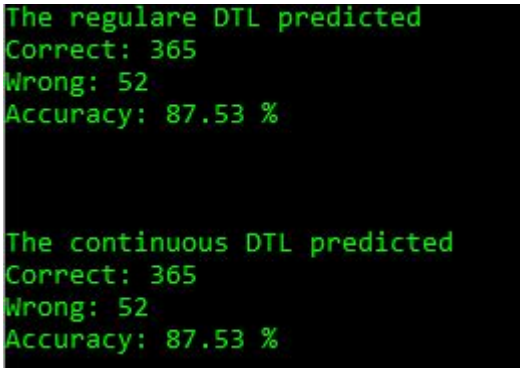
1 Decision Trees

a)

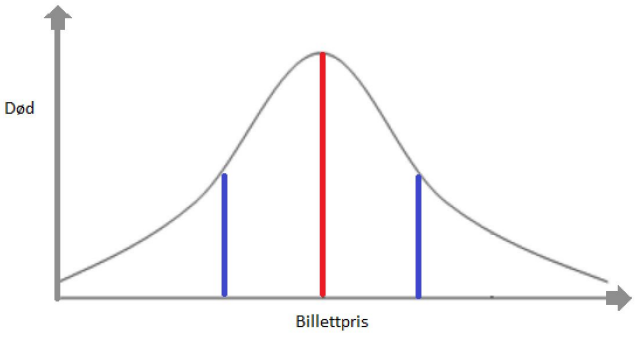
Kontinuerlig attributter er attributter som er uavbrutte. ‘Age’ er et godt eksempel på dette siden man kontinuerlig over tid blir eldre, uten noen plutselige hopp. Andre kontinuerlige attributter er SibSp, Parch og Fare. Disse er ikke like kontinuerlige med tanke på at man ikke kan ha 0.5 søsken. Grunnen til at vi velger dem kontinuerlige er at det blir dumt å ha løvnoder for 8 søsken f.eks. Da er det bedre å ha sjekker for > 4 søsken f.eks. I oppgaven valgte jeg kun å benytte meg av 'Pclass', 'Sex' og 'Embarked'. Jeg så bort ifra ‘Name’ siden den ikke har noe betydning for miljøet personen befant seg i, eller hvem er personen er. Det samme gjelder ‘Ticket’. ‘Cabin’ måte jeg også se bort ifra siden den mangler verdier.

c)



Decisions treet med kun kategoriske attributter, dvs. 'Pclass', 'Sex' og 'Embarked', endte opp med en nøyaktighet på 87.53%. Når jeg la til de kontinuerlig attributtene 'Fare', 'Parch' og 'SibSp' økte ikke nøyaktighet. Dette kan komme av at algoritmen jeg skrev kun kan splitt fordeling i to. Dersom prisen ikke følger en strengt økende/synkende korrelasjon til dødelighet, kan dette lede til en lite hensiktsmessig split (eksempelvis i figuren under med rød strek). I en normalfordeling f.eks. Ville det vært bedre å splitte fordeling i 3, som indikert i figuren under med de blå linjene.

En annen mulighet er selvfølgelig at billettpris og død ikke har noe særlig korrelasjon, og at vi dermed ender opp med å *overfitte*. Det vil da altså si at algoritmen prøver å finne et mønster, der det ikke er noe mønster å finne. Slik at nøyaktigheten blir bra når det gjelder trenings dataen, men slår nøytralt/negativt utover test dataen.



Dermed er en god måte å forbedre algoritmen på å sjekke om flere splits fører til en mer nøyaktig DTL.

For å løse overfitting problemet kan man f.eks. Benytte seg av *pre-pruning* eller *post-pruning*. Pre-pruning går ut på at man kun splitter på en attributt dersom man har nok prøve-data til å godkjenne at det er et mønster, slik at DTL-en ikke blir for dyp. En enklere løsning kunne også vært å bare sette en max_depth. I post-pruning så lar man DTL-en kjøre fullt ut, også sammenligner man errors med test dataen i forhold til om man splitter treet, eller lager en løvnode istedenfor.

2 Missing Values (Training)

En av de enkleste løsningene for å løse dette problemet er å velge verdien til å være det det er mest av. Slik at dersom attributten er kategorisk og eksemplene som brukes er f.eks. {Rød:20, Blå:10, Grønn:5} så velges verdien automatisk til å bli Rød. Fordelen er at denne metoden er svært enkel å implementere, ulempen derimot er at vi overser informasjonen som gis av de andre attributtene til rekken. Dersom attributten vi mangler er kontinuerlig kan vi istedenfor velge den manglende verdien til å være gjennomsnittet.

En bedre løsning er å finne den kolonnen med størst korrelasjon til kolonnen med den manglende verdien. Deretter setter man verdien fra den mest korrelerte kolonnen til den manglende verdien. F.eks. harLungekreft {sant, usant} og røyker {sant, usant} vil mest sannsynlig være sterkt korrelert, hvor harLungekreft = sant => røyker = sant. Fordelen her er at vi får utbytte av informasjonen til de andre attributtene i rekken, ulempen er at vi må anta at det faktisk finnes en attributt med en korrelasjon til den manglende verdien, som det f.eks. Med ‘Navn’ ikke finnes. Dersom attributten er kontinuerlig kan vi bruke lineær regresjon på de to mest korrelerte kolonnene, deretter bruke minste kvadraters metode til å finne den manglende verdien. For at dette skal gi mening må vi anta at sammenhengen mellom verdiene i de to attributtene må være lineære, og residualene må være uavhengige.

Pseudokode for kategoriske attributter metode 2 ved hjelp av Pandas:

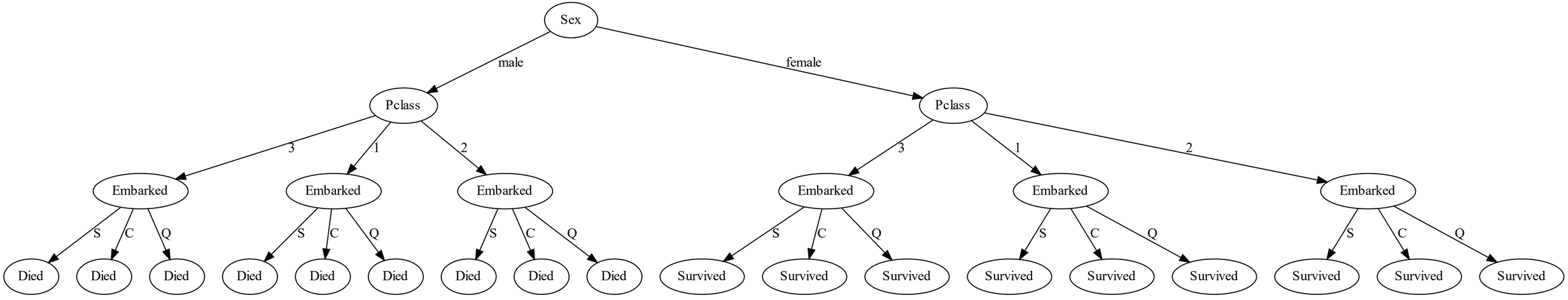
```
correlation_table = df.corr(method = ‘pearson’)

 #Finds the most correlated attribute to missing value attribute
corr_att = max_corr(correlation_table, missing_attribute)

#Value of most correlated attribute in row with missing val
corr_val = row_missing_val[corr_att]

#Most common value correlated to corr_val for attribute in row with missing val
most_common_val = df[df[corr_att] == corr_val].value_counts().idxmax()

Set the missing value to most_common_val
```


Sex
male
female
Pclass
Pclass
3
1
2
3
1
2
Embarked
Embarked
Embarked
Embarked
Embarked
Embarked
S
C
Q
Died
Survived


Sex
male
female
Fare
< 26.26875
> 26.26875
Pclass
3
1
2
Parch
< 0.5
> 0.5
SibSp
< 2.5
> 2.5
< 23.799999999999997
> 23.799999999999997
< 29.85625
> 29.85625
< 22.0
> 22.0
Embarked
S
C
Q
Survived
Died
< 1.5
> 1.5
< 7.0
> 7.0
< 2.5
> 2.5
< 0
> 0
< 2.0
> 2.0