Midterm

Johnny Nino Ladino; University of Houston

This paper was prepared as partial completion of the course MATH 6350 (Statistical Learning & Data Mining) taught during the Fall 2020 semester at University of Houston.

Background and Class Creation

This report will cover the tasks for the Midterm Project presented by Dr. Azencott. The dataset that will be used for this project is named age_Gender.csv and was retrieved from the Kaggle website. Each row represents a 'case' and each case describes numerically a digitized image of an individual's face. Each image has a 48 by 48 size and therefore 2,304 pixels. The pixels for each case are listed in within a single column named "pixels." Each one of these 2,304 pixels has a "gray level" which is an integer value between 0 and 255. There's a total of 23,705 images in this dataset, each row representing the face of a distinct individual. The dataset consists of 5 columns which are age, ethnicity, Gender, image_name, and pixels. The age column describes the person's age in the image for each case. The ethnicity column describes the ethnicity of the person in the image for each case. The Gender column describes the Gender of the of the person in the image for each case. The image_name column is the specific name of the jpg file for each image. Lastly, the pixels column has the list of "gray level" integer values in the image for each case.

For this dataset, Gender and age will be our target variable and the 2,304 values describing the gray levels will be our features. To keep the number of classes greater than 5 and less than 10, we will only look at ages from 1 to 75. This changes the number of cases from 23,705 images to 22,854 images. Bins will be created to categorize age. Ages 1 to 25 will be bin 1 and they will be labeled as the "Young" Age Group. Ages 26 to 50 will be bin 2 and they will be labeled as the "Middle" Age Group. Lastly, Ages 51 to 75 will be bin 3 and they will be labeled as the "Old" Age Group. Our target variable will have 6 classes. 3 classes will be the females within their respective Age Group. The other 3 classes will be the males within their 5 respective Age Groups. Table 1 below defines our classes. Our features are integer values ranging from 0 to 255. The aim of this report will be to classify the Gender and Age Group of the person displayed in each image. The underlying code sustaining this report will be created using python and the IDE of Jupyter notebook. Figure 1 below is a sample of images from our dataset.

The ability to classify gender and age has many practical applications. This could lead to creating devices that are able to do face recognition. Face recognition already exists today. There are features in smartphones that can unlock your phone using the camera. Being able to determine who's face it is, will help your phone decide whether to be unlock or not. Being able to determine age and gender fits into those categories.

CLASS	Gender	Age Group
Class FY	Female	Young (Age:0-25)
Class FM	Female	Middle (Age:26-50)
Class FO	Female	Old (Age:51-75)
Class MY	Male	Young (Age:0-25)
Class MM	Male	Middle (Age:26-50)
Class MO	Male	Old (Age:51-75)

Table 1:Class Definition

Figure 1:Sample of Images from Dataset

Methodology

The following methodology was used for developing the code and answering the questions in the document

1. Code will be written in python and use python libraries such as pandas, numpy, scikit learn etc
2. Code and report will abide to the requirements presented in Dr. Azencott's instructions
3. The report/code will use the KNN algorithm to classify gender and age from the data containing images
4. The report/code will use PCA to improve computing efficiency and attempt to improve model results
5. The report/code will aim to optimize the KNN model to determine if KNN is a good model for image classification

Preliminary Treatment of Data

To begin the analysis, data cleaning and preparation needs to be done on the dataset. Out of the 5 columns in our dataset, we will keep only 3 of them; age, gender, and pixels. Table 1 below explains what each of these columns represent. Luckily, the dataset has no missing data.

COLUMN	DESCRIPTION
AGE	Column lists values of the person's age in the image, ranging from 1 to 116
GENDER	Column lists values either equal to 0 or to 1. 0 = Male, 1 = Female
PIXELS	Column lists array of values ranging from 0 to 255; each value representing gray level intensity

Table 2: Column Description

As mentioned in the "Background and Class Creation" section, to keep class size greater than 5 and less than 10, we drop the images of the people that are strictly older than 75. For our pixel column, we partition the pixel into each column for each image, thus we get 2,308 added columns to our dataset. This will make it easier to work with when standardizing, creating the test and train sets, and feeding our data into our model. We then create the classes that were labeled in Table 1. Table 3 below will give the size of each class along with a visual representation of the distribution of the classes.

CLASS	ROWS	COLUMNS
CLFY (FEMALE AND YOUNG)	4,464	2,308
CLFM (FEMALE AND MIDDLE)	5,166	2,308
CLFO (FEMALE AND OLD)	1,205	2,308
CLMY(MALE AND YOUNG)	3,173	2,308
CLMM(MALE AND MIDDLE)	6,296	2,308
CLMO(MALE AND OLD)	2,550	2,308
TOTAL	**22,854**	

Table 3:Class Size

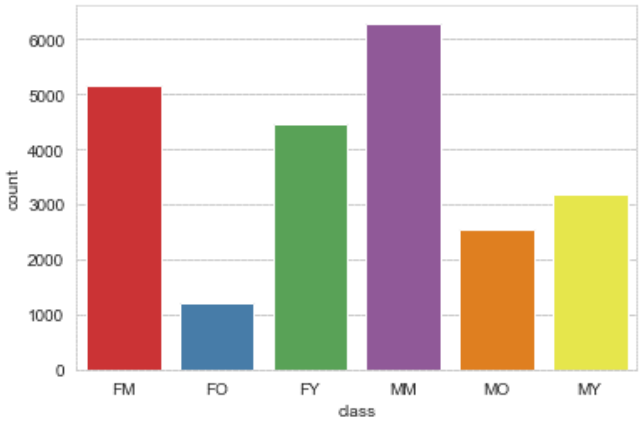

Figure 2: Class Size by Count

From Table 3 and Figure 2, we can see that there is an imbalance in the classes. The Female and Old class is the smallest in comparison to the other class sizes. The Male and Middle class is the class with the highest size. To work with this imbalance, we will have to look for sampling methods to attempt to balance these classes before creating a model.

Histograms of Features and Discriminating Power

Before moving onto standardization, we want to look at the histograms and discriminating power of some of the features. There are 2,308 features. That would be too many histograms to display. We are choosing pixel locations r18c23, r19c24, r23c6, r14c30, and r5c12. Figure 3 below shows the histograms of the 5 pixel locations where each row represents a class.

