MINIPROJECT:

SALARY PREDICTION

TEAM MEMBERS:

FAHEEM B

VIPIN CHANDRAN

INDRODUCTION

The salary dataset is a comprehensive collection of information on 6704 employees , who are doing different jobs in a company with different salaries. This dataset has been curated to find the salaries of the employees.

OBJECTIVE

To develop a machine learning model for predicting and analyzing the salary of a new employee based on his age , experience and qualification

PROCEDURE

The machine learning process involves several keys steps, from defining the problem and deploying the model.

1. Define the problem

2. Collecting data

3. Exploratory data analysis

4. Choose a model

5. Train a model

6. Evaluate the model

7. Interpret the result

8. Deploy the model

IMPORTING LIBRARIES

```
In [167]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn.preprocessing import LabelEncoder
          from sklearn.preprocessing import StandardScaler
          from sklearn.model_selection import train_test_split
          from sklearn.linear_model import LinearRegression,Ridge,Lasso
          from sklearn.tree import DecisionTreeRegressor,plot_tree
          from sklearn.neighbors import KNeighborsRegressor
          from sklearn.model_selection import cross_val_score,KFold
          from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
```

1. Import pandas as pd: import the pandas library with the alias pd
2. Import numpy as np: import the numpy library with the alias np
3. Import matplotlib.pyplot as plt: **`matplotlib`** is a popular plotting library that allows you to create a wide variety of static, animated, and interactive plots in Python.
4. import seaborn as sns: Seaborn is a statistical data visualization library based on **matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics.
5. from sklearn.preprocessing import LabelEncoder: The **`LabelEncoder`** is a utility class in scikit-learn that can be used to encode categorical labels with numerical values. This is particularly useful when working with machine learning algorithms that require numerical inputs.
6. The **`StandardScaler`** is a part of scikit-learn and is commonly used for standardizing features by removing the mean and scaling to unit variance. Standardization is an essential step in preprocessing data for machine learning models, especially when features have different scales.
7. Train_test_split function is commonly used for splitting a dataset into training and testing sets, which is a fundamental step in machine learning model development. The training set is used to train the model, while the testing set is used to evaluate its performance on unseen data.
8. : **`LinearRegression`**, **`Ridge`**, and **`Lasso`**. These are different linear regression algorithms that can be used for modeling relationships between dependent and independent variables in a linear fashion.
9. importing the **`DecisionTreeRegressor`** and **`plot_tree`** from scikit-learn. These are components of scikit-learn's decision tree implementation, which is used for both classification and regression tasks.
10. importing the **`KNeighborsRegressor`** from scikit-learn. This class is used for k-nearest neighbors (KNN) regression, a type of instance-based or memory-based learning where predictions are made based on the majority of the k-nearest neighbors in the feature space.

11. importing the **`cross_val_score`** and **`KFold`** classes from scikit-learn. These are useful tools for performing cross-validation, a technique commonly used to assess the performance of a machine learning model and to mitigate issues related to data splitting.

Read data

1. data=pd.read_csv(r"C:\Users\Admin\Desktop\Salary_Data.csv")
print(data)

```
In [168]: data=pd.read_csv(r"C:\Users\Admin\Desktop\Salary_Data.csv")
          print(data)
```

```
       Age  Gender    Education Level                 Job Title  \
0     32.0    Male         Bachelor's         Software Engineer
1     28.0  Female           Master's              Data Analyst
2     45.0    Male                PhD            Senior Manager
3     36.0  Female         Bachelor's           Sales Associate
4     52.0    Male           Master's                  Director
...    ...     ...                ...                       ...
6699  49.0  Female                PhD  Director of Marketing
6700  32.0    Male        High School           Sales Associate
6701  30.0  Female  Bachelor's Degree         Financial Manager
6702  46.0    Male    Master's Degree         Marketing Manager
6703  26.0  Female        High School           Sales Executive

      Years of Experience    Salary
0                     5.0   90000.0
1                     3.0   65000.0
2                    15.0  150000.0
3                     7.0   60000.0
4                    20.0  200000.0
...                   ...       ...
6699                 20.0  200000.0
6700                  3.0   50000.0
6701                  4.0   55000.0
6702                 14.0  140000.0
6703                  1.0   35000.0

[6704 rows x 6 columns]
```

read the data in the file in a particular location of the system and display the contents of the csv file.

2. data.info()

```
In [169]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6704 entries, 0 to 6703
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  6702 non-null   float64
 1   Gender               6702 non-null   object
 2   Education Level      6701 non-null   object
 3   Job Title            6702 non-null   object
 4   Years of Experience  6701 non-null   float64
 5   Salary               6699 non-null   float64
dtypes: float64(3), object(3)
memory usage: 314.4+ KB
```

The **`info()`** method in pandas provides a concise summary of a DataFrame, including information about the data types, non-null values, and memory usage.

3. print(data[data.duplicated()].shape)

```
In [171]: data1=data.drop_duplicates()
          data1
```

- **`data.duplicated()`** returns a boolean Series indicating whether each row is a duplicate of a previous row.
- **`data[data.duplicated()]`** filters the DataFrame to only include the duplicated rows.
- **`.shape`** is then used to get the number of rows and columns in the resulting DataFrame.

4. data1=data.drop_duplicates()
data1

- **`drop_duplicates`** method to remove duplicate rows from your DataFrame and storing the result in a new DataFrame called **`data1`**.

5. print(data1.isnull().sum())

```
In [172]: print(data1.isnull().sum())

Age                    1
Gender                 1
Education Level        2
Job Title              1
Years of Experience    2
Salary                 4
dtype: int64
```

```
In [174]: print(data2.isnull().sum())

Age                    0
Gender                 0
Education Level        0
Job Title              0
Years of Experience    0
Salary                 0
dtype: int64
```

- The code **`data1.isnull().sum()`** is checking for the presence of null (missing) values in each column of the DataFrame **`data1`** and then summing up the number of null values for each column.
- **`data1.isnull()`** returns a DataFrame of the same shape as **`data1`**, where each entry is a boolean value indicating whether the corresponding element is null.
- **`.sum()`** is then used to sum these boolean values along each column, resulting in the count of null values for each column

