

## User guide for Customer Retention Model

### Prerequisites

- 1) Python 3
- 2) Jupyter Notebook
- 3) Visual Studio Code
- 4) GitHub Account

### Section 1: Data Cleaning

- 1) Open Jupyter notebook
- 2) Open Data Cleaning.ipynb
- 3) Uncomment and run the first cell to install pandas, numpy, sklearn.

```
1 # !pip install pandas
2 # !pip install numpy
3 # !pip install sklearn
```

- 4) Run the 2<sup>nd</sup> cell to import pandas and numpy packages

```
1 import pandas as pd
2 import numpy as np
```

- 5) Import the dataset to be cleaned. Change “train.csv” to the name of the file. This can be both the training file and testing file.  
**\*\*Note that for training file, labels need to be the at the right most column and column name must be ‘labels’**

```
1 # Labels to be last column
2 df = pd.read_csv("train.csv")
```

- 6) Continue running the cells to check how many records there are in the dataset, number of rows that contain NA values and drop these NA values.
- 7) Check for features that are unique to each customer using df.nunique()
  - a. If there are features that are unique to each customer for eg. CustomerID, remove them by running the next 2 cell.

#### **Remove columns where features are unique to each customer**

```
1 # get columns where nunique count == no. of records in dataframe
2 columns = []
3 for col in df.columns:
4     if len(df[col]) == len(df[col].unique()):
5         columns.append(col)
6
7 columns
```

[]

```
1 # drop columns where nunique num == no. of records --> meaning columns where it is unique to each customer
2 df = df.drop(columns = columns)
```

- 8) Check that columns that have values unique to each customer has been removed by running the next cell, df.head()

- 9) Next, we will be performing data standardization. Data standardization should only be applied to column where values are **continuous** (decimal for eg. 1.23) NOT discrete (whole number for eg. 0,1,2,3)

Note: Change columns\_to\_scale to column headers where its values are continuous

```
1 #scale continuous variables only (i.e. feature 0 to feature 6)
2 from sklearn import preprocessing
3
4 #change columns_to_scale to column headers where values in that column is continuous
5 columns_to_scale = ['feature_0', 'feature_1', 'feature_2', 'feature_3', 'feature_4', 'feature_5', 'feature_6']
6 standardized_X = preprocessing.scale(df[columns_to_scale])
7 standardized_features = pd.DataFrame(standardized_X)
8 df[columns_to_scale] = standardized_features
9
10 # after standardisation
11 df.head()
```

- 10) Next, we will perform label encoding to convert textual data to numerical data as machine learning models cannot work with text.

- a. Add all the columns name into a list called feature

```
1 # add all column names into a list called 'feature'
2 feature=[]
3 for col in df.columns:
4     feature.append(col)
```

- b. Check value type of each column

```
1 #check value type for each column
2 for i in feature:
3     print(df[i].dtypes)
```

- c. If value type of column is 'object', it will be encoded into numbers

```
1 # if value type is object (which means it is a text), Label encoder will encode it into numerical values
2 from sklearn.preprocessing import LabelEncoder
3 lb_style = LabelEncoder()
4
5 for i in feature:
6     if df[i].dtypes=='object':
7         df[i] = lb_style.fit_transform(df[i])
```

- d. Check that textual values have been changed into numbers by running df.head()

- 11) Export the cleaned dataset. You can rename the file to any name you prefer by changing 'train\_cleaned'. Note that the extension '.csv' is required.

```
1 df.to_csv('train_cleaned.csv', index=False)
```

- 12) Perform data cleaning for both the train and test file and export them so that they can be uploaded into the flask application (Section 3).

