Coding

The IDE we will be using is Jupyter Notebook/ Google Colab (according to your preferences)

1. The first thing we do is import the required modules for the data we will be handling

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score,accuracy_score
```

2. We will load the dataset and read from the given dataset file and print out its data details to understand more about the dataset

```
#Loading the dataset
data=pd.read_csv("file_link/location")
#Data Description
print(data.head())  #Prints out first five row values
print(data.tail())  #Prints out last five row values
print(data.shape)  #Prints out no of rows and columns
print(data.describe())
#Prints out the statistical data such as count,mean etc for data have integer
values
print(data.info())
#Prints out the the column heads, the no of values it has stored along with
the data type present
print(data.nunique())  #Prints out the amount of data that is unique in each
column
```

```
      remote_ratio company_location company_size
19770          100               US            L
19771          100               US            L
19772          100               US            S
19773          100               US            L
19774           50               IN            L

(19775, 11)

          work_year        salary  salary_in_usd  remote_ratio
count  19775.000000  1.977500e+04   19775.000000  19775.000000
mean    2023.353527  1.628728e+05  150935.295322     29.886220
std        0.712468  3.128112e+05   68561.127186     45.427765
min     2020.000000  1.400000e+04   15000.000000      0.000000
25%     2023.000000  1.039770e+05  103200.000000      0.000000
50%     2023.000000  1.430000e+05  142200.000000      0.000000
75%     2024.000000  1.900000e+05  189650.000000    100.000000
max     2024.000000  3.040000e+07  800000.000000    100.000000

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19775 entries, 0 to 19774
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   work_year           19775 non-null  int64
 1   experience_level    19775 non-null  object
 2   employment_type     19775 non-null  object
 3   job_title           19775 non-null  object
 4   salary              19775 non-null  int64
 5   salary_currency     19775 non-null  object
 6   salary_in_usd       19775 non-null  int64
 7   employee_residence  19775 non-null  object
 8   remote_ratio        19775 non-null  int64
 9   company_location    19775 non-null  object
 10  company_size        19775 non-null  object
dtypes: int64(4), object(7)
memory usage: 1.7+ MB
None

work_year                5
experience_level         4
employment_type          4
job_title              148
salary                2925
```

```
salary                 [illegible]
salary_currency          24
salary_in_usd          3319
employee_residence       88
remote_ratio              3
company_location         78
company_size              3
dtype: int64
```

3. Now , we will perform data cleaning by handling missing values, encoding categorical variables, and removing duplicates. It fills missing values in numeric columns with their mean, encodes categorical columns using label encoding, and checks for any duplicated entries in the dataset.