6. data2=data1.dropna()
   print(data2)
   - The code **`data2 = data1.dropna()`** is creating a new DataFrame **`data2`** by removing rows with any missing values (NaN) from the original DataFrame **`data1`**.
   - **`data1.dropna()`** returns a new DataFrame with any rows containing NaN values removed.
   - Printing **`data2`** will show you the DataFrame without the rows containing missing values.
7. print(data2.shape)

```
In [175]: print(data2)

       Age  Gender    Education Level                 Job Title  \
0     32.0    Male          Bachelor's         Software Engineer
1     28.0  Female            Master's              Data Analyst
2     45.0    Male                 PhD            Senior Manager
3     36.0  Female          Bachelor's           Sales Associate
4     52.0    Male            Master's                  Director
...    ...     ...                 ...                       ...
6623  43.0  Female    Master's Degree  Digital Marketing Manager
6624  27.0    Male        High School              Sales Manager
6625  33.0  Female  Bachelor's Degree      Director of Marketing
6628  37.0    Male  Bachelor's Degree             Sales Director
6631  30.0  Female  Bachelor's Degree              Sales Manager

      Years of Experience    Salary
0                     5.0   90000.0
1                     3.0   65000.0
2                    15.0  150000.0
3                     7.0   60000.0
4                    20.0  200000.0
...                   ...       ...
6623                 15.0  150000.0
6624                  2.0   40000.0
6625                  8.0   80000.0
6628                  7.0   90000.0
6631                  5.0   70000.0
```

   - The code **`print(data2.shape)`** is printing the shape of the DataFrame **`data2`**. The **`shape`** attribute of a DataFrame returns a tuple representing its dimensions - the number of rows and columns.

8.

```
Education Level
Bachelor's Degree    506
Master's Degree      446
PhD                  340
Bachelor's           262
Master's             122
High School          110
phD                    1
Name: count, dtype: int64
Axes(0.125,0.11;0.775x0.77)
```

```
In [179]: replace_dict = {'phD': 'PhD',"Bachelor's Degree": "Bachelor's","Master's Degree" : "Master's"}
          data2['Education_Level'] = data2['Education Level'].replace(replace_dict)
```

```
In [242]: print(data2['Education Level'].value_counts())

Education Level
Bachelor's     768
Master's       568
PhD            341
High School    110
Name: count, dtype: int64
```

```
replace_dict = {'phD': 'PhD',"Bachelor's Degree":
"Bachelor's","Master's Degree" : "Master's"}
data2['Education Level'] = data2['Education
Level'].replace(replace_dict)
```

- the code is replacing values in the 'Education Level' column of the DataFrame **`data2`** using the **`replace`** method. It is replacing certain education level values with their standardized forms.
- **`replace_dict`** is a dictionary where keys are the values to be replaced, and values are the replacement values.
- **`data2['Education Level'].replace(replace_dict)`** is used to replace values in the 'Education Level' column of **`data2`** based on the dictionary.

9. `print(data2['Education Level'].value_counts())`
   `print(data2)`

- The code **`print(data2['Education Level'].value_counts())`** is printing the counts of unique values in the 'Education Level' column of the DataFrame **`data2`**. The **`value_counts()`** method is useful for understanding the distribution of different education levels in the dataset.
- **`data2['Education Level']`** extracts the 'Education Level' column from the DataFrame **`data2`**.
- **`.value_counts()`** then counts the occurrences of each unique value in the column.

- The output will be a count of how many times each unique education level appears in the 'Education Level' column.

10. `print(data2["Job Title"].unique()`

```
Job Title
Software Engineer Manager    127
Full Stack Engineer          122
Senior Software Engineer      96
Senior Project Engineer       95
Back end Developer            81
                            ...
Financial Advisor              1
Junior Designer                1
Chief Technology Officer       1
Technical Recruiter            1
Delivery Driver                1
```

- The **`unique()`** method is used to obtain an array of unique values in a particular column.
- **`data2["Job Title"]`** extracts the 'Job Title' column from the DataFrame **`data2`**.
- **`.unique()`** then returns an array containing the unique values in that column

11. `print(data2['Job Title'].value_counts()[:11])`

```
In [185]: print(data2['Job Title'].value_counts()[:11])

Job Title
Software Engineer Manager    127
Full Stack Engineer          122
Senior Software Engineer      96
Senior Project Engineer       95
Back end Developer            81
Data Scientist                80
Software Engineer             78
Front end Developer           71
Marketing Manager             55
Product Manager               53
Data Analyst                  51
Name: count, dtype: int64
```

- The code **`print(data2['Job Title'].value_counts()[:11])`** is printing the top 11 most frequently occurring values in the 'Job Title' column of the DataFrame **`data2`**. The **`value_counts()`** method counts the occurrences of each unique value, and **`[:11]`** is used to select the top 11 values.

```
In [187]: data3= ['Software Engineer Manager', 'Full Stack Engineer', 'Senior Project Engineer', 'Senior Software Engineer', 'Data Scie
          data4 = data2[data2['Job Title'].isin(data3)]
          print(data4.info())

<class 'pandas.core.frame.DataFrame'>
Index: 909 entries, 0 to 6618
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  909 non-null    float64
 1   Gender               909 non-null    object
 2   Education Level      909 non-null    object
 3   Job Title            909 non-null    object
 4   Years of Experience  909 non-null    float64
 5   Salary               909 non-null    float64
 6   Education_Level      909 non-null    object
dtypes: float64(3), object(4)
memory usage: 56.8+ KB
None
```

- **`data2['Job Title']`** extracts the 'Job Title' column from the DataFrame **`data2`**
- **`.value_counts()`** then counts the occurrences of each unique value in that column.
- **`[:11]`** is used to select the top 11 values.

12.
```
data3= ['Software Engineer Manager', 'Full Stack Engineer', 'Senior Project Engineer', 'Senior Software Engineer', 'Data Scientist', 'Back end Developer', 'Software Engineer', 'Front end Developer', 'Marketing Manager', 'Product Manager', 'Data Analyst']
data4 = data2[data2['Job Title'].isin(data3)]
print(data4.info())
```

```
print(data4)
```

In [192]:
```
d_col=['Gender','Education Level','Job Title']
data5=data4.drop(columns=d_col)
print(data5.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1780 entries, 0 to 6631
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  1780 non-null   float64
 1   Years of Experience  1780 non-null   float64
 2   Salary               1780 non-null   float64
 3   Education_Level      1780 non-null   object
 4   Gender_Encode        1780 non-null   int32
 5   qualification_Encode 1780 non-null   int32
 6   JobTitle_Encode      1780 non-null   int32
dtypes: float64(3), int32(3), object(1)
memory usage: 90.4+ KB
None
```

- The code is creating a new DataFrame **`data4`** by selecting rows from **`data2`** where the 'Job Title' column matches any of the job titles in the list **`data3`**. Then, it prints information about the **`data4`** DataFrame using the **`info()`** method.
- **`data2['Job Title'].isin(data3)`** creates a boolean mask indicating whether each job title in the 'Job Title' column of **`data2`** is in the **`data3`** list.
- **`data2[data2['Job Title'].isin(data3)]`** selects rows from **`data2`** where the job title is in the **`data3`** list, creating a new DataFrame **`data4`**.
- **`print(data4.info())`** prints information about the **`data4`** DataFrame using the **`info()`** method.

