JOBS FOR ALL

TEAM Galaxy

(Prashant Raghuwanshi & Jay Pfister)

DSC630-T301 Predictive Analytics (2223-1)

Professor Fadi Alsaleem

Milestone 4, Project Proposal

01/29/2022

Each phase of the process:

Template Source : https://www.sv-europe.com/crisp-dm-methodology/

Abstract:

-This term-end project aims to evaluate the Students should be able to identify a business problem to address through predictive analytics. The goal is to select appropriate models and model specifications and apply the respective methods to enhance data-driven decision-making related to the business problem.

Students will identify the potential use of predictive analytics, formulate the problem, identify the right sources of data, analyze data and prescribe actions to improve not only the process of decision making but also the outcome of decisions.

1. Stage One - Determine Business Objectives and Assess the Situation

1.1 Assess the Current Situation

We are targeting a sub set of survey respondents who we have identified as gainfully and regularly employed and potentially in need of financial services such as savings accounts or CDs.

Business Application - The goal of this project is to assist in the marketing of financial services to customers based on self identified demographic data from the Bureau of Labor Statistics, allowing for the creation of pools of demographically sorted candidates for client services. Based on our findings, we can craft tailored survey data to better target ideal customers for our clients. This goal is met by creating a predictive model to determine if a given group of respondents is likely to be a good fit for our clients' financial products and as a result an ideal target for marketing campaigns attempting to solicit customers for those products.

1.1.1. Inventory of resources

List the resources available to the project including:

- Personnel: Prashant, Jay
- Data: ATUS 2019 Respondent Data
- Computing resources: Personal computers
- Software: Jupyter Notebook(Python), R Studio, IBM SPSS

1.1.2. Requirements, assumptions and constraints -

- Major Requirements: A valid a predictive variable to determine viability of a candidate based on survey response in order to validate the output of our predictive model that would then enable us to target specific demographics based on respondent inputs.
- A pool of survey respondents providing demographic data about their employment status as well as details about their willingness to work and relevant experience (previous employment related responses).
- Major Constraints: A possible constraint is a signifcant number of declined responses from those surveyed. Without enough input, we won't be able to make a meaningful prediction based on the data, or lack thereof.
- The model also relies on a candidate's real time behavior, rather than a prepared list of responses. Because of this, there is a need for respondents to respond to the survey candidly and truthfully.

1.1.3.Risks and contingencies

- Risks include potential PII issues with respondents, which can largely be mitigated by anonymized respondent IDs. As long as a particular individual is never personally idenitified the risk of PII information leaking is mitigated.
- Potential for respondents to misrepresent their individual situations in their survey replies. Survey responses are to some extent unverifiable. For example, if a respondent indicates that the reason they remain unemployed is a long term medical issue, we can't verify that claim and that could potentially skew results.
- Lack of suitable candidates for a given application based on survey responses. This can be mitigated by the model indicating viability within the candidate pool.

1.1.4.Terminology

CRISP-DM The Cross-Industry Standard Process for Data-Mining – CRISP-DM is a model of a data mining process used to solve problems by experts. The model identifies the different stages in implementing a data mining project, as described bellow The model proposes the following steps: • Business Understanding – to understand the rules and business objectives of the company. • Understanding Data – to collect and describe data. • Data Preparation – to prepare data for import into the software. • Modeling – to select the modeling technique to be used. • Evaluation – Evaluate the process to see if the technique solves modeling and creating rules. • Deployment – to deploy the system and train its users

1.1.5.Costs and benefits

- Costs:
- Primary cost associated with this project is the time of the people working on it.
- Computing resources for modeling
- Benefits:
- Social benefit of assisting the process of filling employment gaps, and potentially enabling otherwise unemployable people to find employment
- Financial benefit to the company from the ability to sell this product to the target consumer.

1.2 What are the desired outputs of the project?

Business success criteria

- Provide this information to financial institutions for targeting for financial products.

Data mining success criteria

- Finding predictive demographic data from the respondent pool in order to determine what variables are best used for candidate selection.

Produce project plan

- Adhere to CRISP-DM methodology. Data prep Understanding > Data Prep > Modelling > Evaluation > Deployment

progress.png

1.3 What Questions Are We Trying To Answer?

Can we identify survey respondent segments that are candidates for potential packaging as demographic data to sell to our customers?

What traits make an individual most suitable for the financial services for our customers?

Which of the survey respondents are good fits for the financial products of our customers?

2. Stage Two - Data Understanding

The second stage of the CRISP-DM process requires you to acquire the data listed in the project resources. This initial collection includes data loading, if this is necessary for data understanding. For example, if you use a specific tool for data understanding, it makes perfect sense to load your data into this tool. If you acquire multiple data sources then you need to consider how and when you're going to integrate these.

- Our Data set is a survey dataset and it contains the question related to two sections,
- The first section is income/employment
- The second section is related to time use

2.1 Initial Data Report

Initial data collection report - List the data sources acquired together with their locations, the methods used to acquire them and any problems encountered. Record problems you encountered and any resolutions achieved. This will help both with future replication of this project and with the execution of similar future projects.

In [32]:
```
# Import Libraries Required
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import seaborn as sns
pd.set_option("display.max_columns", None)
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
from sklearn.decomposition import PCA
# supress warnings
import warnings
import seaborn as sns
from matplotlib.pyplot import xticks
from sklearn import preprocessing
from kmodes.kmodes import KModes
```

In [33]:
```
#Data source:
#Source Query location:
path =  '2019 BLS.csv'
# reads the data from the file - denotes as CSV, it has no header, sets column headers
df =  pd.read_csv(path, sep=',')
df0 = df.copy()
```

2.2 Describe Data

Data description report - ATUS Respondent File This file contains case-specific variables collected in ATUS (that is, variables for which there is one value for each respondent). These include, for example, labor force and earnings information, total time providing secondary childcare, and ATUS statistical weights.

There is one record for each ATUS respondent.

Below is a simplified example. The variable TUCASEID identifies each household, and the variable TULINENO identifies each individual within the household. The example contains responses from five individuals; note that the respondent always has TULINENO = 1. In the example, each respondent has a corresponding statistical weight for use in generating estimates representative of the U.S. civilian, noninstitutionalized population (TUFINLWGT), and values for school enrollment (TESCHENR), labor force status (TELFS), and total number of minutes spent alone on the diary day (TRTALONE). The actual ATUS Respondent file contains more variables and records.

TUCASEID	TULINENO	TUFINLWGT	TESCHENR	TELFS	TRTALONE
20060101020210	1	22261358.19	1	1	40
20060101020211	1	5019645.31	1	1	350
20060101020212	1	2926068.74	1	5	0
20060101020213	1	25780574.07	2	5	556
20060101020214	1	3414645.94	1	4	100

Income & Employment:

TEERNHRY "Edited: hourly-non-hourly status" / TEERNPER "Edited: for your main job, what is the easiest way for you to report your total earnings before taxes or other deductions: hourl" /
TEERNUOT "Edited: do you usually receive overtime pay, tips, or commissions at your main job?" / TEHRUSL1 "Edited: how many hours per week do you usually work at your main job?" / TEHRUSLT "Edited: total hours usually worked per week [sum of TEHRUSL1 and TEHRUSL2]" / TEIO1COW "Edited: individual class of worker code [main job]" / TEIO1ICD "Edited: industry code [main job]" / TEIO1OCD "Edited: occupation code [main job]" / TELFS "Edited: labor force status" /
TEMJOT "Edited: in the last seven days did you have more than one job?" / TESPEMPNOT "Edited: employment status of spouse or unmarried partner" / TRDPFTPT "Full time or part time employment status of respondent" / TRDTIND1 "Detailed industry recode [main job]" / TRDTOCC1 "Detailed occupation recode [main job]" / TRIMIND1 "Intermediate industry recode [main job]" / TRMJIND1 "Major industry recode [main job]" / TRMJOCC1 "Major occupation recode [main job]" / TRMJOCGR "Major occupation category [main job]" / TTWK "Weekly earnings topcode flag" / TUIO1MFG "Is this business or organization mainly manufacturing, retail trade, wholesale trade, or something else? [main job]"/ TUIODP1 "Last time we spoke to someone in this household, you were reported to work for [employer's name]. TUSPWK "In the last seven days, did your spouse or unmarried partner do any work for pay or profit?" / TUBUS "Does anyone in the household own a business or a farm?" /