```
#Data Cleaning
print(data.isnull().sum(),"\n") #Prints out missing values of data
numeric_columns = data.select_dtypes(include=[np.number]).columns
non_numeric_columns = data.select_dtypes(exclude=[np.number]).columns
data[numeric_columns] =
data[numeric_columns].fillna(data[numeric_columns].mean())
# Fill numeric columns with their mean
print(data.duplicated().sum(),"\n") #Prints out duplicated values as a sum
categorical_columns = data.select_dtypes(include=[object]).columns
le = LabelEncoder()
for col in categorical_columns:
data[col] = le.fit_transform(data[col])
# Encodes the categorical variables by using LabelEncoder
```

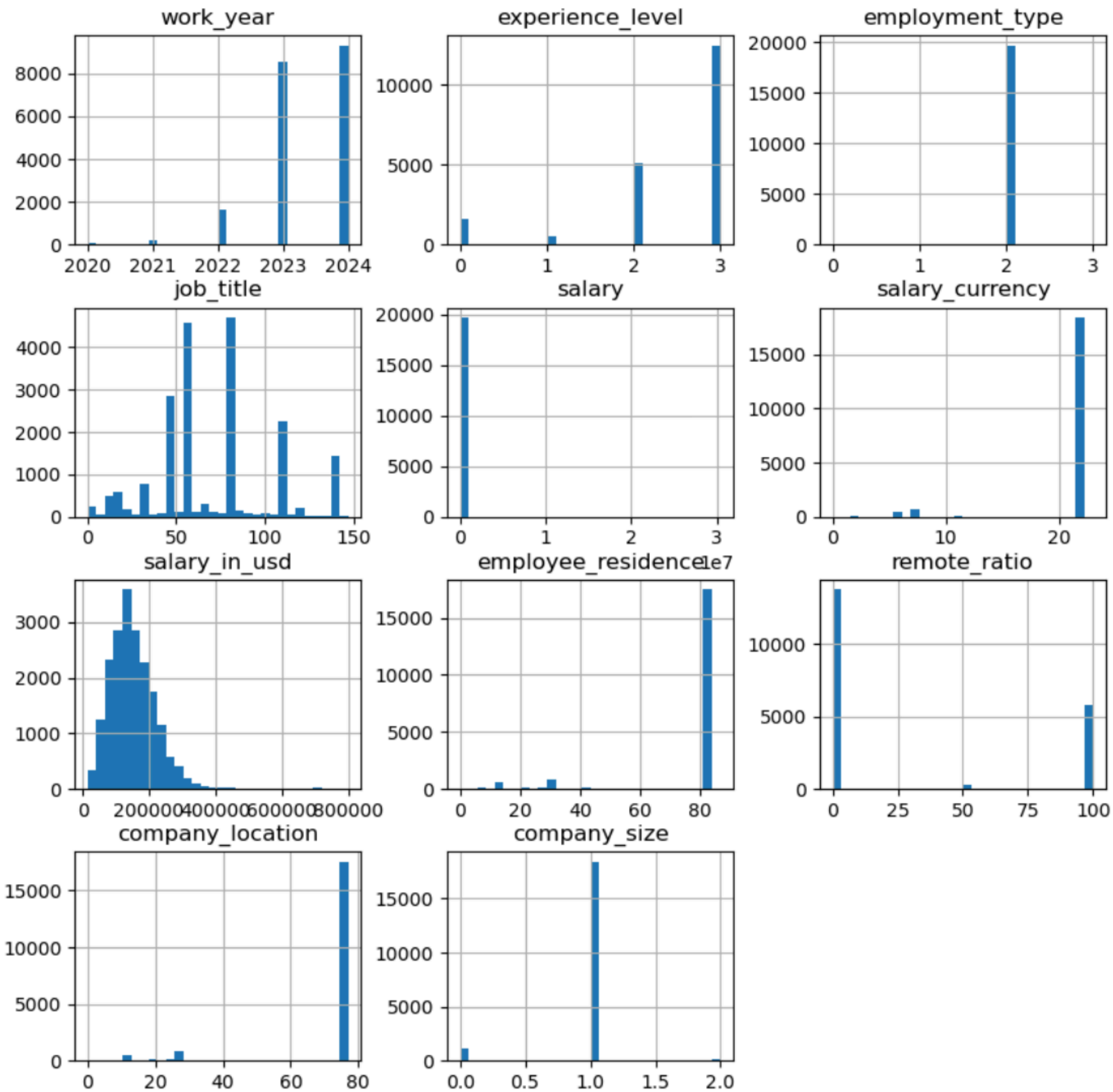
[Scikit-Learn]-> To understand what a categorical variable is.

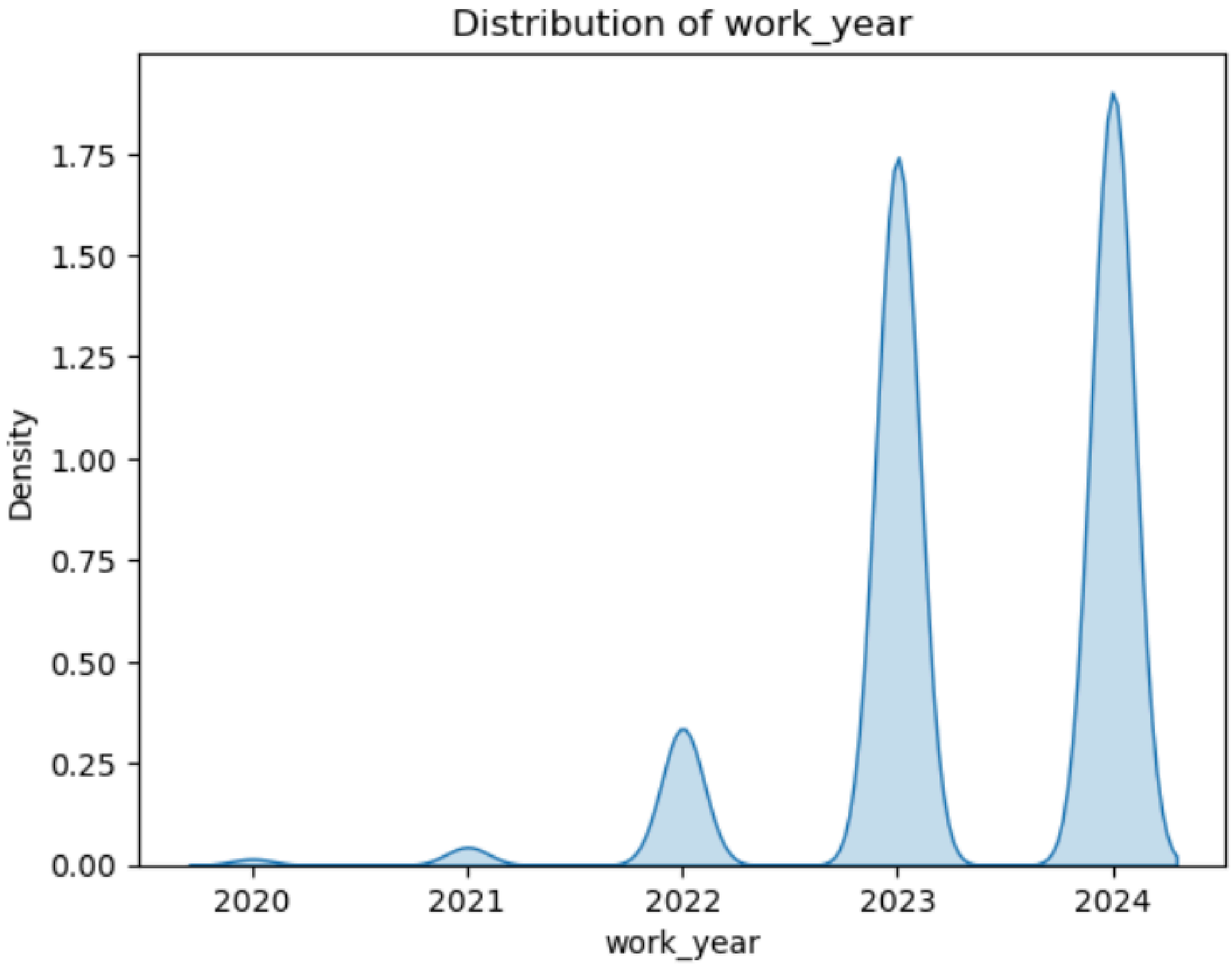
```
work_year             0
experience_level      0
employment_type       0
job_title             0
salary                0
salary_currency       0
salary_in_usd         0
employee_residence    0
remote_ratio          0
company_location      0
company_size          0
dtype: int64

