Toronto

College of Professional Studies

EAI6020 | 21176 | AI Systems Technology

Module 2| AI Solution Assignment

Dr. Mohammad Shafiqul Islam

Pranav Sankar Mohan Indira
NU ID: 002800895
March 16, 2025

Airbnb Market Analysis & Real Estate Sales Data Analysis

Introduction

This "Market Analysis and Real Estate Sales Data (2019)" dataset aggregates monthly information on two Californian regions: Big Bear and Joshua Tree. In the analysis, the dataset provides data regarding key metrics such as the property ID, generated revenue, availability or openness, occupancy ratios, nightly rates, lead times, and average lengths of stay of Airbnb listings and property sales. The dataset includes geographical data for spatial analysis, plus property amenities information.

The purpose of this project is to establish a complete machine learning pipeline to forecast property revenue by means of normalized features drawn from this dataset. The present work outlines the procedures of data preprocessing, the way models are selected, tuning hyperparameters, evaluation metrics, and finally the deployment of the trained model as an API. This analysis provides data-driven decision-making options in the real estate and Airbnb industry, also providing real-world insights into market trends.

Data Preprocessing

The dataset obtained from Kaggle had various issues that were to be taken care of before the model could be made. The raw data were in a CSV file, with semicolon delimiters; they had some form of inconsistent formatting, with extra tabs or rows not conforming to the column count rules. The first preprocessing was to read through the entire file in line mode, extract the header, and only keep the rows that had the correct number of columns while skipping some rows to ensure that the data remained intact and uncorrupted.

The second stage was to have separate columns that contain each major numeric feature designated in this study: ie, the numbers of bedrooms, bathrooms and guests, revenue, openness, occupancy, nightly rate, lead time, and length of stay were to be converted from string representations to numeric types. Special emphasis was placed here on treating commas used as thousand separators versus those used as decimal points. A helper function was developed to cleanly and properly convert these strings to floating point numbers. Missing values of numeric fields were replaced by the median of each column, since it is a more robust indicator to use in the presence of outliers as compared to the mean.

One-hot encoding was applied to transfer categorical variables including month, zipcode, city, and host_type into binary indicator variables so that these values would be able to enter the sculptured machine learning model. Numeric features were standardized using scikit-learn’s StandardScaler afterward to scale each variable in the cleaned dataset to have a zero mean and unit variance, making sure every feature was equally impactful toward model training. Besides that, the IQR (interquartile range) method was also used to trim extreme outliers from numeric variables so as not to make extraordinary values exert too much leverage. Histograms, boxplots, and correlation heatmaps were produced during all these preprocessing stages to help confirm cleaning and help with feature selection.

Exploratory Data Analysis (EDA) and Visualization

It comprised explorative data analysis to see the structure and distribution of the data. Summary statistics were computed for every numeric feature, composing mean, standard deviation, quartile values, and range as key property data. Features like number of bedrooms, bathrooms, and revenue per night were visualized in histograms to identify skews in distribution and outlier values. Boxplots also confirmed such insights into the spread of the various features, confirming the outlier's existence. A correlation matrix plotted into a heatmap provided insights on relationships among variables.

Moderate correlations were there between revenue and occupancy and revenue with the nightly rate: they confirmed there to be some important predictors to property revenue. Such visualizations helped in diagnosis of data quality issues, as well as steering the ways of preprocessing by assuring upon normalization, outlier capping, and robust imputation methods. All the visualizations were part of the final report and codebase, which ensured transparency and reproducibility of the data cleaning process.

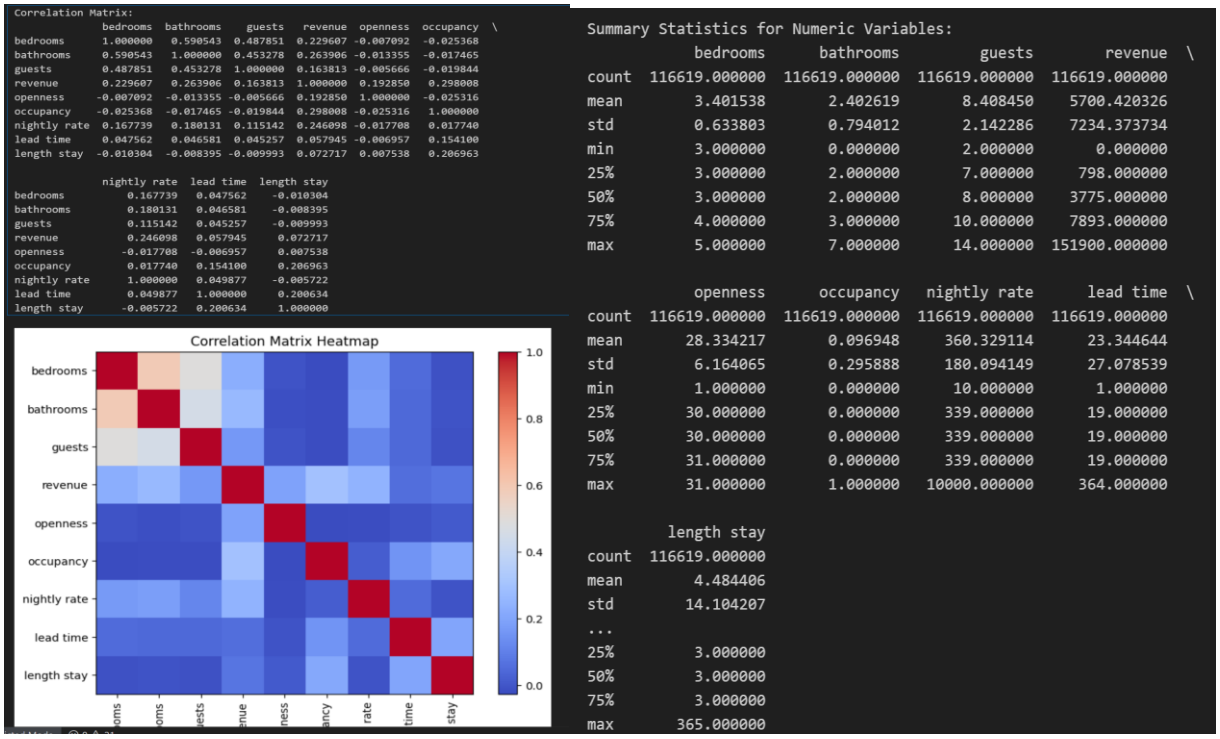

Fig 1 : Correlation Matrix and Summary Statistics

Model Building and Hyperparameter Tuning

Thus, as the problem was of regression type in the sense it was trying to predict the revenue from properties, RandomForestRegressor was selected on account of its robustness, ease of use, and skillfulness in modeling complex nonlinear relationships. The Random Forest is an ensemble learning method that combines those trees so that decision trees would minimize overfitting and enhance generalizability.

Hyperparameter tuning was done with the help of GridSearchCV in order to provide better performance for models. The grid of values to be searched considers the following key hyperparameters:

- **n_estimators**: The number of trees.
- **max_depth**: The maximum depth of each tree.
- **min_samples_split**: The minimum number of samples required to split an internal node.

Due to computational limitations and the need for a timely solution, a compressed parameter grid was implemented with a 2-fold cross-validation strategy on the training set. This way, it enables a good compromise between tuning granularity and computational efficiency. Therefore, the best parameters enabled through this process were as follows:

- max_depth: 10
- min_samples_split: 2
- n_estimators: 100

These hyperparameters were set in a way to minimize the object value of negative mean squared error MSE in cross-validation; therefore, it was believed that the hyperparameter model would capture the workload trends inherent in the data.

