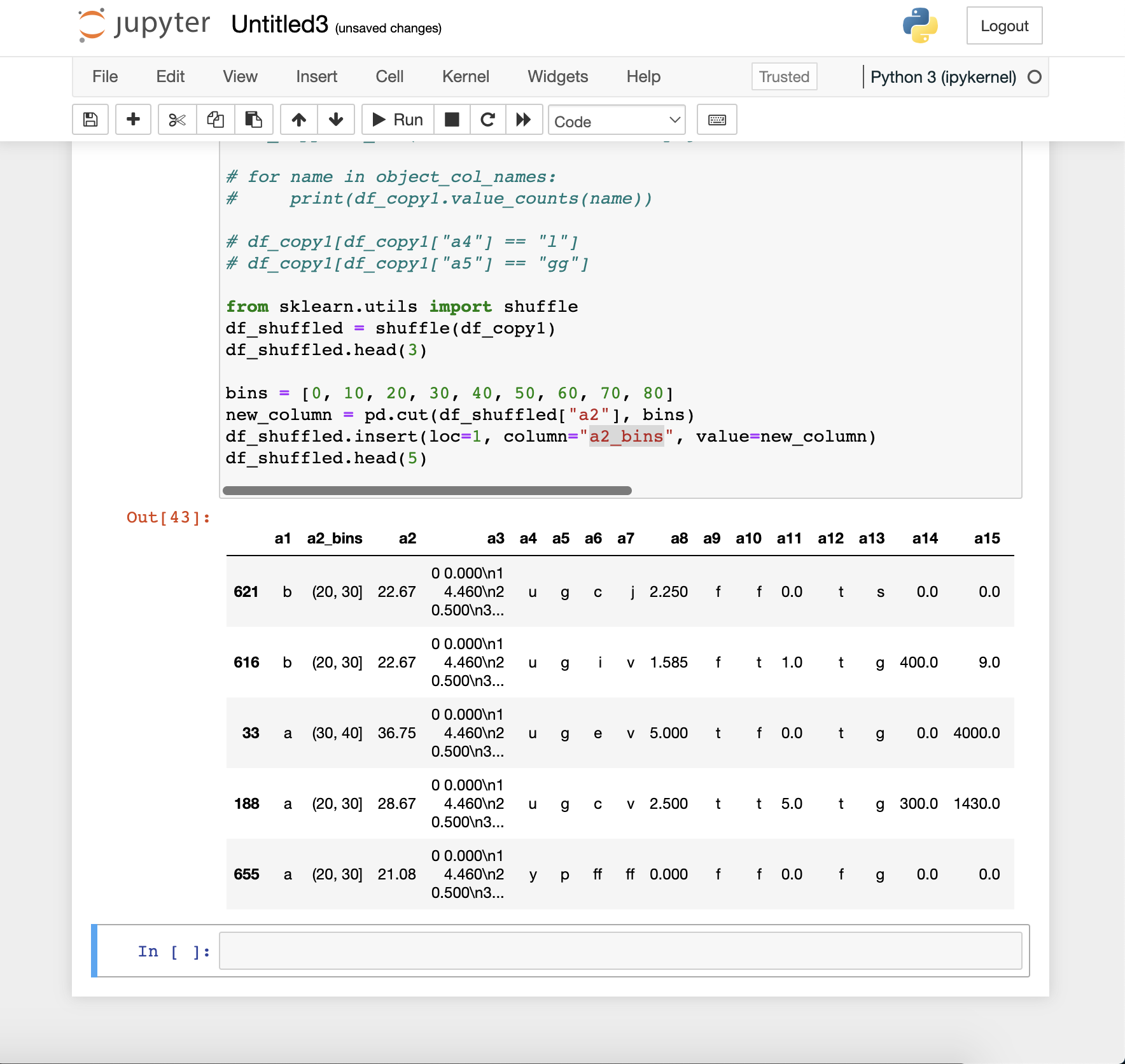
1. **Please read the document about pandas’ cut() method and convert a2 column of the DataFrame after manipulations in Figure 3.10 into bins [0, 10, 20, 30, 40, 50, 60, 70, 80], and store the result in a new column called a2\_bins (3 points)**

**Put all your code and screenshots of the corresponding output in a Word**  **document. Submit your assignment to the Lesson 3 dropbox by 11:59 pm ET on**  **day 7 of Week 4. For more information, consult the “Assignments” page in the**  **Syllabus Module.**



**Code specific to question:**

from sklearn.utils import shuffle

df\_shuffled = shuffle(df\_copy1)

df\_shuffled.head(3)

bins = [0, 10, 20, 30, 40, 50, 60, 70, 80]

new\_column = pd.cut(df\_shuffled["a2"], bins)

df\_shuffled.insert(loc=1, column="a2\_bins", value=new\_column)

df\_shuffled.head(5)

**Complete code (including lesson code to manipulate data):**

import pandas as pd

df = pd.read\_csv("/Users/msadowsk/Downloads/crx\_data.csv", na\_values="?", header=None)

df.columns=["a" + str(i) for i in range(1, 16)] + ["approved"]

df["a2"] = pd.to\_numeric(df["a2"], errors='coerce')

df["a3"] = str(df["a3"])

def get\_missing\_counts(df: pd.DataFrame):

for i in df.columns:

# count number of rows with missing values

# get the count of missing values for the i column. isnull() set all NaN values to be True and others are False.

# and sum() counts the number of True values in this column

n\_miss = df[i].isnull().sum()

# df.shape[0] is the count of all values in a column, including NaN values.

# So we can get missing percentage of each column via the code below

perc = n\_miss / df.shape[0] \* 100

print(f'> {i}, Missing: {n\_miss} {perc}')

df.head(5)

# get\_missing\_counts(df) # Run the function and print the missing rate of each column

df[df.isna().any(axis=1)]

df\_copy = df.copy()

# #use the mean of all available values in column a2 and a4 to fill their missing ones

df\_copy[["a2", "a14"]] = df\_copy[["a2", "a14"]].fillna(df\_copy[["a2", "a14"]].mean().iloc[0])

# #use the mode value of a1, a4, a5, a6 and a7 to fill their missing values

df\_copy[["a1", "a4", "a5", "a6", "a7"]] = df\_copy[["a1", "a4", "a5", "a6", "a7"]].fillna(df\_copy[["a1", "a4", "a5", "a6", "a7"]].mode().iloc[0])

# # display the last row of df\_copy which used to have missing values in a4, a5, a6, a7 and a14

df\_copy.iloc[622]

from sklearn.impute import KNNImputer, SimpleImputer

import numpy as np

# Make another copy of df

df\_copy1 = df.copy()

# find all object columns so we can use SimleImputer on them

object\_col\_names = [name for name in df\_copy1.columns if df\_copy1[name].dtype == np.dtype('O')]

#strategy means how do we fill the missing values. Here we use `most\_frequent`. It is the same as our mode value way above

imp = SimpleImputer(strategy='most\_frequent')

#apply the SimplerImputer tranformer on object/categorical columns

df\_copy1[object\_col\_names] = imp.fit\_transform(df\_copy1[object\_col\_names])

# find all numeric columns so we can use KNNImputer on them

numeric\_col\_names = [name for name in df\_copy1.columns if df\_copy1[name].dtype != np.dtype('O')]

#create a KNNImputer and it uses the 5 nearest neighbours of a row to infer the missing numeric values

#weights = 'uniform' means all its neighbours have the same impact on inferring this.

knn\_imp = KNNImputer(n\_neighbors=5, weights='uniform', metric='nan\_euclidean')

#apply the KNNImputer transformer on the numeric columns

df\_copy1[numeric\_col\_names] = knn\_imp.fit\_transform(df\_copy1[numeric\_col\_names])

# display row 622

# df\_copy1.iloc[622]

# df\_copy1.to\_csv("/Users/msadowsk/Desktop/git/Wilfrid-Laurier/CP610/course-material/datasets/crx\_data\_wo\_missing\_data.csv")

# for name in object\_col\_names:

# print(df\_copy1.value\_counts(name))

# df\_copy1[df\_copy1["a4"] == "l"]

# df\_copy1[df\_copy1["a5"] == "gg"]

from sklearn.utils import shuffle

df\_shuffled = shuffle(df\_copy1)

df\_shuffled.head(3)

bins = [0, 10, 20, 30, 40, 50, 60, 70, 80]

new\_column = pd.cut(df\_shuffled["a2"], bins)

df\_shuffled.insert(loc=1, column="a2\_bins", value=new\_column)

df\_shuffled.head(5)