Data Analysis 3: Assignment 2: Technical Report

Mohammad Ahmed

Introduction:

The goal of this report is to provide a detailed description for the price prediction model to help a company in setting a price for their new apartments that are yet to be introduced to the public. Our objective is to derive a price for apartments that accommodate 2-6 guests and are small to mid-sized in San Deigo, United States. The dataset is obtained from Inside Airbnb. The first step in carrying out a decent analysis is to clean the dirty dataset, preparing variables that will be used and to understand the domain. We will be using various prediction models and analyze which one helps us the most and what model provides the best results. The major predictor variables include number of accommodates, total number of beds in the accommodation, number of bathrooms, the neighborhood and various amenities present in the data. Eventually, we will be providing a conclusion on what model gives the best prediction measured by relative RMSE values.

Data Cleaning and Preparation:

The data used is from the date 23rd March 2023 and it refers to one night rental prices. The target variable Is price per night per person in USD. Initially, in the data, we have a lot of information that requires in depth cleaning and formatting since in a lot of columns, there are values that can't be used directly and characters like percentage, currency signs, blank spaces, quotation marks, etc. so they must be fixed first before moving further with the analysis. In addition to that, there are a lot of columns in the data that we do not require therefore, they were dropped. The initial data contains 12871 rows and 75 columns. After dropping various columns like host URL, host picture URL, host thumbnail URL, name, host about, last scraped etc. the new shape of our data is (12871, 46). Then we transformed the amenities column to binary variables so they can be used for plotting and carrying out the analysis.

```
drops = ["name", "listing_url", "last_scraped", "description", "host_thumbnail_url", "neighborhood_overview",
         "host_picture_url", "picture_url", "host_url", "host_about",
         "host_location", "host_total_listings_count", "bathrooms",
         "minimum_minimum_nights", "maximum_maximum_nights", "minimum_maximum_nights", "maximum_minimum_nights",
         "minimum_nights_avg_ntm", "maximum_nights_avg_ntm", "number_of_reviews_ltm",
         "calculated_host_listings_count_entire_homes", "calculated_host_listings_count_private_rooms",
         "calculated_host_listings_count_shared_rooms", "calendar_updated", "review_scores_accuracy",
         "review_scores_checkin", "review_scores_location", "review_scores_value", "review_scores_communication"]

df = df.drop(columns=drops)
```

The below attached code snippets entail the various transformations done in the data to make it useable.

```
df = df[(df['accommodates'] >= 2) & (df['accommodates'] <= 6)]
df = df[df['property_type'].isin(["Apartment", "Condominium", "Serviced apartment", "Entire loft",
                                  "Entire condominium (condo)", "Entire serviced apartment", "Entire home/apt",
                                  "Entire rental unit"])]
```

```
df = df[df['bathrooms_text'] != "null"]
df['bathrooms'] = df['bathrooms_text'].str.extract(r'(\d+\.*\d*)')
df['bathrooms'] = df['bathrooms'].astype(float)
```

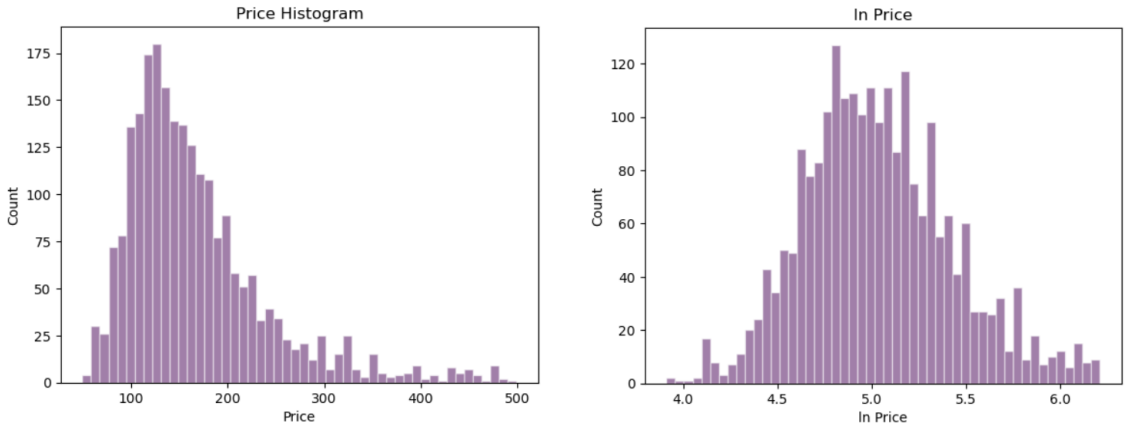
```
df['f_property_type'] = pd.factorize(df['property_type'])[0]
df['f_room_type'] = pd.factorize(df['room_type'])[0]
df['f_neighbourhood_group_cleansed'] = pd.factorize(df['neighbourhood_group_cleansed'])[0]
df['f_neighbourhood_cleansed'] = pd.factorize(df['neighbourhood_cleansed'])[0]
```

