Lab – Assignment 04

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
| **1** | **import pandas as** **pd**  **import numpy as** **np**  **data = pd.read\_csv("Titanic.csv")**  **np.random.seed(0)** |  |
| **2** | **data.head()**  **df = data** |  |
| **3** | **df.drop("Name",axis=1,inplace=True)**  **df.drop("Ticket",axis=1,inplace=True)**  **df.drop("PassengerId",axis=1,inplace=True)**  **df.drop("Cabin",axis=1,inplace=True)**  **df.drop("Embarked",axis=1,inplace=True)** |  |
| **4** | **from sklearn.preprocessing import LabelEncoder**  **le = LabelEncoder()**  **df['Sex'] = le.fit\_transform(df['Sex'])**  **newdf=df** |  |
| **5** | y = df['Survived']  df.drop("Survived",axis=1,inplace=True) |  |
| **6** | df.info() |  |
| **7** | print(df.isnull().sum()) |  |
| **8** | **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test,y\_train,y\_test = train\_test\_split(df,y,test\_size=0.3)**  **from sklearn.linear\_model import LogisticRegression**  **lr = LogisticRegression()**  **lr.fit(X\_train,y\_train)** |  |
| **9** | **updated\_df = df.dropna(axis=1)** |  |
| **10** | **updated\_df.info()** |  |
| **11** | **from sklearn import metrics**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test,y\_train,y\_test = train\_test\_split(updated\_df,y,test\_size=0.3)**  **from sklearn.linear\_model import LogisticRegression**  **lr = LogisticRegression()**  **lr.fit(X\_train,y\_train)**  **pred = lr.predict(X\_test)**  **print(metrics.accuracy\_score(pred,y\_test))** |  |
| **12** | **updated\_df = newdf.dropna(axis=0)** |  |
| **13** | **y1 = updated\_df['Survived']**  **updated\_df.drop("Survived",axis=1,inplace=True)** |  |
| **14** | **updated\_df.info()** |  |
| **15** | **#Copy 11 and Run** |  |
| **16** | **updated\_df = df**  **updated\_df['Age']=updated\_df['Age'].fillna(updated\_df['Age'].mean())**  **updated\_df.info()** |  |
| **17** | **y1 = updated\_df['Survived']**  **updated\_df.drop("Survived",axis=1,inplace=True)** |  |
| **18** | **#Copy 11 and Run** |  |
| **19** | **updated\_df = df**  **updated\_df['Ageismissing'] = updated\_df['Age'].isnull()**  **from sklearn.impute import SimpleImputer**  **my\_imputer = SimpleImputer(strategy = 'median')**  **data\_new = my\_imputer.fit\_transform(updated\_df)**  **updated\_df.info()** |  |
| **20** | **#Copy 11 and Run** |  |

|  |  |  |
| --- | --- | --- |
| **21** | **from sklearn.linear\_model import LinearRegression**  **lr = LinearRegression()**  **df.head()**  **testdf = df[df['Age'].isnull()==True]**  **traindf = df[df['Age'].isnull()==False]**  **y = traindf['Age']**  **traindf.drop("Age",axis=1,inplace=True)**  **lr.fit(traindf,y)**  **testdf.drop("Age",axis=1,inplace=True)**  **pred = lr.predict(testdf)**  **testdf['Age']= pred** |  |
| **22** | **traindf['Age']=y** |  |
| **23** | **y = traindf['Survived']**  **traindf.drop("Survived",axis=1,inplace=True)**  **from sklearn.linear\_model import LogisticRegression**  **lr = LogisticRegression()**  **lr.fit(traindf,y)** |  |
| **24** | **LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,intercept\_scaling=1, l1\_ratio=None, max\_iter=100,multi\_class='auto', n\_jobs=None, penalty='l2',random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,warm\_start=False)** |  |
| **25** | **y\_test = testdf['Survived']**  **testdf.drop("Survived",axis=1,inplace=True)**  **pred = lr.predict(testdf)** |  |
| **26** | **print(metrics.accuracy\_score(pred,y\_test))** |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
|  | **import numpy as np**  **import pandas as pd**  **from sklearn.model\_selection import train\_test\_split**  **df = pd.read\_csv(‘phpMYEkMl.csv’)**  **df = df.replace('?', np.nan)**  **def get\_first\_cabin(row):**  **try:**  **return row.split()[0]**  **except:**  **return np.nan**  **df['cabin'] = df['cabin'].apply(get\_first\_cabin)**  **df['cabin'] = df['cabin'].str[0]**  **df.fillna("Missing", inplace=True)**  **usecols=['sex', 'embarked', 'cabin', 'pclass', 'sibsp', 'parch', 'survived']**  **df[usefols].head()** |  |
|  | **X\_train, X\_test, y\_train, y\_test = train\_test\_split(**  **df[usecols],**  **df['survived'],**  **test\_size=0.3,**  **random\_state=0,**  **)** |  |
|  | **X\_train\_enc = pd.get\_dummies(X\_train, drop\_first=True)**  **X\_test\_enc = pd.get\_dummies(X\_test , drop\_first=True)**  **X\_train\_enc.head()** |  |
|  | **from sklearn.compose import ColumnTransformer**  **from sklearn.preprocessing import OneHotEncoder**  **encoder = OneHotEncoder(drop=”first”, sparse=False)**  **ohe = ColumnTransformer(**  **transformers=[("ohe", encoder, ["sex", "embarked", "cabin"]),],**  **remainder="passthrough",**  **)**  **ohe.set\_output(transform="pandas")** |  |
|  | **ohe.fit(X\_train)**  **X\_train\_t = ohe.transform(X\_train)**  **X\_test\_t = ohe.transform(X\_test)**  **X\_train\_t.head()** |  |
|  | **from feature\_engine.encoding import OneHotEncoder**  **ohe\_enc = OneHotEncoder(drop\_last=True)**  **X\_train\_enc = ohe\_enc.fit\_transform(X\_train)**  **X\_test\_enc = ohe.transform(X\_test)**  **X\_train\_enc.head()** |  |
|  | **from category\_encoders.one\_hot import OneHotEncoder**  **ohe\_enc = OneHotEncoder(use\_cat\_names=True)**  **ohe\_enc.fit(X\_train)**  **X\_train\_enc = ohe\_enc.transform(X\_train)**  **X\_test\_enc = ohe\_enc.transform(X\_test)** |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
| **1** | **import pandas as pd**  **import numpy as np**  **from scipy import stats**  **from mlxtend.preprocessing import minmax\_scaling**  **import seaborn as** **sns**  **import matplotlib.pyplot as** **plt**  **np.random.seed(0)** |  |
| **2** | **original\_data = np.random.exponential(size=1000)**  **scaled\_data = minmax\_scaling(original\_data, columns=[0])**  **fig, ax = plt.subplots(1, 2, figsize=(15, 3))**  **sns.histplot(original\_data, ax=ax[0], kde=True, legend=False)**  **ax[0].set\_title("Original Data")**  **sns.histplot(scaled\_data, ax=ax[1], kde=True, legend=False)**  **ax[1].set\_title("Scaled data")**  **plt.show()** |  |
| **3** | **normalized\_data = stats.boxcox(original\_data)**  **fig, ax=plt.subplots(1, 2, figsize=(15, 3))**  **sns.histplot(original\_data, ax=ax[0], kde=True, legend=False)**  **ax[0].set\_title("Original Data")**  **sns.histplot(normalized\_data[0], ax=ax[1], kde=True, legend=False)**  **ax[1].set\_title("Normalized data")**  **plt.show()** |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
| **1** | import pandas as pd  import numpy as np  import seaborn as sns  import datetime  *# read in our data*  data = pd.read\_csv("../catalog.csv")  *# set seed for reproducibility*  np.random.seed(0) |  |
| **2** | data.head() |  |
| **3** | *# print the first few rows of the date column*  print(data['date'].head()) |  |
| **4** | *# check the data type of our date column*  data['date'].dtype |  |
| **5** | *# create a new column, date\_parsed, with the parsed dates*  data['date\_parsed'] = pd.to\_datetime(data['date'], format="%m/**%d**/%y") |  |
| **6** | *# print the first few rows*  data['date\_parsed'].head() |  |
| **7** | *# get the day of the month from the date\_parsed column*  day\_of\_month = data['date\_parsed'].dt.day  day\_of\_month.head() |  |
| **8** | *# remove na's*  day\_of\_month = day\_of\_month.dropna()  *# plot the day of the month*  sns.distplot(day\_of\_month, kde=False, bins=31) |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
| **1** | import pandas as pd  import numpy as np  *# helpful character encoding module*  import charset\_normalizer  *# set seed for reproducibility*  np.random.seed(0) |  |
| **2** | *# start with a string*  before = "This is the euro symbol: €"  *# check to see what datatype it is*  type(before) |  |
| **3** | *# encode it to a different encoding, replacing characters that raise errors*  after = before.encode("utf-8", errors="replace")  *# check the type*  type(after) |  |
| **4** | *# take a look at what the bytes look like*  after |  |
| **5** | *# convert it back to utf-8*  print(after.decode("utf-8")) |  |
| **6** | *# try to decode our bytes with the ascii encoding*  print(after.decode("ascii")) |  |
| **7** | *# start with a string*  before = "This is the euro symbol: €"  *# encode it to a different encoding, replacing characters that raise errors*  after = before.encode("ascii", errors = "replace")  *# convert it back to utf-8*  print(after.decode("ascii"))  *# We've lost the original underlying byte string! It's been*  *# replaced with the underlying byte string for the unknown character :(* |  |
| **8** | *# try to read in a file not in UTF-8*  kickstarter\_2016 = pd.read\_csv("../ks-projects-201612.csv") |  |
| **9** | *# look at the first ten thousand bytes to guess the character encoding*  with open("../ks-projects-201801.csv", 'rb') as rawdata:  result = charset\_normalizer.detect(rawdata.read(10000))  *# check what the character encoding might be*  print(result) |  |
| **10** | *# read in the file with the encoding detected by charset\_normalizer*  kickstarter\_2016 = pd.read\_csv("../ks-projects-201612.csv", encoding='Windows-1252')  *# look at the first few lines*  kickstarter\_2016.head() |  |
| **11** | *# save our file (will be saved as UTF-8 by default!)*  kickstarter\_2016.to\_csv("ks-projects-201801-utf8.csv") |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Code.** |  |
| **1** | import pandas as pd  import numpy as np  *# helpful modules*  import fuzzywuzzy  from fuzzywuzzy import process  import charset\_normalizer  *# read in all our data*  professors = pd.read\_csv("../pakistan\_intellectual\_capital.csv")  *# set seed for reproducibility*  np.random.seed(0) |  |
| **2** | professors.head() |  |
| **3** | *# get all the unique values in the 'Country' column*  countries = professors['Country'].unique()  *# sort them alphabetically and then take a closer look*  countries.sort()  countries |  |
| **4** | *# convert to lower case*  professors['Country'] = professors['Country'].str.lower()  *# remove trailing white spaces*  professors['Country'] = professors['Country'].str.strip() |  |
| **5** | *# get all the unique values in the 'Country' column*  countries = professors['Country'].unique()  *# sort them alphabetically and then take a closer look*  countries.sort()  countries |  |
| **6** | *# get the top 10 closest matches to "south korea"*  matches = fuzzywuzzy.process.extract("south korea", countries, limit=10, scorer=fuzzywuzzy.fuzz.token\_sort\_ratio)  *# take a look at them*  matches |  |
| **7** | *# function to replace rows in the provided column of the provided dataframe*  *# that match the provided string above the provided ratio with the provided string*  def replace\_matches\_in\_column(df, column, string\_to\_match, min\_ratio = 47):  *# get a list of unique strings*  strings = df[column].unique()    *# get the top 10 closest matches to our input string*  matches = fuzzywuzzy.process.extract(string\_to\_match, strings,  limit=10, scorer=fuzzywuzzy.fuzz.token\_sort\_ratio)  *# only get matches with a ratio > 90*  close\_matches = [matches[0] for matches **in** matches if matches[1] >= min\_ratio]  *# get the rows of all the close matches in our dataframe*  rows\_with\_matches = df[column].isin(close\_matches)  *# replace all rows with close matches with the input matches*  df.loc[rows\_with\_matches, column] = string\_to\_match    *# let us know the function's done*  print("All done!")  *# use the function we just wrote to replace close matches to "south korea" with "south korea"*  replace\_matches\_in\_column(df=professors, column='Country', string\_to\_match="south korea") |  |
| **8** | *# get all the unique values in the 'Country' column*  countries = professors['Country'].unique()  *# sort them alphabetically and then take a closer look*  countries.sort()  countries |  |