Airbnb Competition

Team BPK

Jared Bocchinfuso

Krithik Kannan

Aaron Panitz

Data Cleaning/Management Process

```r
# 0. Packages
library(dplyr)
library(lubridate)
library(lightgbm)
library(tidyr)
```

We start by loading the necessary datasets, which include property information, reservation data for Q3 2016, and listings data for Q1 and Q2 2016.

Next, we merge the Q3 reservation data with the property information to create a single dataframe (df). This allows us to have the target variable (NumReserveDays2016Q3) alongside the property features.

```r
# 2. Merge Q3 target
df <- property_info %>%
  left_join(reserve_2016Q3_train %>% select(PropertyID, NumReserveDays2016Q3),
            by = "PropertyID")
```

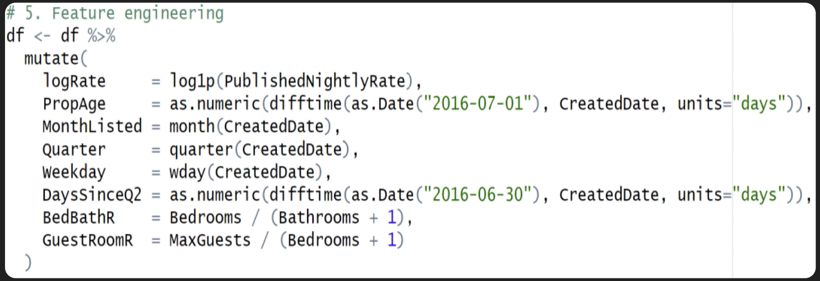
We calculate the percentage of missing values for each column and drop columns where more than 50% of the data is missing. This helps in reducing noise and improving model performance.

```r
# 3. Drop cols >50% missing
na_pct <- colMeans(is.na(df))
df     <- df %>% select(-names(na_pct[na_pct > 0.5]))
```

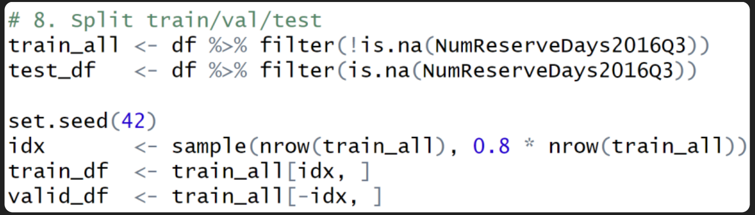
We combine the listings data from Q1 and Q2 and create aggregate features such as the number of booked days, average price, and total revenue for each property. These features are then merged into our main dataframe (df).

```r
# 4. Q1/Q2 booking aggregates
all_listings <- bind_rows(listing_2016Q1, listing_2016Q2)
q_feats <- all_listings %>%
  mutate(Booked = ifelse(!is.na(ReservationID), 1L, 0L)) %>%
  group_by(PropertyID) %>%
  summarise(
    Q1_Days    = sum(Booked[Date < "2016-04-01"], na.rm=TRUE),
    Q1_Price   = mean(Price[Date < "2016-04-01"], na.rm=TRUE),
    Q2_Days    = sum(Booked[Date >= "2016-04-01" & Date < "2016-07-01"], na.rm=TRUE),
    Q2_Price   = mean(Price[Date >= "2016-04-01" & Date < "2016-07-01"], na.rm=TRUE),
    Q1_Revenue = sum(Price[Date < "2016-04-01"] * Booked[Date < "2016-04-01"], na.rm=TRUE),
    Q2_Revenue = sum(Price[Date >= "2016-04-01" & Date < "2016-07-01"] * Booked[Date >= "2016-04-01"
    & Date < "2016-07-01"], na.rm=TRUE),
    .groups = "drop"
  )
```

Data Cleaning/Management Process

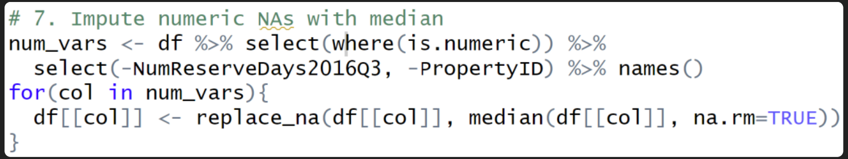


```r
# 5. Feature engineering
df <- df %>%
  mutate(
    logRate     = log1p(PublishedNightlyRate),
    PropAge     = as.numeric(difftime(as.Date("2016-07-01"), CreatedDate, units="days")),
    MonthListed = month(CreatedDate),
    Quarter     = quarter(CreatedDate),
    Weekday     = wday(CreatedDate),
    DaysSinceQ2 = as.numeric(difftime(as.Date("2016-06-30"), CreatedDate, units="days")),
    BedBathR    = Bedrooms / (Bathrooms + 1),
    GuestRoomR  = MaxGuests / (Bedrooms + 1)
  )
```

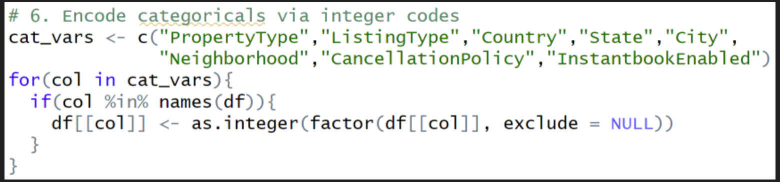


```r
# 8. Split train/val/test
train_all <- df %>% filter(!is.na(NumReserveDays2016Q3))
test_df   <- df %>% filter(is.na(NumReserveDays2016Q3))

set.seed(42)
idx       <- sample(nrow(train_all), 0.8 * nrow(train_all))
train_df  <- train_all[idx, ]
valid_df  <- train_all[-idx, ]
```



```r
# 7. Impute numeric NAs with median
num_vars <- df %>% select(where(is.numeric)) %>%
  select(-NumReserveDays2016Q3, -PropertyID) %>% names()
for(col in num_vars){
  df[[col]] <- replace_na(df[[col]], median(df[[col]], na.rm=TRUE))
}
```



```r
# 6. Encode categoricals via integer codes
cat_vars <- c("PropertyType","ListingType","Country","State","City",
              "Neighborhood","CancellationPolicy","InstantbookEnabled")
for(col in cat_vars){
  if(col %in% names(df)){
    df[[col]] <- as.integer(factor(df[[col]], exclude = NULL))
  }
}
```

5. We create new features to capture more information about each property.

6. Categorical variables are encoded as integer codes to be used in the model. This includes features like property type, listing type, location details, and booking policies.

7. For numeric columns, we impute missing values with the median of each column. This ensures that our dataset is complete and ready for modeling.

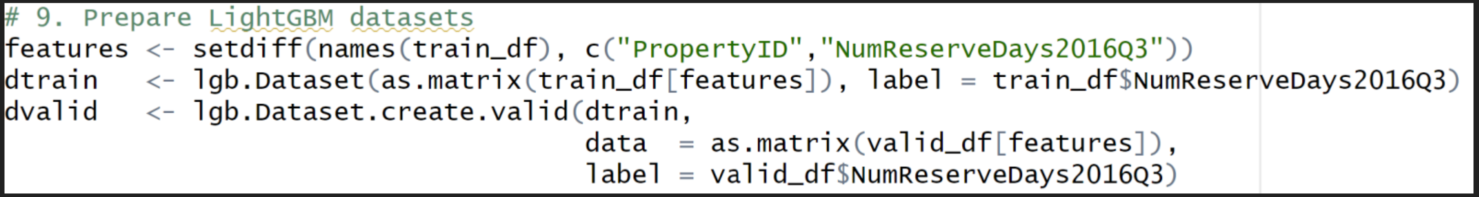
8. We split the data into training and testing sets (validation). The training set includes properties with known reservation days for Q3, while the test set includes properties without this information.

Data Cleaning/Management Process

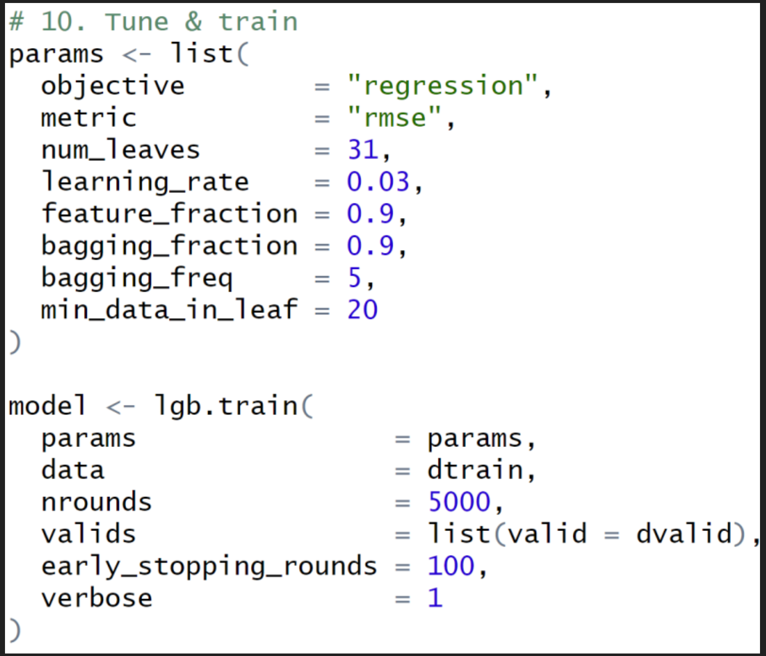
9. We prepare the datasets for LightGBM, specifying the features and labels for training and validation.
10. We define the parameters for the LightGBM model and train it using the training data. We also include early stopping to prevent overfitting.
11. We predict the reservation days for the validation set and calculate the RMSE to evaluate model performance.
12. Finally, we use the trained model to predict the reservation days for the test set and save the predictions.

RMSE in our model: 15.015

Final RMSE: 14.902



```
# 9. Prepare LightGBM datasets
features <- setdiff(names(train_df), c("PropertyID","NumReserveDays2016Q3"))
dtrain   <- lgb.Dataset(as.matrix(train_df[features]), label = train_df$NumReserveDays2016Q3)
dvalid   <- lgb.Dataset.create.valid(dtrain,
                                      data  = as.matrix(valid_df[features]),
                                      label = valid_df$NumReserveDays2016Q3)
```



```
# 10. Tune & train
params <- list(
  objective        = "regression",
  metric           = "rmse",
  num_leaves       = 31,
  learning_rate    = 0.03,
  feature_fraction = 0.9,
  bagging_fraction = 0.9,
  bagging_freq     = 5,
  min_data_in_leaf = 20
)

model <- lgb.train(
  params                = params,
  data                  = dtrain,
  nrounds               = 5000,
  valids                = list(valid = dvalid),
  early_stopping_rounds = 100,
  verbose               = 1
)
```

Feature Selection

Initial Data Exploration:

- Missing Values: Dropped columns with more than 50% missing values to ensure data quality.
- Correlation Analysis: Identified features with high correlation to the target

Relevance to Business Objectives:

- Selected features that are crucial for understanding booking patterns, such as PublishedNightlyRate, CreatedDate,

Aggregated Features:

- Created booking aggregates from Q1 and Q2 data (Q1_Days, Q1_Price, Q2_Days, Q2_Price, etc.) to capture historical booking patterns.

Feature Construction

Derived Features:

- Log Transformation: Applied log transformation to PublishedNightlyRate to handle skewness.
- Property Age: Calculated PropAge as the number of days since the property was listed.
- Temporal Features: Extracted MonthListed, Quarter, Weekday, and DaysSinceQ2 from CreatedDate.

Ratios:

- BedBathR: Ratio of bedrooms to bathrooms.
- GuestRoomR: Ratio of maximum guests to bedrooms.

Categorical Encoding:

- Converted categorical variables (PropertyType, ListingType, etc.) to integer codes for model compatibility.

Imputation:

- Imputed missing numeric values with the median to maintain data consistency.

Prediction models and methods

Model	RMSE
Linear Regression	~ 30-35
Random Forest	~18-19
XGBoost	~ 17-18
Light GBM (original model)	~ 17.68
Applied **winsorization**, **log-transform**, and **skew stabilization** to nightly rates.	~ 17.01
Switched to **Poisson** loss, used 5-fold **cross-validation**, and turned on **early stopping**.	~ 16.67
Added **booking history** aggregates, listing **recency**, and **weekday** features.	~ 15.03

Most Important Features

- Q2_Revenue
- Q2_Revenue
- Q1_Revenue
- Q1 Days
- LogRate
- PropAge
- BedBathR
- GuestRoom R

Why?

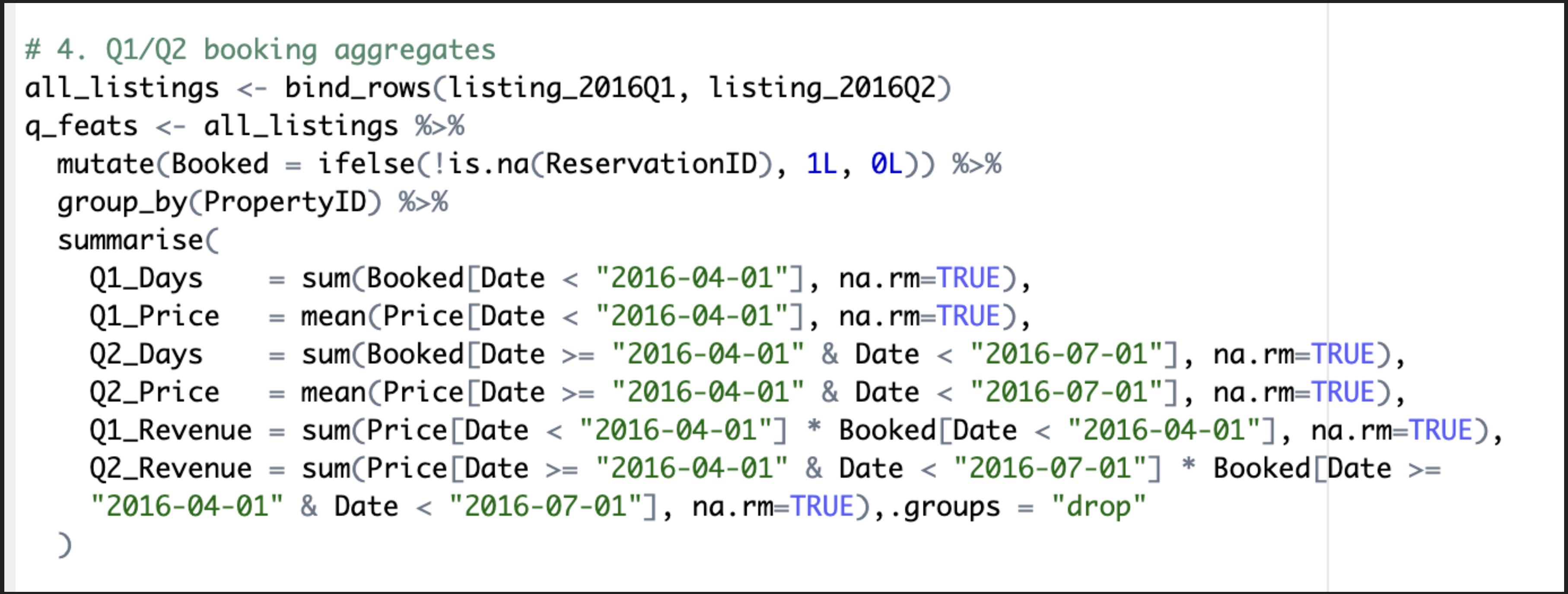
•**Q1/Q2 Days & Revenue:** direct measures of past demand.

•**logRate:** normalizes price to capture elasticity.

•**PropAge:** listing age reflects trust and freshness.

•**BedBathR & GuestRoomR:** summarize size and capacity.

R Implementation (Vectorized Data Summary)

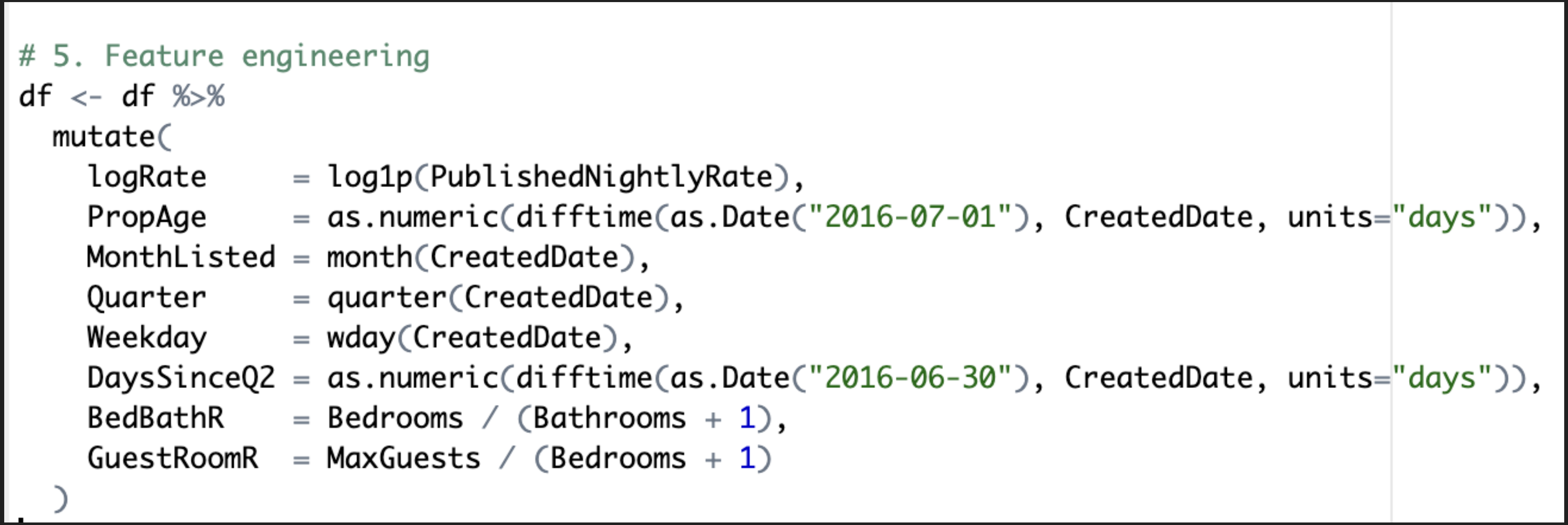


```r
# 4. Q1/Q2 booking aggregates
all_listings <- bind_rows(listing_2016Q1, listing_2016Q2)
q_feats <- all_listings %>%
  mutate(Booked = ifelse(!is.na(ReservationID), 1L, 0L)) %>%
  group_by(PropertyID) %>%
  summarise(
    Q1_Days     = sum(Booked[Date < "2016-04-01"], na.rm=TRUE),
    Q1_Price    = mean(Price[Date < "2016-04-01"], na.rm=TRUE),
    Q2_Days     = sum(Booked[Date >= "2016-04-01" & Date < "2016-07-01"], na.rm=TRUE),
    Q2_Price    = mean(Price[Date >= "2016-04-01" & Date < "2016-07-01"], na.rm=TRUE),
    Q1_Revenue = sum(Price[Date < "2016-04-01"] * Booked[Date < "2016-04-01"], na.rm=TRUE),
    Q2_Revenue = sum(Price[Date >= "2016-04-01" & Date < "2016-07-01"] * Booked[Date >=
    "2016-04-01" & Date < "2016-07-01"], na.rm=TRUE),.groups = "drop"
  )
```

We collape two large tables into Q1/Q2 booking Aggregates in one poped chain and no manual loops over properties

Benefit – All aggregation logic runs in optimized C++ through dplyr, making it fast and concise

R Implementation (Multi-Feature Engineering)

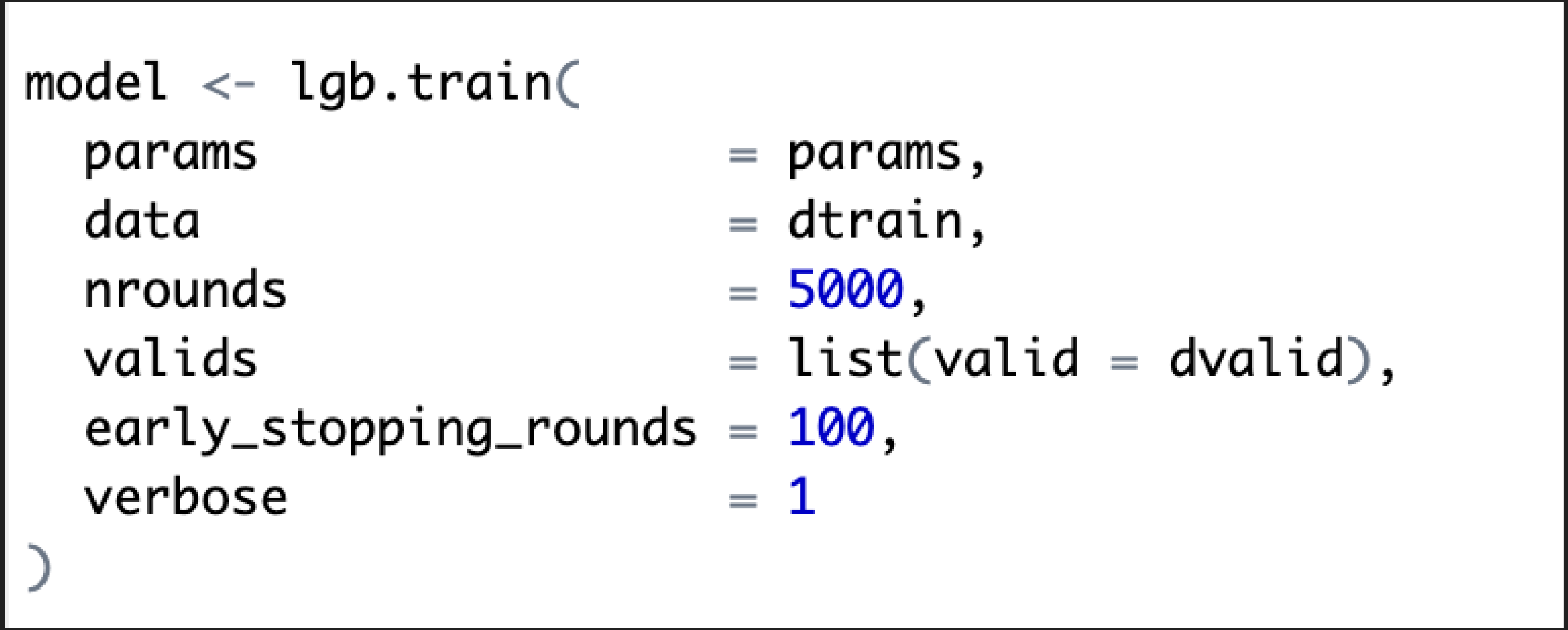


```r
# 5. Feature engineering
df <- df %>%
  mutate(
    logRate     = log1p(PublishedNightlyRate),
    PropAge     = as.numeric(difftime(as.Date("2016-07-01"), CreatedDate, units="days")),
    MonthListed = month(CreatedDate),
    Quarter     = quarter(CreatedDate),
    Weekday     = wday(CreatedDate),
    DaysSinceQ2 = as.numeric(difftime(as.Date("2016-06-30"), CreatedDate, units="days")),
    BedBathR    = Bedrooms / (Bathrooms + 1),
    GuestRoomR  = MaxGuests / (Bedrooms + 1)
  )
```

We compute log-rates, listing age, weekday, and room ratios in a single mutate(), avoiding repeated passes over the data.

Benefit – Vectorized transformations eliminate explicit loops, keeping the code DRY and performant.

R Implementation (Early Stopping)



```r
model <- lgb.train(
  params                = params,
  data                  = dtrain,
  nrounds               = 5000,
  valids                = list(valid = dvalid),
  early_stopping_rounds = 100,
  verbose               = 1
)
```

We let LightGBM handle the iterative search for the optimal number of boosting rounds, rather than writing our own grid-search loops.

Benefit – Early stopping halts training once validation RMSE stalls, preventing overfit and removing the need for manual round-count tuning.

Reflection On The Prediction Challenge

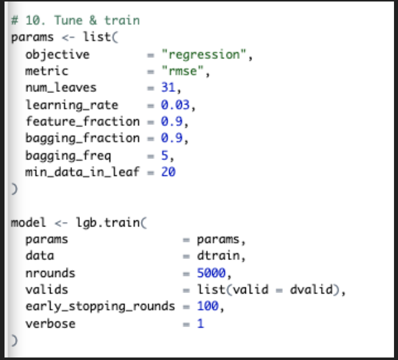
- What we have learned about improving prediction accuracy
- Additional analyses we have conducted

What we have learned about improving prediction accuracy

- Respecting the timeline is critical
    - Only use Q1 and Q2 booking and pricing data to create features
    - Including anything from Q3 prices or bookings would have created data leakage

- Focus on adding high-quality features only
    - Designed features that truly matter for Airbnb bookings: price, demand, and the guest experience
    - Adding unnecessary features hurts performance if the new features are noisy, redundant, or not predictive

- We Tuned our model efficiently
    - Customized key hyperparameters (num_leaves, learning_rate, etc.)
    - Used early stopping to save time



```
# 10. Tune & train
params <- list(
  objective        = "regression",
  metric           = "rmse",
  num_leaves       = 31,
  learning_rate    = 0.03,
  feature_fraction = 0.9,
  bagging_fraction = 0.9,
  bagging_freq     = 5,
  min_data_in_leaf = 20
)

model <- lgb.train(
  params                = params,
  data                  = dtrain,
  nrounds               = 5000,
  valids                = list(valid = dvalid),
  early_stopping_rounds = 100,
  verbose               = 1
)
```

Additional analyses we conducted

- 5-Fold Cross-Validation
  - To confirm model stability
  - Average RMSE: 14.61 and standard deviation: 0.48
  - Split training data into 5 parts, trained on 4 parts, and validated on the 5th
  - Low variance gave us confidence that the final model would perform reliably on new, unseen data

- Feature Importance Analysis (LightGBM)
  - LightGBM shows three metrics: Gain, Cover, Frequency
  - Ranks features based on how much they reduce RMSE across all decision trees
  - Some of our top features included Q2_Days, PropAge, and Q2_Revenue – all features we created

Biggest Challenge

- The most significant challenge we encountered while working on the project?
- How we overcame it?

Biggest challenge

- Initial submission had internal RMSE ~16 but scored ~ 38 on leaderboard
  - Huge gap, and we weren't sure why

<u>How we overcame this:</u>

- We realized our test-set was misaligned
- We used merge to join our predictions onto PropertyID_test (creating a new submission data set), so our pred vector got shuffled relative to the original order of PropertyID_test, so almost every prediction was scored against the wrong property
- We manually examined the PropertyID_test data, and our new submission set and saw that the listings were not in the original order

airbnb

Thank you!