Nedostajuće vrednosti

Jedan od najčešćih problema sa kojima se srećemo tokom „čišćenja“ podataka (u istraživačkoj analizi podataka ili na engleskom Exploratory Data Analysis - EDA) je rukovanje vrednostima koje nedostaju. Ono što prvo treba razumeti, a to je da **ne postoji** dobar način za rukovanje nedostajućim vrednostima. Postoje mnoga rešenja za umetanje vrednosti u zavisnosti od vrste problema - analiza vremenskih serija, regresija, klasifikacija itd, i teško je pružiti opšte rešenje. U nastavku su date najčešće korišćene metode sa njihovim prednostima i manama.

Umetanje ili uklanjanje podataka

Zašto podaci nedostaju.

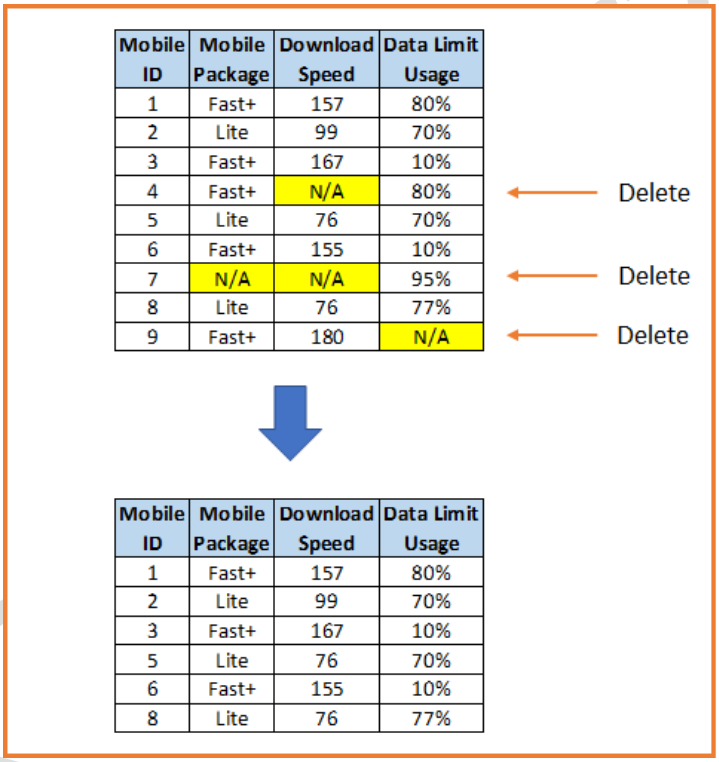
- **Missing at Random (MAR),** *Podaci nasumično nedostaju:* sklonost da podatak nedostaje nije povezana sa samim podacima koji nedostaju, **već je povezana sa nekim od posmatranih podataka**. Na primer, što je starija osoba, veće su šanse da nedostaju vrednosti u koloni sa prihodima. Još jedan primer, pretpostavimo da žene uglavnom ne žele da otkriju svoju starost. Ovde na vrednost koja nedostaje u promenljivoj za starost utiče rodna promenljiva.
- **Missing Completely at Random (MCAR):** Činjenica da nedostaje određena vrednost nema nikakve veze sa njenom hipotetičkom vrednošću i vrednostima drugih promenljivih. Ako se može osigurati dovoljno podataka, ove redove podataka koji sadrže nedostajuće vrednosti možemo bez problema izbrisati iz skupa podataka.
- **Missing not at Random (MNAR):** U ovom slučaju vrednost koja nedostaje zavisi od **hipotetičke vrednosti promenljive (ne od druge promenljive)**. Na primer, ljudi sa visokim platama uglavnom ne žele da otkrivaju svoje prihode u anketama i zato ta ćelija ostaje prazna u tabeli.

