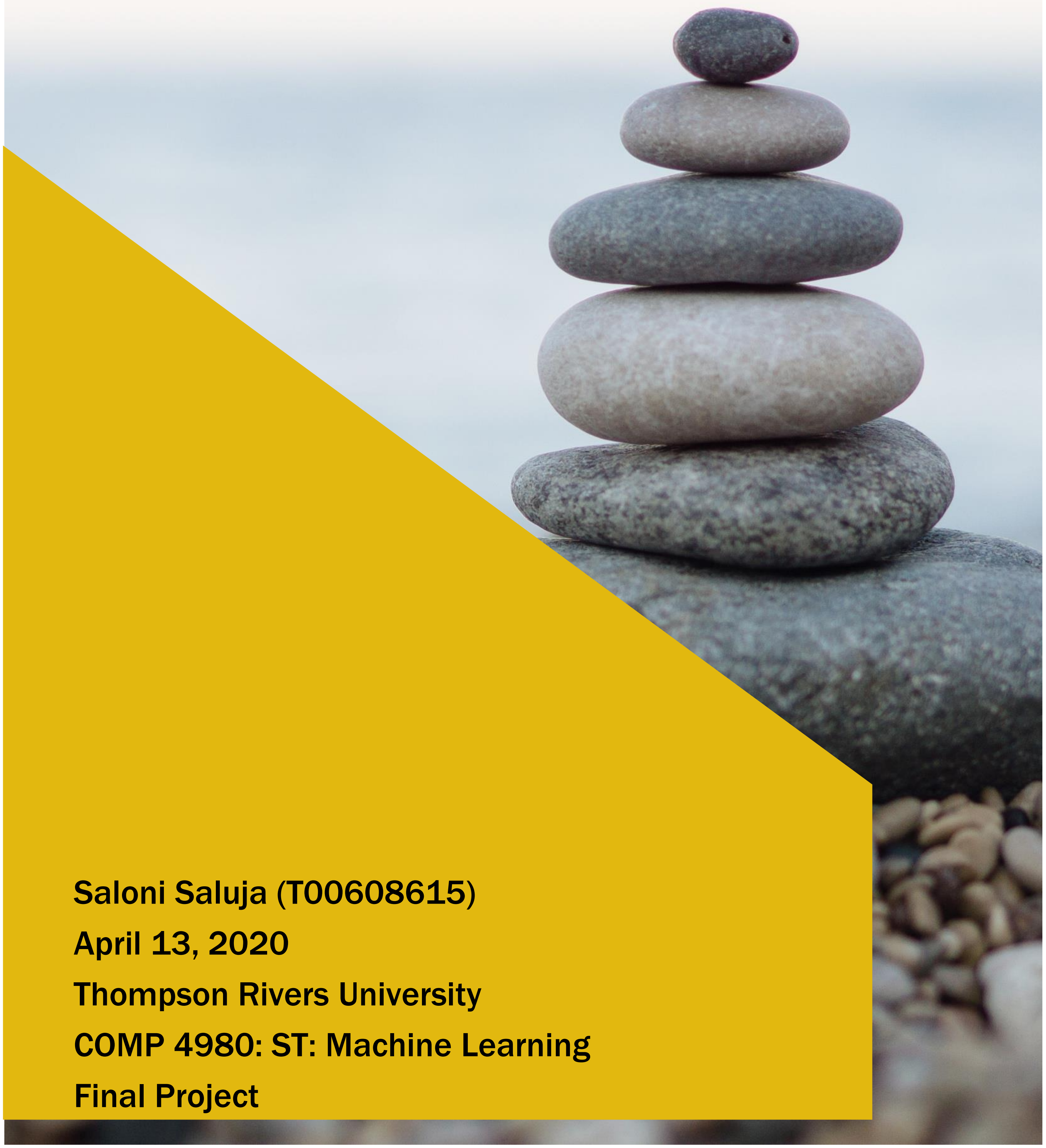
# Mental Health in Tech



Saloni Saluja (T00608615)