## Section 2: Model optimization (Get hyperparameters for models)

- 1) Open Model Optimisation.ipynb
- 2) Run the first box to import required modules
- 3) Import only the training file that you have just cleaned. (Change the filename accordingly if you saved it in another name)

```
In [2]: 1 train = pd.read_csv("train_cleaned.csv") #import training data, training data has features and label
        2 train.head()
```

Out[2]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	feature_11	feature_12	feature_13	feature_14
0	-0.276515	-0.424429	1.344997	-0.012283	0.076230	1.076648	0.182198	3	0	1	0	0	0	0	0
1	0.853573	0.150991	0.503892	-0.979179	-0.569351	-0.411453	-0.251940	4	1	2	0	1	0	0	0
2	0.947747	-0.173832	1.825628	-0.703478	0.076230	-0.411453	-0.251940	6	1	2	0	0	0	0	0
3	0.853573	-0.381404	0.984523	-0.039464	-0.569351	-0.411453	-0.251940	4	0	2	0	1	0	0	0
4	1.324443	1.590527	-1.178318	-0.097711	-0.246560	-0.411453	-0.251940	0	1	1	0	0	0	0	0

- 4) Run all the codes below
- 5) After running all the codes, go to 'Optimisation' section (as shown below).

### Optimisation

#### XGBoost

```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.model_selection import RandomizedSearchCV

1 import time
2
3 clf_xgb = XGBClassifier()
4
5 max_depth = range(5,8)
6 learning_rate = [0.1, 0.2, 0.3]
7 colsample_bytree = [0.2,0.3,0.4]
8 reg_alpha = [0.7,0.8,0.9]
9 param_grid = dict(max_depth=max_depth, learning_rate=learning_rate, colsample_bytree=colsample_bytree, reg_alpha=reg_alpha)
10
