Spaceships

Hanzholah Shobri

2023-11-22

In the ever-evolving field of data science, proficiency in a diverse array of tools is invaluable. My journey has been primarily anchored in the use of R for data analysis, harnessing the power of `tidyverse` packages to parse through datasets, extract meaningful insights, and generate beautiful visualisations. In parallel, when the task has shifted towards machine learning and deep learning, I have found comfort in the familiar territories of Python, utilizing well-established libraries like `scikit-learn` and `TensorFlow` to train models for tackling different problems.

However, a comprehensive skill set in data science demands adaptability and a willingness to learn. Recognizing the importance of this versatility, I am now turning my interest to learning the R's `tidymodels` framework. This suite promises a unified and systematic approach to modeling that could parallel the functionality of `scikit-learn`.

The functionalities come from `tidymodels` that I have learned is going to be applied to the Getting Started Competition: Spaceship Titanic hosted on Kaggle. This competition, tailored for those beginners, offers the perfect backdrop for applying `tidymodels`. Through this project, I aim to enrich my capabilities in R and evaluate `tidymodels` as a viable alternative for future projects, thereby enhancing my analytical repertoire.

Setup Environment and Prepare the Data

To begin our analysis, let's load several crucial R libraries. These libraries provide functions and tools that will be utilized throughout our data analysis process. Additionally, we set a minimal theme for any subsequent visualizations and define the base path to our data directory.

```
library(discrim)
library(doParallel)
library(tidyverse)
library(tidymodels)
```

```
library(patchwork)

theme_set(theme_minimal() +
            theme(plot.title.position = "plot"))

path_to_data <- here::here("data/")
```

Next, import the datasets downloaded from the Kaggle site, consisting of train and test datasets within the same directory. For the test data, we need to add a new column named 'Transported' and initialize it with NA values, representing unknown values to be predicted after this.

```
train_data <-
  fs::dir_ls(path_to_data, glob = "*/train.csv") |>
  read_csv()

test_data <-
  fs::dir_ls(path_to_data, glob = "*/test.csv") |>
  read_csv() |>
  mutate(Transported = NA)
```

To ensure our data is structured properly for analysis, we apply a cleaning function to the spaceship data. This function takes care of several operations:

- Separating the 'Cabin' column into multiple columns for 'Deck', 'Number', and 'Side'.
- Extracting a 'TravelGroup' based on the 'PassengerId'.
- Calculating the 'GroupSize' by counting members in each travel group.
- Converting logical columns to numeric.
- Factoring several columns, including 'HomePlanet', those that start with 'Cabin', and others like 'Destination', 'Transported', 'GroupSize', 'CryoSleep', and 'VIP'.
- Reordering columns to have 'PassengerId' and 'GroupSize' first, while also removing the temporary 'TravelGroup' column.

After applying the cleaning function to both training and test data, we briefly glimpse into the structure of the cleaned training data to verify the changes.

```
clean_spaceship <-
  function(spaceship_data) {
    spaceship_data |>
      separate(Cabin, into = paste0("Cabin", c("Deck", "Num", "Side"))) |>
      mutate(TravelGroup = str_extract(PassengerId, "^[0-9]*"),
             Transported = if_else(Transported, "Yes", "No")) |>
```

```
      mutate(GroupSize = n(), .by = TravelGroup) |>
      mutate_if(is.logical, as.numeric) |>
      mutate(across(c(HomePlanet, starts_with("Cabin"), Destination,
                      Transported, GroupSize, CryoSleep, VIP), factor)) |>
      select(PassengerId, GroupSize, everything()) |>
      select(-TravelGroup)
  }

train_data <- clean_spaceship(train_data)
test_data  <- clean_spaceship(test_data)
```

Exploratory Analysis

After setting up project and cleaning data, we will conduct an initial review of the dataset by leveraging the `glimpse` function. This allows us to see the data types of each column. A more extensive exploration per column will follow to uncover deeper insights.

```
glimpse(train_data)
```

```
Rows: 8,693
Columns: 17
$ PassengerId  <chr> "0001_01", "0002_01", "0003_01", "0003_02", "0004_01", "0~
$ GroupSize    <fct> 1, 1, 2, 2, 1, 1, 2, 2, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, ~
$ HomePlanet   <fct> Europa, Earth, Europa, Europa, Earth, Earth, Earth, Earth~
$ CryoSleep    <fct> 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, ~
$ CabinDeck    <fct> B, F, A, A, F, F, F, G, F, B, B, B, F, G, F, NA, F, F, F,~
$ CabinNum     <fct> 0, 0, 0, 0, 1, 0, 2, 0, 3, 1, 1, 1, 1, 1, 2, NA, 3, 4, 5,~
$ CabinSide    <fct> P, S, S, S, S, P, S, S, S, P, P, P, P, S, P, NA, P, P, P,~
$ Destination  <fct> TRAPPIST-1e, TRAPPIST-1e, TRAPPIST-1e, TRAPPIST-1e, TRAPP~
$ Age          <dbl> 39, 24, 58, 33, 16, 44, 26, 28, 35, 14, 34, 45, 32, 48, 2~
$ VIP          <fct> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
$ RoomService  <dbl> 0, 109, 43, 0, 303, 0, 42, 0, 0, 0, 0, 39, 73, 719, 8, 32~
$ FoodCourt    <dbl> 0, 9, 3576, 1283, 70, 483, 1539, 0, 785, 0, 0, 7295, 0, 1~
$ ShoppingMall <dbl> 0, 25, 0, 371, 151, 0, 3, 0, 17, 0, NA, 589, 1123, 65, 12~
$ Spa          <dbl> 0, 549, 6715, 3329, 565, 291, 0, 0, 216, 0, 0, 110, 0, 0,~
$ VRDeck       <dbl> 0, 44, 49, 193, 2, 0, 0, NA, 0, 0, 0, 124, 113, 24, 7, 0,~
$ Name         <chr> "Maham Ofracculy", "Juanna Vines", "Altark Susent", "Sola~
$ Transported  <fct> No, Yes, No, No, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, ~
```

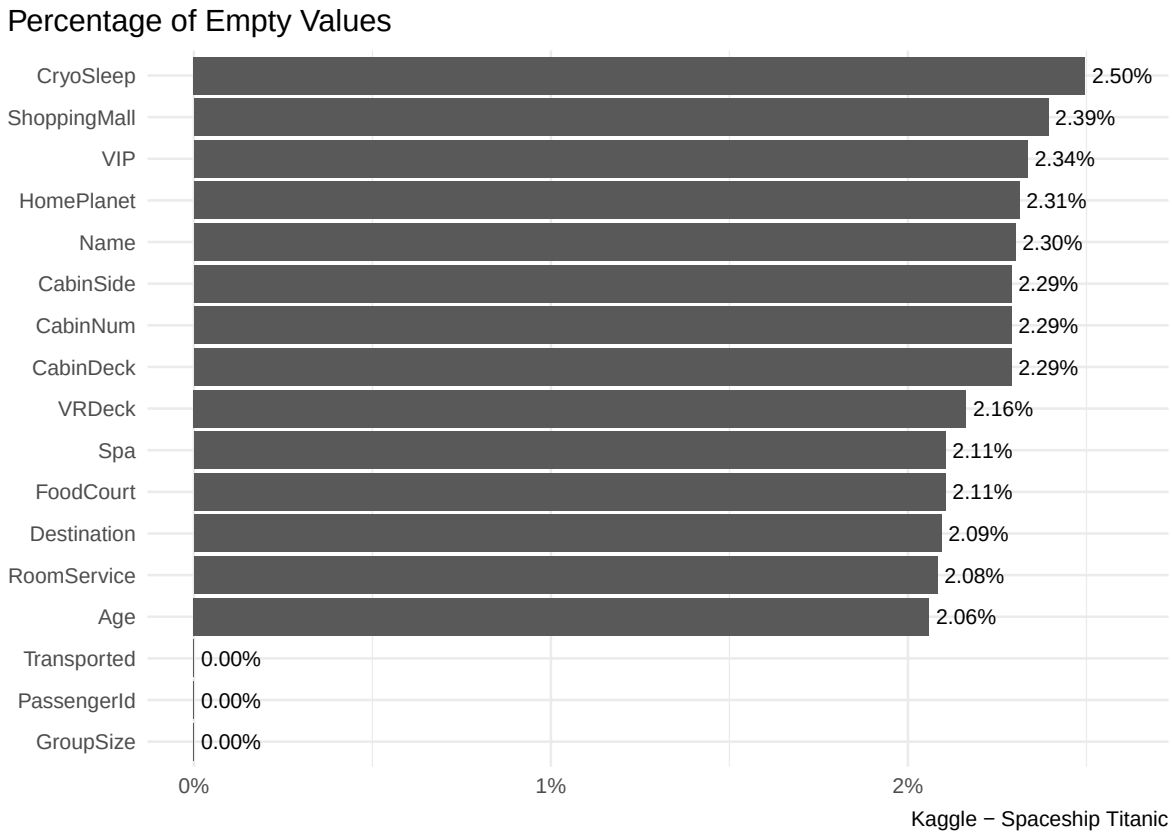
The primary aim of this project is to build a model that can predict if a passenger was transported to another dimension. Based on the competition's description, approximately half of the passengers were transported. The data presented in the table below supported this notion.

```
train_data |>
  count(Transported) |>
  mutate(prop = n / sum(n)) |>
  mutate(prop = scales::percent(prop, accuracy = 0.001))
```

```
# A tibble: 2 x 3
  Transported     n prop
  <fct>       <int> <chr>
1 No           4315 49.638%
2 Yes          4378 50.362%
```

