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 Atlanterhavet. 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.

Data wrangling

Data wrangling er prosessen med å forberede, rense og strukturere data for analyse. 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 main:

```
set.seed(121)
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

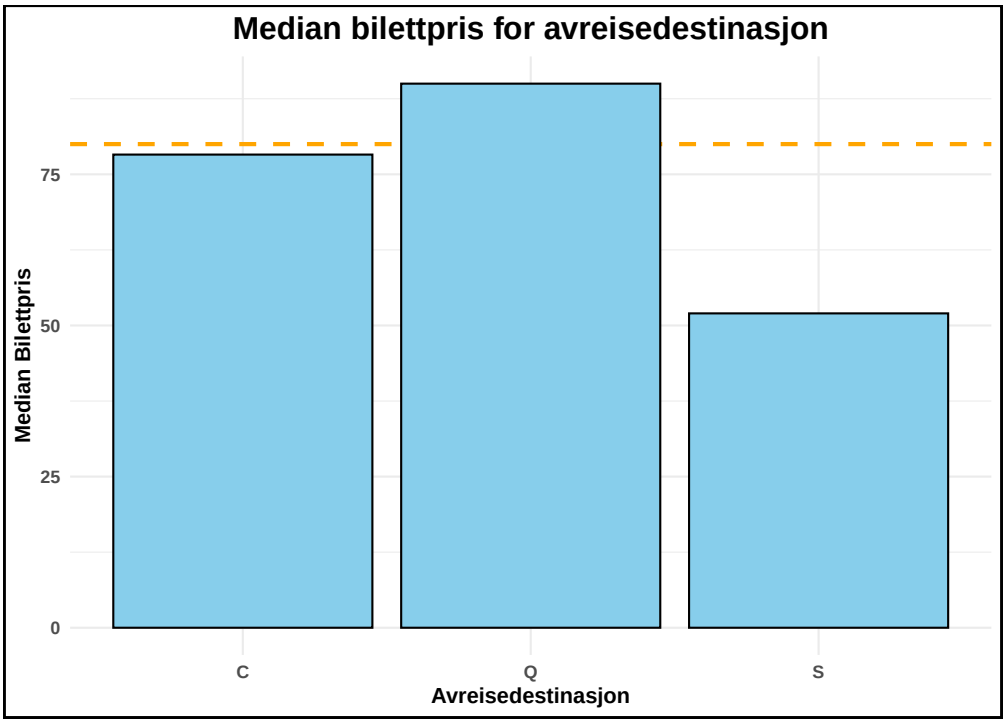

Figure 1: Pris for 1.klasse og avreisehavn.

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.

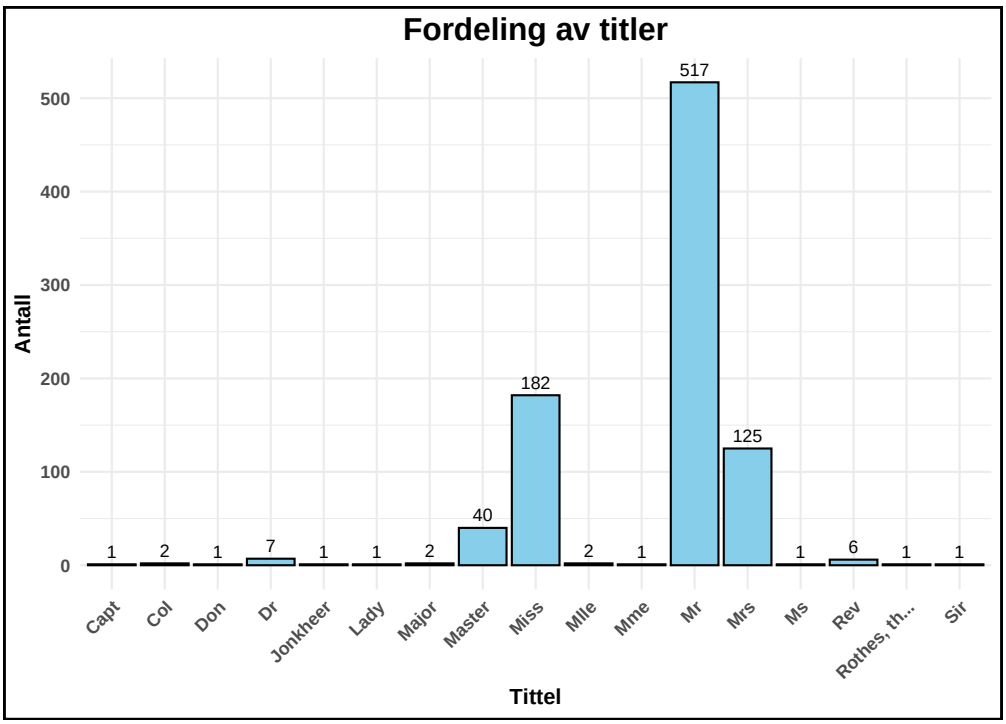

Figure 2: Antall passasjerer per tittel

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 hele data-wrangling prosessen sørger vi for at datasettet er renere, mer konsistente og at de manglende variablene er håndtert. Wranglingen kunne vært gjort på mange måter, og har en stor påvikning på det endelige resultatet. Om vi skulle gjenntatt prosjektet burde vi i større grad undersøkt datasettet i større grad før vi startet wranglingen. Til tross for dette føler vi at dataen er håndtert på en god måte.