11 grid = GridSearchCV(estimator=clf_xgb, param_grid=param_grid, cv = 3, n_jobs=-1)
12
13 start_time = time.time()
14 grid_result = grid.fit( X_train, y_train)
15 # Summarize results
16 print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
17 print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

Best: 0.925128 using {'colsample\_bytree': 0.4, 'learning\_rate': 0.1, 'max\_depth': 7, 'reg\_alpha': 0.7}  
Execution time: 134.55280017852783 ms

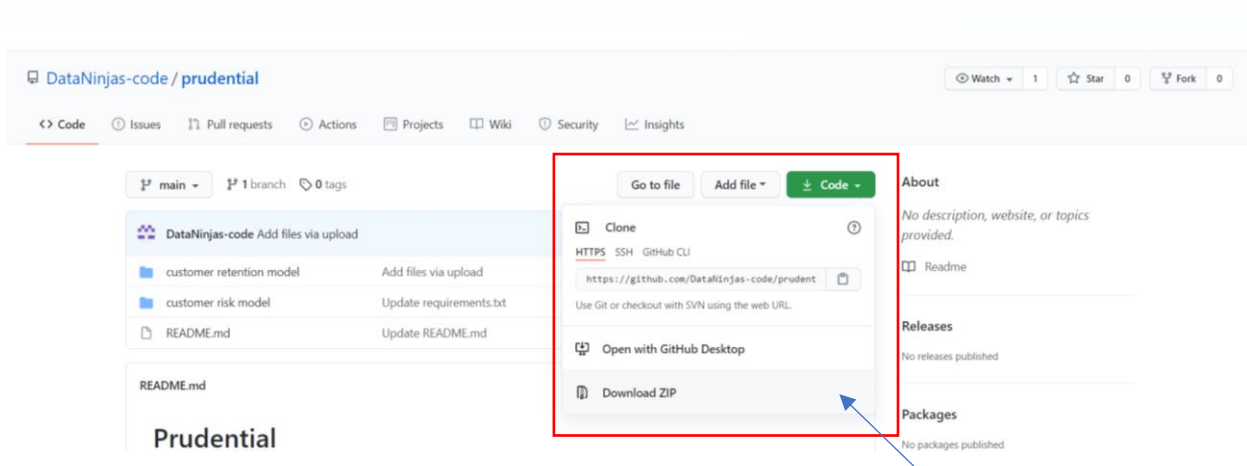
For each of the models, get the best hyperparameters.

For eg. For XGBoost, the best parameters are {'colsample\_bytree': 0.4, 'learning\_rate': 0.1, 'max\_depth': 7, 'reg\_alpha': 0.7}

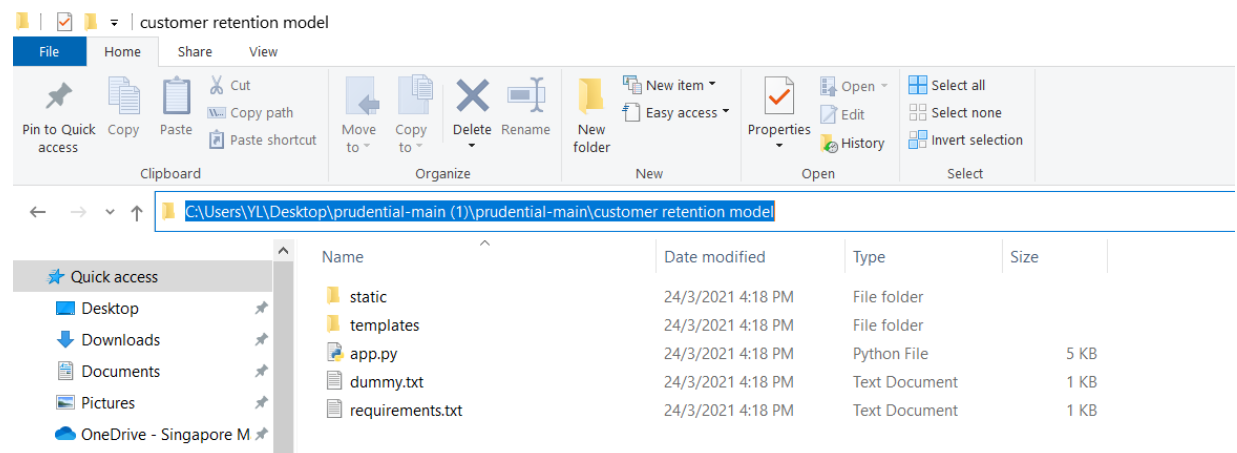
- 6) Take note of all these best hyperparameters as you need to replace them in the flask application in the next section.

### Section 3: Customer Retention Prediction Model

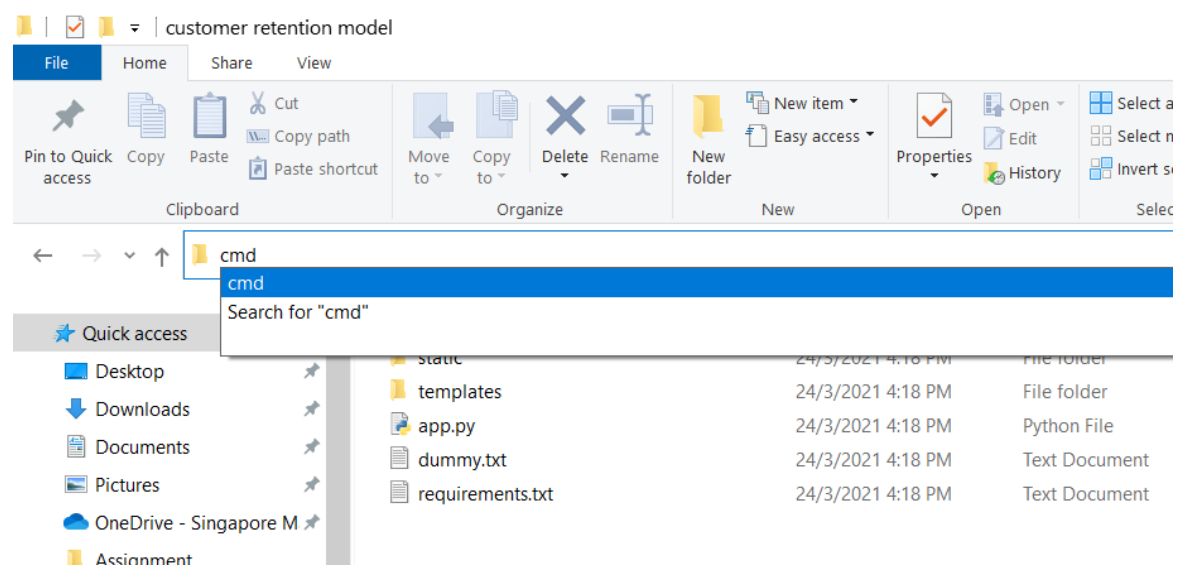
- 1) Go to <https://github.com/DataNinjas-code/prudential>
  - Download the file



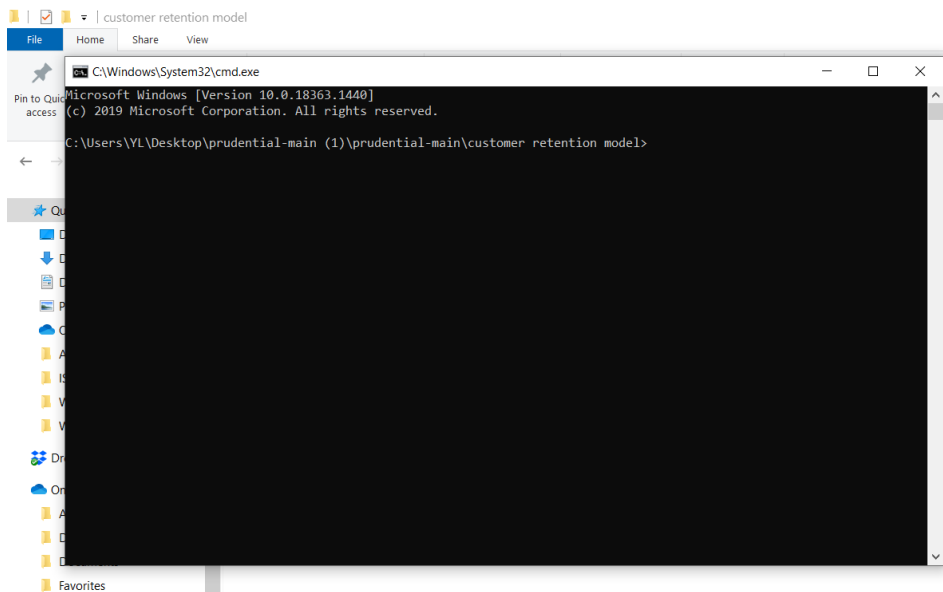
- Extract the contents in the zip file
- Click into the folder and into 'customer retention model'
- Click on the file path (make sure your file path is similar to the screenshot)



- Type 'cmd' and press enter

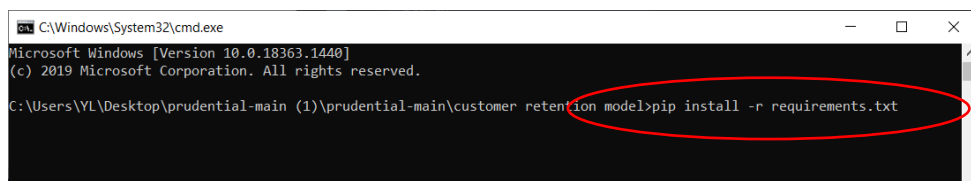


- You will see your command prompt being opened up.

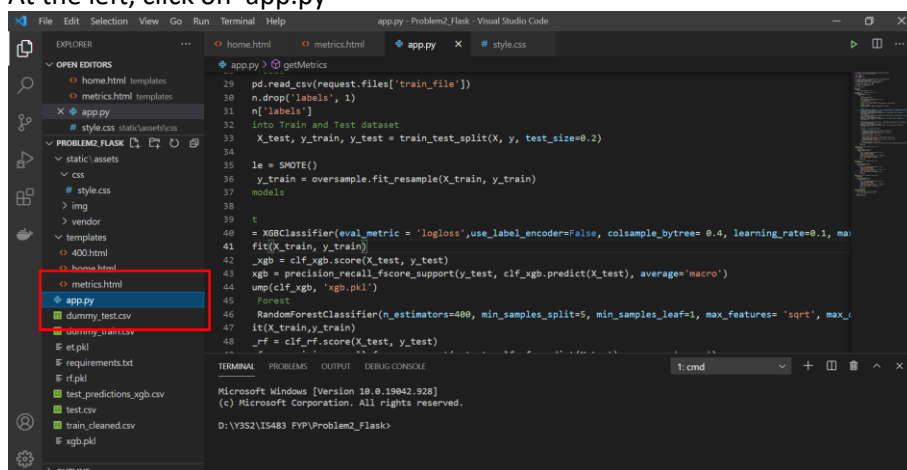


- 2) Pip install all the required packages using requirements.txt by typing:

```
pip install -r requirements.txt
```



- 3) Open the flask application in Visual Studio Code or any Software Development Environment.  
If you are using Visual Studio Code, follow the instructions below:
  - a. Open Visual Studio Code
  - b. Click on File > Open Folder > Select the 'customer retention model' folder
  - c. At the left, click on 'app.py'



- d. Inside app.py, change the hyperparameters of the 3 models (XGBoost, Random Forest and Extra Trees) which was derived in section 2.

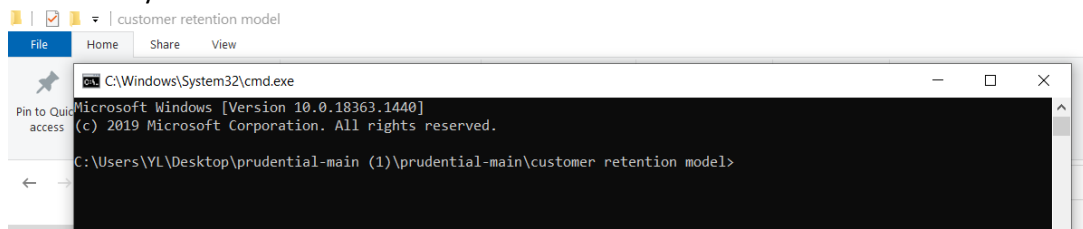
```
# XGBoost
clf_xgb = XGBClassifier(eval_metric = 'logloss', use_label_encoder=False, colsample_bytree= 0.4, learning_rate=0.1, max_depth=7, reg_alpha=
```