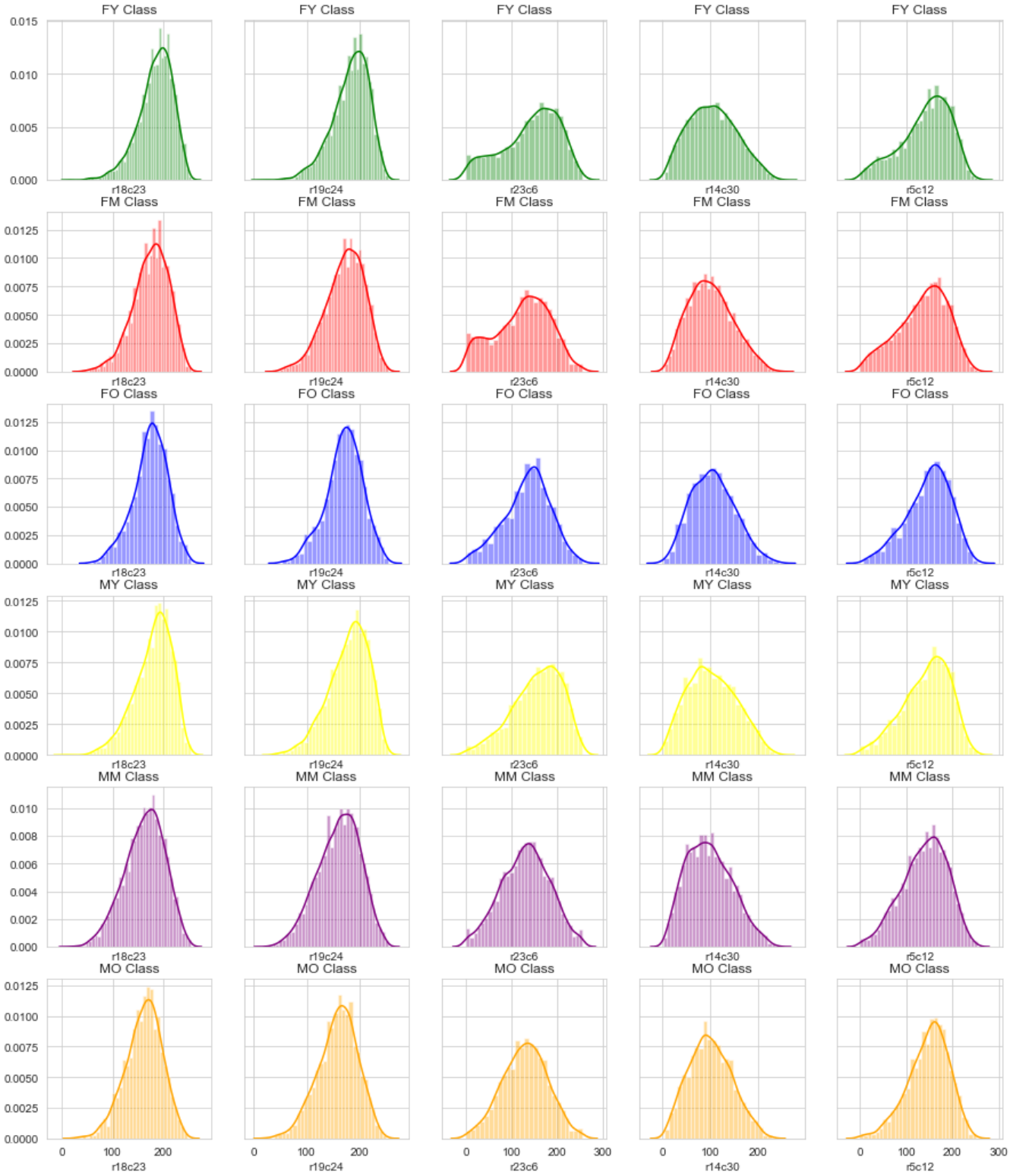

Figure 3:Histograms for rLcM Position

From the Histograms, the only visual interpretation we can see is that the distributions for each feature tend to follow the same distribution for each class. Let's look at a specific feature, r23c6. We see that the FY and FM class have similar distributions. Same goes for the FO and MM classes and MM and MO classes. It's very difficult to determine anything from these histograms since there are 2,308 features. We will proceed to calculate discriminating power by using the Kolmogorov–Smirnov test, which is a test that looks at pairs of histograms and determines if they follow the same distribution. Formally, the hypothesis are as follows:

$$H_0: 2\ Independent\ Samples\ are\ drawn\ from\ the\ same\ continuous\ distribution$$
$$H_a: 2\ Independent\ Samples\ are\ \boldsymbol{not}\ drawn\ from\ the\ same\ continuous\ distribution$$

Samples will be the cases for each feature amongst the different classes. We have 6 classes so we need to do the Kolmogorov–Smirnov test for each distinct pair. In Table 4 below, we see the discriminating power for each class with respect to the feature location. Discriminating power is calculated by doing 1 – *p-value* from the test.

	FY & FM	**FY & FO**	**FY & MY**	**FY & MM**	**FY & MO**	**FM & FO**	**FM & MY**	**FM & MM**	**FM & MO**	**FO & MM**	**FO & MO**	**MY & MM**	**MY & MO**	**MM & MO**
r18c23	1	1	0.99	1	1	0.92	1	1	1	1	1	1	1	0.99
r19c24	1	1	0.99	1	1	0.99	1	1	1	1	1	1	1	0.99
r23c6	1	1	1	1	1	1	1	1	1	0.99	0.99	1	1	0.98
r14c30	0.99	0.98	0.66	1	0.99	0.99	0.99	0.99	0.56	1	0.99	0.99	0.99	0.99
r5c12	1	0.98	0.88	1	1	1	0.99	0.99	1	1	0.51	0.99	0.99	1

Table 4:Discriminating Power (1-p) value Between Each Class

We can see that some features hold a high discriminating power. Specifically, the features r18c23, r19c24, and r23c6 have high discriminating power among all classes. Depending on which feature is used, we can have other features help in the classification problem.

Standardizing the Data

We will denote our dataset of pixels and classes as DATA. The means and standard deviations of DATA are computed to be used to standardize the dataset DATA. We standardize the dataset so values in certain features are not over weighed or under weighed when we are applying our classifier. We compute the mean and standard deviation of each column leaving us with $\hat{\mu}(mean) = \mu_{X1}, \dots, \mu_{X2,308}$ and $\hat{\sigma}(standard\ deviation) = \sigma_{X1}, \dots, \sigma_{X2,308}$. We then perform the following to rescale DATA:

$$\boldsymbol{SDATA(X_i, X_j) = \frac{DATA(X_i, X_j) - \mu_j}{\sigma_j}}$$

Equation 1:Standard Matrix Equation

We store the standardized dataset DATA into a separate data frame named SDATA. From SDATA, we compute the correlation matrix, $CORR(X_i, X_j)$, and extract the 10 pairs X_i, X_j that have the highest absolute values in $|CORR(X_i, X_j)|$. Here, X_i and X_j, refer to the position of the pixel. Table 3 below highlights the 10 highest correlated pairs by pixel position. From our pixel positions X_i, X_j, we see that the highest correlated pairs tend to share the same row and column position.

$Corr(X_i, X_j)$	X_i	X_j
0.9880	r18c23	r19c23
0.9876	r19c24	r18c24
0.9876	r23c6	r22c6
0.9876	r24c39	r25c39
0.9872	r22c41	r23c41
0.9871	r23c40	r22c40
0.9870	r24c39	r23c39
0.9869	r24c38	r25c38
0.9867	r24c7	r25c7
0.9866	r24c37	r25c37

Table 5: Highest Correlated Pairs by Pixel Position

Before moving forward, we would like to look at the features with the highest correlation within each class. This will allow us to see which pixel locations have higher correlations than others from different classes.

CLFY			**CLFM**			**CLFO**		
r18c23	**r19c23**	0.99	**r19c24**	**r18c24**	0.99	**r20c33**	**r20c34**	0.99
r25c40	**r24c40**	0.99	**r20c24**	**r19c24**	0.99	**r21c13**	**r21c14**	0.99
r24c39	**r25c39**	0.99	**r19c23**	**r18c23**	0.99	**r24c38**	**r25c38**	0.99
r23c6	**r22c6**	0.99	**r22c6**	**r23c6**	0.99	**r19c33**	**r19c34**	0.99
CLMY			**CLMM**			**CLMO**		
r24c9	**r25c9**	0.99	**r2c22**	**r2c23**	0.99	**r0c21**	**r0c20**	0.99
r18c5	**r19c5**	0.99	**r1c24**	**r1c23**	0.99	**r1c26**	**r1c27**	0.99
r21c34	**r21c33**	0.99	**r1c23**	**r1c22**	0.99	**r0c29**	**r0c28**	0.99
r25c9	**r26c9**	0.99	**r1c25**	**r1c24**	0.99	**r0c26**	**r0c25**	0.99

Table 6:Top 4 Correlation

Table 6 is great because we see where the highest correlations appear for each class. For the young female class, we see that the highest correlation happens at pixel location (**r18c23**, **r19c23**) and for the old male class, we see that the highest correlation happens at pixel location (**r0c21**, **r0c20**). This is great distinguishing our classes using our features!

