ICP-3

Kavya Reddy Nagulapally
700759486

Github link: https://github.com/Kavyareddy03/ICP-3

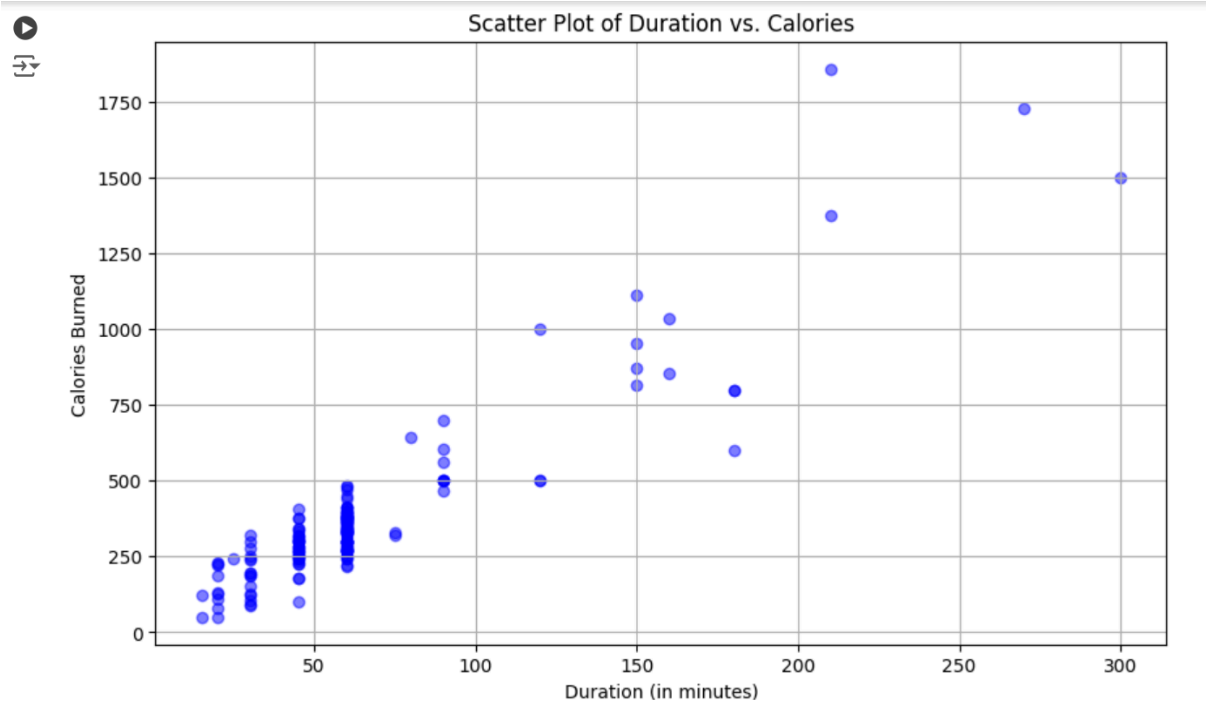
Question 1

Code:

```
import pandas as pd

# Load the CSV file into a DataFrame
df = pd.read_csv('data.csv')

# Display the basic statistical description about the data
basic_stats = df.describe()

# Check if the data has null values
null_values = df.isnull().sum()

# Replace the null values with the mean of their respective column
df_filled = df.fillna(df.mean())

# Aggregating data for at least two columns: min, max, count, mean
selected_columns = df_filled[['Calories', 'Pulse']]  # Example: selecting 'Calories' and 'Pulse'
aggregated_data = selected_columns.agg(['min', 'max', 'count', 'mean'])

# Filter the dataframe for calories values between 500 and 1000
filtered_500_1000 = df_filled[(df_filled['Calories'] >= 500) & (df_filled['Calories'] <= 1000)]

# Filter the dataframe for calories values > 500 and pulse < 100
filtered_calories_pulse = df_filled[(df_filled['Calories'] > 500) & (df_filled['Pulse'] < 100)]

# Creating a new "df_modified" dataframe without the "Maxpulse" column
df_modified = df_filled.drop(columns=['Maxpulse'])
```

```
# Deleting the "Maxpulse" column from the main df dataframe
df.drop(columns=['Maxpulse'], inplace=True)

# Convert the datatype of Calories column to int
df['Calories'] = pd.to_numeric(df['Calories'], downcast='integer', errors='coerce')
df['Calories'].fillna(df['Calories'].mean(), inplace=True)  # Re-fill if any NaN introduced by coercion