```
# Random Forest
clf_rf = RandomForestClassifier(n_estimators=400, min_samples_split=5, min_samples_leaf=1, max_features= 'sqrt', max_depth=None, bootstrap=
```

```
# Extra Trees
clf_et = ExtraTreesClassifier(criterion='gini', max_depth=32, max_features='sqrt', n_estimators= 50)
```

- e. Save app.py using Ctrl + S

- 4) Go back to your cmd



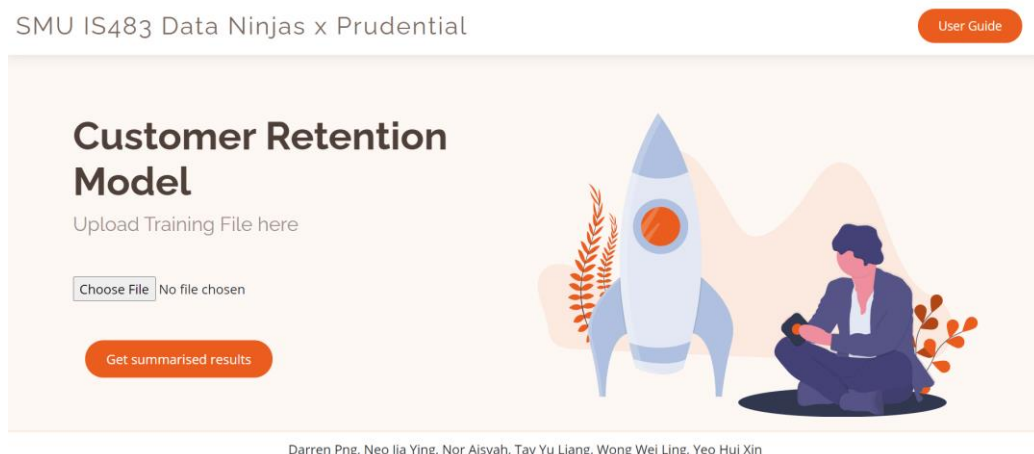
- 5) Run the Flask App by typing:

```
python app.py
```

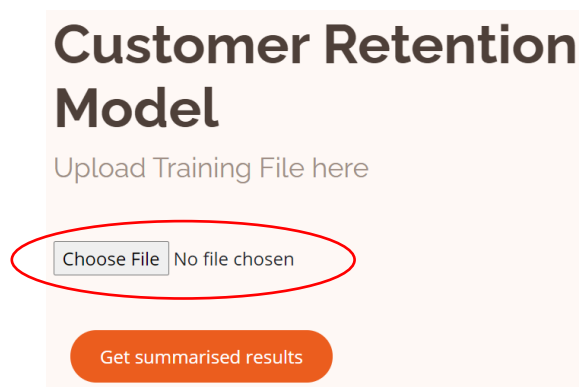
You should see this which means the flask app is live and running.

```
C:\Users\User\prudential\customer_retention_model>python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 256-036-531
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- 6) Go to the link: <http://127.0.0.1:5000/> and you should see this page below:



- 7) Upload cleaned training file by clicking on the “Choose File” button.



**\*\* Note: Training file needs to contain at least 10,000 rows of data**

- 8) Click on ‘Get summarised results’ button to get the metrics for these models: Random Forest, Extra Trees and XGBoost.
- 9) After clicking on ‘Get summarized results’, you will need to wait for a while, depending on how much data you have uploaded, before seeing this page. At this page, you can see the accuracy, precision, recall and f-score of the 3 models.

## SMU IS483 Data Ninjas x Prudential

[User Guide](#)

### Results

Elapsed time: 0:01:05.054489

Model	Accuracy	Precision	Recall	F-score	Upload test file and predict		
XGBoost	0.8959009141846063	0.7485255210869708	0.7956171061077999	0.7688558002658976	Choose File	No file chosen	Export
Random Forest	0.8905927455028015	0.7373462026017679	0.7793876787108598	0.7556442375916315	Choose File	No file chosen	Export
Extra Trees	0.8876437629017989	0.7311845063926085	0.7716572325286369	0.7488118846530726	Choose File	No file chosen	Export

### Performance Measure

Highest **accuracy** model: XGBoost

Highest **F-score** model: XGBoost

### Definitions

- Accuracy: Represents the percentage of correctly predicted data points out of all the data points
- Precision: Represents the percentage of the true positive out of all the positive outputs of the system
- Recall: Represents the percentage of items actually present in the input that were correctly identified as positive by the system
- F-Score: A combined measure that is derived from Precision (P) and Recall (R) (weighted harmonic mean)

- 10) To choose the best model to predict the labels on your test file:

For the model chosen

- Upload test file
- Click “export”

### Results

Elapsed time: 0:01:05.054489

Model	Accuracy	Precision	Recall	F-score	Upload test file and predict		
XGBoost	0.8959009141846063	0.7485255210869708	0.7956171061077999	0.7688558002658976	Choose File	No file chosen	Export
Random Forest	0.8905927455028015	0.7373462026017679	0.7793876787108598	0.7556442375916315	Choose File	No file chosen	Export
Extra Trees	0.8876437629017989	0.7311845063926085	0.7716572325286369	0.7488118846530726	Choose File	No file chosen	Export

- 11) Once you click on ‘export’, a file will be downloaded. This file contains the labels predicted by the respective model on the test file you uploaded.

#### Section 4: Clustering Model

- 1) Open Jupyter notebook
- 2) Open "Clustering for Customer who churned.ipynb" OR "Clustering for Customer who retained.ipynb"
- 3) Uncomment and run the first cell to install pandas, numpy, seaborn and matplotlib.

```
1 # !pip install pandas
2 # !pip install numpy
3 # !pip install seaborn
4 # !pip install matplotlib
```

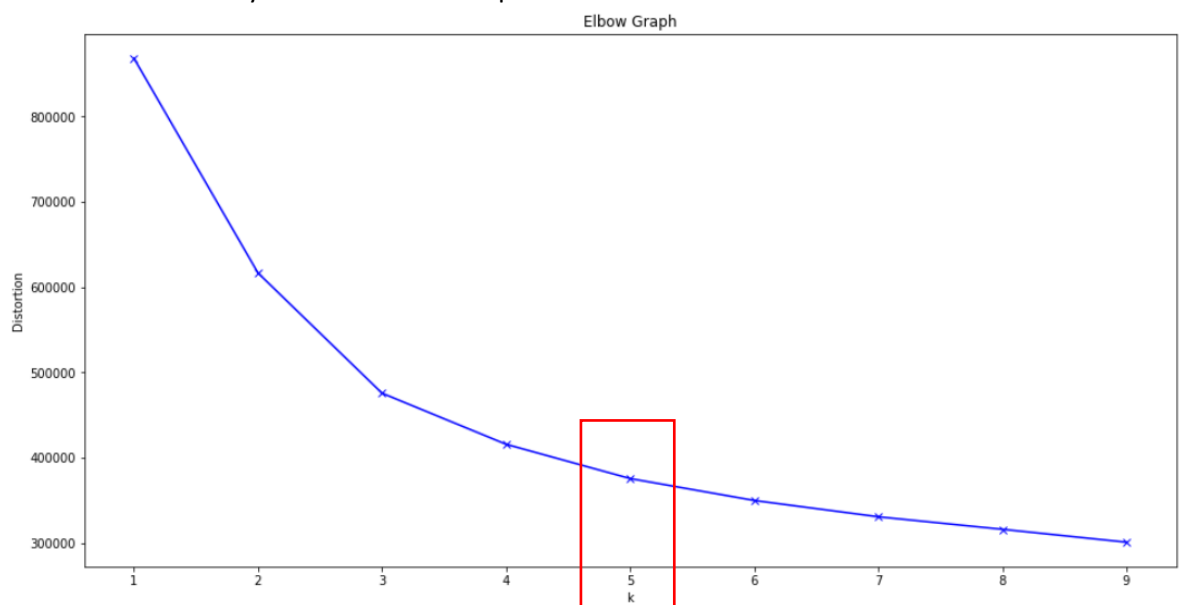
- 4) Run the 2<sup>nd</sup> cell to import packages

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
```

- 5) Import the test file that you exported in section 2 (the file with labels predicted). Change the file name to import accordingly.

```
1 train = pd.read_csv("train_cleaned.csv") #import file accordingly
2 train.head()
```

- 6) Run all the cells until you reach the elbow plot.



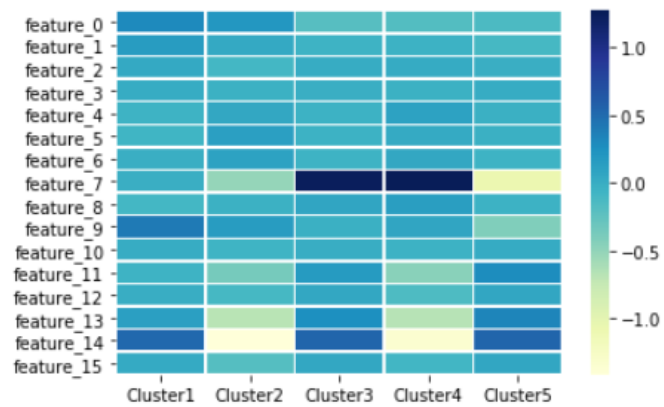
Elbow point is when the decrease in distortions is not as significant as k increases. In this case, the elbow point occurs when k=5.

- 7) For Kmeans clustering, change n\_clusters to the elbow point's k.

```
1 kmeanModel = KMeans(n_clusters=5) #change n_clusters accordingly to elbow point
2 kmeanModel.fit(features)
```



8) Continue running the cells until you reach this heatmap.



This heatmap tells you the highly ranked variables in each cluster. The highly ranked variables are shaded in very light yellow or very dark blue.