```python
import pandas as pd
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Path to the dataset file in Google Drive
file_path = '/content/data.csv'

# Reading the dataset file using pandas
data = pd.read_csv(file_path)

# Displaying summary
summary = data.describe()
print("Summary:")
print(summary)

# Displaying header
header = data.head()
print("\nHeader:")
print(header)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Summary:
              Age  Years of Experience         Salary
count  373.000000           373.000000     373.000000
mean    37.431635            10.030831  100577.345845
std      7.069073             6.557007   48240.013482
min     23.000000             0.000000     350.000000
25%     31.000000             4.000000   55000.000000
50%     36.000000             9.000000   95000.000000
75%     44.000000            15.000000  140000.000000
max     53.000000            25.000000  250000.000000

Header:
    Age  Gender Education Level          Job Title  Years of
Experience  \
0  32.0    Male      Bachelor's  Software Engineer
5.0
1  28.0  Female        Master's       Data Analyst
3.0
2  45.0    Male             PhD     Senior Manager
15.0
3  36.0  Female      Bachelor's    Sales Associate
7.0
4  52.0    Male        Master's           Director
20.0

     Salary
```

```
0    90000.0
1    65000.0
2   150000.0
3    60000.0
4   200000.0
```

#**Data Cleaning, preprocessing and transformation**

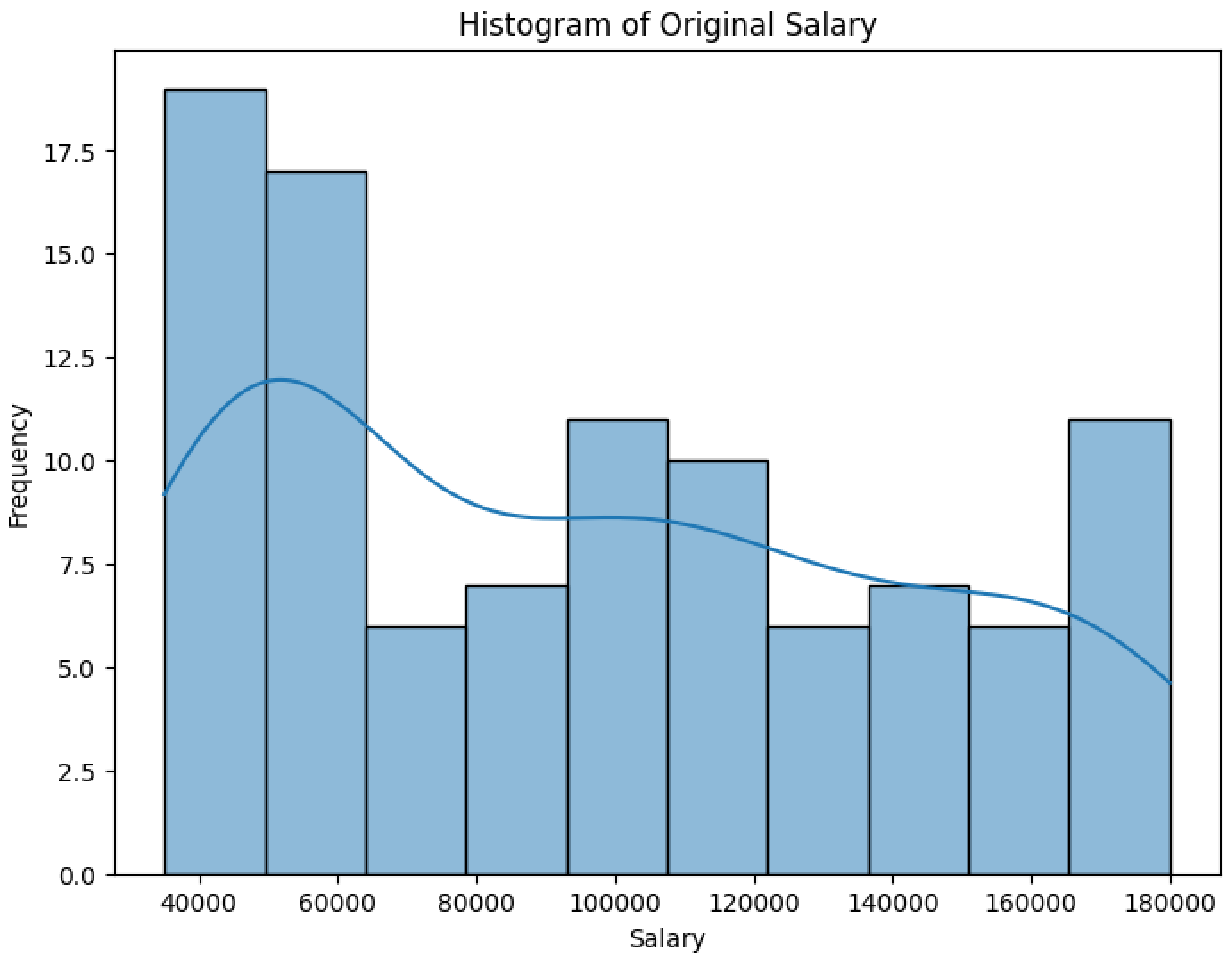
```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Format and clean the data
# Assuming the column names have whitespace, remove leading/trailing
spaces
data.columns = data.columns.str.strip()

# Remove rows with null values
data = data.dropna()

# Remove outliers using z-score
z_scores = np.abs((data['Salary'] - data['Salary'].mean()) /
data['Salary'].std())
data = data[z_scores < 3]

# Sample a subset of the data
sample_size = 100
data_sample = data.sample(n=sample_size, random_state=42)

# Label encode the education level
label_encoder = LabelEncoder()
data_sample['Education Level'] =
label_encoder.fit_transform(data_sample['Education Level'])

# Scale the numerical columns using Min-Max scaling
numeric_columns = ['Age', 'Years of Experience']
scaler = MinMaxScaler()
data_sample[numeric_columns] =
scaler.fit_transform(data_sample[numeric_columns])

# Aggregate the data by job title and calculate average salary
data_aggregated = data_sample.groupby('Job Title')['Salary'].mean()
data_aggregated = data_aggregated.reset_index()

# Merge aggregated data back into the sample data and rename columns
data_sample = pd.merge(data_sample, data_aggregated, on='Job Title',
how='left')
data_sample.rename(columns={'Salary_x': 'original_salary', 'Salary_y':
'average_salary'}, inplace=True)
```