The plot below indicates that a majority of the variables exhibit a relatively low percentage of missing values, ranging between 2 and 2.5 percent. Variables with no empty values are `Transported`, `PassengerId`, and `GroupSize` , which can be expected as those are the Identifier and target values.

```
bind_cols(
  col_name = names(train_data),
  map_dfr(train_data, function(x) {
    list(n = sum(is.na(x)), prop = sum(is.na(x)) / length(x))
  })) |>
  ggplot(aes(x = prop, y = fct_reorder(col_name, n))) +
  geom_col() +
  geom_text(aes(label = scales::percent(prop, accuracy = 0.01), x = prop + 0.0002),
            hjust = 0, size = 3) +
  expand_limits(x = c(0, 0.026)) +
  labs(title = "Percentage of Empty Values",
       caption = "Kaggle - Spaceship Titanic",
       x = NULL, y = NULL) +
  scale_x_continuous(labels = scales::percent)
```



Numeric Variables

Let's observe each of numerical columns.

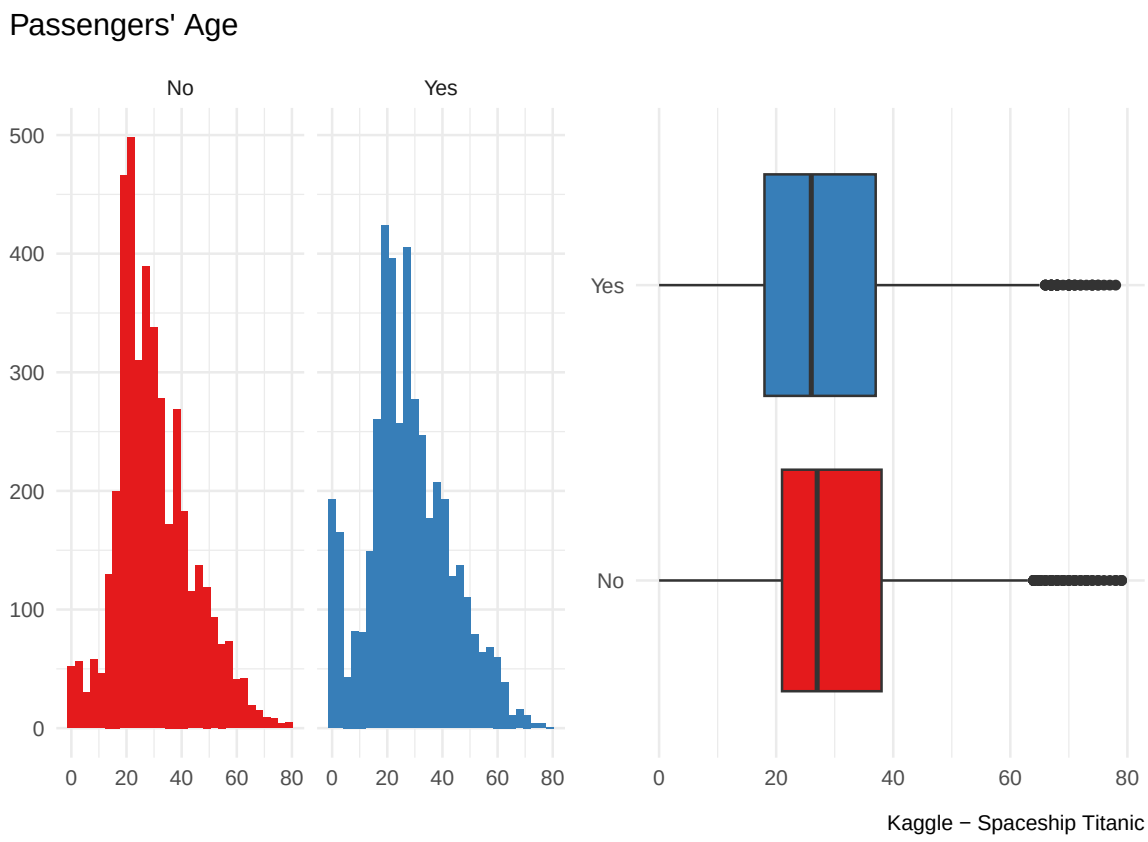
Age

The first column is about the age of passengers. The age distribution of the Spaceship Titanic's passengers reveals that approximately half are between 20 to 40years old. When comparing those who were transported to an alternate dimension with those who weren't, we see a tendency for the transported group to skew younger.

```
p_age1 <-
  train_data |>
    ggplot(aes(x = Age, fill = Transported)) +
    geom_histogram() +
    facet_wrap(. ~ Transported, nrow = 1)
```

```
p_age2 <-
  train_data |>
    ggplot(aes(x = Age, y = Transported, fill = Transported)) +
    geom_boxplot()

(p_age1 + p_age2) +
  plot_annotation(title = "Passengers' Age",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```



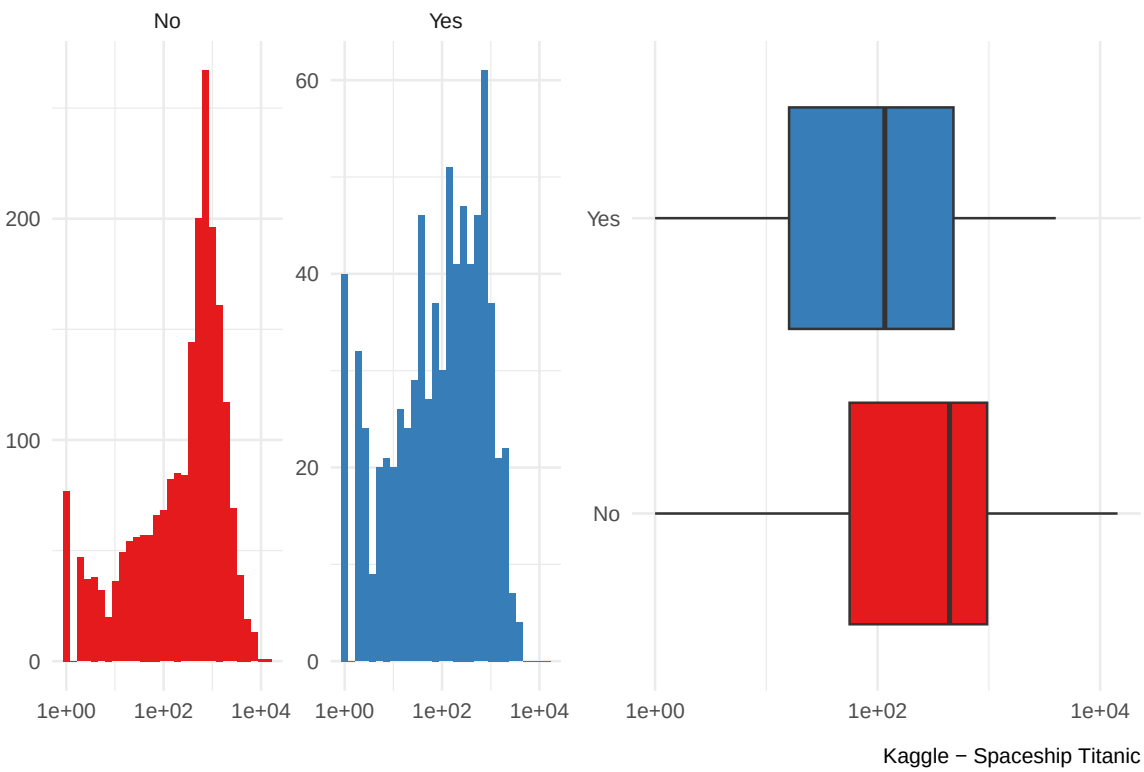
Room Service

Moving on to the room service, it can be seen that there is a skewness in the room service charges, with a few passengers incurring significantly higher expenses. To improve analytical processes, a logarithmic transformation for this data could be applied.

