Capstone Project Report.

Project Name: Customer Segmentation Report for Arvato Financial Solutions

1. Domain Background

Arvato is a financial solutions company with solutions such as credit management, fraud management, debt collection management, payment solutions, and other financial services.

Like any other company, Arvato uses analytics to help them understand their customers better and also find new customers based on the existing customers they have.

In this project, were using population demographics to create customer segments and find out people who are likely to be Arvato's customers.

We then used features from a mailing campaign to create a model that predicted who are likely going to be Arvato's customers by responding to that campaign.

Prediction of customer churn has been done in papers such as Xia, G. E., & Jin, W. D. (2008). Model of customer churn prediction on support vector machine. Systems Engineering-Theory & Practice, 28(1), 71-77. and Huang, B., Kechadi, M. T., & Buckley, B. (2012). Customer churn prediction in telecommunications. Expert Systems with Applications, 39(1), 1414-1425., where they used SVM to predict customer churn in telecommunications and later an improvement of the same through the use of K-means clustering, SVM to improve churn prediction.

In this project, however, we used other models, created an ensemble from them to create a prediction model.

For the customer segmentation we used KMeans Clustering.

2. Problem Statement

In order to serve their clients better, find new customers, and also create better products, Arvato needs to create customer segments from its existing clientele.

The first part of the problem is a classification problem, that will make use of classification algorithms to identify customers from the general demographic population.

The 2nd part of the problem is a prediction problem to help Arvato predict who is likely to respond to a mailing campaign, hence becoming customers of Arvato.

This project will therefore help Arvato have a better view of their different clientele types, to help them grow their business as well as leverage proper revenue generation through marketing and advertising to target customers.

3. Datasets and Inputs

There were four data files associated with this project:

Udacity_AZDIAS_052018.csv: Demographics data for the general population; 891 211 persons (rows) x 366 features (columns).

Udacity_CUSTOMERS_052018.csv: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns).

Udacity_MAILOUT_052018_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign; 42 962 persons (rows) x 367 (columns).

Udacity_MAILOUT_052018_TEST.csv: Demographics data for individuals who were targets of a marketing campaign; 42 8962 persons (rows) x 367 (columns).

```
: # Load in the data, and getting a sample of the data
  azdias = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_AZDIAS_052018.csv', sep=';')
  customers = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_CUSTOMERS_052018.csv', sep=';')
  attributes = pd.read_excel('DIAS Attributes - Values 2017.xlsx')
  print('Data Loaded Successfully.......!')

  Data Loaded Successfully.......!
```

4. **Solution Statement**

After analysis I was able to create a model that helped Arvato determine who is likely to become a customer through their marketing campaigns as well as a customer segmentation report by creating clusters from the general population.

5. **Benchmark Model**

I used KNN as my benchmark for this project. Although I used 3 models, and created an ensemble model, I found that all of these models performed equally well as the ensemble model.

6. **Algorithm and techniques**

I used the below algorithms in my modelling:

a. **KMeans Clustering for Segmentation**

Being a clustering algorithm, k-Means takes data points as inputs and groups them into k clusters. This process of grouping is the training phase of the learning algorithm.

b. RandomForest Classifier/Logistic Regression and KNN for Classification.

The random forest classifier is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Logistic regression transforms its output using the logistic sigmoid function to return a probability value, and then outputs as required, in this case, a binary class.

KNN works on a principle assuming every data point falling in near to each other is falling in the same class.

I chose these algorithms because they are simple and robust, and also they proved to work well with my problem, which was a classification problem.

7. **Evaluation Metrics**

I used the elbow method to create clear clusters from the general demographics. I got 10 clusters from the model and also visualized these clusters.

For performance, I used the ROC/AUC score to determine if my model was truly predicting accurately. The AUC score was 75%, which means my model was predicting fairly well, especially given that my target variable was generally unbalanced, meaning my model was prone to overfitting.

8. **Project Design**

I began the project by:

a. **Loading dependencies and libraries.**

```
# import libraries here; add more as necessary
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score


# magic word for producing visualizations in notebook
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

b. **Loading the data**

I then loaded the data. The data was quite bulky, and took some time to load.