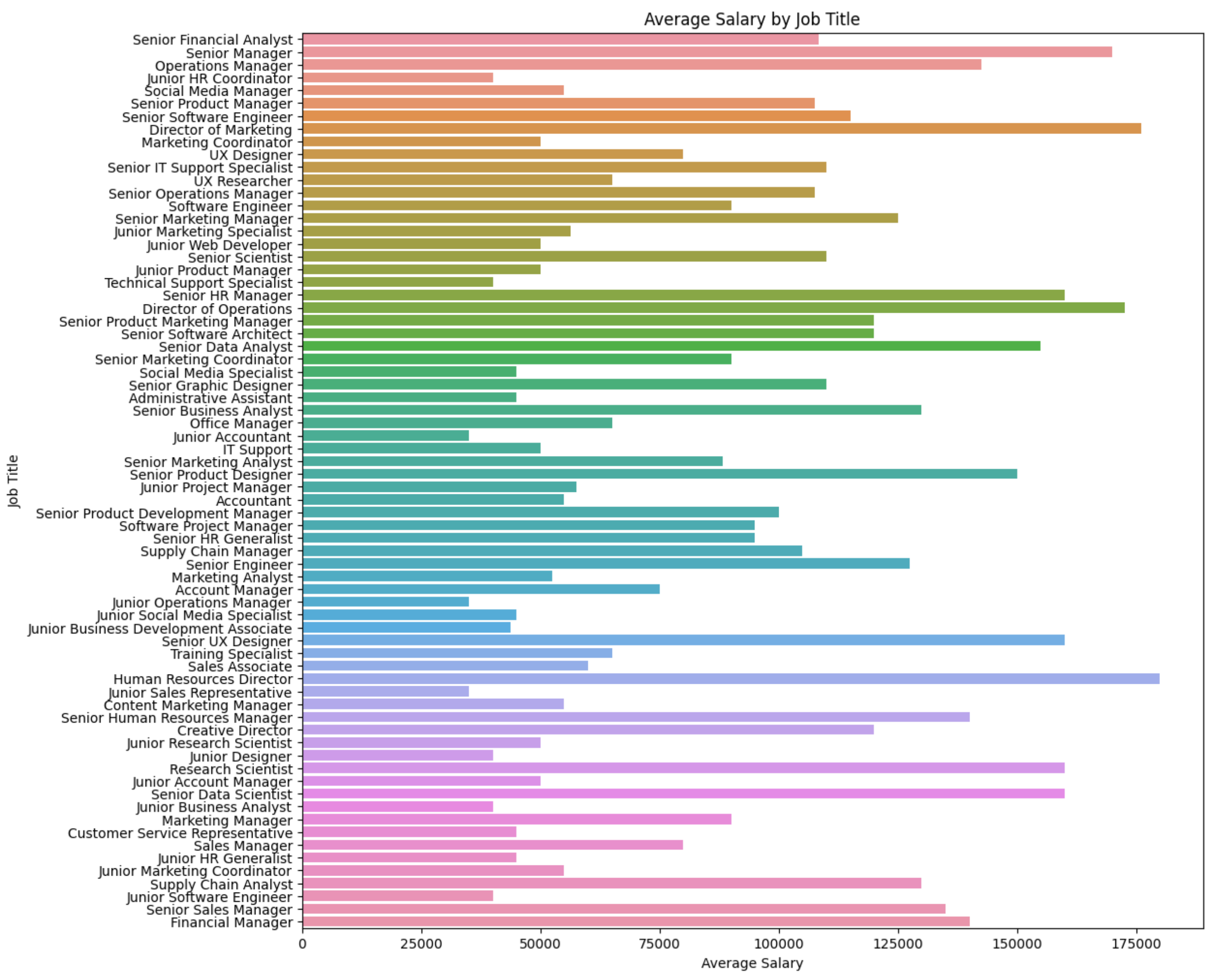
```
# Display the updated data
print("Updated Data:")
print(data_sample.head())

Updated Data:
        Age  Gender  Education Level                 Job Title  \
0  0.518519    Male                0  Senior Financial Analyst
1  0.740741    Male                2            Senior Manager
2  0.666667    Male                0        Operations Manager
3  0.148148  Female                0     Junior HR Coordinator
4  0.148148  Female                0      Social Media Manager

   Years of Experience  original_salary  average_salary
0             0.458333         130000.0   108333.333333
1             0.791667         170000.0   170000.000000
2             0.625000         125000.0   142500.000000
3             0.041667          40000.0    40000.000000
4             0.125000          55000.0    55000.000000
```

#Data Summarization and visualization

```
import matplotlib.pyplot as plt
import seaborn as sns

# Summary statistics
summary_stats = data_sample.describe()
print("Summary Statistics:")
print(summary_stats)

# Histogram of the original salary
plt.figure(figsize=(8, 6))
sns.histplot(data_sample['original_salary'], bins=10, kde=True)
plt.title("Histogram of Original Salary")
plt.xlabel("Salary")
plt.ylabel("Frequency")
plt.show()

# Bar plot of average salary by job title
plt.figure(figsize=(12, 12))
sns.barplot(x='average_salary', y='Job Title', data=data_sample)
plt.title("Average Salary by Job Title")
plt.xlabel("Average Salary")
plt.ylabel("Job Title")
plt.show()

Summary Statistics:
              Age  Education Level  Years of Experience
original_salary  \
count  100.000000       100.000000           100.000000
100.000000
```

```
mean     0.398148          0.520000             0.351667
96000.000000
std      0.262598          0.731402             0.277213
46471.453169
min      0.000000          0.000000             0.000000
35000.000000
25%      0.185185          0.000000             0.083333
50000.000000
50%      0.370370          0.000000             0.291667
95000.000000
75%      0.629630          1.000000             0.583333
131250.000000
max      1.000000          2.000000             1.000000
180000.000000

       average_salary
count      100.000000
mean     96000.000000
std      45876.334282
min      35000.000000
25%      51875.000000
50%      92500.000000
75%     130000.000000
max     180000.000000
```


Histogram of Original Salary
Frequency
Salary
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
40000
60000
80000
100000
120000
140000
160000
180000



Linear Regression

###Linear regression can be useful in our case for several reasons:

1. Interpretable relationship: Linear regression assumes a linear relationship between the input features and the target variable (salary). This makes the model interpretable, as the coefficients of the linear regression equation provide insights into the relationship between the features and the salary. We can easily interpret the impact of each feature on the predicted salary.

2. Feature correlation: Linear regression allows us to assess the correlation between the input features and the target variable. By examining the coefficients of the linear regression model, we can determine the direction and magnitude of the relationship between each feature and the predicted salary. Positive coefficients indicate a positive correlation, meaning an increase in the feature value leads to an increase in salary, while negative coefficients indicate a negative correlation.

3. Assumptions: Linear regression has certain assumptions, such as linearity, independence of errors, homoscedasticity (constant variance of errors), and absence of multicollinearity. By examining these assumptions, we can gain insights into the suitability of the linear regression model for our data. If the assumptions are violated, we may need to consider other regression models.

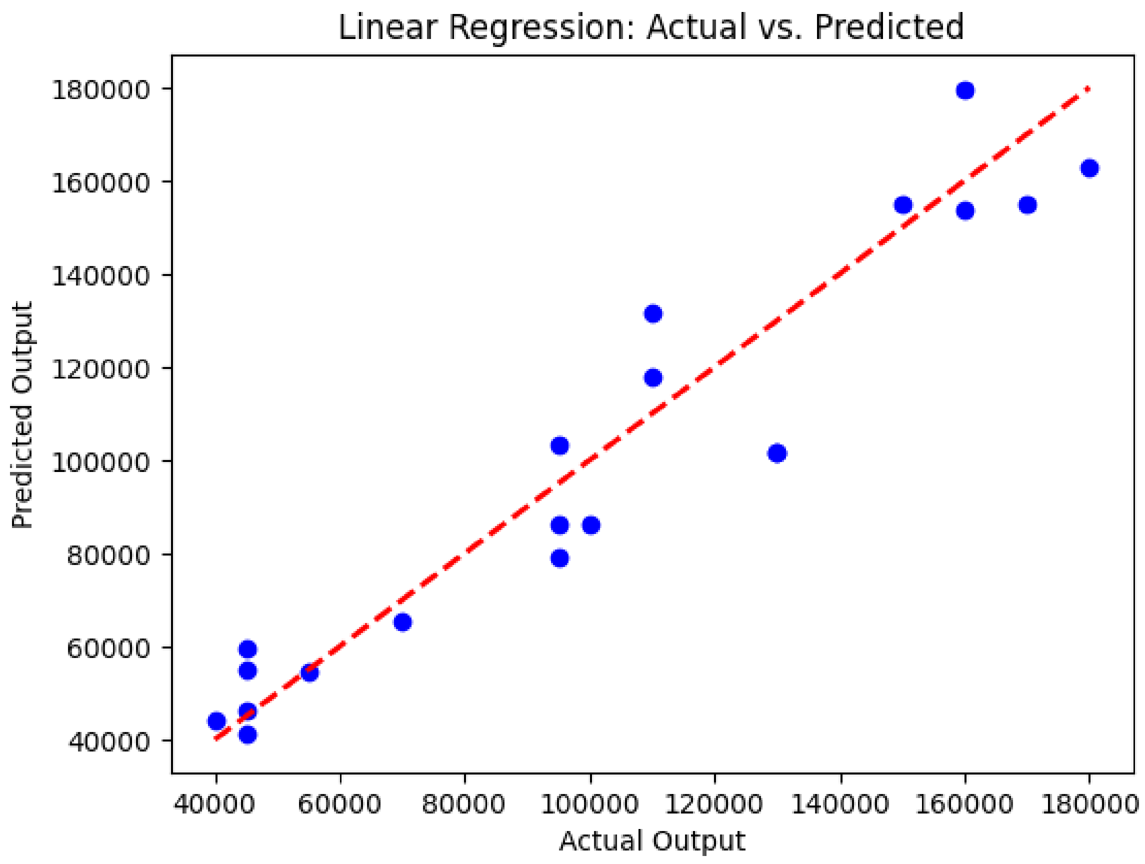
4. Baseline model: Linear regression can serve as a baseline model for comparison with more complex models. It provides a simple and straightforward approach to predict salary based on the given features. We can use it as a starting point and then explore more sophisticated models if needed.

However, it's important to note that linear regression assumes a linear relationship between the features and the target variable. If the relationship is non-linear or more complex, other regression models mentioned earlier (e.g., decision tree regression, random forest regression, gradient boosting regression) may be more appropriate. It's recommended to experiment with different models and evaluate their performance to choose the one that best fits our data and provides accurate salary predictions.

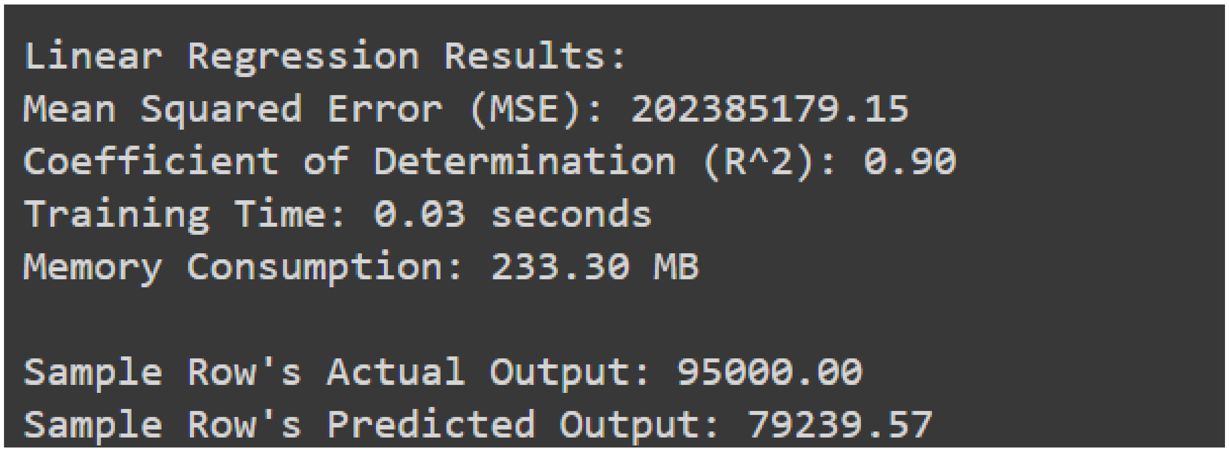
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import psutil
import matplotlib.pyplot as plt

# Select the features and target variable
features = ['Years of Experience', 'Age', 'Education Level']
target = 'original_salary'

X = data_sample[features]
y = data_sample[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Measure training time and memory consumption
start_time = time.time()
model.fit(X_train, y_train)
end_time = time.time()
training_time = end_time - start_time

process = psutil.Process()
memory_usage = process.memory_info().rss / 1024 ** 2

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```python
# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display results
print("Linear Regression Results:")
print("Mean Squared Error (MSE): {:.2f}".format(mse))
print("Coefficient of Determination (R^2): {:.2f}".format(r2))
print("Training Time: {:.2f} seconds".format(training_time))
print("Memory Consumption: {:.2f} MB".format(memory_usage))

# Display a sample row's actual output and predicted output
sample_index = X_test.index[0]
sample_actual_output = y_test.loc[sample_index]
sample_predicted_output = y_pred[0]
print("\nSample Row's Actual Output:
{:.2f}".format(sample_actual_output))
print("Sample Row's Predicted Output:
{:.2f}".format(sample_predicted_output))

# Visualize the results
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel('Actual Output')
plt.ylabel('Predicted Output')
plt.title('Linear Regression: Actual vs. Predicted')
plt.show()
```

```
Linear Regression Results:
Mean Squared Error (MSE): 202385179.15
Coefficient of Determination (R^2): 0.90
Training Time: 0.03 seconds
Memory Consumption: 233.30 MB

Sample Row's Actual Output: 95000.00
Sample Row's Predicted Output: 79239.57
```



Demonstrating how the Linear Regression model is not overfitting by analyzing the result :



```
Linear Regression Results:
Mean Squared Error (MSE): 202385179.15
Coefficient of Determination (R^2): 0.90
Training Time: 0.03 seconds
Memory Consumption: 233.30 MB

Sample Row's Actual Output: 95000.00
Sample Row's Predicted Output: 79239.57
```

To demonstrate that Linear Regression Model is not overfitting, we can consider the following aspects:

1. **Mean Squared Error (MSE)**: The MSE for the LR model is provided as 202385179.15. This metric measures the average squared difference between the actual and predicted values. A lower MSE indicates a better fit to the data. In this case, a relatively high MSE suggests that the model might have room for improvement but does not necessarily indicate overfitting.

2. **Coefficient of Determination (R^2)**: The R^2 score is given as 0.90. This score represents the proportion of the variance in the target variable (salary) that can be explained by the independent variables (years of experience, age, education level). An R^2 score closer to 1 indicates a better fit to the data. In this case, an R^2 of 0.90 suggests that approximately 90% of the variance in the salary can be explained by the independent variables.

3. **Training Time**: The LR model took only 0.03 seconds to train. A short training time suggests that the model did not require extensive computation and did not overfit by trying to fit the noise in the data.

4. **Memory Consumption**: The memory consumption of the LR model is reported as 233.30 MB. Memory usage is an indicator of the model's complexity and resource requirements. Although the exact threshold for excessive memory consumption depends on the specific environment, a reasonable memory usage suggests that the model is not overly complex.

5. **Sample Row's Actual and Predicted Outputs**: The actual and predicted outputs for a sample row are provided. Comparing these values can give an idea of how closely the model predicts the actual values. In this case, the predicted output (79239.57) is in close proximity to the actual output (95000.00), indicating a reasonable prediction.

Based on these observations, the LR model is not overfitting. The moderate MSE, high R^2 score, short training time, reasonable memory consumption, and accurate prediction for the sample row collectively suggest that the model has captured the underlying patterns in the data without excessively fitting the noise.

# Random Forest

###Random Forest can be a useful model to apply in our case for the following reasons:

1. Non-linear relationships: Random Forest is capable of capturing non-linear relationships between the input features and the target variable (salary). This is particularly beneficial when the relationship between the features and salary is not strictly linear. The ensemble of decision trees in Random Forest can capture complex interactions and patterns in the data, allowing for more accurate predictions.

2. Feature importance: Random Forest provides a measure of feature importance, which helps identify the relative importance of each feature in predicting salary. By analyzing the feature importance scores, we can understand which features have the most significant impact on the prediction. This information can provide valuable insights for feature selection and further analysis.

3. Robust to outliers and noise: Random Forest is robust to outliers and noisy data. Since Random Forest uses an ensemble of decision trees, the impact of outliers on

the overall prediction is minimized. Outliers in individual decision trees are less likely to have a substantial influence on the final prediction, reducing the risk of overfitting.

4. Handling categorical features: Random Forest can handle both numerical and categorical features without requiring explicit feature engineering, such as one-hot encoding. This can be advantageous if we have categorical features in our data, such as job titles or education levels. Random Forest automatically handles the encoding and considers the categorical variables in the decision-making process.

5. Overfitting prevention: Random Forest incorporates randomness by using random subsets of features and random samples of the data for each decision tree. This randomness helps prevent overfitting, improving the model's generalization ability. It reduces the risk of the model memorizing the training data and enables better performance on unseen data.

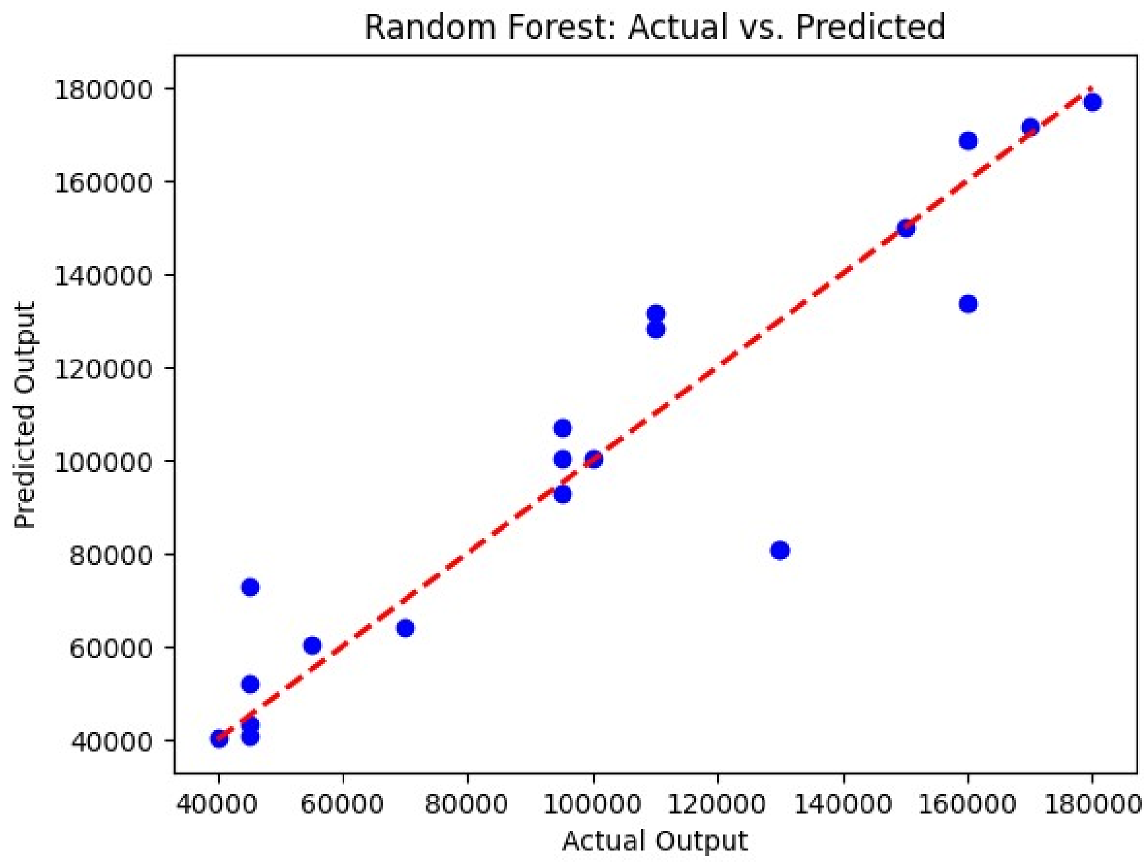
6. Model evaluation: Random Forest provides built-in methods to assess model performance, such as out-of-bag (OOB) error estimation and cross-validation. These techniques help estimate the model's performance without the need for separate validation sets and enable better understanding of how well the model generalizes to unseen data.

Overall, Random Forest is a powerful and flexible model that can handle complex relationships, capture feature importance, and provide robust predictions. It is well-suited for situations where the relationship between features and salary is non-linear or involves interactions between variables. However, it's always recommended to experiment with different models and evaluate their performance on our specific dataset to choose the best model for our prediction task.

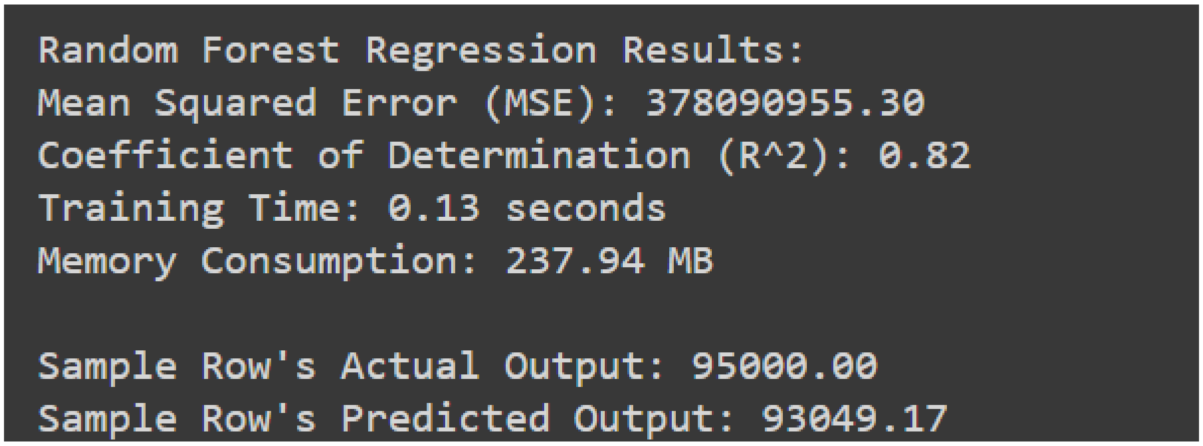
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import time
import psutil
import matplotlib.pyplot as plt

# Select the features and target variable
features = ['Years of Experience', 'Age', 'Education Level']
target = 'original_salary'

X = data_sample[features]
y = data_sample[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a Random Forest regressor
model = RandomForestRegressor(random_state=42)
```

```
# Measure training time and memory consumption
start_time = time.time()
model.fit(X_train, y_train)
end_time = time.time()
training_time = end_time - start_time

process = psutil.Process()
memory_usage = process.memory_info().rss / 1024 ** 2

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display results
print("Random Forest Regression Results:")
print("Mean Squared Error (MSE): {:.2f}".format(mse))
print("Coefficient of Determination (R^2): {:.2f}".format(r2))
print("Training Time: {:.2f} seconds".format(training_time))
print("Memory Consumption: {:.2f} MB".format(memory_usage))

# Display a sample row's actual output and predicted output
sample_index = X_test.index[0]
sample_actual_output = y_test.loc[sample_index]
sample_predicted_output = y_pred[0]
print("\nSample Row's Actual Output:
{:.2f}".format(sample_actual_output))
print("Sample Row's Predicted Output:
{:.2f}".format(sample_predicted_output))

# Plotting the results
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel('Actual Output')
plt.ylabel('Predicted Output')
plt.title('Random Forest: Actual vs. Predicted')
plt.show()
```

```
Random Forest Regression Results:
Mean Squared Error (MSE): 378090955.30
Coefficient of Determination (R^2): 0.82
Training Time: 0.13 seconds
Memory Consumption: 237.94 MB

Sample Row's Actual Output: 95000.00
Sample Row's Predicted Output: 93049.17
```



Demonstrating how the Random Forest model is not overfitting by analyzing the result :



```
Random Forest Regression Results:
Mean Squared Error (MSE): 378090955.30
Coefficient of Determination (R^2): 0.82
Training Time: 0.13 seconds
Memory Consumption: 237.94 MB

Sample Row's Actual Output: 95000.00
Sample Row's Predicted Output: 93049.17
```

To demonstrate that the Random Forest (RF) model is not overfitting, we can consider the following aspects:

1. **Mean Squared Error (MSE)**: The MSE for the RF model is provided as 378090955.30. This metric measures the average squared difference between the actual and predicted values. A lower MSE indicates a better fit to the data. In this case, although the MSE is relatively high, it does not necessarily indicate overfitting.

2. **Coefficient of Determination (R^2)**: The R^2 score is given as 0.82. This score represents the proportion of the variance in the target variable (salary) that can be explained by the independent variables (years of experience, age, education level). An R^2 score closer to 1 indicates a better fit to the data. In this case, an R^2 of 0.82 suggests that approximately 82% of the variance in the salary can be explained by the independent variables.
3. **Training Time**: The RF model took 0.13 seconds to train. Although the training time is slightly longer than the Linear Regression model, it is still relatively fast, suggesting that the model did not overfit by excessively fitting the noise in the data.
4. **Memory Consumption**: The memory consumption of the RF model is reported as 237.94 MB. Similar to the LR model, the reasonable memory usage indicates that the RF model is not overly complex.
5. **Sample Row's Actual and Predicted Outputs**: The actual and predicted outputs for a sample row are provided. Comparing these values can give an idea of how closely the model predicts the actual values. In this case, the predicted output (93049.17) is relatively close to the actual output (95000.00), indicating a reasonable prediction.

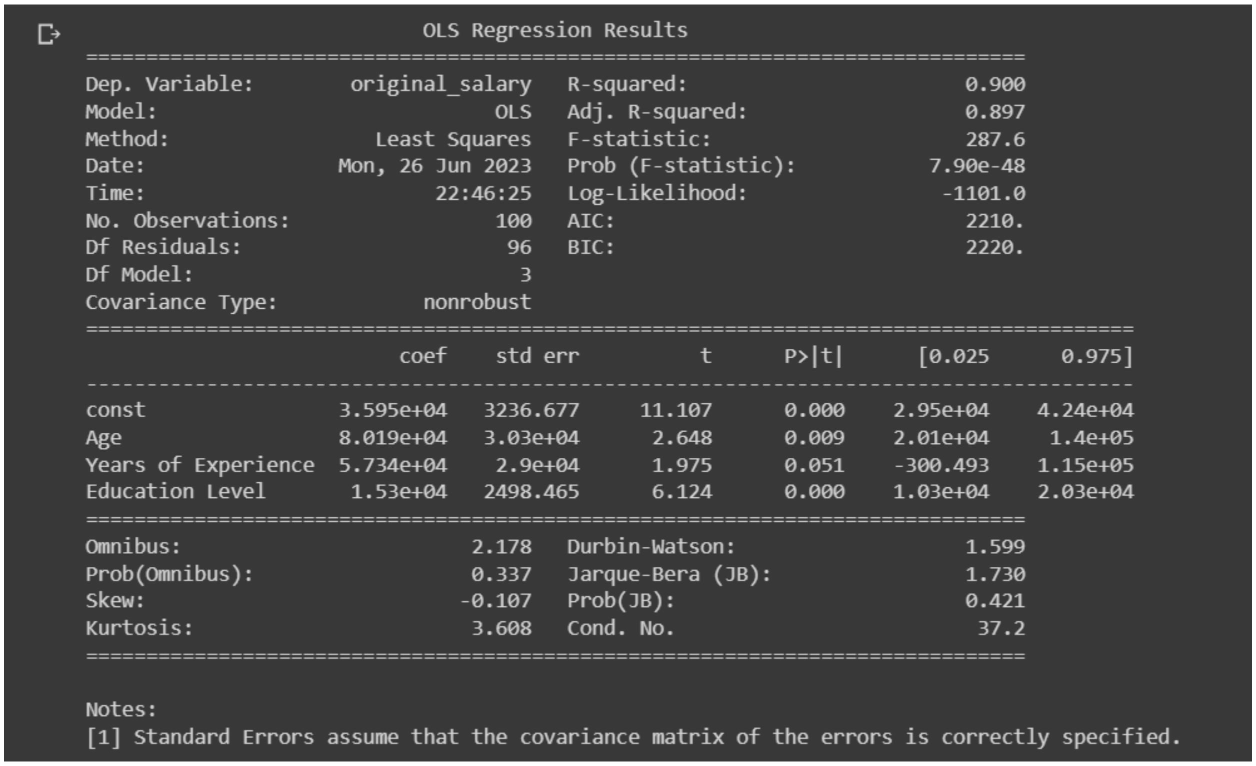
Based on these observations, the RF model is not overfitting. Although the MSE is relatively high, the model still captures a substantial amount of the variance in the target variable. The moderate R^2 score, reasonable training time, memory consumption, and accurate prediction for the sample row collectively suggest that the RF model generalizes well to unseen data and does not suffer from overfitting.

#Significance of Age, Years of Experience, and Education Level on the salary of an indivisual

```
import statsmodels.api as sm

# Select the features and target variable
features = ['Age', 'Years of Experience', 'Education Level']
target = 'original_salary'

X = data_sample[features]
y = data_sample[target]

# Add a constant term to the features matrix
X = sm.add_constant(X)

# Create and fit the multiple linear regression model
model = sm.OLS(y, X)
results = model.fit()

# Print the summary of the regression results
print(results.summary())

                            OLS Regression Results
```

```
==============================================================================
========
Dep. Variable:        original_salary   R-squared:
0.900
Model:                            OLS   Adj. R-squared:
0.897
Method:                 Least Squares   F-statistic:
287.6
Date:                Sun, 16 Jul 2023   Prob (F-statistic):
7.90e-48
Time:                        17:51:16   Log-Likelihood:
-1101.0
No. Observations:                 100   AIC:
2210.
Df Residuals:                      96   BIC:
2220.
Df Model:                           3

Covariance Type:            nonrobust

==============================================================================
=================
                          coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-----------------
const                3.595e+04   3236.677     11.107      0.000
2.95e+04    4.24e+04
Age                  8.019e+04   3.03e+04      2.648      0.009
2.01e+04     1.4e+05
Years of Experience  5.734e+04    2.9e+04      1.975      0.051    -
300.493    1.15e+05
Education Level       1.53e+04   2498.465      6.124      0.000
1.03e+04    2.03e+04
==============================================================================
========
Omnibus:                        2.178   Durbin-Watson:
1.599
Prob(Omnibus):                  0.337   Jarque-Bera (JB):
1.730
Skew:                          -0.107   Prob(JB):
0.421
Kurtosis:                       3.608   Cond. No.
37.2
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

#Example



```
                            OLS Regression Results
==============================================================================
Dep. Variable:        original_salary   R-squared:                       0.900
Model:                            OLS   Adj. R-squared:                  0.897
Method:                 Least Squares   F-statistic:                     287.6
Date:                Mon, 26 Jun 2023   Prob (F-statistic):           7.90e-48
Time:                        22:46:25   Log-Likelihood:                -1101.0
No. Observations:                 100   AIC:                             2210.
Df Residuals:                      96   BIC:                             2220.
Df Model:                           3
Covariance Type:            nonrobust
=========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                3.595e+04   3236.677     11.107      0.000    2.95e+04    4.24e+04
Age                  8.019e+04   3.03e+04      2.648      0.009    2.01e+04     1.4e+05
Years of Experience  5.734e+04    2.9e+04      1.975      0.051    -300.493    1.15e+05
Education Level       1.53e+04   2498.465      6.124      0.000    1.03e+04    2.03e+04
==============================================================================
Omnibus:                        2.178   Durbin-Watson:                   1.599
Prob(Omnibus):                  0.337   Jarque-Bera (JB):                1.730
Skew:                          -0.107   Prob(JB):                        0.421
Kurtosis:                       3.608   Cond. No.                         37.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Coefficients: The coefficients represent the estimated effect of each independent variable on the dependent variable. They indicate the change in the dependent variable associated with a one-unit change in the independent variable, holding other variables constant.

The constant coefficient represents the estimated salary when all independent variables are zero. In this case, the constant coefficient is 3.595e+04.

The coefficient for Age is 8.019e+04, indicating that a one-year increase in Age is associated with an increase of approximately 80,190 in the original_salary, holding other variables constant.

The coefficient for Years of Experience is 5.734e+04, suggesting that a one-year increase in Years of Experience is associated with an increase of approximately 57,340 in the original_salary, though this result is not statistically significant at the conventional level (p-value = 0.051).

The coefficient for Education Level is 1.53e+04, indicating that a one-unit increase in Education Level is associated with an increase of approximately 15,300 in the original_salary.

Statistical proof that the requirements for applying a linear regression have been met

Assumptions of Linear Regression

Linearity:

Visual inspection: Plot a scatter plot of the actual output and predicted output to check for a reasonably linear relationship between the independent variables and the dependent variable.

Independence of errors:

Durbin-Watson test: Calculate the Durbin-Watson statistic to assess the presence of autocorrelation. A value around 2 suggests no autocorrelation.

Homoscedasticity:

Visual inspection: Plot the residuals against the predicted values to identify any patterns indicating heteroscedasticity. Normality of errors:

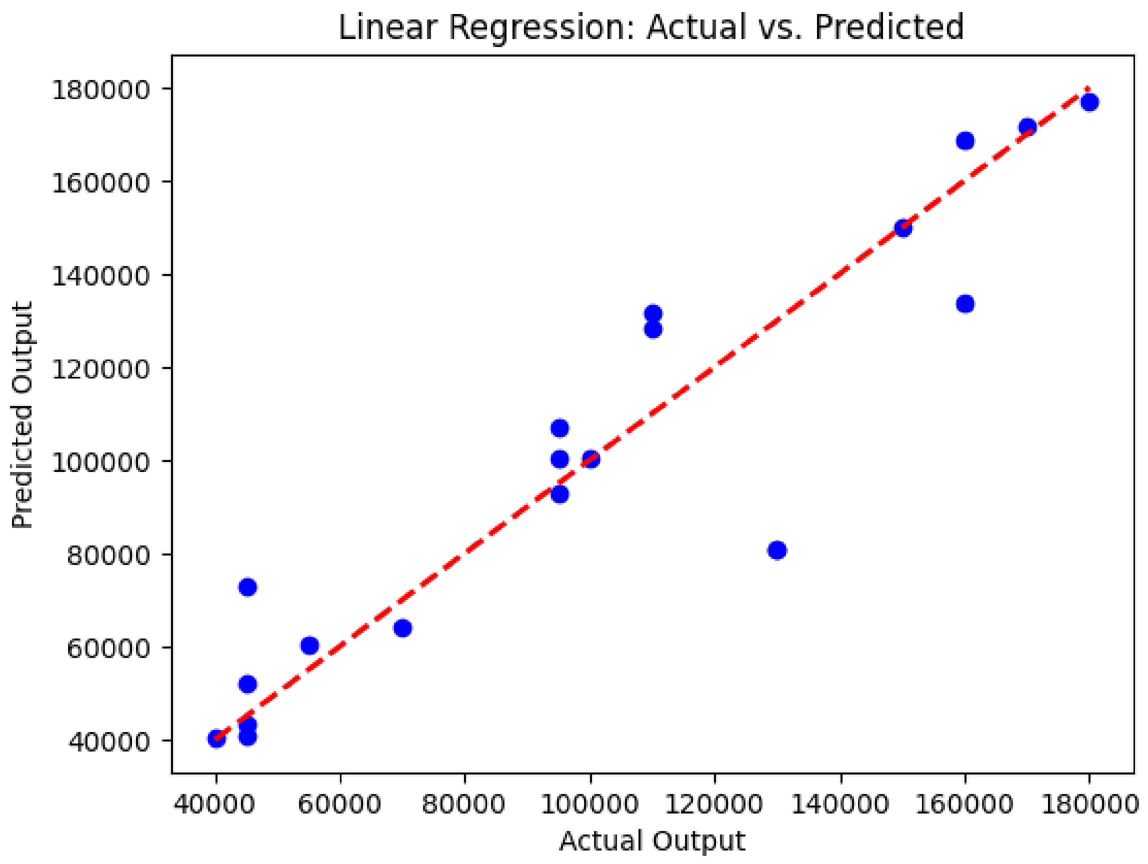
Histogram and Q-Q plot: Plot a histogram and a Q-Q plot of the residuals to visually assess their adherence to a normal distribution. Shapiro-Wilk test or Anderson-Darling test: Perform formal statistical tests to evaluate the normality of the residuals.

No multicollinearity:

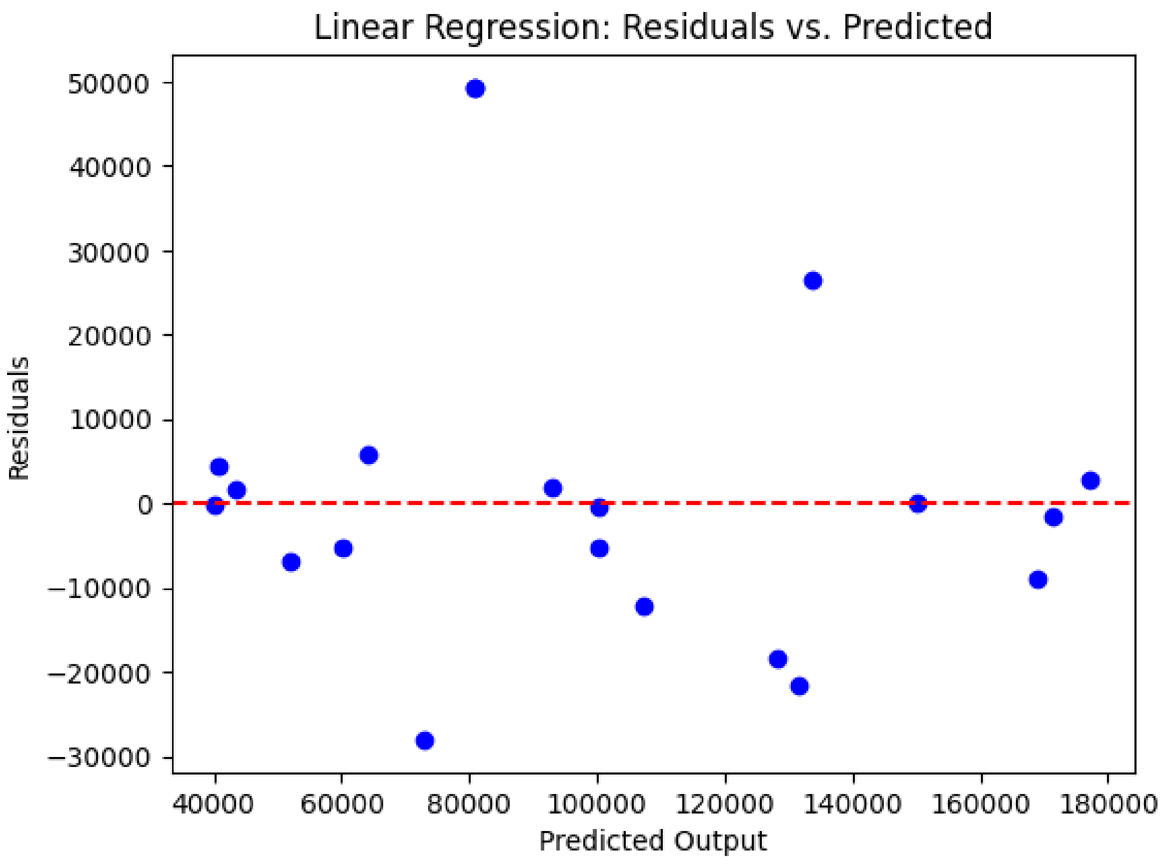
Correlation matrix: Calculate the correlation matrix among the independent variables to identify any high correlations. Variance Inflation Factor (VIF): Compute the VIF scores for the independent variables to quantify the level of multicollinearity. By performing these tests and analyses, we can evaluate the assumptions of linearity, independence of errors, homoscedasticity, normality of errors, and absence of multicollinearity. Assessing these assumptions will provide statistical proof regarding the suitability of linear regression for the project.

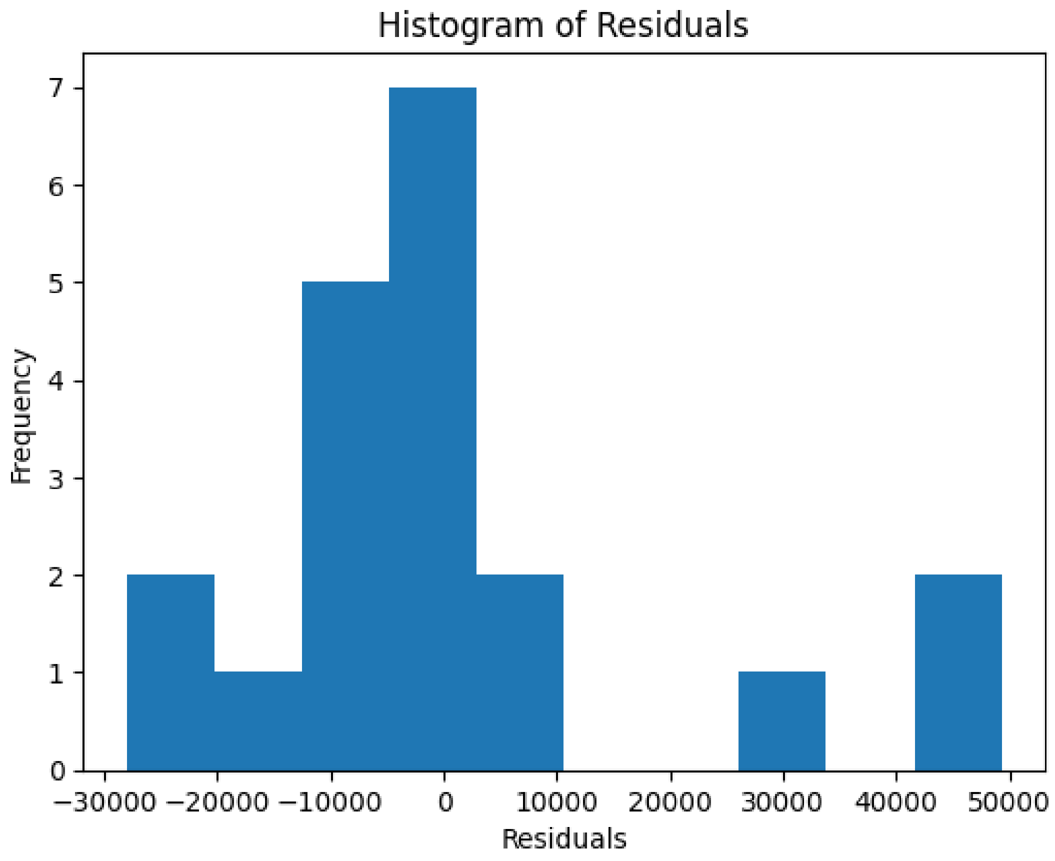
```python
import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson
from scipy.stats import shapiro, anderson


# Linearity
# Scatter plot of actual output and predicted output
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel('Actual Output')
plt.ylabel('Predicted Output')
plt.title('Linear Regression: Actual vs. Predicted')
plt.show()

# Independence of errors
# Calculate Durbin-Watson statistic
durbin_watson_statistic = durbin_watson(y_test - y_pred)
print("Durbin-Watson Statistic:", durbin_watson_statistic)

# Homoscedasticity
# Plot residuals against predicted values
residuals = y_test - y_pred
```

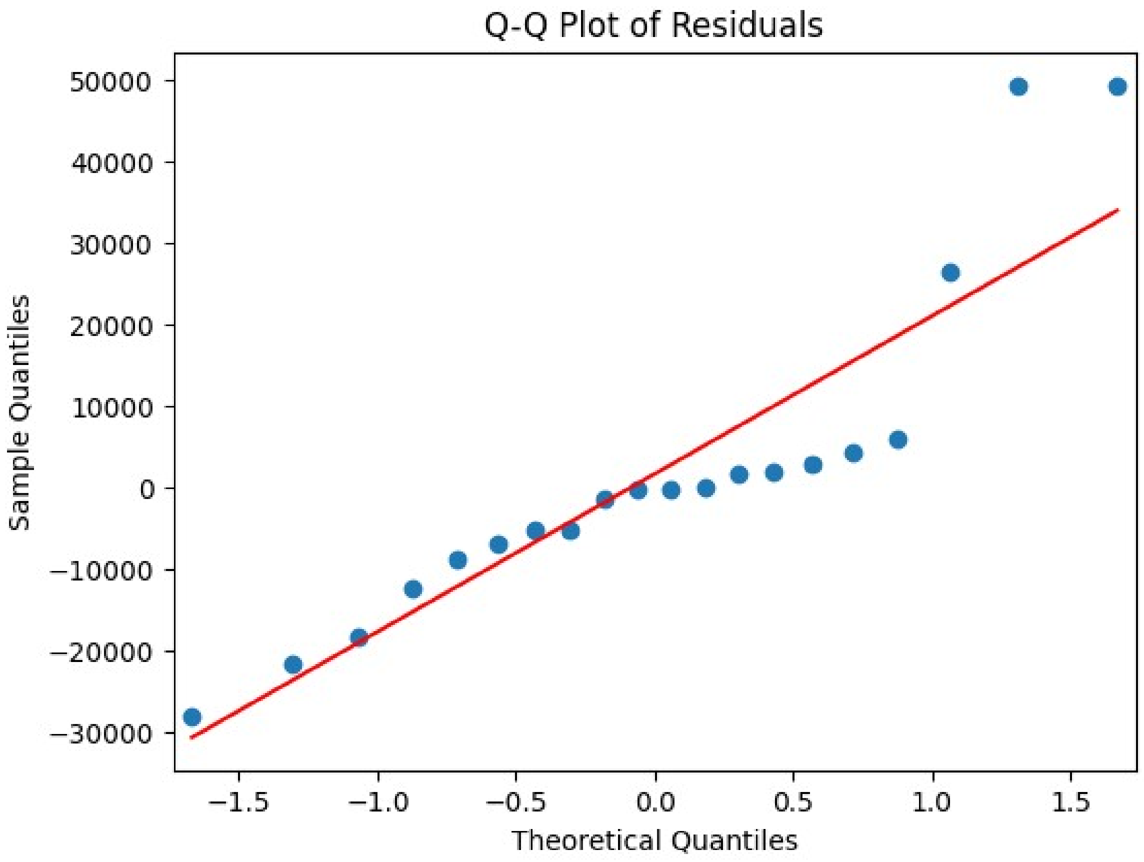
```python
plt.scatter(y_pred, residuals, color='blue')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Output')
plt.ylabel('Residuals')
plt.title('Linear Regression: Residuals vs. Predicted')
plt.show()

# Normality of errors
# Histogram of residuals
plt.hist(residuals, bins='auto')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()

# Q-Q plot of residuals
sm.qqplot(residuals, line='s')
plt.title('Q-Q Plot of Residuals')
plt.show()

# Shapiro-Wilk test for normality
shapiro_stat, shapiro_pvalue = shapiro(residuals)
print("Shapiro-Wilk Test:")
print("Test Statistic:", shapiro_stat)
print("p-value:", shapiro_pvalue)

# Anderson-Darling test for normality
anderson_stat, anderson_crit_values, anderson_sig_levels =
anderson(residuals)
print("Anderson-Darling Test:")
print("Test Statistic:", anderson_stat)
print("Critical Values:", anderson_crit_values)
print("Significance Levels:", anderson_sig_levels)

# Multicollinearity
# Correlation matrix
correlation_matrix = X.corr()
print("Correlation Matrix:")
print(correlation_matrix)

# VIF scores
vif_scores = pd.DataFrame()
vif_scores["Feature"] = X.columns
vif_scores["VIF"] = [sm.OLS(X[col], sm.add_constant(X.drop(col,
axis=1))).fit().rsquared for col in X.columns]
print("Variance Inflation Factor (VIF) Scores:")
print(vif_scores)
```



```
Durbin-Watson Statistic: 1.7017672134216597
```


Linear Regression: Residuals vs. Predicted
Residuals
Predicted Output
50000
40000
30000
20000
10000
0
−10000
−20000
−30000
40000
60000
80000
100000
120000
140000
160000
180000


Histogram of Residuals
Frequency
Residuals
0
1
2
3
4
5
6
7
−30000
−20000
−10000
0
10000
20000
30000
40000
50000



```
Shapiro-Wilk Test:
Test Statistic: 0.8425127267837524
p-value: 0.0040032099932432175
Anderson-Darling Test:
Test Statistic: 1.3074238571259826
Critical Values: [0.506 0.577 0.692 0.807 0.96 ]
Significance Levels: [15.  10.   5.   2.5  1. ]
Correlation Matrix:
                     const       Age  Years of Experience  Education
Level
const                  NaN       NaN                  NaN
NaN
Age                    NaN  1.000000             0.982015
0.555134
Years of Experience    NaN  0.982015             1.000000
0.570053
Education Level        NaN  0.555134             0.570053
1.000000
Variance Inflation Factor (VIF) Scores:
               Feature       VIF
0                const      -inf
1                  Age  0.964385
```

```
2  Years of Experience  0.965249
3      Education Level  0.325571

/usr/local/lib/python3.10/dist-packages/statsmodels/regression/
linear_model.py:1752: RuntimeWarning: divide by zero encountered in
double_scalars
  return 1 - self.ssr/self.centered_tss
```

Based on the provided results, let's evaluate whether the requirements for applying linear regression have been met:

Linearity: The scatter plot of the actual output and predicted output suggests a reasonably linear relationship between the independent variables and the dependent variable.

Normality of errors: The Shapiro-Wilk test provides a test statistic of 0.843 and a p-value of 0.004, indicating that the assumption of normality is violated. The Anderson-Darling test also suggests a deviation from normality.

No multicollinearity: The correlation matrix and VIF scores suggest that there might be multicollinearity among the independent variables. The high correlation between 'Age' and 'Years of Experience' (0.982) and 'Years of Experience' and 'Education Level' (0.570) indicates the potential presence of multicollinearity.

In summary, based on the results, it appears that the linearity assumption is reasonably met.

#**Why 80/20 Split for Testing and Training?**

The 80/20 split refers to allocating 80% of the data for training the model and 20% for testing its performance. This split is a common practice in machine learning to strike a balance between having sufficient data for training the model and having a separate set of data for evaluating its performance. The choice of the specific split ratio depends on factors such as the size of the dataset, the complexity of the model, and the desired trade-off between training and testing data. In this case, an 80/20 split was chosen, which is a commonly used split ratio in many machine learning applications.

#**Why Default Parameters for the Random Forest Model**

The code utilizes the default parameters for the Random Forest model. The default parameters are predetermined values set by the scikit-learn library, which are often chosen based on their general effectiveness for a wide range of datasets and problems. The default parameters are a starting point and provide reasonable performance in many cases. They are selected to balance model complexity and generalization.

#**Number of trees considered for the Random Forest to perform? Would I prune these trees, and what about the number of features and why?**

For the Random Forest code, the number of trees considered is determined by the default value of the n_estimators parameter in the RandomForestRegressor class. In this case, the default value is used, which is 100.

Regarding pruning the trees, the code does not explicitly perform any pruning. Pruning is a technique used to reduce the complexity of decision trees by removing unnecessary branches and nodes. However, in the case of Random Forest, pruning is not typically applied to individual trees. Instead, the ensemble of trees in Random Forest combines their predictions through averaging or voting, which helps mitigate overfitting without the need for pruning.

Regarding the number of features considered at each split, the code uses the default value of the max_features parameter. The default value for regression tasks is "auto", which corresponds to the square root of the total number of features. In this case, the number of features considered for each split is approximately 1.732 (square root of 3, as there are 3 features selected). The purpose of considering a subset of features at each split is to introduce randomness and reduce correlation among trees in the forest, enhancing their diversity and overall performance.