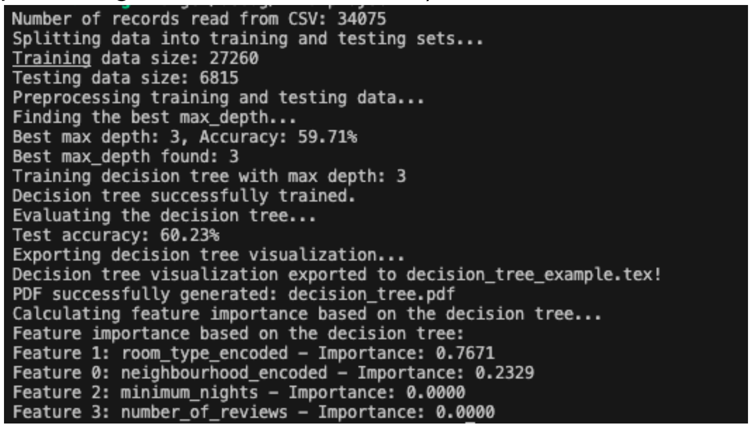
DS210 final project write-up

Name: Chih-Ting (Karina) Yang **Collaborator:** None
Sources: Lecture Note, ChatGPT for testing data sample

- **Source:** New York City Airbnb Open Data https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data/data
    - 48896 rows; 34075 rows after cleaning
- **Research question**: What factors contribute to a listing being in the high-price category?
- **Main goal:** To train a decision tree model and evaluate its performance, particularly determining the most important price predictors.
- **How to run it**: I already wrote files for data preprocessing, trained and tested the decision tree, and identified the importance of the feature. Therefore, the command of "cargo run" can generate the expected outcome.
    - Using the command "cargo test" will run the testing code in the main file
- **Output expectation**:
    - the folder will contain a cleaned data file and a PDF/LaTeX output because I export the tree to LaTeX for visualization (a .tex file will be generated) and compile it with pdflatex to display the decision tree.
    - In the terminal, the progress with explanation is printed. It will include
        - Total size of CSV
        - training/testing data size
        - Result of the best max depth
        - Accuracy of the trained decision tree
        - The list of important features and their importance (not converted into percentages but 0.7671=76.01%)



```
Number of records read from CSV: 34075
Splitting data into training and testing sets...
Training data size: 27260
Testing data size: 6815
Preprocessing training and testing data...
Finding the best max_depth...
Best max depth: 3, Accuracy: 59.71%
Best max_depth found: 3
Training decision tree with max depth: 3
Decision tree successfully trained.
Evaluating the decision tree...
Test accuracy: 60.23%
Exporting decision tree visualization...
Decision tree visualization exported to decision_tree_example.tex!
PDF successfully generated: decision_tree.pdf
Calculating feature importance based on the decision tree...
Feature importance based on the decision tree:
Feature 1: room_type_encoded - Importance: 0.7671
Feature 0: neighbourhood_encoded - Importance: 0.2329
Feature 2: minimum_nights - Importance: 0.0000
Feature 3: number_of_reviews - Importance: 0.0000
```

- **Outcome analyze:**
    - Room Type is the most important factor influencing an Airbnb's price while the neighborhood is the second most important one.
- **Model selection**:
    - I chose a decision tree to train and test to identify the factors that contribute to Airbnb's price because it provides clear interpretability, allowing us to understand how different factors, such as neighborhood, room type, and other amenities, impact the price. Additionally, decision trees can capture complex interactions between features, making them a strong choice for this type of analysis.
- **Future**:

    - Since the model's accuracy is 60.23% which is not very high. We should try other methods such as regression analysis, random forest, and other methods to train and test the dataset.
- **Dependencies**
    - [dependencies]
    - linfa = "0.6.0"
    - linfa-trees = "0.6.0"
    - ndarray = "0.15"
    - rand = "0.8"
    - csv = "1.1"
    - serde = { version = "1.0", features = ["derive"] }

In this project, I use four different files: main, data_prep, decision_tree, and further_eval.

Data_prep: I tried different ways to clean the dataset and encode them. It is the final version.
Main task:

- Load the dataset and create a new cleaned dataset
- Determine the variables I want to include
    - Drop id, name, neighbourhood, host_id, host_name, last_review, host_listing, and availability_365
- Determine the outlier and remove them from the dataset
- For missing data, drop rows if 'price' and 'neighborshood_group' is miss
- Map neighbourhood_group and room types
- Assign our target, dependent variable price, into low, medium, and high categories

1. AirbnbRecord struct
    a. It represents an individual record in the dataset that I will use for this project. I also create new fields to store the encoded values of the features for machine learning models, including neighbourhood_group_encoded, room_type_encoded, and price_category.
    b. I removed the neighborhood column for this project because the high correlation between neighbourhood_group and neighborhood may cause multicollinearity.
2. Function encode_features:
    a. encodes categorical features
        i. Neighbourhood_group uses a map, which is another function of neighbourhood_group_map, to convert neighborhood names into numerical values
        ii. Here, I directly convert room type to a numerical value, where 0 for "Entire home/apt", 1 for "Private room", etc.
    b. assigns price category based on percentiles to low, medium, and high
3. Function calculate percentile:
    a. The function is used for calculating percentile value for a given list of data
    b. It is a helper for calculating the appropriate rank based on the percentile argument for both price range and other variable's outlier
4. Function load_and_clean_data
    a. The function loads, cleans, and encodes the data.
    b. Using ReaderBuilder and WriterBuilder, it can reads data from a CSV file and writes cleaned and encoded data in a vector.
    c. To specify the variables I want to include in the new CSV, I write the heads for the cleaned data file. Then, I initialize each variable's vector by using Vec::new().

    d. for result in rdr.deserialize(): This iterates over each record in the CSV file. The deserialize() function reads each row and deserializes it into an AirbnbRecord struct.
        i. record.neighbourhood_group is an optional value (Option<String>). The if let Some(ref group) syntax checks if the neighbourhood_group field is Some, meaning it has a value. If it does, it clones the value and inserts it into the neighbourhood_group_set.
        ii. If record.price is Some, meaning the price field has a value, the code pushes that value to the prices vector.
        iii. This step is to ensure every entry does not have missing data while adding the data to the vector
    e. Next, I calculated the IQR and bounds for outlier detection and applied them to min_nights, num_reviews, price, and availability. Right after this, I filter those data to make the dataset have a valid range
    f. Since I want to encode neighbourhood_group and price, I create a mapping from neighborhood groups that is stored in neighbourhood_group_set to numeric values for encoding purposes.
        i. ".enumerate()" is necessary here to give each element of the iterator a tuple (index, value). Lastly, it collects the resulting tuples into the Hashmap.
    g. To categorize the price range well, I calculate the 33rd and 66th percentiles of the prices vectors, using them to categorize the price into "low", "medium", and "high" categories.
    h. for record in &mut data { record.encode_features(&neighbourhood_group_map, &price_percentiles); }
        i. using the previously created encoding map (neighbourhood_group_map) and price percentiles (price_percentiles), the for loop last iterates over each record in the data vector and encodes its features
    i. The final step of data_prep is to write the data to a new CSV file after cleaning and encoding.

In the decision tree module

Main task:

- Train and test the decision tree
    - Normalize the data and find the best hyparameter in terms of the max depth of the decision tree
- Evaluate the decision tree
- Export the decision tree visualization

1. Struct AirbnbCleanedRecord: defines the structure of a row in a dataset and automatically maps CSV columns (cleaned data file) to this struct
2. Function process_csv_file uses csv::Reader to reads the cleaned file and deserialize its row into a vector of AirbnbCleanedRecord
3. Function split_data splits the data into training and testing sets by shuffling the data using rand::seq::SliceRandom based on the train_ratio
    a. Need to shuffle the data randomly to avoid biases
    b. Train_ratio is provided in the main file's function but I create a variable called split_index
        i. let split_index = (records.len() as f64 * train_ratio) as usize
        ii. let train_set = records[..split_index].to_vec(), which shuffled data up to split_index

            iii. let test_set = records[split_index..].to_vec(), which slices the remaining data to form the test set

4. Function scale_features (learn this online)
    a. Before processing our data for training, although decision trees are less affected by feature scaling, scaling can still enhance interpretability and consistency. Most importantly, it allows features with higher magnitudes such as minimum_nights or numer_of_reviews to not dominate over other features
    b. For the function, I use min-max scaling to transform each column.
        i. let min = features.fold_axis(Axis(0), f64::INFINITY, |&a, &b| a.min(b)); let max = features.fold_axis(Axis(0), f64::NEG_INFINITY, |&a, &b| a.max(b));
        ii. Axis(0) specifies the operation is performed column-wise
        iii. Fold_axis is a method to aggregate values along the axis
        iv. f64::INFINITY means infinity is the initial value to find the minimum
        v. Lastly, the closure compares values to compute the min/max
    c. calculate e the range of each column
    d. Scale each row by iterating over each row
        i. converting them to an owned Array1<f64>, subtracting the min values, and dividing the adjusted row values by the range

$$\text{scaled\_row} = \frac{\text{row} - \text{min}}{\text{range}}$$

    e. Lastly, flatten the 2D vector into a single vector of f64. Here, I clone the individual elements during flattening so the data is no longer borrowed
    f. Array2::from_shape_vec: Reshapes the flattened vector back into a 2D array with the original dimensions (data points - rows and features - column)
5. Function preprocess_data
    a. Here, I convert the AirbnbCleanedRecord vector into features (encoded and scaled features) and targets (price_category where 0 for “low”, 1 for “medium”, 2 for “high”) in the array
    b. Scale_features function is used here to scale the features
6. Function train_decision_tree trains a decision tree using linfa_trees.
    a. It converts the features and target into a dataset first. Then, I set the hyperparameters, including max_depth and min_impurity_decrease.
        i. Tuning hyperparameters is important, so the next function is to find best max depth.
    b. Lastly, fits the tree to the training dataset.
7. Function find_best_max_depth
    a. It finds the best maximum depth for the decision tree by training multiple decision trees with varying max_depth values and evaluating accuracy for each depth using for loop.
    b. The variable best_depth and best_accuracy are initialized first for recording the update for each loop.
8. Function evaluate_decision_tree
    a. After training the decision tree with best maximum depth, the function is to evaluate the decision tree’s performance.
        i. It predicts target values for test features first
        ii. Then, compares the prediction with actual test targets (|&(pred, actual)| pred == actual)

                1. I use .zip() to pair each predicted value with its corresponding actual value, and .filter() to only count the cases where the prediction matches the actual target

        iii. Lastly, it calculates the accuracy as the percentage of correct prediction

9. Function export_decision_tree (follow the code on the lecture note)
    a. First, I try to export the trained decision tree as a LaTex document using linfa_trees. However, I want to have a clear visualization that allows readers and me to read easily.
    b. Hence, I generate a TikZ representation, which can be compiled into a PDF
10. Function compile_to_pdf (follow the code on the lecture note)
    a. I execute pdflatex to compile the LaTex file into a PDF
    b. Outputs the resulting visualization as decision_tree.pdf

In the feature importance module

Main task:

- Retrieve the feature importance of a pre-trained decision tree
- Get the outcome of a sorted list of tuples that list their features and importance value

1. Function get_decision_tree_feature_importance
    a. I use linfa_trees and its function feature_importance to retrieve the feature importance of the trained decision tree
    b. Then, I map the feature importance to a vector of feature index and its importance value, so the return is more organized.
    c. Next, I use sort_by and partial_cmp to make a comparison between the feature importance, so they can present in descending order based on the value
2. Function get_feature_names
    a. The next function is to retrieve the feature names based on the feature indices, so it is easier to interpret
    b. I use HashMap to define feature names corresponding to their indices

In the main file

Main task:

- Include test to ensure every function works
- Structure to load, preprocess, train, and evaluate the decision tree model using data from the dataset I chose “AB_NYC_2019.csv”

1. Use the mod keyword to include the other three modules from the file
2. Loading raw data and creating a new file path
    a. two file paths: raw_file (the raw data) and cleaned_file (the cleaned data)
    b. calls the load_and_clean_data function to load and clean the raw data, writing the cleaned data to the cleaned_file. The result of this operation is ignored by using _ =.
3. Using the function from decision_tree module
    a. Process_csv_file enables loading and preprocessing data, where deserialize the cleaned file into a list of AirbnbCleanedRecord structs
    b. Split_data split the loaded records into training and testing set. Here, I clarify the ratio as 0.8
    c. Preprocess_data extract features and targets from both the training and testing records.
    d. Find_best_max_depth train decision trees for each depth in the range I declare (3 to 20) and evaluate their performance on the training data

   e. Train_decision_tree uses the best max depth found in the previous step to train the decision tree
   f. Lastly, evaluate_decision_tree calculates the accuracy of the model by comparing its predictions to the actual targets in the test dataset; export_decision_tree exports a visualization and use compile_to_pdf to save into a PDF
4. Calling the function from feature importance to retrieve a list of importance values.
   a. To display the feature's importance, for loop is used here. For each feature, it retrieves its name from feature_names and prints its corresponding importance.
5. Lastly, the test module contains two primary tests, which are test_data_pipeline and test_machine_learning_pipeline
   a. Test_data_pipeline verifies the CSV file creation, data loading, and cleaning process.
      i. I ask ChatGpt to generate sample data for me, then I create a test CSV file (test_data.csv) with sample data.
      ii. Calls load_and_clean_data to clean the data and write it to test_cleaned_data.csv.
      iii. Asserts that the data cleaning process is successful and the output file exists.
      iv. Cleans up by deleting the test files (test_data.csv and test_cleaned_data.csv)
      v. To check whether the function works or not, I create the assertion to check load_and_clean_data works and test_cleaned_data.csv exists
   b. Another verifies the entire machine learning pipeline, including preprocessing, decision tree training, and evaluation.
      i. I ask ChatGpt to create a sample set of AirbnbCleanedRecord data again
      ii. Then, I apply the preprocessing, decision tree training, feature importance, and model evaluation function
      iii. The key assertions I create are
         1. check feature and target vector should not be empty
         2. The number of rows in features should match the number of records, and the number of columns should be 4 based on the sample data
         3. The decision tree model should be trained without errors.
         4. Feature importance should not be empty.
         5. The accuracy of the model should be greater than 0.