basic_stats, null_values, aggregated_data, filtered_500_1000.shape, filtered_calories_pulse.shape, df_modified.head(), df.head()

import matplotlib.pyplot as plt

# Creating a scatter plot for the Duration and Calories columns
plt.figure(figsize=(10, 6))
plt.scatter(df['Duration'], df['Calories'], color='blue', alpha=0.5)
plt.title('Scatter Plot of Duration vs. Calories')
plt.xlabel('Duration (in minutes)')
plt.ylabel('Calories Burned')
plt.grid(True)
plt.show()
```

Output:



Explanation:

Load Data: It reads a CSV file into a DataFrame for analysis.
Basic Statistics: It generates statistical summaries (like mean, min, max) for numerical columns.
Null Values: It checks for missing values in the data and counts them.
Handle Missing Values: It fills in missing values with the mean of their respective columns.
Data Aggregation: It calculates aggregated statistics (min, max, count, mean) for specified columns ('Calories' and 'Pulse').
Filtering Data: It filters the data to include only rows with 'Calories' between 500 and 1000.
It also filters rows where 'Calories' is over 500 and 'Pulse' is under 100.
Modify DataFrame: It creates a new DataFrame by removing a column ('Maxpulse'). It also removes this column from the original DataFrame.
Data Type Conversion: It converts the 'Calories' column to integer type and handles any errors by filling in missing values with the mean.
Visualisation: It plots a scatter plot of 'Duration' versus 'Calories' to visualize their relationship.

