1-Understand the data :

- Upload the data and take a look of columns and data types
- Identfy the target Label

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random as rd
import warnings
import plotly.express as px
import plotly.io as pio
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv('salary.csv')
```

Data size:

- how much the data size (columns and rows)
- shape function return the number of columns and rows

In [3]:
```
df.shape
```

Out[3]:
```
(10, 3)
```

- we can see this data set is a small one

Data Preview:

- in this step we want to see how data Looks like
- head() display the first few rows of the dataset
- sample() display random sample of the dataset

In [4]:
```
df.head(7)
```

Out[4]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manger	5	110000
5	Region Manager	6	150000
6	Partner	7	200000

In [5]:
```
df.sample(7)
```

Out[5]:

	Position	Level	Salary
9	CEO	10	1000000
2	Senior Consultant	3	60000
3	Manager	4	80000
1	Junior Consultant	2	50000
5	Region Manager	6	150000
4	Country Manger	5	110000
6	Partner	7	200000

Data Types:

- check the types of the data using info() or dtypes
- info() provide information about dataset
- dtypes return the data type of each column

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Position  10 non-null     object
 1   Level     10 non-null     int64
 2   Salary    10 non-null     int64
dtypes: int64(2), object(1)
memory usage: 372.0+ bytes
```

In [7]:
```
df.dtypes
```

Out[7]:
```
Position    object
Level        int64
Salary       int64
dtype: object
```

Missing Values:

- check if nulls or missing values is exist
- isnull().sum() gives the total number of missing values per column

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Position    0
Level       0
Salary      0
dtype: int64
```

- no null values of missing values are exist

Stastical Overview

- obtain stastical measure of the data
- describe() gives stastical measure of each column

In [9]:
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
Level	10.0	5.5	3.027650	1.0	3.25	5.5	7.75	10.0
Salary	10.0	216500.0	285054.088435	45000.0	65000.00	130000.0	215000.00	1000000.0

Duplicated Data

- check for dublicated values and remove it
- duplicated().sum() check for duplicated values

In [10]:
```
df.duplicated().sum()
```

Out[10]: 0

- No duplicated values

Exploring Diversity:

- see how many unique values in the dataset
- nunique() return number of unique values

In [11]:
```
print(df['Level '].nunique())
print(df['Salary'].nunique())
```