Subsampling the Classes

There are various ways to fix the imbalance in these classes. In this report we will explore which method gives us the best results. To makes things simple and quick starting off, we will use Random Under Sampler from the imblearn python library. This function looks at all the classes and grabs a smaller subsample from a class. We will let Random Under Sampler make a subsample of all the 6 classes, except for the minority class, which in our case is Female and Old class. After subsampling, we see that we get 1,205 cases for each class and 7,230 cases for our entire SDATA.

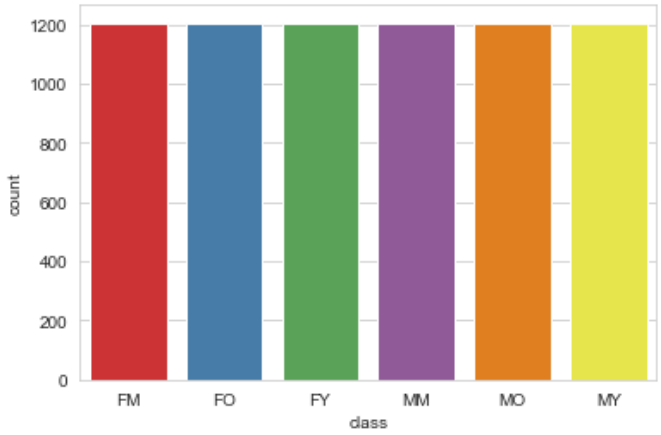

Figure 4:Class Sizes after Subsampling

Splitting SDATA into Train and Test Sets

The 80/20 approach of separating data will be taken for the standardized matrix, SDATA. CLFY, CLFM, CLFO, CLMY, CLMM, and CLMO were split into subsets of 80% and 20% at random to be used in the KNN model. The 80% subset will be for training data and the other 20% subset will be for the test data. Once the 80/20 split for each of the 6 classes was created, a union of each training and test set was created to have a singular training and test dataset that is equally divided by each class and subset in order to accurately classify in the KNN model. The 80/20 approach helps prevent bias towards one specific class and helps ensure that all classes are considered for the training and test portion of the model evaluation. Table 4 below highlights how this approach splits the data evenly.

DATASET	CLFY ROW COUNT	CLFM ROW COUNT	CLFO ROW COUNT	CLMY ROW COUNT	CLMM ROW COUNT	CLMO ROW COUNT	TOTAL
TRAINSET (80%)	964	964	964	964	964	964	5,784
TESTSET (20%)	241	241	241	241	241	241	1.446

Table 7: Train and Test Sets Sizes

Applying K-Nearest Neighbors

The K nearest neighbor algorithm (commonly referred to as KNN) is the supervised machine learning model that will be used to automatically classify arbitrary cases into one of the 6 classes. The way the KNN algorithm works is by setting a value $k = n$, where n is a natural number. KNN then looks for the training cases that are closest to a test case i. KNN then classifies the test case i case by doing a majority vote count of the nearest n training classes. KNN computes probabilities by the following formula

$$\Pr(Test\ Case\ i\ is\ class = k) = \frac{Number\ of\ cases\ where\ class = k}{n}.$$

Equation 2:Probabilities Given by KNN

This process is repeated for all cases in the test set.

For our SDATA, we implement a range of k values, particularly $k = [10, 30, 50, 70, 90, 110]$, on the training and test datasets created from SDATA. The accuracy scores for the train and test set is plotted below in Figure 4 where our x-axis is our number of neighbors, k, and the y-axis is the percentage of accuracy.

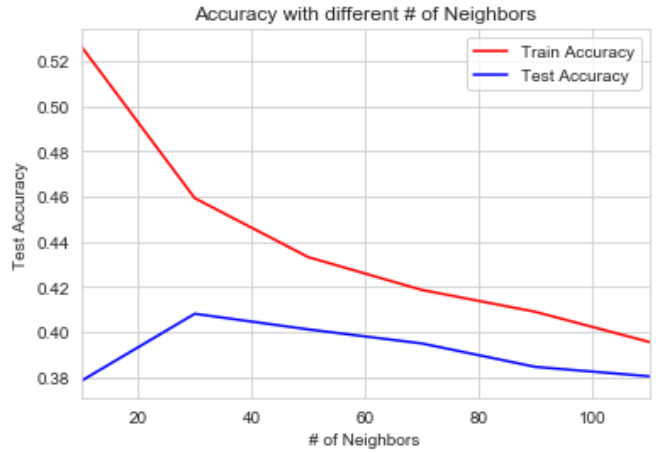

Figure 5: Train and Test Accuracy Scores on Line plot

From the plot, we can see that the training accuracy score decreases as the number of neighbors increases. As for the test accuracy score, we see that it starts off low at around 36 % for $k = 10$. It then jumps up to approximately 41% at $k = 30$ and then levels off from there. To avoid overfitting and underfitting, we wish to find the number of neighbors where the discrepancy between the train set accuracy score and the test set accuracy score are minimum. To do this we will zoom in on the range where we see the lowest discrepancy occurs in Figure 4. This happens at $k = 100$ to $k = 125$. We will run this test again, zooming into the results within this range.

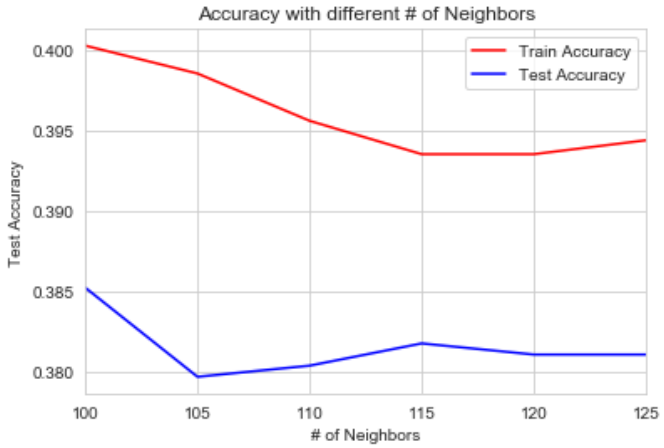

Figure 6:Train and Test Scores with Zoomed in K

From our second run of KNN with $k = [100, 105, 110, 115, 120, 125]$, we see our discrepancy is lowest at $k = 115$. Table 6 lists out the differences between the training accuracy score and the test accuracy score for each k neighbor. $K = 115$ gives us the best score the least overfitting and underfitting from this range.

K	TRAINING ACCURACY – TEST ACCURACY
100	0.015
105	0.018
110	0.015
115	0.011
120	0.012
125	0.013

Table 8: Calculations of Discrepancy between Training and Test Accuracy for Varying K Neighbors

K NEIGHBORS	TRAIN SET SCORE	TEST SET SCORE
115	0.39	0.38

Table 9:Best K Train and Set Accuracy Score

Confusion Matrices

To finish up, we will look at the confusion matrices of the train and test sets at $k = 115$ neighbors. We can explicitly see in Figure 6 which class is predicted correctly, and which is being misclassified. We see that KNN does best in predicting who is an old-aged male with 63.69% and 62.66% accuracy for the train and test set respectively. Our model is having a hard time classifying who is a young-aged female. We see that the model predicts young-aged females as middle-aged females and old-aged females.