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
Best parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
Best training MSE: 0.7567608458967954
Test MSE: 0.7680315814754998
Test R2: 0.2274360119214236
```

Fig 2: Hyper parameter Tuning

Model Evaluation

The final model performance was evaluated using common regression metrics.

• **Mean Squared Error MSE**: The average squared difference between the model predicted and actual revenue: final test MSE apart from 0.768 on adequately scaled data.

• **R^2 Score**: Proportion of variance explained by the regression model for target variables. The model had an R^2 score of about 0.227 in this project, indicating that it captures some of the variability but still has quite a few rooms for further improvement.

• **Mean Absolute Error MAE**: The average measure of absolute difference between actual and predicted values. The model's MAE was somewhat around 0.668 in the normalized scale.

The residual analysis also involved the following considerations. A scatterplot of predicted values against the residuals appeared to exhibit randomness about zero and no clear apparent systematic bias. A histogram of the residuals likewise illustrated that the error distribution is close to normal.

These plots lent some further credence to the performance of the model and its potential for making predictions. Notice in this case that the predictions and evaluation metrics are expressed in a normalized scale. An inverse transformation would have to be applied in a production environment to return predictions to the original revenue scale, giving rise to interpretable monetary values.

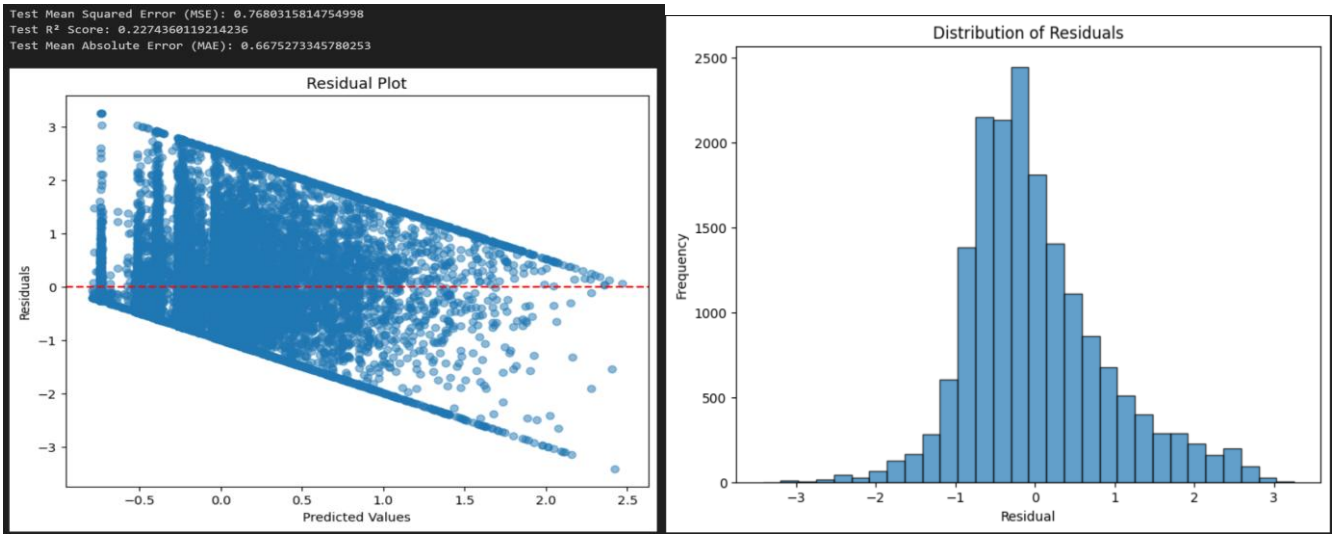

Fig 3: Visualizations: Residual Analysis

Deployment and API Integration

Once the model had been validated, the next step was to save it with the Pickle library to allow for its deployment. The next step basically consisted of developing a trivial RESTful API with the help of Flask. The Flask API was set up to load the saved model and do predictions by way of two endpoints:

1. A home route (/) with a return of a standard prediction based on the preset sample input.

2. An actual /predict endpoint that listens only for POST requests containing JSON-formatted input data and proceeds with the corresponding prediction.as easy as possible for the users, the API was implemented such that it would simply show a default prediction right on the browser when opening the root URL.

This was an excellent choice for the interface of API testing for those users who didn't bother with queries in format POST.

The following was the sample structure of the API code:

•At startup, the model is imported from best_model.pkl.

•The input vector was defined as a combination of 8 normalized numeric values and 43 zeros for one-hot encoded categorical features and reshaped before passing to the model for prediction.

•The returned output is JSON that includes the given prediction by the model and an elaborate statement mentioned clearly that the prediction is regarding property revenue for the specified features.

Detailed instructions for running the API were provided:

1. Place the app.py file and best_model.pkl in the same directory.
2. Open Command Prompt or PowerShell, navigate to the project folder using the cd command, and run the command python app.py.
3. The Flask server will start in development mode on http://127.0.0.1:5000/.
4. To test the API, users can either visit the URL in a browser or use tools such as Postman or cURL. For example, the following cURL command can be used in PowerShell:
5. curl.exe -X POST -H "Content-Type: application/json" -d "{\"input\": [-0.5, -0.3, 0.2, 0.1, 0.0, 0.05, -0.1, 0.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]}" http://127.0.0.1:5000/predict

This command sends a POST request with a sample input array and returns the predicted value in JSON format.

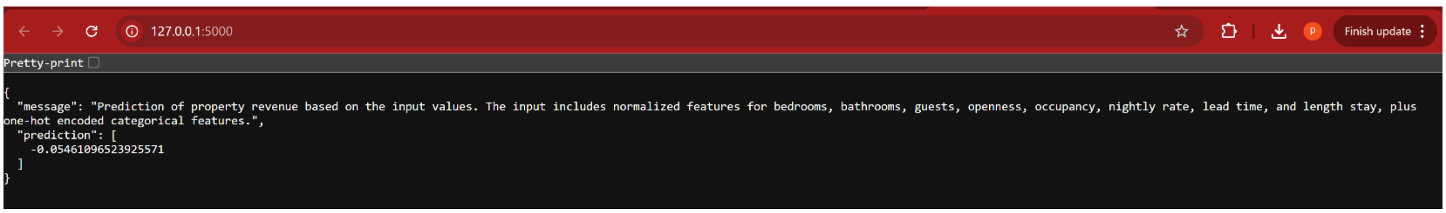

Fig 4: Prediction