```
10
10
```

Correlation Analysis:

- check the Correlation between features and target Label
- corr() calculate the Correlation matrix

In [12]:
```
num_cols=df.select_dtypes('number')
corr_matrix=num_cols.corr()
corr_matrix
```

Out[12]:

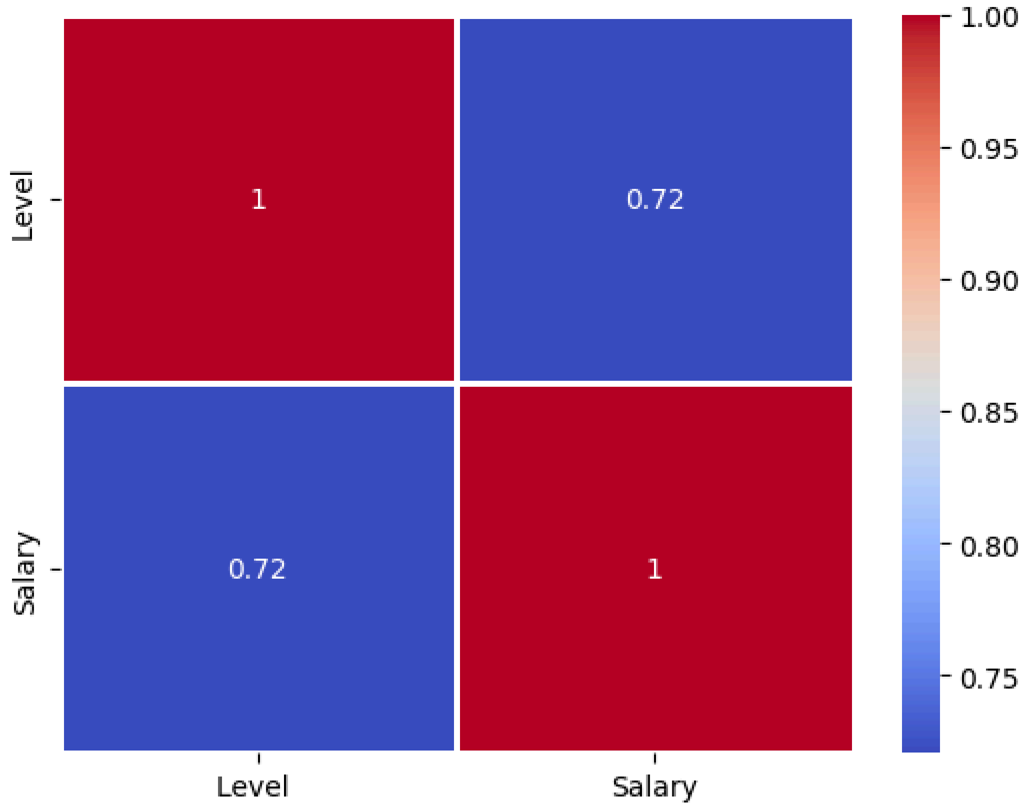
	Level	Salary
Level	1.00000	0.72064
Salary	0.72064	1.00000

Heatmap:

- heatmap can show the corr matrix as visluzation

In [13]:
```
sns.heatmap(corr_matrix,annot=True,linewidths=2,cmap='coolwarm')
```

Out[13]: <Axes: >



2-Visulaization :

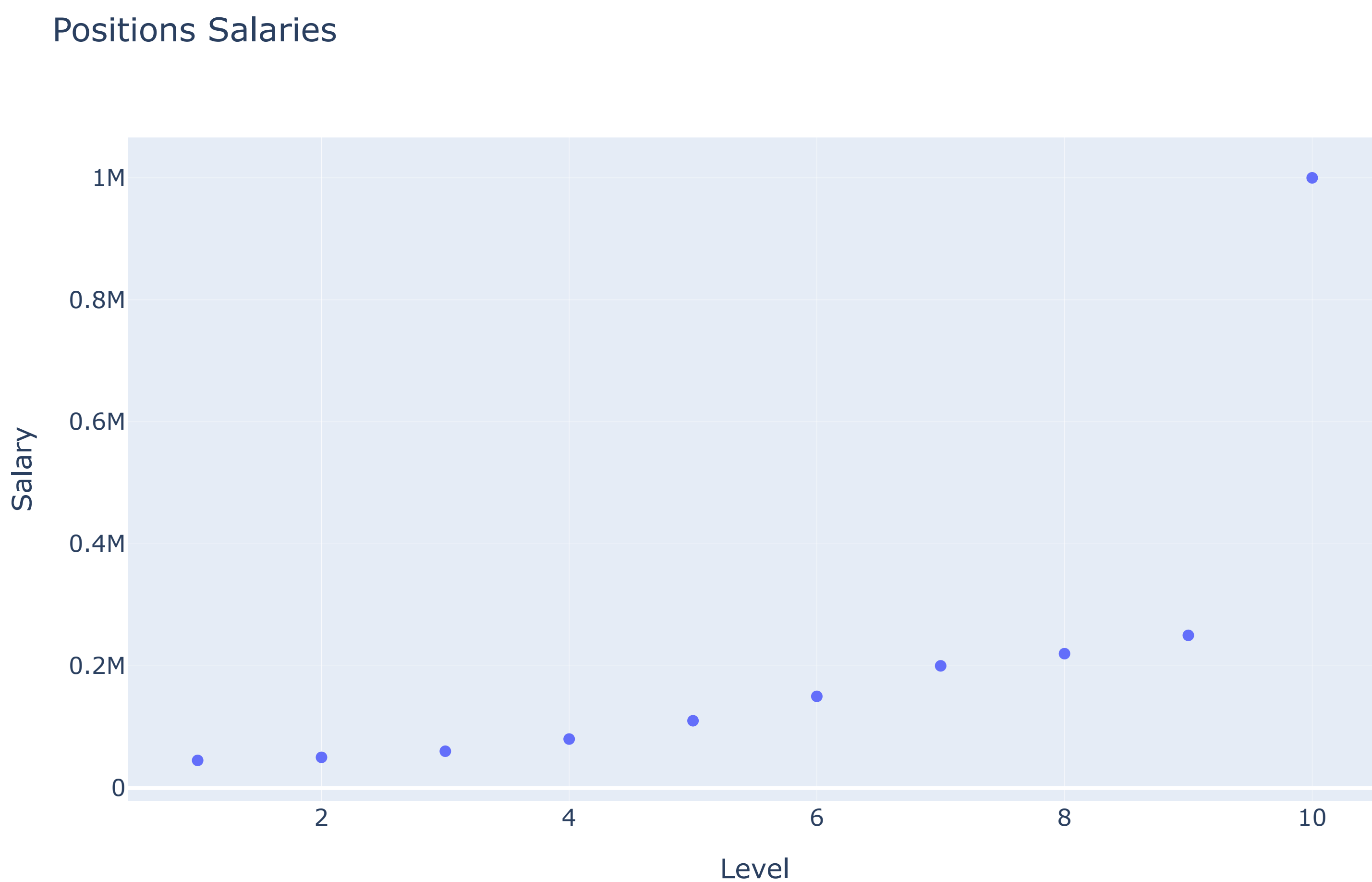