```r
p_room1 <-
  train_data |>
    ggplot(aes(x = RoomService, fill = Transported)) +
    geom_histogram() +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    facet_wrap(. ~ Transported, scales = "free_y")

p_room2 <-
  train_data |>
    ggplot(aes(x = RoomService, y = Transported, fill = Transported)) +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    geom_boxplot()

(p_room1 + p_room2) +
  plot_annotation(title = "Room Service Charges",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```



Food Court

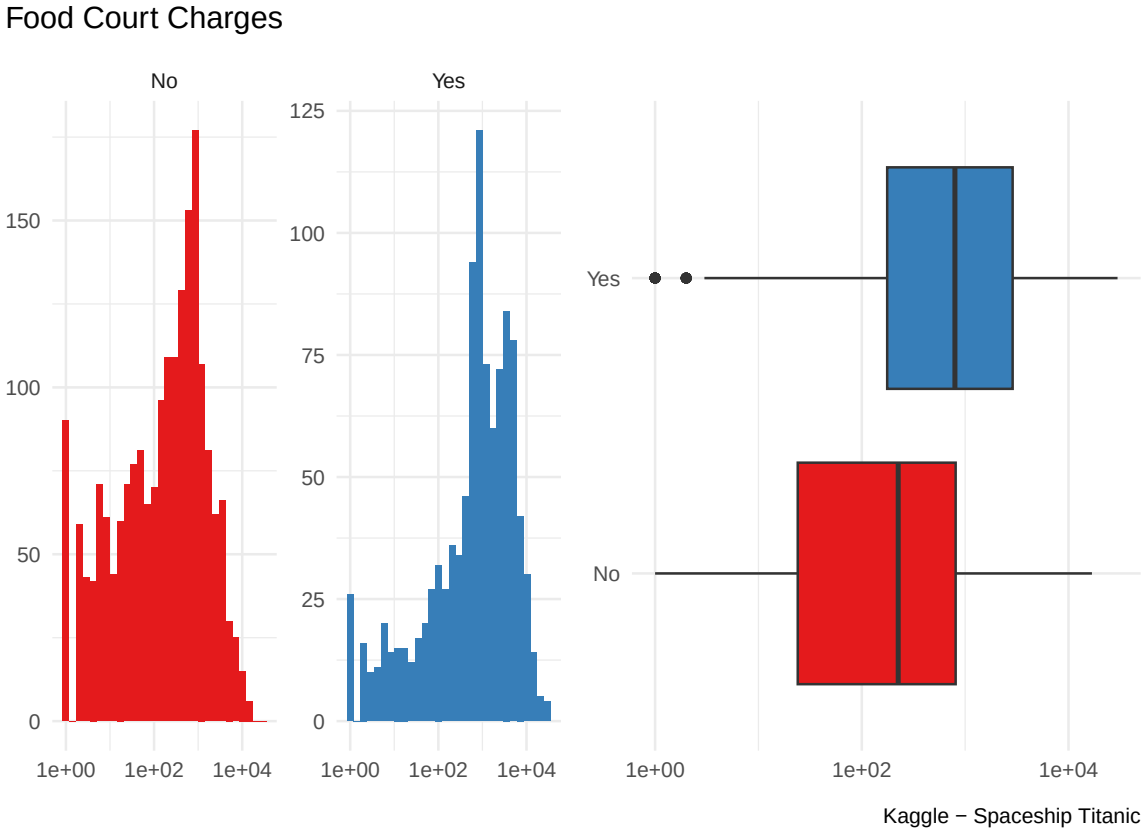
This pattern of skewness extends to the amounts spent at the food court, where a small subset of passengers spent substantially more than their peers. Logarithmic transformation is again recommended here. Preliminary analysis of this transformed data suggests that transported passengers might have higher food court expenses, indicating a potential trend worth investigating further.

```
p_food1 <-
  train_data |>
    ggplot(aes(x = FoodCourt, fill = Transported)) +
    geom_histogram() +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    facet_wrap(. ~ Transported, scales = "free_y")

p_food2 <-
  train_data |>
```

```
    ggplot(aes(x = FoodCourt, y = Transported, fill = Transported)) +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    geom_boxplot()

(p_food1 + p_food2) +
  plot_annotation(title = "Food Court Charges",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```



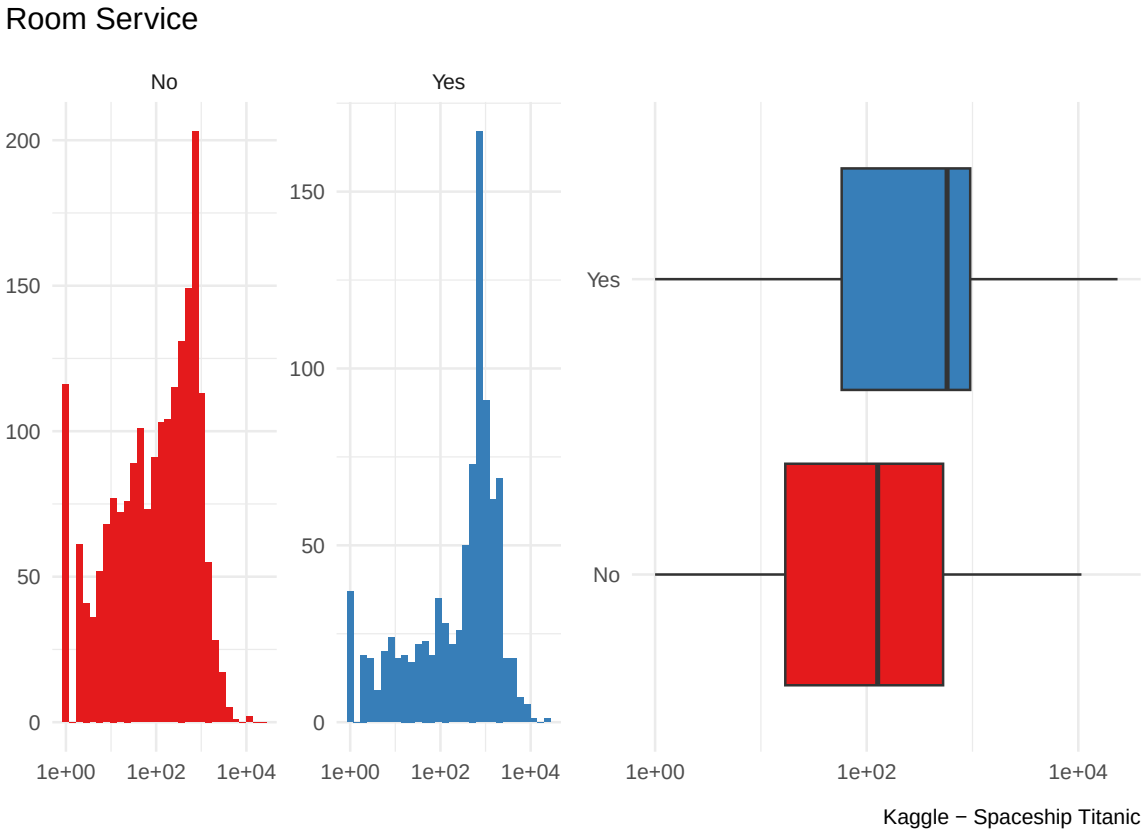
Shopping Mall

Analyzing the shopping expenses, we observe similar skewness, which we will address with data transformation. Post-transformation observations hint that passengers who were not transported may generally spend less on shopping.

```r
p_shopping1 <-
  train_data |>
    ggplot(aes(x = ShoppingMall, fill = Transported)) +
    geom_histogram() +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    facet_wrap(. ~ Transported, scales = "free_y")

p_shopping2 <-
  train_data |>
    ggplot(aes(x = ShoppingMall, y = Transported, fill = Transported)) +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    geom_boxplot()


(p_shopping1 + p_shopping2) +
  plot_annotation(title = "Room Service",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```



Spa

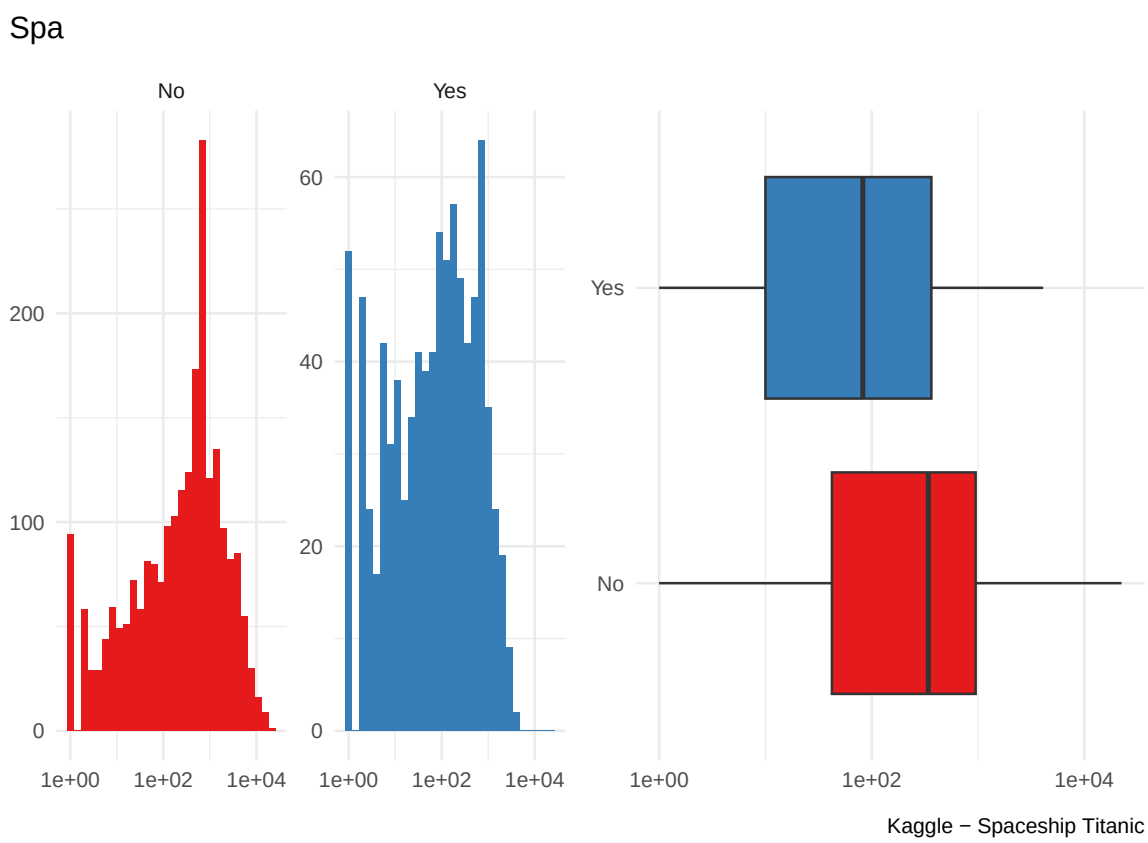
The expenditure data for the spa services also presents outliers, with certain individuals spending significantly more. After applying a log transformation to mitigate the effects of these outliers, we note that non-transported individuals appear to allocate more towards spa services.

```
p_spa1 <-
  train_data |>
    ggplot(aes(x = Spa, fill = Transported)) +
    geom_histogram() +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    facet_wrap(. ~ Transported, scales = "free_y")

p_spa2 <-
  train_data |>
    ggplot(aes(x = Spa, y = Transported, fill = Transported)) +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
```

```
    geom_boxplot()

(p_spa1 + p_spa2) +
  plot_annotation(title = "Spa",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```



VR Deck

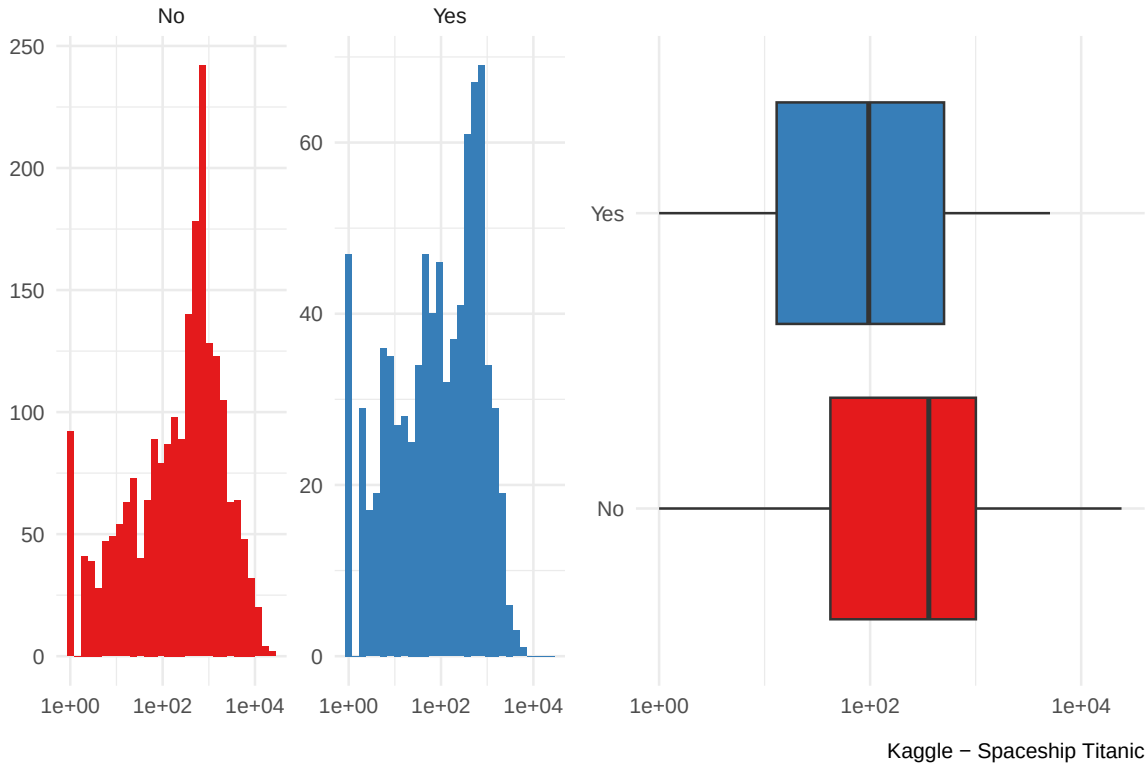
Lastly, the expenses on the VR deck also exhibit right skewness, with a few passengers spending substantially more. The transformed data indicates a correlation where passengers with higher VR deck expenses are less likely to be transported.

```r
p_vr1 <-
  train_data |>
    ggplot(aes(x = VRDeck, fill = Transported)) +
    geom_histogram() +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    facet_wrap(. ~ Transported, scales = "free_y")

p_vr2 <-
  train_data |>
    ggplot(aes(x = VRDeck, y = Transported, fill = Transported)) +
    scale_x_log10(labels = scales::scientific, breaks = c(1, 1e2, 1e4)) +
    geom_boxplot()

(p_vr1 + p_vr2) +
  plot_annotation(title = "VR Deck",
                  caption = "Kaggle - Spaceship Titanic") &
  scale_fill_brewer(type = "qual", palette = 6) &
  labs(x = NULL, y = NULL) &
  guides(fill = "none")
```

VR Deck



Categorical Variables

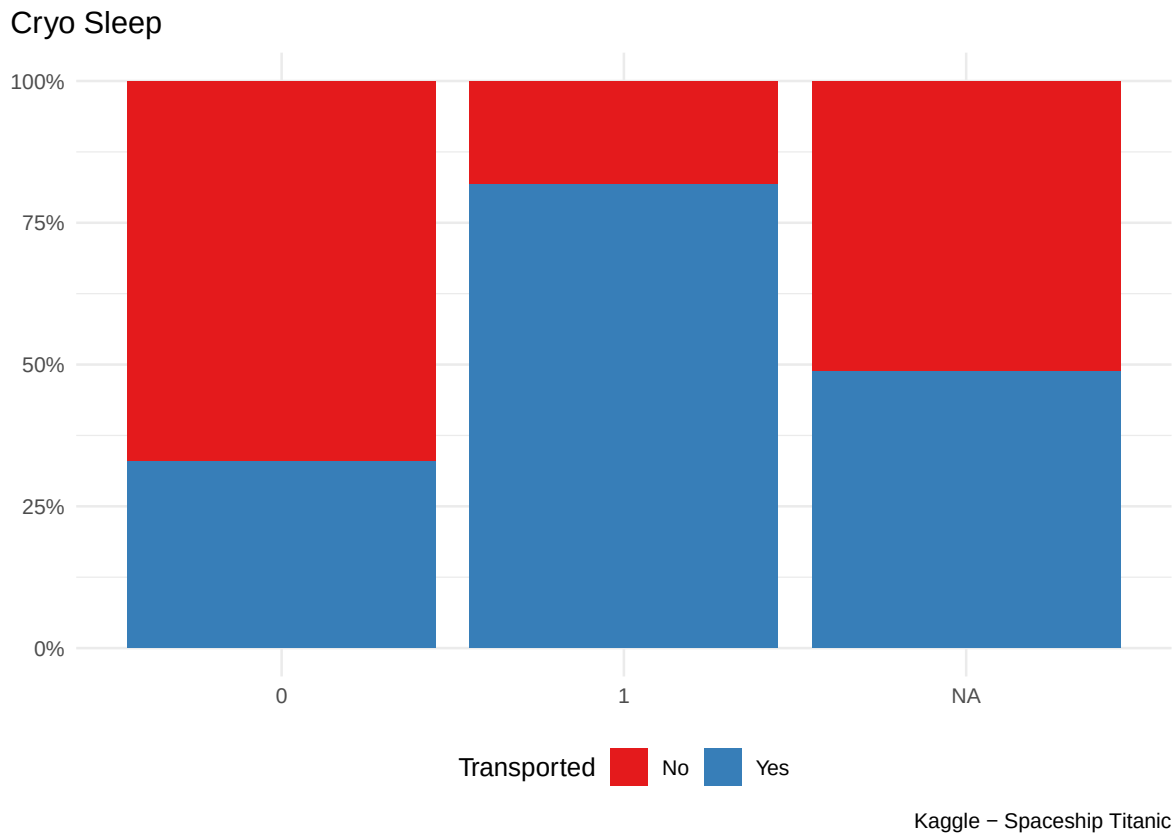
Moving on to examining categorical variables.

Cryo Sleep

Our exploration begins with passengers' being put into cryo sleep during the travel. Slightly less than 35% was for suspended to be put into cryo sleep. Notably, over three-quarters of these passengers were transported, compared to a mere 30% transport rate among those who stayed awake.

```
train_data |>
  ggplot(aes(x = CryoSleep, fill = Transported)) +
  geom_bar(position = "fill") +
  scale_fill_brewer(type = "qual", palette = 6) +
  scale_y_continuous(labels = percent) +
```

```
    labs(title = "Cryo Sleep",
         caption = "Kaggle - Spaceship Titanic",
         x = NULL, y = NULL) +
    theme(legend.position = "bottom")
```

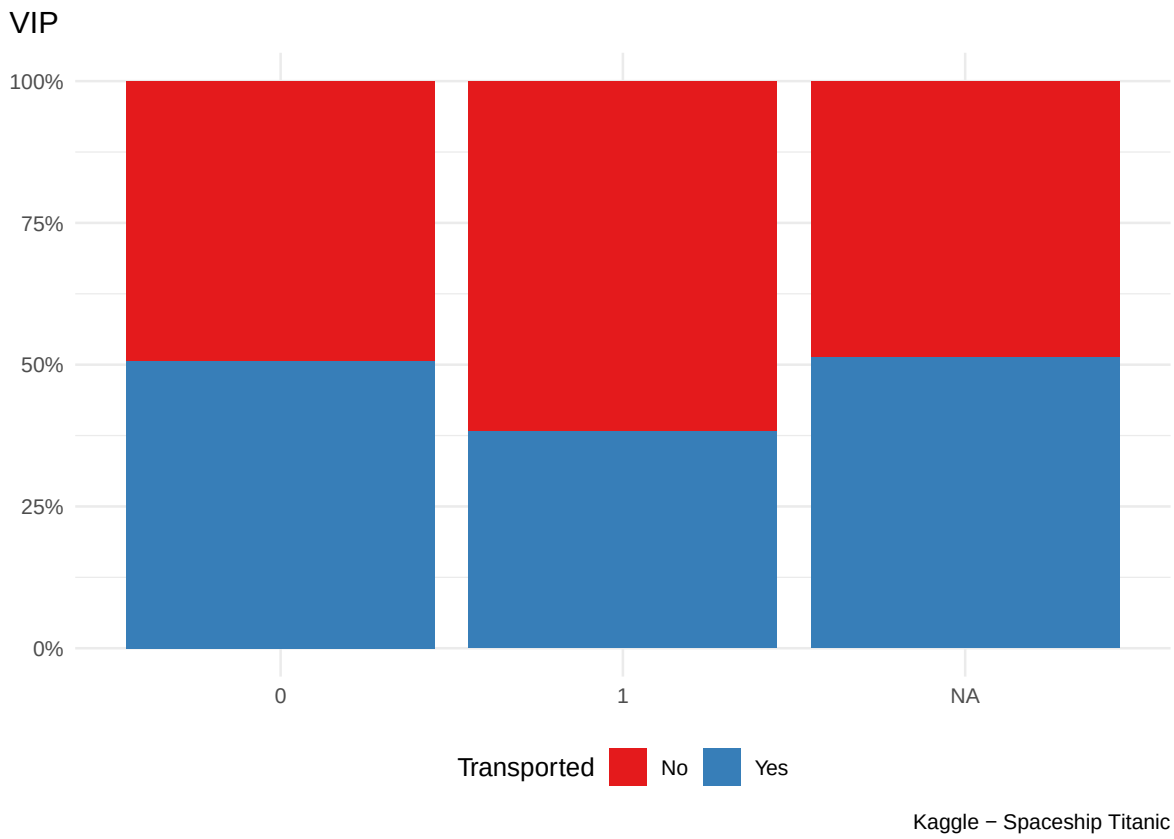


VIP

The ship's records also show that VIP services as an option for passengers. These exclusive amenities were enjoyed by only about 2.2% of passengers. Interestingly, while non-VIPs had a roughly equal chance of transport, a larger proportion of VIP passengers ended up not being transported.

```
train_data |>
  ggplot(aes(x = VIP, fill = Transported)) +
  geom_bar(position = "fill") +
   scale_fill_brewer(type = "qual", palette = 6)  +
```

```
    scale_y_continuous(labels = percent) +
    labs(title = "VIP",
         caption = "Kaggle - Spaceship Titanic",
         x = NULL, y = NULL) +
    theme(legend.position = "bottom")
```

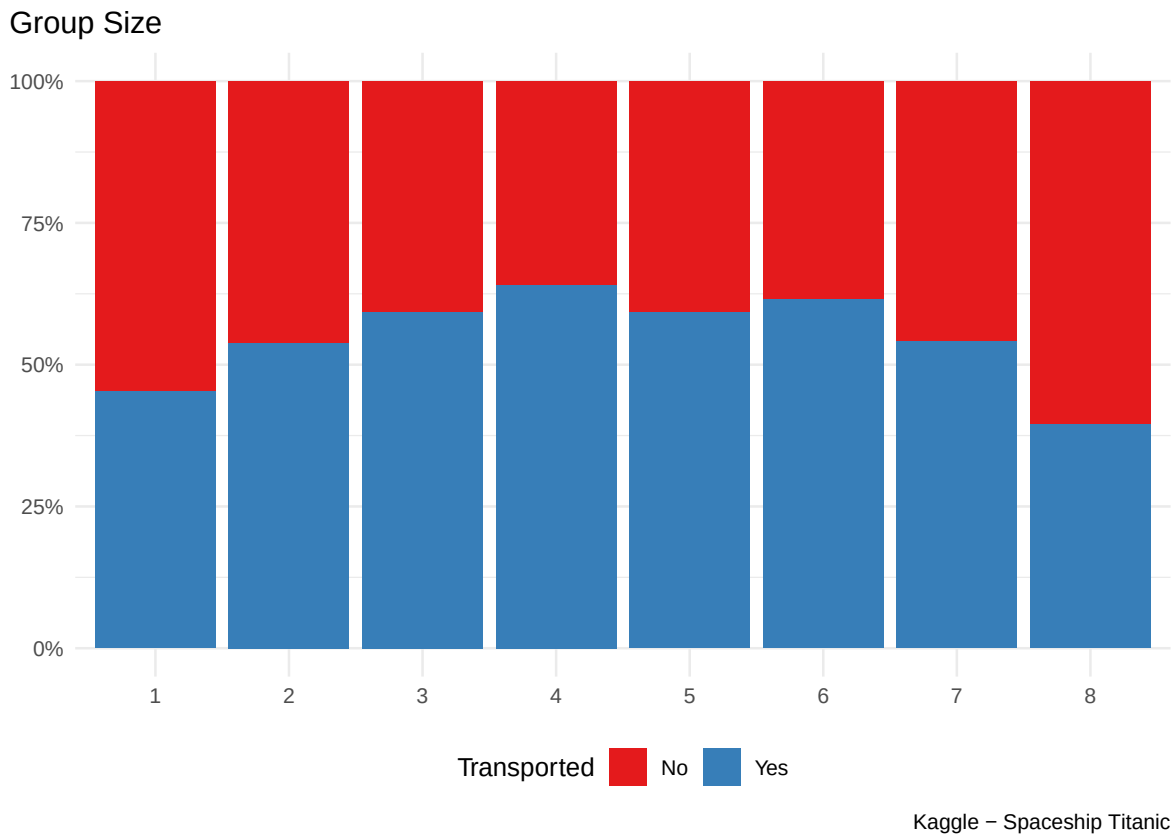


Group Size

Most passengers embarked on their space voyage alone, but there were also those who traveled in groups. These groups ranged in size, sometimes including as many as eight individuals. The data suggests that groups of three to six had higher transport rates.