We can quickly see from Figure 5, Table 6, and Figure 6 that KNN does poorly in being able to classify gender and age. Due to the limited amount of time for this report, we will keep using KNN, but attempt other methods that we hope will improve our model, even just a little bit, and to also apply different concepts learned in Dr. Azencott's course.

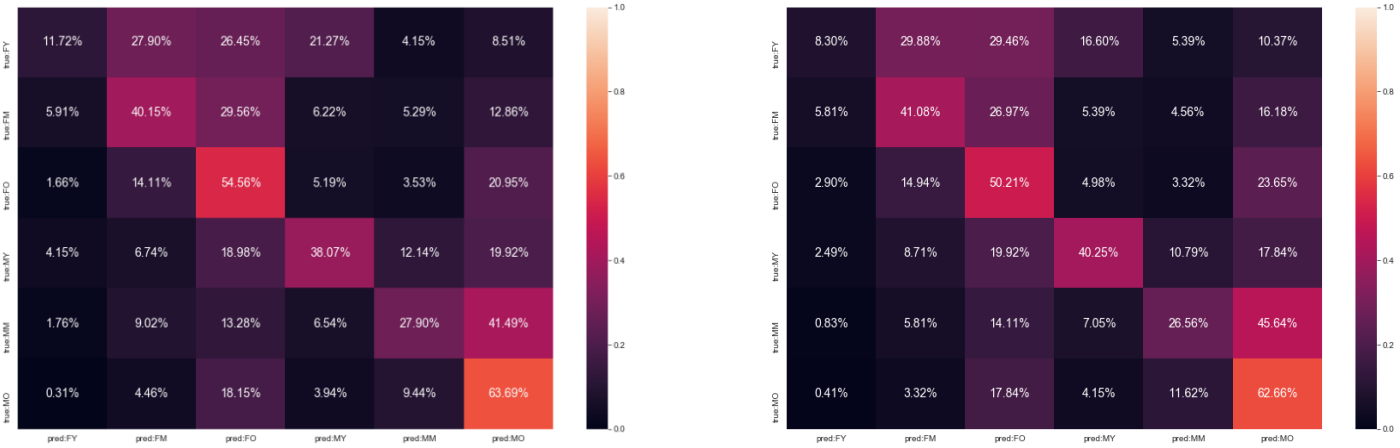

Figure 7:Confusion Matrix for Train and Test Set

Oversampling the Classes

We will now take a different approach to balancing the classes. Removing cases can delete important information from our dataset and can hurt models results. We will use Random Over Sampler which is a function within the imblearn library too. The way Random Over Sampler works is that it looks at each class, then it clones some of the cases in that class and adds it to the class size. We will resample all 6 classes except the majority class. If we recall, the majority class is the "MM" class which is shown in Figure 2 in the purple bar. After applying the Random Over Sampler, we see the size of the classes in Figure 7. After oversampling, we see that we get 6,296 cases for each class and 37,776 cases for our entire SDATA. This is a very large dataset! We will explore later and apply Principal Component Analysis to see if we are able to compress the size of our data without removing valuable predictive information.

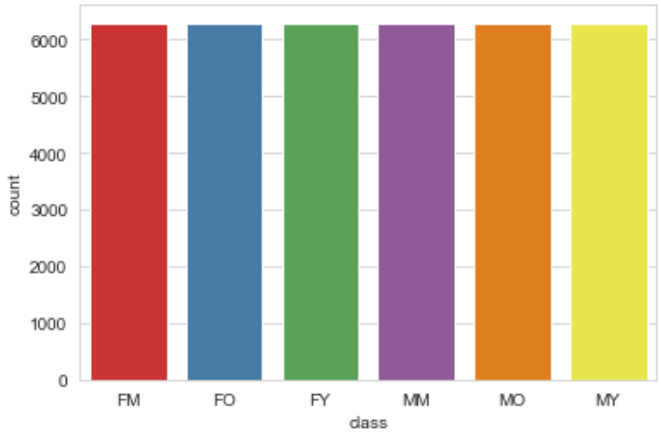

Figure 8:Class Sizes after Oversampling

Splitting SDATA into Train and Test Sets

The same 80/20 approach will be down for the class sizes, SDATA. CLFY, CLFM, CLFO, CLMY, CLMM, and CLMO were split into subsets of 80% and 20% at random to be used in the KNN model. The 80% subset will be for training data and the other 20% subset will be for the test data. Once the 80/20 split for each of the 6 classes was created, a union of each training and test set was created to have a singular training and test dataset that is equally divided by each class and subset in order to accurately classify in the KNN model.

DATASET	CLFY ROW COUNT	CLFM ROW COUNT	CLFO ROW COUNT	CLMY ROW COUNT	CLMM ROW COUNT	CLMO ROW COUNT	TOTAL
TRAINSET (80%)	5,036	5,036	5,036	5,036	5,036	5,036	30,216
TESTSET (20%)	1,260	1,260	1,260	1,260	1,260	1,260	7,560

Table 10:Class Sizes After Oversampling

Principal Component Analysis and Applying KNN

As mentioned earlier, this is a huge amount data that must be fed into our KNN model. We will explore a technique called Principal Component Analysis. This technique compresses the data by reducing the number of features. When we apply the PCA to our dataset, we can determine that 95% of the data is explained with only 188 new features. This is great because it drastically lowers our feature size count from 2,308 to 188. For clarification, we feed the training set and test set into our PCA and then transform it using the weights obtained through PCA to our features.

We then feed our new train and test sets into our KNN model and use the best k we found in our previous application of KNN using the subsampling method. We then get the following train and test scores for $k = 115$ and its confusion matrices.

K NEIGHBORS	TRAIN SET SCORE	TEST SET SCORE
115	0.45	0.44

Table 11:Best K Train and Test Score

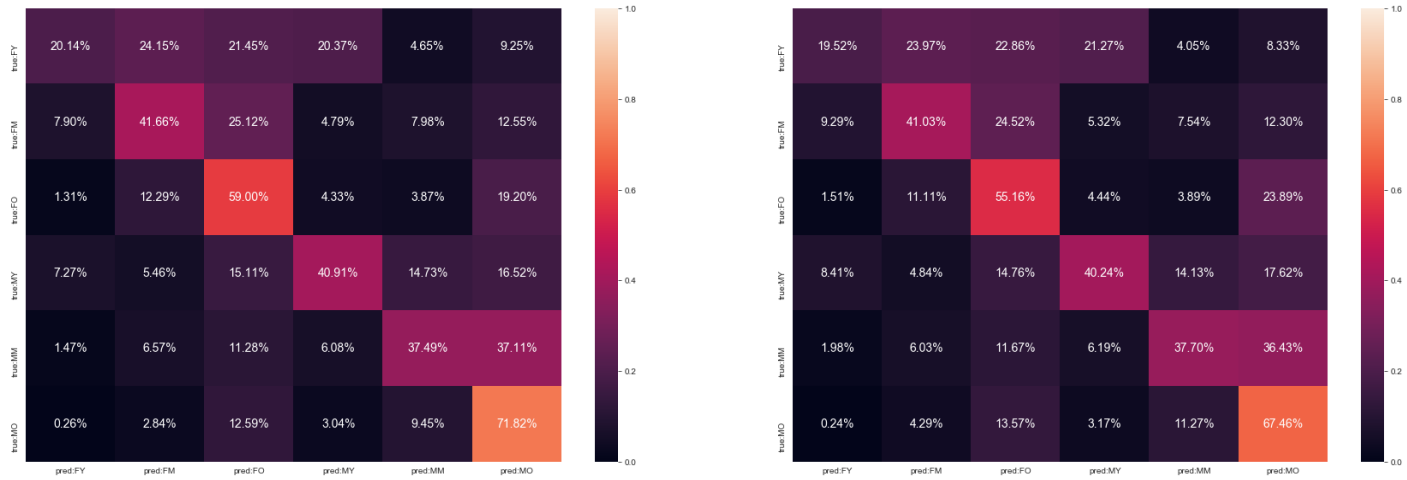

Figure 9:Confusion Matrix for Train and Test Set at K=115 After PCA

We see that we get a 10 percent boost by having applied over sampling and principal component analysis. Moreover, in our confusion matrix, we see that our classes prediction is a bit higher in comparison to Figure 7. We see that KNN does best in predicting who is an old-aged male with 71.82% and 67.46% accuracy for the train and test set respectively. Our model is still having a hard time classifying who is a young-aged female, but still does better in comparison to the previous run. We see that the model still predicts some young-aged females as middle-aged females and old- aged females or even young-aged males.