- Visualization allows us to quickly grasp complex data by presenting it in a visual format, making it easier to identify patterns, trends, and outliers that may not be apparent in raw data

Numerical Data:

1-Scatter Diagram

- Visualize Relationships Between Features and Target Label:

In [14]:
```
px.scatter(data_frame=df,y='Salary',x='Level ',hover_name='Position',title='Positions Salaries')
```

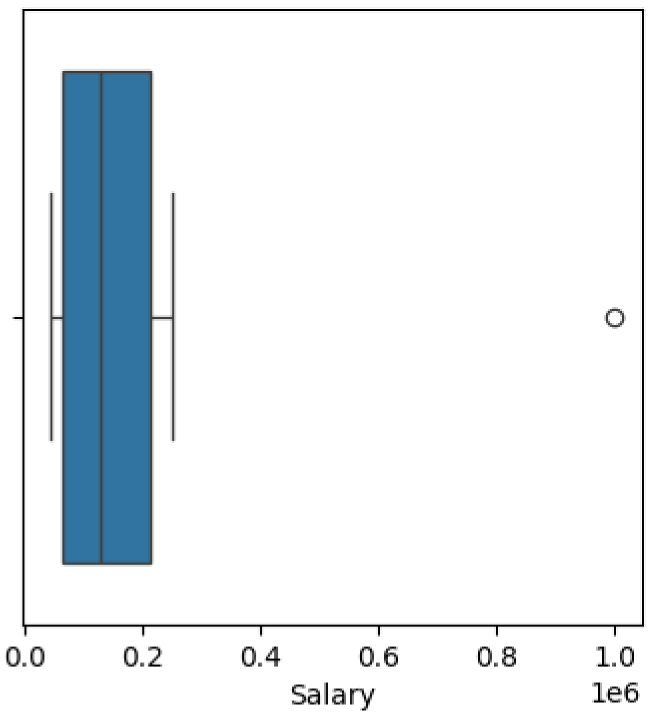


- We can notice there is outlier in the upper limit this could increase the error
- there is a big diffrence in salary between level 9 and level 10