In [34]:
```
df.columns
```

Out[34]:
```
Index(['TUCASEID', 'TULINENO', 'TUYEAR', 'TUMONTH', 'TEABSRSN', 'TEERN',
       'TEERNH1O', 'TEERNH2', 'TEERNHRO', 'TEERNHRY',
       ...
       'TXSPEMPNOT', 'TXSPUHRS', 'TXTCC', 'TXTCCTOT', 'TXTCOC', 'TXTHH',
       'TXTNOHH', 'TXTO', 'TXTOHH', 'TXTONHH'],
      dtype='object', length=174)
```

In [35]:
```
df.shape
```

Out[35]:
```
(9435, 174)
```

In [36]:
```
df.dtypes
```

Out[36]:
```
TUCASEID    int64
TULINENO    int64
TUYEAR      int64
TUMONTH     int64
TEABSRSN    int64
            ...
TXTHH       int64
TXTNOHH     int64
TXTO        int64
TXTOHH      int64
TXTONHH     int64
Length: 174, dtype: object
```

In [37]:
```
df.describe()
```

Out[37]:

	TUCASEID	TULINENO	TUYEAR	TUMONTH	TEABSRSN	TEERN	TEERNH1O	TEERNH2	TEERNHRO	TEERNHRY	TEER
count	9.435000e+03	9435.0	9435.0	9435.000000	9435.00000	9435.000000	9435.000000	9435.000000	9435.000000	9435.000000	9435.0
mean	2.019065e+13	1.0	2019.0	6.404981	-0.78124	948.019714	356.827981	221.663911	5.537573	0.337785	1.5
std	3.496214e+08	0.0	0.0	3.478020	1.42014	7882.777017	883.983735	811.009246	14.481475	1.286188	2.8
min	2.019010e+13	1.0	2019.0	1.000000	-1.00000	-1.000000	-1.000000	-1.000000	-2.000000	-1.000000	-1.0
25%	2.019030e+13	1.0	2019.0	3.000000	-1.00000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.0
50%	2.019061e+13	1.0	2019.0	6.000000	-1.00000	-1.000000	-1.000000	-1.000000	-1.000000	1.000000	1.0
75%	2.019091e+13	1.0	2019.0	9.000000	-1.00000	-1.000000	-1.000000	-1.000000	-1.000000	2.000000	5.0
max	2.019121e+13	1.0	2019.0	12.000000	14.00000	225000.000000	9999.000000	9999.000000	99.000000	2.000000	7.0

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9435 entries, 0 to 9434
Columns: 174 entries, TUCASEID to TXTONHH
dtypes: float64(1), int64(171), object(2)
memory usage: 12.5+ MB
```

In [39]:
```
df.head(5)
```

Out[39]:

	TUCASEID	TULINENO	TUYEAR	TUMONTH	TEABSRSN	TEERN	TEERNH1O	TEERNH2	TEERNHRO	TEERNHRY	TEERNPER	TEERNRT
0	20190101190022	1	2019	1	-1	-1	-1	-1	-1	-1	-1	-1
1	20190101190031	1	2019	1	-1	-1	1100	-1	15	1	1	-1
2	20190101190057	1	2019	1	-1	-1	-1	1500	-1	1	2	1
3	20190101190107	1	2019	1	-1	-1	-1	-1	-1	-1	-1	-1
4	20190101190555	1	2019	1	-1	-1	-1	-1	-1	-1	-1	-1

In [40]:
```
#splitting the data sets on required inclome & employment subject area .
df = df[["TEERNHRY", "TEERNPER", "TEERNUOT", "TEHRUSLT", "TEIO1COW", "TEIO1ICD", "TELFS", "TEMJOT", "TESPEMPNOT
```

2.3 Verify Data Quality

Examine the quality of the data, addressing questions such as:

- Is the data complete (does it cover all the cases required)?
- Is it correct, or does it contain errors and, if there are errors, how common are they?
- Are there missing values in the data? If so, how are they represented, where do they occur, and how common are they?

2.3.1. Missing Data

In addition to incorrect datatypes, another common problem when dealing with real-world data is missing values. These can arise for many reasons and have to be either filled in or removed before we train a machine learning model. First, let's get a sense of how many missing values are in each column

While we always want to be careful about removing information, if a column has a high percentage of missing values, then it probably will not be useful to our model. The threshold for removing columns should depend on the problem

In [41]:
```
# checking null value
df.isnull().sum()
```

Out[41]:
```
TEERNHRY      0
TEERNPER      0
TEERNUOT      0
TEHRUSLT      0
TEIO1COW      0
TEIO1ICD      0
TELFS         0
TEMJOT        0
TESPEMPNOT    0
TRDPFTPT      0
TRDTIND1      0
TRDTOCC1      0
TRIMIND1      0
TRMJIND1      0
TRMJOCC1      0
TRMJOCGR      0
TUFWK         0
TUIO1MFG      0
TUIODP1       0
TUSPWK        0
TUBUS         0
dtype: int64
```

In [42]:
```
# finding the columns which contains more missing values (greater than 50% of total records count)
# if data valuye contains -1. -2 , -3 entry then it means the user havent responded the survey question
a = [-1, -2, -3]
pct_null = df.isin(a).sum() / len(df)
missing_features = pct_null[pct_null > 0.50].index
missing_features
```

Out[42]:
```
Index([], dtype='object')
```

In [43]:
```
# dropping the missing value columns
df.drop(missing_features, axis=1, inplace=True)
```

In [44]:
```
# filling out missing values
df = df.fillna(method="ffill")
df = df.fillna(method="bfill")
```

In [45]:
```
df.shape
```

Out[45]:
```
(9435, 21)
```

2.3.2 Removing Duplicate columns

In [46]:
```
# This function take a dataframe
# as a parameter and returning list
# of column names whose contents
# are duplicates.
def getDuplicateColumns(df):

    # Create an empty set
    duplicateColumnNames = set()

    # Iterate through all the columns
    # of dataframe
    for x in range(df.shape[1]):

        # Take column at xth index.
        col = df.iloc[:, x]

        # Iterate through all the columns in
        # DataFrame from (x + 1)th index to
        # last index
        for y in range(x + 1, df.shape[1]):

            # Take column at yth index.
            otherCol = df.iloc[:, y]

            # Check if two columns at x & y
            # index are equal or not,
            # if equal then adding
            # to the set
            if col.equals(otherCol):
                duplicateColumnNames.add(df.columns.values[y])