13.
```
label_encoder = LabelEncoder()
data4['Gender_Encode'] =
label_encoder.fit_transform(data4['Gender'])
print(data4[['Gender', 'Gender_Encode']])
data4['qualification_Encode'] =
label_encoder.fit_transform(data4['Education Level'])
print(data4[['Education Level', 'qualification_Encode']])
data4['JobTitle_Encode'] =
label_encoder.fit_transform(data4['Job Title'])
print(data4[['Job Title', 'JobTitle_Encode']])
print(data4.head())
data4
```

- we are using **`LabelEncoder`** from scikit-learn to encode categorical variables in your DataFrame **`data4`**. The **`fit_transform`** method of **`LabelEncoder`** is applied to encode the 'Gender', 'Education Level', and 'Job Title' columns.
- For each categorical variable ('Gender', 'Education Level', 'Job Title'), you are using **`LabelEncoder`** to transform the original categorical values into numerical labels.
- The encoded values are added as new columns with names like 'Gender_Encode', 'qualification_Encode', and 'JobTitle_Encode'.
- The **`print(data4.head())`** statement displays the first few rows of the DataFrame to check the encoding results.

14.
```
d_col=['Gender','Education Level','Job Title']
data5=data4.drop(columns=d_col)
```

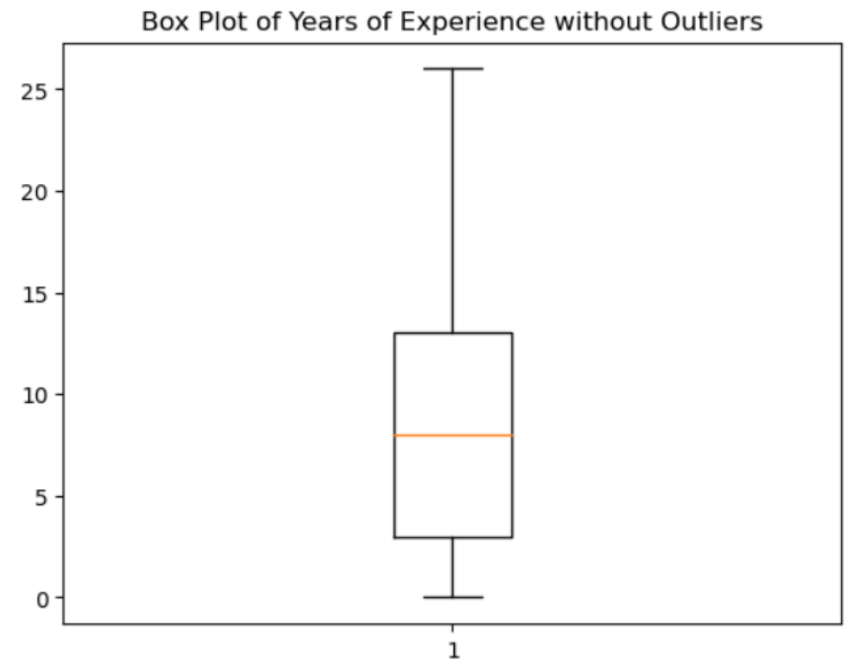
print(data5.info())

```
In [192]: d_col=['Gender','Education Level','Job Title']
          data5=data4.drop(columns=d_col)
          print(data5.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1780 entries, 0 to 6631
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Age                  1780 non-null   float64
 1   Years of Experience  1780 non-null   float64
 2   Salary               1780 non-null   float64
 3   Education_Level      1780 non-null   object 
 4   Gender_Encode        1780 non-null   int32  
 5   qualification_Encode  1780 non-null  int32  
 6   JobTitle_Encode      1780 non-null   int32  
dtypes: float64(3), int32(3), object(1)
memory usage: 90.4+ KB
None
```

-
-

```
count    1748.000000
mean        8.752860
std         6.261943
min         0.000000
25%         3.000000
50%         8.000000
75%        13.000000
max        26.000000
Name: Years of Experience, dtype: float64
```



-
-

In [193]:
```
print(data5)
```

```
       Age  Years of Experience    Salary Education_Level  Gender_Encode  \
0     32.0                  5.0   90000.0      Bachelor's              1   
1     28.0                  3.0   65000.0        Master's              0   
2     45.0                 15.0  150000.0             PhD              1   
3     36.0                  7.0   60000.0      Bachelor's              0   
4     52.0                 20.0  200000.0        Master's              1   
...    ...                  ...       ...             ...            ...   
6623  43.0                 15.0  150000.0        Master's              0   
6624  27.0                  2.0   40000.0     High School              1   
6625  33.0                  8.0   80000.0      Bachelor's              0   
6628  37.0                  7.0   90000.0      Bachelor's              1   
6631  30.0                  5.0   70000.0      Bachelor's              0   

      qualification_Encode  JobTitle_Encode  
0                        0              175  
1                        2               18  
2                        3              144  
3                        0              115  
4                        2               25  
...                    ...              ...  
6623                     2               23  
6624                     1              118  
6625                     0               33  
6628                     0              116  
6631                     0              118  

[1780 rows x 7 columns]
```

- The code is dropping columns with names specified in the list **`d_col`** from the DataFrame **`data4`** and creating a new DataFrame **`data5`**. The **`drop`** method is used to remove the specified columns.
- **`data4.drop(columns=d_col)`** drops the columns specified in **`d_col`** from the DataFrame **`data4`** and creates a new DataFrame **`data5`**.
- **`print(data5.info())`** prints information about the DataFrame **`data5`** using the **`info()`** method.
- The output of **`data5.info()`** will provide details about the structure of the **`data5`** DataFrame, including the number of non-null values in each remaining column, data types, and memory usage.

