`upload_to_s3 Function`

1. `Define the function upload_to_s3(data, filename).`
2. `Create an in-memory buffer using StringIO to store the CSV data.`
3. `Convert the data to CSV format and write it to the buffer.`
4. `Use the S3 client to upload the content of the buffer to S3.`
5. `The CSV file is uploaded to the specified S3 bucket with the given filename.`

`download_from_s3 Function`

1. `Define the function download_from_s3(filename).`
2. `Use the S3 client to get the object from the specified S3 bucket with the given filename.`
3. `Read the CSV file from the object body into a pandas DataFrame.`
4. `Return the DataFrame containing the data from the CSV file.`

---

load_data Function

1. Define the function `load_data()`.
2. Download the heart disease dataset from S3 using the `download_from_s3` function.
3. Check if the DataFrame is empty: If the DataFrame is empty, print "DataFrame is empty!". If the DataFrame has data, print "DataFrame has data.".
4. Download the heart disease CSV file from S3 to a local file using the S3 client.
5. Read the CSV file into a Spark DataFrame using Spark's `read.csv()` method.
6. Check if the Spark DataFrame is empty: 8. If the Spark DataFrame is empty, print "Spark DataFrame is empty!". 9. If the Spark DataFrame has data, print "Spark DataFrame has data.".

Pseudocode for `sklearn_impute_smoking_1` function:

1. Define the function `sklearn_impute_smoking_1(data_source, data_frame)`.
2. Fetch the data from the provided source.
3. IF data retrieval is successful:
4. Parse the data to extract relevant information.

5. Identify and extract the necessary data.
6. Initialize an empty dictionary to store the relevant information.
7. FOR each data entry:
8. Extract and process the required values.
9. Store the processed data in the dictionary.
10. Add a new column to the `data_frame` with initial values.
11. Define a helper function to map data to corresponding categories.
12. Define a function to fill missing values based on the mapping.
13. IF the value is missing:
14. Use the helper function to determine the corresponding category.
15. IF the category exists in the dictionary:
16. Return the corresponding value from the dictionary.
17. ELSE:
18. Return a default value (e.g., None).
19. ELSE:
20. Return the existing value.
21. Apply the function to the entire `data_frame` to fill missing values.
22. Return the updated `data_frame`.

Pseudocode for `sklearn_impute_smoking_2` function:

1. Set smoking_rate_female to 0.1
2. Set smoking_rate_male to 0.132
3. Set smoking_rate_by_age dictionary with rates for '18–24', '25–44', '45–64', '65 and above'
4. Create a new 'smoke_source_2' column in df_cleaned, initialized with the 'smoke' column values
5. FUNCTION get_age_group_sklearn_2(age):
6. IF age is between 18 and 24:
7. Return '18–24'
8. ELSE IF age is between 25 and 44:
9. Return '25–44'
10. ELSE IF age is between 45 and 64:
11. Return '45–64'
12. ELSE IF age is 65 or older:
13. Return '65 and above'
14. Return None
15. FUNCTION impute_smoking_rate_sklearn_2(row):
16. IF 'smoke_source_2' is missing for this row:
17. Map age to the corresponding age group using get_age_group_sklearn_2
18. IF age group exists in smoking_rate_by_age:
19. IF sex is female (0):

20. Return the smoking rate for the corresponding age group
21. ELSE IF sex is male (1):
22. Return the smoking rate adjusted by the male-to-female ratio
23. Apply impute_smoking_rate_sklearn_2 function to 'df_cleaned'
24. Return df_cleaned

Pseudocode for `clean_impute_sklearn` function:

1. Download 'heart_disease.csv' from S3 and load it into df
2. Retain columns: 'age', 'sex', 'painloc', 'painexer', 'cp', 'trestbps', 'smoke', 'fbs', 'prop', 'nitr', 'pro', 'diuretic', 'thaldur', 'thalach', 'exang', 'oldpeak', 'slope', 'target'
3. Impute missing values in 'painloc' and 'painexer' with the mode
4. Clip values in 'trestbps' to a minimum of 100
5. Clip 'oldpeak' values between 0 and 4
6. Impute missing 'thaldur' and 'thalach' with their mean
7. Impute missing values and clip values for 'fbs', 'prop', 'nitr', 'pro', 'diuretic'
8. Impute missing 'exang' and 'slope' with the mode
9. Impute missing 'age', 'sex', 'cp', 'trestbps', 'oldpeak', and 'target' values with their mode or mean
10. Retain only the first 899 rows of df_cleaned
11. Call sklearn_impute_smoking_1 with the URL and df_cleaned
12. Call sklearn_impute_smoking_2 with df_cleaned
13. Upload the final cleaned dataframe to S3 as "sklearn_cleaned_data.csv"

Pseudocode for `pyspark_impute_smoking_1` function:

1. **Fetch Web Page Data**:
   - Make a GET request to the provided URL.
   - Check if the request was successful (status code 200).
   - Parse the webpage content using BeautifulSoup.
2. **Extract Smoking Rate Data**:
   - Find the table containing the relevant data.
   - Initialize an empty dictionary `smoking_rate_by_age`.
   - Loop through each row of the table (skipping the header row).
   - For each row:
     - Extract the age group.
     - Extract the second percentage value.

        - Store the smoking rate (converted to a decimal) in the dictionary, with the age group as the key.
3. **Impute Missing Smoking Data**:
    - Add a new column `smoke_source_1` to the DataFrame and copy values from the existing `smoke` column.
4. **Map Age to Age Group**:
    - Define a function `get_age_group` to map the age to a corresponding age group based on predefined ranges.
5. **Impute Missing Values**:
    - Define a function `impute_smoking_rate` to impute missing `smoke` values:
        - If `smoke_value` is missing, map the `age` to an age group and fetch the corresponding smoking rate.
        - If `smoke_value` is not missing, return the original value.
6. **Apply the Imputation Function**:
    - Convert `impute_smoking_rate` to a User Defined Function (UDF) and apply it to the `smoke_source_1` column of the DataFrame, using the `age` column for imputation.
7. **Return Data**:
    - Show the first 5 rows of the updated DataFrame.
    - Return the DataFrame with the imputed `smoke_source_1` values.

Pseudocode for `pyspark_impute_smoking_2` function:

1. **Define Hardcoded Smoking Rates**:
    - Set the smoking rates for females and males (`smoking_rate_female`, `smoking_rate_male`).
    - Define a dictionary `smoking_rate_by_age` for the smoking rates by age group.
2. **Impute Missing Smoking Data**:
    - Add a new column `smoke_source_2` to the DataFrame and copy values from the existing `smoke` column.
3. **Map Age to Age Group**:

    - Define a function `get_age_group` to map the `age` to an age group based on predefined ranges.
4. **Impute Missing Values Based on Sex**:
    - Define a function `impute_smoking_rate` to impute missing `smoke` values:
        - If `smoke_value` is missing:
            - Map `age` to an age group.
            - Impute the smoking rate for females (`sex == 0`) or males (`sex == 1`), adjusting for gender difference.
        - If `smoke_value` is not missing, return the original value.
5. **Apply the Imputation Function**:
    - Convert `impute_smoking_rate` to a UDF and apply it to the `smoke_source_2` column of the DataFrame, using `age` and `sex` columns for imputation.
6. **Replace Missing Values in `smoke_source_2`**:
    - If `smoke_source_2` is still missing, fill it with values from the original `smoke` column.
7. **Return Data**:
    - Show the first 5 rows of the updated DataFrame.
    - Return the DataFrame with the imputed `smoke_source_2` values.

Pseudocode for `clean_impute_pyspark` function:

1. **Download and Read Data**:
    - Download the dataset (`heart_disease.csv`) from S3 to the local file system.
    - Read the CSV file into a Spark DataFrame (`df_cleaned`).
2. **Select Relevant Columns**:
    - Define a list of columns to retain.
    - Select only the columns in the list.
3. **Impute Missing Values for `painloc`**:
    - Calculate the mode (most frequent value) of `painloc`.
    - Fill missing values in the `painloc` column with the mode.
4. **Impute Missing Values for `painexer`**:

- Calculate the mode of `painexer`.
- Fill missing values in the `painexer` column with the mode.

5. **Impute `trestbps` Values**:

    - Replace values in `trestbps` that are less than 100 with 100.

6. **Impute `oldpeak` Values**:

    - Replace values in `oldpeak` that are less than 0 with 0 and values greater than 4 with 4.

7. **Impute `thaldur` and `thalach` Values**:

    - Calculate the mean values for `thaldur` and `thalach`.
    - Fill missing values in these columns with the respective mean values.

8. **Impute Missing Values for Selected Columns**:

    - Define a list of columns (`fbs`, `prop`, `nitr`, `pro`, `diuretic`).
    - Loop through each column:
        - Calculate the mode of the column.
        - Fill missing values with the mode.
        - Clip values greater than 1 to 1.

9. **Impute `exang` and `slope` Values**:

    - Calculate the mode for the `exang` and `slope` columns.
    - Impute missing values in these columns with the respective modes.

10. **Impute Missing Values for Other Columns**:

    - Define a list of columns (`age`, `sex`, `cp`, `trestbps`, `target`).
    - Loop through each column:
        - Calculate the mode of the column.
        - Fill missing values with the mode.

11. **Impute `oldpeak` with Mean**:

    - Calculate the mean of `oldpeak`.
    - Fill missing values in the `oldpeak` column with the mean.

12. **Limit the Number of Rows**:

    - Limit the DataFrame to the first 899 rows.

13. **Return Data**:

    - Show the first 5 rows of the cleaned DataFrame.
    - Return the cleaned DataFrame.

Pseudocode for `feature_engineering_1`:

1. **Download dataset** from S3 (sklearn_cleaned_data.csv).
2. **Calculate max heart rate**:
   - For each record, compute max heart rate as `206.9 - (0.67 * age)`.
   - Add this value to a new column called "max_HR".
3. **Upload** the updated dataset to S3 with a new filename (fe_data_1.csv).

Pseudocode for `feature_engineering_2`:

1. **Download dataset** from S3 (pyspark_cleaned_data.csv) to a local file.
2. **Read dataset** into a Spark DataFrame.
3. **Calculate blood pressure difference** from normal:
   - For each record, subtract 120 from the "trestbps" column and store it in a new column "bp_diff_from_norm".
4. **Upload** the updated DataFrame to S3 as a Pandas DataFrame (fe_data_2.csv).

Pseudocode for `train_svm_fe1`:

1. **Download dataset** from S3 (fe_data_1.csv).
2. **Prepare features and target**:
   - Drop "target" and "smoke" columns from features (as "smoke" contains NaN values).
   - Convert all feature columns to numeric values.
   - Set "target" as the label (y).
3. **Split data** into training and test sets (90-10 split with stratification).
4. **Train SVM model** using cross-validation (5-fold) and hyperparameter tuning (C and kernel).
5. **Evaluate** the model on the test set using accuracy score.
6. **Push the accuracy** to XCom.

Pseudocode for `train_logistic_fe1`:

1. **Download dataset** from S3 (fe_data_1.csv).
2. **Prepare features and target**:
   - Drop "target" and "smoke" columns from features (as "smoke" contains NaN values).

    - Convert all feature columns to numeric values.
    - Set "target" as the label (y).
3. **Split data** into training and test sets (90-10 split with stratification).
4. **Train Logistic Regression model** using cross-validation (5-fold) and hyperparameter tuning (C).
5. **Evaluate** the model on the test set using accuracy score.
6. **Push the accuracy** to XCom.

Pseudocode for `train_svm_fe2`:

1. **Download dataset** from S3 (fe_data_2.csv) to a local file.
2. **Read dataset** into a Spark DataFrame.
3. **Drop the "smoke" column** from the dataset.
4. **Prepare features**:
    - Convert all feature columns to double type.
    - Create a "features" column by combining the feature columns into a vector using `VectorAssembler`.
5. **Split data** into training and test sets (90-10 split).
6. **Train SVM model** using cross-validation (5-fold) and hyperparameter tuning (regParam).
7. **Evaluate** the model using AUC and accuracy score.
8. **Push the accuracy** to XCom.

Pseudocode for `train_logistic_fe2`:

1. **Download dataset** from S3 (fe_data_2.csv) to a local file.
2. **Read dataset** into a Spark DataFrame.
3. **Drop the "smoke" column** from the dataset.
4. **Prepare features**:
    - Convert all feature columns to double type.
    - Create a "features" column by combining the feature columns into a vector using `VectorAssembler`.
5. **Split data** into training and test sets (90-10 split).
6. **Train Logistic Regression model** using cross-validation (5-fold) and hyperparameter tuning (regParam).
7. **Evaluate** the model using AUC, F1 score, and accuracy.
8. **Push the accuracy** to XCom.

Pseudocode for `merge_results`:

1. **Retrieve accuracies** from XCom for all models (SVM_FE1, Logistic_FE1, SVM_FE2, Logistic_FE2).
2. **Filter out invalid accuracies** (None values).
3. **Identify the best model** based on the highest accuracy.
4. **Push the best model** and its accuracy to XCom.

Pseudocode for `evaluate_test`:

1. **Retrieve the best model** and its accuracy from XCom.
2. **Print the final model** and its accuracy.