```
# Load in the data, and getting a sample of the data
azdias = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_AZDIAS_052018.csv', sep=';')
customers = pd.read_csv('../../data/Term2/capstone/arvato_data/Udacity_CUSTOMERS_052018.csv', sep=';')
attributes = pd.read_excel('DIAS Attributes - Values 2017.xlsx')
print('Data Loaded Successfully.......!')
```

```
Data Loaded Successfully.......!
```

c. **Data preprocessing (data cleaning/ missing values imputation, data normalization so that the dataset is balanced).**

I did some general analysis into the data, and the data had a lot of missing values.
Case in point:

```
#number of unknown values in general
unknowns=azdias.isnull().sum().sum()
print('Total Unknown Values for the Population Dataset is: ',unknowns)
```

```
Total Unknown Values for the Population Dataset is:  16902223
```

Also, from the attributes file, it was clear that categorical variables had unknowns in form of (-1/9/10/0) and therefore needed conversion as well.

```
attributes.head(2)
```

	Unnamed: 0	Attribute	Description	Value	Meaning
0	NaN	AGER_TYP	best-ager typology	-1	unknown
1	NaN	NaN	NaN	0	no classification possible

I therefore created functions to clean the data, replace the 'unknowns' with NaNs for actual cleaning and then also carried out some label encoding, to change some labels into categorical values, and so on.

I also dropped some features which had, in particular too many categories in them.
I did this for both datasets.
To accelerate runtime, I used a sample size of random sample of 5000 for both datasets for my modelling.

```
#to accelerate the runtime, we will use a sample of the data.
population = azdias.sample(n=5000)
customers= customers.sample(n=5000)
print(population.shape,customers.shape)
```

```
(5000, 366) (5000, 369)
```

```
#Label encoding
customers.replace({'PRODUCT_GROUP':{'COSMETIC':0,'FOOD':1,'COSMETIC_AND_FOOD':2},
                   'CUSTOMER_GROUP':{'MULTI_BUYER':0,'SINGLE_BUYER':1}
                   },inplace=True)
```

I also standardized the numerical features so that my model performance would not be affected by the variability in the means.

```
#Lets now clean and encode  our data
population = clean_data(population, attributes)
customers = clean_data(customers,attributes)
```

```
Cleaning attributes data ...
==============================
Deleting Nas columns ...
Old Data shape:  (5000, 366)
==============================
New Data shape:  (5000, 360)
==============================
==============================
Replacing Unknowns by Nas ...
==============================
Imputing data ...
Number of Nas before filling : 1391490
Number of Nas after filling Numeric features : 1391490
Number of Nas after  filling All features: 0
==============================
Encoding data ...
==============================
Final data size:  (5000, 356)
Cleaning attributes data ...
==============================
Deleting Nas columns ...
Old Data shape:  (5000, 369)
==============================
New Data shape:  (5000, 363)
==============================
==============================
```

d. **Exploring the data to understand the various underlying characteristics, statistical analysis, graphical analysis, and visualizations.**

After cleaning, I observed some statistical analyses, ensuring that my data had been changed as per the functions expectations.

e. **Customer Segmentation Report**

My data was now ready for segmentation.

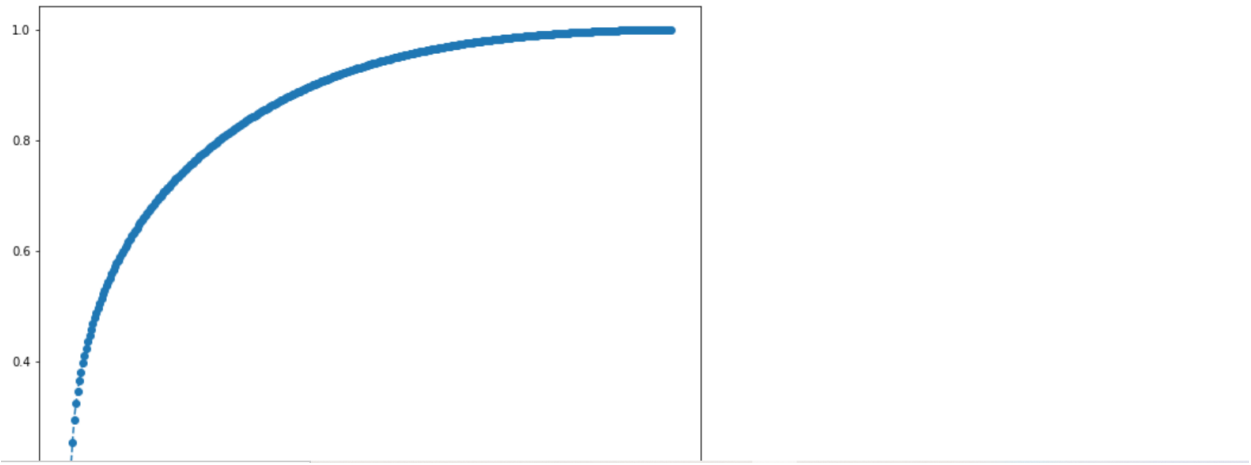
First, I performed Principal Component Analysis, to determine which features were most important and explained 80% of the variance. I obtained 89 features.

I then used these features to transform my dataset and create a new dataset for both population and customers' datasets.