8261
```

4. We now visualize the data to help us get a better understanding of the dataset

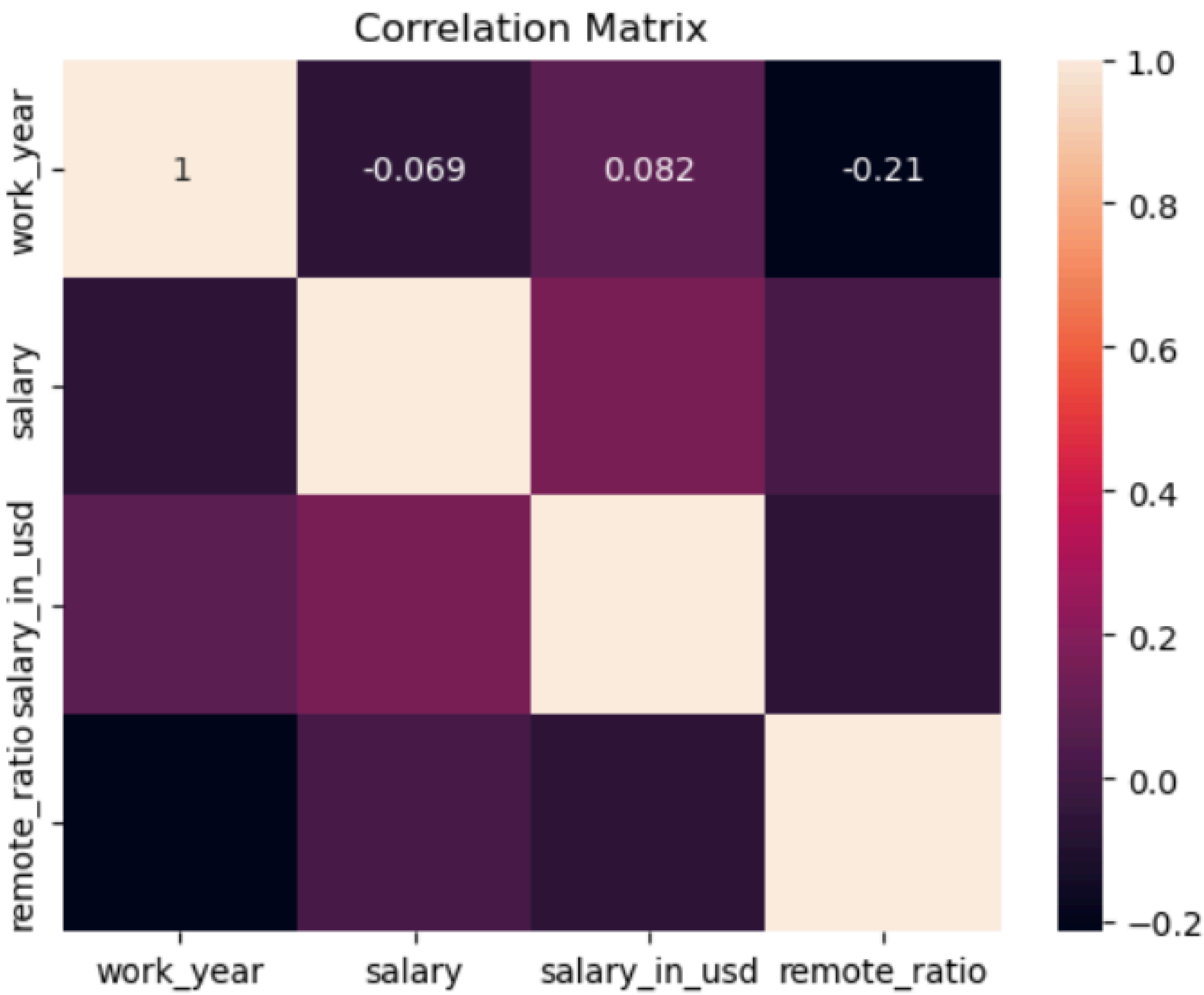
```
#Data Visualization
data.hist(bins=30, figsize=(10, 10))
plt.show()  #Prints out histogram of data
for column in numeric_columns:
    sns.kdeplot(data[column], shade=True)
    plt.title(f'Distribution of {column}')
    plt.show()  #Prints out KDE Plots
numeric_data = data[numeric_columns]
correlation_matrix = numeric_data.corr()
sns.heatmap(correlation_matrix,xticklabels=correlation_matrix.columns,
yticklabels=correlation_matrix.columns,annot=True)
plt.title('Correlation Matrix')
plt.show()  #Prints out a heatmap by using correlation matrix
```

Scikit-Learn -> To understand what correlation matrix is and its function





KDE Plot ->KDE Plot described as ****Kernel Density Estimate**** is used for visualizing the Probability Density of a continuous variable. It depicts the density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization. It provides a smoothed representation of the underlying distribution of a dataset.



5. We will not making testing data sets and preprocessing it to test our models before standardizing any excess data. We fit the models using train_test_split and then check if they are placed into 1D arrays/2D arrays to properly work with the model.

```
#Data Testing and preprocessing
X=data.drop('recent_review_count',axis=1)
#This is the feature matrix and it contains all the independent variables
needed for model training, excluding the target variable.
y=data['recent_review_count']
#This is the target variable which will be used as dependent variable for the
machine learning model.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=1/3,random_state=
0)
sc_X=StandardScaler()
#Used to standardize data by making mean=0 and standard deviation=1
```

```
X_train=sc_X.fit_transform(X_train) #Method used to do the standardization
X_test=sc_X.transform(X_test)  #Secondary method
print(X_train,"\n") #Used to print out the standardized data
print(X_test)
```

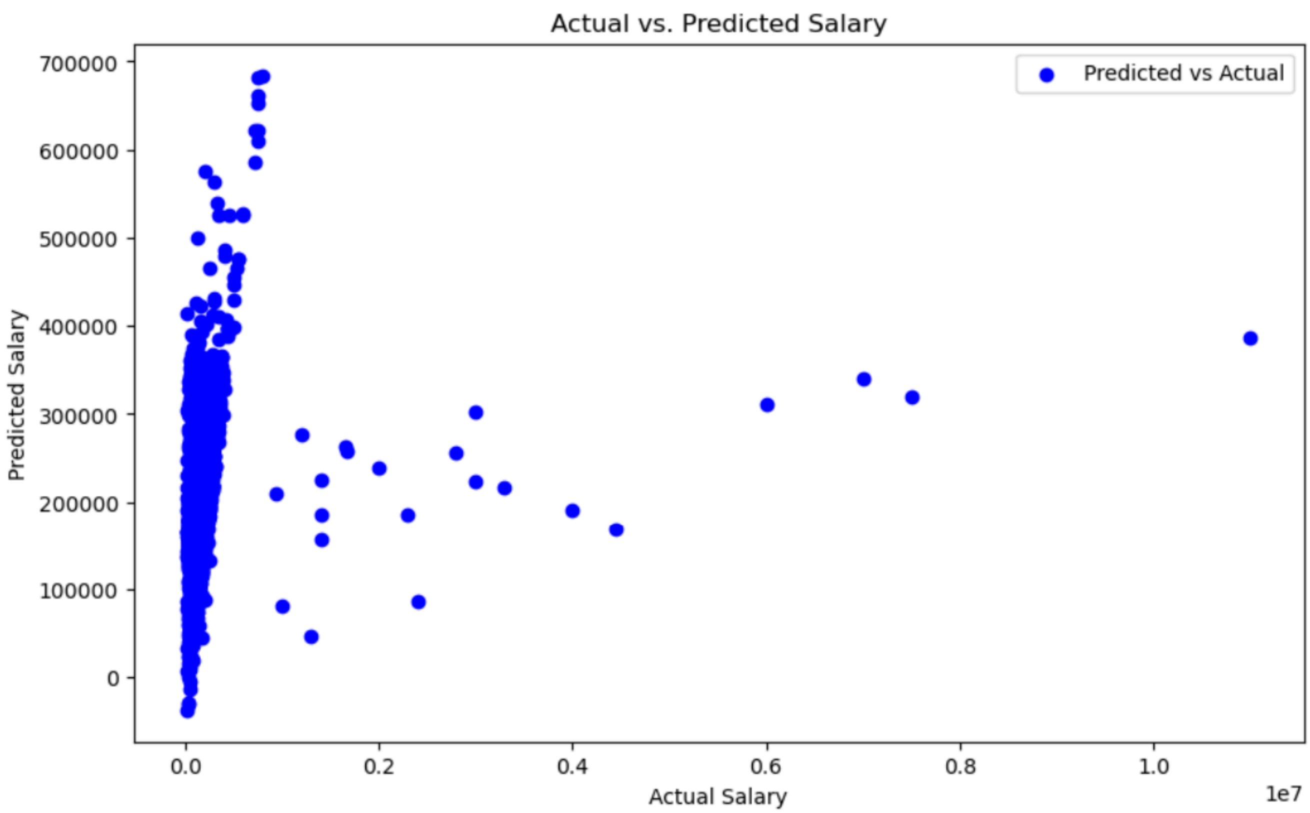
```
[[-0.49541184  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [ 0.90549212  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [-0.49541184  0.63352107  0.01494919 ... -0.65460647 -2.54784655
   0.19425164]
 ...
 [-0.49541184 -0.47914989  0.01494919 ...  1.55247207  0.34064834
   0.19425164]
 [-0.49541184 -2.70449181  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [ 0.90549212  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]]

[[-0.49541184 -1.59182085  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [-0.49541184  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [-0.49541184  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 ...
 [-1.89631579 -1.59182085  0.01494919 ...  1.55247207  0.34064834
   0.19425164]
 [ 0.90549212  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]
 [-0.49541184  0.63352107  0.01494919 ... -0.65460647  0.34064834
   0.19425164]]
```

6. Now we will training the model based on the training sets and thus evaluating it.

```
# Training the model
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction on the test set
y_pred_linR = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred_linR)
r2 = r2_score(y_test, y_pred_linR)
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)

# Visualization - Actual vs. Predicted
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_linR, color='blue', label='Predicted vs Actual')
plt.xlabel('Actual Salary')
plt.ylabel('Predicted Salary')
plt.title('Actual vs. Predicted Salary')
plt.legend()
plt.show()
```

```
Mean Squared Error: 52714727347.57942
R-squared Score: 0.07584134406064946
```



You can see the scatter plot and u can see based on the MSE and R^2 score that this model is well defined but can be improved if we use another models like Logistic Regression.
How to improve LinearRegression Model?
[Improvements]