PHASE 3

Fully Documented Code:

Data cleaning:

- It imports the necessary libraries (pandas, numpy, math, matplotlib, sklearn, seaborn, and datetime).

- It reads in two CSV files, 'calendar.csv' and 'listings.csv', into pandas dataframes called 'calendar' and 'listings', respectively.

- It cleans up the 'calendar' dataframe by converting the 'date' column to a datetime format, replacing the 't' and 'f' values in the 'available' column with 1s and 0s, respectively, and creating new columns for the year, month, day, and day of the week.

- It cleans up the 'calendar' dataframe further by removing duplicate rows, checking for missing values, and converting the 'price' column to a float format (after removing , and $ signs).

- Select useful columns from the 'listings' dataframe and assigns them to a new dataframe called 'listings1'.

- To clean up the 'listings1' dataframe, convert the categorical columns - property_type, room_type, bed_type, and neighbourhood_group_cleansed - to dummy variables. For each unique value in these columns, a new column will be created and the value will be 1 if that value is present for the respective record, and 0 otherwise. This process helps to represent categorical data as numerical features that can be used in machine learning models.

- Convert the 'zipcode' column to a string. For columns property_type, room_type, bed_type, neighbourhood_group_cleansed, instant_bookable, extra_people, bathrooms, bedrooms, beds, reviews_per_month, number_of_reviews, and guests_included if value of not available it is replaced by 0 so that it will not affect the selected models for analysis.

- Two data frames are created for each listings1 and calendar1 for further analysis. listings1 has real world data, listing2 has normalized data, calendar1 has real world data and calendar2 has normalized data. To normalize the data maximum and minimum values of columns were used and a function called normalization is defined to do that.

- calendar1 and calendar2 , listing_id is renamed to id column.

- listings1 and calendar1 is merged on id column and listings2 and calendar2 is merged on id column. All the not available rows are removed.

EDA:

- Checked the summary statistics of all the numerical columns in the dataset and also checked the data type and number of null values in the dataset.
- Get the maximum number of available listings for each property.
- Get the sum, mean, and median of available listings for each property
- Perform PCA on the data and plot a biplot
- Histogram of bed count: This plot shows the count of different types of beds available in the listings dataset.
- Scatter plot of bedrooms and price: This plot shows the relationship between the number of bedrooms and the price of the listings.
- Plot count of room by room type: This plot shows the count of different types of rooms available in the listings dataset.
- Analysis of the calendar file for price and date variations: This code loads the calendar file and displays its information.
- Plot of average price vs weekday: This plot shows the average price of listings for each day of the week.
- Availability of houses in different zip codes in Seattle with price: This plot shows the relationship between the price and availability of listings in different zip codes in Seattle.
- Histogram to plot total number of listings in the price range: This plot shows the total count of listings in different price ranges.
- Scatter plot to see area-wise price: This plot shows the relationship between the price of the listings and their location on a map of Seattle.
- Scatter plot to see area-wise availability: This plot shows the availability of the listings and their location on a map of Seattle.

Model created by ML, MR, and/or statistical modeling algorithms:

- Import necessary libraries and modules, such as pandas, numpy, and scikit-learn.
- Input features will be saved X. It has columns guests_included, accommodates, extra_people, bathrooms, bedrooms, beds, instant_bookable, reviews_per_month, number_of_reviews, Ballard, Beacon Hill, Capitol Hill, Cascade, Central Area, Delridge, Downtown, Interbay, Lake City, Magnolia, Northgate, Other neighborhoods, Queen Anne, Rainier Valley, Seward Park, University District,West Seattle, Apartment, Bed & Breakfast, Boat, Bungalow,Cabin, Camper/RV, Chalet, Condominium, Dorm, House, Loft, Other, Tent, Townhouse, Treehouse, Yurt, room_type_Entire home/apt, room_type_Private room, room_type_Shared room, Airbed, Couch, Futon, Pull-out Sofa, Real Bed, available, year, month, day.
  Output is saved in Y and is Price.
- Split the data into a training set and a testing set using scikit-learn's train_test_split function.
- Now models are applied to train and test the dataset.

Random Forest

- Modules - RandomForestRegressor, r2_score, mean_squared_error, export_text, and matplotlib.pyplot are imported.
- The RandomForestRegressor function is used to create a random forest model object with hyperparameters - n_estimators=200, max_features='sqrt', criterion='mse', random_state=42, n_jobs=1, and max_depth=10.
- The random forest model is trained on X_train and y_train.squeeze(), and evaluated using mean_squared_error and r2_score metrics on the training and testing data.
- The decision tree structure of the first tree in the random forest is printed using the export_text method with the spacing=3 and decimals=3 arguments.
- The trained random forest model object is saved to a file named 'model2.pkl' using the pickle.dump() method.
- The model's performance is visualized by plotting A histogram of predicted values and scatter plot of predicted values against actual values.