Finding New Best K Neighbor

This improved accuracy scores begs us to ponder further if we can find an even better k that will give us the least discrepancy and better score. In the left chart in Figure 10, our $k = [10,30,50,70,90]$. We see that the scores are overall much better than the ones we were able to get in Figure 5. Now we must look for lowest discrepancy which happens after 90 neighbors. In the right chart in Figure 10, we specifically look at the accuracy scores for $k = [90,95,100,105,110]$. From here we can see that $k =$ has the lowest discrepancy and higher test accuracy in comparison to the previous run. We were able to increase the test accuracy score by 6%. In future application, it may be wise to use PCA with a different model as well. Lastly, over sampling could have made a significant impact by receiving more much more information for the model to be trained on.

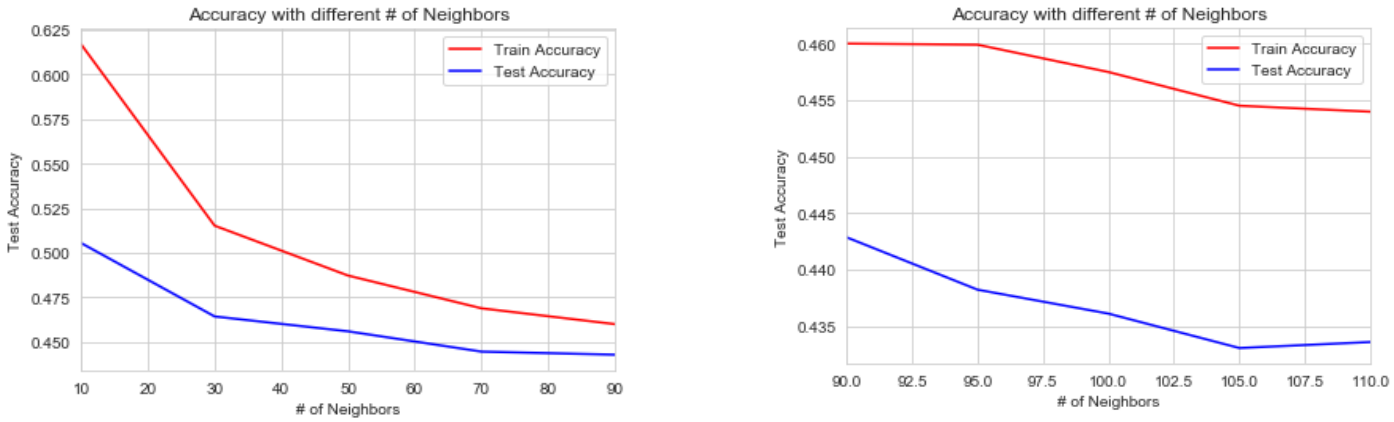

Figure 10: Range of Accuracy Scores for Range of K and Zoomed in Range

KNN Version	Test Accuracy
Under Sampling Version	38 %
Over Sampling and PCA Version	44 %

Table 12:Test Accuracy for Different Iterations of KNN

Summary

We performed pre-processing and filtering of the data to properly setup the data. We performed data analysis to see which features had great predictive power. Next, we normalized the features to be used in the classification of the data and created a correlation matrix to ensure that the data was correlated. There was an issue of imbalance, therefore we applied a subsampling technique. The data was split using the 80/20 approach to ensure a balanced sample was taken from each class. The KNN algorithm was applied in multiple iterations to find the optimal parameter of k. Variations of both the sampling of the data and the version of KNN were ran in order to find the optimal model to be used, including applying PCA. The best global accuracy was accomplished using a KNN value of with the application of over sampling and PCA. To conclude, even though it was possible to improve our results using PCA and subsampling, we see that KNN is not a great model for this classification and other models should be explored.

Acknowledgements

The author gratefully acknowledges the guidance of Professor Robert Azencott.

Fonts Dataset Source

This dataset was taken from the Kaggle website. The dataset can be found at following link:
https://www.kaggle.com/nipunarora8/age-gender-and-ethnicity-face-data-csv

Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy import stats
import math
from numpy import genfromtxt
import png
from numpy import genfromtxt
from PIL import Image
from matplotlib.colors import ListedColormap
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import PCA
from scipy import stats

## Import up sound alert dependencies
from IPython.display import Audio, display


# In[2]:


# Read in csv
df = pd.read_csv('age_gender.csv')

df.head()


# # Step 1: Data Cleaning

# In[3]:


df['pixels'] = df['pixels'].map(lambda x: np.array(x.split(' '), dtype=np.float32))


# In[4]:
```

```
df = df[df.age <= 75]
df = df.reset_index(drop=True)


# In[5]:


df['age_group_i'] = pd.cut(df.age, [0, 25, 50, 75], labels= [1, 2, 3])


# In[6]:


conditions = [
    (df["age_group_i"] == 1) & (df['gender'] == 1),
    (df["age_group_i"] == 2) & (df['gender'] == 1),
    (df["age_group_i"] == 3) & (df['gender'] == 1),
    (df["age_group_i"] == 1) & (df['gender'] == 0),
    (df["age_group_i"] == 2) & (df['gender'] == 0),
    (df["age_group_i"] == 3) & (df['gender'] == 0)
]

choices = ['FY', 'FM', 'FO', 'MY', 'MM', 'MO']

df['class'] = np.select(conditions, choices)


# In[7]:


df = df.drop(columns = ['ethnicity', 'img_name'])


# In[8]:


df.isna().sum()


# In[9]:


# Aesthetics
sns.set_style("whitegrid")


# In[10]:


sns.countplot(df['class'].sort_values(), palette="Set1")
```

In[11]:

```
len(df)
```

In[12]:

```
image = Image.fromarray((df.pixels[22853].reshape(48,48)).astype(np.uint8))

image.convert('RGB')
```

In[13]:

```
col_names = []
for i in range(0,48):
    for j in range(0,48):
        col_names.append('r'+str(i)+'c'+str(j))

df[col_names] = pd.DataFrame(df.pixels.tolist(), index= df.index, columns = col_names)
```

Step 2: Discriminating Power, Standardization, and Correlation Matrix

In[14]:

```
select_feat = ['r18c23','r19c24','r23c6','r14c30','r5c12','class']
```

In[15]:

```
class_set = ['FY', 'FM', 'FO', 'MY', 'MM', 'MO']
```

In[16]:

```
select_df = df[select_feat]
```

In[17]:

```
fig, axes = plt.subplots(nrows=6, ncols=5, figsize = (15,18), sharex='col', sharey='row')
```

