INTELIGENTNI SISTEMI

as. ms Vladimir Jocović
as. ms Adrian Milaković

Stabla odlučivanja

Python

11

seaborn BIBLIOTEKA

Šta je seaborn biblioteka?

- **Seaborn** je besplatna softverska biblioteka napisana za programski jezik Pajton namenjena visoko-kvalitetnoj vizuelizaciji podataka.
- Predstavlja nadgradnju na veoma popularnu Python biblioteku namenjenu iscrtavanju i vizuelizaciji statičkih, dinamičkih i interaktivnih objekata – **matplotlib**. Biblioteka je usko povezana sa još jednom veoma popularnom Python bibliotekom za analizu i manipulaciju podacima – **pandas**.

Brzi tutorijali:

- seaborn introduction: https://seaborn.pydata.org/introduction.html
- seaborn tutorial: https://seaborn.pydata.org/tutorial.html

sklearn, matplotlib, pyplot

Kako preuzeti seaborn biblioteku?

- Seaborn nije standardna Python biblioteka. Moguće je preuzeti je na više različitih načina:
  - Iz PyCharm Python okruženja: File -> Settings -> Project: *Project Name* -> Python Interpreter, a zatim izabrati dugme **+** za preuzimanje novog paketa. Zatim je potrebno pretražiti **seaborn** i instalirati paket na dugme **Install Package**. Na dugme **-** moguće je ukloniti instalirani paket.
  - Iz PyCharm Python Terminala komandom **pip install seaborn**.
  - Pisanjem naredbe **import seaborn** u fajlu sa Pajton izvornim kodom, a zatim prelaskom mišem preko naredbe (*hover*) iz tooltip prozora izabrati opciju **Install package seaborn**.
  - Korišćenjem popularnog package manager-a **Anaconda**.

Zadatak – TITANIK

Dat je skup podataka koji predstavlja informacije o putnicima koji su putovali brodom Titanik. Potrebno je realizovati model koji će vršiti predikciju preživljavanja brodoloma od strane putnika. Poznati su sledeći atributi za svakog putnika:

- survived - da li je putnik preživeo (1) ili ne (0). Vrši se predikcija ovog atributa.
- pclass – klasa putnika (1, 2, 3)
- name – ime i prezime putnika, titula (string)
- sex – pol putnika (male, female)
- age – godine putnika (ceo broj)
- sibsp – broj braće/sestara, supruga
- parch – broj roditelja, dece
- ticket – karta putnika
- fare – cena karte putnika (realan broj)
- cabin – oznaka kabine putnika
- embarked – jednoslovna oznaka luke u kojoj se putnik ukrcao

Realizovati stablo odlučivanja koristeći model iz Python biblioteke sklearn.

Zadatak – Rešenje

```
# Uvoz pandas modula za manipulaciju nad podacima.
import pandas as pd

# Uvoz pyplot modula za vizuelizaciju podataka.
import matplotlib.pyplot as plt

# Uvoz seaborn modula za vizuelizaciju podataka.
import seaborn as sb

# Uvoz numpy modula za rad sa visedimenzionim nizovima.
import numpy as np

# Model stabla odlucivanja; vizuelizacija stabla;
# deljenje podataka na train/test set; enkodovanje kategorickih atributa;
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Uvoz modula za rad sa regularnim izrazima.
import re
```

Zadatak - Rešenje

```python
# 1. PROBLEM STATEMENT AND DATA READING

pd.set_option('display.max_columns', 20)
pd.set_option('display.max_rows', 20)

# Bez ogranicenja za sirinu ispisa.
pd.set_option('display.width', None)
data = pd.read_csv('datasets/titanic_train.csv')

# Dodavanje nove kolone na prvo mesto u DataFrame-u.
data.insert(loc=0, column='Passenger Id', value=np.arange(0, len(data), 1))

print(data.head())

# Ispis:
```

	Passenger Id	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
0	0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/02. 3101282	7.9250	NaN	S
3	3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Zadatak – Rešenje

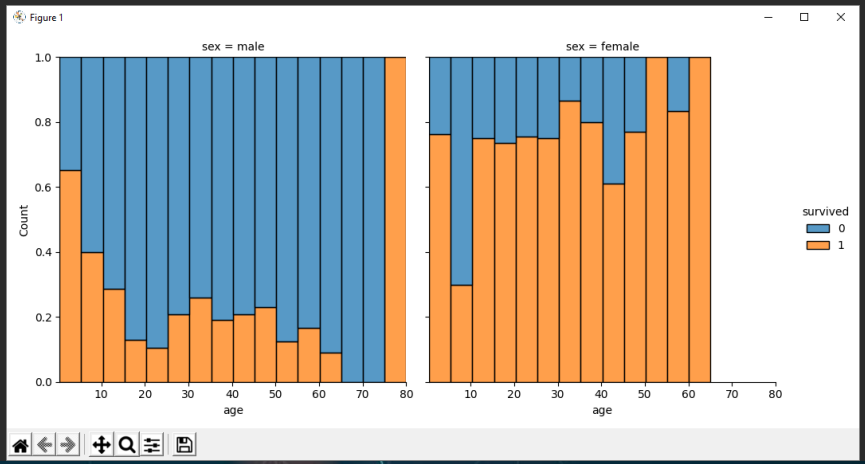
```
# 2. DATA ANALYSIS
# Bazicne informacije o DataFrame-u.
print(data.info())
# Ispis:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Passenger Id  891 non-null    int32
 1   survived      891 non-null    int64
 2   pclass        891 non-null    int64
 3   name          891 non-null    object
 4   sex           891 non-null    object
 5   age           714 non-null    float64
 6   sibsp         891 non-null    int64
 7   parch         891 non-null    int64
 8   ticket        891 non-null    object
 9   fare          891 non-null    float64
 10  cabin         204 non-null    object
 11  embarked      889 non-null    object
dtypes: float64(2), int32(1), int64(4), object(5)
memory usage: 80.2+ KB
# Statistika numerickih atributa.
print(data.describe())
# Ispis:
       Passenger Id    survived      pclass         age       sibsp       parch        fare
count    891.000000  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
mean     445.000000    0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std      257.353842    0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min        0.000000    0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      222.500000    0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%      445.000000    0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      667.500000    1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max      890.000000    1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

Zadatak - Rešenje

```
# Statistika kategorickih atributa.
print(data.describe(include=[object]))
# Ispis:
                        name   sex ticket        cabin embarked
count                    891   891    891          204      889
unique                   891     2    681          147        3
top     Artagaveytia, Mr. Ramon  male   1601  C23 C25 C27        S
freq                       1   577      7            4      644

# Prikaz top 5 kolona sa najvise podataka koji nedostaju.
total = data.isnull().sum().sort_values(ascending=False)
perc1 = data.isnull().sum() / data.isnull().count() * 100
perc2 = (round(perc1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, perc2], axis=1, keys=['Total', '%'])
print(missing_data.head(5))
# Ispis:
          Total     %
cabin       687  77.1
age         177  19.9
embarked      2   0.2
fare          0   0.0
ticket        0   0.0
```

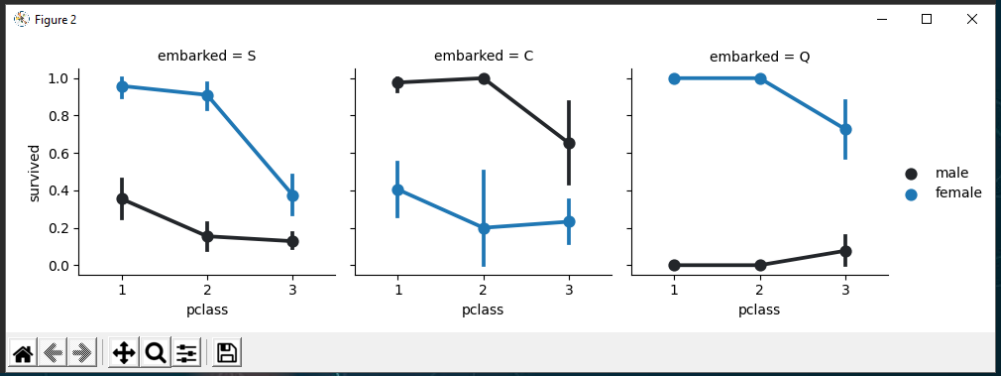
Zadatak – Rešenje

```
# Prikaz prezivelih po starosti i polu; godine grupisane u 16 grupa (bins).
# Parametar multiple moze biti: stack, layer, fill, dodge.
# Pol je atribut koji definitivno utice na ishod prezivljavanja.
sb.displot(data[data['age'].notna()], x='age', col='sex', hue='survived',
hue_order=[0, 1], multiple='fill', bins=16)
plt.show()
```



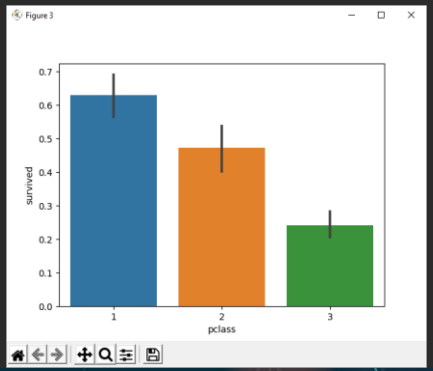
Zadatak - Rešenje

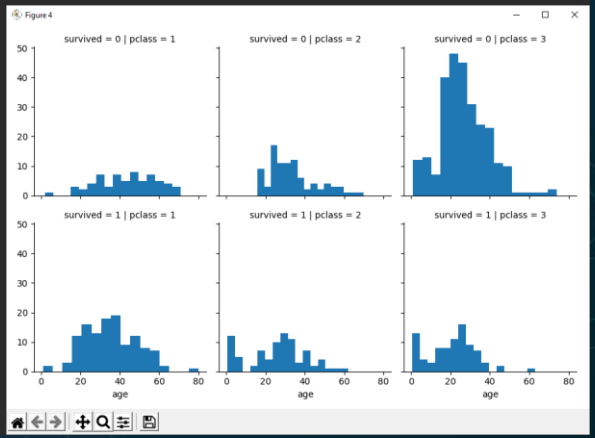
```
# Prikaz vise subplot-ova u okviru figure objekta. Subplot-ova ima koliko
# i razlicitih vrednosti u koloni col.
grid = sb.FacetGrid(data=data, col='embarked')

# Na x osi je pclass, na y osi survived, a prikazan je pol putnika (hue).
# Redosled prikaza kategorickih vrednosti nije bitan (None).
grid.map(sb.pointplot, 'pclass', 'survived', 'sex',
order=None, hue_order=None)
grid.add_legend()
plt.show()
```



Zadatak – Rešenje

```
# Prikaz prezivelih po klasi putnika.
# Putnik vise klase ima vise sanse da prezivi.
plt.figure()
sb.barplot(x='pclass', y='survived', data=data)

# Brojcani prikaz godina putnika po klasi i prezivljavanju.
grid = sb.FacetGrid(data=data, col='pclass', row='survived')
grid.map(plt.hist, 'age', bins=16)
grid.add_legend()
plt.show()
```



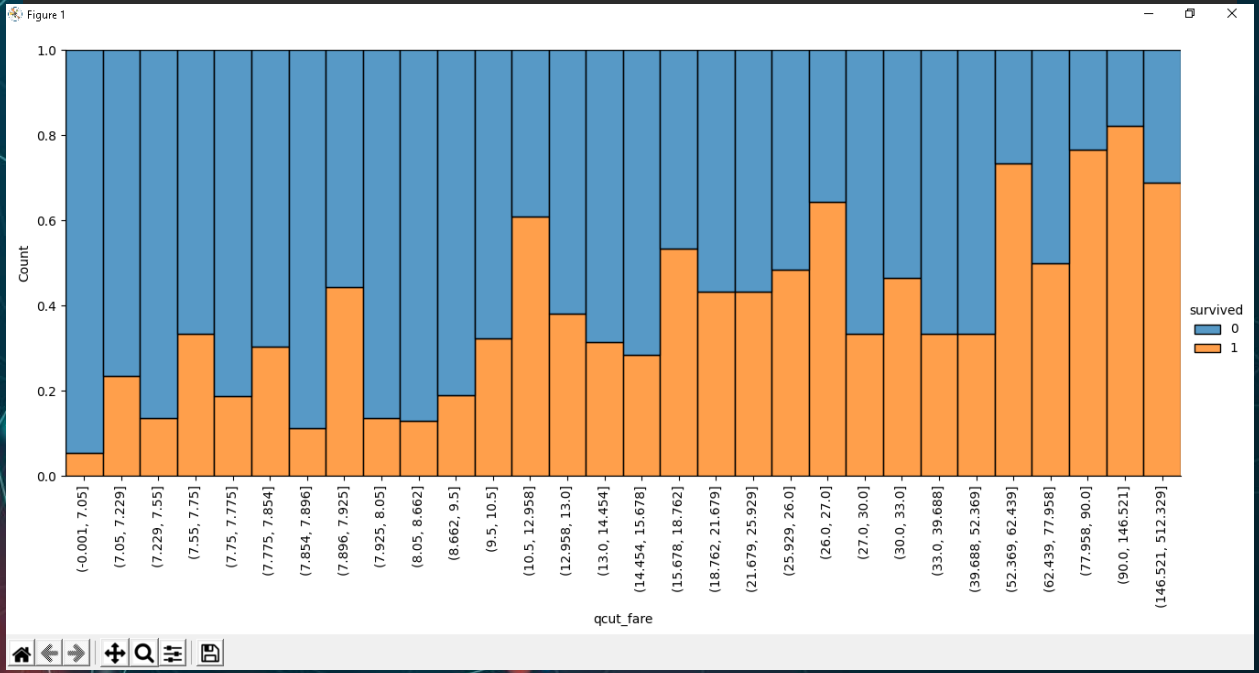


Zadatak - Rešenje

```
# Podela podataka iz kolone fare na 30 grupa podjednake velicine.
bins = 30
data['qcut_fare'] = pd.qcut(data['fare'], bins)

# Sortiranje DataFrame-a po koloni qcut_fare.
data = data.sort_values(by='qcut_fare')

# Konverzija vrednosti kolone qcut_fare (interval) u string.
data['qcut_fare'] = data['qcut_fare'].astype(str)

# Prikaz zavisnosti prezivljavanja od cene karte.
sb.displot(data, x='qcut_fare', hue='survived', hue_order=[0, 1],
multiple='fill', bins=bins)
plt.xticks(rotation=90)
plt.show()

data.drop(columns=['qcut_fare'], inplace=True)
```

Zadatak - Rešenje

```
# Jasno je da cena karte utice na ishod prezivljavanja.
# Poenta vizuelizacije je da nam ukaze kako grupisati vrednosti.
```

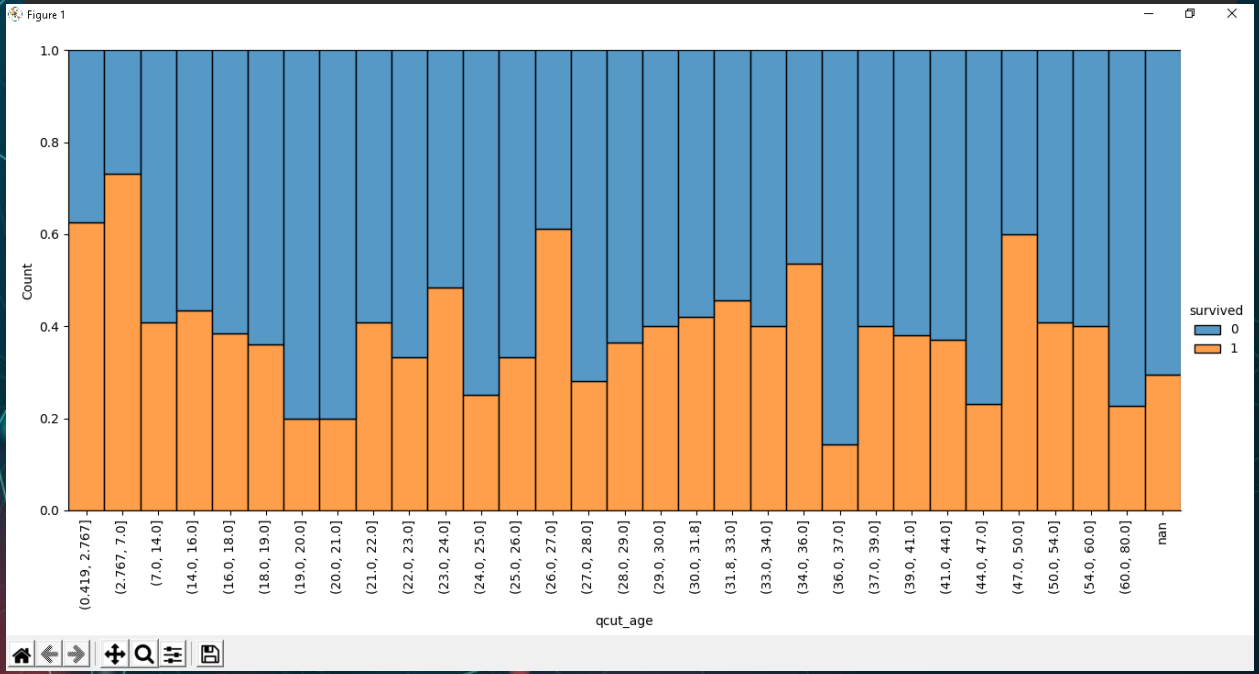


Zadatak – Rešenje

```
# Podela podataka iz kolone age na 30 grupa podjednake velicine.
bins = 30
data['qcut_age'] = pd.qcut(data['age'], bins)

# Sortiranje DataFrame-a po koloni age.
data = data.sort_values(by='qcut_age')

# Konverzija vrednosti kolone age u string.
data['qcut_age'] = data['qcut_age'].astype(str)

# Prikaz zavisnosti preživljavanja od starosti.
sb.displot(data, x='qcut_age', hue='survived', hue_order=[0, 1],
multiple='fill', bins=bins)
plt.xticks(rotation=90)
plt.show()

data.drop(columns=['qcut_age'], inplace=True)

# Sortirano kao i originalni DataFrame.
data = data.sort_values(by='Passenger Id')
```

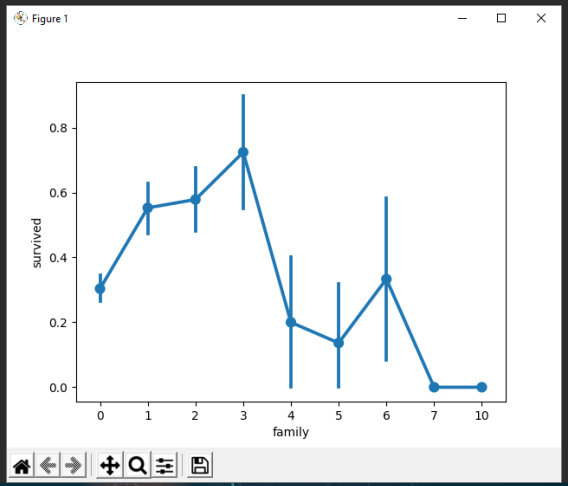
Zadatak – Rešenje

```
# Kao i cena karte i starost utice na ishod prezivljavanja.
# Poenta vizuelizacije je da nam ukaze kako grupisati vrednosti.
```



Zadatak – Rešenje

```
# Transformacija kolona sibsp i parch u jednu kolonu family.
# Vizuelizacija nam olaksava da identifikujemo grupe.
plt.figure()
data['family'] = data['sibsp'] + data['parch']
data = data.drop(columns=['sibsp', 'parch'])
sb.pointplot(x='family', y='survived', data=data)
plt.show()
```

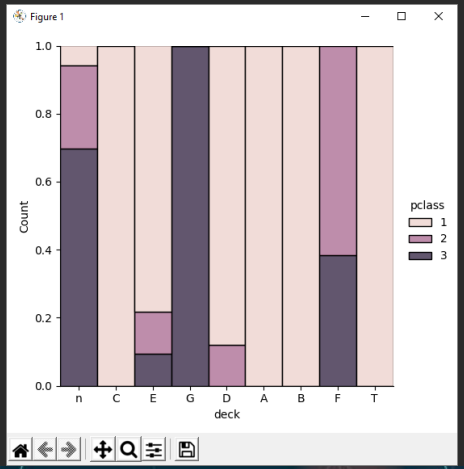


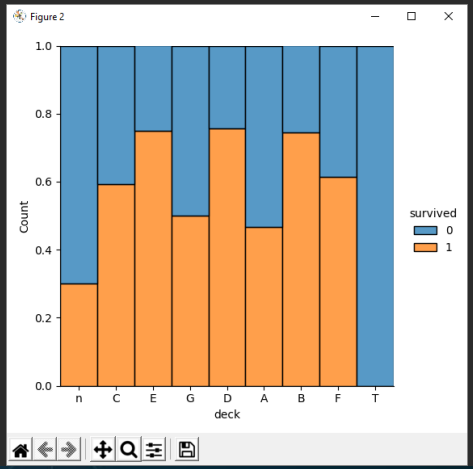
Zadatak - Rešenje

```
# 3. DATA CLEANSING

# Kolona cabin ima veoma mnogo vrednosti koje nedostaju (NaN).
# Iz tog razloga nije moguce popuniti nedostajuce vrednosti
# rezultatom neke funkcije (mean, mode, itd.). Medjutim, ne bi bilo dobro
# odbaciti tu kolonu u potpunosti, jer prvo slovo krije ime palube.
cabin = set(map(lambda val: str(val)[0], data['cabin'].unique()))
print(sorted(cabin))

# Ispis:
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'T', 'n']

# Uvodjenje nove kolone deck i izbacivanje kolone cabin.
data['deck'] = data['cabin'].apply(lambda val: str(val)[0])
data = data.drop(columns=['cabin'])
```

Zadatak - Rešenje

```
# Prikaz zavisnosti klase od palube.
# A, B, C i T su 1. klasa, D i E su mesane, F i G su uglavnom 3. klasa.
# D i E imaju slicnu stopu prezivljavanja.
sb.displot(data, x='deck', hue='pclass', multiple='fill')
sb.displot(data, x='deck', hue='survived', multiple='fill')
plt.show()
```





Zadatak - Rešenje

```
# Pored kolone cabin, kolona age ima veci broj vrednosti koje nedostaju.
# Nedostajuce vrednosti bice popunjene vrednostima dobijenim
# grupisanjem po grupi g.
g = ['pclass', 'sex', 'deck', 'family']
data['age'] = data.groupby(by=g)['age'].apply(lambda c: c.fillna(c.mean()))

# Ako su i dalje neke vrednosti za age NaN (grupa po g sa svim NaN za age),
# vrednost za popunu je srednja vrednost kolone age.
data['age'] = data['age'].fillna(data['age'].mean())
```

Zadatak - Rešenje

```
# Kolona embarked ima samo 2 nedostajuce vrednosti
# pa ce biti popunjena najcesce prisutnom vrednoscu u koloni embarked.
data['embarked'] = data['embarked'].fillna(data['embarked'].mode()[0])

print(data.info())

# Ispis:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Passenger Id  891 non-null    int32
 1   survived      891 non-null    int64
 2   pclass        891 non-null    int64
 3   name          891 non-null    object
 4   sex           891 non-null    object
 5   age           891 non-null    float64
 6   ticket        891 non-null    object
 7   fare          891 non-null    float64
 8   embarked      891 non-null    object
 9   family        891 non-null    int64
 10  deck          891 non-null    object
dtypes: float64(2), int32(1), int64(3), object(5)
memory usage: 73.2+ KB
```

Zadatak - Rešenje

```
# 4. FEATURE ENGINEERING

# Izbacivanje nebitne vestacke kolone, koja ne doprinosi klasifikaciji.
data = data.drop(columns=['Passenger Id'])

# Izbacivanje kolone ticket,
# imajuci u vidu da sadrzi 681/891 jedinstvenih vrednosti.
data = data.drop(columns=['ticket'])

# Konverzija kolone age u int.
data['age'] = data['age'].astype(int)

# Grupisanje vrednosti kolone deck na osnovu grafika.
data['deck'] = data['deck'].replace(['A', 'B', 'C', 'T'], '1st')
data['deck'] = data['deck'].replace(['F', 'G'], '2nd_3rd')
data['deck'] = data['deck'].replace(['D', 'E'], 'mixed')
data['deck'] = data['deck'].replace(['n'], 'N')
```

Zadatak - Rešenje

```
# Grupisanje vrednosti kolone family na osnovu grafika.
ind_alone = (data['family'] == 0)
ind_small = ((data['family'] >= 1) & (data['family'] <= 3))
ind_med = ((data['family'] >= 4) & (data['family'] <= 6))
ind_big = (data['family'] > 6)

data.loc[ind_alone, 'family'] = 'alone'
data.loc[ind_small, 'family'] = 'small'
data.loc[ind_med, 'family'] = 'medium'
data.loc[ind_big, 'family'] = 'big'
```

Zadatak – Rešenje

```
# Ekstrahovanje titule na osnovu imena.
# Titula se nalazi iza prezimena i zareza i zavrsava se tackom.
# Koristeci regularne izraze, pretrazuju se alfanumericki karakteri,
# koje prati tacka. Na osnovu kolone name gradi se kolona title.
fun_tit = lambda name: re.search(r'(\w+)\.', name).group(1)
data['title'] = data['name'].apply(fun_tit)
titles = set(data['title'].unique())

print(sorted(titles))
# Ispis:
['Capt', 'Col', 'Countess', 'Don', 'Dr', 'Jonkheer', 'Lady', 'Major',
'Master', 'Miss', 'Mlle', 'Mme', 'Mr', 'Mrs', 'Ms', 'Rev', 'Sir']

# Sve titule se grupisu u 3 grupe:
# Mr (obicni muskarci), Mrs (obicne zene) i Rare (retke titule).
female_common = {'Ms', 'Mrs', 'Mme', 'Miss', 'Mlle'}
titles = set.difference(titles, female_common)
titles.remove('Mr')
data.loc[data['title'].isin(titles), 'title'] = 'Rare'
data.loc[data['title'].isin(female_common), 'title'] = 'Mrs'
data = data.drop(columns=['name'])
```

Zadatak - Rešenje

```
# Neophodno je transformisati kategoricke kolone: embarked, title,
# deck i family. To se moze uraditi na nekoliko nacina:

# 1) LabelEncoder - kodira kategoricke vrednosti leksikografski rastuce
# do ukupnog broja razlicitih vrednosti kategoricke kolone (0, n-1).
# Nije pogodno ako ne postoji redosled vrednosti (ordering).
# Moze se koristiti u slucaju dve vrednosti gde ne postoji redosled.
le = LabelEncoder()
data['sex'] = le.fit_transform(data['sex'])

# 2) OneHotEncoder - kodira kategoricke vrednosti u onoliko kolona
# koliko kodirajuca kolona ima razlicitih vrednosti.
# Za svaki uzorak samo jedna kolona ima vrednost 1, ostale vrednost 0.

# sparse=False - ne zelimo retku matricu
ohe = OneHotEncoder(dtype=int, sparse=False)
# fit_transform zahteva promenu oblika
embarked = ohe.fit_transform(data['embarked'].to_numpy().reshape(-1, 1))
data.drop(columns=['embarked'], inplace=True)
data = data.join(pd.DataFrame(data=embarked,
columns=ohe.get_feature_names(['embarked'])))
```

Zadatak - Rešenje

```
# 3) get_dummies - radi ekvivalentnu stvar kao i OneHotEncoder.
# Razlika je sto get_dummies zahteva da podaci budu tipa string,
# dok OneHotEncoder podrzava i string i celobrojni tip.

# prefix - imena novih kolona dobijaju se spajanjem prefix-a i vrednosti.
titles = pd.get_dummies(data['title'], prefix='tit')
data.drop(columns=['title'], inplace=True)
data = data.join(pd.DataFrame(data=titles))

titles = pd.get_dummies(data['deck'], prefix='deck')
data.drop(columns=['deck'], inplace=True)
data = data.join(pd.DataFrame(data=titles))

titles = pd.get_dummies(data['family'], prefix='fam')
data.drop(columns=['family'], inplace=True)
data = data.join(pd.DataFrame(data=titles))
```

Zadatak - Rešenje

```
# Grupisanje kolone age prema grafiku.
data.loc[(data['age'] <= 5), 'age'] = 0
data.loc[(data['age'] > 7) & (data['age'] <= 19), 'age'] = 1
data.loc[(data['age'] > 19) & (data['age'] <= 21), 'age'] = 2
data.loc[(data['age'] > 21) & (data['age'] <= 28), 'age'] = 3
data.loc[(data['age'] > 28) & (data['age'] <= 36), 'age'] = 4
data.loc[(data['age'] > 36) & (data['age'] <= 47), 'age'] = 5
data.loc[(data['age'] > 47) & (data['age'] <= 60), 'age'] = 6
data.loc[(data['age'] > 60), 'age'] = 7
```

Zadatak - Rešenje

```
# Grupisanje kolone fare prema grafiku.
data.loc[(data['fare'] > 0) & (data['fare'] <= 7.9), 'fare'] = 0
data.loc[(data['fare'] > 7.9) & (data['fare'] <= 10.5), 'fare'] = 1
data.loc[(data['fare'] > 10.5) & (data['fare'] <= 13), 'fare'] = 2
data.loc[(data['fare'] > 13) & (data['fare'] <= 15.7), 'fare'] = 3
data.loc[(data['fare'] > 15.7) & (data['fare'] <= 26), 'fare'] = 4
data.loc[(data['fare'] > 26) & (data['fare'] <= 27), 'fare'] = 5
data.loc[(data['fare'] > 27) & (data['fare'] <= 52.5), 'fare'] = 6
data.loc[(data['fare'] > 52.5) & (data['fare'] <= 62.5), 'fare'] = 7
data.loc[(data['fare'] > 62.5) & (data['fare'] <= 78), 'fare'] = 8
data.loc[(data['fare'] > 78), 'fare'] = 9

# Konverzija kolone fare u int.
data['fare'] = data['fare'].astype(int)
```

Zadatak – Rešenje

```python
# Pregled izmenjenog DataFrame-a.
print(data.head(10))

# Ispis:
```

	survived	pclass	sex	age	fare	embarked_C	embarked_Q	embarked_S	tit_Mr	tit_Mrs	tit_Rare	deck_1st	deck_2nd_3rd	deck_N	deck_mixed	fam_alone	fam_big	fam_medium	fam_small
0	0	3	1	3	0	0	0	1	1	0	0	0	0	1	0	0	0	0	1
1	1	1	0	5	8	1	0	0	0	1	0	1	0	0	0	0	0	0	1
2	1	3	0	3	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0
3	1	1	0	4	7	0	0	1	0	1	0	1	0	0	0	0	0	0	1
4	0	3	1	4	1	0	0	1	1	0	0	0	0	1	0	1	0	0	0
5	0	3	1	4	1	0	1	0	1	0	0	0	0	1	0	1	0	0	0
6	0	1	1	6	6	0	0	1	1	0	0	0	0	0	1	1	0	0	0
7	0	3	1	0	4	0	0	1	0	0	1	0	0	1	0	0	0	1	0
8	1	3	0	3	2	0	0	1	0	1	0	0	0	1	0	0	0	0	1
9	1	2	0	1	6	1	0	0	0	1	0	0	0	1	0	0	0	0	1

Zadatak - Rešenje

```
# 5. MODEL TRAINING
# Model stabla odlucivanja; koristi se kriterijum entropije
# za pronalazenje atributa za podelu cvora (postoji i gini kriterijum).
dtc_model = DecisionTreeClassifier(criterion='entropy')

# X - atributi na osnovu kojih se vrsi klasifikacija;
# y - labele (tacni odgovori)
X = data.drop(columns=['survived'])
y = data['survived']

# train_test_split deli podatke na train i test skupove.
# random_state i shuffle sluze za reproducibilne slucajeve.
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=234, shuffle=True)

# Pozivom ove metode vrsi se obucavanje modela.
dtc_model.fit(X_train, y_train)
```

Zadatak - Rešenje

```
# Dobijanje rezultata predikcije za podatke koji nisu prezentovani modelu.
labels_predicted = dtc_model.predict(X_test)

# Generisanje rezultujuceg DataFrame-a konkatenacijom test atributa,
# test ocekivanih odgovora i predikcija.
ser_pred = pd.Series(data=labels_predicted, name='Predicted',
index=X_test.index)
res_df = pd.concat([X_test, y_test, ser_pred], axis=1)

print(res_df.head(10))

# Ispis:
```

	pclass	sex	age	fare	embarked_C	embarked_Q	embarked_S	tit_Mr	tit_Mrs	tit_Rare	deck_1st	deck_2nd_3rd	deck_N	deck_mixed	fam_alone	fam_big	fam_medium	fam_small	survived	Predicted
725	3	1	2	1	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0
566	3	1	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0
661	3	1	5	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0
709	3	1	3	3	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1
251	3	0	4	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
195	1	0	6	9	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0
325	1	0	4	9	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1
611	3	1	4	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0
303	2	0	5	2	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	1
616	3	1	4	3	0	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0

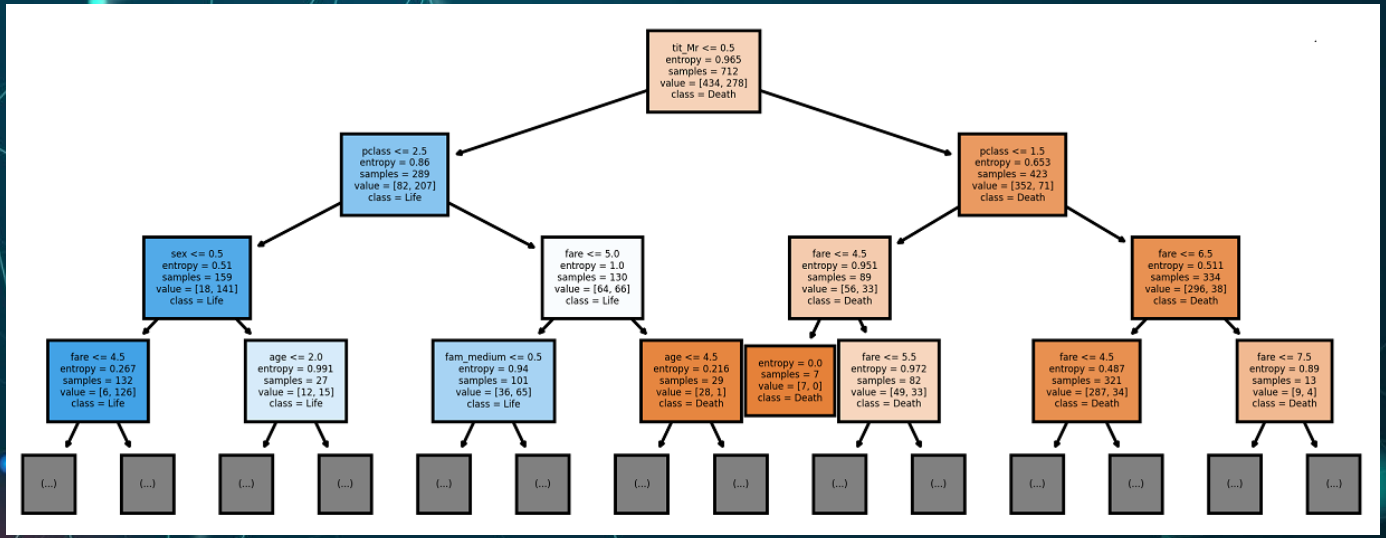
Zadatak - Rešenje

```
# Racunanje tacnosti modela nad test podacima:
print(f'Model score: {dtc_model.score(X_test, y_test):0.3f}')

# Ispis:
Model score: 0.838

# CrossValidation predstavlja tehniku validacije modela,
# koja radi po sledecem principu:
# podaci se podele na cv podskupova podataka, pri cemu se (cv-1) skupova
# koristi za treniranje, a 1 skup podataka za validaciju.
# Postupak se ponavlja cv puta tako da svaki skup po jednom ucestvuje
# kao skup za testiranje, kao takav.
cv_res = cross_validate(dtc_model, X, y, cv=10)
print(sorted(cv_res['test_score']))
print(f'CV score: {cv_res["test_score"].mean():0.3f}')

# Ispis:
CV score: 0.833
```

Zadatak - Rešenje

```
# Vizuelizacija stabla:
# Kreiranje figure i axes objekta. Grid 1 x 1.
# Zadaju se dimenzije figure u incima i dpi (dots per inch).
fig, axes = plt.subplots(1, 1, figsize=(8, 3), dpi=400)

# Za isrtavanje stabla zadaje se model stabla.
# Moguce je zadati maksimalnu dubinu stabla, imena atributa, imena labela,
# velicinu fonta, farbanje cvorova stabla na osnovu klase pripadnosti itd.
tree.plot_tree(decision_tree=dtc_model, max_depth=3,
feature_names=X.columns, class_names=['Death', 'Life'],
fontsize=3, filled=True)

# Cuvanje stabla kao slike.
fig.savefig('tree.png')

# Ili vizuelizacija u okviru prozora.
# plt.show()
```

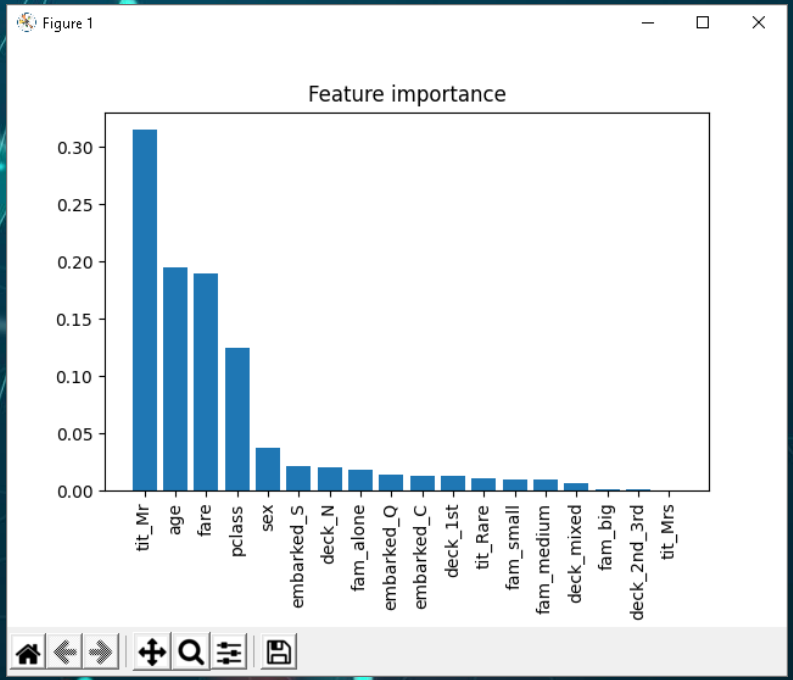
Zadatak – Rešenje



Zadatak - Rešenje

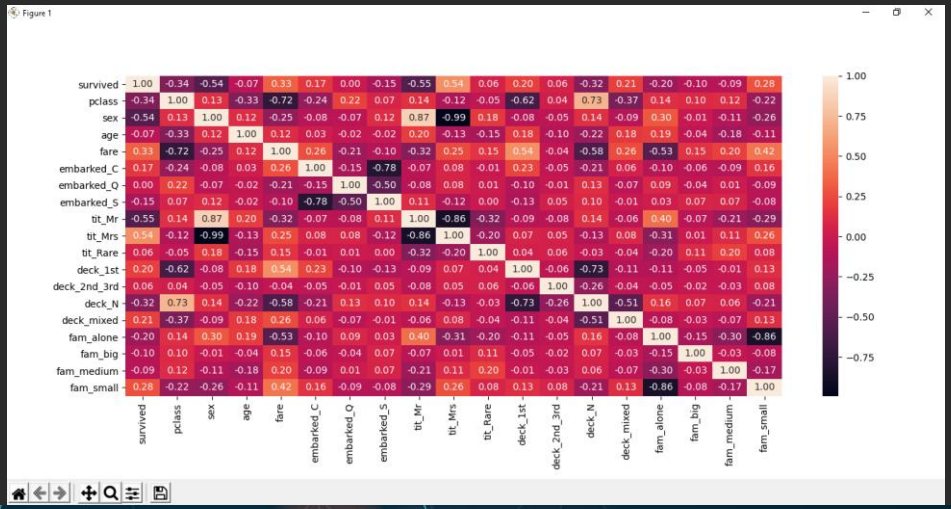
```
# Vizuelizacija bitnih atributa:
plt.figure()

# Kreiranje recnika - kljuc je naziv atributa, vrednost je njegov znacaj.
feat = dict(zip(X.columns, dtc_model.feature_importances_))

# Kreiranje sortirane liste parova kljuc-vrednost (torki)
# prema vrednosti opadajuce.
items = sorted(feat.items(), key=lambda it: it[1], reverse=True)

# Raspakivanje sortirane liste na sortirane kljuceve i sortirane vrednosti.
keys, values = zip(*items)

# Prikazivanje stubaca znacaja atributa.
plt.bar(x=range(len(keys)), height=values)
plt.xticks(ticks=range(len(keys)), labels=keys, rotation=90)
plt.title('Feature importance')
plt.show()
```

Zadatak – Rešenje



Zadatak – Rešenje

```
# Vizuelizacija korelacione matrice (povezanost atributa).
# +-1 oznacava jaku vezu izmedju dva atributa, 0 odsustvo povezanosti.
plt.figure()
sb.heatmap(data.corr(), annot=True, fmt='.2f')
plt.show()
```



PITANJA?

http://ri4es.etf.rs/