    # Return list of unique column names
    # whose contents are duplicates.
    return list(duplicateColumnNames)
```

In [47]:
```
drop_columns2= getDuplicateColumns(df)
# list duplicate data columns
drop_columns2
```

Out[47]:
```
[]
```

In [48]:
```
# drop columns from df whose of values are duplicate
df.drop(drop_columns2, axis=1, inplace=True)
```

In [49]:
```
def rep_data(df1):
    for col in df1.columns:
        df1[col] = df1[col].map( lambda x : df1[col].median() if x == -1 else x)
        df1[col] = df1[col].map( lambda x : df1[col].median() if x == -2 else x)
        df1[col] = df1[col].map( lambda x : df1[col].median() if x == -3 else x)
        df1[col] = df1[col].map( lambda x : df1[col].median() if x == -4 else x)
```

In [50]:
```
# replacing missing value with median
rep_data(df)
```

3. Stage Three - Data Preperation

This is the stage of the project where you decide on the data that you're going to use for analysis. The criteria you might use to make this decision include the relevance of the data to your data mining goals, the quality of the data, and also technical constraints such as limits on data volume or data types. Note that data selection covers selection of attributes (columns) as well as selection of records (rows) in a table.

3.1 Label Encoding

In [51]:
```
df.dtypes
```

Out[51]:
```
TEERNHRY      float64
TEERNPER      float64
TEERNUOT      float64
TEHRUSLT      float64
TEIO1COW      float64
TEIO1ICD      float64
TELFS           int64
TEMJOT        float64
TESPEMPNOT    float64
TRDPFTPT      float64
TRDTIND1      float64
TRDTOCC1      float64
TRIMIND1      float64
TRMJIND1      float64
TRMJOCC1      float64
TRMJOCGR      float64
TUFWK         float64
TUIO1MFG      float64
TUIODP1       float64
TUSPWK        float64
TUBUS         float64
dtype: object
```

In [52]:
```
# Creating copy data set
df_employment = df.copy()
```

In [53]:
```
df_employment1 = df_employment.copy()
```

3.2 One Hot Encoding

- Here we used One Hot Encoding to help with our largely categorical data set. The purpose of One Hot Encoding is to map various categorical variables like the ones in our data to integers that can be better used and understood by machine learning algorithms. This is particularly useful when working with data that may not all be related. It also helps when working with integers because of the way machine learning treats number order (Higher is better), as well as allowing for better scalability.

In [54]:
```
# get wholeset of dummy variable for all columns
def one_hot_top_x(df2, variable, top_x_labels):
    for label in top_x_labels:
        df2[variable+'_'+str(label)] = np.where(df2[variable] == label, 1, 0)
```

In [55]:
```
# hot encoding for top 5 variables from each columns
top_5 = [x for x in df_employment1.TEERNPER.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TEERNPER', top_5)
top_5 = [x for x in df_employment1.TEHRUSLT.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TEHRUSLT', top_5)
top_5 = [x for x in df_employment1.TEIO1COW.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TEIO1COW', top_5)
top_5 = [x for x in df_employment1.TEIO1ICD.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TEIO1ICD', top_5)
top_5 = [x for x in df_employment1.TELFS.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TELFS', top_5)
top_5 = [x for x in df_employment1.TRDTIND1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRDTIND1', top_5)
top_5 = [x for x in df_employment1.TRDTOCC1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRDTOCC1', top_5)
top_5 = [x for x in df_employment1.TRIMIND1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRIMIND1', top_5)
top_5 = [x for x in df_employment1.TRMJIND1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRMJIND1', top_5)
top_5 = [x for x in df_employment1.TRMJOCC1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRMJOCC1', top_5)
top_5 = [x for x in df_employment1.TRDTOCC1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRDTOCC1', top_5)
top_5 = [x for x in df_employment1.TRMJOCGR.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRMJOCGR', top_5)
top_5 = [x for x in df_employment1.TEERNHRY.value_counts().sort_values(ascending=False).head(2).index]
one_hot_top_x(df_employment1, 'TEERNHRY', top_5)
top_5 = [x for x in df_employment1.TRMJOCGR.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TRMJOCGR', top_5)
top_5 = [x for x in df_employment1.TUFWK.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TUFWK', top_5)
top_5 = [x for x in df_employment1.TUIO1MFG.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TUIO1MFG', top_5)
top_5 = [x for x in df_employment1.TUIODP1.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TUIODP1', top_5)
top_5 = [x for x in df_employment1.TUSPWK.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TUSPWK', top_5)
top_5 = [x for x in df_employment1.TUBUS.value_counts().sort_values(ascending=False).head(5).index]
one_hot_top_x(df_employment1, 'TUBUS', top_5)
```

In [56]:
```
# drop the main columns
dcol = ['TEERNHRY', 'TEERNPER', 'TEERNUOT', 'TEHRUSLT', 'TEIO1COW', 'TEIO1ICD', 'TELFS', 'TEMJOT', 'TESPEMPNOT'
df_employment2 = df_employment1.drop(columns = dcol,  axis = 1)
df_employment2.shape
```

Out[56]:
```
(9435, 75)
```

In [57]:
```
# validating hot encoding
#for cols in df_employment2.columns:
#     print(cols, len(df_employment2[cols].unique()), df_employment2[cols].unique())
```

3.2.2 Drop Unnecessary Columns

- Sometimes we may not need certain columns. We can drop extra columns in order to keep only relevent data and avoid potentially spurious associations.

In [58]:
```
# find out columns with bias data values
dtl2 = []
def bias_data1(df1):
    for col1 in df1.columns:
        dt1 = df1[col1].value_counts()
        dt3 = pd.DataFrame(dt1)
        dtl2.append(dt3/9435)
        dt1 = ''
```

In [59]:
```
bias_data1(df_employment2)
# after analysing the biased data results, we found below columns contains the biased categorical data vaues (c
#['TEERNPER_2.0', 'TEERNPER_3.0', 'TEERNPER_5.0', 'TEHRUSLT_50.0', 'TEHRUSLT_60.0', 'TEIO1COW_5.0', 'TEIO1COW_
#       'TEIO1COW_7.0', 'TEIO1COW_2.0', 'TEIO1ICD_7860.0', 'TEIO1ICD_770', 'TEIO1ICD_8190.0', 'TEIO1ICD_8680.0
#       'TUSPWK_4.0', 'TUSPWK_5.0', 'TUIO1MFG_3.0', 'TUFWK_5.0', 'TUFWK_3.0', 'TELFS_3', 'TELFS_4']
```

In [63]:
```
dcols =  ['TEERNPER_2.0', 'TEERNPER_3.0', 'TEERNPER_5.0', 'TEHRUSLT_50.0', 'TEHRUSLT_60.0', 'TEIO1COW_5.0', 'TE
       'TEIO1COW_7.0', 'TEIO1COW_2.0', 'TEIO1ICD_7860.0', 'TEIO1ICD_8190.0', 'TEIO1ICD_8680.0', 'TELFS_2', 'TEI
       'TUSPWK_4.0', 'TUSPWK_5.0', 'TUIO1MFG_3.0', 'TUFWK_5.0', 'TUFWK_3.0', 'TELFS_3', 'TELFS_4']
df_employment3 = df_employment2.drop(columns = dcols,  axis = 1)
df_employment3.shape
```

Out[63]:
```
(9435, 54)
```

In [64]:
```
#df_employment3.head()
```

3.2 Dealing With Zeros

- Replacing all the zeros from columns.
- Zero values were previously addressed using mean value imputation. Some of our data contained 0's or other indicated non values. We chose to use Mean Value Imputation to account for these zeroes. Mean Value Imputation does exactly what it sounds like, it replaces the non-values in our data, for example 0, with the mean value of all the values in the column the data is found in.

4. Stage Four - Exploratory Data Analysis

4.1. Outliers

At this point, we may also want to remove outliers. These can be due to typos in data entry, mistakes in units, or they could be legitimate but extreme values. For this project, we will remove anomalies based on the definition of extreme outliers:

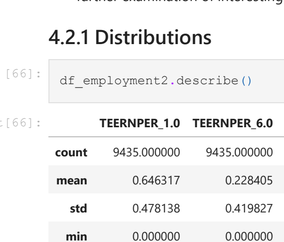
https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm

- Below the first quartile − 3 ∗ interquartile range
- Above the third quartile + 3 ∗ interquartile range

In [65]:
```
# showing boxplot & histogram of each features before removing outliers
for column in df_employment3:
    plt.figure()
    # creating a figure composed of two matplotlib.Axes objects (ax_box and ax_hist)
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (.15, .85)})

   # assigning a graph to each ax
    sns.boxplot(df[column], ax=ax_box)
    sns.histplot(data=df, x=column, ax=ax_hist)

    # Remove x axis name for the boxplot
    ax_box.set(xlabel='')
    # comment out break statment to generate plots for al features
    break
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Roaming\Python\Python38\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, toler
ance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

~\AppData\Roaming\Python\Python38\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_lo
c()

~\AppData\Roaming\Python\Python38\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_lo
c()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'TEERNPER_1.0'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-65-5208298aa5e7> in <module>
      6
      7    # assigning a graph to each ax
----> 8     sns.boxplot(df[column], ax=ax_box)
      9     sns.histplot(data=df, x=column, ax=ax_hist)
     10

~\AppData\Roaming\Python\Python38\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3456             if self.columns.nlevels > 1:
   3457                 return self._getitem_multilevel(key)
-> 3458             indexer = self.columns.get_loc(key)
   3459             if is_integer(indexer):
   3460                 indexer = [indexer]

~\AppData\Roaming\Python\Python38\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, toler
ance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'TEERNPER_1.0'
```