```
for i, column in enumerate(select_df.columns[:5]):
    sns.distplot(select_df[select_df['class'] == 'FY'][column], ax = axes[0,i], color = 'green').set_title('FY
Class')
    sns.distplot(select_df[select_df['class'] == 'FM'][column], ax = axes[1,i], color = 'red').set_title('FM
Class')
    sns.distplot(select_df[select_df['class'] == 'FO'][column], ax = axes[2,i], color = 'blue').set_title('FO
Class')
    sns.distplot(select_df[select_df['class'] == 'MY'][column], ax = axes[3,i], color =
'yellow').set_title('MY Class')
    sns.distplot(select_df[select_df['class'] == 'MM'][column], ax = axes[4,i], color =
'purple').set_title('MM Class')
    sns.distplot(select_df[select_df['class'] == 'MO'][column], ax = axes[5,i], color =
'orange').set_title('MO Class')


# In[18]:


# KS-test to Compare Histograms Between Pairs of Classes
 #   Null hypotheses: Two group histograms are equal
 #   Alternative hypotheses: Two group histograms are different (two-tailed)

KS_test_results_1_2 = {}
KS_test_results_1_3 = {}
KS_test_results_1_4 = {}
KS_test_results_1_5 = {}
KS_test_results_1_6 = {}
KS_test_results_2_3 = {}
KS_test_results_2_4 = {}
KS_test_results_2_5 = {}
KS_test_results_2_6 = {}
KS_test_results_3_4 = {}
KS_test_results_3_5 = {}
KS_test_results_3_6 = {}
KS_test_results_4_5 = {}
KS_test_results_4_6 = {}
KS_test_results_5_6 = {}

# loop over column_list and execute code explained above
for column in select_df.columns[:5]:
    groupFY = select_df[select_df['class'] == 'FY'][column]
    groupFM = select_df[select_df['class'] == 'FM'][column]
    groupFO = select_df[select_df['class'] == 'FO'][column]
    groupMY = select_df[select_df['class'] == 'MY'][column]
    groupMM = select_df[select_df['class'] == 'MM'][column]
    groupMO = select_df[select_df['class'] == 'MO'][column]
    # add the output to the dictionary
    KS_test_results_1_2[column] = stats.ks_2samp(groupFY,groupFM)
    KS_test_results_1_3[column] = stats.ks_2samp(groupFY,groupFO)
    KS_test_results_1_4[column] = stats.ks_2samp(groupFY,groupMY)
    KS_test_results_1_5[column] = stats.ks_2samp(groupFY,groupMM)
```

```
    KS_test_results_1_6[column] = stats.ks_2samp(groupFY,groupMO)
    KS_test_results_2_3[column] = stats.ks_2samp(groupFM,groupFO)
    KS_test_results_2_4[column] = stats.ks_2samp(groupFM,groupMY)
    KS_test_results_2_5[column] = stats.ks_2samp(groupFM,groupMM)
    KS_test_results_2_6[column] = stats.ks_2samp(groupFM,groupMO)
    KS_test_results_3_4[column] = stats.ks_2samp(groupFO,groupMY)
    KS_test_results_3_5[column] = stats.ks_2samp(groupFO,groupMM)
    KS_test_results_3_6[column] = stats.ks_2samp(groupFO,groupMO)
    KS_test_results_4_5[column] = stats.ks_2samp(groupMY,groupMM)
    KS_test_results_4_6[column] = stats.ks_2samp(groupMY,groupMO)
    KS_test_results_5_6[column] = stats.ks_2samp(groupMM,groupMO)


    # KS Test
KS_results_df_1_2 = pd.DataFrame.from_dict(KS_test_results_1_2,orient='Index')
KS_results_df_1_2.columns = ['KS statistic for Group FY & FM','pvalue for Group FY & FM']
KS_results_df_1_3 = pd.DataFrame.from_dict(KS_test_results_1_3,orient='Index')
KS_results_df_1_3.columns = ['KS statistic for Group FY & FO','pvalue for Group FY & FO']
KS_results_df_1_4 = pd.DataFrame.from_dict(KS_test_results_1_4,orient='Index')
KS_results_df_1_4.columns = ['KS statistic for Group FY & MY','pvalue for Group FY & MY']
KS_results_df_1_5 = pd.DataFrame.from_dict(KS_test_results_1_5,orient='Index')
KS_results_df_1_5.columns = ['KS statistic for Group FY & MM','pvalue for Group FY & MM']
KS_results_df_1_6 = pd.DataFrame.from_dict(KS_test_results_1_6,orient='Index')
KS_results_df_1_6.columns = ['KS statistic for Group FY & MO','pvalue for Group FY & MO']
KS_results_df_2_3 = pd.DataFrame.from_dict(KS_test_results_2_3,orient='Index')
KS_results_df_2_3.columns = ['KS statistic for Group FM & FO','pvalue for Group FM & FO']
KS_results_df_2_4 = pd.DataFrame.from_dict(KS_test_results_2_4,orient='Index')
KS_results_df_2_4.columns = ['KS statistic for Group FM & MY','pvalue for Group FM & MY']
KS_results_df_2_5 = pd.DataFrame.from_dict(KS_test_results_2_5,orient='Index')
KS_results_df_2_5.columns = ['KS statistic for Group FM & MM','pvalue for Group FM & MM']
KS_results_df_2_6 = pd.DataFrame.from_dict(KS_test_results_2_6,orient='Index')
KS_results_df_2_6.columns = ['KS statistic for Group FM & MO','pvalue for Group FM & MO']
KS_results_df_3_4 = pd.DataFrame.from_dict(KS_test_results_3_4,orient='Index')
KS_results_df_3_4.columns = ['KS statistic for Group FO & MY','pvalue for Group FO & MY']
KS_results_df_3_5 = pd.DataFrame.from_dict(KS_test_results_3_5,orient='Index')
KS_results_df_3_5.columns = ['KS statistic for Group FO & MM','pvalue for Group FO & MM']
KS_results_df_3_6 = pd.DataFrame.from_dict(KS_test_results_3_6,orient='Index')
KS_results_df_3_6.columns = ['KS statistic for Group FO & MO','pvalue for Group FO & MO']
KS_results_df_4_5 = pd.DataFrame.from_dict(KS_test_results_4_5,orient='Index')
KS_results_df_4_5.columns = ['KS statistic for Group MY & MM','pvalue for Group MY & MM']
KS_results_df_4_6 = pd.DataFrame.from_dict(KS_test_results_4_6,orient='Index')
KS_results_df_4_6.columns = ['KS statistic for Group MY & MO','pvalue for Group MY & MO']
KS_results_df_5_6 = pd.DataFrame.from_dict(KS_test_results_5_6,orient='Index')
KS_results_df_5_6.columns = ['KS statistic for Group MM & MO','pvalue for Group MM & MO']


KS_test_Results_df = pd.concat([KS_results_df_1_2,
                        KS_results_df_1_3,
                        KS_results_df_1_4,
                        KS_results_df_1_5,
                        KS_results_df_1_6,
                        KS_results_df_2_3,
```

```
                    KS_results_df_2_4,
                    KS_results_df_2_5,
                    KS_results_df_2_6,
                    KS_results_df_3_4,
                    KS_results_df_3_5,
                    KS_results_df_3_6,
                    KS_results_df_4_5,
                    KS_results_df_4_6,
                    KS_results_df_5_6], axis=1)
KS_test_Results_df