Gradient Boosting Regressor

- Import modules from the scikit-learn library, GridSearchCV and GradientBoostingRegressor.
- Dictionary named param_grid contains hyperparameter ranges for the grid search. GradientBoostingRegressor is instantiated with random_state=42 and passed to GridSearchCV to select the best hyperparameters using cross-validation.
- The fit() method is used to train the model with every hyperparameter combination and evaluate it with cross-validation.
- The best estimator predicts target values for both training and test data, and model performance is evaluated using MSE and R-squared score.
- The model's performance is visualized by plotting A histogram of predicted values and scatter plot of predicted values against actual values.

K-Nearest Neighbors (KNN) Regressor

- Import necessary modules and packages: GridSearchCV, Pipeline, StandardScaler, SelectKBest, f_regression, and KNeighborsRegressor from respective sklearn libraries.
- Define a pipeline using the Pipeline class with a series of steps as tuples.
- Define a dictionary of parameter grids for GridSearchCV with keys as parameter names and values as a list of values to be searched.
- Create an instance of GridSearchCV with pipeline and parameter grid as arguments and set cv parameter for cross-validation.

- Fit the GridSearchCV object to the training data, print best parameters and score, predict on test data, calculate R-squared.
- The model's performance is visualized by plotting A histogram of predicted values and scatter plot of predicted values against actual values.

ElasticNetCV is a linear regression model

- Import necessary modules and packages, including ElasticNetCV from sklearn.linear_model and RFE from sklearn.feature_selection.
- Scale the data using StandardScaler from sklearn.preprocessing.
- Apply feature selection using ElasticNetCV with L1 ratio values [0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1] and cross-validation of 5 folds, using RFE to select 15 features.
- Transform the data using the selected features.
- Define a dictionary of parameter grids for GridSearchCV with 'tol' and 'max_iter' parameters and their corresponding values, create an instance of GridSearchCV with ElasticNetCV and the parameter grid, and fit the GridSearchCV object to the training data. Predict target variable for test data and print the mean squared error (MSE) and R-squared values for both the training and test sets.
- The model's performance is visualized by plotting A histogram of predicted values and scatter plot of predicted values against actual values.

Ridge Regression model with Polynomial Features

- Import modules and packages: PolynomialFeatures, make_pipeline, GridSearchCV, Ridge, StandardScaler, mean_squared_error, r2_score, and MinMaxScaler from sklearn.preprocessing.
- Scale the data using StandardScaler from sklearn.preprocessing. Specify the degree of polynomial features to be used and create a PolynomialFeatures object.
- Define a dictionary of parameter grids for GridSearchCV with the hyperparameters alpha, solver, tol, and max_iter, and their corresponding values.
- Create a pipeline with the PolynomialFeatures object and a Ridge regression model.Use GridSearchCV to find the best hyperparameters for the pipeline.Fit the pipeline to the scaled training data.
- Predict the target variable for the scaled test data and the scaled training data.
- Calculate and print the root mean squared error (RMSE) and R-squared values for both the training and test sets.
- The model's performance is visualized by plotting A histogram of predicted values ,scatter plot of predicted values against actual values and residual value vs the predicted value plot.

Creating an application/model for the project

- Imported libraries: Flask for creating a Flask web application, render_template for rendering HTML templates, request for handling HTTP requests, pickle for loading the trained machine learning model from a binary file, numpy for performing mathematical operations on arrays.

- Created instance of the Flask class with the name app.

- Loaded trained random forest model using the pickle module and saved as model.
- Defined the route for the home page ("/") and rendering the "index.html" template using the render_template function.
- Defined the route for the form submission ("/predict") and handling the POST request using the predict function.
- Retrieved the input values submitted by the user using the request.form object.
- Created a numpy array with all the features got from the user input and reshaped the numpy array to have a shape of (1, number of features) to make it in input shape of random forest model.
- Predicted Price using the model and rounded to 2 decimal places. Returned the predicted price to the "index.html" template as a dictionary with the key "predicted_price".
- If an exception occurs during the prediction process, render the "index.html" template with an error message.

Working instructions to demo/use your finished product

The end product is a web page application of Airbnb in seattle.

Below are the working instructions to help you effectively demo and use our product:

- Access the web page application by running the file vjain24_saritaku_phase_3.ipyn. There will be a link after running the code, it will take you to the GUI interface of web application.
- Once on the web page, you will be presented with a form where you can input the necessary details about Airbnb in Seattle. These details include:
    - Guests included: The number of guests that will be staying at the Airbnb.
    - Accommodates: The number of guests the Airbnb can accommodate.
    - Extra people: The additional cost for extra guests.
    - Bathrooms: The number of bathrooms in the Airbnb.

    - Bedrooms: The number of bedrooms in the Airbnb.
    - Beds: The number of beds in the Airbnb.
    - Instant bookable: A yes or no option to indicate whether the Airbnb is instantly bookable.
    - Reviews per month: The number of reviews the Airbnb receives per month.
    - Number of reviews: The total number of reviews the Airbnb has received.
    - Property type: The type of property the Airbnb is (e.g., apartment , treehouse etc).

- Room type: The type of room in the Airbnb (e.g., shared room, private room, etc.).
- Bed type: The type of bed in the Airbnb (e.g., airbed, real bed, etc)
- Available: The number of days the Airbnb is available for rent.
- Year, month, and day: The date you wish to rent the Airbnb.
- Neighbourhood: The neighbourhood where the Airbnb is located.

- Once you have input all the necessary details, click on the "submit" button.

- Application will analyze the input and predict the price of the Airbnb based on the details provided. The predicted price of the Airbnb will be displayed on the screen. Process can be repeated by inputting different details to predict the prices of other Airbnbs in Seattle.

Requirements to run the application:
local computer needs to have Flask, Sklearn and Random Forest.

Model used:

- The product end up using the Random Forest model. It was trained in the phase two and gave the best accuracy among all the other models. It was saved as 'model1.pkl' using the pickle.dump() method. Random Forest classifier is an ensemble learning model that helps to increase the accuracy.
- Hyperparameter tuning for a RandomForestRegressor model using GridSearchCV. The hyperparameters being tuned are:
  - n_estimators: the number of trees in the forest
  - max_features: the maximum number of features to consider when splitting a node
  - criterion: measure the quality of a split
  - max_depth: the maximum depth of the tree
- GridSearchCV uses cross-validation to evaluate the performance of the model with each combination of hyperparameters and selects the best set of hyperparameters based on scoring metric (r2).
- After selecting the best hyperparameters using GridSearchCV, the code retrains the model with the selected hyperparameters, and calculates the performance metrics (MSE and R^2) on the training and test data using the tuned model.

```
# Define the hyperparameters to search over
param_grid = {
    'n_estimators': [100, 200],
    'max_features': ['auto', 'sqrt'],
    'criterion': ['mse', 'mae'],
    'max_depth': [5, 10]
}
S
# Define the RandomForestRegressor model
forest = RandomForestRegressor(random_state=42)

# Create a grid search object
grid_search = GridSearchCV(forest, param_grid, cv=5, scoring='r2', n_jobs=-1)
```

Use cases and recommendations:

- What can users learn from your product?

  - Users can estimate the price of their listing based on the housing specifications such as bedrooms, beds, bathroom, and neighborhood. This information can be helpful for both renters and hosts to get an idea of what they should charge or expect to pay for a particular property. Users can also use the product to gain insights into the various neighborhoods in Seattle and their average listing prices.

- How does it help them solve problems related to your problem statement?

  - The Airbnb Seattle dataset analysis and model can help users solve problems related to finding the right accommodation in Seattle at the right price. By providing an estimate of the price of an Airbnb listing based on its features such as the number of bedrooms, beds, bathrooms, neighborhood, and other relevant factors, users can make informed decisions about their lodging choices. This can help users save time and money by identifying listings that meet their needs and budget. Additionally, the model can help hosts price their listings appropriately, which can lead to more bookings and better revenue management. Overall, the analysis and model provide valuable insights into the Seattle Airbnb market and can help users make more informed decisions related to their accommodation needs.

- Other ideas for how to extend project, or other avenues that could be explored related to the problem?

  - Extension of the project: The product can be extended by adding more features to the model, length of stay or map showing location of housing, to provide a more accurate price estimate. Additionally, the model can be trained on more recent data to provide up-to-date pricing information.

Citations:

Dataset link
https://www.kaggle.com/datasets/airbnb/seattle
Gradient Boosting Regressor
https://medium.com/mlearning-ai/gradient-boosting-for-regression-from-scratch-bba968c16c57
https://www.geeksforgeeks.org/ml-gradient-boosting/
ElasticNetCV with linear regression model and Modeling in Python
https://medium.com/mlearning-ai/elasticnet-regression-fundamentals-and-modeling-in-python-8668f3c2e39e
Ridge regression model with polynomial features
https://medium.com/@sidharths758/polynomial-regression-overfitting-and-ridge-regression-an-overview-70de53f0ccab
https://www.geeksforgeeks.org/ml-ridge-regressor-using-sklearn/
Flask application tutorial:
https://www.geeksforgeeks.org/flask-creating-first-simple-application/
html tutorial:
https://html.com/