Group 5

10/11/2022

CSCE 4143 Practice Project

Data pre-processing:

Using the pandas library and the adult dataset download URLs, we put the train & test data into their own respective DataFrames and manually assigned the columns based on adult.names:

```python
import pandas as pd
import numpy as np

# Create Train & Test DataFrames from Adult Dataset files
data = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
test = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test'

# Part 1 DataFrames
train_df1 = pd.read_csv(data, header=None, na_values=" ?")
test_df1 = pd.read_csv(test, skiprows=1, na_values=" ?")

# Add columns to Train DataFrames using adult.names column descriptions
columns = [
  'age',
  'workclass',
  'fnlwgt',
  'education',
  'education-num',
  'marital-status',
  'occupation',
  'relationship',
  'race',
  'sex',
  'capital-gain',
  'capital-loss',
  'hours-per-week',
  'native-country',
  'income',
 ]

train_df1.columns = columns
test_df1.columns = columns

# Part 2-4 DataFrames
train_df2 = train_df1
test_df2 = test_df1

# Print datatypes & training DataFrame
print(train_df1.dtypes, end='\n---------------------------')
train_df1
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

32561 rows × 15 columns

For part 1 of the project, we remove unknown records (using dropna since we defined na_values as those with a " ?") and continuous attributes. The test dataset had a "." at the end of each row so we removed those from the income column to allow for proper predictions:

```python
# Drop rows with null values
train_df1.dropna(inplace = True)
train_df1.dropna(inplace = True)

# Drop continuous attributes
continuous_attributes = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

train_df1 = train_df1.drop(continuous_attributes, axis = 1)
test_df1 = test_df1.drop(continuous_attributes, axis = 1)

# Fix income column of test_df (remove unnecessary '.')
test_df1['income'] = test_df1['income'].str.replace(".","", regex=False)

# Print training DataFrame
train_df1
```

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...
32556	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Female	United-States	<=50K
32557	Private	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	United-States	>50K
32558	Private	HS-grad	Widowed	Adm-clerical	Unmarried	White	Female	United-States	<=50K
32559	Private	HS-grad	Never-married	Adm-clerical	Own-child	White	Male	United-States	<=50K
32560	Self-emp-inc	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Female	United-States	>50K

30162 rows × 9 columns

We then one-hot encode the multi-domain categorical attributes which can be found in adult.names. After that, we need to ensure that our DataFrames have the same shape (in case a one-hot encoded column exists in either train or test but not the other). To do so we inner join the 2 DataFrames and then split them into the features (X) and target (Y):

```python
[ ] # Function to replace encodable features with their encoded column
    def oneHotEncode(df, feature):
      dummies = pd.get_dummies(df[feature])
      new_df = pd.concat([df, dummies], axis=1).drop(feature, axis=1)
      return(new_df)

    # Columns to be one-hot encoded (multi-domain categorical attribute)
    encodable_columns = [
      'workclass',
      'education',
      'marital-status',
      'occupation',
      'relationship',
      'race',
      'sex',
      'native-country',
    ]

    train_df1 = oneHotEncode(train_df1, encodable_columns)
    test_df1 = oneHotEncode(test_df1, encodable_columns)
```

```
[ ] # Inner join to only include columns that exist in both train & test (removes 1 from train)
    train_df1, test_df1 = train_df1.align(test_df1, join='inner', axis=1)

    # X and Y columns for later evaluation
    target = 'income'

    X_train = train_df1.loc[:, train_df1.columns != target]
    Y_train = train_df1[target]

    X_test = test_df1.loc[:, test_df1.columns != target]
    Y_test = test_df1[target]
```

For part 2-4 of the project, our data pre-processing is almost identical except we keep our continuous attributes and transform them into binary attributes before splitting into X & Y. We do this by comparing each value in the columns to the mean of that column and making it a 0 if the value is lower and a 1 if it is higher:

```
# Function to replace numerical features with their binary feature
def numerical_to_binary(df, features):
  for feature in features:
    mean = df[feature].mean()
    df.loc[(df[feature] < mean), feature] = 0
    df.loc[(df[feature] >= mean), feature] = 1

# Columns to be converted to binary
numerical_columns = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week',
]
numerical_to_binary(train_df2, numerical_columns)
numerical_to_binary(test_df2, numerical_columns)
```

<u>Metrics function:</u>

Since we want to find the metrics (accuracy, recall/TPR, f1-score, FPR, etc.) for most of our algorithms/classifiers, we made a function that we can call with the test dataset's target & predictions:

```
[ ] from sklearn.metrics import accuracy_score
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix

    def metrics(y_test, predictions):
      cm = confusion_matrix(y_test, predictions)
      FP = cm.sum(axis=0) - np.diag(cm)
      FN = cm.sum(axis=1) - np.diag(cm)
      TP = np.diag(cm)
      TN = cm.sum() - (FP + FN + TP)

      TPR = dict(zip(['<=50K', '>50k'], (TP/(TP+FN))))
      FPR = dict(zip(['<=50K', '>50k'], (FP/(FP+TN))))

      print(f"Accuracy = {accuracy_score(y_test, predictions)}\n")
      print(f"TP rate = {TPR}\n")
      print(f"FP rate = {FPR}\n")
      print(classification_report(y_test, predictions))
```

1a) Decision Tree Classifier:

We use scikit-learn's built in DecisionTreeClassifer() to create the classifier and fit it to our training data. From there we create our predictions to plug into our metrics function to get the necessary evaluation:

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train, Y_train)
predictions = dt.predict(X_test)

metrics(Y_test, predictions)
```

```
Accuracy = 0.817014742014742

TP rate = {'<=50K': 0.8972173073829821, '>50k': 0.5577223088923557}

FP rate = {'<=50K': 0.4422776911076443, '>50k': 0.10278269261701785}

              precision    recall  f1-score   support

       <=50K       0.87      0.90      0.88     12434
        >50K       0.63      0.56      0.59      3846

    accuracy                           0.82     16280
   macro avg       0.75      0.73      0.74     16280
weighted avg       0.81      0.82      0.81     16280
```

1b) Naïve Bayesian Classifier:

We use scikit-learn's built in GaussianNB() to create the classifier and fit it to our training data. From there we create our predictions to plug into our metrics function to get the necessary evaluation:

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, Y_train)
predictions = nb.predict(X_test)

metrics(Y_test, predictions)
```

```
Accuracy = 0.562960687960688

TP rate = {'<=50K': 0.45311243364967024, '>50k': 0.9180967238689548}

FP rate = {'<=50K': 0.08190327613104524, '>50k': 0.5468875663503298}

              precision    recall  f1-score   support

       <=50K       0.95      0.45      0.61     12434
        >50K       0.34      0.92      0.50      3846

    accuracy                           0.56     16280
   macro avg       0.64      0.69      0.56     16280
weighted avg       0.80      0.56      0.59     16280
```

2a) K-means clustering:

Note that we are now in part 2 of the assignment so our DataFrames are different since our data pre-processing includes the continuous attributes (converted to binary). We use scikit-learn's built in KMeans() to create the clusters and fit it to our training data. Since we are making 3 different K-means clustering for different numbers of clusters, we made a function that we call for each different k-value:

```
[ ] from sklearn.cluster import KMeans

    def km(k, X_train, Y_train):
      km = KMeans(n_clusters=k, random_state=0).fit(X_train, Y_train)
      return(km)
```

We report the cluster centroids for each k-value (*note that this can be hard to read here but it is also available on project.ipynb which includes outputs*):

K = 3

```
[ ]  print(km(3, X_train, Y_train).cluster_centers_)
```

```
[[ 2.79105431e-01  4.78466454e-01  2.74249201e-01  5.38019169e-02
   3.46325879e-02  2.84984026e-01  2.70926518e-02  4.93290735e-02
   7.88370607e-01  2.41533546e-02  7.33546326e-02  3.71884984e-02
   5.11182109e-04  3.47603834e-02  4.89456869e-02  2.03194888e-02
   5.36741214e-03  1.16293930e-02  1.63578275e-02  1.82747604e-02
   3.11821086e-02  3.46325879e-02  1.52971246e-01  6.90095847e-03
   3.39808307e-01  3.64217252e-02  2.30031949e-03  1.21405751e-02
   2.27987220e-01  2.15335463e-01 -2.04914211e-17  4.98402556e-03
   2.31309904e-02  6.91884984e-01  4.66453674e-02  1.80191693e-02
   7.71884984e-02  7.66773163e-04  1.69329073e-01  8.85623003e-02
   4.69009585e-02  9.61022364e-02  7.01597444e-02  1.21022364e-01
   8.94568690e-04  1.00830671e-01  2.50479233e-02  1.08242812e-01
   2.84984026e-02  6.64536741e-02  1.14908083e-14  5.31246006e-01
   5.54632588e-02  3.17571885e-01  9.57188498e-02  1.57512892e-15
   1.13738019e-02  2.95207668e-02  1.01725240e-01  8.56230032e-03
   8.48817891e-01 -7.43849426e-15  1.00000000e+00  7.66773163e-04
   3.32268371e-03  1.66134185e-03  2.17252396e-03  1.66134185e-03
   2.55591054e-03  7.66773163e-04  4.85623003e-03  2.93929712e-03
   7.66773163e-04  2.55591054e-03  7.66773163e-04  3.70607029e-03
   1.91693291e-03  7.66773163e-04  6.38977636e-04  2.55591054e-04
   3.45047923e-03  1.78913738e-03  1.15015974e-03  1.15015974e-03
   2.17252396e-03  1.78913738e-03  3.83386581e-04  2.77316294e-02
   1.27795527e-03  6.38977636e-04  1.02236422e-03  4.72843450e-03
   1.78913738e-03  8.94568690e-04  3.83386581e-03  5.11182109e-04
   2.17252396e-03  1.15015974e-03  3.83386581e-04  1.27795527e-04
   9.06709265e-01  2.81150160e-03  2.55591054e-04]
```

```
[ 6.09876543e-01  4.25248905e-01  3.74990044e-01  1.21306252e-01
  6.52329749e-02  4.22700119e-01  3.36121067e-02  6.82596575e-02
  6.75029869e-01  6.04540024e-02  1.22102748e-01  4.01433692e-02
  3.98247710e-04  2.37355635e-02  2.34169654e-02  7.64635603e-03
  5.25686977e-03  1.01951414e-02  2.36559140e-02  1.53723616e-02
  2.93906810e-02  4.62763839e-02  1.85185185e-01  1.91158901e-02
  3.24571884e-01  6.63480685e-02  1.03544405e-03  2.86738351e-02
  1.90123457e-01 -4.52415883e-15  7.16845878e-04  9.99283154e-01
  7.37257477e-16  3.38062911e-14 -3.43475248e-16 -3.08433834e-15
  4.81879729e-02  2.38948626e-04  1.98247710e-01  1.71724413e-01
  4.43647949e-02  3.45679012e-02  6.96136997e-02  4.03823178e-02
  7.96495420e-05  1.40023895e-01  2.96296296e-02  1.18598168e-01
  2.77180406e-02  7.66228594e-02  9.92592593e-01  2.38948626e-04
  5.49581840e-03  1.59299084e-03 -3.26128013e-15  7.96495420e-05
  7.16845878e-03  2.94703305e-02  4.95420151e-02  6.13301474e-03
  9.07686181e-01 -7.88258347e-15  1.00000000e+00  7.96495420e-04
  3.74352847e-03  2.94703305e-03  1.27439267e-03  3.26563122e-03
  1.03544405e-03  9.55794504e-04  2.30983672e-03  2.62843489e-03
  8.76144962e-04  4.30107527e-03  1.43369176e-03  1.11509359e-03
  7.16845878e-04  8.13151629e-19  7.16845878e-04  3.98247710e-04
  4.93827160e-03  1.67264038e-03  6.37196336e-04  3.26563122e-03
  1.67264038e-03  2.15053763e-03  5.57546794e-04  2.11071286e-02
  8.76144962e-04  1.59299084e-04  6.37196336e-04  6.29231382e-03
  1.99123855e-03  1.19474313e-03  2.46913580e-03  2.38948626e-04
  2.30983672e-03  1.83193947e-03  3.18598168e-04  5.57546794e-04
  9.14137794e-01  1.59299084e-03  8.76144962e-04]
```

```
[ 4.20363934e-01  4.21386220e-01  3.11694950e-01  5.37722347e-02
  3.44510325e-02  1.69699448e-01  3.15886322e-02  8.42363525e-02
  7.81230832e-01  1.28808015e-02  4.00736046e-02  4.94786342e-02
  5.11142916e-04  2.55571458e-02  3.79268043e-02  1.24718871e-02
  4.39582907e-03  7.05377223e-03  1.34941730e-02  1.21652014e-02
  4.03802903e-02  4.65140053e-02  1.55591903e-01  8.28051523e-03
  3.17521979e-01  5.20343488e-02  1.43120016e-03  8.89388673e-03
  2.56287058e-01  2.58536087e-01  1.22674300e-03  1.51298303e-01
  1.93212022e-02  4.40809650e-01  5.86792067e-02  7.01288080e-02
  2.56798201e-01 -2.39608680e-17  2.20813740e-02  1.16847270e-01
  6.64485790e-03  1.67654876e-02  5.55101206e-02  1.79717849e-01
  1.38008587e-02  1.52422817e-01  7.76937232e-03  1.27581272e-01
  3.48599468e-02  9.20057248e-03  1.02228583e-04  3.64547127e-01
  3.94602331e-02  2.00470251e-01  2.51789000e-01  1.43631159e-01
  1.09384584e-02  3.00552034e-02  1.43017788e-01  8.89388673e-03
  8.07094664e-01  1.00000000e+00  8.10462808e-15  2.04457166e-04
  3.47577183e-03  1.84011450e-03  2.35125741e-03  3.88468616e-03
  3.47577183e-03  9.20057248e-04  3.37354324e-03  3.06685749e-03
  1.02228583e-03  5.52034349e-03  5.11142916e-04  2.04457166e-03
  1.84011450e-03  6.13371499e-04  5.11142916e-04  6.13371499e-04
  1.12451441e-03  7.15600082e-04  7.15600082e-04  1.84011450e-03
  4.29360049e-03  1.84011450e-03  7.15600082e-04  1.30852586e-02
  1.22674300e-03  7.15600082e-04  1.43120016e-03  7.36045798e-03
  1.73788591e-03  1.22674300e-03  4.90697199e-03  4.08914332e-04
  2.55571458e-03  1.02228583e-03  1.02228583e-03  1.02228583e-03
  9.13105704e-01  2.24902883e-03  3.06685749e-04]]
```

K = 5

```
[ ]  print(km(5, X_train, Y_train).cluster_centers_)
```

```
[[ 5.87614446e-01  4.31078332e-01 -1.29896094e-14  9.25737538e-02
   5.12461851e-02  3.69277721e-01  2.79755849e-02  5.77314344e-02
   7.10579858e-01  4.74313327e-02  1.28687691e-01  2.70854527e-02
   5.08646999e-04  3.78942014e-02  3.73855544e-02  1.22075280e-02
   8.39267548e-03  1.62767040e-02  3.77670397e-02  2.45422177e-02
  -1.31838984e-16 -4.25354196e-15 -4.27435864e-15  6.83481050e-16
   5.20091556e-01  4.87110352e-15  1.65310275e-03 -1.20042865e-15
   3.03789420e-01  1.03805853e-14  7.62970498e-04  9.99237030e-01
   5.70724024e-16  1.66533454e-15  3.17801341e-15  5.37764278e-16
   4.93387589e-02  2.54323499e-04  2.67166836e-01  1.13301119e-01
   5.73499491e-02  4.89572737e-02  1.00330621e-01  5.50610376e-02
   1.80411242e-16  3.25534079e-02  3.20447609e-02  1.09867752e-01
   2.12360122e-02  1.12538149e-01  9.89954222e-01  1.32116540e-14
   6.61241099e-03  3.43336724e-03 -1.00891517e-14  1.61676228e-15
   1.00457782e-02  1.86927772e-02  6.02746694e-02  7.62970498e-03
   9.03357070e-01  1.27161750e-04  9.99872838e-01  1.14445575e-03
   2.54323499e-03  1.65310275e-03  1.27161750e-03  3.68769074e-03
   1.52594100e-03  1.01729400e-03  2.67039674e-03  1.27161750e-03
   2.54323499e-04  3.43336724e-03  1.14445575e-03  1.65310275e-03
   8.90132248e-04 -3.68086638e-17  2.54323499e-04  3.81485249e-04
   1.01729400e-03  3.81485249e-04  6.35808749e-04  3.43336724e-03
   1.78026450e-03  7.62970498e-04  6.35808749e-04  3.11546287e-02
   1.01729400e-03  2.54323499e-04  7.62970498e-04  4.06917599e-03
   2.28891150e-03  1.52594100e-03  2.92472024e-03  2.54323499e-04
   1.65310275e-03  5.08646999e-04  3.81485249e-04  8.90132248e-04
   9.16836216e-01  1.52594100e-03  5.08646999e-04]
```

```
[ 1.31852880e-01  4.44367337e-01  3.33795975e-01  3.56234097e-02
  2.84524636e-02  1.44575526e-01  2.49826509e-02  6.06060606e-02
  8.37150127e-01  6.93962526e-03  2.17441591e-02  4.85773768e-02
  8.23993651e-18  2.52139718e-02  5.01966227e-02  1.73490632e-02
  3.23849179e-03  5.78302105e-03  6.24566273e-03  8.79019200e-03
  3.56234097e-02  3.93245431e-02  1.97085357e-01  7.40226694e-03
  2.54684247e-01  4.62641684e-02  2.31320842e-03  8.09622947e-03
  2.92389544e-01  3.41393580e-15  2.31320842e-04  2.08188758e-03
  1.01781170e-02  9.78024520e-01  8.32755031e-03  1.15660421e-03
  2.54915568e-01  6.45100293e-18  1.57298173e-02  8.79019200e-02
  4.85773768e-03  1.89683090e-02  4.64954892e-02  1.94078186e-01
  1.29539672e-02  1.49433264e-01  8.55887115e-03  1.61461948e-01
  3.72426556e-02  7.40226694e-03  1.06026299e-14  4.44829979e-01
  4.69581309e-02  4.07356003e-01  1.00855887e-01 -1.04777298e-15
  7.63358779e-03  3.28475596e-02  1.45500810e-01  9.71547536e-03
  8.04302568e-01  1.00000000e+00  7.54951657e-15  2.31320842e-04
  2.54452926e-03  4.62641684e-04  1.85056674e-03  2.54452926e-03
  3.46981263e-03  6.93962526e-04  3.93245431e-03  2.08188758e-03
  9.25283368e-04  4.39509600e-03  2.31320842e-04  2.77585010e-03
  1.15660421e-03  2.31320842e-04  4.62641684e-04  4.62641684e-04
  1.15660421e-03  2.31320842e-04  6.93962526e-04  9.25283368e-04
  5.55170021e-03  1.38792505e-03  6.93962526e-04  1.54984964e-02
  1.61924589e-03  6.93962526e-04  1.38792505e-03  7.17094610e-03
  1.85056674e-03  9.25283368e-04  2.77585010e-03 -4.66206934e-18
  1.38792505e-03  1.15660421e-03  9.25283368e-04  1.15660421e-03
  9.20888272e-01  3.00717095e-03  2.31320842e-04]
```

```
[ 6.51128915e-01  4.02403496e-01  2.94610342e-01  6.84632192e-02
  3.93299345e-02  1.88455936e-01  3.73270211e-02  1.03787327e-01
  7.34887109e-01  1.76620539e-02  5.44428259e-02  5.09832484e-02
  9.10415149e-04  2.56737072e-02  2.82228696e-02  8.55790240e-03
  5.28040787e-03  8.01165331e-03  1.91187181e-02  1.47487254e-02
  4.46103423e-02  5.22578296e-02  1.22541879e-01  8.92206846e-03
  3.67807720e-01  5.66278223e-02  7.28332119e-04  9.65040058e-03
  2.27239621e-01  4.66678806e-01  2.00291333e-03  2.67479971e-01
  2.64020393e-02  1.52949745e-02  9.79606701e-02  1.24180626e-01
  2.59286235e-01  8.18572640e-18  2.71303714e-02  1.40203933e-01
  8.01165331e-03  1.49308084e-02  6.22723962e-02  1.67516387e-01
  1.43845594e-02  1.54770575e-01  7.10123816e-03  1.00145666e-01
  3.33211945e-02  1.09249818e-02  1.09912079e-14  2.99162418e-01
  3.31391114e-02  3.64166060e-02  3.75455208e-01  2.55826657e-01
  1.38383103e-02  2.82228696e-02  1.41478514e-01  8.19373634e-03
  8.08266570e-01  9.93627094e-01  6.37290605e-03  1.82083030e-04
  4.18790969e-03  2.91332848e-03  2.73124545e-03  4.91624181e-03
  3.45957757e-03  1.09249818e-03  2.91332848e-03  3.82374363e-03
  1.09249818e-03  6.37290605e-03  7.28332119e-04  1.45666424e-03
  2.36707939e-03  1.09249818e-03  5.46249090e-04  7.28332119e-04
  1.09249818e-03  1.09249818e-03  7.28332119e-04  2.54916242e-03
  3.27749454e-03  2.18499636e-03  7.28332119e-04  1.11070648e-02
  9.10415149e-04  7.28332119e-04  1.45666424e-03  7.82957028e-03
  1.63874727e-03  1.45666424e-03  6.55498908e-03  7.28332119e-04
  3.45957757e-03  9.10415149e-04  1.09249818e-03  9.10415149e-04
  9.06955572e-01  1.63874727e-03  3.64166060e-04]
```

```
[ 6.50521921e-01  4.13778706e-01  1.00000000e+00  1.70981211e-01
  8.76826722e-02  5.15240084e-01  4.36325678e-02  8.58037578e-02
  6.13152401e-01  8.26722338e-02  1.12317328e-01  6.22129436e-02
  2.08768267e-04 -1.15185639e-15 -9.15933995e-16 -3.07046055e-16
  1.46584134e-16  2.32452946e-16  1.83880688e-16 -4.52762827e-16
  7.70354906e-02  1.21503132e-01  4.96868476e-01  5.13569937e-02
 -1.15463195e-14  1.77035491e-01  6.59194921e-17  7.62004175e-02
 -1.88737914e-15  1.06471816e-02  6.26304802e-04  9.82672234e-01
  1.46137787e-03 -1.65978342e-14  2.29645094e-03  2.29645094e-03
  4.59290188e-02  2.08768267e-04  8.26722338e-02  2.71189979e-01
  2.25469729e-02  1.04384134e-02  1.79540710e-02  1.58663883e-02
  2.08768267e-04  3.20459290e-01  2.52609603e-02  1.32776618e-01
  3.77870564e-02  1.67014614e-02  9.76617954e-01  3.54906054e-03
  4.38413361e-03  2.29645094e-03  1.29436326e-02  2.08768267e-04
  2.29645094e-03  4.71816284e-02  3.15240084e-02  3.75782881e-03
  9.15240084e-01  2.08768267e-04  9.99791232e-01  2.08768267e-04
  5.63674322e-03  5.01043841e-03  1.25260960e-03  2.50521921e-03
  2.08768267e-04  8.35073069e-04  1.87891441e-03  4.80167015e-03
  2.29645094e-03  5.84551148e-03  1.87891441e-03  2.08768267e-04
  4.17536534e-04 -2.54245409e-17  1.46137787e-03  4.17536534e-04
  1.14822547e-02  3.75782881e-03  6.26304802e-04  2.92275574e-03
  1.46137787e-03  4.38413361e-03  4.17536534e-04  4.59290188e-03
  6.26304802e-04 -2.16840434e-18  4.17536534e-04  9.81210856e-03
  1.67014614e-03  6.26304802e-04  1.67014614e-03  2.08768267e-04
  3.34029228e-03  3.96659708e-03  2.08768267e-04  1.45283091e-17
  9.09812109e-01  1.67014614e-03  1.46137787e-03]
```

```
[ 2.69075783e-01  4.81216690e-01  2.66735994e-01  5.10854023e-02
  3.44468998e-02  2.81424672e-01  2.61276485e-02  4.80956714e-02
  7.93708566e-01  2.31379176e-02  7.22734954e-02  3.61367477e-02
  5.19953204e-04  3.53568179e-02  4.96555310e-02  2.06681399e-02
  5.45950864e-03  1.18289354e-02  1.66385025e-02  1.85883271e-02
  3.11971923e-02  3.48368647e-02  1.48056675e-01  6.23943845e-03
  3.41739243e-01  3.48368647e-02  2.33978942e-03  1.15689588e-02
  2.30989211e-01  2.07981282e-01 -1.83230167e-17  2.85974262e-03
  2.26179644e-02  7.03756662e-01  4.60158586e-02  1.67684908e-02
  7.63031327e-02  7.79929806e-04  1.71064604e-01  8.47523723e-02
  4.74457299e-02  9.76212141e-02  7.12335890e-02  1.22708956e-01
  9.09918107e-04  9.76212141e-02  2.53477187e-02  1.08670220e-01
  2.85974262e-02  6.69439750e-02  1.14352972e-14  5.38541531e-01
  5.60249578e-02  3.20681139e-01  8.47523723e-02  1.43635104e-15
  1.13089822e-02  2.92473677e-02  1.02170805e-01  8.57922787e-03
  8.48693618e-01 -7.43849426e-15  1.00000000e+00  7.79929806e-04
  3.37969583e-03  1.68984791e-03  2.20980112e-03  1.68984791e-03
  2.59976602e-03  7.79929806e-04  4.80956714e-03  2.98973092e-03
  5.19953204e-04  2.46977772e-03  7.79929806e-04  3.76966073e-03
  1.94982452e-03  6.49941505e-04  6.49941505e-04  2.59976602e-04
  3.37969583e-03  1.81983621e-03  1.16989471e-03  1.16989471e-03
  2.20980112e-03  1.81983621e-03  3.89964903e-04  2.79474847e-02
  1.29988301e-03  6.49941505e-04  1.03990641e-03  4.54959054e-03
  1.68984791e-03  9.09918107e-04  3.89964903e-03  5.19953204e-04
  2.20980112e-03  1.16989471e-03  3.89964903e-04  1.29988301e-04
  9.06538412e-01  2.85974262e-03  2.59976602e-04]]
```

K = 10

```
[ ]  print(km(10, X_train, Y_train).cluster_centers_)

     [[5.14266304e-01 3.93342391e-01 3.94021739e-01 ... 8.78396739e-01
       2.71739130e-03 6.79347826e-04]
      [5.73313783e-01 4.11290323e-01 6.60582700e-15 ... 9.44525904e-01
       1.22189638e-03 4.88758553e-04]
      [7.13421053e-01 4.06578947e-01 2.37631579e-01 ... 9.17368421e-01
       1.05263158e-03 2.63157895e-04]
      ...
      [6.51644885e-01 4.38035151e-01 2.62280306e-01 ... 9.24740874e-01
       4.50653447e-04 1.73472348e-18]
      [1.02297765e-01 4.50424929e-01 7.36543909e-02 ... 9.19420837e-01
       3.46238590e-03 3.14762354e-04]
      [6.01223730e-01 4.52779995e-01 5.88418203e-15 ... 8.87204044e-01
       1.59616919e-03 5.32056398e-04]]
```

2b) KNN:

Like K-means, we create a function that we can call for each value of k. From there, we plug in the predictions from that model into the metrics function we made earlier:

```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     def knn(k, X_train, Y_train, X_test):
       knn = KNeighborsClassifier(n_neighbors=k)
       knn.fit(X_train, Y_train)
       predictions = knn.predict(X_test)
       return(predictions)
```

K = 3

```
[ ] metrics(Y_test, knn(3, X_train, Y_train, X_test))
```

```
Accuracy = 0.805830400424995

TP rate = {'<=50K': 0.897438154767145, '>50k': 0.5245945945945946}

FP rate = {'<=50K': 0.4754054054054054, '>50k': 0.10256184523285501}

              precision    recall  f1-score   support

       <=50K       0.85      0.90      0.87     11359
        >50K       0.62      0.52      0.57      3700

    accuracy                           0.81     15059
   macro avg       0.74      0.71      0.72     15059
weighted avg       0.80      0.81      0.80     15059
```

K = 5

```
metrics(Y_test, knn(5, X_train, Y_train, X_test))
```

```
Accuracy = 0.8221661464904708

TP rate = {'<=50K': 0.9108196144026763, '>50k': 0.55}

FP rate = {'<=50K': 0.45, '>50k': 0.0891803855973237}

              precision    recall  f1-score   support

       <=50K       0.86      0.91      0.89     11359
        >50K       0.67      0.55      0.60      3700

    accuracy                           0.82     15059
   macro avg       0.76      0.73      0.74     15059
weighted avg       0.81      0.82      0.82     15059
```

K = 10

```
[ ] metrics(Y_test, knn(10, X_train, Y_train, X_test))

    Accuracy = 0.8284746663125041

    TP rate = {'<=50K': 0.9318601989611761, '>50k': 0.5110810810810811}

    FP rate = {'<=50K': 0.4889189189189189, '>50k': 0.06813980103882385}

                  precision    recall  f1-score   support

           <=50K       0.85      0.93      0.89     11359
            >50K       0.71      0.51      0.59      3700

        accuracy                           0.83     15059
       macro avg       0.78      0.72      0.74     15059
    weighted avg       0.82      0.83      0.82     15059
```

3) SVM:

We use scikit-learn's built in SVC() to create the classifier and fit it to our training data. From there we create our predictions to plug into our metrics function to get the necessary evaluation:

```
[ ] from sklearn.svm import SVC

    svc = SVC(kernel='linear')
    svc.fit(X_train, Y_train)
    predictions = svc.predict(X_test)

    metrics(Y_test, predictions)

    Accuracy = 0.8434158974699515

    TP rate = {'<=50K': 0.925873756492649, '>50k': 0.5902702702702702}

    FP rate = {'<=50K': 0.4097297297297297, '>50k': 0.074126243507351}

                  precision    recall  f1-score   support

           <=50K       0.87      0.93      0.90     11359
            >50K       0.72      0.59      0.65      3700

        accuracy                           0.84     15059
       macro avg       0.80      0.76      0.77     15059
    weighted avg       0.84      0.84      0.84     15059
```

4) Neural Network:

We use scikit-learn's built in MLPClassifier() to create the classifier and fit it to our training data. From there we create our predictions to plug into our metrics function to get the necessary evaluation:

```
[ ] from sklearn.neural_network import MLPClassifier

    nn = MLPClassifier(random_state=0, max_iter=300)
    nn.fit(X_train, Y_train)
    predictions = nn.predict(X_test)

    metrics(Y_test, predictions)
```

```
Accuracy = 0.8201739823361445

TP rate = {'<=50K': 0.8889867065762831, '>50k': 0.6089189189189189}

FP rate = {'<=50K': 0.3910810810810811, '>50k': 0.11101329342371688}

              precision    recall  f1-score   support

       <=50K       0.87      0.89      0.88     11359
        >50K       0.64      0.61      0.62      3700

    accuracy                           0.82     15059
   macro avg       0.76      0.75      0.75     15059
weighted avg       0.82      0.82      0.82     15059
```

In conclusion, we can see a clear winner in terms of accuracy with the Decision Tree Classifier having an 89.7% accuracy. However, also within part 1, the Naïve Bayes Classifier had the lowest reported accuracy with 56.3%. For the K-Nearest-Neighbors, we saw an increase in accuracy and recall as the number of clusters increased from 3 to 10. A future goal we could try to implement is plotting the K-means clustering and the cluster centroids to better see the relations. Another goal would be to try different values of k in search of the optimal number of clusters for our dataset.