- level 9 salary = 250k , level10=1M
- we can double check by drawing box plot

2-boxplot

- display five numbers summary (minumum,first quartile,median,third quartile and maximum)

In [15]:
```
plt.figure(figsize=(4,4))
sns.boxplot(df['Salary'],orient='h')
plt.show()
```

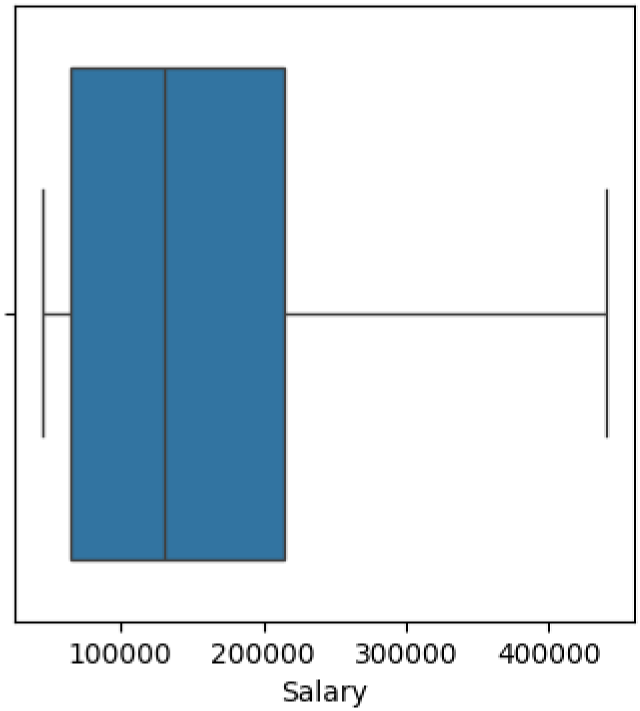


Handle outliers

- the assumption is true and there is ouliter in the upper limit , and we need to remove this outlier.

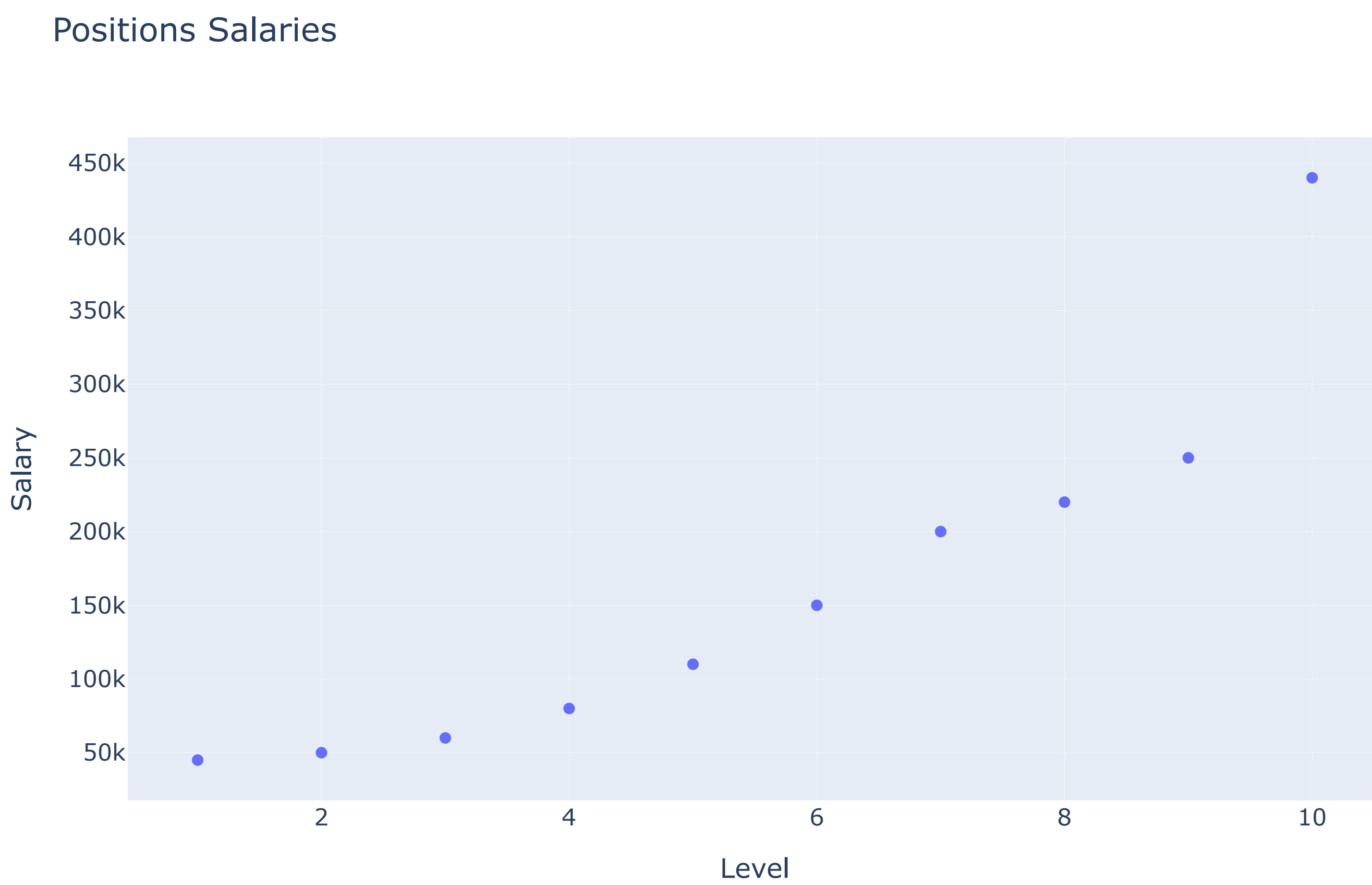
In [16]:
```
Q1 = df['Salary'].quantile(.25)
Q3 = df['Salary'].quantile(.75)
IQR= Q3-Q1
upper_fence= Q3 + 1.5 * IQR
upper_outliers= df[df['Salary'] > upper_fence]['Salary'].values
df['Salary'].replace(upper_outliers, upper_fence, inplace=True)
```

In [17]:
```
plt.figure(figsize=(4,4))
sns.boxplot(df['Salary'],orient='h')
plt.show()
```



- draw scatter digram again to see the diagram after handling the outliers

In [18]:
```
px.scatter(data_frame=df,y='Salary',x='Level ',hover_name='Position',title='Positions Salaries')
```

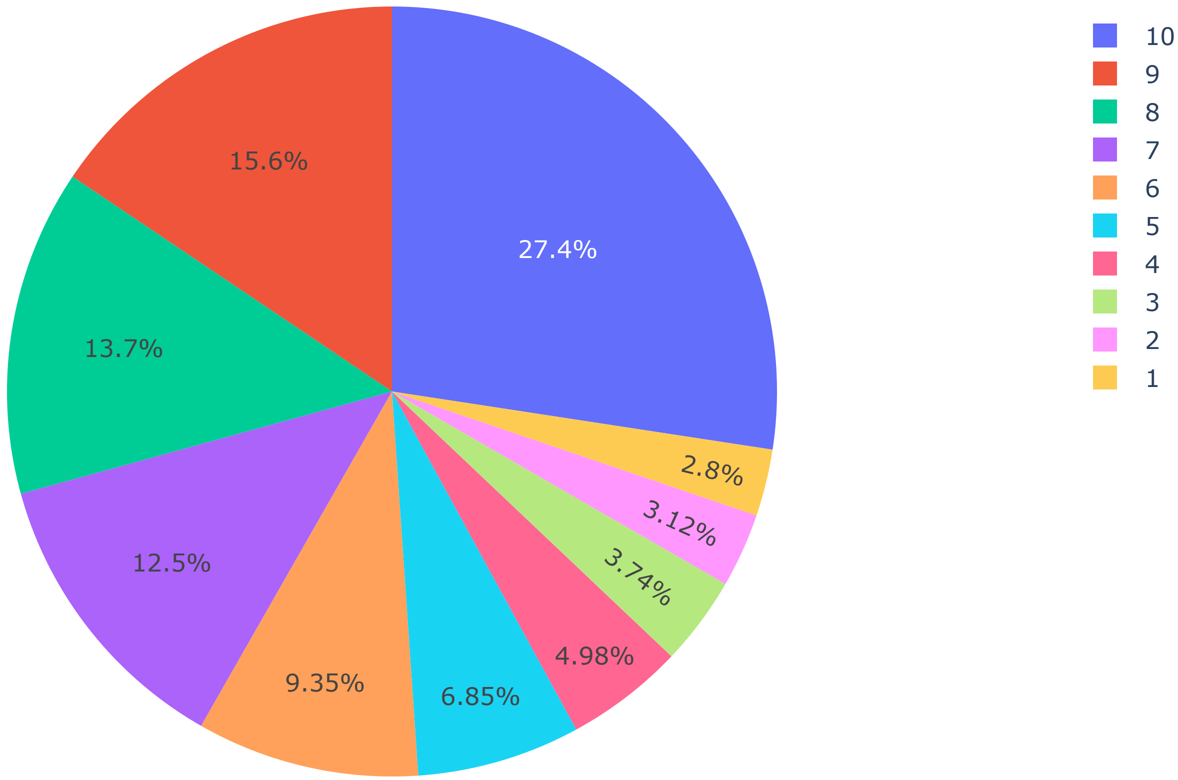


- we can see now :
- level =250k , level10=450k

3- Piechart

- Display the proportion of each category

```
In [19]: px.pie(df,values='Salary', names = 'Level ')
```



3-Feature Engineering

- decide to drop Position column since Level column represnt the position also

```
In [20]: df.drop('Position',axis=1,inplace=True)
```

- check the column Position is reomved using info()

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Level   10 non-null     int64
 1   Salary  10 non-null     int64
dtypes: int64(2)
memory usage: 292.0 bytes
```

4-Normalization

-Scaling Features to a Common Range using MinMax Scaler

In [22]:
```
from sklearn.preprocessing import MinMaxScaler
num_cols=df.select_dtypes('number').columns
scaler=MinMaxScaler()
scaler.fit_transform(df[num_cols])
```

Out[22]:
```
array([[0.        , 0.        ],
       [0.11111111, 0.01265823],
       [0.22222222, 0.03797468],
       [0.33333333, 0.08860759],
       [0.44444444, 0.16455696],
       [0.55555556, 0.26582278],
       [0.66666667, 0.39240506],
       [0.77777778, 0.44303797],
       [0.88888889, 0.51898734],
       [1.        , 1.        ]])
```

5- Split the Data:

- Divide Dataset for Training and Testing:

- x: features , y: target label

In [23]:
```
y=df[['Salary']]
x=df.drop('Salary',axis=1)
```

In [24]:
```
x
```

Out[24]:

	Level
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

In [25]:
```
y
```

Out[25]:

	Salary
0	45000
1	50000
2	60000
3	80000
4	110000
5	150000
6	200000
7	220000
8	250000
9	440000

5-Machine Learning Model :

a-Split the data into train and test models

In [26]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=297)
```

- check the shape of the train and test models:

In [27]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8, 1)
(2, 1)
(8, 1)
(2, 1)
```

b-Apply Polynomial

- using polynimial because the data is relation between the target and the data is not lineary, the accuarcy will be better if using polynomial equation.
- set deegre of the polynomial to 3 to fit all the points and to get the best accuarcy

In [28]:
```
from sklearn.preprocessing import PolynomialFeatures
p = PolynomialFeatures(degree = 3)
x_train_poly = p.fit_transform(x_train.values.reshape(-1,1))
x_test_poly =  p.fit_transform(x_test.values.reshape(-1,1))
```

c-Import Linear Regression Model

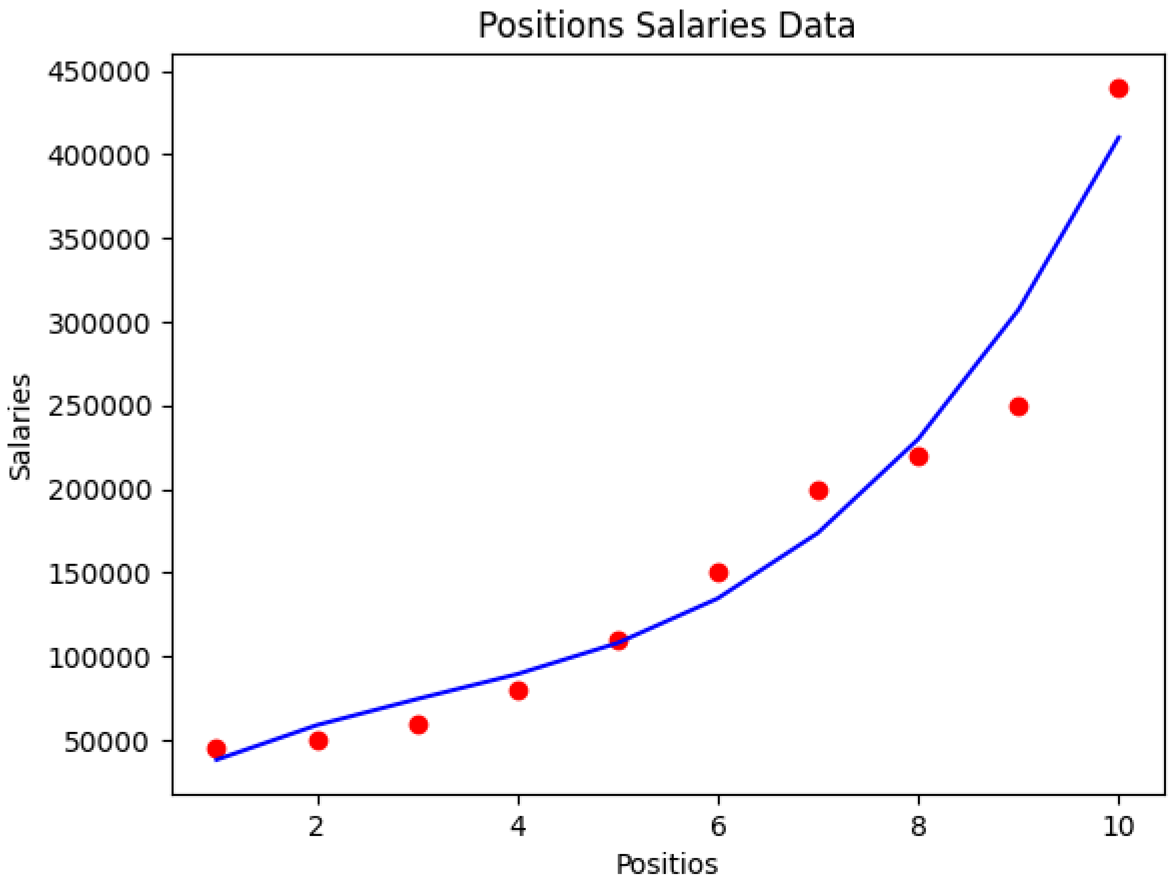
- importing the LR model from sklearn library
- train the data using fit()
- predict the output using predict()

In [29]:
```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train_poly, y_train)
y_test_pred=LR.predict(x_test_poly)
```

d-Testing model accuarcy

- draw a scatter diagram to see how polynomial equation will fit the points
- calculate :
- 1-mean_absolute_error
- 2-mean_squared_error
- 3-Score Matrix

In [30]:
```
plt.scatter(x,y, color = 'red')
plt.plot(x, LR.predict(p.fit_transform(x)), color = 'blue')
plt.title('Positions Salaries Data')
plt.xlabel('Positios')
plt.ylabel('Salaries')
plt.show()
```



In [31]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_test_pred)
print(f"Mean Absolute Error: {mae:.2f}")

mse = mean_squared_error(y_test, y_test_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
r2 = r2_score(y_test, y_test_pred)
print(f"R-squared: {r2:.2f}")
```

```
Mean Absolute Error: 9268.15
Mean Squared Error: 86123504.76
R-squared: 0.99
```

- **Model has accuarcy 99%**