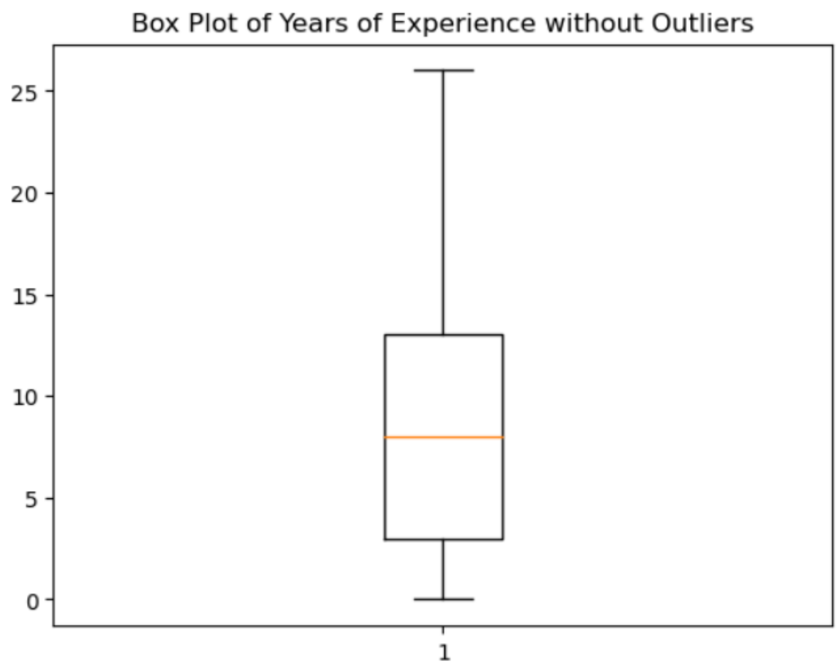
15.
```
print(data5['Years of Experience'].describe())
plt.boxplot(data5)
plt.title('Box Plot of Fitered Data')
plt.show()
```

- we're using **`matplotlib`** to create a box plot for the columns in your DataFrame **`data5`**. However, it's important to note that creating a box plot for all columns at once may not be visually informative if the data types and scales of the columns vary significantly.
- **`print(data5['Years of Experience'].describe())`** prints summary statistics for the 'Years of Experience' column, such as mean, standard deviation, minimum, maximum, and quartiles.
- **`plt.boxplot(data5)`** creates a box plot for all columns in the DataFrame **`data5`**.
- **`plt.title('Box Plot of Filtered Data')`** sets the title of the plot.
- **`plt.show()`** displays the plot.

16.

```
outlier_column2 = 'Years of Experience'
lower_bound = 0
upper_bound = 26
data5 = data5[(data5[outlier_column2] >= lower_bound) &
(data5[outlier_column2] <= upper_bound)]
print(data5[outlier_column2].describe())
plt.boxplot(data5[outlier_column2].dropna())
plt.title(f'Box Plot of {outlier_column2} without Outliers')
plt.show()
```

```
count    1748.000000
mean        8.752860
std         6.261943
min         0.000000
25%         3.000000
50%         8.000000
75%        13.000000
max        26.000000
Name: Years of Experience, dtype: float64
```

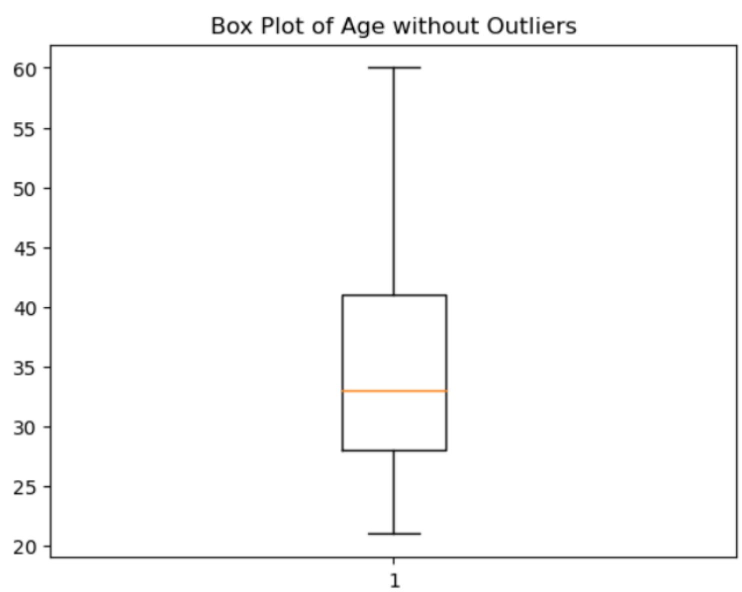


- handling outliers in the 'Years of Experience' column of your DataFrame **`data5`**. The code you provided filters out the rows where the 'Years of Experience' values fall outside a specified range, and then creates a box plot without outliers.
- **`data5[(data5[outlier_column2] >= lower_bound) & (data5[outlier_column2] <= upper_bound)]`** filters out rows where 'Years of Experience' is outside the specified range.
- **`print(data5[outlier_column2].describe())`** prints summary statistics for the filtered 'Years of Experience'.

- `plt.boxplot(data5[outlier_column2].dropna())` creates a box plot for 'Years of Experience' without outliers.
- `plt.title(f'Box Plot of {outlier_column2} without Outliers')` sets the title of the plot.
- This approach helps visualize the distribution of 'Years of Experience' while excluding outliers. Adjust the `lower_bound` and `upper_bound` as needed based on your criteria for considering values as outliers.

17.
```
outlier_column1 = 'Age'
lower_bound = 0
upper_bound = 60
data5 = data5[(data5[outlier_column1] >= lower_bound) &
(data5[outlier_column1] <= upper_bound)]
print(data5[outlier_column1].describe())
plt.boxplot(data5[outlier_column1].dropna())
plt.title(f'Box Plot of {outlier_column1} without Outliers')
plt.show()
```

```
count    1745.000000
mean       34.722636
std         7.747660
min        21.000000
25%        28.000000
50%        33.000000
75%        41.000000
max        60.000000
Name: Age, dtype: float64
```