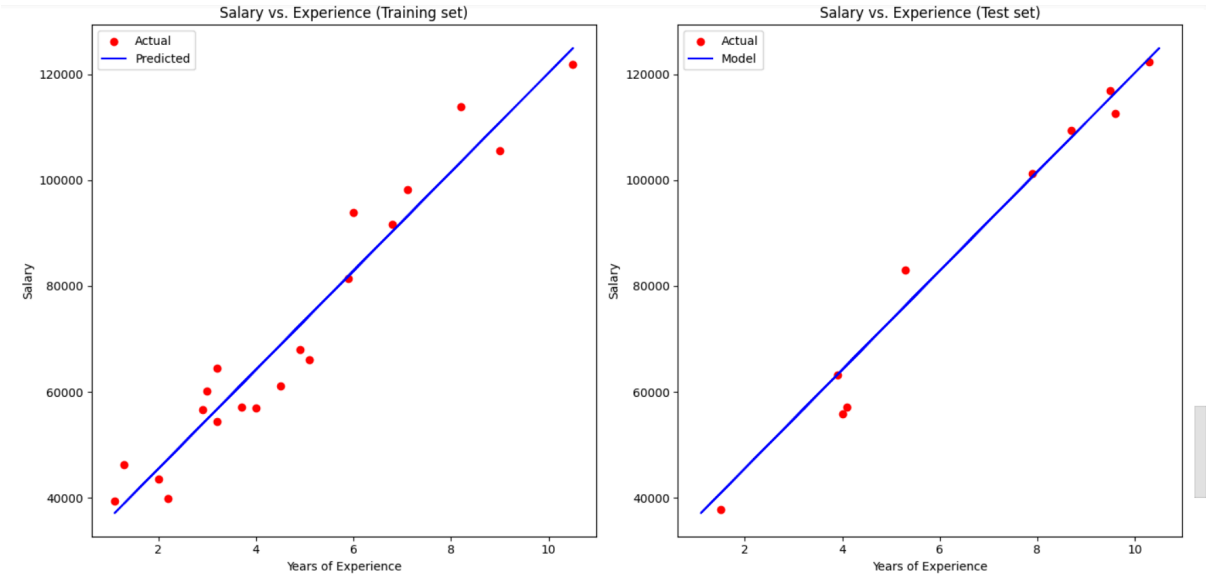
Question 2

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset
salary_data = pd.read_csv('Salary_Data.csv')  # Replace with your file path

# Splitting the dataset into training and testing sets (1/3 for testing)
X = salary_data.iloc[:, :-1].values  # Features (Years of Experience)
y = salary_data.iloc[:, -1].values   # Target (Salary)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

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

# Predicting the Test set results
y_pred = model.predict(X_test)

# Calculating the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Visualizing the Training set results
plt.figure(figsize=(14, 7))
```

```
plt.subplot(1, 2, 1)
plt.scatter(X_train, y_train, color='red', label='Actual')
plt.plot(X_train, model.predict(X_train), color='blue', label='Predicted')
plt.title('Salary vs. Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()

# Visualizing the Test set results
plt.subplot(1, 2, 2)
plt.scatter(X_test, y_test, color='red', label='Actual')
plt.plot(X_train, model.predict(X_train), color='blue', label='Model')
plt.title('Salary vs. Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()

plt.tight_layout()
plt.show()
```

Output:



Explanation:

1. `pd.read_csv()`: Reads data from a CSV file and converts it into a DataFrame. This function helps in importing structured data from a file for further analysis.

2. `iloc[ ]`: Allows for integer-based indexing to select rows and columns from a DataFrame. This function is useful for slicing data and extracting specific portions based on position.

3. `train_test_split()`: Splits data into training and testing sets. This function is crucial for creating a train-test split to evaluate model performance on unseen data.

4. `LinearRegression()`: Initialises a Linear Regression model. This function sets up an instance of the linear regression model that can be trained on data.

5. `fit()`: Trains the model using the training data. This function adjusts the model parameters based on the provided features and target values to fit the data.

6. `predict()`: Makes predictions using the trained model. This function generates predictions based on the test data.

7. `mean_squared_error()`: Computes the Mean Squared Error between actual and predicted values. This function measures the average squared difference between the actual and predicted values, providing a performance metric for the model.

8. `print()`: Outputs information to the console. This function displays the calculated Mean Squared Error for the user to review.

9. `plt.figure()`: Creates a new figure for plotting. This function initialises a new plotting area with specified dimensions, allowing for customization of the plot layout.

10. `plt.subplot()`: Adds a subplot to the current figure. This function divides the figure into a grid and specifies the location of individual plots, facilitating the organisation of multiple plots in one figure.

11. `plt.scatter()`: Creates a scatter plot. This function plots individual data points, which helps in visualising the distribution and relationships between variables.

12. `plt.plot()`: Plots lines or markers on a graph. This function is used to draw lines or curves connecting data points, often used for visualising trends or model predictions.

13. `plt.title()`: Sets the title for the plot. This function adds a title to the plot, providing context or explanation for what the plot represents.

14. `plt.xlabel()`: Labels the x-axis of the plot. This function adds a descriptive label to the x-axis to indicate what variable or data the axis represents.

15. `plt.ylabel()`: Labels the y-axis of the plot. This function adds a descriptive label to the y-axis to indicate what variable or data the axis represents.

16. `plt.legend()`: Displays a legend on the plot. This function adds a legend to identify different data series or elements in the plot, helping to differentiate between various plotted elements.

17. `plt.tight_layout()`: Adjusts subplot parameters to give specified padding. This function ensures that subplots do not overlap and that the layout is optimised for visibility.

18. `plt.show()`: Displays the plot on the screen. This function renders the figure and plots, making them visible to the user.