```
<Figure size 432x288 with 0 Axes>
```



- After doing hot encoding & removing columns with bias records,
- Now all of our columns contain 0 or 1, so we dont required outlayer deduction and historgram plot

4.2 Initial Data Exploration

During this stage you'll address data mining questions using querying, data visualization and reporting techniques. These may include:

- **Distribution** of key attributes (for example, the target attribute of a prediction task)
- **Relationships** between pairs or small numbers of attributes
- Results of **simple aggregations**
- **Properties** of significant sub-populations
- **Simple** statistical analyses

These analyses may directly address your data mining goals. They may also contribute to or refine the data description and quality reports, and feed into the transformation and other data preparation steps needed for further analysis.

- **Data exploration report** - Describe results of your data exploration, including first findings or initial hypothesis and their impact on the remainder of the project. If appropriate you could include graphs and plots here to indicate data characteristics that suggest further examination of interesting data subsets.

4.2.1 Distributions

In [66]:
```
df_employment2.describe()
```

Out[66]:

	TEERNPER_1.0	TEERNPER_6.0	TEERNPER_2.0	TEERNPER_3.0	TEERNPER_5.0	TEHRUSLT_25.0	TEHRUSLT_40.0	TEHRUSLT_50.0	TEHRUSLT
count	9435.000000	9435.000000	9435.000000	9435.000000	9435.000000	9435.00000	9435.000000	9435.000000	9435.00
mean	0.646317	0.228405	0.045681	0.045045	0.019926	0.43964	0.232962	0.053630	0.03
std	0.478138	0.419827	0.208803	0.207414	0.139753	0.49637	0.422741	0.225298	0.19
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00
75%	1.000000	0.000000	0.000000	0.000000	0.000000	1.00000	0.000000	0.000000	0.00
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.00

4.2.2 Correlations

Can we derive any correlation from this data-set? Pairplot chart gives us correlations, distributions and regression path Correlogram are awesome for exploratory analysis. It allows to quickly observe the relationship between every variable of your matrix. It is easy to do it with seaborn: just call the pairplot function

Pairplot Documentation can be found here: https://seaborn.pydata.org/generated/seaborn.pairplot.html

In [67]:
```
#corr = df_employment.apply(lambda x : pd.factorize(x)[0]).corr(method='pearson', min_periods=1)
#corr.style.background_gradient(cmap='coolwarm')

import numpy as np
import pandas as pd
import scipy.stats as ss
import seaborn as sns

def cramers_v(confusion_matrix):
    """ calculate Cramers V statistic for categorial-categorial association.
        uses correction from Bergsma and Wicher,
        Journal of the Korean Statistical Society 42 (2013): 323-328
    """
    chi2 = ss.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))
```

In [68]:
```
factors_paired = [(i,j) for i in df_employment2.columns.values for j in df_employment2.columns.values]

chi2, p_values =[], []