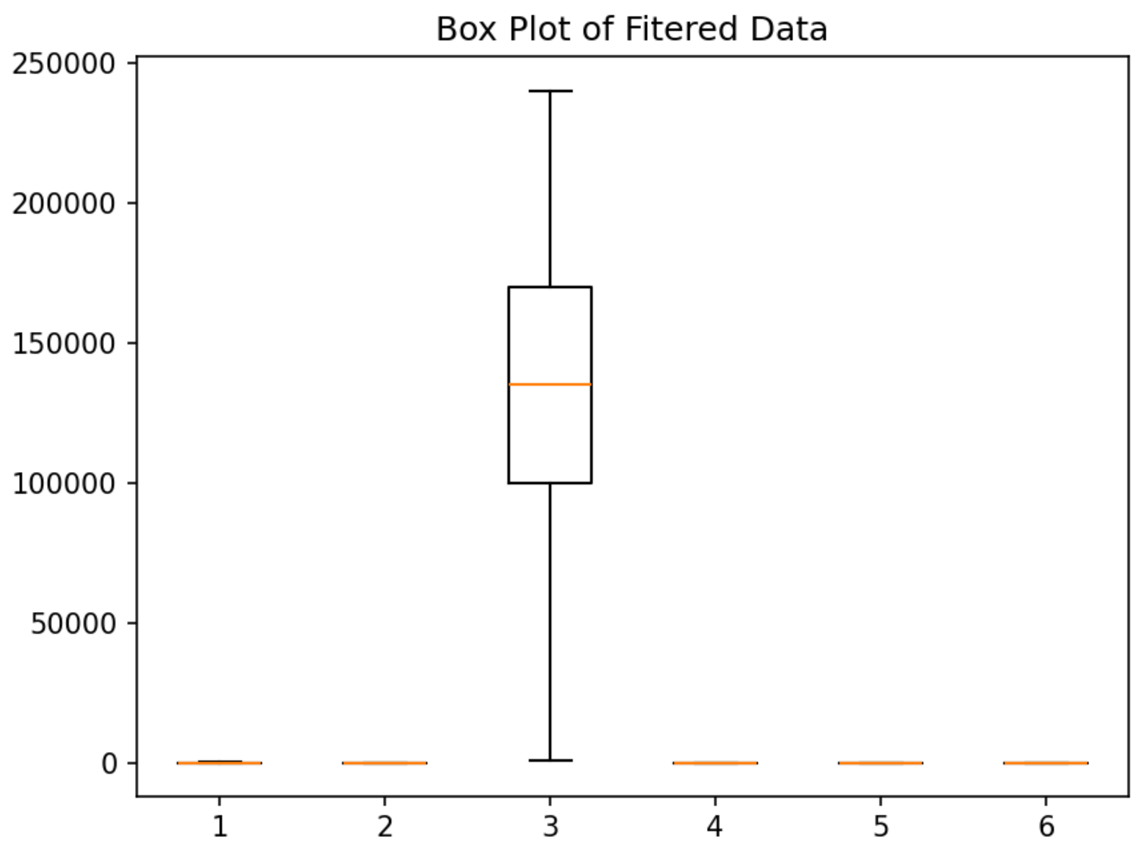


- you are applying a similar approach to handle outliers in the 'Age' column of your DataFrame `data5`. The code filters out the rows where the 'Age' values fall outside a specified range and then creates a box plot without outliers.

- `data5[(data5[outlier_column1] >= lower_bound) & (data5[outlier_column1] <= upper_bound)]` filters out rows where 'Age' is outside the specified range.
- `print(data5[outlier_column1].describe())` prints summary statistics for the filtered 'Age'.
- `plt.boxplot(data5[outlier_column1].dropna())` creates a box plot for 'Age' without outliers.
- `plt.title(f'Box Plot of {outlier_column1} without Outliers')` sets the title of the plot.

- This approach helps visualize the distribution of 'Age' while excluding outliers. Adjust the `lower_bound` and `upper_bound` as needed based on your criteria for considering values as outliers.

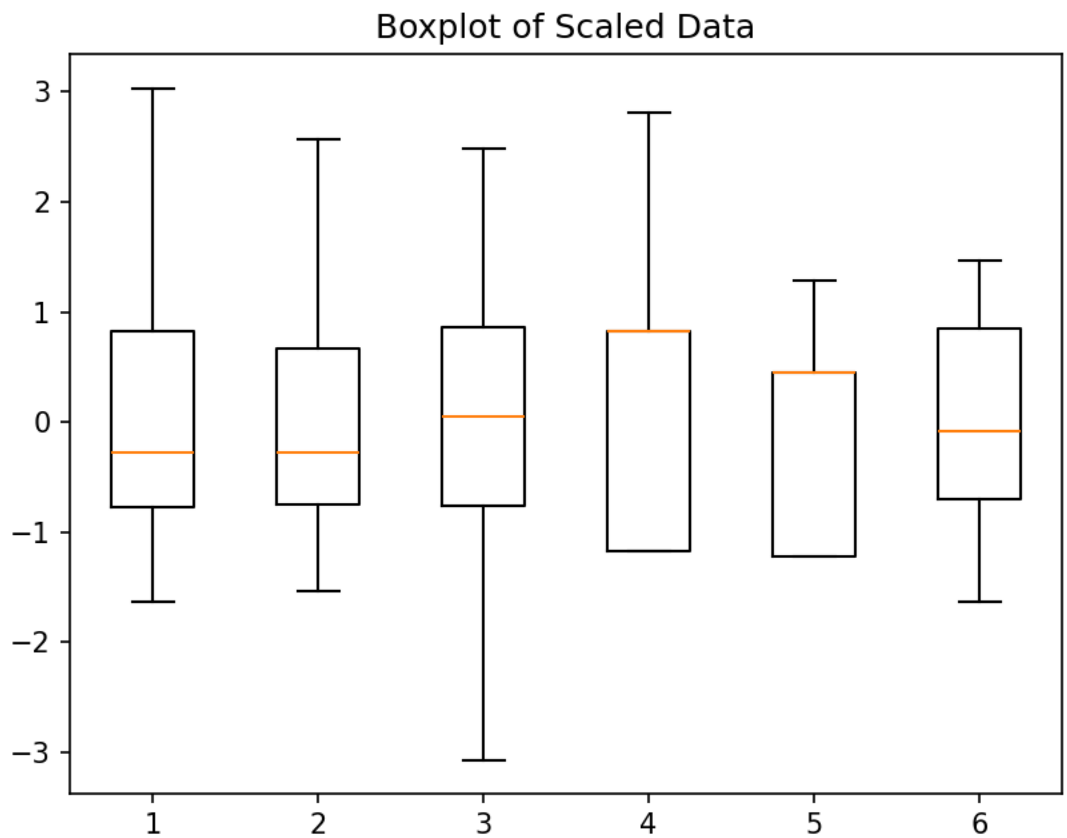
18.
```
print(data5.describe())
plt.boxplot(data5)
plt.title('Box Plot of Fitered Data')
plt.show()
```



- we are attempting to create a box plot for all columns in your DataFrame `data5` after applying filtering to handle outliers. However, creating a box plot for all columns at once might not provide a clear and meaningful visualization if the columns have different scales and data types.

- `print(data5.describe())` prints summary statistics for all numeric columns in the DataFrame `data5`.
- `plt.boxplot(data5)` attempts to create a box plot for all columns in `data5`.
- `plt.title('Box Plot of Filtered Data')` sets the title of the plot.
- `plt.show()` displays the plot.

19.
```
scaler=StandardScaler()
data5_scaled=scaler.fit_transform(data5)
plt.boxplot(data5_scaled)
plt.title('Boxplot of Scaled Data')
plt.show()
```



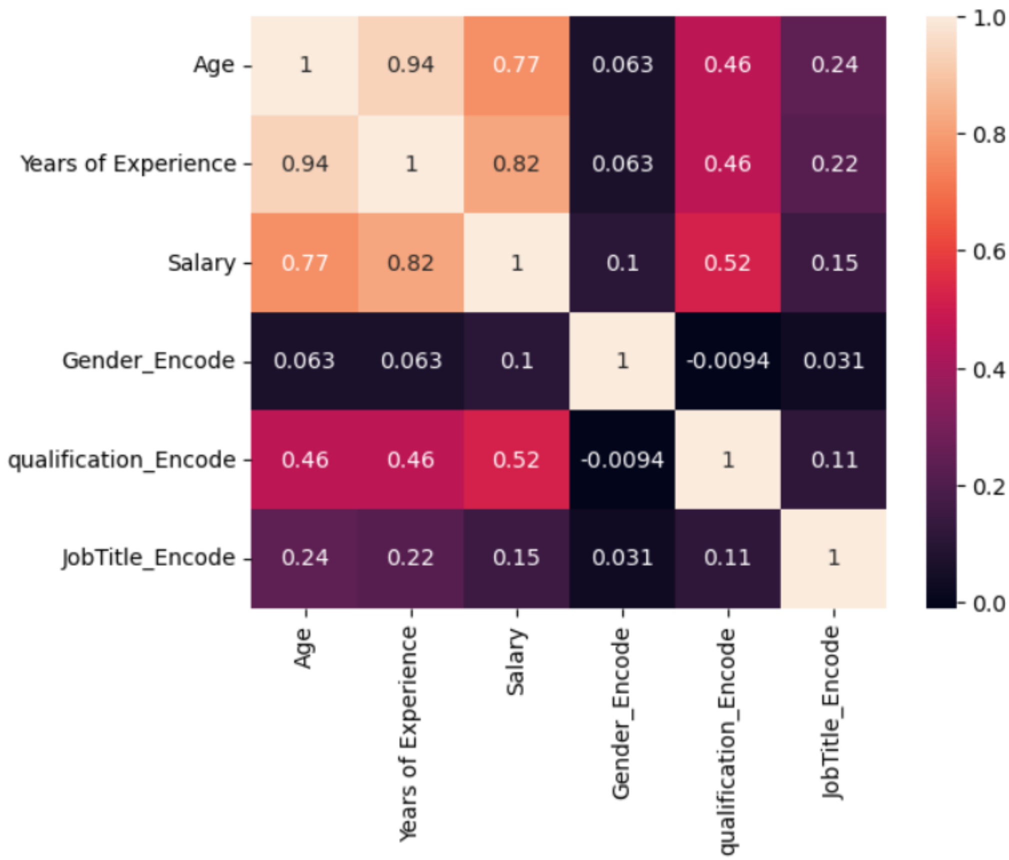
- we are using `StandardScaler` from scikit-learn to scale the numeric columns in your DataFrame `data5`. After scaling, you are creating a box plot to visualize the distribution of the scaled data.
- `scaler = StandardScaler()` initializes the StandardScaler.
- `data5_scaled = scaler.fit_transform(data5)` scales the numeric columns in `data5`.

- `plt.boxplot(data5_scaled)` creates a box plot for the scaled data.
- `plt.title('Boxplot of Scaled Data')` sets the title of the plot.
- `plt.show()` displays the plot.

20.
```
corr=data5[['Age','Years of Experience','Salary','Gender_Encode','qualification_Encode','JobTitle_Encode']].corr()
print(corr)
sns.heatmap(corr,annot=True)
plt.show()
```

```
                          Age  Years of Experience    Salary  Gender_Encode  \
Age                  1.000000             0.936040  0.767024       0.062529
Years of Experience  0.936040             1.000000  0.820616       0.062581
Salary               0.767024             0.820616  1.000000       0.103880
Gender_Encode        0.062529             0.062581  0.103880       1.000000
qualification_Encode 0.461906             0.462581  0.521614      -0.009422
JobTitle_Encode      0.237586             0.218004  0.151214       0.031360

                     qualification_Encode  JobTitle_Encode
Age                              0.461906         0.237586
Years of Experience              0.462581         0.218004
Salary                           0.521614         0.151214
Gender_Encode                   -0.009422         0.031360
qualification_Encode             1.000000         0.112931
JobTitle_Encode                  0.112931         1.000000
```



-

-
- code calculates the correlation matrix for a subset of columns in the DataFrame **data5**, which includes 'Age', 'Years of Experience', 'Salary', 'Gender_Encode', 'qualification_Encode', and 'JobTitle_Encode'. It then creates a heatmap using Seaborn to visualize the correlation values.
- `data5[['Age', 'Years of Experience', 'Salary', 'Gender_Encode', 'qualification_Encode', 'JobTitle_Encode']]` selects a subset of columns from `data5`.
- `.corr()` calculates the correlation matrix for the selected columns.
- `print(corr)` prints the correlation matrix to the console.
- `sns.heatmap(corr, annot=True)` creates a heatmap using Seaborn to visualize the correlation values with annotations.
- `plt.show()` displays the heatmap.
- The heatmap provides a visual representation of the correlation between the selected variables. The values in the cells of the heatmap represent the correlation coefficients, and the color intensity indicates the strength and direction of the correlation.

21. X

```
In [234]: x=data5[['Age','Years of Experience']]
          y=data5[['Salary']]
          print(x)
          print(y)
```

we are defining feature variables (**x**) and the target variable (**y**) for a machine learning model. The features include 'Age', 'Years of Experience', and 'qualification_Encode', while the target variable is 'Salary'.

- `x = data5[['Age', 'Years of Experience', 'qualification_Encode']]` selects the feature variables (independent variables) from the DataFrame `data5`. The features are 'Age', 'Years of Experience', and 'qualification_Encode'.
- `y = data5[['Salary']]` selects the target variable (dependent variable) from the DataFrame `data5`. The target variable is 'Salary'.

- Printing `x` and `y` will display the selected feature variables and the target variable, respectively.
- If you're planning to use these variables for a machine learning model, you can proceed with further steps such as splitting the data into training and testing sets, selecting a machine learning algorithm, training the model, and evaluating its performance.

22.
```
x1_scaled=scaler.fit_transform(x)
print(x1_scaled.shape)
```

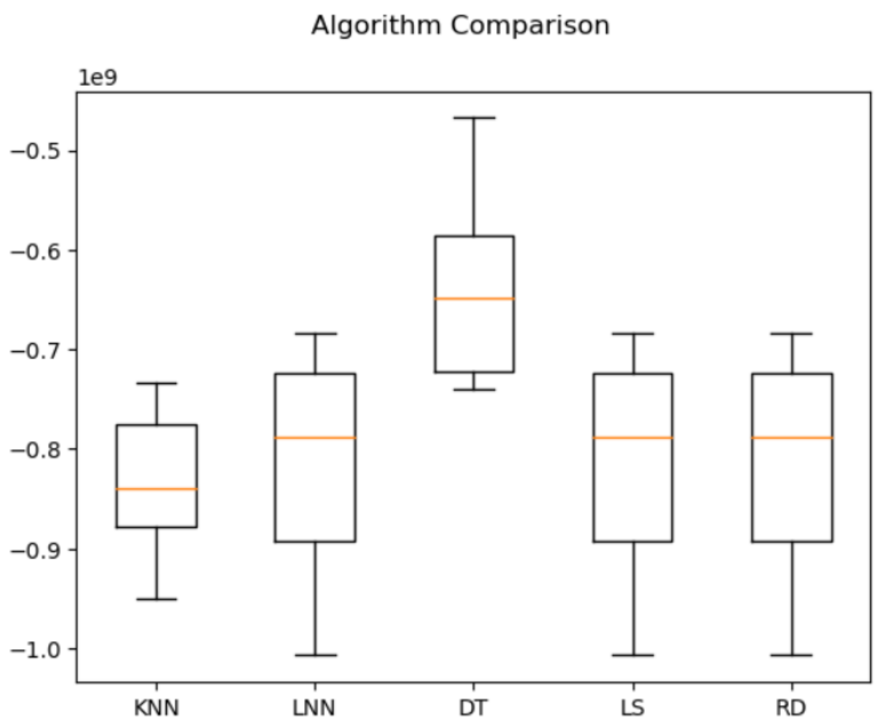
- we are using the `StandardScaler` to scale the feature variables `x` (Age, Years of Experience, and qualification_Encode). The `fit_transform` method is used to scale the features, and you are printing the shape of the resulting scaled array.

- `scaler.fit_transform(x)` scales the feature variables in `x`.
- `x1_scaled` holds the scaled features.
- `x1_scaled.shape` prints the shape of the scaled array.
- The `shape` attribute of a NumPy array returns a tuple representing the dimensions of the array. In this case, it prints the shape of the scaled feature array.

23. models=[]

```
In [236]: models=[]
models.append(('KNN',KNeighborsRegressor()))
models.append(("LNN",LinearRegression()))
models.append(('DT',DecisionTreeRegressor()))
models.append(('LS',Lasso()))
models.append(('RD',Ridge()))
result1=[]
names=[]
scoring = 'neg_mean_squared_error'
Kfold= KFold(n_splits=10, shuffle=True, random_state=42)
for name, model in models:
    cv_results = cross_val_score(model, x, y, cv=Kfold, scoring=scoring)
    result1.append(cv_results)
    names.append(name)
    print(f"MSE of {name}: {cv_results.mean()}")
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(result1)
ax.set_xticklabels(names)
plt.show()
```

```
MSE of KNN: -832775949.0733341
MSE of LNN: -815848475.7131746
MSE of DT: -639584561.6658901
MSE of LS: -815848355.7532276
MSE of RD: -815847641.7096734
```



-
-
- we are comparing the performance of different regression models using cross-validation and evaluating them based on the negative mean squared error. The models include KNeighborsRegressor, Linear Regression (LNN), Decision Tree Regressor (DT), Lasso (LS), and Ridge (RD).
- `models` is a list containing tuples with the names and instances of different regression models.
- `KFold` is used for cross-validation with 10 folds.
- The loop iterates over each model, performs cross-validation, prints the mean squared error, and appends the results to the `result1` list.
- A box plot is then created to compare the performance of different models.
- this code provides a visual representation of how the models compare in terms of negative mean squared error across the cross-validation folds.

24. x_
25.
26.

In [237]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
Tree_Model= DecisionTreeRegressor(max_depth=2)
print(Tree_Model.fit(x_train, y_train))
result2=Tree_Model.predict(x_test)
print(y)
print(result2)
```

27.

you are splitting your data into training and testing sets using `train_test_split` and then fitting a Decision Tree Regressor model to the training data. Finally, you are making predictions on the test set and printing the actual and predicted values.

- `train_test_split` is used to split your data into training and testing sets.
- The shapes of the training and testing sets are printed to check the dimensions.
- A Decision Tree Regressor model with a maximum depth of 2 is created and fitted to the training data.
- Predictions are made on the test set (`x_test`), and both the actual (`y_test`) and predicted (`result2`) values are printed.

- This code is a typical workflow for training and evaluating a machine learning model. It helps you understand the performance of the Decision Tree Regressor on unseen data.

28. print

```
In [238]: print(mean_squared_error(y_test,result2))
          print(np.sqrt(mean_squared_error(y_test,result2)))
          print(mean_absolute_error(y_test,result2))
          print(r2_score(y_test,result2))