```
#obtaining the best features
pca = PCA() # init pca
pca.fit(population) # fit the dataset into pca model

num_components = len(pca.explained_variance_ratio_)
ind = np.arange(num_components)
cumulativeValue = pca.explained_variance_ratio_.cumsum()

#we can plot the cumulative value as below
plt.figure(figsize= (10, 8)) # size of the chart(size of the vectors)
plt.plot(ind, cumulativeValue, marker = 'o', linestyle="--");
```




```
#obtaining columns that have explain 80% variance in the population dataset
df= pd.DataFrame(cumulativeValue,columns=["cum_variance"])
mask=df.cum_variance > 0.80
k=df[mask].index[0]

print('Columns that explain 80% variance are: ',k)
```

```
Columns that explain 80% variance are:  89
```

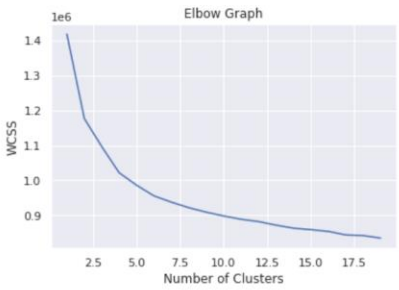
```
pca = PCA(n_components=89)
newpop_data = pca.fit_transform(population)
print('New Population Data after PCA has: ',newpop_data.shape)
```

```
New Population Data after PCA has:  (5000, 89)
```

Using the new dataset, I was able to create clusters.

First, I visualized the clusters using an elbow graph. After cluster 10, the graph is seen to drop, hence the reason why I settled for 10 clusters.

```
#plotting the elbow graph
sns.set()
plt.plot(range(1,20),wcss)
plt.title('Elbow Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



I then used these clusters to visualize the various groups that existed.

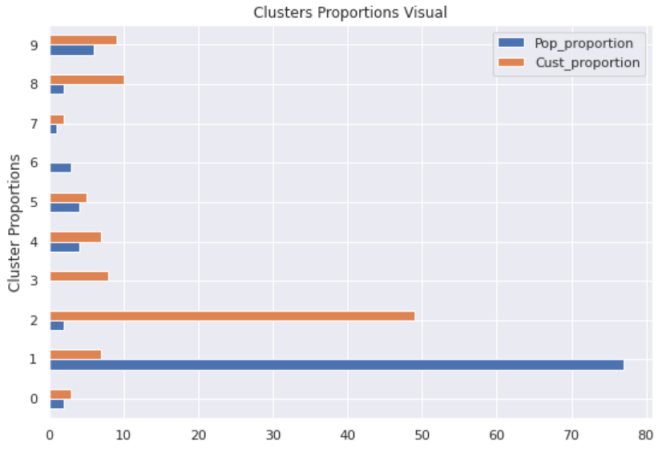
```
# lets check out the clusters
cluster_info = pd.DataFrame([])

cluster_info["Population"] = pd.Series(Y).value_counts().sort_index()
cluster_info["Customers"] = pd.Series(Z).value_counts().sort_index()
cluster_info.reset_index(inplace=True)
cluster_info.rename(columns={"index":"Cluster"}, inplace=True)
```

```
cluster_info
```

	Cluster	Population	Customers
0	0	91	164
1	1	3849	347
2	2	96	2452
3	3	10	395
4	4	179	332
5	5	205	229
6	6	138	14
7	7	52	122
8	8	87	518
9	9	293	427

```
cluster_info.plot(x='Cluster', y = ['Pop_proportion', 'Cust_proportion'], kind='barh', figsize=(9,6))
plt.title('Clusters Proportions Visual')
plt.ylabel('Cluster Proportions')
plt.show()
```



I concluded the below from these observations:

The above intel shows that, customers are mostly in clusters 2,8,9,4,1 However, potential customers will be coming from clusters 1,9,4,5,6. This is because there are a lot more people in the population in these clusters.

f. **Creating a model using supervised models.**
Then I used the mail-out dataset to create models.

My first step was to clean the data using the previous functions.

```
newmail_data=clean_data(mailout_train,attributes)
```

```
Cleaning attributes data ...
==============================
Deleting Nas columns ...
Old Data shape:  (42962, 367)
==============================
New Data shape:  (42962, 361)
==============================
==============================
Replacing Unknowns by Nas ...
==============================
Imputing data ...
Number of Nas before filling : 6671515
Number of Nas after filling Numeric features : 6671515
Number of Nas after  filling All features: 0
==============================
Encoding data ...
==============================
Final data size:  (42962, 357)
```

I then also encoded my target variable, as it was previously a continuous variable even though it was supposed to be binary.

```
newmail_data.RESPONSE.value_counts()
```

```
-0.089843     42618
 11.130568      344
Name: RESPONSE, dtype: int64
```

```
from sklearn import preprocessing
from sklearn import utils

lab_enc = preprocessing.LabelEncoder()
Y_encoded = lab_enc.fit_transform(newmail_data.RESPONSE)
print(utils.multiclass.type_of_target(newmail_data.RESPONSE))
print(utils.multiclass.type_of_target(newmail_data.RESPONSE.astype('int')))
print(utils.multiclass.type_of_target(Y_encoded))
```

```
continuous
binary
binary
```

After that, I created my X and Y datasets, X being the training set and Y being the target.

I then split into X_train,X_test,Y_train and Y_test sets.

I also wanted to use different models, to check performance of my dataset. I used KNN, Logistic Regression, and Random Forest Classifiers.

I then used GridSearchCV for the best parameters, chose the best models and then created an ensemble model.

```
#test the three models with the test data and print their accuracy scores
print('knn: {}'.format(knn_best.score(X_test, Y_test)))
print('rf: {}'.format(rf_best.score(X_test, Y_test)))
print('log_reg: {}'.format(log_reg.score(X_test, Y_test)))
```

```
knn: 0.9920875029090063
rf: 0.9920875029090063
log_reg: 0.99185478240633
```

```
from sklearn.ensemble import VotingClassifier
#create a dictionary of our models
estimators=[('knn', knn_best), ('rf', rf_best), ('log_reg', log_reg)]
#create our voting classifier, inputting our models
ensemble = VotingClassifier(estimators, voting='hard')
```

```
%%time
#fit model to training data
ensemble.fit(X_train, Y_train)
#test our model on the test data
print(ensemble.score(X_test, Y_test))
```

```
0.9920875029090063
CPU times: user 14.2 s, sys: 790 ms, total: 15 s
Wall time: 9.39 s
```

It's interesting to note that all the models performed well, this initially, however, Its was highly likely my model was overfitting because my target variable was quite unbalanced.

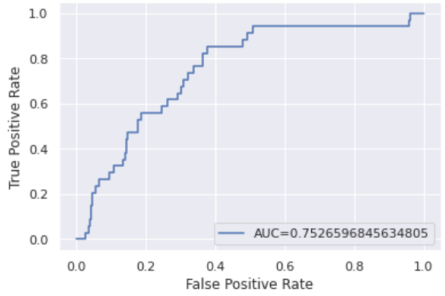
I therefore checked my ROC/AUC Score for the same, to verify that indeed, my model was classifying well.

I used Logistic Regression for this part.

I then plotted an AUC/ROC graph for visualization. The score was 75%, which means my model classified most of the data correctly.

```
Y_pred_proba = log_reg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test,  Y_pred_proba)
auc = metrics.roc_auc_score(Y_test, Y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



The closer AUC is to 1, the better the model. A model with an AUC equal to 0.5 is no better than a model that makes random classifications. Our model has an AUC of 0.75, which makes it a good classifier.

g. **Submit my solution to Kaggle for ranking.**

Although I generated a file for submission, I found that the Kaggle-Udacity requirements had changed, and my dataset has more variables than required.

This might have been because the competition is way past and my test data set was new(as it had more rows).

h. **Write my findings through a blog post.**

I documented my project on medium. This can be obtained here:

i. **Conclusion**

Using real-world data, was great. Real-world data is quite messy, so this was a great stretch for me.

This report can be improved by:

1. Use of other models say, LightGBM, XGBoost, and others.
2. Addition of more data, to correct the imbalance in the target variable. That would perhaps improve our AUC score.