roaming\Python\Python312\site-packages\pandas\core\indexes\base.py:3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)

KeyError: (0, 3)
```

In [78]:
```
clean_data.iloc[0, 3]
```

Out[78]:
```
'Mumbai'
```

In [79]:
```
x_iv=clean_data.drop(['Salary'],axis=1)# axis=1: Specifies that the operation shoul
```

In [80]:
```
x_iv
```

Out[80]:

	Name	Domain	Age	Location	Exp
0	Mike	Datascience	34	Mumbai	2
1	Teddy	Testing	45	Bangalore	3
2	Umar	Dataanalyst	50	Bangalore	4
3	Jane	Analytics	50	Hyderbad	4
4	Uttam	Statistics	67	Bangalore	5
5	Kim	NLP	55	Delhi	10

In [81]:
```
clean_data
```

Out[81]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [82]:
```
x_iv.columns
```

Out[82]:
```
Index(['Name', 'Domain', 'Age', 'Location', 'Exp'], dtype='object')
```

In [83]:
```
clean_data.columns
```

Out[83]:
```
Index(['Name', 'Domain', 'Age', 'Location', 'Salary', 'Exp'], dtype='object')
```

In [84]:
```
y_dv = clean_data.drop(['Name', 'Domain', 'Age', 'Location','Exp'],axis=1)# Delete
```

In [85]:
```
y_dv
```

Out[85]:

	Salary
0	5000
1	10000
2	15000
3	20000
4	30000
5	60000

In [86]:
```
clean_data
```

Out[86]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [87]:
```
x_iv
```

Out[87]:

	Name	Domain	Age	Location	Exp
0	Mike	Datascience	34	Mumbai	2
1	Teddy	Testing	45	Bangalore	3
2	Umar	Dataanalyst	50	Bangalore	4
3	Jane	Analytics	50	Hyderbad	4
4	Uttam	Statistics	67	Bangalore	5
5	Kim	NLP	55	Delhi	10

In [88]:
```
y_dv
```

Out[88]:

	Salary
0	5000
1	10000
2	15000
3	20000
4	30000
5	60000

In [89]:
```
clean_data
```

Out[89]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

NoW we introduce the Imputation(or transformer)

In [90]:
```
#Defination
#transformer :- Converting the Catagorical Vale  into Numerical value in Dataset
#The pd.get_dummies(clean_data) function in pandas is used for one-hot encoding, wh
```

In [91]:
```
imputation = pd.get_dummies(clean_data,dtype=int) # by default it will be Bool
```

In [92]:
```
imputation
```

Out[92]:

	Age	Salary	Exp	Name_Jane	Name_Kim	Name_Mike	Name_Teddy	Name_Umar	Nan
0	34	5000	2	0	0	1	0	0	
1	45	10000	3	0	0	0	1	0	
2	50	15000	4	0	0	0	0	1	
3	50	20000	4	1	0	0	0	0	
4	67	30000	5	0	0	0	0	0	
5	55	60000	10	0	1	0	0	0	

In [ ]: