Pricing Airbnb Apartments in Sicily, Italy - Technical Report

The code and both reports can be found on this GitHub.

1. Data

As described in the summary report, we have to do some file manipulation and feature engineering to obtain the necessary predictors.

For file manipulation, since the original data file is too large, GitHub refuses to accept the file. Hence, we have to split the file into 2 parts and read into 2 dataframes. The final working dataframe is the concatenation of those 2 dataframes.

When applying filters, we decided to drop the observations with missing values as they only cover a very small amount of the observations. Also, aside from the business reasons, we limit the dataset to popular cities otherwise the data set remains too large and take time to run the ML models.

During the feature engineering steps, some decisions are taken when imputing values. We fill 1s as values for observations with no bedrooms values as upon spot checking, these are studio apartments. We also convert listing duration from text to number of days before 31/03/2023 because the data is collected for observation up until 31/03/2023. Values may change after that period hence we use the cut-off date as 31/03/2023 to ensure correctness.

In [37]:
```
def get_cleaned_data(src=None) -> pd.DataFrame:
    '''
    This function takes a path to a csv file, cleans it and returns the cleaned dataframe
    :param src: path to file
    :return: dataframe with cleaned data
    '''
    if src:
        if isinstance(src, list):
            dfs = []
            for file in src:
                u_df = pd.read_csv(file)
                dfs.append(u_df)
            df = pd.concat(dfs, ignore_index=True)
        else:
            df = pd.read_csv(src)
    else:
        urls = ['https://raw.githubusercontent.com/viethngn/Data_Analysis_3_ECBS5171/main/assignment2/listings_1.cs
        dfs = []
        for url in urls:
            u_df = pd.read_csv(url)
            dfs.append(u_df)
        df = pd.concat(dfs, ignore_index=True)

    # Filter the data
    working_sample = df[['id','name','host_id','host_name','host_since','host_is_superhost','host_total_listings_co
                        ][(df['price'].notna())
                        & (df['beds'].notna())
                        & (df['host_is_superhost'].notna())
                        & (df['host_since'].notna())
                        & (df['bathrooms_text'].notna())
                        & (df['room_type'] != 'Hotel room')
                        & (df['accommodates'] <= 6) & (df['accommodates'] >= 2)
                        & (df['neighbourhood_cleansed'].isin(['Palermo', 'Catania', 'Gravina di Catania', 'San Greg

    # fill NA data for reviews with 0s
    working_sample['review_scores_value'].fillna(0, inplace=True)

    # fill NA data for bedrooms with 1s
    working_sample['bedrooms'].fillna(1, inplace=True)

    # fill NA data for license with NAN
    working_sample['license'].fillna('NAN', inplace=True)

    # add boolean variables from text columns
    working_sample['d_host_is_superhost'] = working_sample['host_is_superhost'].apply(lambda x: 1 if x == 't' else
    working_sample['d_host_has_profile_pic'] = working_sample['host_has_profile_pic'].apply(lambda x: 1 if x == 't'
    working_sample['d_host_identity_verified'] = working_sample['host_identity_verified'].apply(lambda x: 1 if x ==
    working_sample['d_instant_bookable'] = working_sample['instant_bookable'].apply(lambda x: 1 if x == 't' else 0)
    working_sample['d_has_license'] = working_sample['license'].apply(lambda x: 1 if x != 'NAN' else 0)

    # calculate host time
    working_sample['n_host_since'] = working_sample['host_since'].apply(lambda x: (pd.Timestamp('2023-03-31') - pd.

    # convert price to numerical
    working_sample['price'] = working_sample['price'].apply(lambda x: float(x.replace('$', '').replace(',', '')))

    # filter for price <
    working_sample = working_sample[(working_sample['price'] <= 400)]
```

```
    # add numerical variable for number of bath
    working_sample['n_bathrooms'] = working_sample['bathrooms_text'].apply(lambda x: 0.5 if 'half-bath' in x.lower(

    # clean the property type and room type
    working_sample['property_type'] = working_sample['property_type'].apply(lambda x: x.lower().replace('entire ',
    working_sample['room_type'] = working_sample['room_type'].apply(lambda x: x.lower())

    # add amenities columns
    working_sample['d_entertainment'] = working_sample['amenities'].apply(lambda x: 1 if 'tv' in x.lower() or 'game
    working_sample['d_wifi'] = working_sample['amenities'].apply(lambda x: 1 if 'wifi' in x.lower() else 0)
    working_sample['d_kitchenware'] = working_sample['amenities'].apply(lambda x: 1 if 'fridge' in x.lower() or 'fr
    working_sample['d_washer'] = working_sample['amenities'].apply(lambda x: 1 if ('washer' in x.lower() or 'dryer'
    working_sample['d_sauna_hot_tub'] = working_sample['amenities'].apply(lambda x: 1 if 'sauna' in x.lower() or 't
    working_sample['d_pool'] = working_sample['amenities'].apply(lambda x: 1 if ' pool' in x.lower() or 'Pool' in x
    working_sample['d_aircon'] = working_sample['amenities'].apply(lambda x: 1 if 'air con' in x.lower() else 0)
    working_sample['d_heating'] = working_sample['amenities'].apply(lambda x: 1 if 'heating' in x.lower() else 0)
    working_sample['d_scenic_view_access'] = working_sample['amenities'].apply(lambda x: 1 if 'view' in x.lower() o
    working_sample['d_parking'] = working_sample['amenities'].apply(lambda x: 1 if 'parking' in x.lower() or 'carpo
    working_sample['d_pets_allowed'] = working_sample['amenities'].apply(lambda x: 1 if 'pets allowed' in x.lower()
    working_sample['d_patio_balcony'] = working_sample['amenities'].apply(lambda x: 1 if 'patio' in x.lower() or 'b
    working_sample['d_bodyshower'] = working_sample['amenities'].apply(lambda x: 1 if 'shampoo' in x.lower() or 'co

    # rename numerical and categorical columns to have prefix
    working_sample.rename(columns={
        'host_total_listings_count': 'n_host_total_listings_count',
        'accommodates': 'n_accommodates',
        'bedrooms': 'n_bedrooms',
        'beds': 'n_beds',
        'minimum_nights': 'n_minimum_nights',
        'maximum_nights': 'n_maximum_nights',
        'neighbourhood_cleansed': 'f_neighbourhood_cleansed',
        'property_type': 'f_property_type',
        'room_type': 'f_room_type',
        'review_scores_value': 'n_review_scores_value',
        'number_of_reviews': 'n_number_of_reviews'
    }, inplace=True)

    del df
    if not src or isinstance(src, list):
        del dfs[0]
        del dfs[1]

    return working_sample
```

---

2. Model Construction

OLS and LASSO

For OLS and LASSO models, we use the patsy.dmatrices() method to create the matrices for training with the training set and diagnostic on the holdout set to handle the interaction terms.

In [38]:
```
# define function to get train test split for OLS and LASSO
def get_ols_train_test_split(data, lasso=False):
    if lasso:
        y_, ols_df = dmatrices('price~' + '+'.join(numerical_columns) + '+' + '+'.join(categorical_columns) + '+' +
    else:
        y_, ols_df = dmatrices('price~' + '+'.join(numerical_columns) + '+' + '+'.join(categorical_columns), data,
    ols_terms_list = ols_df.design_info.column_names
    ols_df['price'] = y_
    ols_data_train, ols_data_holdout = train_test_split(ols_df, train_size=0.7, random_state=42)
    return ols_data_train, ols_data_holdout, ols_terms_list
```

With the LASSO model, we have to write another function to standardize the values in the matrices. This function is reused when making prediction with the holdout set.

In [39]:
```
# get the matrices for LASSO
def get_lasso_matrices(data):
    y_ = data['price']
    X_ = data[lasso_terms_list]
    scaler = StandardScaler()
    X_ = scaler.fit_transform(X_)
    return y_, X_
```

In [40]:
```
ols_model
```

Out[40]:
```
▾ LinearRegression
LinearRegression()
```

In [41]:
```
lasso_results.best_estimator_
```

Out[41]:
```
▾ ElasticNet
ElasticNet(alpha=0.4, l1_ratio=1)
```

Random Forest and GBM

For the Random Forest and GBM models, we use the scikit-learn OneHotEncoder() to create the dummy values for categorical predictors. We also use the scikit-learn pipeline for both of these models instead of patsy.dmatrices() as it will handle the data transformer automatically. The defined data preprocessing step is reused in both Random Forest and GBM pipelines.

In [42]:
```
# building preprocessing for pipeline
categorical_encoder = OneHotEncoder(handle_unknown="ignore")

preprocessing = ColumnTransformer(
    [
        ("cat", categorical_encoder, categorical_columns),
        ("num", "passthrough", numerical_columns),
    ]
)
```

In [43]:
```
lasso_results.best_estimator_
```

Out[43]:
```
▾ ElasticNet
ElasticNet(alpha=0.4, l1_ratio=1)
```

In [44]:
```
gbm_model_cv.best_estimator_
```

Out[44]:
```
▾ GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.01, max_depth=15, max_features=12,
                          min_samples_leaf=5, min_samples_split=10,
                          n_estimators=500, random_state=42)
```

---

3. Model evaluation

Diagnostic

In [34]:
```
plt_ols_grouped_coef = df_ols_var_coefs.groupby(['grouped_term']).sum().sort_values(by="abs_ols_coefficient", ascen
plt_ols_grouped_coef.reset_index(inplace=True)
plt_ols_grouped_coef
```

Out[34]:

	grouped_term	ols_coefficient	abs_ols_coefficient
0	f_property_type	-1416.989	1738.185
1	f_neighbourhood_cleansed	3.327	118.913
2	f_room_type	-47.871	47.871
3	d_pool	36.514	36.514
4	n_bathrooms	18.799	18.799
5	n_bedrooms	11.076	11.076
6	d_sauna_hot_tub	10.713	10.713
7	d_wifi	10.306	10.306
8	d_pets_allowed	-5.712	5.712
9	n_accommodates	5.410	5.410

Table 1: Simple OLS grouped term coefficients

In [35]:
```
df_lasso_grouped_var_coefs.sort_values(by="abs_lasso_coefficient", ascending=False)[['grouped_term', 'lasso_coeffic
```

Out[35]:

	grouped_term	lasso_coefficient	abs_lasso_coefficient
48	f_neighbourhood_cleansed:f_property_type	4.394	17.616
100	n_accommodates:n_bathrooms	7.549	7.549
72	f_property_type:d_pool	2.708	6.722
67	f_property_type:d_instant_bookable	1.538	5.814
73	f_property_type:d_sauna_hot_tub	4.499	5.609
91	f_room_type:n_bathrooms	-5.363	5.363
83	f_property_type:n_host_total_listings_count	3.692	5.358
101	n_accommodates:n_bedrooms	4.536	4.536
150	n_review_scores_value	-4.481	4.481
43	f_neighbourhood_cleansed:d_pool	-2.898	4.204

Table 2: LASSO grouped term coefficients (independent terms only)

In Table 1 and Table 2, the 'abs_lasso_coefficient' column is the sum of all absolute coefficient values of the categorical and interaction predictors for the simple OLS and LASSO models. This is to display how much power these terms have in determining the price. The values from 'abs_lasso_coefficient' column are the basis for the argument about the OLS and LASSO models' feature importance in the **Model evaluation** in the summary report.

Performance

To build the performance summary table, we create a dataframe consisting of the RMSEs from both the training set and holdout set with tracked runtime for all models.

In [45]:

```
# build the performance summary table
def get_time_delta(delta):
    return f'{delta.seconds // 60}m{delta.seconds % 60}s'

lasso_y_holdout, lasso_X_holdout = get_lasso_matrices(lasso_data_holdout)
diagnostic_df = pd.DataFrame({'Model': ['Simple OLS', 'LASSO', 'Random Forest', 'GBM'],
                              'Train RMSE': ['{:.4f}'.format(ols_rmse), '{:.4f}'.format(lasso_search.best_score_*-1
                              'Holdout RMSE': ['{:.4f}'.format(mean_squared_error(ols_model.predict(ols_data_holdou
                                               '{:.4f}'.format(mean_squared_error(lasso_search.predict(lasso_X_hold
                                               '{:.4f}'.format(mean_squared_error(rf_pipe.predict(data_holdout[nume
                                               '{:.4f}'.format(mean_squared_error(gbm_pipe.predict(data_holdout[num
                              'Training time': [get_time_delta(ols_time), get_time_delta(lasso_time), get_time_delt
                              })
diagnostic_df
```

Out[45]:

	Model	Train RMSE	Holdout RMSE	Training time
0	Simple OLS	44.3212	45.0548	0m0s
1	LASSO	43.8334	44.6018	4m28s
2	Random Forest	42.9022	43.2020	0m23s
3	GBM	40.1745	40.6951	7m36s

The detailed cross-validated training performances for each model (except OLS since there is no cross-validation) are as follows:

LASSO

In [46]:

```
df_lasso_model_cv_results = pd.DataFrame(lasso_results.cv_results_)[['param_alpha', 'rank_test_score', 'mean_fit_ti
df_lasso_model_cv_results.columns = ['alpha', 'rank', 'fit time', 'RMSE']
df_lasso_model_cv_results
```

Out[46]:

	alpha	rank	fit time	RMSE
0	0.1	9	12.424075	-45.266518
1	0.15	8	7.153259	-44.570358
2	0.2	7	5.442965	-44.206283
3	0.25	6	5.172487	-44.017477
4	0.3	5	4.416456	-43.909048
5	0.35	4	3.556646	-43.855604
6	0.4	1	2.899762	-43.833404
7	0.45	2	2.516872	-43.833573
8	0.5	3	2.378954	-43.852494

Random Forest

In [47]:
```
df_rf_model_cv_results = pd.DataFrame(rf_random.cv_results_)[[
    'param_max_features', 'param_min_samples_leaf', 'mean_test_score']]
df_rf_model_cv_results.columns = ['max features', 'min node size', 'RMSE']
df_rf_model_cv_results.pivot(
    index = 'max features',
    columns = 'min node size',
    values = 'RMSE').round(2)*-1
```

Out[47]:

min node size	5	10	15
max features			
8	43.43	44.43	44.98
10	43.12	44.09	44.66
12	42.90	43.75	44.32

GBM

In [48]:
```
df_gbm_model_cv_results = pd.DataFrame(gbm_model_cv.cv_results_)[[
    'param_max_features', 'param_min_samples_leaf', 'param_max_depth', 'param_n_estimators', 'mean_fit_time', 'mean
df_gbm_model_cv_results.columns = ['max features', 'min node size', 'max depth', '# estimators', 'fit time', 'RMSE'
df_gbm_model_cv_results
```

Out[48]:

	max features	min node size	max depth	# estimators	fit time	RMSE
0	8	5	5	200	0.849614	-45.660642
1	8	5	5	300	1.268279	-44.381817
2	8	5	5	500	2.090474	-43.182553
3	8	5	5	200	0.847587	-45.630114
4	8	5	5	300	1.258055	-44.370839
...	...	...	...	...	...	...
238	12	15	15	300	4.183899	-41.660155
239	12	15	15	500	7.129586	-40.789161
240	12	15	15	200	2.687418	-42.738153
241	12	15	15	300	4.095615	-41.660155
242	12	15	15	500	5.524981	-40.789161

243 rows × 6 columns