

▼ Task

Configure Kaggle API credentials by setting the `KAGGLE_USERNAME` and `KAGGLE_KEY` environment variables.

▼ Configure Kaggle API Credentials

Subtask:

Set the `KAGGLE_USERNAME` and `KAGGLE_KEY` environment variables using the provided credentials. This will allow the Kaggle API to authenticate without needing a `kaggle.json` file.

Reasoning: The user needs to set Kaggle API credentials as environment variables. This code block will provide the necessary structure for them to input their username and API key.

```
import os

# Set your Kaggle username and API key as environment variables
# Replace 'YOUR_KAGGLE_USERNAME' and 'YOUR_KAGGLE_KEY' with your actual credentials
os.environ['KAGGLE_USERNAME'] = 'nareddyjasmitha'
os.environ['KAGGLE_KEY'] = '09c0866fc358c092b69e00fde54436f2'
```

KAGGLE_USERNAME and KAGGLE_KEY environment variables have been set.

▼ Task

Download the 'salary-dataset-simple-linear-regression' dataset from Kaggle and unzip the downloaded file.

▼ Download Dataset

Subtask:

Download the 'salary-dataset-simple-linear-regression' dataset from Kaggle using the configured Kaggle API. This step also includes unzipping the downloaded file to access the CSV.

Reasoning: To download and unzip the specified Kaggle dataset, I will use the `kaggle` client library and its `dataset_download_files` method. This method will handle both downloading and unzipping the data into the current directory.

```
import kaggle
import os

# Download the dataset and unzip it
kaggle.api.dataset_download_files(
```

```
'rohankumar700/salary-dataset-simple-linear-regression',
path='.',
unzip=True
)

# Verify that the Salary_Data.csv file exists
csv_file_path = 'Salary_Data.csv'
if os.path.exists(csv_file_path):
    print(f"'{csv_file_path}' has been successfully downloaded and unzipped.")
else:
    print(f"Error: '{csv_file_path}' was not found after download.")
```

Dataset URL: <https://www.kaggle.com/datasets/rohankumar700/salary-dataset-simple-linear-regression>

HTTPError Traceback (most recent call last)
[/tmp/ipython-input-530531534.py](#) in <cell line: 0>()

```
3
4 # Download the dataset and unzip it
----> 5 kaggle.api.dataset_download_files(
6     'rohankumar700/salary-dataset-simple-linear-regression',
7     path='.',
```

◆ 4 frames ◆

[/usr/local/lib/python3.12/dist-packages/requests/models.py](#) in raise_for_status(self)

```
1024
1025     if http_error_msg:
-> 1026         raise HTTPError(http_error_msg, response=self)
1027
1028     def close(self):
```

HTTPError: 403 Client Error: Forbidden for url:

<https://www.kaggle.com/api/v1/datasets/download/rohankumar700/salary-dataset-simple-linear-regression?raw=false>

Next steps: [Explain error](#)

Task

Download the 'abhishek14398/salary-dataset-simple-linear-regression' dataset from Kaggle and unzip the downloaded file to access the CSV.

Download and Unzip Dataset

Subtask:

Download the 'abhishek14398/salary-dataset-simple-linear-regression' dataset from Kaggle using the configured API credentials and unzip the downloaded file to access the CSV.

Reasoning: The previous attempt to download the dataset failed due to a 403 error, which often indicates an issue with authentication or permissions. To re-attempt the download for the specified dataset, I will provide the same code structure as before, but ensure the correct dataset slug is used, and then verify the file's existence.

```

import kaggle
import os

# Download the dataset and unzip it
kaggle.api.dataset_download_files(
    'abhishek14398/salary-dataset-simple-linear-regression',
    path='.',
    unzip=True
)

# Verify that the Salary_Data.csv file exists
csv_file_path = 'Salary_Data.csv'
if os.path.exists(csv_file_path):
    print(f"'{csv_file_path}' has been successfully downloaded and unzipped.")
else:
    print(f"Error: '{csv_file_path}' was not found after download.")

Dataset URL: https://www.kaggle.com/datasets/abhishek14398/salary-dataset-simple-linear-regression
Error: 'Salary_Data.csv' was not found after download.

```

Reasoning: The previous attempt indicated the dataset was downloaded but the expected CSV file was not found. I will list the contents of the current directory to identify the correct filename or understand the directory structure after unzipping.

```

import os

# List all files and directories in the current directory
print("Files and directories after download:")
for item in os.listdir('.'):
    print(item)

Files and directories after download:
.config
Salary_dataset.csv
sample_data

```

▼ Load and Prepare Data

Subtask:

Load the unzipped 'Salary_dataset.csv' into a pandas DataFrame. Inspect the data for any missing values or anomalies and then split the dataset into training and testing sets for model development.

Reasoning: The subtask requires loading the 'Salary_dataset.csv' file into a pandas DataFrame, inspecting it for missing values, and then splitting the data into training and testing sets. This code block will perform all these actions sequentially.

```

import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset into a pandas DataFrame
df = pd.read_csv('Salary_dataset.csv')

```

```
print("First 5 rows of the DataFrame:")
print(df.head())
print("\n")

print("DataFrame Info (including missing values check):")
df.info()
print("\n")

print("Missing values per column:")
print(df.isnull().sum())
print("\n")

# Separate features (X) and target (y)
X = df[['YearsExperience']]
y = df['Salary']

# 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)
```

```
print("Data split into training and testing sets successfully.")
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
First 5 rows of the DataFrame:
   Unnamed: 0  YearsExperience    Salary
0            0           1.2  39344.0
1            1           1.4  46206.0
2            2           1.6  37732.0
3            3           2.1  43526.0
4            4           2.3  39892.0
```

```
DataFrame Info (including missing values check):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Unnamed: 0        30 non-null     int64  
 1   YearsExperience  30 non-null     float64 
 2   Salary           30 non-null     float64 
dtypes: float64(2), int64(1)
memory usage: 852.0 bytes
```

```
Missing values per column:
  Unnamed: 0      0
  YearsExperience  0
  Salary          0
  dtype: int64
```

```
Data split into training and testing sets successfully.
X_train shape: (24, 1)
X_test shape: (6, 1)
y_train shape: (24,)
y_test shape: (6,)
```

▼ Train Linear Regression Model

Subtask:

Import the `LinearRegression` model from `sklearn.linear_model` and train it using the training data.

Reasoning: To train a Linear Regression model, I will import the `LinearRegression` class, instantiate it, and then fit the model using the provided training data `X_train` and `y_train`.

```
from sklearn.linear_model import LinearRegression

# Instantiate the Linear Regression model
linear_regressor = LinearRegression()

# Fit the model to the training data
linear_regressor.fit(X_train, y_train)

print("Linear Regression model trained successfully.")
```

Linear Regression model trained successfully.

▼ Predict and Evaluate Model

Subtask:

Use the trained linear regression model to predict salary values on the test set. Calculate the Mean Squared Error (MSE) and R-squared (R2) value to quantitatively evaluate the model's performance and accuracy.

Reasoning: To predict salary values using the trained linear regression model and evaluate its performance, I will use the `predict` method on the test set and then calculate the Mean Squared Error (MSE) and R-squared (R2) score using `sklearn.metrics`.

```
from sklearn.metrics import mean_squared_error, r2_score

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

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

# Calculate R-squared (R2) score
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R2) Score: {r2:.2f}")
```

Mean Squared Error (MSE): 49830096.86
R-squared (R2) Score: 0.90

▼ Visualize Regression Results

Subtask:

Generate two plots: one showing the regression line learned by Scikit-learn over the training data, and another comparing the actual salary values against the predicted salary values from the test set. Ensure

both plots include appropriate labels and a legend.

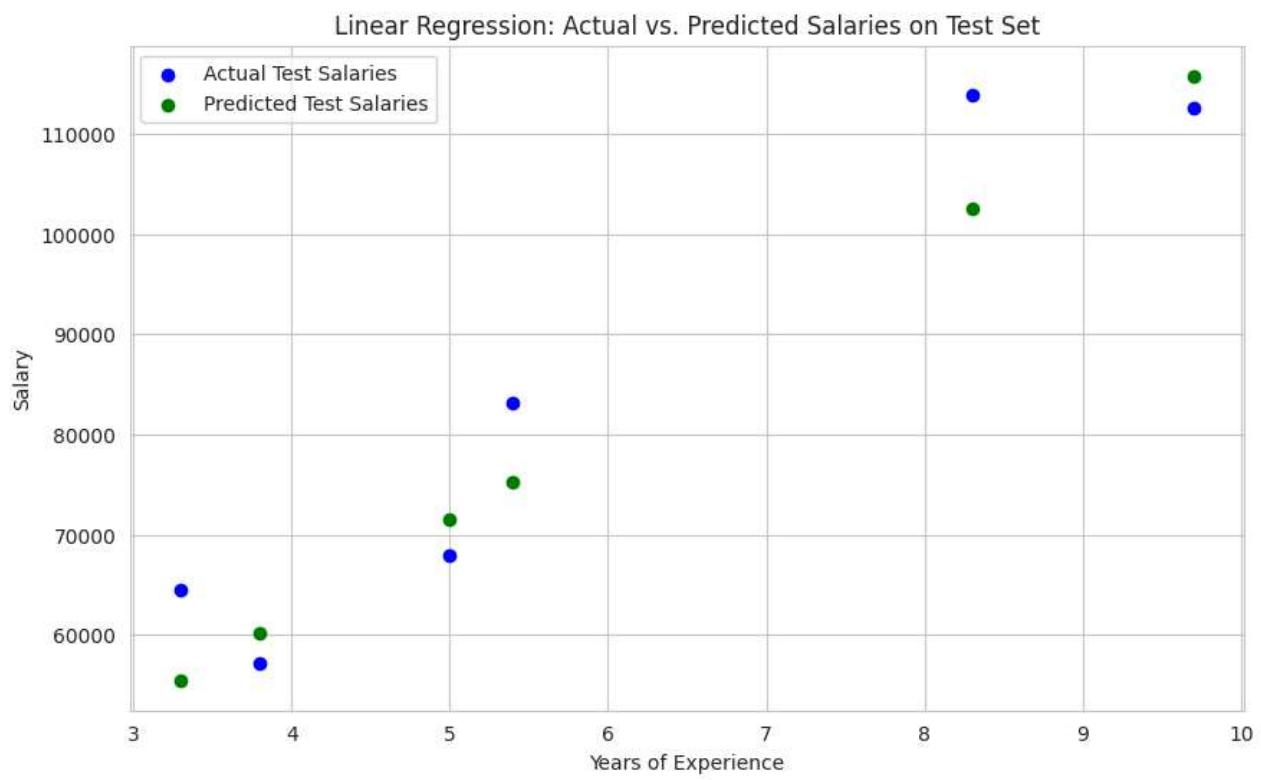
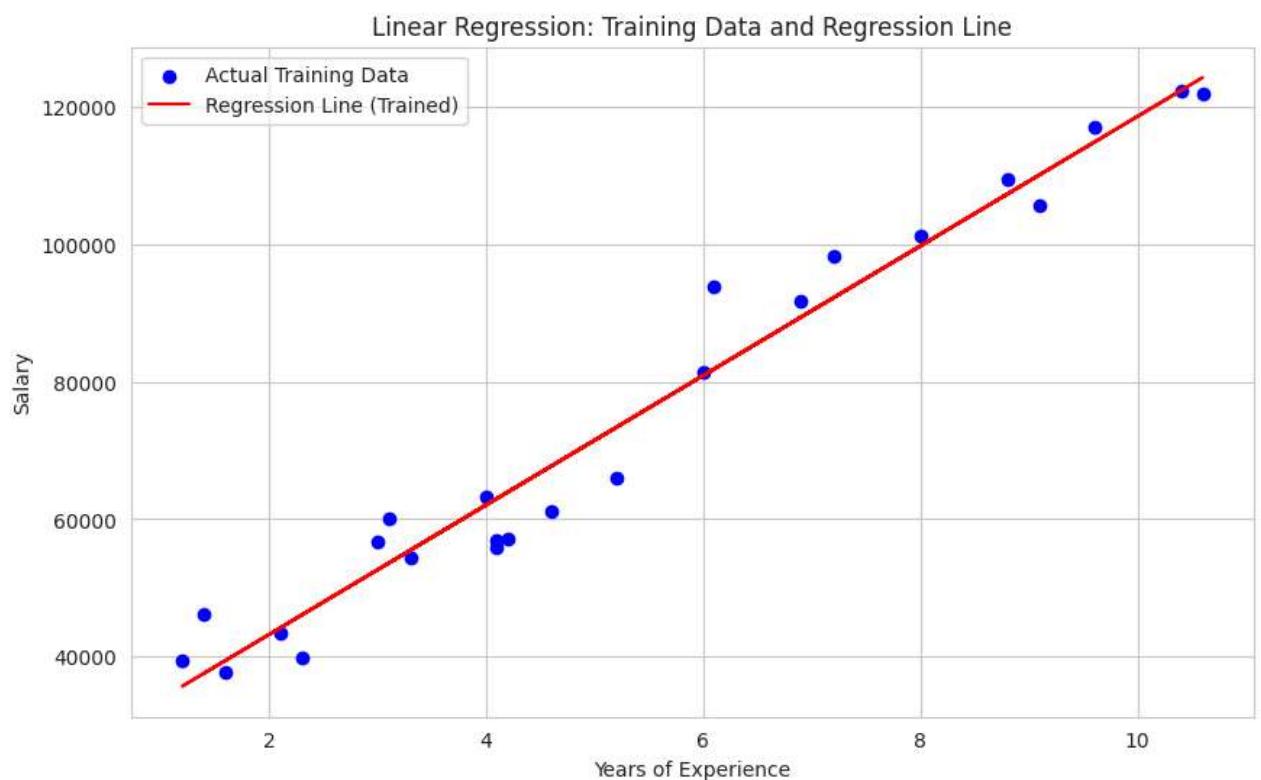
Reasoning: To visualize the regression results as requested, I will generate two plots: one for the training data with the regression line and another for the test data comparing actual and predicted values. This will involve importing `matplotlib.pyplot` and `seaborn` and using the trained `linear_regressor` to make predictions for plotting.

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

# Set aesthetic style for plots
sns.set_style("whitegrid")

# 1. Plotting the regression line on training data
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Actual Training Data')
plt.plot(X_train, linear_regressor.predict(X_train), color='red', label='Regression Line (Trained Model)')
plt.title('Linear Regression: Training Data and Regression Line')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()

# 2. Plotting actual vs. predicted salaries on the test set
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Test Salaries')
plt.scatter(X_test, y_pred, color='green', label='Predicted Test Salaries')
plt.title('Linear Regression: Actual vs. Predicted Salaries on Test Set')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```



Final Task

Subtask:

Summarize the key findings from the linear regression analysis, including the model's accuracy as indicated by MSE and R-squared, and any insights derived from the visualizations.

Summary:

Q&A

The key findings from the linear regression analysis indicate that the model performs well in predicting salary based on years of experience. The model achieved a Mean Squared Error (MSE) of approximately 49,830,096.86 and a strong R-squared (R2) score of 0.90. This R2 score suggests that 90% of the variance in salary can be explained by the 'YearsExperience' feature. Visualizations further confirm the strong linear relationship between 'YearsExperience' and 'Salary', with the regression line closely fitting the training data and predicted salaries aligning well with actual test salaries.

Data Analysis Key Findings

- The 'abhishek14398/salary-dataset-simple-linear-regression' dataset was successfully downloaded and unzipped, revealing the primary CSV file named `Salary_dataset.csv`.
- The dataset, once loaded, contained 30 entries and 3 columns, including 'YearsExperience' and 'Salary', with no missing values identified.
- The data was successfully split into training (24 samples) and testing (6 samples) sets.
- A `LinearRegression` model was successfully trained using the training data.
- The model achieved a Mean Squared Error (MSE) of approximately 49,830,096.86 on the test set.
- The R-squared (R2) score for the model on the test set was 0.90, indicating that 90% of the variance in salary can be explained by years of experience.
- Visualizations confirmed a clear linear relationship between 'YearsExperience' and 'Salary', and the regression line effectively captured this trend, with predicted test salaries closely matching actual test salaries.

Insights or Next Steps

- The high R-squared value demonstrates that 'YearsExperience' is a very strong predictor of 'Salary' in