Prediction with Machine Learning: Assignment 2
Botagoz Amangeldiyeva

Technical Report

The complete code can be found in my github repository:
https://github.com/botagoz00/DA3-phdma/blob/main/A2/
Cleaned dataset is also in my repository:)

1. Data preparation:

As a first step, I started by downloading the data. I chose Athens from this website http://insideairbnb.com/get-the-data/. Downloaded "listings.csv", which contained all necessary data. But this data is not clean, so I had to clean and prepare it to analysis.

After that, I downloaded all the needed libraries:

```
import pandas as pd
import numpy as np
from datetime import date
import warnings
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
import shap

warnings.filterwarnings("ignore")
```

Then, I loaded the data from "listings.csv". Since we need to analyze small and mid-size apartments, I dropped all observations, which are not apartments. And also this apartments should host only 2-6 people, so I filtered the data accordingly based on number of beds:

```
athens_raw = pd.read_csv('listings.csv', delimiter=",", dtype="unicode")
```

```
# selecting only apartments
athens_aparts = athens_raw[athens_raw['room_type']=='Entire home/apt']
```

```
# selecting only 2-6 guests
athens_aparts['beds'] = pd.to_numeric(athens_aparts['beds'], errors="coerce")
athens_limited = athens_aparts[(athens_aparts['beds']>1) & (athens_aparts['beds']<7)]
```

As a next step, I removed all unimportant column, those where data are largely missing, so I dropped them because they could affect our results:

```
# too many nan values
nans = ['host_response_rate',
          'host_acceptance_rate',
          'neighbourhood_group_cleansed',
          'bathrooms',
          'calendar_updated',
          'host_neighbourhood',
          'neighbourhood',
          'review_scores_value',
          'review_scores_checkin',
          'review_scores_communication',
          'review_scores_location',
          'review_scores_accuracy',
          'review_scores_cleanliness',
          'first_review',
          'last_review',
          'reviews_per_month',
          'review_scores_rating',
        'bedrooms'
       ]
athens_nonans = athens_limited.drop(nans, axis=1)
```

Also we have some strange column, like ulr links, which are not needed for our analysis and it is difficult to transform them into numbers, so I dropped them as well:

```
# no prediction
drops = [
    "host_thumbnail_url",
    "host_picture_url",
    "listing_url",
    "picture_url",
    "host_url",
    "last_scraped",
    "description",
    "neighborhood_overview",
    "host_about",
    "host_response_time",
    "name",
    "host_location",
    'source',
    'host_id',
    'host_name',
    'host_verifications',
    'neighbourhood_cleansed',
    'latitude',
    'longitude',
    'bathrooms_text',
    'license',
    'calendar_last_scraped',
    'has_availability',
    'property_type',
    'room_type'
]
athens_dropped = athens_nonans.drop(drops, axis=1)
```

Next we have to deal with remaining columns, and many of their values are written in the form of strings, not numeric values. So, first I drop observations that don't contain numbers as characters:

```
# drop broken lines - where id is not a character of numbers
athens_prep.id = pd.to_numeric(athens_prep.id, errors="coerce")
```

Then, I transform strings to integers:

```
# to numeric
numeric_cols = ['host_listings_count',
                'host_listings_count',
                'accommodates',
                'minimum_nights',
                'maximum_nights',
                'minimum_nights_avg_ntm',
                'maximum_nights_avg_ntm',
                'availability_30',
                'availability_60',
                'availability_90',
                'availability_365',
                'number_of_reviews',
                'number_of_reviews_ltm',
                'number_of_reviews_l30d',
                'calculated_host_listings_count',
                'calculated_host_listings_count_entire_homes',
                'calculated_host_listings_count_private_rooms',
                'calculated_host_listings_count_shared_rooms',
                'host_total_listings_count',
                 'host_has_profile_pic',
                 'host_identity_verified',
                 'minimum_minimum_nights',
                 'maximum_minimum_nights',
                 'minimum_maximum_nights',
                 'maximum_maximum_nights'
               ]
```

```
# formatting to numeric values
for perc in numeric_cols:
    athens_prep[perc] = pd.to_numeric(athens_prep[perc], errors="coerce")
```

For prices, I delete “$” sign and also transform string values into numeric/integer values:

```
# remove $ sign from price
athens_prep['price'] = athens_prep['price'].str.replace("\\$", "")
athens_prep['price'] = pd.to_numeric(athens_prep['price'], errors="coerce")
```

For binary variables, I transform it into binary format:

```
# format binary variables
for binary in [
    "host_is_superhost",
    "host_has_profile_pic",
    "host_identity_verified",
    "instant_bookable"
]:
    athens_prep[binary] = athens_prep[binary].map({"t": True, "f": False})
```

Next, I have amenities of apartments, and I have many of them. The ones that were written in text format with commas, I transform them in order to work with these amenities. Also, to make my data cleaner, so to make predictive models more reliable, I dropped amenities that are present in less than 5% of records.

```
# convert to a string to a list
athens_prep["amenities"] = athens_prep["amenities"].str.strip("[]").str.replace('"', "").str.split(",")
```

```
# get dummies from each ammenity in lists
dummies = pd.get_dummies(athens_prep.amenities.apply(pd.Series).stack()).sum(level=0)
```

```
# know 5% of all records
percent5 = athens_prep.shape[0] * 0.05
```

```
# drop ammenities that are not present in at least 5% of records
drop_ammenities = [i for i in dummies.columns.tolist() if dummies.sum()[i] < percent5]
dummies_significant = dummies.drop(drop_ammenities, axis=1)
```

```
athens_final = pd.concat([athens_prep, dummies_significant], axis=1).drop('amenities', axis=1)
```

Finally, I cleaned and transformed my data, so I saved it:

```
athens_final.to_csv('athens_final.csv', index=False)
```

```
athens_final = pd.read_csv('athens_final.csv')
```

2. Task 1: Developing predictive models

Let's say we are at the end of 2022 year and want to predict a price to Jan 2023. Let's say the price did not change significantly for the last 10 years, which is a very rough assumption made only for the sake of data fullness. So, I create 2 datasets: one for before 2023 [2018 - 2022] (train), and the one for 2023 [January] (test). Because I will have to test the predictive power of models:

```
# for model to understand convert boolean to num
for col in ['host_is_superhost', 'instant_bookable', 'host_has_profile_pic', 'host_identity_verified']:
    athens_dated[col] = athens_dated[col].astype('float')
```

```
# train data = last 5 years before 2023
train_data = athens_dated[(athens_dated['date_format']>='2018-01-01') & (athens_dated['date_format']<'2023-01-01'
train_data = train_data.dropna(subset='price').reset_index(drop=True)
```

```
# test data = Jan of 2023
test_data = athens_dated[(athens_dated['date_format']>='2023-01-01') & (athens_dated['date_format']<'2023-02-01')
test_data = test_data.dropna(subset='price').reset_index(drop=True)
```

```
# make x and y
x_train = train_data.drop(['id', 'scrape_id', 'host_since', 'price', 'date_format'], axis=1)
y_train = train_data[['id', 'price']].copy()

x_test = test_data.drop(['id', 'scrape_id', 'host_since', 'price', 'date_format'], axis=1)
y_test = test_data[['id', 'price']].copy()
```

Then I need to fill what is missing. Only one column had missing values, it is "host_is_superhost" variable, so I fill it with 0s because it was the most common value for the whole set:

```
# host_is_superhost is the only column with nans, fill it with the majority class
print(x_train['host_is_superhost'].value_counts())
x_train['host_is_superhost'] = x_train['host_is_superhost'].fillna(0)
x_test['host_is_superhost'] = x_test['host_is_superhost'].fillna(0)
```

```
.0    1681
.0    1216
lame: host_is_superhost, dtype: int64
```

```
# no nans left
x_test.isna().sum().sum(), x_train.isna().sum().sum()
```

OLS: First model

```
model = sm.OLS(y_train[['price']], x_train)
results = model.fit()
```

So, I run OLS regression for the train dataset with all remaining variables. F statistics is less than 5% showing a statistical significance of prediction. Also. the adjusted R-squared value is high (73.5%) representing good results.

Now, I check whether this OLS predictive model is good and I do it by measuring RMSE. And we can see that RMSE for OLS model is 39, which is a good result (given that we have a scale from 0 to 350):

```
# RMSE  for the holdout is 39.1
y_pred_ols = results.predict(x_test)
mse_ols = mean_squared_error(y_test['price'], y_pred_ols)
np.sqrt(mse_ols)
```

So, I plotted a graph to see if OLS fits good and it does:

```
plt.scatter(y_test['price'], y_pred_ols, label='predictions')
plt.plot([0, 350], [0, 350], label='staright_line')
plt.xlabel('true_values')
plt.ylabel('predicted_values')
plt.title('OLS test results')
plt.legend()
plt.show()
```

Random Forest: Second model

I do the same for the second model, but now I run Random Forest regression:

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30]
}
```

```
rf_model = RandomForestRegressor()
```

```
# searching for the best hyperparameters for RF
rf_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=3)
```

```
rf_search.fit(x_train, y_train['price'])
```

And doing the same steps, I check for RMSE, which turned out to be 50, we can see that OLS is better:

```
# RMSE  for the holdout is 49.5
y_pred_rf = rf_search.predict(x_test)
mse_rf = mean_squared_error(y_test['price'], y_pred_rf)
np.sqrt(mse_rf)
```

```
50.308844720104666
```

ADA boosting: Third model

As a last model, I chose ADA boosting to see whether it is going to predict better than OLS and random forest.

```
ada = AdaBoostClassifier()
ada.fit(x_train, y_train['price'])
```

Calculating RSME we can see that it is very high, 65, which is almost twice more than for OLS:

```
# RMSE  for the holdout is 65.6
y_pred_ada = ada.predict(x_test)
mse_ada = mean_squared_error(y_test['price'], y_pred_ada)
np.sqrt(mse_ada)
```

```
65.56265353432195
```

3. Task 2: Testing best model on a new date

In terms of RMSE, OLS showed the best performance. Meaning the linear relation to the target is more stronger than Tree based splitting. Now I check the model for predicting more recent prices, i.e. for the May instead of Jan 2023.

Now I changed test data to May 2023, previously it was January 2023:

```
# test data2 = May of 2023
test_data2 = athens_dated[(athens_dated['date_format']>='2023-05-01') & (athens_dated['date_format']<'2023-06-01'
test_data2 = test_data2.dropna(subset='price').reset_index(drop=True)
```

```
# make x and y
x_test2 = test_data2.drop(['id', 'scrape_id', 'host_since', 'price', 'date_format'], axis=1)
y_test2 = test_data2[['id', 'price']].copy()
```

Again, fill missing values:

```
# host_is_superhost is the only column with nans, fill it with the majority class
x_test2['host_is_superhost'] = x_test2['host_is_superhost'].fillna(0)
```

```
# no nans left
x_test2.isna().sum().sum()
```

Do OLS regression, and we can see that RMSE is 75, which is very high:

```
# RMSE  for the holdout is 75.3
y2_pred_ols = results.predict(x_test2)
mse2_ols = mean_squared_error(y_test2['price'], y2_pred_ols)
np.sqrt(mse2_ols)
```

```
75.28250341500777
```

But when we look to graph, there was one big outlier, so I dropped it and RMSE turned to be 41, which is good. So our OLS model predicts great with May data as well.

```
# temporary calculations just to check the performance without outlier
temp_y_test = y_test2['price'].tolist()
temp_y_pred = y2_pred_ols.copy()
remove_ind = temp_y_test.index(817.0)
temp_y_test.pop(remove_ind)
temp_y_pred.pop(remove_ind)
```

```
249.65683121870993
```

```
# RMSE without outlier is 41.1
temp_mse = mean_squared_error(temp_y_test, temp_y_pred)
np.sqrt(temp_mse)
```

```
41.41038664877566
```

4. Task 3: Assessing performance

Finally, I consider RF and assess its performance using Shapley values:

```
explainer = shap.TreeExplainer(rf_search.best_estimator_)
```

```
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test)
```