Further transformations in the data are done which can be found in the attached ipynb file which contains the complete codes. In addition, we also understand our price variable because that is our most important variable in this analysis

```
df = df[df['price'] < 500]
```

```
df['ln_price'] = np.log(df['price'])
```

The main goal of the project is to predict prices of apartments which accommodates between 2 to 6 guests; thus, the data was filtered accordingly. The key variable, price, contained extreme value and missing observations. To narrow down the project goal the target variable, price per night included extreme values above 800 USD per night which contained of less than 1% of the observations, thus, price was filtered to less than 500 USD and dropping the observation where price is missing. Moreover, As the goal of project is to build price prediction model, it is crucial to check price and log of price distribution. I created of price and log of price distributions are as following:

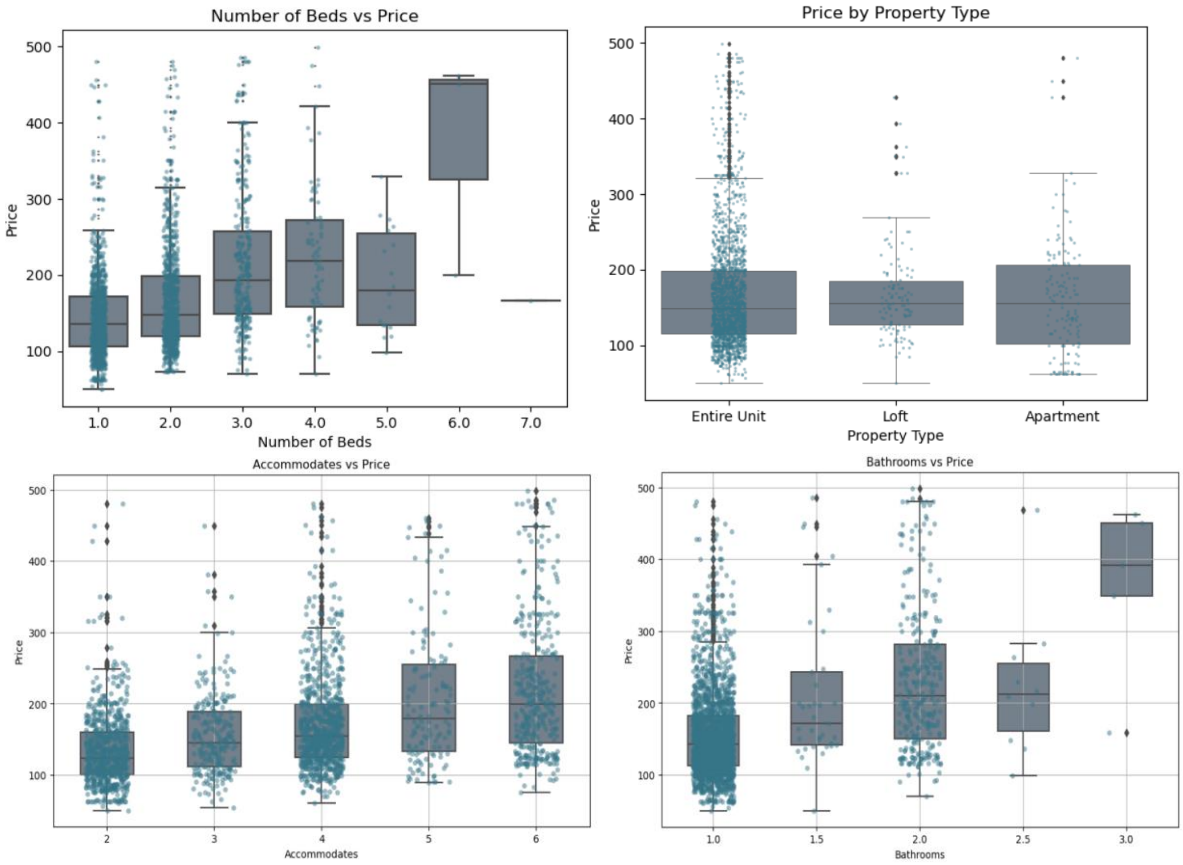


Price distribution shows Airbnb apartment prices is skewed with a long right tail and the log price is close to normally distributed. In this project, log of price is not considered, prediction is carried out with price for all the models.

In order to make a good decision regarding the models, we had to extract valuable information from our Amenities columns which seemed to be formatted in a way that we couldn't gather much information. The next code snippet shows how we prepared our variables by extracting information and creating binary variables for various amenities like Wi-Fi, kitchen, heating etc. Once that was done, we removed the amenities column from our DataFrame that's called 'df'.

```
df['amenities'] = df['amenities'].str.replace(r'\[|\]|\"|2013|2014|2019s|2019|\\\\u', '', regex=True)
df['amenities'] = df['amenities'].str.split(',')

# Defining my function to create dummy variables
def create_dummy(x, pattern):
    return 1 if any(re.search(pattern, item) for item in x) else 0

df['tv'] = df['amenities'].apply(lambda x: create_dummy(x, 'TV'))
df['heating'] = df['amenities'].apply(lambda x: create_dummy(x, 'Heating'))
df['wifi'] = df['amenities'].apply(lambda x: create_dummy(x, 'Wifi'))
df['coffee_maker'] = df['amenities'].apply(lambda x: create_dummy(x, 'Coffee maker'))
df['AC'] = df['amenities'].apply(lambda x: create_dummy(x, 'air conditioning'))
df['refrigerator'] = df['amenities'].apply(lambda x: create_dummy(x, 'Refrigerator'))
df['microwave'] = df['amenities'].apply(lambda x: create_dummy(x, 'Microwave'))
df['hot_water'] = df['amenities'].apply(lambda x: create_dummy(x, 'Hot water'))
df['long_term_stay'] = df['amenities'].apply(lambda x: create_dummy(x, 'Long term stays allowed'))
df['kitchen'] = df['amenities'].apply(lambda x: create_dummy(x, 'kitchen'))
df['pets_allowed'] = df['amenities'].apply(lambda x: create_dummy(x, 'Pets allowed'))
df['private_entrance'] = df['amenities'].apply(lambda x: create_dummy(x, 'Private entrance'))

df.drop(columns=['amenities', 'latitude', 'longitude'], inplace=True)
```

To understand the data better, now we will be creating some plots that include 'Accommodates vs Price', 'Number of Beds vs Price', 'Price by Property Type', 'Bathrooms vs Price'. In our data, we noticed that we have three different types of properties, Entire Unit, Loft and Apartments.



In this process, it is important to mention that some of the variables had missing values that needed to be taken care of. We thought of a few strategies we could use to impute those variables because dropping them wouldn't make sense as they could turn out to be important factors in determining the price of our Airbnb listing. With beds, we decided to impute a few missing values by linking it to the number of accommodates and using half the number of accommodates to replace the value of beds assuming that there are double beds in the listing. That might be suitable in some scenarios and not perfect for others but since there were very few missing values in the beds column, we thought that in our case, this would be a suitable imputation.

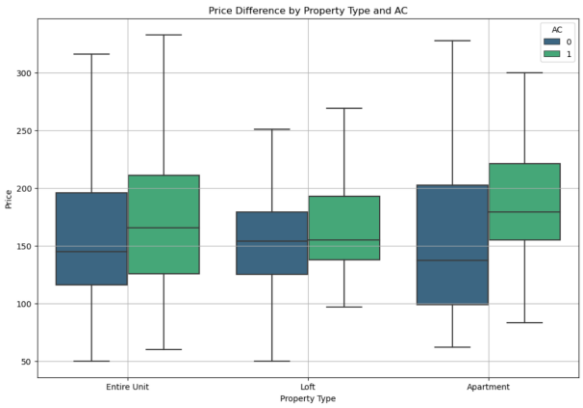
Data Analysis and Feature Engineering:

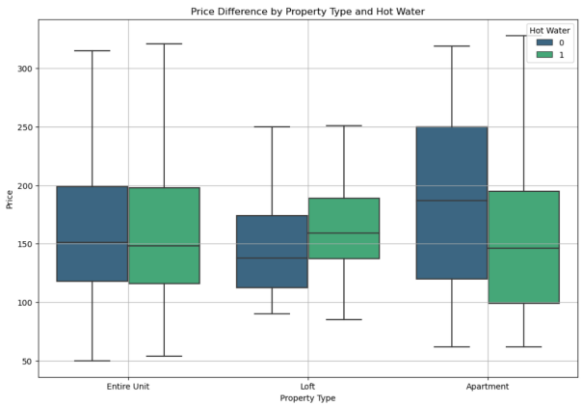
In this step, we understand what variables we need to include in our analysis. First, we looked at the basic variables that include predictors like accommodates, type of property, number of beds, etc. Alongside the basic variables, we also include factorized variables like neighborhood groups etc. We also used the amenities dummies that we created above which include multiple variables. The importance of having all these variables is to figure out the interactions between the property types, our amenities dummies and price. Here you can find the plots for all interactions that could be important in determining the price along with a plot that shows how prices vary with the number of accommodates in each property type. Attaching a code snippet to understand the technical details.

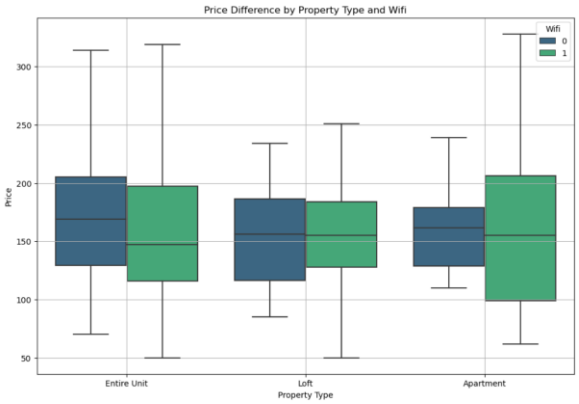
```
def price_diff_by_variables(df, variable1, variable2, label1, label2):
    plt.figure(figsize=(12, 8))
    sns.boxplot(x=variable1, y='price', hue=variable2, data=df, palette='viridis', showfliers=False)
# Add labels and title
    plt.xlabel(label1)
    plt.ylabel('Price')
    plt.title(f'Price Difference by {label1} and {label2}')
# Adjust legend position
    plt.legend(title=label2, loc='upper right')
# Show the plot
    plt.grid(True)
    plt.show()

# Call the function for each pair of variables
p1 = price_diff_by_variables(df, "property_type", "AC", "Property Type", "AC")
p2 = price_diff_by_variables(df, "property_type", "hot_water", "Property Type", "Hot Water")
p3 = price_diff_by_variables(df, "property_type", "wifi", "Property Type", "Wifi")
p4 = price_diff_by_variables(df, "property_type", "tv", "Property Type", "TV")
p5 = price_diff_by_variables(df, "property_type", "microwave", "Property Type", "Microwave")
p6 = price_diff_by_variables(df, "property_type", "coffee_maker", "Property Type", "Coffee Maker")
p7 = price_diff_by_variables(df, "property_type", "refrigerator", "Property Type", "Refrigerator")
p8 = price_diff_by_variables(df, "property_type", "accommodates", "Property Type", "Accommodates")
```

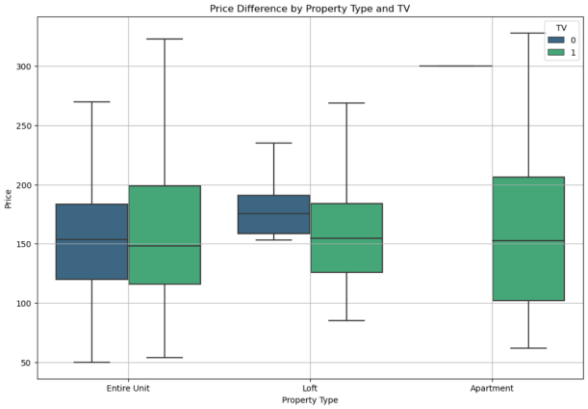
This code snippet shows how a function is used to create plots for each pair of variables. It's also important to create pair interactions to see how much these things matter in our analysis and what difference they make in relation to the pricing of our Airbnb.

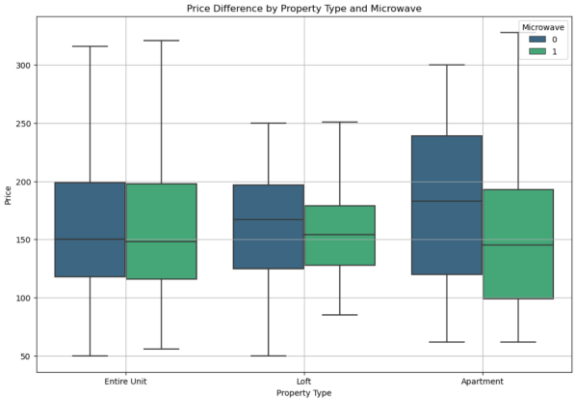
Plots for our pair interactions:

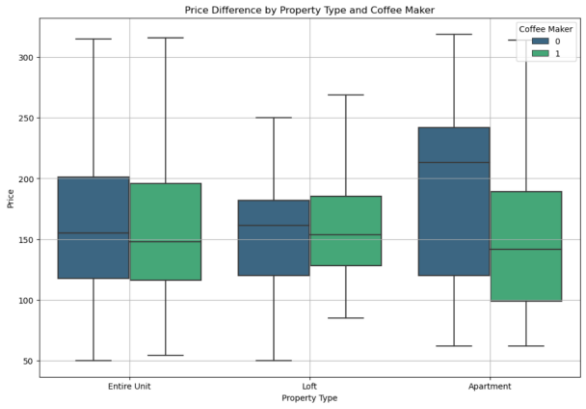


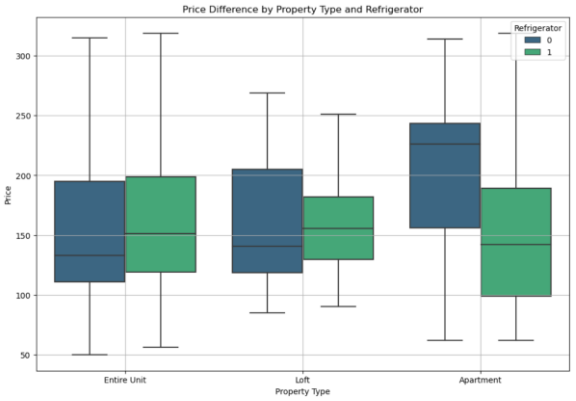


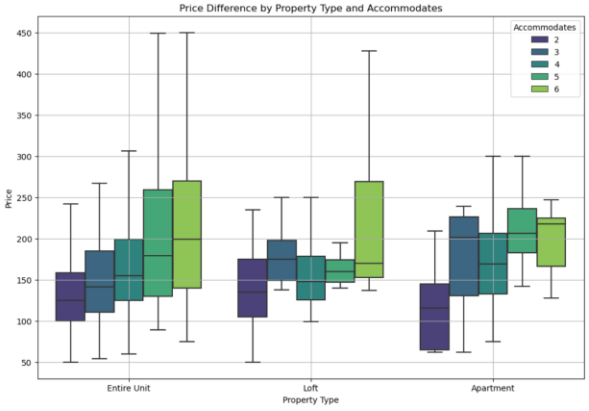












Next step is to split our test and train data in order to run our prediction models and to create different predictors so all the variables we've created are incorporated into the models and we can analyze them on the basis of RMSE values.

Random Forest Regression:

Random forest is an example of ensemble method which uses the results of many predictive models and combine those results to generate a final prediction. I used basic variables, basic additions, reviews and dummies for this model. The RMSE is 62.7003 in the training set and then we got a better value of 56.88 in the holdout set which turns out to be the best model when compared to all others.

The tables below show what RMSE our model got and then we checked what our best RMSE value is.

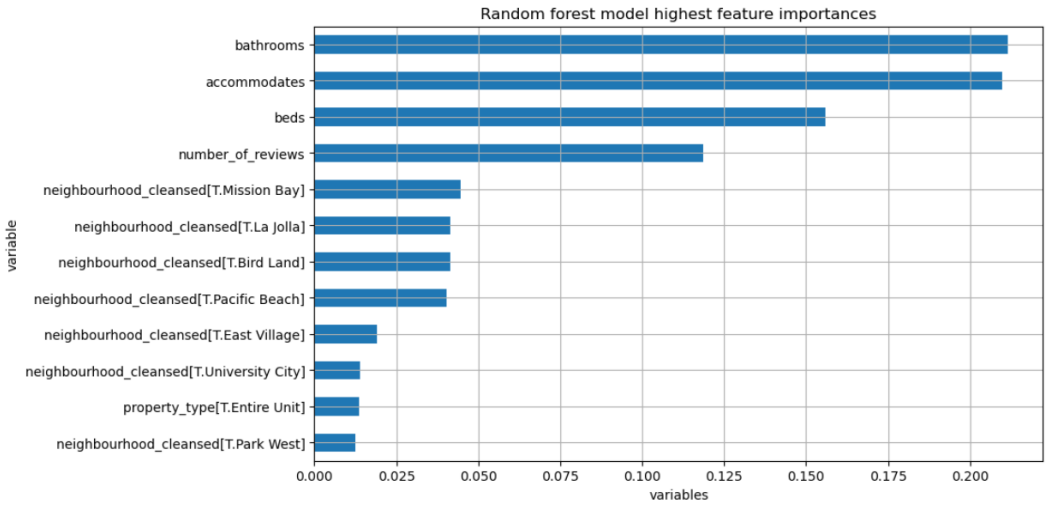
	max features	min node size	RMSE
0	6	5	-64.884036
1	6	10	-67.087759
2	6	15	-68.387451
3	8	5	-63.496647
4	8	10	-65.740722
5	8	15	-66.802052
6	10	5	-63.178463
7	10	10	-65.220136
8	10	15	-65.913941
9	12	5	-62.700355
10	12	10	-64.423575
11	12	15	-65.348757

min node size	5	10	15
max features			
6	64.88	67.09	68.39
8	63.50	65.74	66.80
10	63.18	65.22	65.91
12	62.70	64.42	65.35

```
RandomForestRegressor
RandomForestRegressor(max_features=12, min_samples_leaf=5,
                      random_state=20240129)
```

	rmse	mean_price	rmse_norm
Total	56.88	166.88	0.34

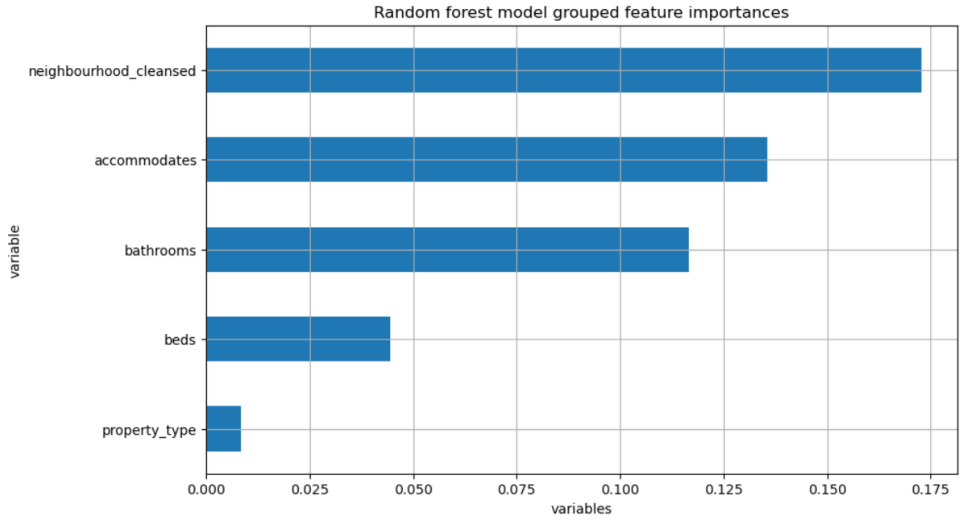
Now we will be understand the feature importance. We have to keep in mind that not all variables are to be shown because there will be some that affect our model very less therefore we define a cutoff value of 0.1 to filter out the main ones. Also providing the code snippets to understand how we create these feature importance plots.

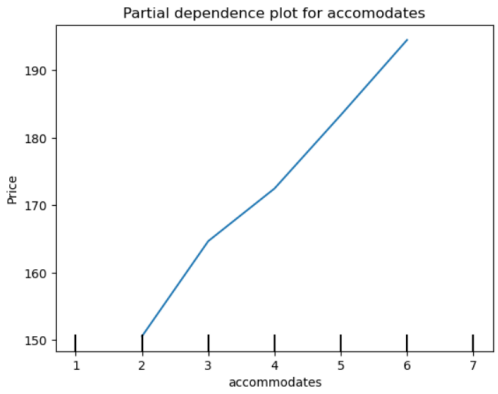



```
# we only care for variables with an importance of more than 1 pct
cutoff = 0.01
```

```
df_var_imp[df_var_imp.imp > cutoff]\
    .sort_values(by = 'imp')\
    .plot(kind = 'barh',
          x = 'variable', y = 'imp',
          figsize = (10,6), grid = True,
          title = 'Random forest model highest feature importances',
          xlabel = 'variables', legend = False);
```

Also creating grouped cumulative feature importance plots for better understanding of what kind of variables affect the price more.





Here is a partial dependace plot for the number of accommodates, this shows how, with increasing number of accommodates, we can see a clear increase.

OLS:

OLS model is one of the simplest and most common techniques for modelling linear relationships between variables. We carried out an OLS Linear regression and the CV RMSE we got from this was 60.733 as it can be seen in the code snippet attached.

```
y, X = dmatrices("price ~ " + " + ".join(predictors_2), data_train)
ols_model = LinearRegression().fit(X,y)
y_test, X_test = dmatrices("price ~ " + " + ".join(predictors_2), data_holdout)
y_hat = ols_model.predict(X)
ols_rmse = mean_squared_error(y,y_hat,squared=False)
ols_rmse
```

```
60.73316772274322
```

LASSO:

LASSO is the most widely used shrinkage method. It is an algorithm that fits a model by shrinking coefficients, some of them to zero by adding a penalty term. After running 5-fold cross validation for the selecting the optimal value for lambda, we got a CV RMSE of 62.1677 as can be seen in the output of our code snippet below:

```
lasso_rmse = pd.DataFrame(lasso_model_cv.cv_results_).loc[
    lambda x: x.param_alpha == lasso_model_cv.best_estimator_.alpha
].mean_test_score.values[0] * -1
lasso_rmse
```

```
62.16771186133881
```

GBM:
The next model for this analysis is GBM which stands for Gradient Boosting Machine. GBM models often incorporate shrinkage and regularization techniques to prevent overfitting. We will be able to see that GBM gives us the best result in terms of relative RMSE values. Our GBM model gives us an RMSE value of 59.253

```
GradientBoostingRegressor(max_depth=10, max_features=10, min_samples_split=20,
                          n_estimators=200)
```

CART:
Regression trees are called CART. It is a building and growing a tree. This algorithm has no formula, the goal is to arrive at a set of bins of predictors. This algorithm splits the bins into smaller bins. For CART algorithm, no functional or interactions were given. The variables were basic variables, basic additions, reviews and dummies. Attaching the code snippet below:

```
cart_rmse = pd.DataFrame(cart_model_cv.cv_results_).loc[
    lambda x: x.param_ccp_alpha == cart_model_cv.best_estimator_.ccp_alpha
].mean_test_score.values[0] * -1
cart_rmse
```

```
76.41248476524403
```

As we can see, the RMSE value obtained through CART is 76.412 which is considerably higher than the other prediction models.

	model	CV RMSE
0	OLS	60.733168
1	LASSO	62.167712
2	CART	76.412485
3	random forest	56.880000
4	GBM	59.253066

The table above shows the RMSE values obtained for each model for comparison. Clearly with our data, Random Forest is our best performing model and the model we will choose to get a price for our new listings.

Conclusion

The goal of this report was to find a better model to predict Airbnb prices in San Diego for small to mid-size apartments. Five models were illustrated to compare and contrast across model performance. Random Forest resulted to be the best model by 56.88 USD RMSE. The second-best model was basic GBM which gives an RMSE value of 59.25 and highlights meaningful characteristics about the nature of Airbnb apartments in San Diego. Key price drivers based on post prediction diagnostics are the number of bathrooms, number of accommodates, number of beds and the neighborhood.