The expected output of -0.05461 corresponds to the standardization of the property revenue with respect to the given input feature values. This means that the model anticipating this input will valuation the revenue of 0.05461 standard deviations lower in comparison with the mean revenue from the training dataset. Because of the preprocessing step, where the information was standardized, it is not now in a monetary unit but in a normalized unit. In context, the normalized prediction would be converted back to the revenue scale so that it makes sense using the inverse transformation (for example, the StandardScaler's inverse_transform method).

Challenges and Mitigation Strategies

Multiple issues arose during project implementation. First, the raw data had rows with mismatched column counts due to formatting errors that created problems when reading through the data. This was fixed by creating a robust file-reading procedure with the capability of skipping over any rows not complying with the required schema. A further challenge entailed the conversion of numeric values which contained vendored commas and additional tab characters. An explicit function was built for the proper parsing of these values, making sure that conversion to floating point was properly executed.

Handling the missing values posed yet another massive challenge: some columns had their values lost by more than 30%, meaning that this could really dent the performance of the model. The choice of using median imputation was due to the needed robustness against outliers and extreme values. Further measures included outlier detection and capping using the IQR method so that the model will not be substantially biased by unusually high or low values. During deployment of the models, file path problems and missing module dependencies (i.e., scikit-learn) were likely to bog down the deployment.

This was remedied by proofing the target environment for the installation of every library referenced, as well as checking the relative position of the model file with respect to that of the API script. Of utmost importance was that the API was meant to be usable and gave understandable outputs: the team set up a default route to return a descriptive message upon making a prediction, which improved the model's accessibility in its deployed state in the eyes of the users with regard to interpreting output.

Conclusion

Summarized, this project developed an end-to-end machine learning pipeline for property revenue prediction based on the "Airbnb Market Analysis & Real Estate Sales Data (2019)" dataset. The process involved heavy cleaning, thorough preprocessing, exploratory data analysis, model building, hyperparameter tuning, and deployment as a RESTful API. Strong preprocessing techniques were adopted despite challenges relating to inconsistencies in data and missing values. When used together, median imputation, outlier capping, and normalization were able to fix the data sufficiently.

The RandomForestRegressor was tuned using a simple grid search method, and reached a promising performance on normalized evaluation metrics. The Flask framework was used for the deployment of the final model combined with a user-friendly API for real-time predictions. Comprehensive guidelines were created on how to run the code and test the API using web browsers, Postman, or cURL commands. During the exploratory phase, all generated visualizations-such as histograms, boxplots, and correlation heatmaps-played a fundamental role in driving the decisions around preprocessing and modeling.

This project is illustrative with respect to a practical approach to using Airbnb and real estate sales data in predictive analytics and consequently provides a view of what stakeholders in the real estate market can expect. This report also provided a strong framework upon which similar predictive modeling tasks can be mounted, which will then prove useful for further improvements and applications in the domain of real estate market analysis.

References

- ComputingVictor. (2019). *Zillow market analysis and real estate sales data* [Data set]. Kaggle. https://www.kaggle.com/datasets/computingvictor/zillow-market-analysis-and-real-estate-sales-data?resource=download
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.
- Grinberg, M. (2018). *Flask web development: Developing web applications with Python* (2nd ed.). O'Reilly Media.
- McKinney, W. (2010). Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference* (pp. 51–56).
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362.