# Discrimnation Power based of KS Results
pow12 = 1 - KS_test_Results_df.iloc[:,1]
pow13 = 1 - KS_test_Results_df.iloc[:,3]
pow14 = 1 - KS_test_Results_df.iloc[:,5]
pow15 = 1 - KS_test_Results_df.iloc[:,7]
pow16 = 1 - KS_test_Results_df.iloc[:,9]
pow23 = 1 - KS_test_Results_df.iloc[:,11]
pow24 = 1 - KS_test_Results_df.iloc[:,13]
pow25 = 1 - KS_test_Results_df.iloc[:,15]
pow26 = 1 - KS_test_Results_df.iloc[:,17]
pow34 = 1 - KS_test_Results_df.iloc[:,19]
pow35 = 1 - KS_test_Results_df.iloc[:,21]
pow36 = 1 - KS_test_Results_df.iloc[:,23]
pow45 = 1 - KS_test_Results_df.iloc[:,25]
pow46 = 1 - KS_test_Results_df.iloc[:,27]
pow56 = 1 - KS_test_Results_df.iloc[:,29]

Discrim_Power = pd.concat([pow12,
                    pow13,
                    pow14,
                    pow15,
                    pow16,
                    pow23,
                    pow24,
                    pow25,
                    pow26,
                    pow35,
                    pow36,
                    pow45,
                    pow46,
                    pow56], axis=1)
# Discrim_Power.columns = ['KS - Discriminating Power of Group FY & FM', 'KS - Discriminating
Power of Group FY & FO', 'KS - Discriminating Power of Group FM & FO']


# In[19]:


KS_test_Results_df
```

In[20]:

Discrim_Power

In[21]:

```
X = df.iloc[:,5:]

y = df.iloc[:,4]
```

In[22]:

```
X_classi = []

for i in class_set:
    X_classi.append(df[df['class'] == i].iloc[:,5:])
```

In[23]:

```
sXi = []
for i in X_classi:
    scaler = StandardScaler()
    scaler.fit(i)
    sXi.append(pd.DataFrame(scaler.transform(i), columns = col_names))
```

In[24]:

```
sXi_corr =[]

for i in sXi:
    sXi_corr.append(i.corr())
```

In[25]:

```
corr_frames = []
for i in sXi_corr:
    corr_frames.append(pd.DataFrame(i.unstack().sort_values(kind = 'quicksort',
ascending=False).drop_duplicates()[1:5]))
```

In[26]:

```
corr_frames[0]
```

In[27]:

```
corr_frames[1]
```

In[28]:

```
corr_frames[2]
```

In[29]:

```
corr_frames[3]
```

In[30]:

```
corr_frames[4]
```

In[31]:

```
corr_frames[5]
```

In[32]:

```
scaler = StandardScaler()
scaler.fit(X)
sX = scaler.transform(X)
sX = pd.DataFrame(sX, columns = col_names)
```

In[33]:

```
sX_corr = sX.corr()
```

In[34]:

```
pd.DataFrame(sX_corr.unstack().sort_values(kind = 'quicksort',
ascending=False).drop_duplicates()[1:11])
```

In[35]:

```
sX = pd.concat([sX, df.iloc[:,0:2]], axis=1)
```

Step 3: Balance Dataset by Undersample

In[36]:

```
rus = RandomUnderSampler(random_state=0, sampling_strategy = 'not minority')
sX_res, y_res = rus.fit_resample(sX, y)
print('Resampled dataset shape %s' % Counter(y_res))
```

In[37]:

```
sns.countplot(y_res, palette = 'Set1')
```

In[38]:

```
sres_df = pd.DataFrame(sX_res)
sres_df['class'] = y_res
len(sres_df)
```

Step 4: Split Train and Test Set

In[39]:

```
# train and test split
sXFY_train, sXFY_test, clFY_train, clFY_test = train_test_split(sres_df[sres_df['class'] ==
'FY'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FY'].iloc[:,2306], test_size=0.2, random_state=0)
sXFM_train, sXFM_test, clFM_train, clFM_test = train_test_split(sres_df[sres_df['class'] ==
'FM'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FM'].iloc[:,2306], test_size=0.2, random_state=0)
sXFO_train, sXFO_test, clFO_train, clFO_test = train_test_split(sres_df[sres_df['class'] ==
'FO'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FO'].iloc[:,2306], test_size=0.2, random_state=0)
sXMY_train, sXMY_test, clMY_train, clMY_test = train_test_split(sres_df[sres_df['class'] ==
'MY'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MY'].iloc[:,2306], test_size=0.2, random_state=0)
```

```
sXMM_train, sXMM_test, clMM_train, clMM_test = train_test_split(sres_df[sres_df['class'] ==
'MM'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MM'].iloc[:,2306], test_size=0.2, random_state=0)
sXMO_train, sXMO_test, clMO_train, clMO_test = train_test_split(sres_df[sres_df['class'] ==
'MO'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MO'].iloc[:,2306], test_size=0.2, random_state=0)


# In[40]:


X_train = pd.concat([sXFY_train, sXFM_train, sXFO_train, sXMY_train, sXMM_train, sXMO_train])
X_test = pd.concat([sXFY_test, sXFM_test, sXFO_test, sXMY_test, sXMM_test, sXMO_test])
y_train = pd.concat([clFY_train, clFM_train, clFO_train, clMY_train, clMM_train, clMO_train])
y_test = pd.concat([clFY_test, clFM_test, clFO_test, clMY_test, clMM_test, clMO_test])


# In[41]:


len(sXFM_train)


# In[42]:


print(len(X_train), len(X_test), len(y_train), len(y_test))


# # Step 5: Apply KNN Models

# In[43]:


k = [10,30,50,70,90,110]

y_pred_train_ki = []
y_pred_test_ki = []

for i in k:
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train, y_train)
    y_pred_train_ki.append(classifier.predict(X_train))
    y_pred_test_ki.append(classifier.predict(X_test))


# In[44]:


train_perf_k = []
test_perf_k = []

for i in y_pred_train_ki:
    train_perf_k.append(accuracy_score(y_true = y_train, y_pred = i))
```

```
for i in y_pred_test_ki:
    test_perf_k.append(accuracy_score(y_true = y_test, y_pred = i))


# In[45]:


train_perf_k


# In[46]:


test_perf_k


# In[47]:


sns.lineplot(x = k, y = train_perf_k, color = 'red', label = "Train Accuracy")
sns.lineplot(x = k, y = test_perf_k, color = 'blue', label = "Test Accuracy")

plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Test Accuracy')
plt.xlim(10, 110)


# In[48]:


k = [100,105,110,115,120,125]

y_pred_train_ki = []
y_pred_test_ki = []

for i in k:
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train, y_train)
    y_pred_train_ki.append(classifier.predict(X_train))
    y_pred_test_ki.append(classifier.predict(X_test))


# In[49]:


train_perf_k = []
test_perf_k = []

for i in y_pred_train_ki:
    train_perf_k.append(accuracy_score(y_true = y_train, y_pred = i))
for i in y_pred_test_ki:
```

```
    test_perf_k.append(accuracy_score(y_true = y_test, y_pred = i))

# In[50]:


train_perf_k

# In[51]:


test_perf_k

# In[52]:


for i in range(0,len(train_perf_k)):
    print(train_perf_k[i] - test_perf_k[i])

# In[53]:


sns.lineplot(x = k, y = train_perf_k, color = 'red', label = "Train Accuracy")
sns.lineplot(x = k, y = test_perf_k, color = 'blue', label = "Test Accuracy")

plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Test Accuracy')
plt.xlim(100, 125)

# In[54]:


y_pred_train_ki[3]

# In[55]:


train_perf_k[3]

# # Step 6: Confusion Matrices

# In[56]:


cmtx_a_train = pd.DataFrame(
```

```
    confusion_matrix(y_true=y_train,y_pred = y_pred_train_ki[3], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])


# In[57]:


cmtx_a_train


# In[58]:


fig = plt.subplots(figsize = (14,10))

sns.heatmap(cmtx_a_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# In[59]:


cmtx_a_test = pd.DataFrame(
    confusion_matrix(y_true=y_test,y_pred = y_pred_test_ki[3], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])


# In[60]:


cmtx_a_test


# In[61]:


fig = plt.subplots(figsize = (14,9))

sns.heatmap(cmtx_a_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# # Step 7: Resample by Over Sampling

# In[62]:


ros = RandomOverSampler(random_state=0, sampling_strategy = 'not majority')
sX_res, y_res = ros.fit_resample(sX, y)
print('Resampled dataset shape %s' % Counter(y_res))
```

In[63]:

```
sns.countplot(y_res.sort_values(), palette = 'Set1')
```

In[64]:

```
sres_df = pd.DataFrame(sX_res)
sres_df['class'] = y_res
len(sres_df)
```

In[65]:

```
# train and test split
sXFY_train, sXFY_test, clFY_train, clFY_test = train_test_split(sres_df[sres_df['class'] ==
'FY'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FY'].iloc[:,2306], test_size=0.2, random_state=0)
sXFM_train, sXFM_test, clFM_train, clFM_test = train_test_split(sres_df[sres_df['class'] ==
'FM'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FM'].iloc[:,2306], test_size=0.2, random_state=0)
sXFO_train, sXFO_test, clFO_train, clFO_test = train_test_split(sres_df[sres_df['class'] ==
'FO'].iloc[:,0:2304], sres_df[sres_df['class'] == 'FO'].iloc[:,2306], test_size=0.2, random_state=0)
sXMY_train, sXMY_test, clMY_train, clMY_test = train_test_split(sres_df[sres_df['class'] ==
'MY'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MY'].iloc[:,2306], test_size=0.2, random_state=0)
sXMM_train, sXMM_test, clMM_train, clMM_test = train_test_split(sres_df[sres_df['class'] ==
'MM'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MM'].iloc[:,2306], test_size=0.2, random_state=0)
sXMO_train, sXMO_test, clMO_train, clMO_test = train_test_split(sres_df[sres_df['class'] ==
'MO'].iloc[:,0:2304], sres_df[sres_df['class'] == 'MO'].iloc[:,2306], test_size=0.2, random_state=0)
```

In[66]:

```
X_train = pd.concat([sXFY_train, sXFM_train, sXFO_train, sXMY_train, sXMM_train, sXMO_train])
X_test = pd.concat([sXFY_test, sXFM_test, sXFO_test, sXMY_test, sXMM_test, sXMO_test])
y_train = pd.concat([clFY_train, clFM_train, clFO_train, clMY_train, clMM_train, clMO_train])
y_test = pd.concat([clFY_test, clFM_test, clFO_test, clMY_test, clMM_test, clMO_test])
```

In[67]:

```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

Step 8: Apply Principal Component Analysis

In[69]:

```
pca = PCA(.95)
pca.fit(X_train)


# In[70]:


pca.n_components_


# In[71]:


X_train = pca.transform(X_train)
X_test = pca.transform(X_test)


# In[72]:


X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)


# # Step 9: Apply KNN on PCA Dataset

# In[ ]:


k = [10,30,50,70,90]

y_pred_train_ki = []
y_pred_test_ki = []

for i in k:
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train, y_train)
    y_pred_train_ki.append(classifier.predict(X_train))
    y_pred_test_ki.append(classifier.predict(X_test))


# In[ ]:


train_perf_k = []
test_perf_k = []

for i in y_pred_train_ki:
    train_perf_k.append(accuracy_score(y_true = y_train, y_pred = i))
for i in y_pred_test_ki:
```

```
    test_perf_k.append(accuracy_score(y_true = y_test, y_pred = i))


# In[ ]:


train_perf_k


# In[ ]:


test_perf_k


# In[ ]:


for i in range(0,len(train_perf_k)):
    print(train_perf_k[i] - test_perf_k[i])


# In[ ]:


sns.lineplot(x = k, y = train_perf_k, color = 'red', label = "Train Accuracy")
sns.lineplot(x = k, y = test_perf_k, color = 'blue', label = "Test Accuracy")

plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Test Accuracy')
plt.xlim(10, 90)


# In[ ]:


k = [90,95,100,105,110]

y_pred_train_ki = []
y_pred_test_ki = []

for i in k:
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train, y_train)
    y_pred_train_ki.append(classifier.predict(X_train))
    y_pred_test_ki.append(classifier.predict(X_test))

train_perf_k = []
test_perf_k = []

for i in y_pred_train_ki:
```

```
    train_perf_k.append(accuracy_score(y_true = y_train, y_pred = i))
for i in y_pred_test_ki:
    test_perf_k.append(accuracy_score(y_true = y_test, y_pred = i))


# In[ ]:


train_perf_k


# In[ ]:


test_perf_k


# In[ ]:


for i in range(0,len(train_perf_k)):
    print(train_perf_k[i] - test_perf_k[i])


# In[ ]:


sns.lineplot(x = k, y = train_perf_k, color = 'red', label = "Train Accuracy")
sns.lineplot(x = k, y = test_perf_k, color = 'blue', label = "Test Accuracy")

plt.title('Accuracy with different # of Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Test Accuracy')
plt.xlim(90, 110)


# In[ ]:


cmtx_a_train = pd.DataFrame(
    confusion_matrix(y_true=y_train,y_pred = y_pred_train_ki[3], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])

fig = plt.subplots(figsize = (14,10))

sns.heatmap(cmtx_a_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# In[ ]:
```

```
cmtx_a_test = pd.DataFrame(
    confusion_matrix(y_true=y_test,y_pred = y_pred_test_ki[3], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])

fig = plt.subplots(figsize = (14,9))

sns.heatmap(cmtx_a_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# In[ ]:


k = [115]

y_pred_train_ki = []
y_pred_test_ki = []

for i in k:
    classifier = KNeighborsClassifier(n_neighbors = i)
    classifier.fit(X_train, y_train)
    y_pred_train_ki.append(classifier.predict(X_train))
    y_pred_test_ki.append(classifier.predict(X_test))

train_perf_k = []
test_perf_k = []

for i in y_pred_train_ki:
    train_perf_k.append(accuracy_score(y_true = y_train, y_pred = i))
for i in y_pred_test_ki:
    test_perf_k.append(accuracy_score(y_true = y_test, y_pred = i))


# In[ ]:


print(train_perf_k)
print(test_perf_k)


# In[ ]:


cmtx_a_train = pd.DataFrame(
    confusion_matrix(y_true=y_train,y_pred = y_pred_train_ki[0], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])

fig = plt.subplots(figsize = (14,10))
```

```
sns.heatmap(cmtx_a_train, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# In[ ]:


cmtx_a_test = pd.DataFrame(
    confusion_matrix(y_true=y_test,y_pred = y_pred_test_ki[0], labels = ['FY', 'FM', 'FO', 'MY', 'MM',
'MO'], normalize = 'true'),
    index=['true:FY', 'true:FM', 'true:FO', 'true:MY', 'true:MM', 'true:MO'],
    columns=['pred:FY', 'pred:FM', 'pred:FO', 'pred:MY', 'pred:MM', 'pred:MO'])

fig = plt.subplots(figsize = (14,10))

sns.heatmap(cmtx_a_test, annot=True, fmt = '.02%', annot_kws={"size": 14}, vmin = 0, vmax = 1)


# In[ ]:
```