for f in factors_paired:
    if f[0] != f[1]:
        confusion_matrix = pd.crosstab(df_employment2[f[0]], df_employment2[f[1]])
        chitest = cramers_v(confusion_matrix.values)
        chi2.append(chitest)
    else:       # for same factor pair
        chi2.append(1)
```

In [69]:
```
chi2 = np.array(chi2).reshape((75,75))
```

In [70]:
```
chi2 = pd.DataFrame(chi2, index=df_employment2.columns.values, columns=df_employment2.columns.values) # then a
```

In [71]:
```
#chi2= p values
chi2.style.background_gradient(cmap='coolwarm')
```

Out[71]:

	TEERNPER_1.0	TEERNPER_6.0	TEERNPER_2.0	TEERNPER_3.0	TEERNPER_5.0	TEHRUSLT_25.0	TEHRUSLT_40.0	TEHRUSLT_50.0
TEERNPER_1.0	1.000000	0.735187	0.295063	0.292895	0.191691	0.576347	0.325572	0.189185
TEERNPER_6.0	0.735187	1.000000	0.117989	0.117111	0.075984	0.427135	0.254514	0.182306
TEERNPER_2.0	0.295063	0.117989	1.000000	0.045137	0.027518	0.163306	0.065357	0.042452
TEERNPER_3.0	0.292895	0.117111	0.045137	1.000000	0.027261	0.171999	0.100411	0.000000
TEERNPER_5.0	0.191691	0.075984	0.027518	0.027261	1.000000	0.106706	0.070491	0.005136
TEHRUSLT_25.0	0.576347	0.427135	0.163306	0.171999	0.106706	1.000000	0.487810	0.210142
TEHRUSLT_40.0	0.325572	0.254514	0.065357	0.100411	0.070491	0.487810	1.000000	0.130236
TEHRUSLT_50.0	0.189185	0.182306	0.042452	0.000000	0.005136	0.210142	0.130236	1.000000
TEHRUSLT_45.0	0.149678	0.149754	0.027255	0.000000	0.000000	0.175824	0.108789	0.045098
TEHRUSLT_60.0	0.092265	0.079504	0.029840	0.000000	0.017777	0.136091	0.083897	0.033780
TEIO1COW_4.0	0.167650	0.150416	0.043461	0.048876	0.101198	0.369523	0.159014	0.063003
TEIO1COW_5.0	0.138900	0.118868	0.000000	0.058065	0.015821	0.172706	0.073374	0.000000
TEIO1COW_3.0	0.172981	0.135278	0.000000	0.051190	0.106081	0.165236	0.095002	0.000000
TEIO1COW_7.0	0.154562	0.113241	0.043514	0.043160	0.026124	0.101257	0.039100	0.028847
TEIO1COW_2.0	0.176139	0.129820	0.000000	0.044917	0.131121	0.150560	0.107703	0.022703
TEIO1ICD_3895.0	0.593708	0.436602	0.172819	0.174641	0.113957	0.905722	0.441791	0.190605
TEIO1ICD_7860.0	0.210843	0.160282	0.000000	0.018450	0.165398	0.160593	0.055429	0.020128
TEIO1ICD_770.0	0.009645	0.006087	0.081084	0.006835	0.016777	0.128613	0.066031	0.060453
TEIO1ICD_8190.0	0.076522	0.065601	0.000000	0.061317	0.009424	0.140993	0.076592	0.000000
TEIO1ICD_8680.0	0.000000	0.035394	0.055509	0.049408	0.000000	0.094335	0.004971	0.000000
TELFS_1	0.560600	0.412841	0.167806	0.167343	0.096072	0.858152	0.419726	0.183800
TELFS_5	0.561093	0.412524	0.165196	0.163976	0.106931	0.856532	0.417863	0.179883
TELFS_2	0.081990	0.057602	0.010998	0.007355	0.044167	0.119049	0.053777	0.008675
TELFS_4	0.118883	0.086873	0.032292	0.032008	0.018000	0.182218	0.088027	0.035682
TELFS_3	0.031843	0.021332	0.000000	0.000000	0.000000	0.051347	0.021729	0.000000
TRDTIND1_19.0	0.584779	0.429195	0.171087	0.171827	0.114859	0.898435	0.435016	0.190181
TRDTIND1_40.0	0.243624	0.199855	0.003594	0.022337	0.166268	0.194790	0.064648	0.019263
TRDTIND1_36.0	0.174720	0.203277	0.020589	0.018546	0.000000	0.183187	0.125186	0.089781
TRDTIND1_22.0	0.043706	0.000000	0.053070	0.037249	0.000000	0.159249	0.046776	0.000000
TRDTIND1_42.0	0.075660	0.035869	0.000000	0.078742	0.014787	0.150865	0.062956	0.000000
TRDTOCC1_2.0	0.527073	0.363709	0.175352	0.169732	0.116149	0.839069	0.396392	0.169275
TRDTOCC1_1.0	0.203510	0.215098	0.008145	0.000000	0.011522	0.228875	0.078123	0.136575
TRDTOCC1_17.0	0.095883	0.047424	0.050412	0.032469	0.038808	0.203430	0.154200	0.010642
TRDTOCC1_16.0	0.099141	0.065183	0.018237	0.041652	0.018643	0.158096	0.036304	0.046931
TRDTOCC1_8.0	0.201536	0.157470	0.000000	0.025261	0.126547	0.154479	0.043003	0.012812
TRIMIND1_5.0	0.571595	0.421990	0.161168	0.169032	0.110746	0.870255	0.415909	0.190480
TRIMIND1_16.0	0.111375	0.067092	0.003159	0.115006	0.011048	0.238138	0.118054	0.000000
TRIMIND1_15.0	0.243624	0.199855	0.003594	0.022337	0.166268	0.194790	0.064648	0.019263
TRIMIND1_13.0	0.174720	0.203277	0.020589	0.018546	0.000000	0.183187	0.125186	0.089781
TRIMIND1_7.0	0.043706	0.000000	0.053070	0.037249	0.000000	0.159249	0.046776	0.000000
TRMJIND1_4.0	0.513428	0.370515	0.139500	0.158191	0.109642	0.790920	0.365759	0.166381
TRMJIND1_10.0	0.255778	0.191116	0.013104	0.107853	0.127134	0.321631	0.138111	0.010477
TRMJIND1_9.0	0.164206	0.167769	0.000000	0.027606	0.000000	0.219906	0.143372	0.081246
TRMJIND1_5.0	0.077908	0.023657	0.061092	0.038592	0.014756	0.184833	0.070766	0.026974
TRMJIND1_8.0	0.135565	0.120992	0.037680	0.009630	0.000000	0.163254	0.093554	0.073065
TRMJOCC1_1.0	0.408351	0.240098	0.165789	0.165800	0.105602	0.703199	0.348399	0.090737
TRMJOCC1_2.0	0.351799	0.325023	0.013006	0.069347	0.096276	0.343072	0.197871	0.073062
TRMJOCC1_3.0	0.032650	0.054826	0.057749	0.110537	0.028832	0.211800	0.046559	0.000000
TRMJOCC1_5.0	0.095883	0.047424	0.050412	0.032469	0.038808	0.203430	0.154200	0.010642
TRMJOCC1_4.0	0.099141	0.065183	0.018237	0.041652	0.018643	0.158096	0.036304	0.046931
TRMJOCGR_1.0	0.149547	0.000000	0.193808	0.120639	0.032081	0.476077	0.212346	0.035662
TRMJOCGR_3.0	0.142575	0.082292	0.053957	0.055461	0.045657	0.265868	0.145503	0.017721
TRMJOCGR_2.0	0.032650	0.054826	0.057749	0.110537	0.028832	0.211800	0.046559	0.000000
TRMJOCGR_6.0	0.030944	0.045602	0.133315	0.028196	0.006395	0.179211	0.083146	0.012271
TRMJOCGR_5.0	0.025994	0.012050	0.103602	0.000000	0.022023	0.149175	0.084784	0.043453
TEERNHRY_1.0	0.792818	0.676556	0.137018	0.120952	0.163654	0.459643	0.241449	0.212032
TEERNHRY_2.0	0.792818	0.676556	0.137018	0.120952	0.163654	0.459643	0.241449	0.212032
TUFWK_1.0	0.567016	0.418846	0.167549	0.170146	0.097948	0.848995	0.423798	0.184014
TUFWK_2.0	0.493558	0.364736	0.146156	0.149021	0.082985	0.737409	0.368947	0.159642
TUFWK_3.0	0.161911	0.119441	0.043644	0.045740	0.027942	0.244225	0.121008	0.050564
TUFWK_4.0	0.069566	0.048435	0.017044	0.016832	0.003479	0.107807	0.051739	0.019520
TUFWK_5.0	0.009662	0.004242	0.000000	0.000000	0.000000	0.027809	0.000000	0.000000
TUIO1MFG_2.0	0.551785	0.437285	0.111959	0.145761	0.113603	0.778682	0.413295	0.170182
TUIO1MFG_4.0	0.474133	0.374258	0.072428	0.136583	0.107302	0.671020	0.343082	0.143345
TUIO1MFG_1.0	0.142980	0.123922	0.070202	0.005325	0.000000	0.194006	0.118244	0.041538
TUIO1MFG_3.0	0.053741	0.025623	0.024224	0.008472	0.017598	0.085183	0.069639	0.026719
TUIODP1_1.0	0.061895	0.026648	0.049456	0.014060	0.000000	0.136343	0.066513	0.033269
TUIODP1_2.0	0.061895	0.026648	0.049456	0.014060	0.000000	0.136343	0.066513	0.033269
TUSPWK_1.0	0.062832	0.045013	0.000000	0.029555	0.005810	0.111725	0.071449	0.000000
TUSPWK_2.0	0.016461	0.008250	0.000000	0.013358	0.000000	0.047989	0.039795	0.013272
TUSPWK_3.0	0.098442	0.072313	0.026152	0.029216	0.018430	0.142216	0.075177	0.033234
TUSPWK_4.0	0.029813	0.024311	0.000000	0.000000	0.000000	0.050829	0.020212	0.001401
TUSPWK_5.0	0.000000	0.000000	0.000000	0.000000	0.011567	0.000000	0.016088	0.000000
TUBUS_2.0	0.018711	0.000000	0.000000	0.040964	0.000000	0.140870	0.045424	0.071476
TUBUS_1.0	0.018711	0.000000	0.000000	0.040964	0.000000	0.140870	0.045424	0.071476

- Above is a correlation table of our variables using a heat map style visualization to help identify levels of correlation between variables. In the above map, strongly correlated variables are colored red and uncorrelated or very weakly correlated variables are colored blue.

4.3 Data Quality Report

List the results of the data quality verification. If quality problems exist, suggest possible solutions. Solutions to data quality problems generally depend heavily on both data and business knowledge.

- Primary data quality issue is missing values in certain parts of the data. To correct for this, we imputed the mean value of the columns into those missing fields in order to have a more complete approximation of the missing data.

5. Stage Four - Modelling

As the first step in modelling, you'll select the actual modelling technique that you'll be using. Although you may have already selected a tool during the business understanding phase, at this stage you'll be selecting the specific modelling technique e.g. decision-tree building with C5.0, or neural network generation with back propagation. If multiple techniques are applied, perform this task separately for each technique.

5.1. Modelling technique

We opted for an unsupervised learning approach to our data, specifically clustering. We chose this method because with unsupervised learning, you can find relationships in data that aren't immmediately apparent. It does this by comparing the data based on similarity. In clustering, this is used to group the unlabeled data into clusters that represent patterns that were found in the data.

Our current focus is using K – Mode clustering, an extension of K-Means clustering, to determine the relationships that exist between our variables by presenting them as a group of visually clustered data points which reveal relationships not immediately apparent from the data set. The number of clusters is determined by the K value provided to the method, with several ways to determine the correct K value including an Elbow Chart or a Silhouette Graphs that can provide a visual representation of ideal K values for our data set.

We chose K-Mode clustering because of the nature of our data. K-Mode clusters are extremely efficient when working with large amounts of categorical data. K-Mode doesn't need to calculate all of the pair wise distances between data points. Additionally, it is distribution free, meaning that it does not require imposing distribution assumptions on the data in order to work.

K-Means clustering on the other hand does not work well with categorical data sets, primarily because it has the issues above that K-Mode does not, and ultimately it's objective function simply doesn't work with categorical data.

In [72]:
```
# First we will keep a copy of data
df_employment_copy = df_employment2.copy()
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=2, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(df_employment2)
```
```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 4329, cost: 55055.0
Run 1, iteration: 2/100, moves: 1321, cost: 53190.0
Run 1, iteration: 3/100, moves: 8, cost: 53190.0
```

In [73]:
```
# Predicted Clusters
fitClusters_cao
```
Out[73]:
```
array([0, 1, 1, ..., 1, 1, 1], dtype=uint16)
```

In [74]:
```
clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
clusterCentroidsDf.columns = df_employment2.columns
```

In [75]:
```
# Mode of the clusters
clusterCentroidsDf
```
Out[75]:

	TEERNPER_1.0	TEERNPER_6.0	TEERNPER_2.0	TEERNPER_3.0	TEERNPER_5.0	TEHRUSLT_25.0	TEHRUSLT_40.0	TEHRUSLT_50.0	TEHRUSLT_45.(