Lag dummy-data og initial split

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

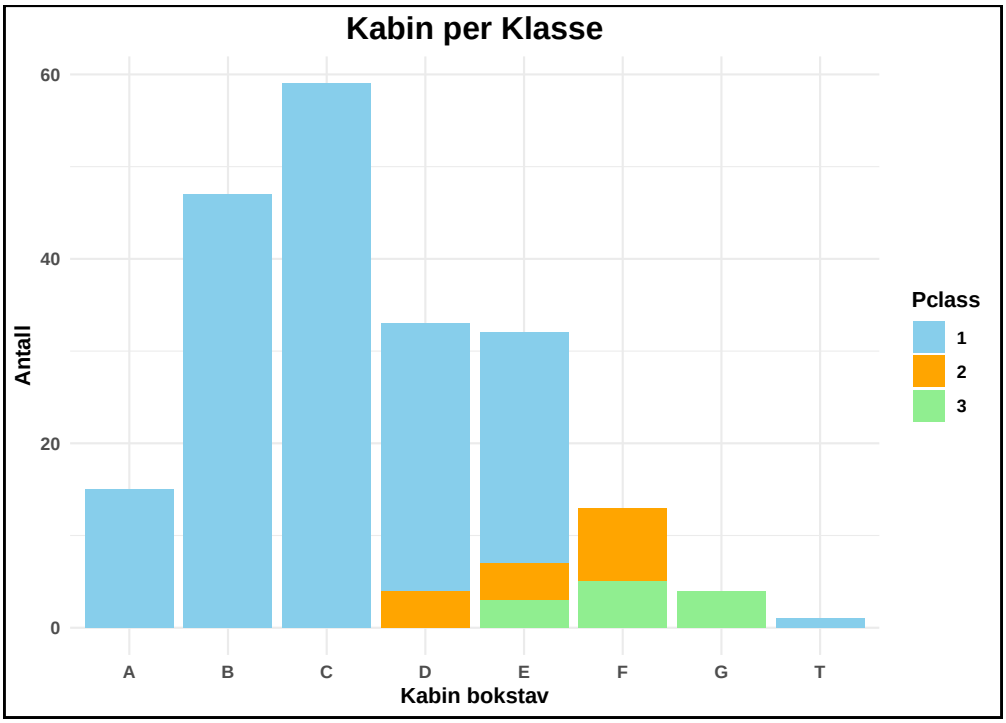

Figure 3: Kabin fordeling per klasse

Tren modeller

Vi fant ut at de mest relevant modellene for problemet var Lasso, Random Forest og Gradient Boosting Tree. Her implementerer vi dem, og finner de beste hyperparameterne. Vi laget en funksjon som tar inn modelnavn som parameter, og utifra navnet lager den en modelspesifikkasjon samt en grid med potensielle hyperparameter. Spennet av verdier i dette griddet har vi valgt basert på det som virker som konsensus på blant annet Stack Overflow. Videre bruker den tune_grid() fra tune-biblioteket i Tidyverse i kombinasjon med select_best() til å finne hvilke hyperparametere som sammen gir den beste treffsikkerheten.

```
# 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)
```

Deretter trener vi modellene 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)

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)
```

```r
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"),
  Treffsikkerhet = c(tuned_lso_acc, tuned_rf_acc, tuned_xgb_acc) #code in Norwegian because knitr::kabl
)

knitr::kable(accs, caption = "Nøyaktighet for ulike modeller")
```

Table 1: Nøyaktighet for ulike modeller

Model	Treffsikkerhet
LASSO	0.8491620
RANDOM FOREST	0.8547486
XGBOOST	0.8324022

Modellene har relativt lik treffsikkerhet. Ca. 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. Vi kommer tilbake til dette i siste avsnitt.

Hvilke faktorer påvirker resultatet mest

Videre ønsker vi å se på hvilken innvirkning hver enkelt faktor har på de ulike modellene. Gradient Boosting Tree fokuserte i stor grad på et fåtall variabler. Vi valge derfor å fremstille den i et eget plot. Vip-biblioteket ble brukt til å finne de viktigste faktorene.

Det vi ser er at de forskjellige modellene vektlegger forskjellige faktorer. Dette ser ut til å gjelde også når vi ikke bruker set.seed(). Dette stammer trolig fra at modellene gjennomfører treningen forskjellig, spesielt mellom de tre-baserte modellene og Lasso.

Et resultat som er spesielt interessant ifra disse figurene er hvor lite viktighet Lasso gir til faktorer som blant annet sex_male, title_mr. Disse faktorene forteller oss at passasjerene er menn, noe som er avdekket i studier til å være en av de viktigste faktorene. Likevell har to av modellene ingen fokus på dem. Vi konkluderer med at dette er fordi sex_female - som samtlige modeller har satt mye fokus på - forteller modellen nok om hvilket kjønn passasjeren har. Dette er spesielt aktuelt i Lasso, hvor en av karakteristikkene er å straffe bruk av koeffisienter eller veldig sammenhengende faktorer, men vi ser det også i Gradient Boosting Tree.

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.

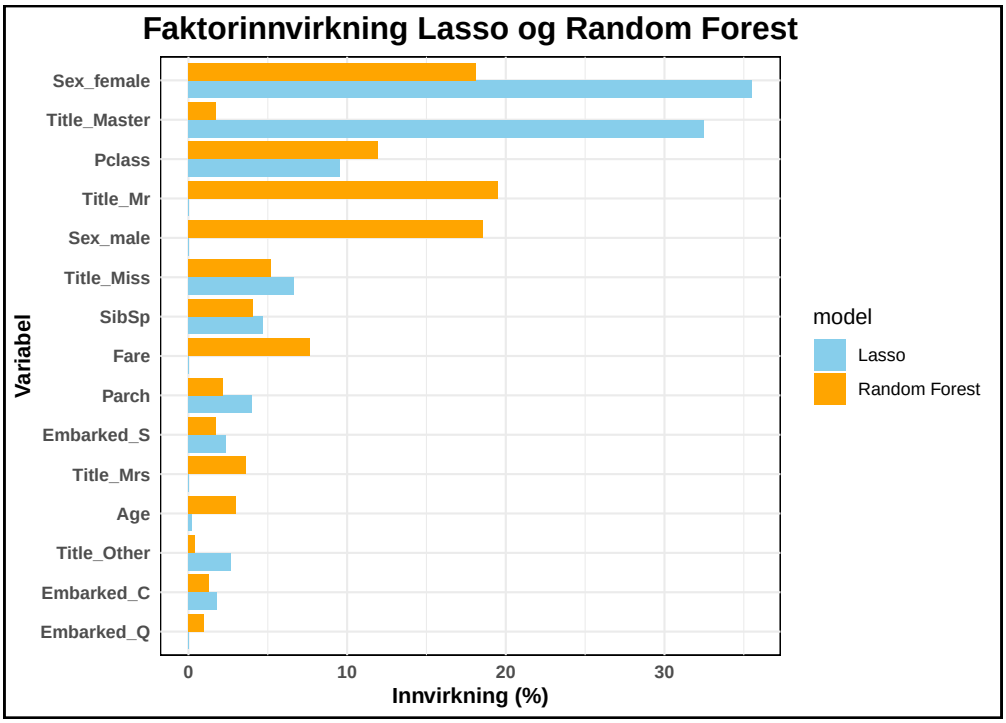

Figure 4: Faktorinnvirkning Lasso og Random Forrest

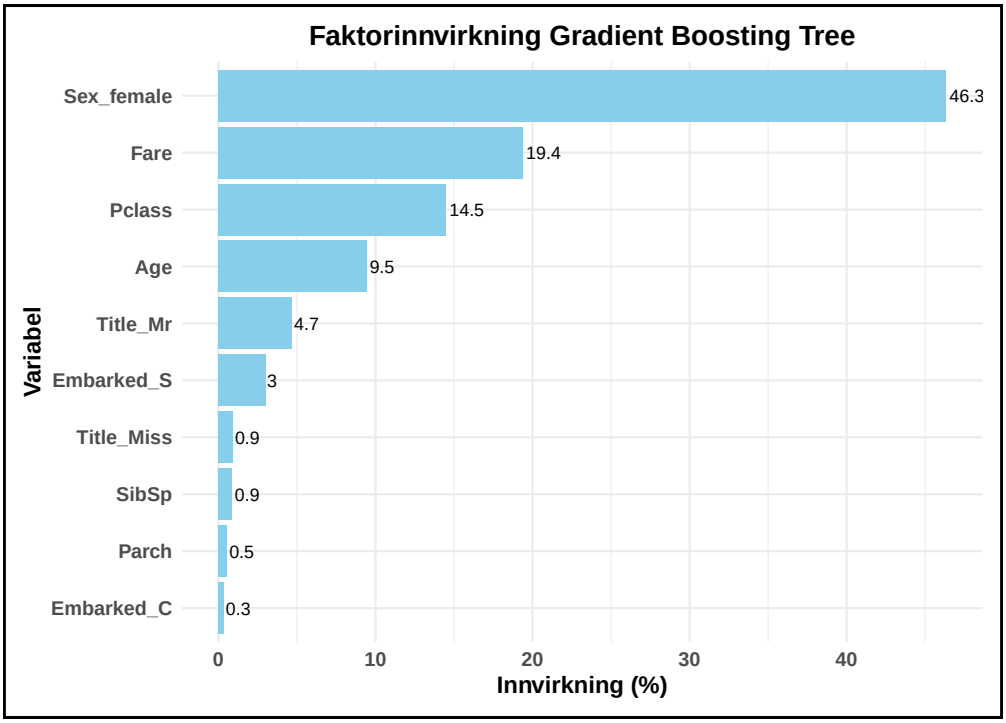

Figure 5: Faktorinnvirkning XGB

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 test-data. Vi så på muligheten for å splitte opp denne funksjonen, 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

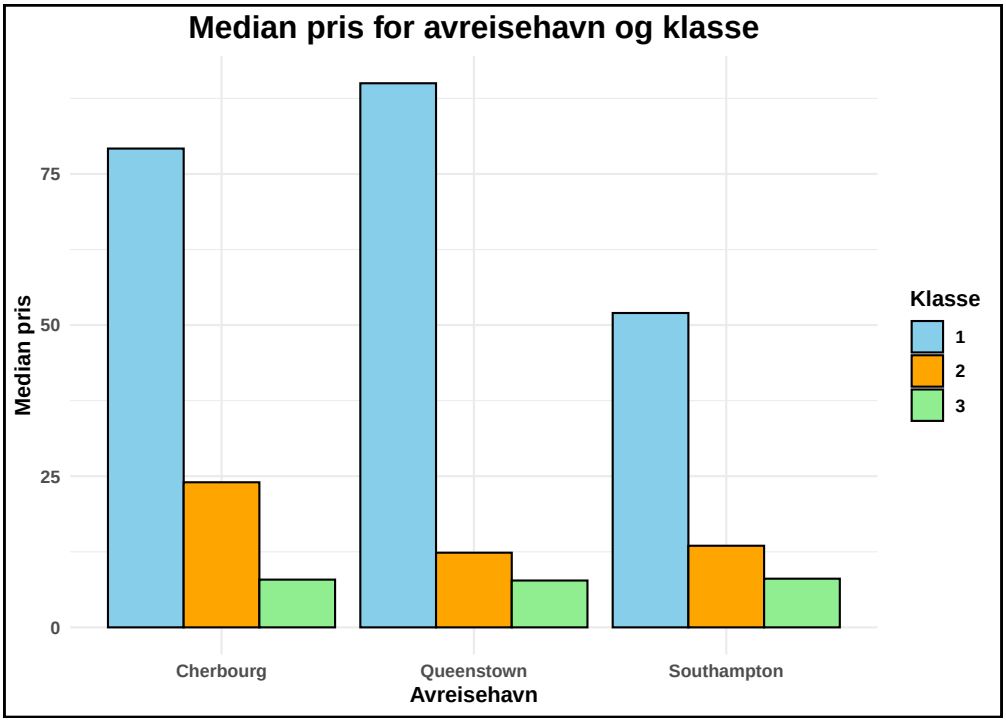

Figure 6: Pris for klasse og avreisedestinasjon.

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.

Oppsummering og refleksjon

I dette prosjektet har vi jobbet med å bygge en maskinlæringsmodell for å forutsi overlevelse på Titanic ved å analysere passasjerdata. Mye av arbeidet gikk med til å rense og forbedre dataen. Vi ser på det som svært sannsynlig at mye av modellenes treffsikkerhet ligger i dette steget.

Ettersom dette er en kjent oppgave, med mange tidligere gode løsninger var vi opptatt av å ikke gjøre noen research på forhånd på hvordan andre har løst problemet. Vi ønsket å løse oppgaven på best mulig måte uten hjelp eller inspirasjon fra andre løsninger. Etter å ha gjennomført vår beste løsning har vi dog sett at andre har klart å løse problemet med høyere treffsikkerhet.

En grunn til viktigheten av wrangling prosessen er størrelsen på datasettet. Med kun ca. 900 rader med data gjør det viktigheten av å håndtere manglende data på en god måte svært verdifullt. For hver variabel man tar ut av settet, eller setter unøyaktige verdier vil modellen ha et dårligere grunnlag for å gjøre nøyaktige prediksjoner.

Vi mener fremdeles at vi har gjort en god innsats med å forberede dataen til vår evne. Selv om vi nå - med vår nye læring og kunnskap - ser at det er forbedringer vi kunne gjort.

Inkluderingen av et grensesnitt er noe som gir oppgaven et preg av interaktivitet, som vi synes er et spennende tillegskomponent. Å kunne benytte modellen og gjøre den tilgjengelig for brukere og gi en prediksjon for overlevelse gir prosjektet et mer praktisk og engasjerende aspekt. Til tross for mangel på erfaring med utvikling av grensesnitt, samt noen raske nødløsninger, synes vi at det fungerer godt.

Til tross for forbedringspotensialet er vi fornøyd med hva vi har oppnådd. Vi har fått en bedre forståelse for maskinlæring, og kanskje særlig viktigheten av å håndtere dataen på en god måte.