Prosjekt Titanic overlevelse

Tobias Windingstad og Ørjan Hammer

Introduksjon til oppgaven:

I denne oppgaven skal vi benytte et datasett fra Kaggle.com, som inneholder informasjon om passasjerene på Titanic – historiens mest kjente skipsforlis. Natt til 15. april 1912 sank Titanic etter å ha kollidert med et isfjell i Nord-Atlanteren. Datasettet gir oss informasjon om blant annet passasjerenes alder, kjønn, klasse, billettpris og avreisested.

Målet med prosjektet er å anvende maskinlæring for å forutsi hvilke faktorer som hadde størst betydning for overlevelse. Vi skal trene en modell som kan forutsi om en gitt passasjer ville ha overlevd ulykken. Vi vil analysere variabler og deres innvirkning på overlevelse, og til slutt bygge en prediksjonsmodell som kan teste ulike scenarier.

Prosjektet er skrevet i R Markdown, hvor vi benytter både R og ulike maskinlæringsbiblioteker for å gjennomføre analysen og prediksjonen. Gjennom prosjektet vil vi også vurdere modellens ytelse og mulige forbedringer for å øke prediksjonens presisjon.

Helt til slutt vil vi implementere et brukergrensesnitt som lar brukeren "kjøpe en billett" til Titanic, transformere dataen og predikere om personen ville overlevd basert på dataen hen oppgir.

Dependencies

Filer

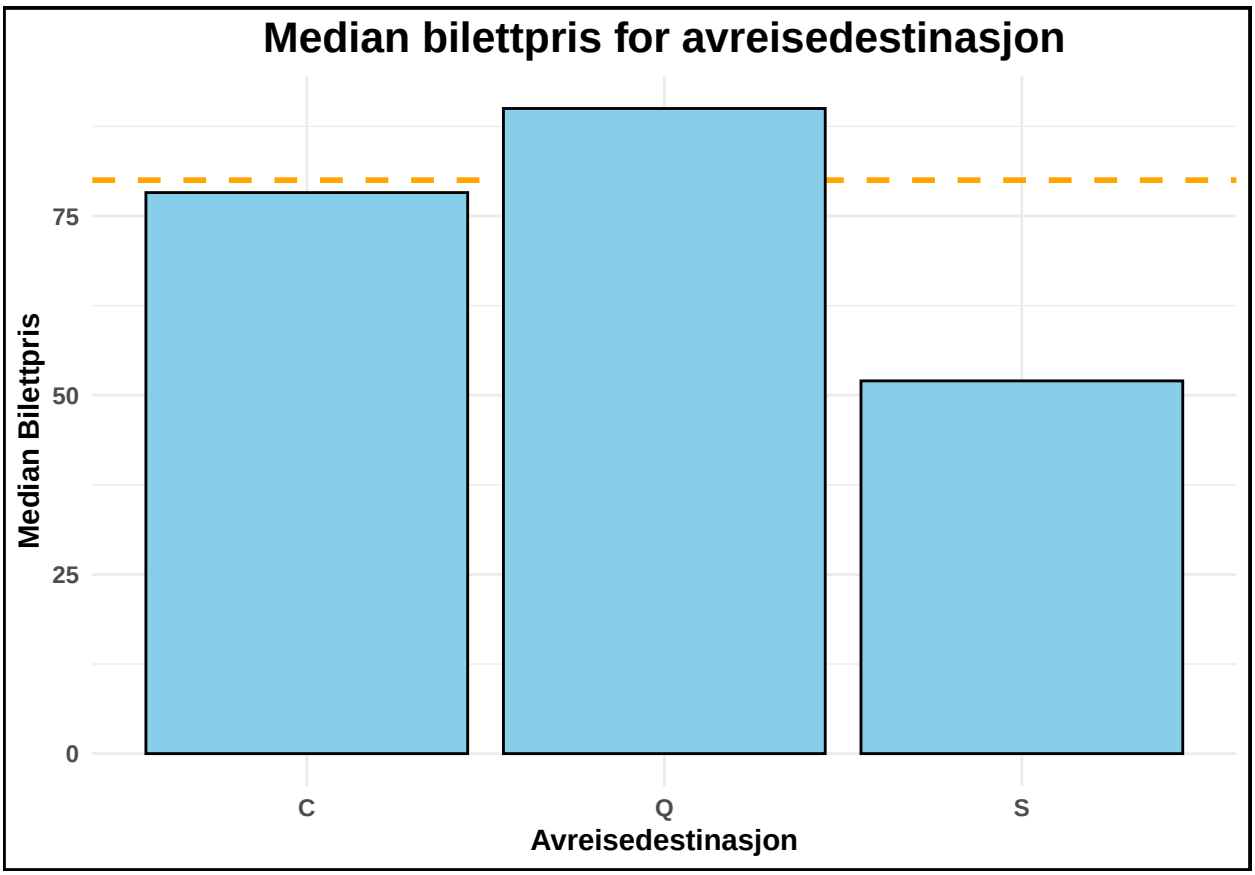
Data wrangling:

Data wrangling er prosessen med å forberede, rense og strukturere data for analyse (Kilde). I dette prosjektet håndterer vi flere viktige aspekter ved datasettet for å sikre at det er klart for modellering. Her er hva vi gjør i koden:

```
path <- paste(getwd(), "/data/", "Titanic-Dataset.csv", sep = '')
data <- wrangle_data(path = path)
title_dist <- data$title_dist
data <- data$data
na_data <- wrangle_data(na = TRUE, path = path)
```

1. Behandling av manglende verdier (NA)

Avreisehavn: To av passasjerene mangler avreisehavn. Siden begge disse passasjerene har reist med førsteklasse og betalt samme pris finner vi gjennomsnittsprisen for alle førsteklasse-reisende for hver havn. Vi setter så avreisehavnen for de to passasjerene til den havnen som korrelerer best med prisen de har betalt. Vi lagde senere en funskjon som gjorde dette for alle havner ettersom vi trengte dette i grensesnittet.

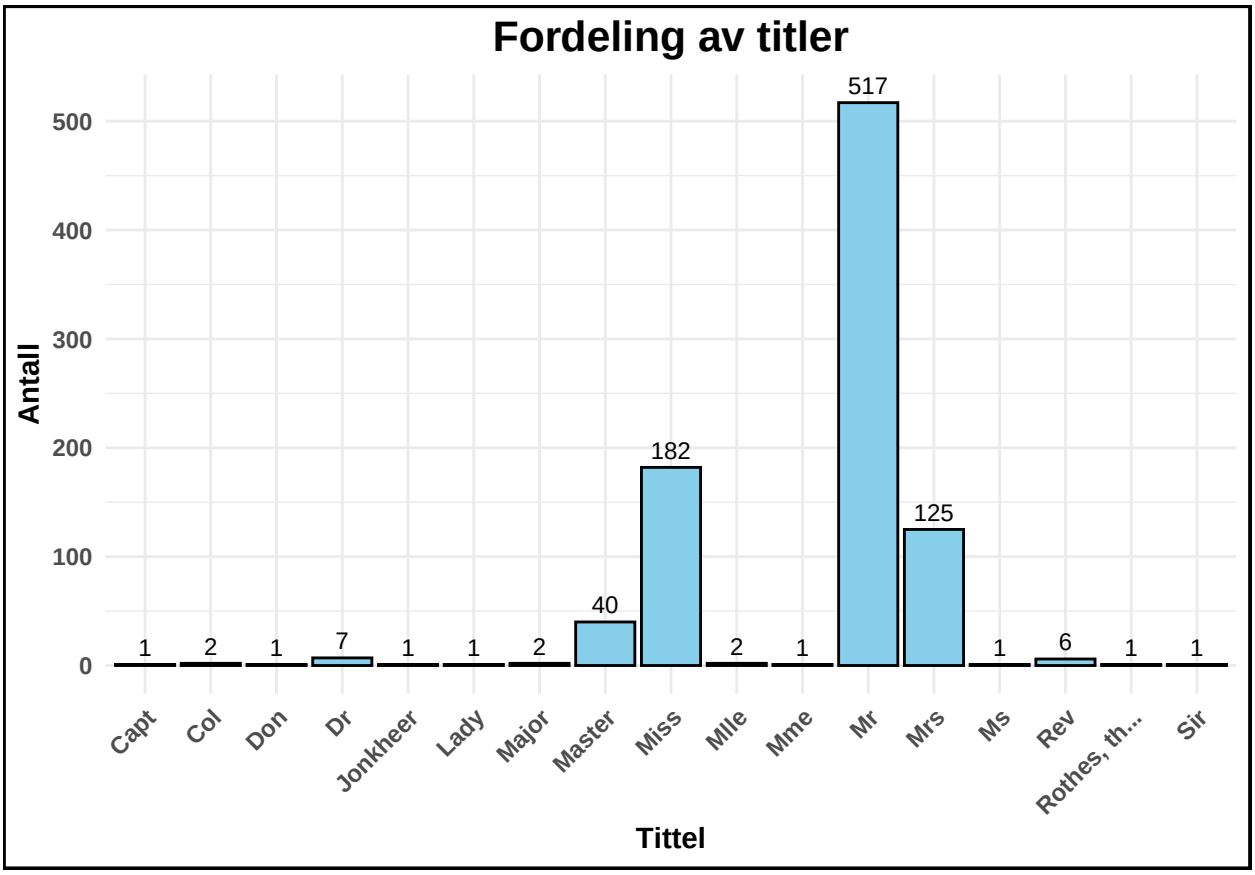


Alder For passasjerer med manglende alder, bruker vi en median-alder som standard, med en ekstra detalj for personer som reiste med søsken eller ektefelle (SibSp > 0), men ikke med barn eller foreldre (Parch $== 0$). I slike tilfeller bruker vi gjennomsnittsalderen for personer med samme etternavn (antatt å være søsken eller ektefeller). Dette gir mange av de reisende medianalder, men vi mener at ved å legge til denne behandlingen vil vi kunne ha noe sterkere antakelser om alderen til noen av passasjerene. Vi ser det kunne vært mulighet for at andre faktorer kan påvirke alder, som f.eks. tittel eller klasse, men vi har ikke tatt hensyn til dette i prosjektet.

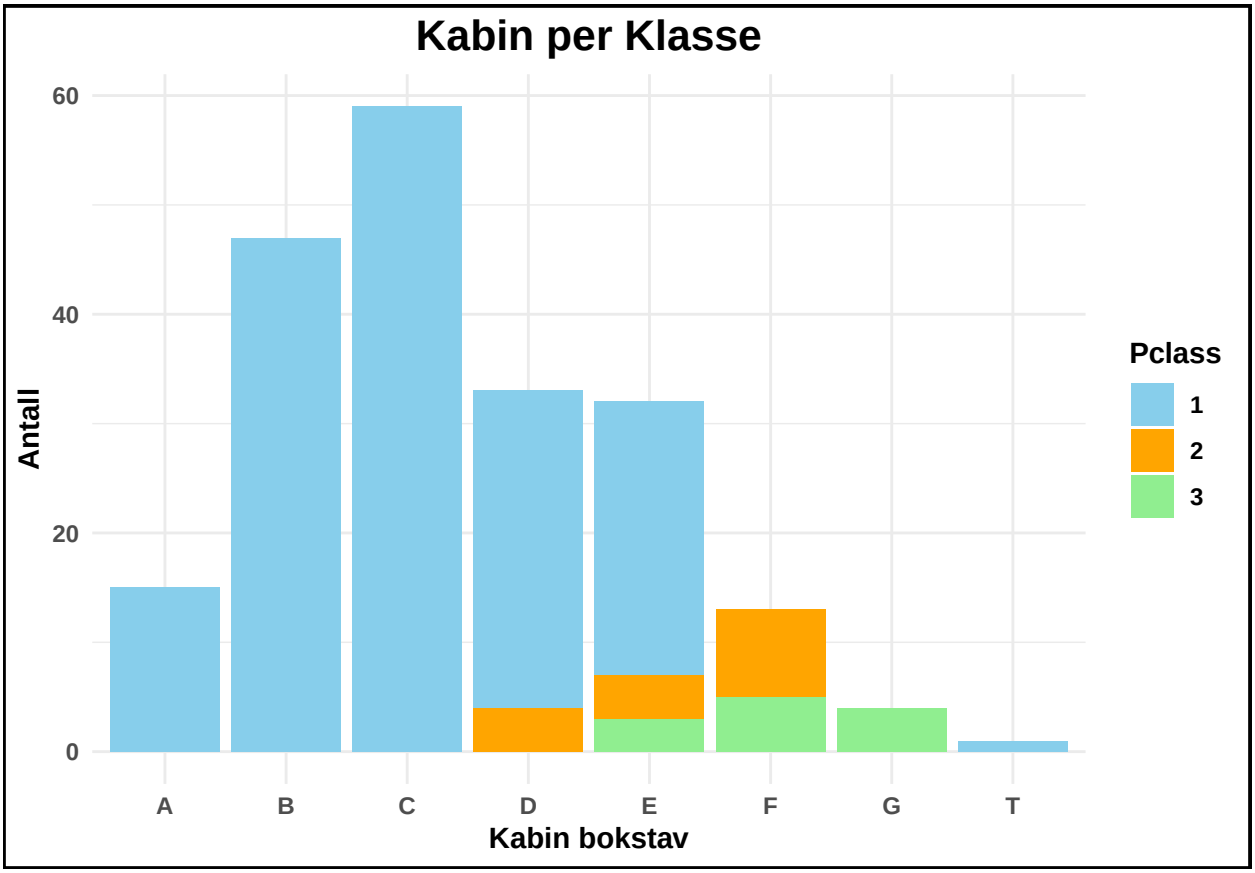
```
handle_na_age <- function(data) {
  median_age <- median(data$Age, na.rm = TRUE)
  data <- data %>%
    mutate(Age = ifelse(is.na(Age), ifelse((SibSp > 0) & (Parch == 0), getSibSpAge(Name, data), median_a
  return(data)
}

getSibSpAge <- function(name, data) {
  last_name <- get_last_name(name)

  sib_sp <- filter(group_by(data, Name), get_last_name(Name) == last_name)

  estimated_age <- round(mean(sib_sp$Age, na.rm = TRUE))
  return (estimated_age)
}

get_last_name <- function(name) {
  last_name <- strsplit(name, ',')[[1]][1]
```

```
  return (last_name)
}
```

2. Ekstrahere tittel Vi ønsket å isolere tittelen fra navnet og legge den til i en egen kolonne. Vår teori er at dette kan gjøre dataen mer tilpasset maskinlæring ettersom tittelen kan være en indikator på faktorer som sivilstatus, alder eller sosioøkonomisk status. Etter å ha testet modellen og sjekket variabler innså vi at det var svært mange ulike titler. Vi valgte derfor å håndtere dette ved å putte alle sjeldne titler (<10) inn i en "other" varaibel.



3. Fjerne irrelevante og manglende variabler For å redusere kompleksiteten i datasettet fjerner vi kolonnen Cabin ettersom den har svært mange i datasettet som ikke hadde denne variabelen. Vi så først på muligheten for at det var en tydelig sammenheng mellom Pclass og Cabin ved bruk av en Chi-Squared. Det var tydelig at passasjerer i Pclass 1 var konsentrert i A, B, C, D og E kabiner, men ettersom det er svært lite data på hvor de resterende reisende bodde, ville vi ikke gjøre noen generalisering rundt dette. Name, Ticket og PassangerId er alle individuelle verdier som ikke vil ha noen påvirkning på modellene så vi fjernet disse.



Gjennom denne prosessen sørger vi for at datasettet er renere, mer konsistente og at de manglende variablene er håndtert.

Lag dummy-data og initial split

```
model_data <- create_dummy_data(data)
t_train <- model_data$t_train
t_test <- model_data$t_test
```

Tren modeller

Vi fant ut at de mest relevant modellene for problemet var Lasso, Random Forest og Gradient Boosting. Her implementerer vi dem, og finner de beste hyperparameterne.

```
# Tuned models
tuned_lso <- create_tuned_model("lasso", t_train)
tuned_rf <- create_tuned_model("random_forest", t_train)
tuned_xgb <- create_tuned_model("xgboost", t_train)
```

Videre trener vi modellen og undersøker treffsikkerheten deres i å avgjøre hvem som overlever.

```
lso_fit <- tuned_lso %>%
  fit(Survived ~., data = t_train)
rf_fit <- tuned_rf %>%
  fit(Survived ~., data = t_train)
xgb_fit <- tuned_xgb %>%
  fit(Survived ~., data = t_train)
```

```r
tuned_lso_pred <- predict(lso_fit, new_data = t_test, type = "class")$.pred_class
tuned_rf_pred <- predict(rf_fit, new_data = t_test, type = "class")$.pred_class
tuned_xgb_pred <- predict(xgb_fit, new_data = t_test, type = "class")$.pred_class

tuned_lso_acc <- mean(tuned_lso_pred == t_test$Survived)
tuned_rf_acc <- mean(tuned_rf_pred == t_test$Survived)
tuned_xgb_acc <- mean(tuned_xgb_pred == t_test$Survived)

accs <- tibble (
  Model = c("LASSO", "RANDOM FOREST", "XGBOOST"),
  acc = c(tuned_lso_acc, tuned_rf_acc, tuned_xgb_acc)
)
print(accs)
```

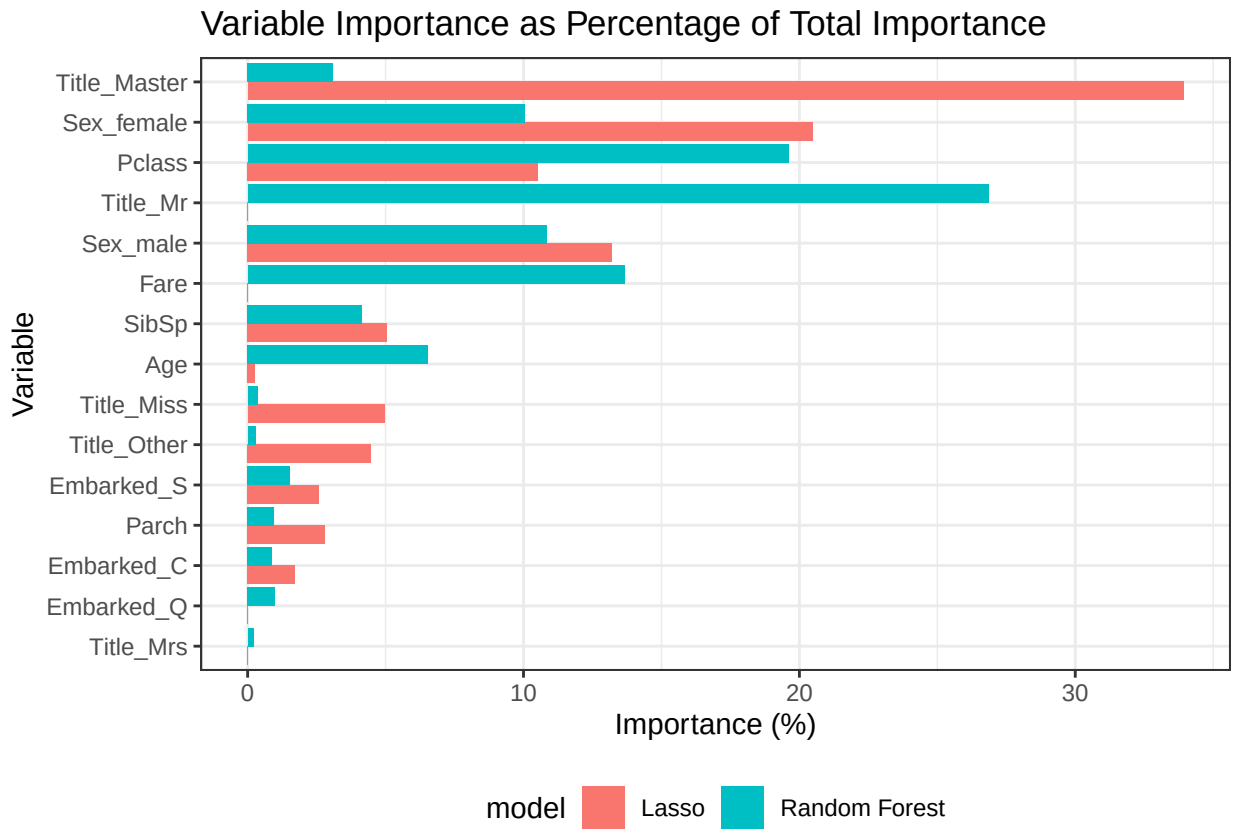
```
## # A tibble: 3 x 2
##   Model           acc
##   <chr>         <dbl>
## 1 LASSO         0.827
## 2 RANDOM FOREST 0.844
## 3 XGBOOST       0.816
```

Som man kan se er modellene relativt like treffsikre. At denne ligger på rundt 85% er ikke isolert sett særlig imponerende. Vi tror likevel dette er omtrent hvor treffsikker en slik model kan bli gitt problemstillingen samt hvor lite datasettet er. NOE MER?

Hvilke faktorer påvirker resultatet mest

Vi bruker VIP-biblioteket for å undersøke hvilke variabler som har størst innvirkning på modellen.

```r
vip_plot <- plot_vip_lso_rf(tuned_lso, tuned_rf, t_train)
print(vip_plot)
```



Brukergrensesnitt

I brukergrensesnittet har vi implementert mulighet for en bruker å «kjøpe en billett» til Titanic. Når dataen er samlet inn bruker vi informasjonen i modellen for å forutsi om brukeren ville overlevd forliset eller ikke. Vi benyttet biblioteket «shiny» for å lage grensesnittet. Informasjonen vi henter fra brukeren er:

1. All relevant informasjon for modellen. Tittel, alder, kjønn, avreisested, billettklasse og om man reiser med søsken, barn, ektefelle eller foreldre.
2. Grensesnittet ber også om brukerens navn til tross for at det er individuelle verdier som ikke benyttes i modellen. Dette gjør vi ettersom vi mener det gir en mer realistisk brukeropplevelse.

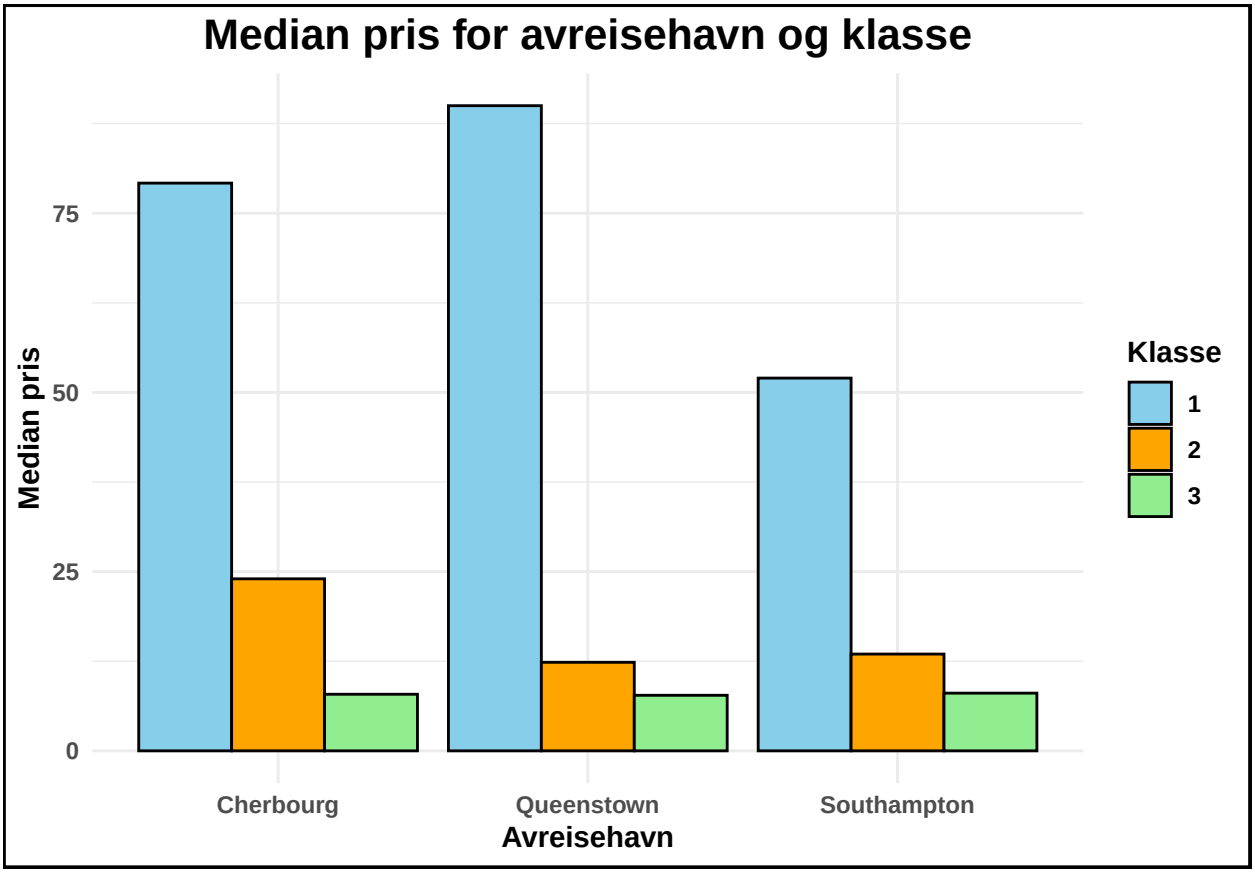
Håndtering av data i grensesnitt Planen fra starten av wranglingen i grensesnittet var å lage en tibble som matchet det råe datasettet slik at vi kunne sende denne dataen inn i wrangling.R og videre inn i model_data.R for å gjøre dataen klar for modellen. Vi innså dog at det hadde dukket opp noe teknisk gjeld, med kun noen dager igjen til prosjektet skulle leveres.

Model_data.R, som håndterte dataen og lagde dummydata splittet også opp dataen i treningsdata og testdata. Vi så på muligheten for å splitte opp denne funskjonen, men når vi kjørte "bake" på den nye dataen med kun en rad fungerte ikke dette, ettersom "bake" ikke kan opprette alle dummy-variablene vi trengte. Ettersom grensesnittet kun besto av en rad med data og skulle håndteres av en ferdig trent modell var vi nødt til å finne en rask nødløsning.

Vi håndterte dette ved å manuelt opprette en tibble som matchet dataen - etter den var prosessert av model_data - i serversiden av appen. Disse valgene ble tatt, delvis grunnet miskommunikasjon og oppdeling av oppgaver. Vi ser at dette muligvis kunne blitt gjort på en både bedre og enklere måte om vi skulle jobbet videre på prosjektet.

Billettprisen er en faktor i modellen, men noe vi ikke så på som særlig relevant for grensesnittet. Vi valgte

derfor å finne medianprisen for hver enkelt avreisehavn og billettklasse og på den måten sette en pris basert på "billetten man kjøper". Resultatet av dette er nå hardkodet inn i app.R



Vi har også noen globale variabler i prosjektet som vi ble nødt til å bruke grunnet mangel på tid og erfaring med utvikling av grensesnitt og shiny pakken i R.

Grensesnittet bruker ferdig trente modeller lagret med saveRDS og hentet med readRDS. Dette ønsket vi å gjøre slik at grensesnittet ikke trenger å vente på trening av modeller for hver gang man benytter det.

Resultat Når brukeren trykker på «Kjøp billett» knappen vil dataen transformeres og den lagrede modellen blir kjørt på dataen fra brukeren. Grensesnittet viser umiddelbart indikasjonen fra modellen om brukeren ville overlevd forliset eller ikke.