0	1	0	0	0	0	1	0	0	
1	0	0	0	0	0	0	0	0	

In [76]:
```
# Using K-Mode with "Huang" initialization
km_huang = KModes(n_clusters=2, init = "Huang", n_init = 1, verbose=1)
fitClusters_huang = km_huang.fit_predict(df_employment2)
```
```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 531, cost: 53190.0
Run 1, iteration: 2/100, moves: 4, cost: 53190.0
```

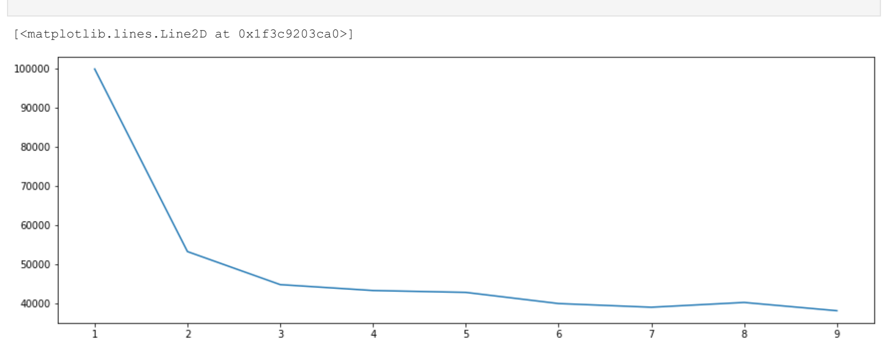
In [77]:
```
# Predicted clusters
fitClusters_huang
```
Out[77]:
```
array([1, 0, 0, ..., 0, 0, 0], dtype=uint16)
```

Building the clusters using K-Mode clustering.

- Clustering is a means of grouping data based on characteristics. The clusters will form based on relationships between the individual data points and potentially reveal new relationships that were not previously apparent.
- Clustering also allows for data compression when working with large data sets. After the data is clustered it can be referred to by cluster ID rather than the individual data points themselves.
- Another use of clustering is the preserve the privacy of the respondent. Data grouped in clusters is practically anonymized by its presence in the cluster and future reference by cluster ID. In our example, no individual respondent ID will need to be used going forward as we can refer to the groups by their cluster IDs.
- Here we used K-Mode clustering, which is similar to K - Means clustering but is based on the the Mode of K. K-Mode clusters are a good choice when working with large amounts of categorical data like what is present in our data set.

In [78]:
```
# Choosing K by comparing Cost against each K
cost = []
for num_clusters in list(range(1,10)):
    kmode = KModes(n_clusters=num_clusters, init = "Huang", n_init = 1, verbose=1)
    kmode.fit_predict(df_employment2)
    cost.append(kmode.cost_)
```
```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 99908.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1860, cost: 53190.0
Run 1, iteration: 2/100, moves: 10, cost: 53190.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1710, cost: 44729.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 2158, cost: 43221.0
Run 1, iteration: 2/100, moves: 754, cost: 43221.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1731, cost: 42743.0
Run 1, iteration: 2/100, moves: 97, cost: 42743.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 3121, cost: 39888.0
Run 1, iteration: 2/100, moves: 147, cost: 39888.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 2994, cost: 40639.0
Run 1, iteration: 2/100, moves: 761, cost: 40006.0
Run 1, iteration: 3/100, moves: 392, cost: 38945.0
Run 1, iteration: 4/100, moves: 16, cost: 38945.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1680, cost: 40329.0
Run 1, iteration: 2/100, moves: 306, cost: 40170.0
Run 1, iteration: 3/100, moves: 39, cost: 40170.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1946, cost: 38051.0
Run 1, iteration: 2/100, moves: 3, cost: 38051.0
```

In [79]:
```
#For KModes, plot cost for a range of K values. Cost is the sum of all the dissimilarities between the clusters
# Select the K where you observe an elbow-like bend with a lesser cost value.
plt.subplots(figsize = (15,5))
y = np.array([i for i in range(1,10,1)])
plt.plot(y,cost)
```
Out[79]:
```
[<matplotlib.lines.Line2D at 0x1f3c9203ca0>]
```



- Here we used what is called the Elbow Method for determining the optimal value of 'k' which will also determine the number of clusters for our purposes. When examining the chart visually, one looks for the point at which the line makes an angle similar to that of an arm bent at the elbow. In our case there are a few bends, but the one with the least dispersion associated with it is at K=3, and as such that was the value we went forward with.

In [80]:
```
## Choosing K=3
km_cao = KModes(n_clusters=3, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(df_employment2)
```
```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 4295, cost: 48781.0
Run 1, iteration: 2/100, moves: 1249, cost: 48722.0
Run 1, iteration: 3/100, moves: 8, cost: 48722.0
```

In [81]:
```
fitClusters_cao.shape
```
Out[81]:
```
(9435,)
```

In [82]:
```
# Combining the predicted clusters with the original DF
df_employment2 = df_employment_copy.reset_index()
```

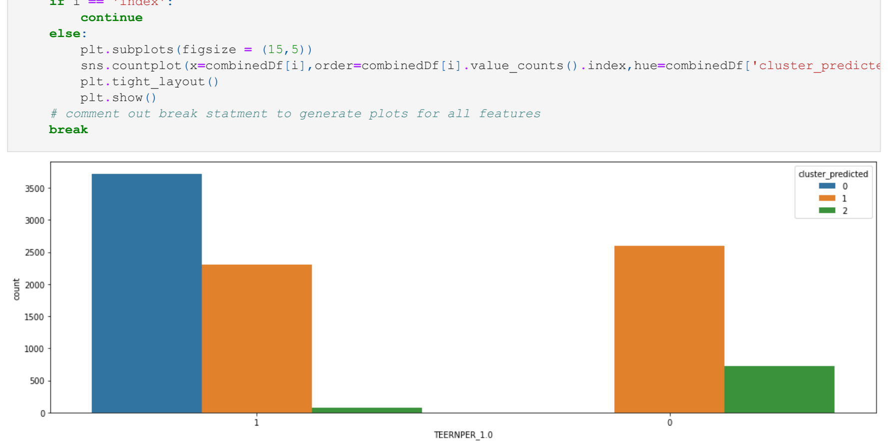
In [83]:
```
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([df_employment2, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index', 'level_0'], axis = 1)
```

In [84]:
```
combinedDf.head()
```
Out[84]:

	TEERNPER_1.0	TEERNPER_6.0	TEERNPER_2.0	TEERNPER_3.0	TEERNPER_5.0	TEHRUSLT_25.0	TEHRUSLT_40.0	TEHRUSLT_50.0	TEHRUSLT_45.(
0	1	0	0	0	0	1	0	0	
1	1	0	0	0	0	0	0	0	
2	0	0	1	0	0	0	0	0	
3	1	0	0	0	0	1	0	0	
4	1	0	0	0	0	1	0	0	

In [85]:
```
# Cluster Identification
cluster_0 = combinedDf[combinedDf['cluster_predicted'] == 0]
cluster_1 = combinedDf[combinedDf['cluster_predicted'] == 1]
cluster_2 = combinedDf[combinedDf['cluster_predicted'] == 2]
```

In [86]:
```
## Plotting the predicted cluster for first feature only
"""
i = 'TEERNPER_1.0'
plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf[i],order=combinedDf[i].value_counts().index,hue=combinedDf['cluster_predicted'])
plt.tight_layout()
plt.show()
"""
## Plotting the predicted cluster for all feature
for i in df_employment3.columns.values:
    if i == 'index':
        continue
    else:
        plt.subplots(figsize = (15,5))
        sns.countplot(x=combinedDf[i],order=combinedDf[i].value_counts().index,hue=combinedDf['cluster_predicte
        plt.tight_layout()
        plt.show()
    # comment out break statment to generate plots for all features
    break
```



- As the above plot shows, per predicted feature the survey repondent with hourly earnings are groupped in cluster 0 while cluster 1 contains the mix of respondent with an hourly earning & non hourly earnings.

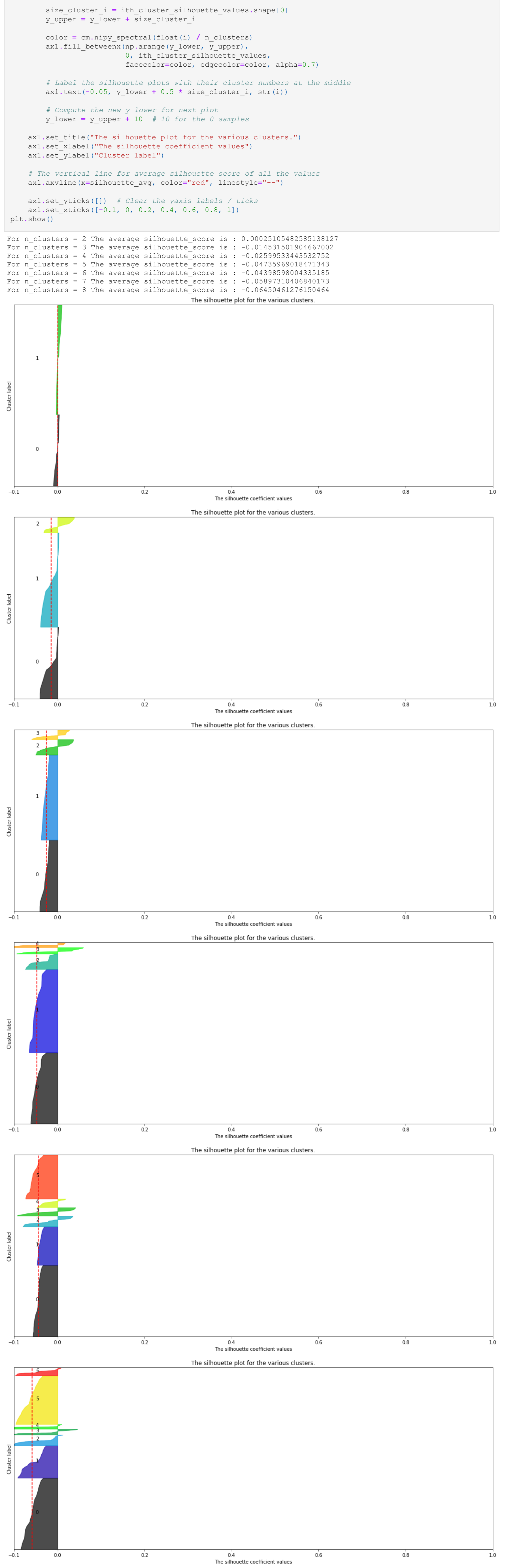
5.4. Assess Model

The silhouette Method is also a method to find the optimal number of clusters and interpretation and validation of consistency within clusters of data. The silhouette method computes silhouette coefficients of each point that measure how much a point is similar to its own cluster compared to other clusters. by providing a succinct graphical representation of how well each object has been classified.

Model assessment - Summarise the results of this task, list the qualities of your generated models (e.g.in terms of accuracy) and rank their quality in relation to each other.

Revised parameter settings - According to the model assessment, revise parameter settings and tune them for the next modelling run. Iterate model building and assessment until you strongly believe that you have found the best model(s). Document all such revisions and assessments.

In [87]:
```
X = df_employment2
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
#X = df_employment
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1) = plt.subplots(1)
    #fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KModes(n_clusters=n_clusters, random_state=10)
    #clusterer = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
plt.show()
```
```
For n_clusters = 2 The average silhouette_score is : 0.00025105482585138127
For n_clusters = 3 The average silhouette_score is : -0.014531501904667002
For n_clusters = 4 The average silhouette_score is : -0.02599533443532752
For n_clusters = 5 The average silhouette_score is : -0.04735969018471343
For n_clusters = 6 The average silhouette_score is : -0.04398598004335185
For n_clusters = 7 The average silhouette_score is : -0.05897310406840173
For n_clusters = 8 The average silhouette_score is : -0.06450461276150464
```



- Silhouette analysis can be used to study the separation distance between clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters.
- Observations from above Silhouette Plots:
- The silhouette plot shows that the n_cluster value of 3 is a bad pick, as all the points in the cluster with cluster_label=0 are below-average silhouette scores.
- The silhouette plot shows that the n_cluster value of 5 is a bad pick, as all the points in the cluster with cluster_label=2 and 4 are below-average silhouette scores.
- The silhouette plot shows that the n_cluster value of 6 is a bad pick, as all the points in the cluster with cluster_label=1,2,4 and 5 are below-average silhouette scores, and also due to the presence of outliers.
- Silhouette analysis is more ambivalent in deciding between 2 and 4.
- The thickness of the silhouette plot for the cluster with cluster_label=1 when n_clusters=2, is bigger in size owing to the grouping of the 3 sub-clusters into one big cluster.
- For n_clusters=4, all the plots are more or less of similar thickness and hence are of similar sizes, as can be considered as best 'k'.

In [88]:
```
#extract features and target variables
x = combinedDf.drop(columns="cluster_predicted")
y = combinedDf["cluster_predicted"]
#save the feature name and target variables
feature_names = x.columns
labels = y.unique()
#split the dataset
from sklearn.model_selection import train_test_split
X_train, test_x, y_train, test_lab = train_test_split(x,y,
                                                      test_size = 0.4,
                                                      random_state = 42)
```

- In the above cell, the data is being divided into multiple sub sets in order to both train and then subsequently test the our modeling solution against. This separation into multiple sets is in an effort to avoid the possibility of over fitting our model against the training set and thus skewing our results.

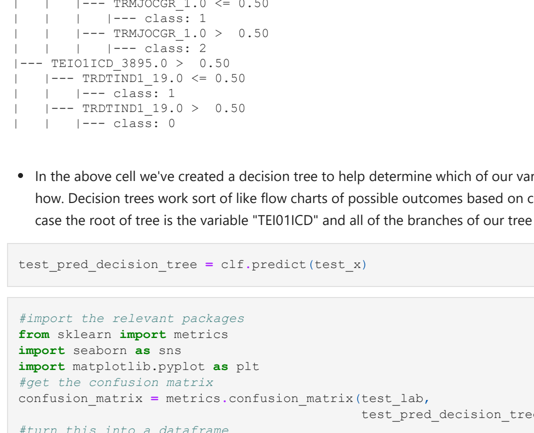
In [89]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
clf.fit(X_train, y_train)
```
Out[89]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [90]:
```
#import relevant functions
from sklearn.tree import export_text
#export the decision rules
tree_rules = export_text(clf,
                        feature_names = list(feature_names))
#print the result
print(tree_rules)
```
```
|--- TEIO1ICD_3895.0 <= 0.50
|   |--- TRIMIND1_15.0 <= 0.50
|   |   |--- TRMJOCC1_2.0 <= 0.50
|   |   |   |--- class: 1
|   |   |--- TRMJOCC1_2.0 >  0.50
|   |   |   |--- class: 1
|   |--- TRIMIND1_15.0 >  0.50
|   |   |--- TRMJOCGR_1.0 <= 0.50
|   |   |   |--- class: 1
|   |   |--- TRMJOCGR_1.0 >  0.50
|   |   |   |--- class: 2
|--- TEIO1ICD_3895.0 >  0.50
|   |--- TRDTIND1_19.0 <= 0.50
|   |   |--- class: 1
|   |--- TRDTIND1_19.0 >  0.50
|   |   |--- class: 0
```

- In the above cell we've created a decision tree to help determine which of our variables are associated with our other variables and how. Decision trees work sort of like flow charts of possible outcomes based on conditions that all start out at one root node. In this case the root of tree is the variable "TEI01ICD" and all of the branches of our tree extend from it.

In [91]:
```
test_pred_decision_tree = clf.predict(test_x)
```

In [92]:
```
#import the relevant packages
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
#get the confusion matrix
confusion_matrix = metrics.confusion_matrix(test_lab,
                                            test_pred_decision_tree)
#turn this into a dataframe
matrix_df = pd.DataFrame(confusion_matrix)
#plot the result
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
#set axis titles
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted label", fontsize =15)
#ax.set_xticklabels(['']+labels)
ax.set_ylabel("True Label", fontsize=15)
ax.set_yticklabels(list(labels), rotation = 0)
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

- The above visual is called a confusion matrix. These closesly resemble heatmaps and are used predominantly to help with statistical classification.
- Confusion matrices provide a way to visually observe the accuracy of a classification model based built from a know ground truth and branching into predicted values.

Assessing the accuracy of our completed model

In [93]:
```
metrics.accuracy_score(test_lab, test_pred_decision_tree)
```
Out[93]:
```
0.965818759936407
```

In [94]:
```
print(metrics.classification_report(test_lab,
                                    test_pred_decision_tree))
```
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1508
           1       0.94      1.00      0.97      1940
           2       0.97      0.63      0.77       326

    accuracy                           0.97      3774
   macro avg       0.97      0.88      0.91      3774
weighted avg       0.97      0.97      0.96      3774
```

In [95]:
```
#extract importance
importance = pd.DataFrame({'feature': X_train.columns,
'importance' : np.round(clf.feature_importances_, 3)})
importance.sort_values('importance', ascending=False, inplace = True)
print(importance)
```
```
            feature  importance
15  TEIO1ICD_3895.0       0.812
37    TRIMIND1_15.0       0.153
50     TRMJOCGR_1.0       0.017
46     TRMJOCC1_2.0       0.013
25    TRDTIND1_19.0       0.005
..              ...         ...
26    TRDTIND1_40.0       0.000
24          TELFS_3       0.000
23          TELFS_4       0.000
22          TELFS_2       0.000
74        TUBUS_1.0       0.000

[75 rows x 2 columns]
```

In [96]:

```python
from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'max_depth': [1,2,3,4,5],
                     'min_samples_split': [2,4,6,8,10]}]
scores = ['recall']
for score in scores:

    print()
    print(f"Tuning hyperparameters for {score}")
    print()

    clf = GridSearchCV(
        DecisionTreeClassifier(), tuned_parameters,
        scoring = f'{score}_macro'
    )
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    means = clf.cv_results_["mean_test_score"]
    stds = clf.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds,
                                 clf.cv_results_['params']):
        print(f"{mean:0.3f} (+/-{std*2:0.03f}) for {params}")
```

```
Tuning hyperparameters for recall

Best parameters set found on development set:

{'max_depth': 5, 'min_samples_split': 10}

Grid scores on development set:
0.665 (+/-0.002) for {'max_depth': 1, 'min_samples_split': 2}
0.665 (+/-0.002) for {'max_depth': 1, 'min_samples_split': 4}
0.665 (+/-0.002) for {'max_depth': 1, 'min_samples_split': 6}
0.665 (+/-0.002) for {'max_depth': 1, 'min_samples_split': 8}
0.665 (+/-0.002) for {'max_depth': 1, 'min_samples_split': 10}
0.888 (+/-0.032) for {'max_depth': 2, 'min_samples_split': 2}
0.888 (+/-0.032) for {'max_depth': 2, 'min_samples_split': 4}
0.888 (+/-0.032) for {'max_depth': 2, 'min_samples_split': 6}
0.888 (+/-0.032) for {'max_depth': 2, 'min_samples_split': 8}
0.888 (+/-0.032) for {'max_depth': 2, 'min_samples_split': 10}
0.880 (+/-0.030) for {'max_depth': 3, 'min_samples_split': 2}
0.880 (+/-0.030) for {'max_depth': 3, 'min_samples_split': 4}
0.880 (+/-0.030) for {'max_depth': 3, 'min_samples_split': 6}
0.880 (+/-0.030) for {'max_depth': 3, 'min_samples_split': 8}
0.880 (+/-0.030) for {'max_depth': 3, 'min_samples_split': 10}
0.893 (+/-0.034) for {'max_depth': 4, 'min_samples_split': 2}
0.893 (+/-0.034) for {'max_depth': 4, 'min_samples_split': 4}
0.893 (+/-0.034) for {'max_depth': 4, 'min_samples_split': 6}
0.893 (+/-0.034) for {'max_depth': 4, 'min_samples_split': 8}
0.893 (+/-0.034) for {'max_depth': 4, 'min_samples_split': 10}
0.944 (+/-0.039) for {'max_depth': 5, 'min_samples_split': 2}
0.945 (+/-0.039) for {'max_depth': 5, 'min_samples_split': 4}
0.945 (+/-0.038) for {'max_depth': 5, 'min_samples_split': 6}
0.945 (+/-0.039) for {'max_depth': 5, 'min_samples_split': 8}
0.947 (+/-0.034) for {'max_depth': 5, 'min_samples_split': 10}
```

- The above indicates that our predictive model, as currently constructed has an accuracy of .97 or 97%. This indicates that our model is an excellent predictor of outcomes based on the relationships between our variables and which of them is likely to be indicative of another.

Model Ensemble

In [97]:

```python
## Train Model
# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# Instantiate model with 10 decision trees
rf = RandomForestClassifier(n_estimators = 10, random_state = 42)
# Train the model on training data
rf.fit(X_train, y_train);
```

In [98]:

```python
# Use the forest's predict method on the test data
predictions = rf.predict(test_x)
```

In [99]:

```python
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(test_lab, predictions))
```

```
Accuracy: 0.998145204027557
```

In [100…

```python
#### Finding Important Features in Scikit-learn
```

In [101…

```python
feature_list = list(X_train.columns)
feature_imp = pd.Series(rf.feature_importances_, index=feature_list).sort_values(ascending=False)
feature_imp_top20 = feature_imp.head(20)
feature_imp.head(20)
```

Out[101…

```
TEIO1ICD_3895.0    0.223306
TELFS_1            0.118777
TRIMIND1_5.0       0.070632
TELFS_5            0.067305
TRMJIND1_4.0       0.061781
TUIO1MFG_2.0       0.053041
TUFWK_2.0          0.048919
TEERNHRY_2.0       0.035996
TRMJOCGR_1.0       0.035099
TRDTIND1_40.0      0.034584
TRMJIND1_10.0      0.027736
TRDTOCC1_8.0       0.025244
TRMJOCC1_1.0       0.025127
TRMJOCC1_2.0       0.024889
TRDTOCC1_2.0       0.019644
TRIMIND1_15.0      0.018605
TEERNHRY_1.0       0.016680
TRDTIND1_19.0      0.014637
TEIO1COW_4.0       0.014212
TEERNPER_6.0       0.009544
dtype: float64
```

In [102…

```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
plt.figure(figsize=(10,7))
sns.barplot(x=feature_imp_top20, y=feature_imp_top20.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are igno
red when legend() is called with no argument.
```


Conclusion

Members of a sub class of the data set are best fits for targeting for advertisement of financial services products based on demographic data revealed in the survey used. The members of that sub class are easily identified by our algorithm for the purposes of identifying a target candidate pool to present to our customers. This suitability is based on a number of key factors such as employment and income level which indicate potential need or desire for specific financial services such as banks accounts or Certificates of Deposit (CDs).

Next Steps

- Business Application:

Now that we've built and assessed the accuracy of our model the next step is to apply it and interpret the results of our findings and how they apply to our ability to answer our questions. Based on the intermediate results of our model, it is clear that the clusters we have created are indicative of the predictive value of our variables. Going forward, we will determine the profile of the ideal candidate based on the output of our predictive modeling, identifying specifically which traits are most indicative of a candidate's suitability for our customers' services and be able to provide that information to clients for use in their marketing campaigns.

References

- Data Science in 5 minutes: What is one hot encoding? Educative. (n.d.). Retrieved February 9, 2022, from https://www.educative.io/blog/one-hot-encoding
- Decision trees in python - step-by-step implementation. AskPython. (2020, December 7). Retrieved February 10, 2022, from https://www.askpython.com/python/examples/decision-trees
- Franklin, S. J. (2019, November 26). Elbow method of k-means clustering algorithm. Medium. Retrieved February 9, 2022, from https://medium.com/analytics-vidhya/elbow-method-of-k-means-clustering-algorithm-a0c916adc540
- Google. (n.d.). Clustering algorithms | clustering in machine learning | google developers. Google. Retrieved February 10, 2022, from https://developers.google.com/machine-learning/clustering/clustering-algorithms
- Jaiswal, S. (2020, July 10). K-modesclustering. Medium. Retrieved February 10, 2022, from https://medium.com/@shailja.nitp2013/k-modesclustering-ef6d9ef06449
- Mean imputation for missing data (example in R & SPSS). Statistics Globe. (2022, January 19). Retrieved February 9, 2022, from https://statisticsglobe.com/mean-imputation-for-missing-data/
- Selecting the number of clusters with silhouette analysis on kmeans clustering. scikit. (n.d.). Retrieved February 9, 2022, from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
- U.S. Bureau of Labor Statistics. (n.d.). Atus News releases. U.S. Bureau of Labor Statistics. Retrieved February 10, 2022, from https://www.bls.gov/tus/

In [ ]: