1.0 導入

Airbnb は、家や部屋を共有するための有名なプラットフォームであり、宿泊施設を探しているゲストと呼ばれるユーザーは、さまざまなタイプの宿泊施設を簡単に見つけることができます。 このプロジェクトの主な目的は、ナイーブベイズとデシジョンツリーの 2 つの機械学習モデルを使用してパフォーマンスを比較し、リスティングの人気に寄与する特徴を見つけ、その属性に基づいてリスティングがトップ 100 になる可能性があるかどうかを特定することです。最初に、データセット全体を正確に理解するためのデータ収集、データクリーニング、および機能エンジニアリングを含むデータ前処理を行います。次に、統計情報と視覚化に基づく分析結果を報告します。最後に各モデルの精度評価を含みます。

2.0 データ前処理

2.1 データコレクション

Airbnb に関しては、リストにデータを公開していませんが、InsideAirbnb という名前の独立した非営利のウェブサイトが賃貸物件のデータを報告しています。 Airbnb を分析する論文のほとんどは、この Web サイトを参照し、主要都市のリストのサンプルに関するデータを抽出しています。 通常、データセットには、名前、説明、ホストのプロフィール、設備の可用性、場所情報などのテキスト情報が含まれています。 さらに、ベッドとバスルームの数、物件のタイプ、レビュースコアなどの詳細情報を取得することもできます。 このホワイトペーパーでは、2009 年 1 月から 2015 年 10 月 3 日までの Airbnb のリストを含むデータセットを利用しています。米国のデータセットには多くの主要都市データが含まれています。

2.2 Data Cleaning

この調査では、ターゲットは Airbnb リストの最大の需要の 1 つであるワシントン D.C とします。 ワシントン D.C のリストのデータを選択するために、「state」変数を使用してデータセットをフィルタリングします。 同時に、データセットをよりよく理解するために、データセットのクリーンアップを開始します。 欠落している値が多く、回答の解決につながる実用的または興味深い情報がないように見える列を取り除きます。 実際、この準備されたデータセットには、3723 の行と 92 の列があります。 最初、データセットには次のような URL リンクを構成する 8 つの列があります。

- listing_url
- thumbnail_url
- medium_url
- picture_url
- xl_picture_url
- host_url
- host_thumbnail_url
- host_picture_url

これらの Web リンク列には値が含まれていないか、代わりに他の変数を使用できるため、有用なリソースではありません。 そのため、取り除きます。

現時点で、データセットにはまだ 84 列と 3723 行があるため（図 2-1）、値が含まれていないか、同じ値または多くの欠落値が含まれている他の列を確認しました。 次に、州の列をチェックして、ワシントン D.C.にあるリストのみをフィルタリングしました。この州の列には、DC が 3696、MD が 24、NY が 1、VA が 1、ワシントン DC が 1 つ、含まれています（図 2-2）。 MD、NY、および VA を持つデータは削除され、ワシントン DC を含むデータは DC として更新されます。よって、データセットには 3697 列を含むこととなりました。（図 2-3）。

```
> dataset = read.csv('/Users/sakanash
v')
> #check number of rows and columns
> dim(dataset)
[1] 3723   84
```

Figure 2-1. Dimension of the dataset

```
> dataset %>% group_by(state) %>% summarize(count=n())
# A tibble: 5 x 2
  state         count
  <fct>         <int>
1 DC             3696
2 MD               24
3 NY                1
4 VA                1
5 Washington DC     1
```

Figure 2-2. The number of unique data in State

```
> dataset %>% group_by(state) %>% summarize(count=n())
# A tibble: 4 x 2
  state count
  <fct> <int>
1 DC     3697
2 MD       24
3 NY        1
4 VA        1
> dataset <- dataset[!(dataset$state == 'MD' | dataset$state == 'NY' | dataset$state == 'VA'),]
> dataset %>% group_by(state) %>% summarize(count=n())
# A tibble: 1 x 2
  state count
  <fct> <int>
1 DC     3697
```

Figure 2-3. Summary of the cleaned ‘State’

さらに、ターゲットの場所はすでにわかっているため、以下の場所に関連するこれら 5 つの変数は削除されます。

- city
- country
- country_code
- market
- smart_location

```
#select state
dataset %>% group_by(state) %>% summarize(count=n())
dataset$state[dataset$state == 'Washington DC'] <- 'DC'
dataset %>% group_by(state) %>% summarize(count=n())
dataset <- dataset[!(dataset$state == 'MD' | dataset$state == 'NY' | dataset$state == 'VA'),]
dataset %>% group_by(state) %>% summarize(count=n())
```

次に、多くの欠落値、テキスト、および必要ではない特徴をもつ変数を削除するために、欠落値の割合を調査しました（図 2-4）。 この図は、欠落値を含む変数を示していますが、文字変数に null 値を含むデータを識別できていません。 したがって、私は他の方法でそれらをチェックしました。 その結果、以下の 7 列には 25%を超える欠落値があり、これらの列は結果に大きな影響を与える可能性があります。

- neighbourhood_group_cleansed
- square_feet
- license
- security_deposit
- cleaning_fee
- weekly_price
- monthly_price

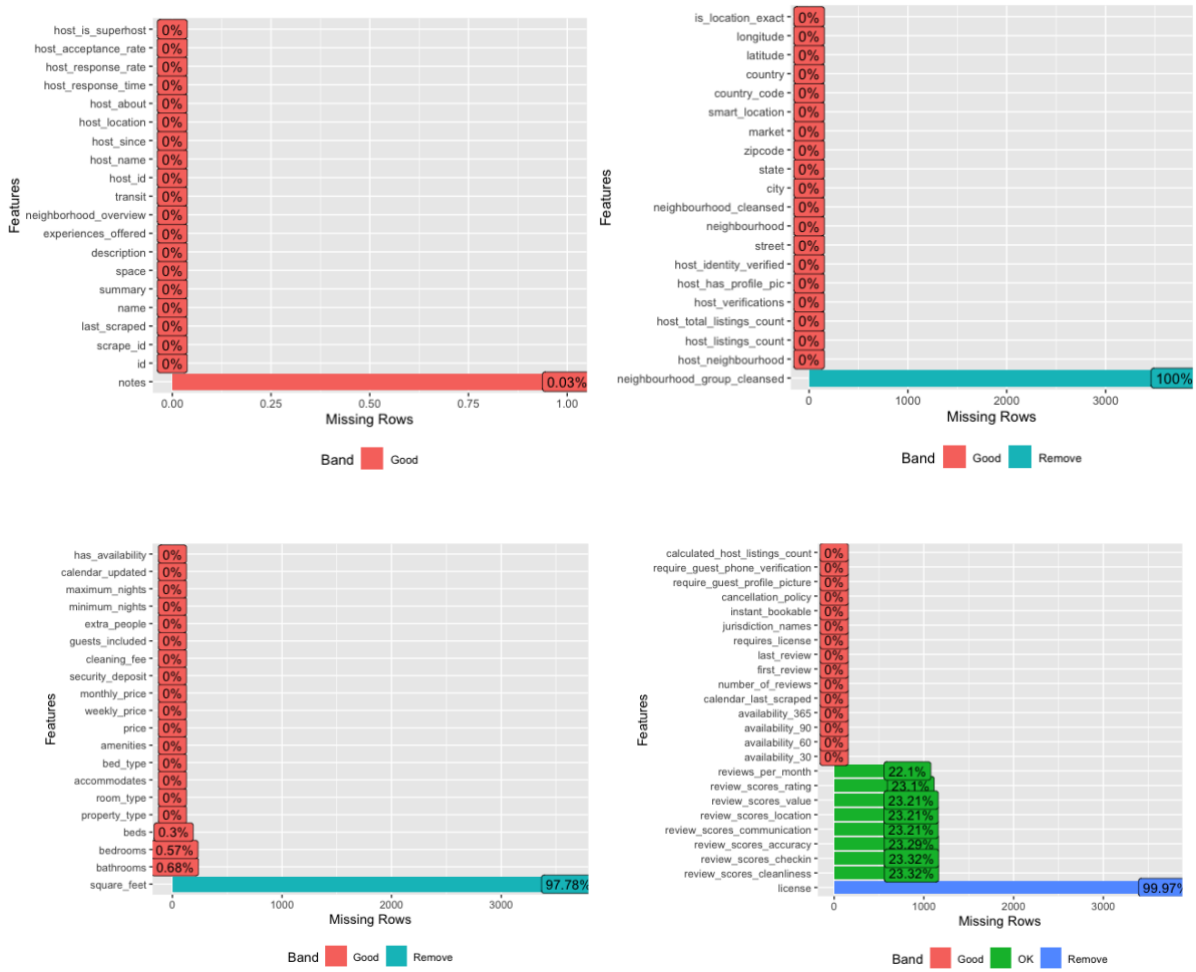

Figure 2-4 Missing Values Plot

```
summary(dataset)

plot_missing(dataset[1:20])
plot_missing(dataset[21:40])
plot_missing(dataset[41:60])
plot_missing(dataset[61:84])

summary(dataset$security_deposit == '') # true = 2297
2297/3723 * 100 # 61.69%
summary(dataset$cleaning_fee == '') # true = 1388
1388/3723 * 100 # 37.28%
summary(dataset$weekly_price == '') # true = 1599 -> weekly_price
1599/3723 * 100 # 42.94%
summary(dataset$monthly_price == '') # true = 1916 -> monthly_price
```

```
1916/3723 * 100 # 51.46%
```

さらに、以下の 7 つの変数には、有用なデータを変換するために多くの前処理が必要なテキストと、すべて同じ値を持つ 2 つの変数が含まれています。

- summary
- space
- description
- neighbourhood_overview
- notes
- transit
- host_about
- requires_license
- experiences_offered

```
summary(dataset$requires_license) # f = 3723 same values
dataset %>% group_by(experiences_offered) %>% summarize(count=n()) #none =3723
```

さらに、以下のこれらの変数は今回必要ではありません。 さらに、host_listings_count と host_total_listings_count は同じであり、2 つの列の精度は calculated_host_listings よりも低いため、同じホストのリストの数を知るために、代わりに calculated_host_listings を使用することをお勧めします。 したがって、これらの以下の列は削除されます。

- name
- scrape_id
- last_scrape
- host_listings_count
- host_total_listings_count
- host_id

私の目的に関しては、将来のデータを含むこれらの変数は興味がなく、それほど重要ではありません。

- availability_30
- availability_60

- availability_90
- availability_365
- calender_updated
- calender_last_scraped
- has_availability

host に関連するデータについては、host_acceptance_rate、host_response_rate、および host_response_time は、host_is_superhost ほど重要ではありません。 Airbnb は、スーパーホストであることが他の競合他社よりも魅力的であり、より高い収益を得ることができるホストに特別なサービスを提供します。 スーパーホストになるために、ホストは 4.8 / 5 の全体的な評価、1%未満のキャンセル率、およびユーザーへの 90%を超える応答率を維持できます。 host_is_suprehost としてカテゴリデータを使用します。

さらに、ホストの場所を識別するために、host_neighbourhood と street の代わりに host_location と neighbourhood_cleansed がそれぞれ使用されます。 host_verifications は、ホスト検証メソッドのリストが host_identity_verified としてカテゴリデータに含まれていることを意味します。 識別の手段はそれほど重要ではありません。 さらに、以下の 6 つの変数が削除されます。

- host_acceptance_rate
- host_response_rate
- host_response_time
- host_neighbourhood
- street
- host_verifications

データセットにはいくつかの評価スコアがありますが、review_socres_rating は他の評価スコアの合計として計算されるため、代わりに各評価スコアを使用します。 is_location_exact は重要ではなく、精度が低く、maximum_nights のほとんどの値は 30 日を超えており、neighborhood_cleansed は neighborhood よりも多くの neighborhood を視覚化するのに役立ちます。 さらに、各リストに滞在できる人数を示す宿泊施設は、guests_includes よりも正確です。 理由として、これらの変数を取り除きました。

- review_scores_rating
- is_location_exact
- maximum_nights
- neighborhood

- guests_includes

最後に、さらなる分析でエラーを生成する残りの欠損値を変換しました。 ベッド、バスルーム、ベッドルームの数に関しては、ほとんどのリストに少なくとも 1 つあり、値が平均であるため、欠損地は 1 として更新されます。 各レビュースコアの平均は約 9 であり、優勢であるため、欠損値は 9 に置き換えられます。

```
dataset$beds[is.na(dataset$beds)] <- 1
dataset$bathrooms[is.na(dataset$bathrooms)] <- 1
dataset$bedrooms[is.na(dataset$bedrooms)] <- 1
dataset$review_scores_accuracy[is.na(dataset$review_scores_accuracy)] <- 9
dataset$review_scores_cleanliness[is.na(dataset$review_scores_cleanliness)] <- 9
dataset$review_scores_checkin[is.na(dataset$review_scores_checkin)] <- 9
dataset$review_scores_communication[is.na(dataset$review_scores_communication)] <- 9
dataset$review_scores_location[is.na(dataset$review_scores_location)] <- 9
dataset$review_scores_value[is.na(dataset$review_scores_value)] <- 9
dataset <- subset(dataset, rowSums(is.na(dataset))!=0)
```

その時点で、データクリーニングはほぼ完了し、新しいデータセットは df として準備されました。

```
df <- dataset[c('id', 'host_name', 'host_since', 'host_location',
                'host_is_superhost', 'host_has_profile_pic', 'host_identity_verified',
                'neighbourhood_cleansed', 'state', 'zipcode', 'latitude', 'longitude',
                'property_type', 'room_type', 'accommodates', 'bathrooms', 'bedrooms',
                'beds', 'bed_type', 'amenities', 'price', 'extra_people',
                'minimum_nights', 'number_of_reviews', 'first_review', 'last_review',
                'review_scores_accuracy', 'review_scores_cleanliness',
'review_scores_checkin',
                'review_scores_communication', 'review_scores_location',
'review_scores_value',
                'instant_bookable', 'cancellation_policy', 'require_guest_profile_picture',
                'require_guest_phone_verification', 'calculated_host_listings_count',
                'reviews_per_month')]
```

調査の目的に関連して、リストのレビュー数が 0 に等しいことはそれほど重要ではないため、データは削除されました。

```
df <- df[!(df$number_of_reviews == 0),]
```

多くの欠落値と重要でない不正確なデータを含む列をクリーンアップした後、データセットは 38 列と 2880 行に減少しました。 表 2-1 に、データセット内の変数とその説明のリストを示します。

Table 2-1 List of variables in the cleaned dataset

No.	variables	description
1	id	listings ID (unique ID)
2	host_name	host name
3	host_since	to calculate host experience duration since the first listing
4	host_location	we can use it to establish if host is local or not
5	host_is_superhost	categorical t or f
6	host_has_profile_pic	categorical t or f
7	host_identity_verified	categorical t or f
8	neighborhood_cleansed	ust it instead of neighborhood_group_cleansed
9	state	focused on listings in DC
10	zipcode	can be used to visualize
11	latitude	to visualise the data on the map
12	longitude	to visualise the data on the map
13	property_type	categorical variable
14	room_type	categorical variable
15	accommodates	describing listings
16	bathrooms	describing listings
17	bedrooms	describing listings
18	beds	describing listings
19	bed_type	categorical variable describing listings
20	amenities	due to number of unique features (over 100) will focus on the total number of amenities
21	price	per night for number of guests_included
22	extra_people	cost of additional person per night
23	minimun_nights	minimum nights
24	number_of_reviews	total number of reviews in entire listing history
25	first_review	to calculate reviews_per_month
26	last_review	can be used to filter out no longer active listings
27	review_scores_accuracy	numbers between 2 and 10
28	review_scores_cleanliness	numbers between 2 and 10
29	review_scores_checkin	numbers between 2 and 10
30	review_scores_communication	numbers between 2 and 10
31	review_scores_location	numbers between 2 and 10
32	review_scores_value	numbers between 2 and 10
33	instant_bookable	categorical t or f
34	cancellation_policy	ordinal value with 5 categories (from lowest to highest level of flexibility)
35	require_guest_profile_picture	categorical value - t or false
36	require_guest_phone_verification	categorical value - t or false
37	calculated_host_listings_count	instead of host_listings_count and host_total_listings_count, is actual number of host listings
38	reviews_per_month	cam be used count reviews

2.3 特徴エンジニアリング

まず、特性値の使用を避けるために、t / f を 1/0 に変換します。

```
df$host_is_superhost <- as.numeric(ifelse(df$host_is_superhost == 't', 1, 0))
df$host_has_profile_pic <- as.numeric(ifelse(df$host_has_profile_pic == 't', 1, 0))
df$host_identity_verified <- as.numeric(ifelse(df$host_identity_verified == 't', 1, 0))
df$instant_bookable <- as.numeric(ifelse(df$instant_bookable == 't', 1, 0))
df$require_guest_profile_picture <- as.numeric(ifelse(df$require_guest_profile_picture ==
't', 1, 0))
```

```
df$require_guest_phone_verification <-
as.numeric(ifelse(df$require_guest_phone_verification == 't', 1, 0))
```

次に、さらに分析するために数値に変換するには、「$」を含む price と extra_people の処理が必要です。

```
df$price <- df$price %>% str_extract_all("¥¥ (?[0-9,.]+¥¥)?") %>% gsub(",", "", .) %>%
as.numeric()
df$extra_people <- df$extra_people %>% str_extract_all("¥¥(?[0-9,.]+¥¥)?") %>%
gsub(",", "", .) %>% as.numeric()
```

データセットをさらに分析しやすくするために、新たな特徴をデータセットに追加します。その特徴はホスティングとリストの期間によって計算された値を含みます（listing_duration & hosting_duration）。

ホストがそうであるかどうかを識別するために、州と国を含む host_location 列を使用しました。 データセットにはワシントン D.C のリストが含まれていたため、ワシントン D.C に住むホストを 1、その他を 0 と定義しました（host_local)。アメニティの充実度とリストの人気の関係を確認するために、アメニティの数を数えたいと思います。 ただし、アメニティの値には括弧が含まれ、各アメニティはコンマで区切られます。 そこで、コンマの数を数え、アメニティの数としてコンマの数に 1 を加えました(total_amenities)。 さらに、一人当たりの価格を計算しました(price_per_person)。

```
df   <- df %>%
  mutate(listing_duration = as.numeric(difftime(df$last_review, df$first_review, units =
'days')),
         hosting_duration = as.numeric(difftime(df$last_review, df$host_since, units =
'days')),
         host_local = as.numeric(str_detect(host_location, 'Washington')),
         total_amenities = ifelse(str_count(amenities) > 2 , str_count(amenities, ',') +
1,0),
         price_per_person = price / accommodates)
```

この目的のために、top_100 として新しい機能を作成します。この機能は、top100 と 0：100 のうちの 2 つのグループに分類されます。

```
df$top_100 <- ifelse(rank(-df$number_of_reviews) <= 100, 1, 0)
```

表 2-2 に追加された変数を示し、データセットは 44 列になります。

Table 2-2. The added variables

39	listing_duration	Duration of listing in days
40	hosting_duration	Duration of hosting in days
41	host_local	whether host is local or not (1 yes 0 no)
42	total_amenities	number of amenitites
43	price_per_person	price / accommodates
44	is_100	based on number of reviews. In rank 100: 1, out of rank 100: 0

また、コードエラーを防ぐために、特殊文字をアンダースコアに置き換えました。

```
df %>% group_by(property_type) %>% summarize(count=n())
df$property_type <- str_replace_all(df$property_type, "&", "and")
df %>% group_by(property_type) %>% summarize(count=n())
df$property_type <- str_replace_all(df$property_type, "[^[:alnum:]]", "_")
df %>% group_by(property_type) %>% summarize(count=n())

df %>% group_by(room_type) %>% summarize(count=n())
df$room_type <- str_replace_all(df$room_type, "[^[:alnum:]]", "_")
df %>% group_by(room_type) %>% summarize(count=n())

df %>% group_by(bed_type) %>% summarize(count=n())
df$bed_type <- str_replace_all(df$bed_type, "[^[:alnum:]]", "_")
df %>% group_by(bed_type) %>% summarize(count=n())
```

3.0 Exploratory Data Analysis（EDA）探索的データ分析

3.1 Discrete　離散型変数

目標値でグループ化された相対密度を示すために、以下の棒グラフが生成されます（図 3-1）。

- host_is_superhost–スーパーホストはトップ 100 の中で 25%を占めますが、のこりはスーパーホストではありません。人気獲得をサポートする強力な要素の一つと考えらないかもしれません。
- host_has_profile_pic–トップ 100 に関わらず、予約を取得するためにはプロフィール画像を付ける必要があります。
- host_identity_verified-　Top100 に入りたいのであれば、確実に Airbnb からホストの身元確認をされるべきです。

- instant_bookable- 直前の予約を受け入れるリスティングはトップ 100 の中で 25%を占めますが、人気獲得をサポートする強力な要素の一つと考えらないかもしれません。
- require_guest_profile_picture– ゲストの写真を要求しない方が人気になる可能性があります。
- require_guest_phone_verification– 同様に、ゲストの電話番号を要求しない方が人気を得られそうです。
- host_local- Top100 になるにはホストがその地域のローカル住民であることが推奨されます。

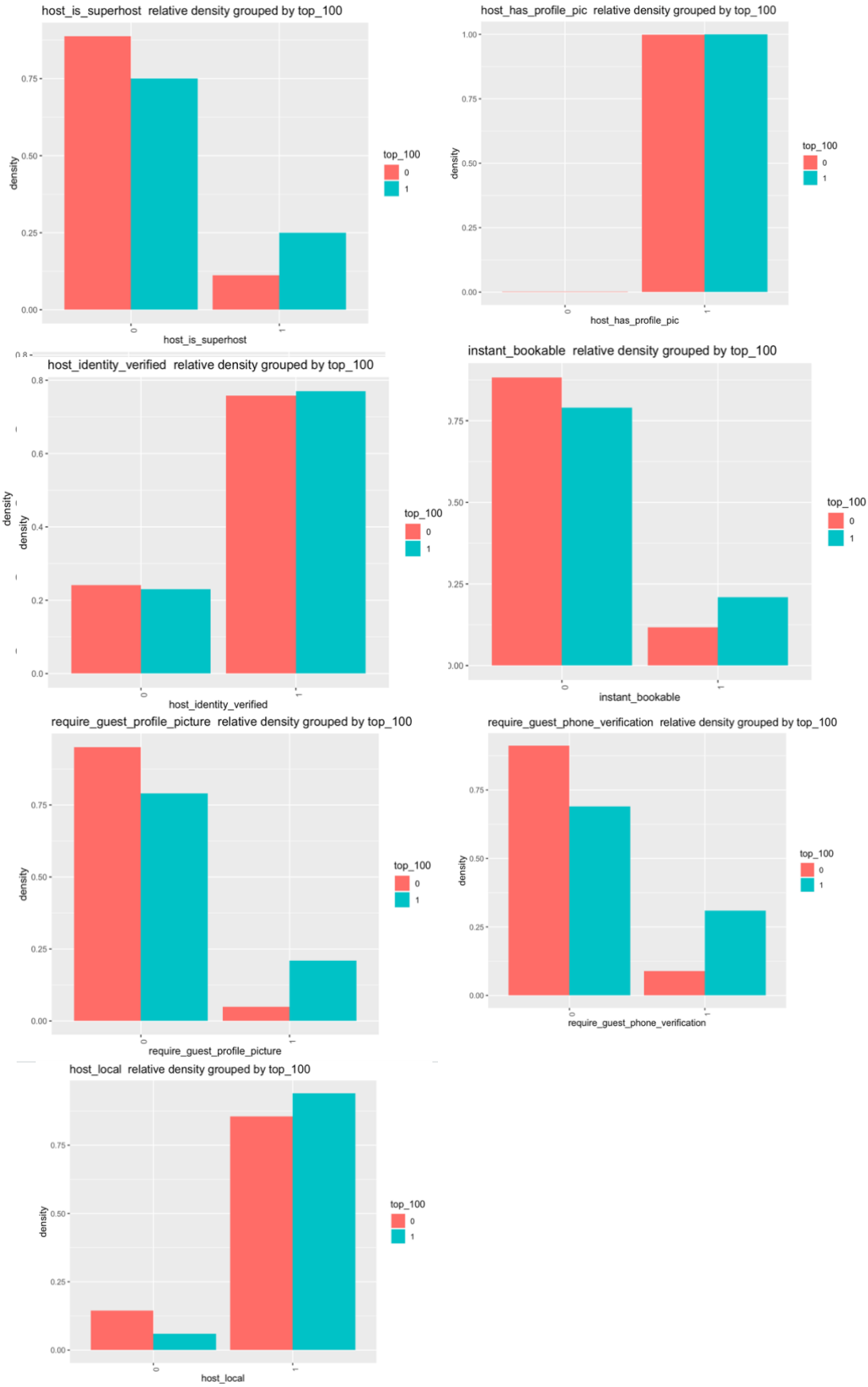

Figure 5.1 The bar charts in the discrete variables

```
discrete <- c("host_is_superhost", "host_has_profile_pic", "host_identity_verified",
                      "instant_bookable", "require_guest_profile_picture",
"require_guest_phone_verification",
                      "host_local")

for (colname in discrete) {

  temp <- subset(df, top_100 == 1)
  temp <- temp %>%
    group_by(top_100, temp[,colname]) %>%
    summarise(density = n()/nrow(.))
  colnames(temp)[2] <- colname

  temp1 <- subset(df, top_100 == 0)
  temp1 <- temp1 %>%
    group_by(top_100, temp1[,colname]) %>%
    summarise(density = n()/nrow(.))
  colnames(temp1)[2] <- colname

  temp2 <- rbind(temp, temp1)

  plot <- ggplot(data=temp2, aes(x=as.factor(temp2[[colname]]), y=density,
fill=as.factor(top_100))) +
    geom_bar(position = 'dodge', stat='identity') + labs(fill = "top_100", x = colname,
                                                         title = paste(colname, "
relative density grouped by top_100")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  print(plot)
}
```

3.2 Categorical カテゴリー変数

目標値でグループ化された相対密度を示す棒グラフを使用しました（図 3-2）。

•property_type-Home / Apt はトップ 100 内でより人気があります

•room_type-トップ 100 内では、家全体/アパートと個室がシェアルームよりも人気があります。

•ベッドタイプ– トップ 100 に入るにはベッドを提供したほうがよさそうです。

•cancel_policy– ホストはキャンセルのポリシーに関して、柔軟に対応しすぎる必要はなく、適度もしくは厳しい程度でもトップ 100 入りは可能です。

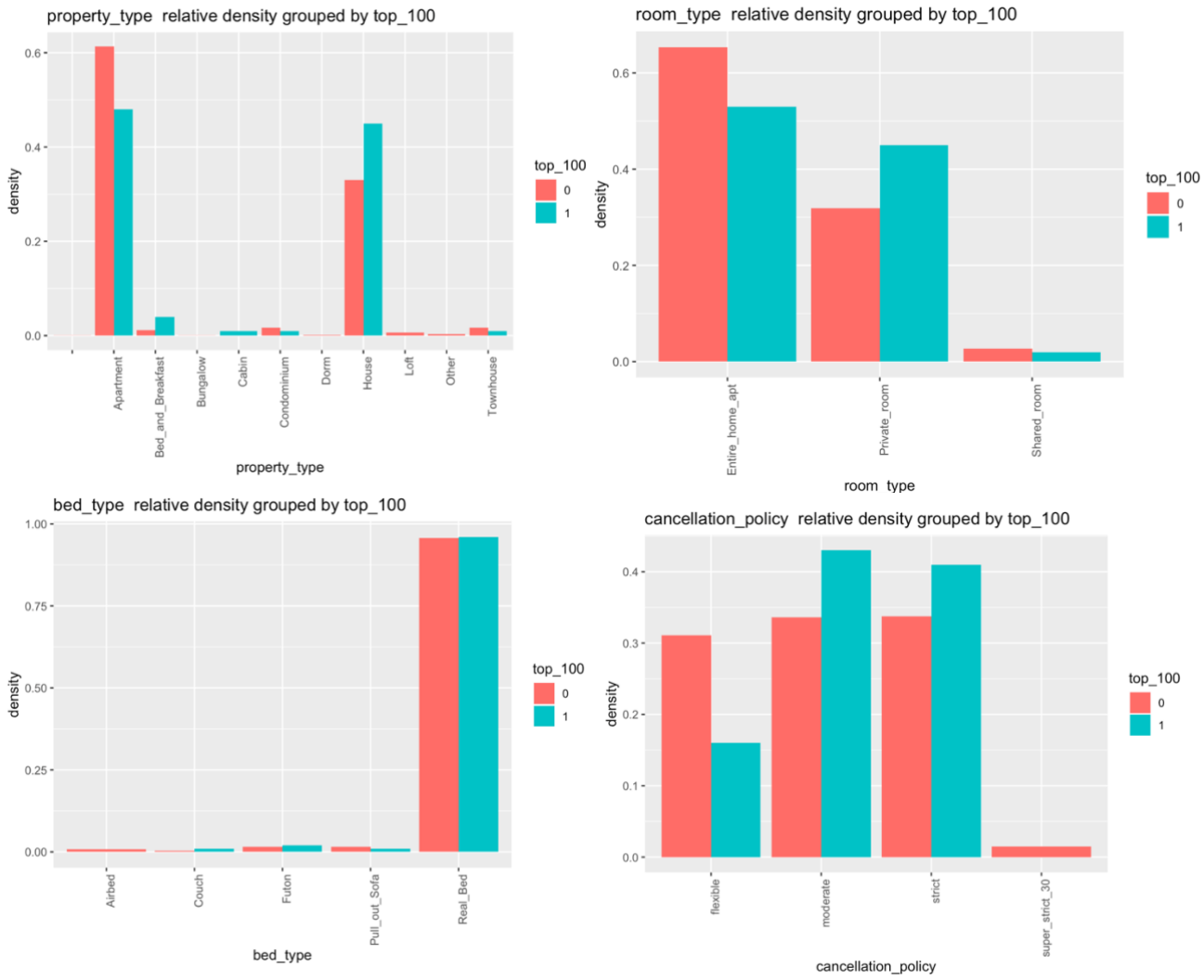

Figure 3.2 The four bar charts in other categorical variables

```
categorical <- c("property_type", "room_type", "bed_type", "cancellation_policy")

for (colname in categorical) {
```

```
  temp <- subset(df, top_100 == 1)
  temp <- temp %>%
    group_by(top_100, temp[,colname]) %>%
    summarise(density = n()/nrow(.))
  colnames(temp)[2] <- colname

  temp1 <- subset(df, top_100 == 0)
  temp1 <- temp1 %>%
    group_by(top_100, temp1[,colname]) %>%
    summarise(density = n()/nrow(.))
  colnames(temp1)[2] <- colname

  temp2 <- rbind(temp, temp1)

  plot <- ggplot(data=temp2, aes(x=temp2[[colname]], y=density,
fill=as.factor(top_100))) +
    geom_bar(position = 'dodge', stat='identity') + labs(fill = "top_100", x = colname,
                                                         title = paste(colname, "
relative density grouped by top_100")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  print(plot)
}
```

3.3 Continuous 連続型データ

連続変数については、箱ひげ図を使用して、各クラス間のデータの分布を理解しました（図 3-3、図 3-4、および図 3-5）。

- バスルーム- トップ 100 入りには必ず最低限 1 つはバスルームが必要です。
- 寝室- トップ 100 入りには必ず最低限 1 つは寝室が必要です。
- ベッド-トップ 100 入りには必ず 1 つ以上のベッドが必要ですが、3 つ以上用意することの効果は小さそうです。
- price_per_person– 価格は 1 人あたり 33 ドルから 50 ドルに設定されている傾向があります。

- extra_people– 上位 100 のリスティングは 10 ドル以上高い価格で罰金を科される傾向があります。言い換えると、高い罰金額を設定してもトップ 100 位のリスティングを維持できると考えられる。
- minimum_nights- 両グループは 1 日または 2 日を設定しており、それによる違いは見られない。3 日以上に設定することはトップ 100 入りするにはさせる方が良いと考えられる。
- calculated_host_listings_count–上位 100 位にリスティングがあるホストは、1〜3 件の他のリストを持つ傾向があります。
- listing_duration-トップ 100 グループのリスト期間ははるかに長く、中央値は約 650 日ですが、それ以外のグループの中央値は約 180 日です。リスティングの期間が長いほど人気に良い影響を与えると考えられます。
- hosting_duration-トップ 100 グループのホスト期間ははるかに長く、中央値は約 1100 日ですが、それ以外グループの中央値は約 650 日です。ホストを行なっている期間が長いほど人気に良い影響を与えると考えられます。
- total_amenities– 両者、約 13〜16 のアメニティが用意されていることが一般的です。上位 100 の中央値は 100 位外のグループよりもわずかに高くなっています。17 個以上と多くのアメニティを用意することは人気への影響に大きな変化は与えなさそうです。しかし、トップ 100 入りには最低 13 個のアメニティを用意する必要があります。
- review_scores– 精度、清潔さ、場所、値が 9〜10 の範囲で同じ分布ですが、チェックインと通信は両方のグループで 10 です。どのリスティングも高い値をキープする必要があります。

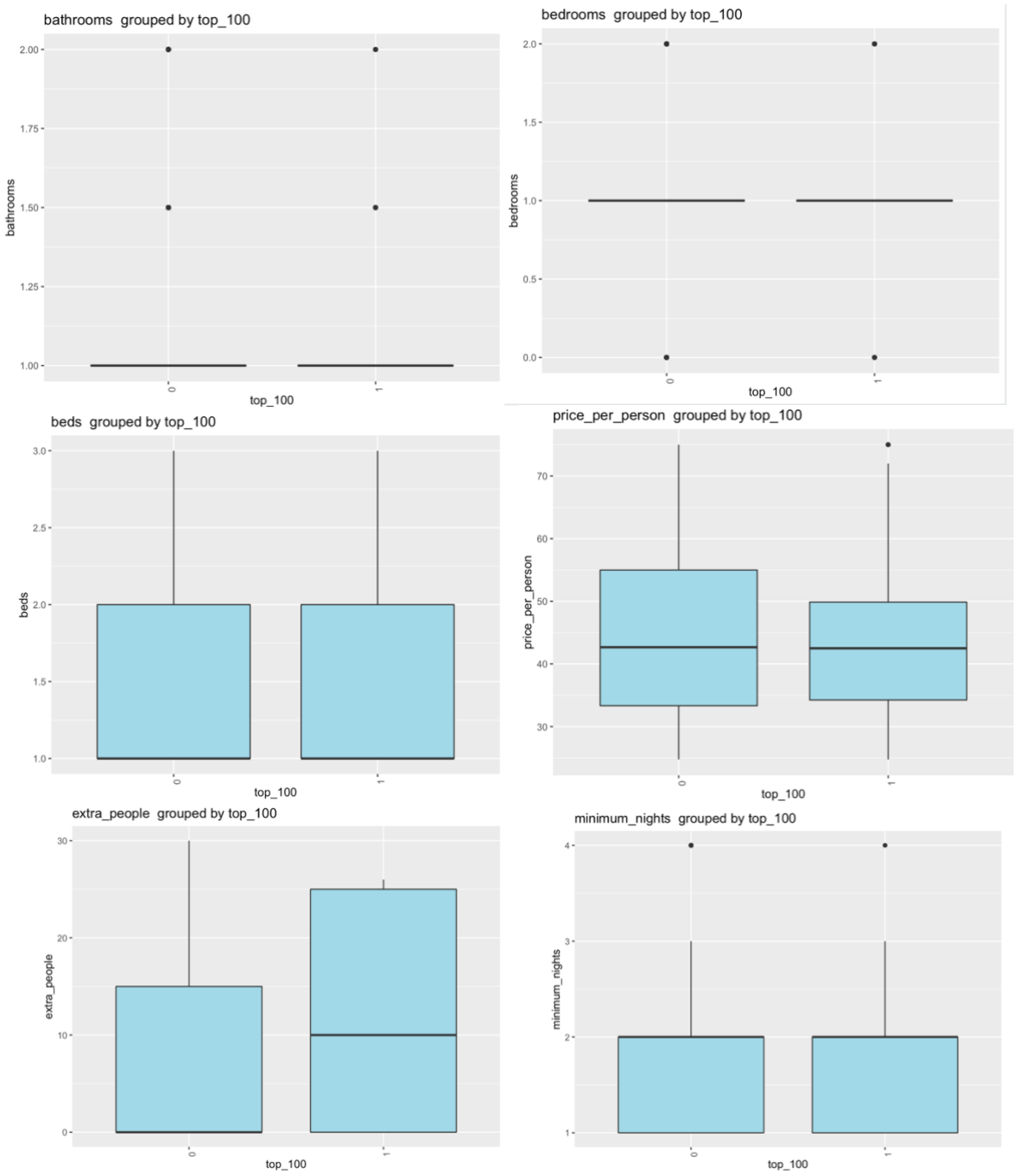

Figure 3-3. The box plot in continuous variables #1

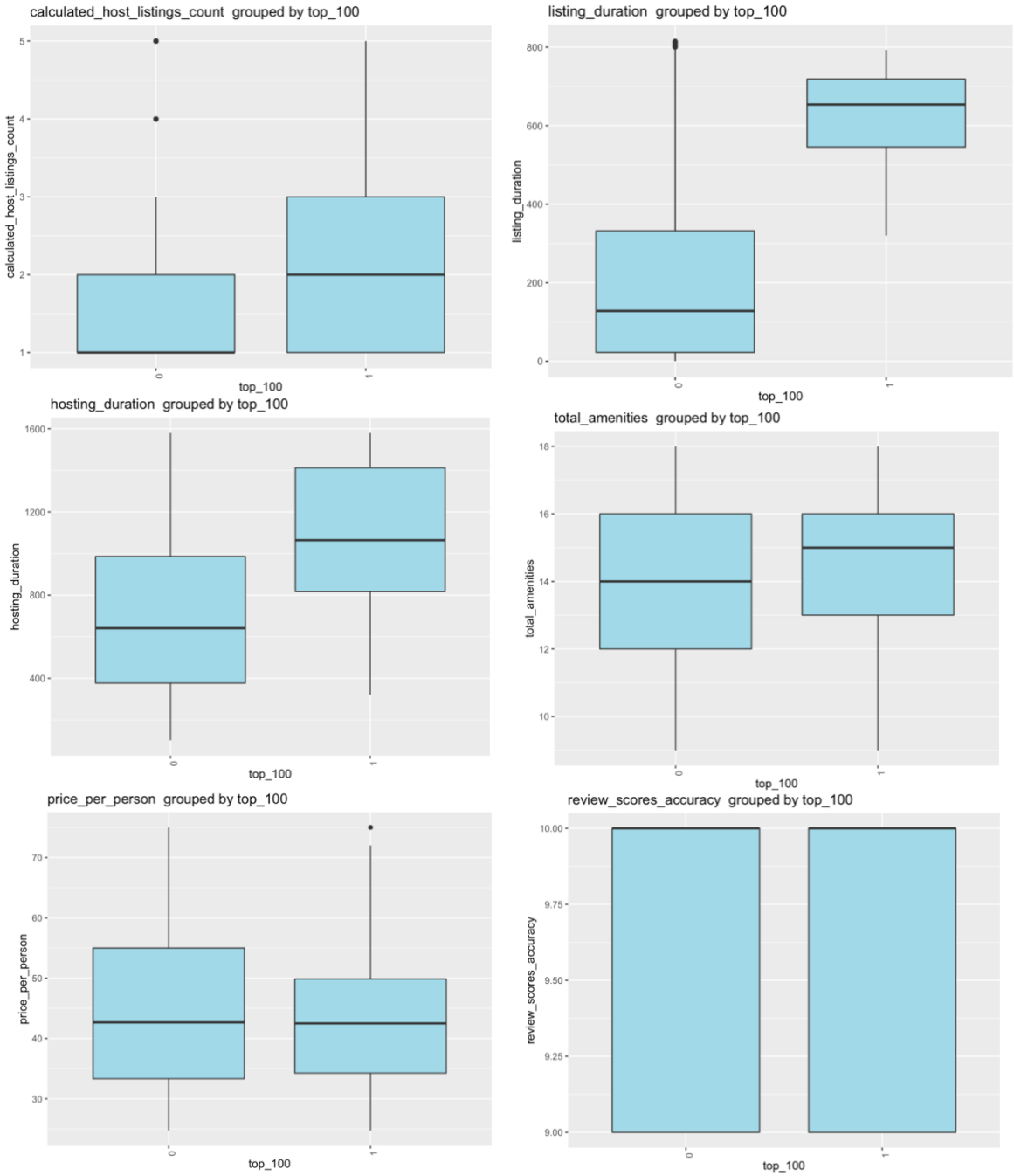

Figure 3-4. The box plot in continuous variables #2

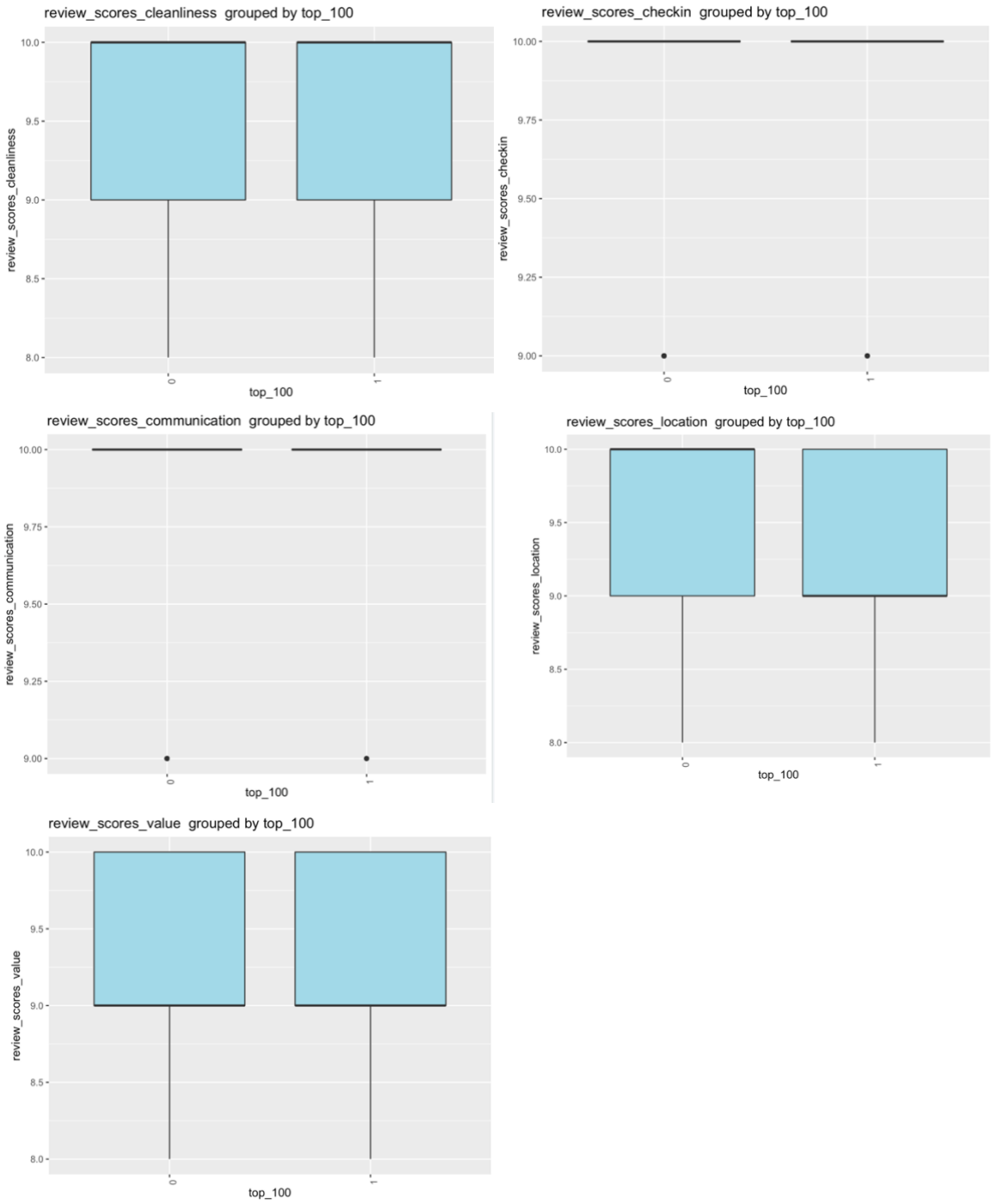

Figure 3-5. The box plot in continuous variables #3

```
continuous <- c("bathrooms", "bedrooms", "beds", "price_per_person",
                "extra_people", "minimum_nights",
                "calculated_host_listings_count", "listing_duration",
                "hosting_duration","total_amenities",
```

```
                    "review_scores_accuracy", "review_scores_cleanliness",
                    "review_scores_checkin", "review_scores_communication",
                    "review_scores_location", "review_scores_value")

for (colname in continuous) {

  plot <- ggplot(data=df, aes(x=as.factor(top_100), y=df[[colname]])) +
    geom_boxplot(fill="lightblue") + labs(x = "top_100", y = colname,
                                          title = paste(colname, " grouped by
top_100")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    scale_y_continuous(limits = quantile(df[[colname]], c(0.1, 0.9)))

  print(plot)
}
```

4.0 実装

さらなる分析の準備として、次のコードを使用してゼロに近い分散変数を特定して削除しました。 この結果、host_has_profile_pic が削除され、28 から 27 列になりました。 さらに、相関の高い変数は分類の結果に影響を与える傾向があるため、モデリングのためにデータセットから 2 つの変数（accommodates と number_of_reviews）を削除することにしました。

```
df_num <- df[, c(5:7, 15:18, 21:24, 27:33, 35:44)]
> dim(df_num)
[1] 2880   28
nzv <- nearZeroVar(select(df_num, -top_100))
df_nzv <- df_num[, -nzv]
> dim(df_nzv)
[1] 2880   27
#check high correlated variables
descrCor <- cor(df_nzv)
highlyCorrelated <- findCorrelation(descrCor, cutoff=0.7)
highlyCorCol <- colnames(df_nzv)[highlyCorrelated]
highlyCorCol
> highlyCorCol
[1] "accommodates"      "number_of_reviews"
```

```
#drop high correlated variables (accommodates, number_of_reviews)
df_nzv_uncor <-
  df_nzv[, -which(colnames(df_nzv) %in% highlyCorCol)]
dim(df_nzv_uncor)
> dim(df_nzv_uncor)
[1] 2880   25
```

4.1 Naïve-Bayes

最初に、データセットを分割して 60/40 でトレーニングとテストのデータを準備し、次に Naive-Bayes 法を使用しました。 その結果、モデルは、正確度が 86.8%、精度が 83.3%でした。 リスティング全体からトップ 100 となる候補を分類する上でよいパフォーマンスをしめしている。しかしながら、モデルのリコールはわずか 17%と低い。これはトップ 100 であるリスティングをトップ 100 であると判断する点においてパフォーマンスが低下していることを示す。

ROC プロット（図 4-1）から、偽陰性率が低い段階で高い真陽性率をしめしています。さらに AUC の領域は大きく、モデルとして良好なパフォーマンスを表していることが判断できます。

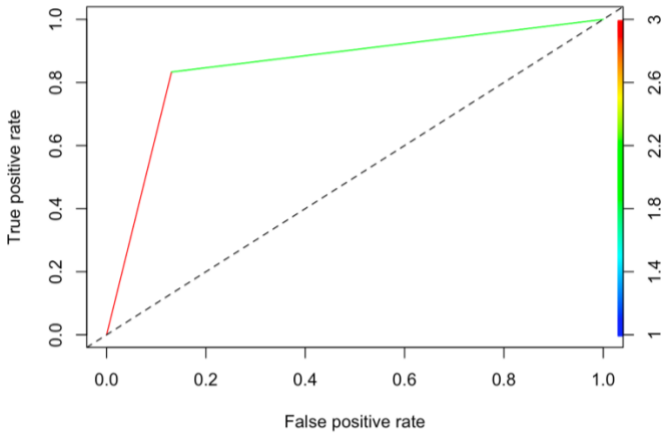

Figure 6-1. ROC Plot in the Naïve-Bayes

```
library(RCurl)
library(e1071)
#install.packages("performance")
#install.packages("prediction")
```

```
library(performance)
library(prediction)

set.seed(132)
nb_sub <- sample(nrow(df_nzv_uncor), floor(nrow(df_nzv_uncor) * 0.6))
nb_train <- df_nzv_uncor[nb_sub, ]
nb_test <- df_nzv_uncor[-nb_sub, ]
nb <- naiveBayes(as.factor(top_100) ~ ., data = nb_train)
nb_prediction <- predict(nb, nb_test)
nb_conf <- table(nb_test$top_100, nb_prediction)
print(nb_conf)
```

```
> print(nb_conf)
   nb_prediction
      0   1
  0 970 146
  1   6  30
```

```
nb_accuracy <- sum(diag(nb_conf))/sum(nb_conf)
print(nb_accuracy)
```

```
> print(nb_accuracy)
[1] 0.8680556
```

```
nb_precision <- nb_conf[2,2] / (nb_conf[2,2] + nb_conf[2,1])
print(nb_precision)
```

```
> print(nb_precision)
[1] 0.8333333
```

```
nb_recall <- nb_conf[2,2] / (nb_conf[2,2] + nb_conf[1,2])
print(nb_recall)
```

```
> print(nb_recall)
[1] 0.1704545
```

```
nb_roc <- performance(prediction(as.numeric(nb_prediction),
as.numeric(nb_test$top_100)), "tpr", "fpr")
plot(nb_roc, colorize=TRUE)
abline(0, 1, lty = 2)
```

4.2 Decision tree

次に、決定木に基づいて、より正確なモデルを構築しようとしました。 まず、パッケージとライブラリを準備し、次にデータをトレーニングデータとテストデータに 60/40 で分割しました。

```
library(rpart)
library(rpart.plot)
# install.packages("rattle")
# install.packages("partykit")
library(rattle)                 # Fancy tree plot
library(RColorBrewer)           # Color selection for fancy tree plot
library(party)                  # Alternative decision tree algorithm
library(partykit)               # Convert rpart object to BinaryTree
```

```
library(formattable)
formattable(df_nzv_uncor)
summary(df_nzv_uncor)
# Proportions of the class values
prop.table(table(df_nzv_uncor$top_100))
```

```
> prop.table(table(df_nzv_uncor$top_100))

         0          1
0.96527778 0.03472222
```

```
table(df_nzv_uncor$top_100)
```

```
> table(df_nzv_uncor$top_100)

   0    1
2780  100
```

```
# Dividing the dataset into training and validation sets.


set.seed(123)
ind <- sample(2, nrow(df_nzv_uncor), replace=TRUE, prob=c(0.7, 0.3))
train_Data <- df_nzv_uncor[ind==1,]
validation_Data <- df_nzv_uncor[ind==2,]
table(train_Data$top_100)
```

```
> table(train_Data$top_100)

   0    1
1953   65
```

table(validation_Data$top_100)

```
> table(validation_Data$top_100)

  0   1
827  35
```

Proportions of the class values

prop.table(table(train_Data$top_100))

```
> prop.table(table(train_Data$top_100))

         0          1
0.96778989 0.03221011
```

次に、以前の探索的データ分析に基づいてツリーを作成しますが、ツリーはアルゴリズムによって最も重要な変数から自動的に選択されます。 これは、このモデルが相関変数が高くなる傾向があることを避けようとしているためです。 さらに、どの変数が以前の作業よりも重要でないかはすでにわかっています。 これより、いくつか変数を削除しツリーを生成します。

```
# Create training and testing sets with selected trees
tree = rpart(top_100~ host_is_superhost +
                    bathrooms + calculated_host_listings_count +
                    hosting_duration +
                    host_local + total_amenities + price_per_person, data=train_Data)
```

生成されたツリー（図 4-2、図 4-3、および図 4-4）の結果として、上位 100 のリストになるためのいくつかの要件があることを理解できます。

- hosting_duration >= 822　2 年強以上のホスト経験の場合
- total_amenities >= 13 & calculated_total_listings > 2 アメニティの数は 13 以上かつホストが持つリスティングが 2 つ以上の場合
- 1 人あたりの価格は $ 23 以上、 $ 45 以下

ただし、ROC 曲線（図 4-6）は、AUC（0.642）を参照して、ナイーブベイズモデルよりもパフォーマンスが低いことを示しています。 これは、決定木モデルが選択したツリーに従って構築されるためです。ツリーを構築するのに重要ではないいくつかの分類子がモデルのパフォーマ

ンスに影響を与える可能性があると思います。 次に、自動生成されたモデルのパフォーマンスを確認しました。

```
print(tree)
prp(tree)
```

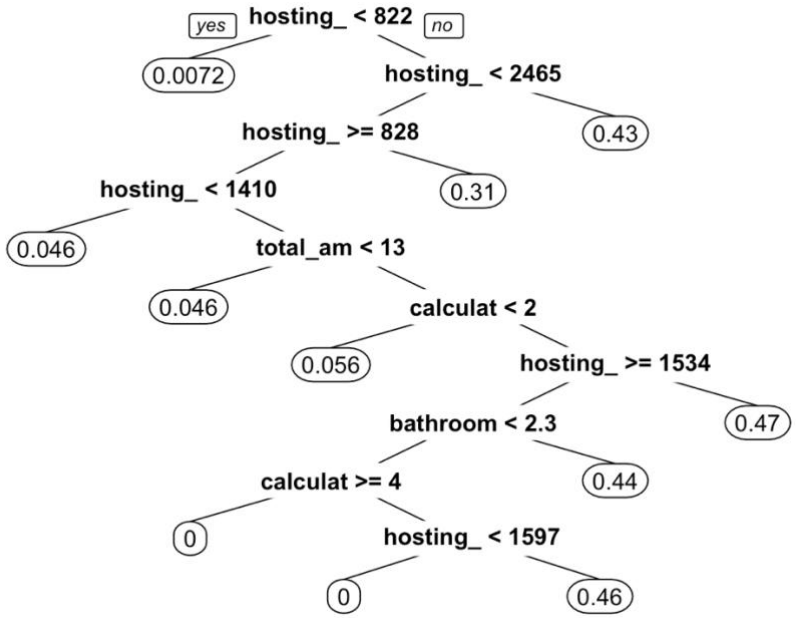

Figure 4-2. Tree for customized model

```
prp (tree, type = 5, extra = 1) #extra = 1 [classification] n=5 [regression]
rpart.plot(tree, extra = 1, nn = TRUE)
```

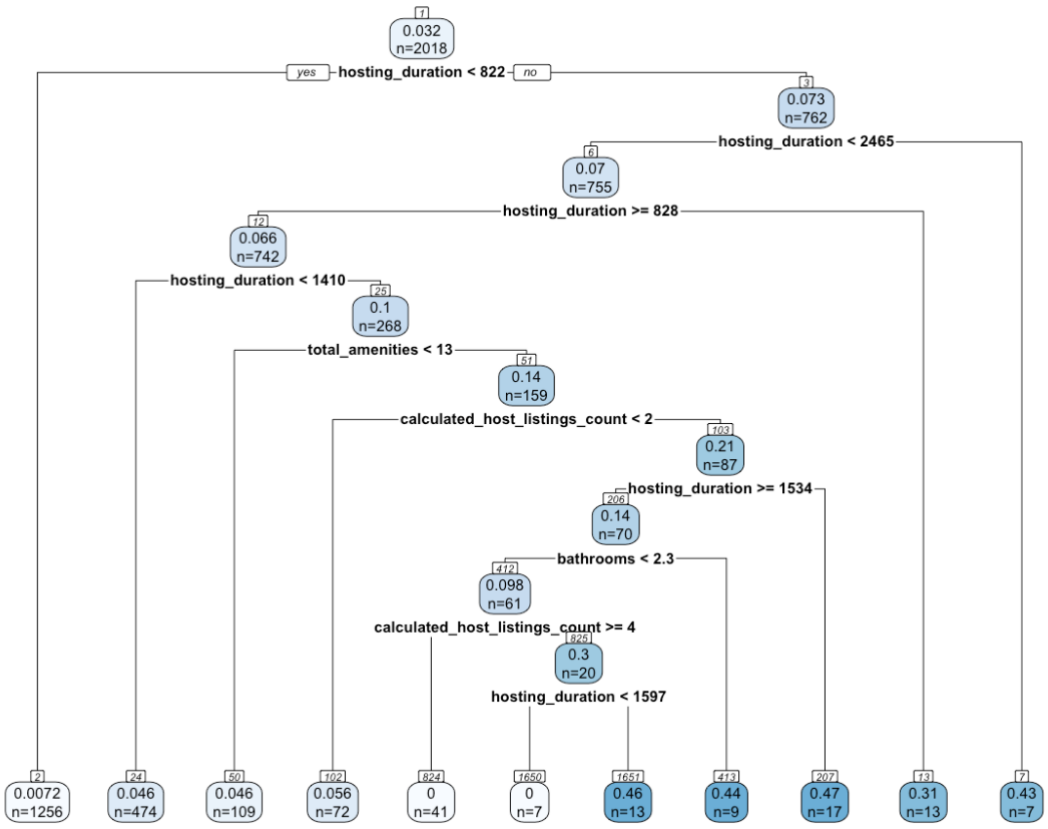

Figure 4-3. Tree Plot for customized model

```
#minsplit how many there are splits, minbucket how many there are
tree_with_params = rpart(top_100~ host_is_superhost +
                         bathrooms + calculated_host_listings_count +
                         hosting_duration +
                         host_local + total_amenities + price_per_person , data=train_Data,
method="class", minsplit = 9, minbucket = 10, cp = -1)
prp(tree_with_params)
```

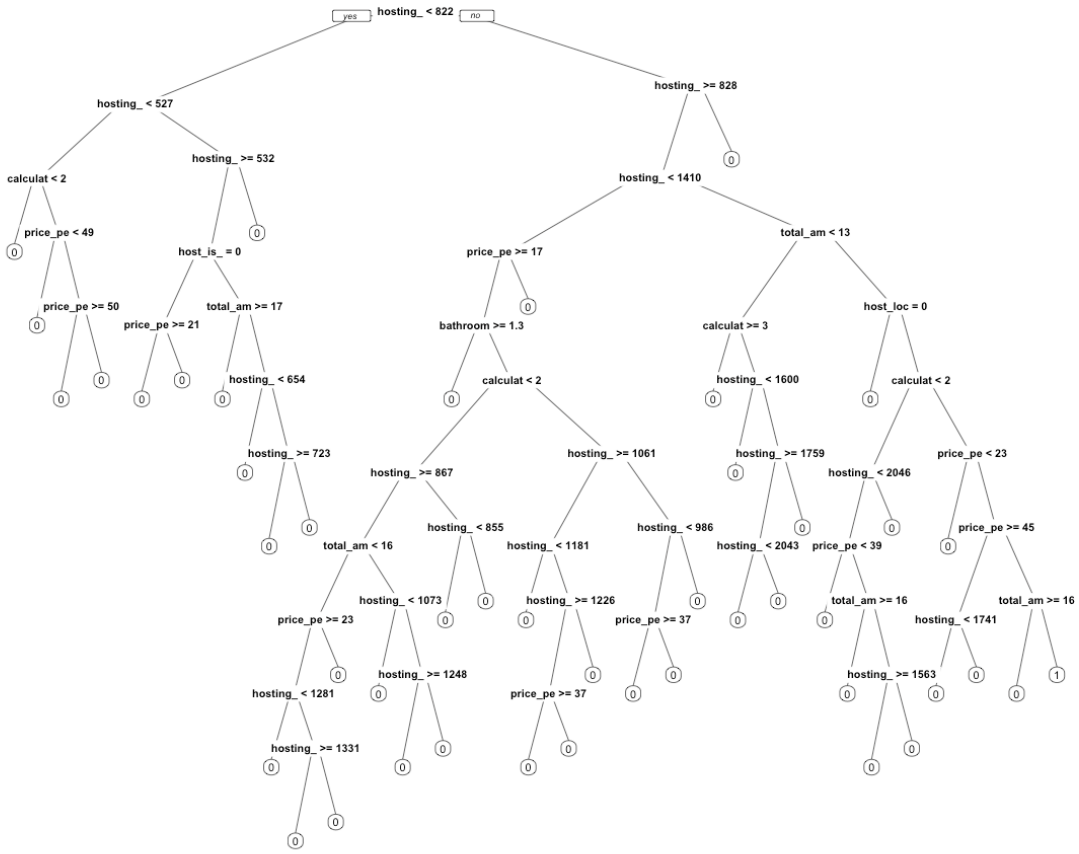

Figure 4-4. Tree with parameters for customized model

print(tree_with_params)
summary(tree_with_params)
plot(tree_with_params)
text(tree_with_params)
plotcp(tree_with_params)

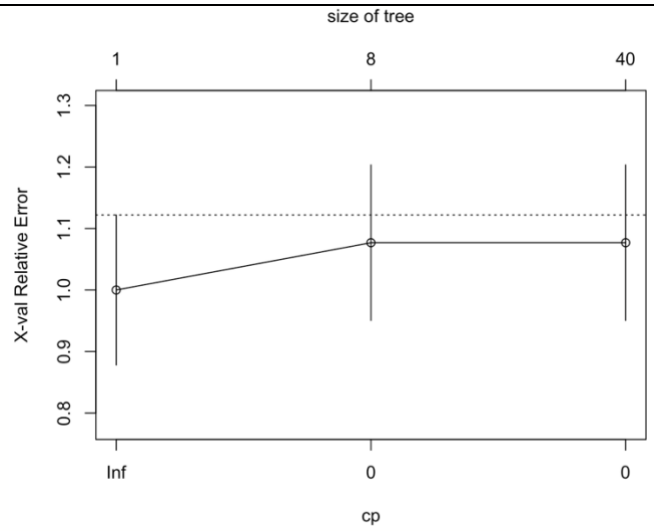

Figure 4.5 CP plot for customized model

```
Predict = predict(tree_with_params, validation_Data)
# Now examine the values of Predict. These are the class probabilities
Predict
# pred <= predict (mymodel, dataset, type = 'prob')
# To produce classes only, without the probabilities, run the next command.
# By default threshold is set at 0.5 to produce the classes
Predict = predict(tree_with_params, validation_Data, type = "class")
Predict

Confusion_matrix = table(Predict, validation_Data$top_100)
print(Confusion_matrix)
```

```
Predict   0   1
      0 820  34
      1   7   1
```

```
# ROC curve
library(ROCR)

# To draw ROC we need to predict the prob values. So we run predict again
# Note that PredictROC is same as Predict with "type = prob"

Predict_ROC = predict(tree_with_params, validation_Data)
```

```
Predict_ROC
Predict_ROC[,2]


pred = prediction(Predict_ROC[,2], validation_Data$top_100)
perf = performance(pred, "tpr", "fpr")
pred
perf
plot(perf, colorize = T)
plot(perf, colorize=T,
     main = "ROC curve",
     ylab = "Sensitivity",
     xlab = "Specificity",
     print.cutoffs.at=seq(0,1,0.3),
     text.adj= c(-0.2,1.7))


# Area Under Curve
auc = as.numeric(performance(pred, "auc")@y.values)
auc = round(auc, 3)
auc
```

```
> auc
[1] 0.642
```

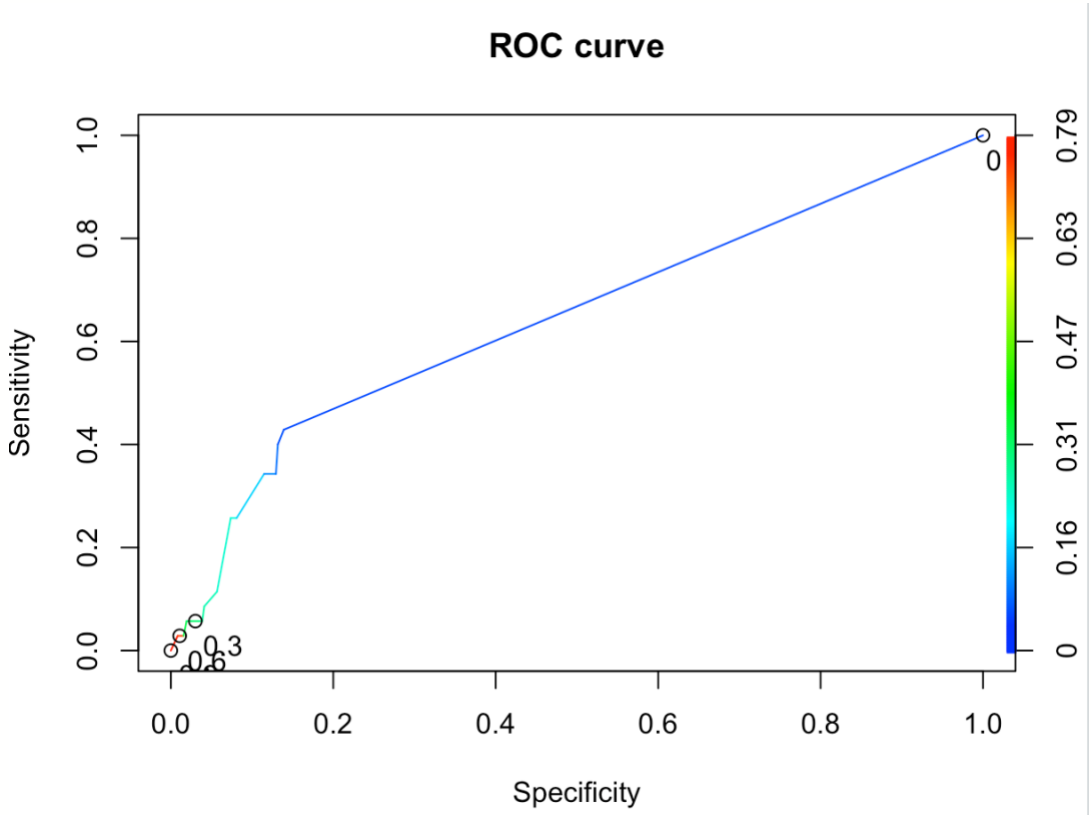

Figure 4-6. ROC curve for customized model

新しく生成されたツリー（図 4-7）から、分類子の数が減少し、ツリー内で 2 つの変数が支配的であることがわかります。 ただし、正確度は約 98.7%、精度はかなり 66.7%です。 前者はナイーブベイズモデルと比較して改善されましたが、後者は低下しました。 AUC は 88.9%に大幅に改善され、ROC プロット（図 4-8）はかなり高いパフォーマンスを表しているようです。このモデルはポジティブにデータが偏っていることを示します。全体の正確度をあげようとしたため、分類気によるネガティブが軽視されているとわかります。

```
set.seed(132)
df_nzv1 <- df_nzv_uncor
dt_sub <- sample(nrow(df_nzv1), floor(nrow(df_nzv1) * 0.6))
dt_train <- df_nzv1[dt_sub, ]
dt_test <- df_nzv1[-dt_sub, ]

dt_model <- rpart(top_100~ ., data = dt_train, method = "class", control = rpart.control(cp = 0.01, minbucket = 5))
fancyRpartPlot(dt_model, caption = "")
```

```
dt_prediction <- predict(dt_model, dt_test, type = "class")
dt_pred <- prediction(predict(dt_model, type = "prob")[, 2], dt_train$top_100)

dt_conf <- table(dt_test$top_100, dt_prediction)

print(dt_conf)
printcp(dt_model)
plotcp(dt_model)
dt_prediction <- predict(dt_model, dt_test, type = "class")
dt_pred <- prediction(predict(dt_model, type = "prob")[, 2], dt_train$top_100)

dt_conf <- table(dt_test$top_100, dt_prediction)

dt_accuracy <- sum(diag(dt_conf))/sum(dt_conf)
print(dt_accuracy)
> print(dt_accuracy)
[1] 0.9869792

dt_precision <- dt_conf[2,2] / (dt_conf[2,2] + dt_conf[2,1])
print(dt_precision)
> print(dt_precision)
[1] 0.6666667

dt_recall <- dt_conf[2,2] / (dt_conf[2,2] + dt_conf[1,2])
print(dt_recall)
> print(dt_recall)
[1] 0.8888889

dt_roc <- performance(dt_pred, measure="tpr", x.measure="fpr")
plot(dt_roc, colorize=TRUE)
abline(0, 1, lty = 2)
```

```
tree_with_params = rpart(top_100~ ., data=train_Data, method="class", minsplit = 1,
minbucket = 7, cp = -1)
prp(tree_with_params)
print(tree_with_params)
summary(tree_with_params)
plot(tree_with_params)
text(tree_with_params)
plotcp(tree_with_params)
```

```
Predict = predict(tree_with_params, validation_Data)
# Now examine the values of Predict. These are the class probabilities
Predict
# pred <= predict (mymodel, dataset, type = 'prob')
# To produce classes only, without the probabilities, run the next command.
# By default threshold is set at 0.5 to produce the classes
Predict = predict(tree_with_params, validation_Data, type = "class")
Predict

Confusion_matrix = table(Predict, validation_Data$top_100)
print(Confusion_matrix)

# ROC curve
library(ROCR)

# To draw ROC we need to predict the prob values. So we run predict again
# Note that PredictROC is same as Predict with "type = prob"

Predict_ROC = predict(tree_with_params, validation_Data)
Predict_ROC
Predict_ROC[,2]

pred = prediction(Predict_ROC[,2], validation_Data$top_100)
perf = performance(pred, "tpr", "fpr")
pred
perf
plot(perf, colorize = T)
plot(perf, colorize=T,
     main = "ROC curve",
     ylab = "Sensitivity",
     xlab = "Specificity",
     print.cutoffs.at=seq(0,1,0.3),
     text.adj= c(-0.2,1.7))
# Area Under Curve
auc = as.numeric(performance(pred, "auc")@y.values)
auc = round(auc, 3)
auc
```

```
> auc
[1] 0.94
```

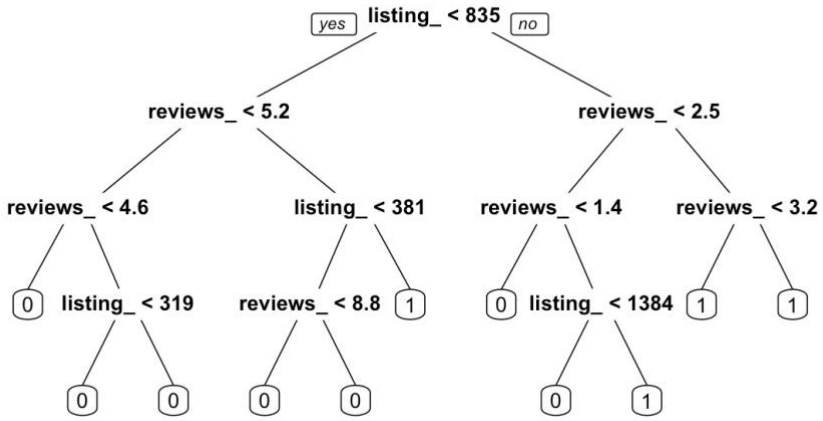

Figure 6-7. Tree for auto generated model

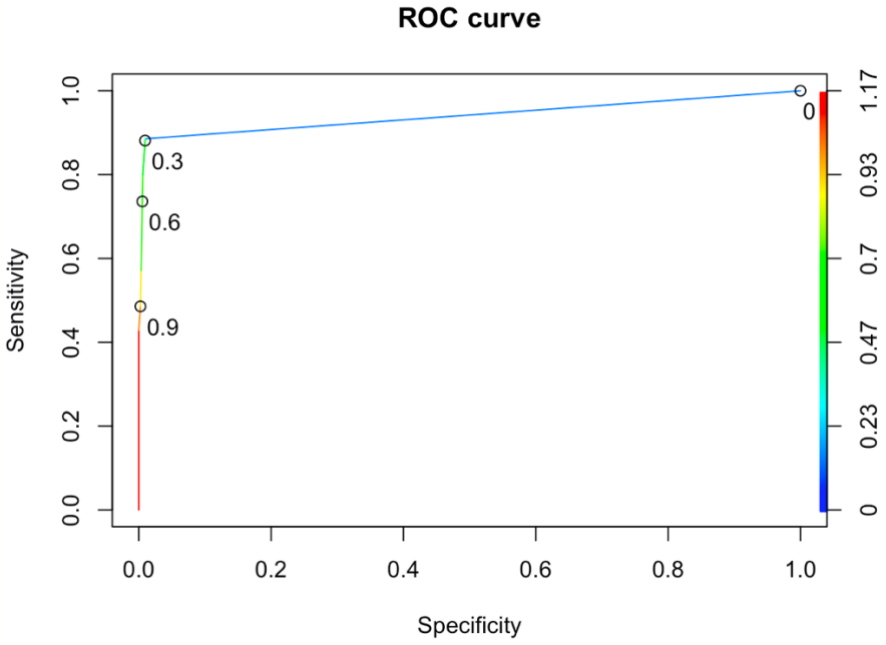

Figure 6-8. ROC curve for auto generated model

5.0 結論

EDA による調査結果は、いくつかの変数は人気を高め、トップ 100 になるための重要な要因のようです。

- スーパーホストであることは大きくトップ 100 に入るには影響しない
- 上位 100 のリストを持つホストは、ゲストのプロフィール写真と電話番号を求めない場合が多いです
- 必ずホストはプロフィール画像が必要です
- ホストは必ず Airbnb から身元確認をされているべきです
- 直前の予約に関して寛容になりすぎる必要はありません
- ホストはローカルであると人気を獲得しやすい傾向にあります

- リスティングタイプは一軒家もしくはアパートメントが大変人気であり、一軒家またはアパートメント一部屋をシェアすることなく利用できることがゲストにとって人気である
- ホストはキャンセル対応に柔軟に対応する必要はなく、適度または厳しいルールを提示することも可能

- ベッド、バスルーム、寝室は最低 1 つ提供すべきです。ベッドは 3 つ以上用意する必要はありません。
- 価格は 1 人あたり 33 ドルから 50 ドルに設定されていると無難です。
- ホストは追加の人数に対し、25 $ までの罰金を科すことができます。
- 最低宿泊日数は 1 もしくは 2 泊に設定します。3 泊に設定されているリスティングは人気がありません
- ホストは別の物件を 1~3 件持っている傾向があります
- 物件のリスティング期間、ホストのホスティング期間が長ければ長いほど、人気に影響を与えます
- アメニティは最低 13 個用意する必要があります。17 個以上は用意しても人気に大きな影響は与えません
- レビュースコアは 10 点満点中 9 点以上を維持することが重要になってきます

次に、予測するナイーブベイズとディシジョンツリーに基づく 2 つのモデルは、高いパフォーマンスを示し、人気に影響を与える特徴を表しました。 まず、ナイーブベイズモデルの精度は 87%で、ROC は高性能ですが、リコールは 17%と低すぎます。 この場合、このモデルは要因を識別するためのリストの分類子にすぎないため、再現率をあまり気にする必要はありません。

決定木モデルの分類精度は高いが、ツリーから特定のパターンを判断することは難しかった。これは一部の連続型データの依存が一つの原因である。依存関係を回避するため、変数を選択し構築されたモデルは、自動生成されたモデルよりもパフォーマンスが低くなりました。 パフ

ォーマンスの低下という結果を除けば、人気に影響を与える特徴を一部理解することできました。

- 2 年強以上のホスト経験の場合
- アメニティの数は 13 以上かつホストが持つリスティングが 2 つ以上の場合
- 1 人あたりの価格は $ 23 以上、 $ 45 以下

EDA の実施および 2 つのモデルを開発したことにより、リスティングがトップ 100 入りする特徴の分析結果を得ることができた。さらに決定木モデルは人気を得られるパターンを表し、分類モデルとしも高精度な結果をもたらした。

Bibliography

Airbnb (2017) About Airbnb. [Online] Available from: https://news.airbnb.com/about-us/ [Accessed: 4th February 2020].

Bajari, P., Nekipelov, D., Ryan, SP and Yang, M. (2015) Demand estimation with machine learning and model combination. National Bureau of Economic Research. [Online] Available from: https://www.nber.org/papers/w20955 [Accessed: 4th February 2020].

Choudhary, P., Jain, A. and Baijal, R. (2018) Unravelling Airbnb Predicting Price for New Listing. [Online] Available from: https://arxiv.org/pdf/1805.12101.pdf [Accessed: 4th February 2020].

Deboosere, R., Kerrigan, J.D., Wachsmuth, D. and El-Geneidy, A. (2019) Location, location and professionalization: a multilevel hedonic analysis of Airbnb listing prices and revenue. [Online] Available from: https://www.tandfonline.com/doi/pdf/10.1080/21681376.2019.1592699?needAccess=true [Accessed: 4th February 2020].

Keating, J., Katnic, E., Hahn, C. and Yang, R. (2018) PREDICTIVE MODELING ON AIRBNB LISTING PRICES [Online] Available from: https://joshuakeating.com/resources/files/airbnb-paper.pdf [Accessed: 4th February 2020].

Lawani, A., Michael, R., Mark, T. and Zheng, Y. (2017) Impact of reviews on price: Evidence from sentiment analysis of Airbnb reviews in Boston [Online] Available from: http://people.wku.edu/alex.lebedinsky/KEA_papers/LAWANI.pdf [Accessed: 4th February 2020].

Li, Y., Wang, S., Yang, T., Pan, Q. and Tang, J. (2017) Price Recommendation on Vacation Rental Websites. [Online] Available from: https://epubs.siam.org/doi/pdf/10.1137/1.9781611974973.45 [Accessed: 4th February 2020].

Luo, Y., Zhou, X. and Zhou, Y. (2019) Predicting Airbnb Listing Price Across Different Cities [Online] Available from: http://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26647491.pdf [Accessed: 4th February 2020].

Moreno-Izquierdo, L., Egorova, G., Pereto-Rovira, A. and Mas-Ferrando, A (2018) Exploring the use of artificial intelligence in price maximisation in the tourism sector: its application in the case of Airbnb in the Valencian Community. [Online] Available from: https://ebuah.uah.es/dspace/bitstream/handle/10017/37170/exploring_moreno_IR_2018.pdf?sequence=1&isAllowed=y [Accessed: 4th February 2020].

Szotek, M. (2018) Understandig Data - Airbnb listing popularity analysis based on Barcelona data. [Online] Available from: https://rstudio-pubs-static.s3.amazonaws.com/407929_afc5ef0f2ad648389447a6ca3f4a7cd4.html [Accessed: 4th April 2020]

Tang, E. and Sangani, K. (2015) Neighborhood and Price Prediction for San Francisco Airbnb Listings. [Online] Available from: https://www.semanticscholar.org/paper/Neighborhood-and-Price-Prediction-for-San-Francisco-Tang/c50a1c28dbe7a886148e8f983fb069d4b1439dc6 [Accessed: 4th February 2020].

VanderPlas, J. (2018) Python Data Science Handbook: Essential For Working With Data. O'REILLY

Ye, P., Qian, J., Chen, J., Wu, CH., Zhou, Y., De Mars, S., Yang, F. and Zhang, L. (2018) Customized Regression Model for Airbnb Dynamic Pricing. [Online] Available from: https://dl.acm.org/doi/abs/10.1145/3219819.3219830 [Accessed: 4th February 2020].

Zhang, Z., Chen, RJ.C., Han, L.D. and Yang, L (2017) Key Factors Affecting the Price of Airbnb Listings: A Geographically Weighted Approach. [Online] Available from: https://www.mdpi.com/2071-1050/9/9/1635 [Accessed: 4th February 202