          859783529.5308685
          29322.06557408377
          23077.137184047435
          0.6734427975320072
```

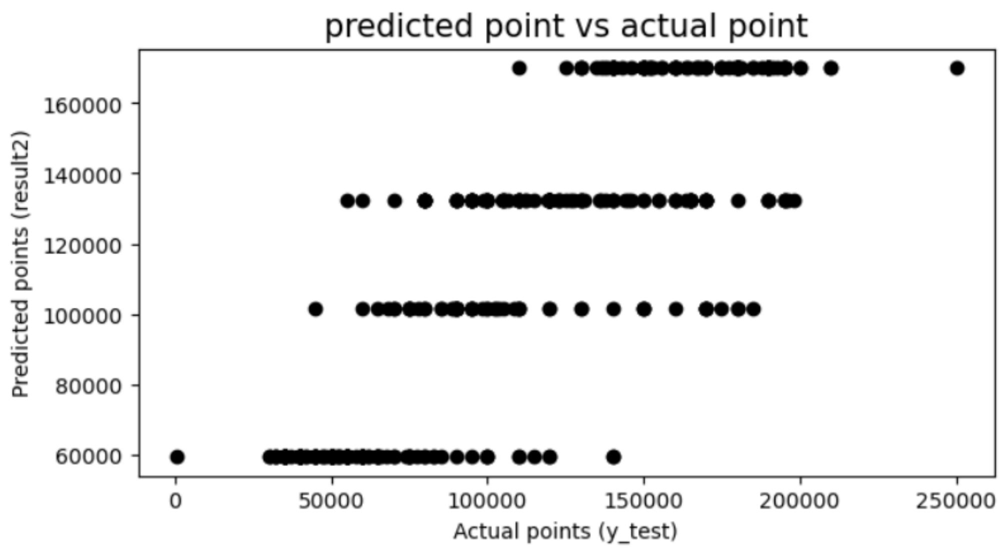
- we are evaluating the performance of your Decision Tree Regressor model on the test set using various regression metrics.

- `mean_squared_error` calculates the mean squared error between the true and predicted values.
- `np.sqrt(mean_squared_error)` calculates the root mean squared error, which is the square root of the mean squared error.
- `mean_absolute_error` calculates the mean absolute error between the true and predicted values.
- `r2_score` calculates the R-squared score, which represents the proportion of the variance in the dependent variable that is predictable from the independent variables.

- these metrics provide insights into different aspects of the model's performance. Lower values for mean squared error, root mean squared error, and mean absolute error indicate better performance, while a higher R-squared score indicates a better fit.

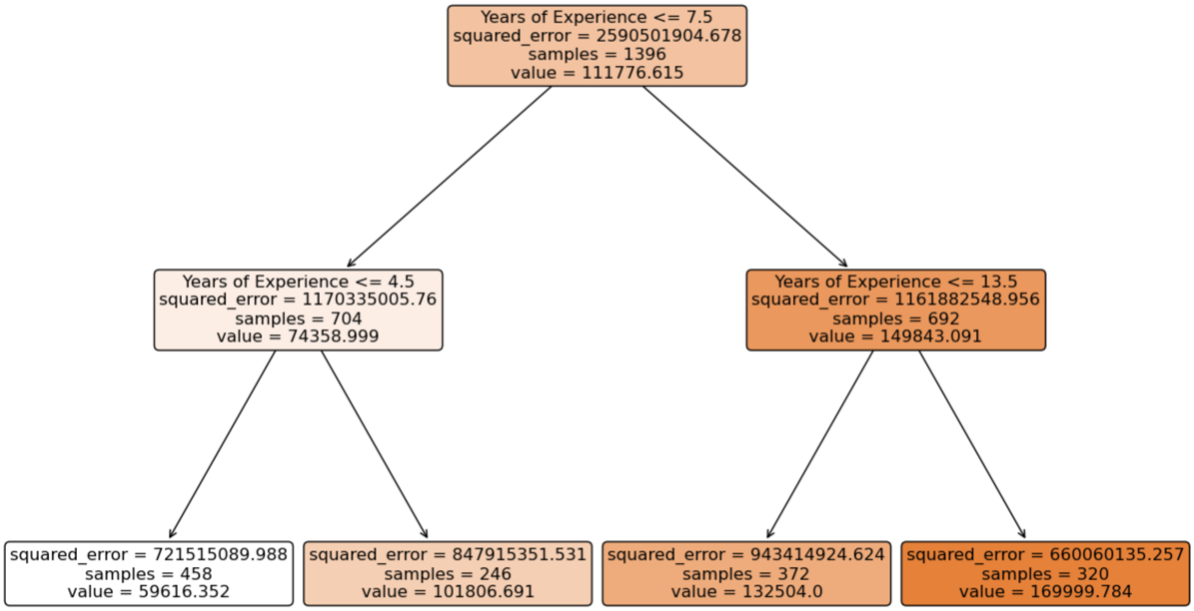
29.
```
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
plt.scatter(x=y_test,y=result2,color='black')
plt.title('predicted point vs actual
point',fontdict={'fontsize':15})
plt.xlabel('Actual points (y_test)',fontdict={'fontsize':10})
plt.ylabel('Predicted points
(result2)',fontdict={'fontsize':10})
plt.show()
print("x=",y_test)
```

```
print("y=",result2)
```



- you are creating a scatter plot to visualize the relationship between the actual values (**`y_test`**) and the predicted values (**`result2`**) from your Decision Tree Regressor model.
- **`plt.scatter`** is used to create a scatter plot where the x-axis represents the actual points (**`y_test`**) and the y-axis represents the predicted points (**`result2`**).
- **`plt.title`**, **`plt.xlabel`**, and **`plt.ylabel`** are used to set the title and axis labels for the plot.

30.
```
    plt.figure(figsize=(15, 10))
plot_tree(Tree_Model, feature_names=x.columns, filled=True,
rounded=True)
plt.show()
```



- we are using **`plot_tree`** from scikit-learn to visualize the decision tree structure of your trained **`Tree_Model`**. The tree diagram is generated with features labeled and filled areas representing different classes or values.
- **`plt.figure(figsize=(15, 10))`** sets the size of the figure.
- **`plot_tree(Tree_Model, feature_names=x.columns, filled=True, rounded=True)`** generates the decision tree plot. **`feature_names`** is set to the column names of your features (**`x`**), and **`filled`** and **`rounded`** control the appearance of the tree nodes.
- This plot allows you to visualize the decision-making process of your trained decision tree model, with each node representing a decision based on a specific feature.

```
 new_data={
 'Age': 30,
 'Years of Experience':3,
 'Job_Title': 9}
new_data_df=pd.DataFrame([new_data])
x_scaled=scaler.transform(new_data_df
predicted_salary=Tree_Model.predict(x_scaled)
print("predicted salary for employee:",predicted_salary)
```

```
predicted salary for employee: [59616.35152838]
```

- you have correctly created a new data point, converted it to a DataFrame (`new_data_df`), scaled the features using the scaler, and then used your trained decision tree model (`Tree_Model`) to predict the salary for the new employee
- his code assumes that your `scaler` object was fitted on the original data (`x`) and your `Tree_Model` is a trained decision tree regression model. The `predicted_salary` variable will contain the predicted salary for the new employee based on the features provided.

CONCLUSION

In conclusion, the analysis of the salary prediction dataset has provided valuable insights into the factors influencing salary levels. Through a comprehensive exploration of various features such as education, experience, and job role, we have identified patterns and trends that contribute to the prediction of salaries.