MAR i **MNAR** su tipovi nedostajućih vrednosti koje dovode do pristrasnosti (eng. ***bias***) prilikom izgradnje modela, ukoliko se obrišu redovi koji nedostaju. U većini slučajeva je teško razumeti kojoj vrsti nedostajuće vrednosti pripadaju, ali osnovni pristup u umetanju nedostajućih vrednosti je korišćenje informacija drugih promenljivih za kompletiranje tih vrednosti. Zato moramo biti veoma oprezni pre uklanjanja observacija. Imajte na umu da umetanje podataka ne mora nužno da dovede do boljih rezultata.

Brisanje podataka

Lista (listwise)

U ovom slučaju se uklanjaju svi podaci za observaciju ukoliko ona ima jednu ili više vrednosti koje nedostaju. Ovo je naročito praksa ako su podaci koji nedostaju ograničeni na mali broj observacija, tada se možemo odlučiti na korak da te observacije eliminišemo iz analize. Međutim, u većini slučajeva često je nepovoljno koristiti brisanje. To je zato što se pretpostavke MCAR-a (nastaju potpuno slučajno) obično retko kada javljaju u praksi. Kao rezultat, brisanje ovakvih observacija dovodi do pristrasnih parametra i procena.



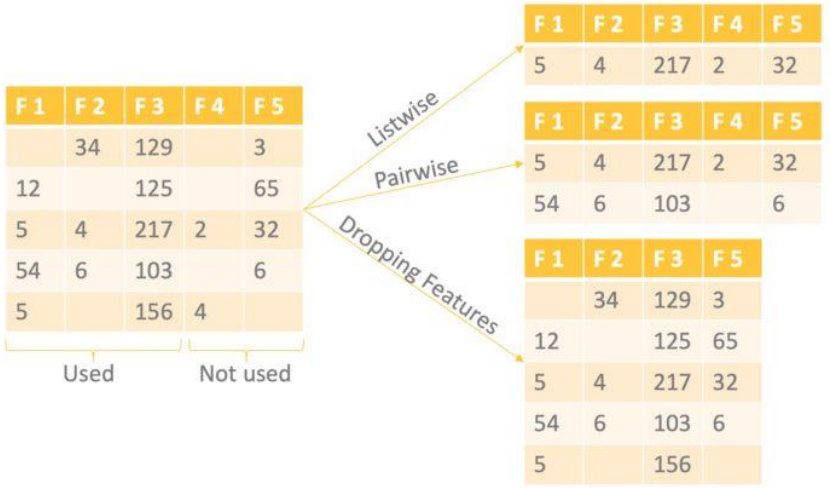
Izvor za sliku: BLOG

```
% In R
newdata <- na.omit(mydata)

# In Python
mydata.dropna(inplace=True)
```

Parovi (pairwise)

Brisanje u paru analizira sve slučajeve u kojima su prisutne promenljive od interesa i na taj način maksimizira sve podatke dostupne na osnovu analize. Prednost ove tehnike je što povećava snagu u vašoj analizi, ali ima i mnogo nedostataka. Osnovni nedostatak je pretpostavka da su podaci koji nedostaju MCAR tipa. Ako se podaci brišu u paru, na kraju ćemo dobiti različiti broj observacija koje doprinose različitim delovima našeg modela, što može otežati njihovu interpretaciju.



Prednosti brisanja redova:

- Potpuno uklanjanje podataka sa vrednostima koje nedostaju može da rezultira vrlo preciznim modelom (ne znači da će model biti robustan)
- Brisanje određenog reda ili kolone bez konkretnih informacija je bolje, jer nema veliki značaj za model

Mane brisanja redova:

- Gubitak informacija i podataka
- Loše radi ako je procenat nedostajućih vrednosti visok (recimo 30%), u odnosu na celi skup podataka

```
# Pairwise Deletion
ncovMatrix <- cov(mydata, use="pairwise.complete.obs")

# Listwise Deletion
ncovMatrix <- cov(mydata, use="complete.obs")
```

Brisanje promenljivih

- brišu se kolone, a ne redovi

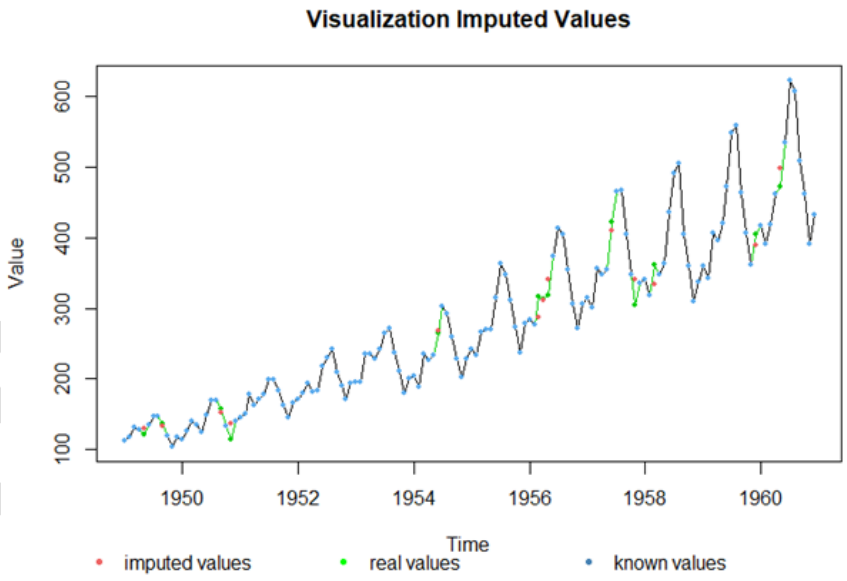
Na osnovu prethodnih iskustava, uvek je bolje zadržati podatke nego ih odbaciti. Ponekad se promenljive mogu obrisati ako podaci nedostaju za više od 60% observacija, ali samo ako je ta promenljiva beznačajna. Imajući to u vidu, umetanje podataka je uvek bolji izbor od brisanja promenljivih.

```
% in R
df <- subset(mydata, select = -c(x, z))
df <- mydata[-c(1,3:4)]

# In Python
del mydata.column_name
mydata.drop('column_name', axis=1, inplace=True)
```

Neke metode koje se odnose na vremenske serije (navode se informativno)

- Last Observation Carried Forward (LOCF) & Next Observation Carried Backward (NOCB)
- Linear Interpolation
- Seasonal Adjustment + Linear Interpolation



Izvor: Blog

Umetanje podataka

Mean, Median and Mode

Izračunavanje ukupne srednje vrednosti, medijane ili modusa je jedna od najčešće korišćenih metoda za umetanje (imputaciju) podataka koja ne koristi prednosti odnosa između promenljivih (druge promenljive nisu uzete u obzir). Veoma je brz pristup, ali ima i očigledne nedostatke. Jedan nedostatak je taj što umetanje srednje vrednosti smanjuje varijansu u skupu podataka.

Ova strategija se može primeniti na promenljivu koja ima numeričke podatke poput starosti osobe ili cene karte. Tada možemo da izračunamo srednju vrednost, medijanu ili modus i tom vrednošću zamenimo vrednosti koje nedostaju. U većini slučajeva ova metoda daje bolje rezultate u odnosu na brisanje redova i kolona. Ovaj metod se naziva i propuštanjem podataka (eng. *leaking data*) tokom treninga.

Drugi način kako se ova metoda može iskoristiti je aproksimacija odstupanja susednih vrednosti (devijacija). **Ovaj pristup bolje funkcioniše ako su podaci linearni.**

Prednosti:

- Ovo je bolji pristup kada je veličina podataka mala
- Može da spreči gubitak podataka koji rezultira uklanjanjem redova i kolona

Mane:

- Umetanje aproksimacija dodaje utiče na varijansu i bias
- Radi lošije u poređenju sa drugim metodama višestruke imputacije (eng. multiple imputation)

```
% In R
library(imputeTS)

na.mean(mydata, option = "mean")   # Mean Imputation
na.mean(mydata, option = "median") # Median Imputation
na.mean(mydata, option = "mode")   # Mode Imputation

# In Python
from sklearn.preprocessing import Imputer

values = mydata.values
imputer = Imputer(missing_values='NaN', strategy='mean')
transformed_values = imputer.fit_transform(values)

# strategy can be changed to "median" and “most_frequent”
```

Linearna regresija

Za početak se identifikuje nekoliko prediktora sa vrednostima koje nedostaju pomoću matrice korelacije. Najbolji prediktori se izaberu i koriste kao **nezavisne promenljive** u regresionoj jednačini (proceniti na osnovu domenskog znanja). Kao **zavisna promenljiva** koristi se promenljiva sa podacima koji nedostaju. Slučajevi sa potpunim podacima za prediktorske promenljive se koriste za generisanje regresione jednačine; jednačina se zatim koristi za predviđanje vrednosti koje nedostaju za nepotpune slučajeve.

U iterativnom procesu ubacuju se vrednosti za promenljivu koja nedostaje, a zatim se svi slučajevi koriste za predviđanje zavisne promenljive. Ovi koraci se ponavljaju dok se ne dođe do male razlike između predviđenih vrednosti od jednog do drugog koraka, tj. do konvergencije.

Ovaj metod „teorijski“ pruža dobre procene vrednosti koje nedostaju. Međutim, postoji nekoliko nedostataka ovog modela koji imaju tendenciju da budu bitniji od prednosti koje model pruža. **Prvo**, zato što su zamenjene vrednosti predviđene iz drugih promenljivih, one se zajedno lepo sklope i tako se standardna greška smanjuje. **Drugo**, mora se pretpostaviti da postoji linearni odnos između promenljivih korišćenih u regresionoj jednačini kada ih možda i nema.

KNN (K Nearest Neighbors)

Postoje i druge tehnike mašinskog učenja kao što su *XGBoost* i *Random Forest* koje se mogu koristiti za umetanje podataka, ali ovde pominjemo samo KNN, jer se on dosta koristi. U ovoj metodi, $\boldsymbol{k}$ suseda se bira na osnovu neke mere distance i njihov prosek se koristi kao procena umetanja. Metoda zahteva izbor *broja najbližih suseda* i *metriku za udaljenost*. KNN može da predvidi diskretne atribute (najčešća vrednost među k najbližih suseda) i kontinualne atribute (srednja vrednost među k najbližih suseda)

Metrika koja se koristi varira u zavisnosti od vrste podataka:

- **Kontinualni podaci**: Najčešće korišćeni metrički podaci za distancu za kontinualne podatke su Euklidska, Manhattan i Kosinusna distanca. (**DOMAĆI**)
- **Kategorijski podaci**: U ovom slučaju se obično koristi Hamming udaljenost. U obzir se uzimaju svi kategorijski atributi i tada se računa po **jedan** za svaku vrednost koja se razlikuje između dve observacije. Hamming-ova udaljenost je tada jednaka broju atributa za koje je vrednost bila različita.
- **Podaci imaju kontinualne i kategorijske promenljive**: U ovom slučaju se koristi Gower distanca. Blog1 Blog2

Jedna od najatraktivnijih karakteristika KNN algoritma je ta što je jednostavan za razumevanje i lak za primenu. Jedan od očiglednih nedostataka KNN algoritma je što analiza može dugo da potraje, jer se slične instance traže kroz čitav skup podataka. Dalje, tačnost KNN-a se može ozbiljno pogoršati u višedimenzionalnim podacima, jer postoji mala razlika između najbližeg i najudaljenijeg suseda.

Prednosti:

- Ne zahteva stvaranje prediktivnog modela za svaki atribut sa podacima koji nedostaju u skupu podataka
- Korelacija podataka se zanemaruje

Mane:

- To je dugotrajan proces, a to nekada može biti dosta bitno
- Izbor funkcija rastojanja može biti euklidski, manhattanski itd. što ne daje pouzdan rezultat

Umetanje vrednosti za kategorijske promenljive

- Imputacija *moda* je jedna od metoda, ali će definitivno dovesti do **bias**-a.
- Vrednosti koje nedostaju mogu se same po sebi tretirati kao posebna kategorija. Možemo kreirati drugu kategoriju za vrednosti koje nedostaju i koristiti ih kao novu kategoriju. Ovo je najjednostavnija i dosta često korišćena metoda. (npr. *I god, II god, III god, IV god, NA*)
- *Modeli predviđanja*: Ovde kreiramo prediktivni model za procenu vrednosti koje će zameniti podatke koji nedostaju. U ovom slučaju, svoj skup podataka delimo na dva skupa: jedan skup bez vrednosti koje nedostaju za promenljivu (*trening*) i drugi sa vrednostima koje nedostaju (*test*). Za predviđanje možemo koristiti metode poput logističke regresije i ANOVA.
- Višestruko umetanje

TOOLS:

Paket mice u R-u: websajt

Python: DataWig, official documenation

Kategorijske promenljive

Kategorijske promenljive se mogu podeliti na: **Nominalne** i **Ordinalne**.

Nominalne promenljive su promenljive koje imaju dve ili više kategorija koje nisu povezane bilo kakvim redosledom. Na primer, ako se pol klasifikuje u dve grupe, tj. muški i ženski, ova promenljiva se može smatrati nominalnom promenljivom.

Ordinalne promenljive, sa druge strane, imaju „nivoe“ ili kategorije sa određenim redosledom koji ih povezuje. Na primer, atribut sa tri različita nivoa: niskim, srednjim i visokim. Redosled je važan.

Što se tiče neke dalje podele, kategorijske promenljive takođe možemo kategorizovati kao **binarne**, tj. kategorijske promenljive sa samo dve kategorije. Nekada se pominje čak i tip „**ciklični**“ za kategorijske promenljive. Ciklične promenljive su prisutne u „ciklusima“, na primer, danima u nedelji: ponedeljak, utorak, sreda, četvrtak, petak, subota i nedelja. Drugi primer bi bili sati u danu ako ih smatramo kategorijama.

Posmatrajmo prvo jednu ordinalnu promenljivu „**ord**“. Moramo znati da računari ne razumeju tekstualne podatke i zato ove kategorije moramo nekako da pretvorimo u brojeve. Jednostavan način za to bi bio stvaranje rečnika koji ove vrednosti preslikava u brojeve koji uzimaju vrednosti od 0 do N-1, gde je N ukupan broj kategorija u datoj osobini.

1. Ordinal Encoding

Ordinalno kodiranje prvenstveno kodira ordinalne kategorije u numeričke vrednosti prema redosledu. Ordinalno kodiranje mapira svaku jedinstvenu vrednost kategorije u specifičnu numeričku vrednost na osnovu njenog reda ili ranga.
Ako posmatramo kolonu *obrazovanje* (*High School*, Associate, Master, PhD...) u nekom skupu podataka, tada možemo da definišemo redosled kategorija prilikom kreiranja ordinalnog kodera koristeći scikit-learn. Dakle, u primeru, redosled unutar kategorija određujemo kao listu u rastućem redosledu. Npr, nivoe obrazovanja ("Srednja škola" < "Bachelor" < "Master" < "Doktorat"), možemo kodirati kao 0, 1, 2 i 3, redom.

 # in Python

```
from sklearn.preprocessing import OrdinalEncoder
ordinal = OrdinalEncoder(categories=[['HS', 'AS', 'M.S','Ph.D']])
df['Education'] = ordinal.fit_transform(df[['Education']])
```

Nedostatak **Ordinal Encoding** pristupa je što daje redosled kategorijskim vrednostima, što može biti neodgovarajuće za neke algoritme mašinskog učenja kao što je linearna regresija, jer su previše osetljivi na vrednosti; u takvom slučaju, **One-hot encoding** pruža bolje rezultate.

Sa druge strane, ovakav tip transformacije možemo koristiti u mnogim modelima koji se zasnivaju na stablima, jer oni dobro rade sa kategorijskim promenljivama (ne zavise od njihovih vrednosti):

- Decision trees
- Random forest
- Extra Trees
- Boosted trees model
  - XGBoost
  - GBM
  - LightGBM

2. Label Encoding

Label Encoding će konvertovati svaku kategoriju u jedinstvenu numeričku vrednost. Ako se implementira sa scikit-learn bibliotekom, onda bi ovaj enkoder trebalo koristiti za kodiranje izlaznih vrednosti, tj. y, a ne ulaza X. Sličan je ordinalnom enkoderu, osim što su ovde numeričke vrednosti dodeljene automatski bez praćenja bilo kakvog prirodnog redosleda. Generalno, abecedni redosled kategorijskih vrednosti koristi se za određivanje koja numerička vrednost dolazi prva.

 # in Python

```
mapping = {
 "Boiling Hot": 0,
 "Cold": 1,
 "Freezing": 2,
"Hot": 3,
"Lava Hot": 4,
 "Warm": 5,
}

df.loc[:, "ord"] = df.ord_2.map(mapping)
```

ili pomoću `LabelEncoder`:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit([1, 2, 2, 6])
le.classes_
le.transform([1, 1, 2, 6])
```

```
le.inverse_transform([0, 0, 1, 2])
```

Kao što je već rečeno, ova vrsta kodiranja se ne može koristiti u linearnim modelima, SVM-u ili neuronskim mrežama, jer se očekuje da će se podaci normalizovati (ili standardizovati). **Tada, za ove modele možemo da binarizujemo podatke.**

	Ordinal-Label Encoding	Binarization		
Kategorija 1	0	0	0	0
Kategorija 2	1	0	0	1
Kategorija 3	2	0	1	0
Kategorija 4	3	0	1	1
Kategorija 5	4	1	0	0
Kategorija 6	5	1	0	1

Koliko promenljivih imamo sada?

Kako je najbolje čuvati ove podatke? Sparse format

```
example = np.array(
   [
      [0, 0, 1],
      [1, 0, 0],
      [1, 0, 1]
   ]
)
```

sparse format:

```
   (0, 2) 1
   (1, 0) 1
   (2, 0) 1
   (2, 2) 1
```

3. One-Hot Encoding

Iako sparse reprezentacija binarnih promenljivih uzima mnogo manje memorije od njegovog gustog (dense) predstavljanja, postoji još jedna transformacija za kategorijske promenljive koja uzima još manje memorije. Ta transformacija se zove **One-Hot Encoding**. Broj kategorija je

	Cat	Dog	Zebra
	1	0	0
	0	1	0
	0	0	1

zapravo jednak broju novih promenljivih, međutim, u praksi se uvek briše jedan kolona, jer se ona može odrediti na osnovu drugih.

U Python-u: sklearn.preprocessing.OneHotEncoder (pogledati argument *drop{'first', 'if_binary'}*)

Ove tri metode (četiri ako računamo i binarizaciju podataka) su najvažniji načini za rukovanje kategorijskim promenljivama. Međutim, postoji mnogo drugih različitih metoda koje se takođe mogu koristiti. Primer jedne takve metode je pretvaranje kategorijskih promenljivih u numeričke promenljive.

- **Na primer**, ako svaku vrednost kolone „ord" zamenimo sa ukupnim brojem pojavljivanja kategorije u koloni, tada smo je pretvorili u promenljivu koja je sada numerička.
- **Još jedan trik:** Kreiranje novih kategorijskih promenljivih od drugih kategorijskih promenljivih:

  Kategorija1_dog
  Kategorija1_cat
  Kategorija2_dog
  Kategorija3_mouse
  ...

Neki praktični saveti:

- **One-Hot Encoding** se primenjuje kada:
  - kategorijska promenljiva nije ordinalna
  - broj kategorijskih promenljivih je manji tako da se one-hot encoding može efikasno primeniti.
- **Ordinal Encoding** se primenjuje kada:
  - kategorijska promenljiva je ordinalna
  - broj kategorijskih promenljivih je veliki tako da će primena **one-hot encoding** pristupa voditi ka velikom korišćenju memorije.
- **Modeli zasnovani na stablima** mogu da rade i sa Ordinal i OH Encoding transformacijama.

Ordinal Encoding

- Ako kategorije imaju smislen redosled ili rangiranje.
- Svaku kategorija se konvertuje u jedinstveni ceo broj na osnovu njihovog redosleda. Na primer, „nisko", „srednje", „visoko" može da se kodira kao 0, 1, 2.
- Pošto stabla odlučivanja dele podatke na osnovu pragova prediktora, ako postoji redosled u prediktoru, OE čuva ovaj odnos, što potencijalno čini podele smislenim.

OHE

- Idealan za nominalne kategoričke varijable gde nema unutrašnjeg uređenja između kategorija.
- Kreira novu binarnu kolonu za svaki nivo kategorije (-1).
- Može da poveća dimenzionalnost podataka, što nije nužno problem za modele zasnovane na stablu jer su oni prilično dobri u rukovanju visokodimenzionalnim retkim podacima. Međutim, to može dovesti do složenijih stabala, jer algoritam razmatra podele na svakoj binarno kodiranoj osobini posebno, što možda nije uvek najefikasniji način za rukovanje kategorijalnim podacima.

Konačno ☺

- Za stabla odlučivanja i ansambl metode zasnovane na stablu (kao što su Random Forests i Gradient Boosting), ordinalno kodiranje može biti dobra polazna tačka ako su kategorijske promenljive ordinalne. Jednostavnije je i zadržava se prirodni redosled kategorija.
- Za nominalne podatke, možete početi sa OH kodiranjem, posebno ako broj kategorija nije prevelik, kako biste sprečili da drvo pravi proizvoljne podele na osnovu numeričkih vrednosti kodiranja.
- *Najbolji pristup se često svodi na eksperimentisanje i validaciju kroz unakrsnu proveru da bi se uporedile performanse modela sa svakim tipom kodiranja na vašem specifičnom skupu podataka.*

- **Neuronske mreže:**

Ordinal Encoding

- *Smanjena dimenzionalnost*: Dok ordinalno kodiranje smanjuje prostor karakteristika u poređenju sa OHE, ova prednost je manje kritična za neuronske mreže u poređenju sa nekim drugim algoritmima, a potencijal za uvođenje pogrešnih rednih odnosa obično nadmašuje prednosti smanjene dimenzionalnosti.

OHE

- Eliminiše redne pretpostavke: One-hot kodiranje tretira svaku kategoriju kao nezavisnu karakteristiku, izbegavajući uvođenje veštačkih rednih odnosa koji bi mogli da dovedu model u zabludu. Ovo je posebno važno za neuronske mreže, koje su sposobne da uče složene obrasce, ali takođe mogu naučiti nenamerne pristrasnosti (biases) ako se podrazumevaju redni odnosi tamo gde oni ne postoje.
- Iako OH kodiranje može značajno povećati dimenzionalnost ulaznih podataka (što može biti problem za modele osetljive na visoku dimenzionalnost), neuronske mreže generalno dobro rukuju retkim podacima visoke dimenzije.
- Neuronske mreže uče transformacije ulaznih podataka kroz svoje skrivene slojeve. One-hot kodiranje omogućava mreži da lako identifikuje prisustvo ili odsustvo kategorije kao jasan signal, što može biti jednostavnije za obučavanje mreže.

Konačno ☺

- Koristite One-Hot Encoding za nominalne kategorijske promenljive, posebno kada je broj kategorija relativno nizak.
- Razmisliti o korišćenju **embedding layers (ovo je informativno, ne treba da znate)** za efikasno rukovanje kategorijskim promenljivama visoke kardinalnosti.
- Ordinal Encoding se može uzeti u obzir za redne kategorijske promenljive. Međutim, za neuronske mreže, alternativni pristupi kao što je **embedding layers** su često efikasniji u hvatanju nijansi i nominalnih i rednih podataka.