```
train_data |>
  ggplot(aes(x = GroupSize, fill = Transported)) +
  geom_bar(position = "fill") +
   scale_fill_brewer(type = "qual", palette = 6)  +
```

```
    scale_y_continuous(labels = percent) +
    labs(title = "Group Size",
         caption = "Kaggle - Spaceship Titanic",
         x = NULL, y = NULL) +
    theme(legend.position = "bottom")
```

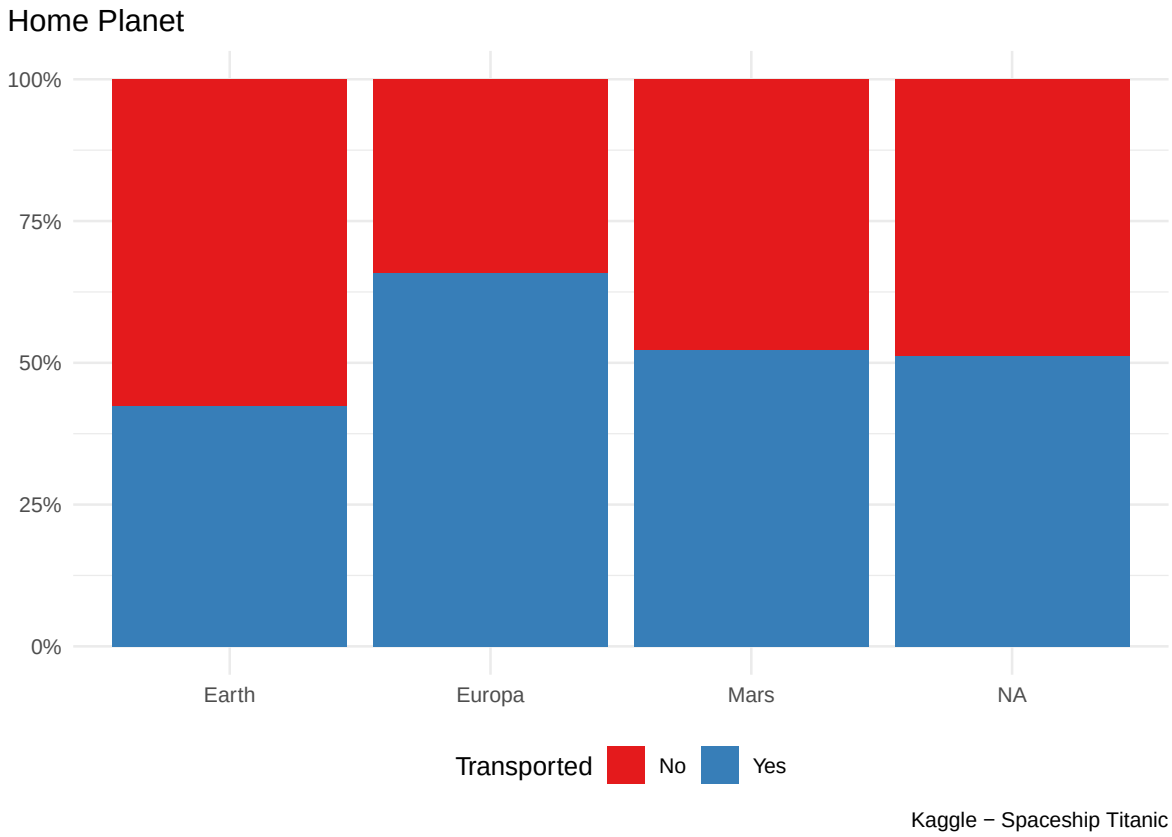


Home Planet

We can also observe the the planet from which passengers depart. More than half were from the earth, while others came from either Europa and Mars. There's a notable trend in transport success rates among these groups: Earth's travelers were less likely to be transported, but, for passengers from Europa, the situation flips as more than 60% of them were transported.

```
train_data |>
  ggplot(aes(x = HomePlanet, fill = Transported)) +
  geom_bar(position = "fill") +
```

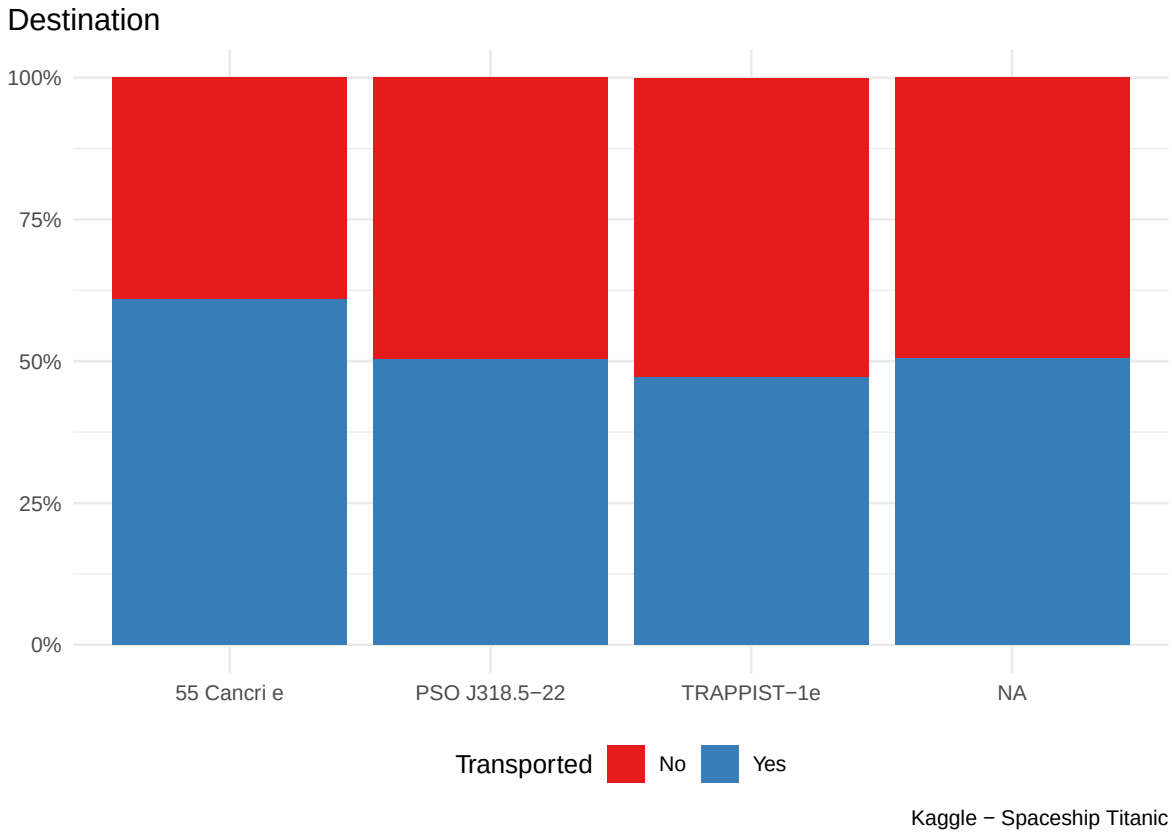
```
    scale_fill_brewer(type = "qual", palette = 6)  +
    scale_y_continuous(labels = percent) +
    labs(title = "Home Planet",
         caption = "Kaggle - Spaceship Titanic",
         x = NULL, y = NULL) +
    theme(legend.position = "bottom")
```



Destination

Most passengers headed to TRAPPIST-1e as their destination, and, from those, there were just below 50% of them being transported. About 20% of passengers intended to traveled to 55 Cancri e with more than 60% of them successfully transported into alternate dimension. The other 10% were intended to go to PSO J31 8.5-22. Within this group, the odds of being transported is approximately 50%.

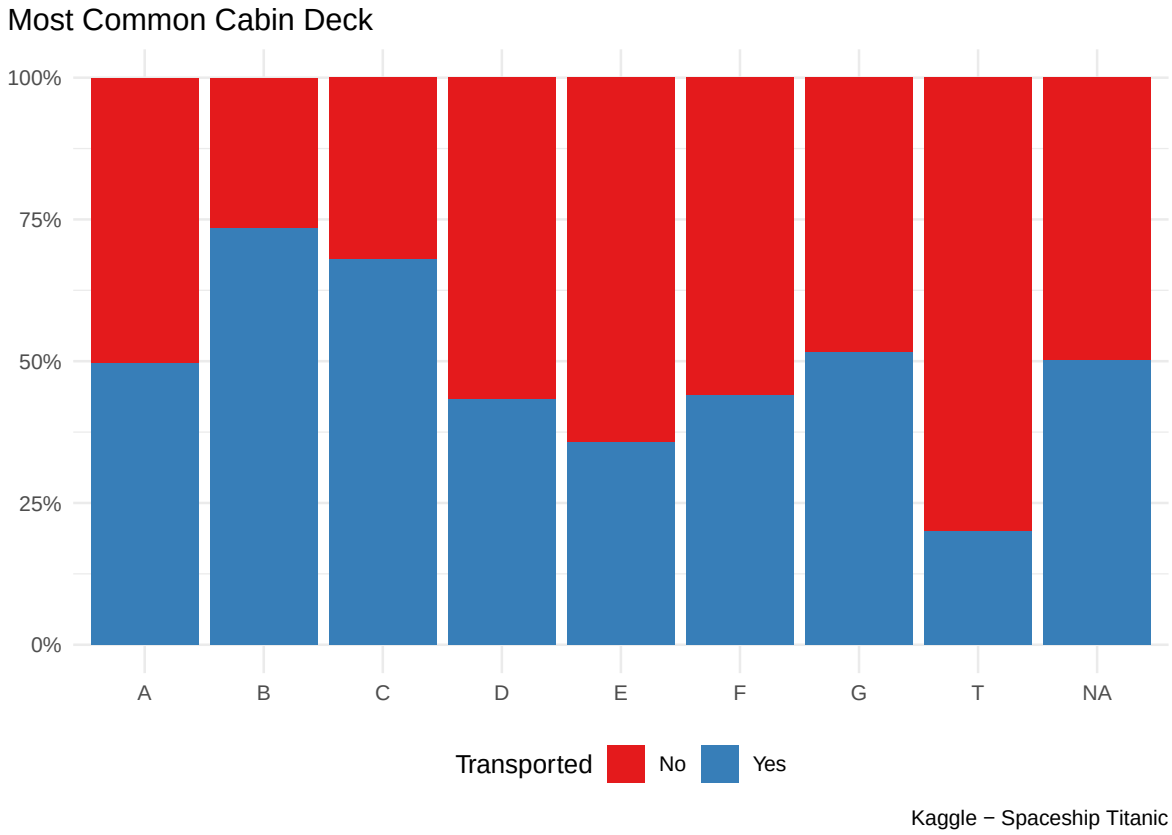
```r
train_data |>
  ggplot(aes(x = Destination, fill = Transported)) +
  geom_bar(position = "fill") +
  scale_fill_brewer(type = "qual", palette = 6)  +
   scale_y_continuous(labels = percent) +
   labs(title = "Destination",
        caption = "Kaggle - Spaceship Titanic",
        x = NULL, y = NULL) +
   theme(legend.position = "bottom")
```



Cabin Deck

From their ticket, we can extract the cabin a passenger were on. About 60% were allocated on cabin F and G, about another 30% were on cabin B, C, and E, while the rest were on cabin A, D, and T. Cabin T was the cabin with the lowest percentage of people getting transported. On the contrary, cabin B and C were cabins with the highest proportion.

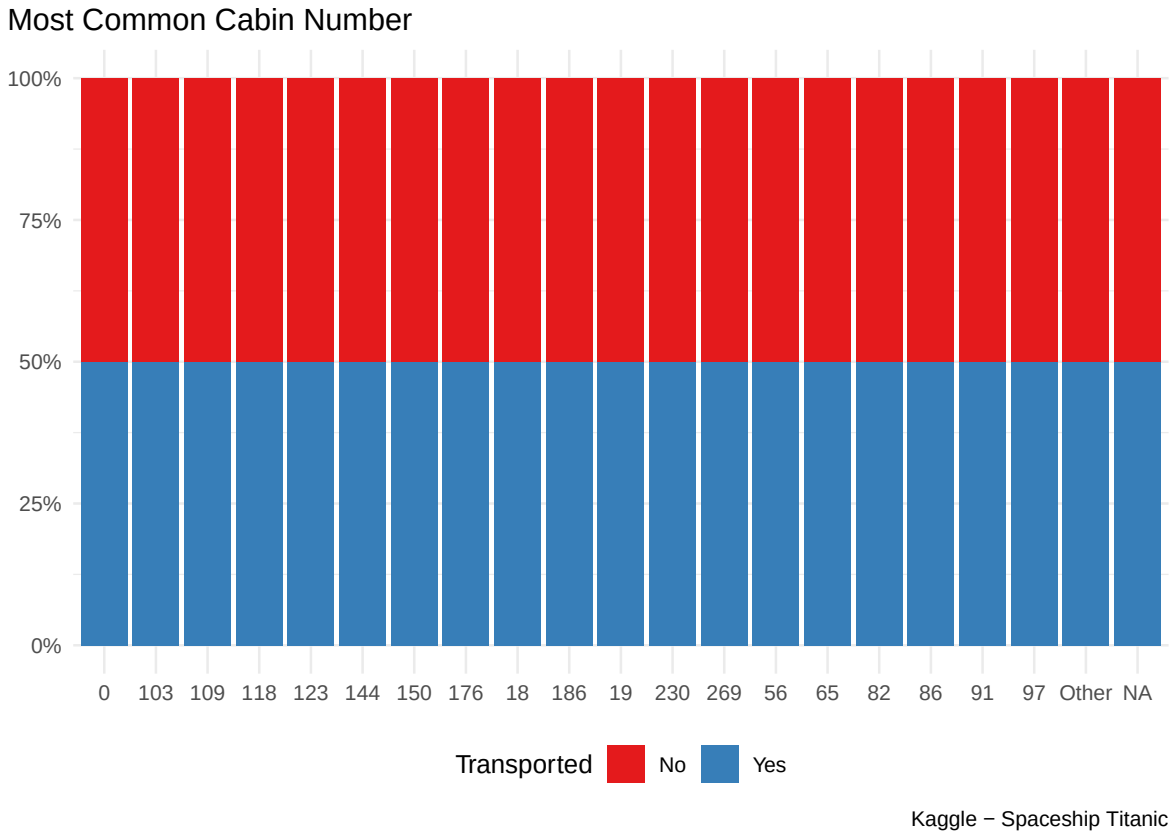
```r
train_data |>
  ggplot(aes(x = CabinDeck, fill = Transported)) +
  geom_bar(position = "fill") +
   scale_fill_brewer(type = "qual", palette = 6)  +
   scale_y_continuous(labels = percent) +
   labs(title = "Most Common Cabin Deck",
        caption = "Kaggle - Spaceship Titanic",
        x = NULL, y = NULL) +
   theme(legend.position = "bottom")
```



Cabin Number

Another inspection that we can made from passenger's tickets is the cabin number. However, the graph below demonstrates that cabin number does not provide any useful information on whether passengers were transported. We can opt to omit this variable from the predictive model development.

```r
train_data |>
  count(CabinNum, Transported) |>
  mutate(TotalCabinNum = sum(n), .by = CabinNum) |>
  mutate(CabinNum = if_else(TotalCabinNum / sum(n) >= 0.002, CabinNum, "Other")) |>
  summarise(n = sum(n), .by = c(CabinNum, Transported)) |>
  ggplot(aes(x = CabinNum, fill = Transported)) +
  geom_bar(position = "fill") +
   scale_fill_brewer(type = "qual", palette = 6)  +
   scale_y_continuous(labels = percent) +
   labs(title = "Most Common Cabin Number",
        caption = "Kaggle - Spaceship Titanic",
        x = NULL, y = NULL) +
   theme(legend.position = "bottom")
```

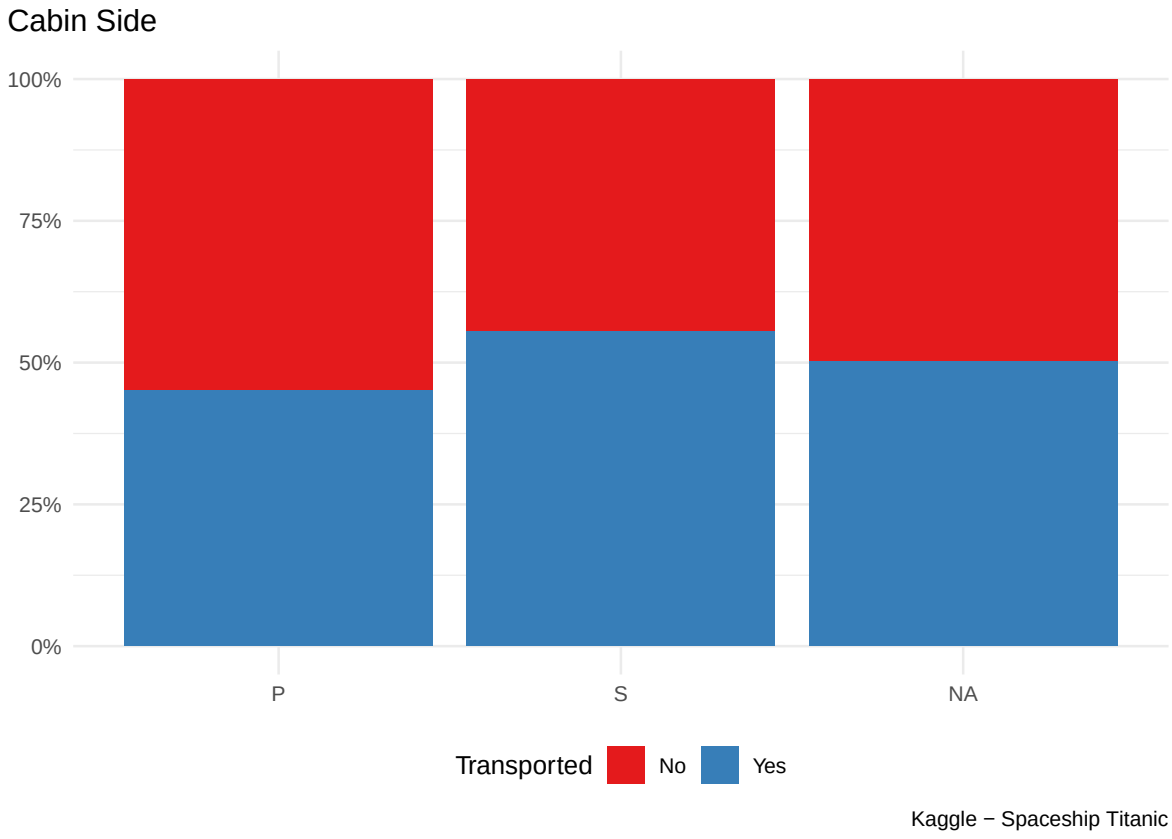


Cabin Side

Lastly, we can obtain information of the side a passenger were on during the travel. We can

see from the table that passengers were proportionally distributed into both cabin side P and S. People on the cabin P seemed less likely to be transported while the opposite was true for the cabin S.

```
train_data |>
  ggplot(aes(x = CabinSide, fill = Transported)) +
  geom_bar(position = "fill") +
   scale_fill_brewer(type = "qual", palette = 6)  +
   scale_y_continuous(labels = percent) +
   labs(title = "Cabin Side",
        caption = "Kaggle - Spaceship Titanic",
        x = NULL, y = NULL) +
   theme(legend.position = "bottom")
```



Model Development

For this section, we will delve into various pre-processing sets as well as machine learning models. The `tidymodels` package presents a compelling methodology to address these challenges. We can leverage and combine different predefined pre-processing functions into a `recipe`. In addition, the package allows standardized approach to define models specification and tune their hyperparameters within a unified `workflow`.

Specifically, we will utilize the k-fold cross-validation methods with k equal to 10. This method partitions the data into ten subsets, or 'folds', and using each fold as a validation set against models trained on the remaining nine folds. This iterative process not only helps in fine-tuning the model parameters for better generalization but also provides a robust measure of model performance, thereby preventing overfitting.

Define Pre-processing Steps

Here, we use three different pre-processing recipes which are designed with escalating complexity with regards to how they handle missing values, transform and normalize the data, and reduce dimensionality. Here is a description of each:

1. **Basic Preprocessing Recipe:** This recipe focuses on cleaning the dataset by handling missing values and preparing categorical variables for modeling. Missing categorical data is filled in with the most frequent category (mode imputation), while missing numerical data is replaced with the median. In the end, categorical variables are converted into a series of binary (dummy) variables as some models cannot directly handle categorical data.
2. **Transformed Preprocessing Recipe:** This recipe enhances the basic preprocessing. It introduces log transformation to the numerical variables related to services (like RoomService, FoodCourt, etc.) to reduce skewness observed previously on exploration phase. It then normalizes all numeric predictors to ensure they're on the same scale, crucial for models that are sensitive to the scale of input data. Additionally, it employs a more sophisticated imputation strategy using bagging method for all predictors to handle missing values across the dataset. Categorical variables are thus processed into dummy variables, similar to the basic recipe.
3. **PCA Preprocessing Recipe:** The PCA recipe takes the transformed preprocessing further by incorporating feature extraction through PCA. This is particularly useful for extracting the most important features from the data, simplifying the model, and potentially improving model performance by removing noise and redundancy in the data.

```
basic_preproc <-
  recipe(Transported ~ GroupSize + HomePlanet + CryoSleep + CabinDeck +
           CabinSide + Destination + Age + VIP + RoomService +
           FoodCourt + ShoppingMall + Spa + VRDeck,
         data = train_data) |>
  step_impute_mode(all_nominal_predictors()) |>
  step_impute_median(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors())

transform_preproc <-
  recipe(Transported ~ GroupSize + HomePlanet + CryoSleep + CabinDeck +
           CabinSide + Destination + Age + VIP + RoomService +
           FoodCourt + ShoppingMall + Spa + VRDeck,
         data = train_data) |>
  step_log(RoomService, FoodCourt, ShoppingMall, Spa, VRDeck, offset = 1) |>
  step_normalize(all_numeric_predictors()) |>
  step_impute_bag(all_predictors()) |>
  step_dummy(all_nominal_predictors())

pca_preproc <-
  transform_preproc |>
  step_pca(all_predictors())

preproc <-
  list(
    "basic" = basic_preproc,
    "trans" = transform_preproc,
    "pca" = pca_preproc
  )
```

Define Model Specifications

```
# Define models
# 1: Logistic Regression
logreg_spec <-
  logistic_reg(engine = "glmnet", penalty = tune(), mixture = tune()) |>
  set_mode("classification")

# 2: Decision Tree
dtree_spec <-
```

```
  decision_tree(#cost_complexity = tune(),
                #tree_depth = tune(),
                min_n = tune()) |>
  set_mode("classification")

# 3: XGBoost
btree_spec <-
  boost_tree(#trees = tune(), min_n = tune(), tree_depth = tune(),
             #mtry = tune(), sample_size = tune(), loss_reduction = tune(),
             stop_iter = tune()) |>
  set_mode("classification")

# 4: Random Forest
rf_spec <-
  rand_forest(trees = tune(),
              #min_n = tune(), mtry = tune()
              ) |>
  set_mode("classification")


# List all models
models <-
  list(
    "LogisticRegression" = logreg_spec,
    "DecisionTree" = dtree_spec,
    "BoostedTree" = btree_spec,
    "RandomForest" = rf_spec
  )
```

Hyperparameter Tuning

Cross-validation

`vfold_cv`: ...

Workflows

`workflow_set`: ...

`workflow_map`: ...

```r
# Tune model's hyperparameters
cl <- makePSOCKcluster(8)

registerDoParallel(cl)

# Split data
set.seed(1234)
train_folds <- vfold_cv(train_data, v = 10)

results <-
  workflow_set(preproc = preproc, models = models, cross = TRUE) |>
  workflow_map(resamples = train_folds, fn = "tune_grid", verbose = TRUE,
               grid = 1, seed = 1234)

stopCluster(cl)

# check the results
results
```

Extracting Information

<info [1 x 4]>: ...

Extracting Options

<opts[2]>: ...

Extracting Results

<tune[+]>: ...

```r
# Make Final Predictions --------------------------------------------------

# find the model with the best hyperparameter
top_model_name <-
  results |>
  rank_results(rank_metric = "accuracy", select_best = TRUE) |>
  pull(wflow_id) |>
  _[1]
```

```r
best_parameters <-
  results |>
  extract_workflow_set_result(top_model_name) |>
  select_best(metric = "accuracy")
```

```
best_parameters
```

Make Submission

```
# train with full data
final_model <-
  results |>
  extract_workflow(top_model_name) |>
  finalize_workflow(best_parameters) |>
  fit(train_data)
```

```
# make predictions
submission <-
  predict(final_model, test_data) |>
  bind_cols(select(test_data, PassengerId)) |>
  mutate(Transported = if_else(.pred_class == 1, "True", "False")) |>
  select(PassengerId, Transported)

write_csv(submission, "../data/submission1.csv")
```

Conclusion