April 13, 2020

Thompson Rivers University

COMP 4980: ST: Machine Learning

Final Project

# Mental Health in Tech

Link to Google Colab Notebook:
https://colab.research.google.com/drive/1pdgZObZjJfLVkvCbDD7iIMzkiNkPyyuT?usp=sharing

## 1. Data Description

### About the dataset

This dataset contains information from a 2014 survey that measures attitudes towards mental health and frequency of mental health disorders in the tech workplace. This is a public dataset available at Kaggle.com. [1]

### Type of data in the dataset

This dataset contains a mix of categorical and text data. The categorical data is mostly binary, with a few ambiguous and some null values. It also contains a few fields that have numerical data, such as Age.

### Format

The format of the dataset file is .csv.

### Volume

This dataset contains a total of 27 columns. 26 columns are String or Object Type. 5 columns have Boolean values. 1 column has integer datatype. The number of rows contained in the dataset are 1259.

### Variety

The fields contained in this dataset are as under:

1. **Timestamp*:*** Stores the date and time for the record. [1]

2. **Age**: Records the age of the respondent. [1]

3. **Gender**: Records the gender of the respondent. [1]

4. **Country**: Records the country that the respondent belongs to. [1]

5. **State**: This dataset is primarily focused on people living in the US. Hence, this column records the state which the respondent is from (if they live in the US). [1]

6. **Self_employed**: This field records if the respondent is self employed or not. [1]

7. **Family_history**: Records if the respondent has family history of mental illnesses. [1]

8. **Treatment**: Records if the respondent has undertaken any medication or treatment for their mental health condition. [1]

9. **Work_interface**: This records if the respondent's mental health condition is affecting their performance at work in a negative way. [1]

10. **No_employees**: Number of employees working in the respondent's company. [1]

11. **Remote Work**: Records if the respondent works remotely atleast 50% of the time. [1]

12. **Tech_Company**: Records if the respondent's employer is primarily a tech employer. [1]

13. **Benefits**: Records if the company provides any mental health benefits to their employees. [1]

14. **Care_options**: Records if the respondent is aware of the mental health care provided by their employer. [1]

15. **Wellness_Program**: Records if the respondent's employer has ever discussed their mental health as a part of "an employee wellness program." [1]

16. **Seek_help**: Records if the respondent's employer provides resources to learn more about mental health issues and how to seek help. [1]

17. **Anonymity**: If the respondent chooses to make use of mental health or substance abuse program, is their anonymity maintained or not. [1]

18. **Leave**: How easy is it for a respondent to take a few days off work due to mental health reasons. [1]

19. **MentalHealthConsequence**: Records if the respondent feels like discussing a mental health issue with their employer would lead to negative consequences. [1]

20. **PhysHealthConsequence**: Records if the respondent feels like discussing a physical health issue with their employer would lead to negative consequences. [1]

21. **Co-workers:** Will the respondent be willing to discuss any mental health issue with their co-workers? [1]

22. **Supervisor**: Will the respondent be willing to discuss their mental health with their supervisor? [1]
23. **MentalHealthInterview**: Would a respondent choose to talk about their mental health issue with a potential employer in an interview? [1]

24. **PhysHealthInterview**: Would a respondent choose to talk about any physical health issue with a potential employer during an interview? [1]

25. **MentalVSPhysical**: Does the respondent's employer take mental health as seriously as physical health. [1]

26. **Obs_consequence**: Has the respondent heard or seen any negative consequences for co-workers who have mental health conditions at their workplace. [1]

27. **Comments**: Any additional comments or notes. [1]

## Veracity

This dataset contains lots of string values that cannot be taken in directly by our machine learning model. The data will need a good amount of preprocessing before it can be fed into any machine learning classifier. It has several null values and unintegrated data. For instance, gender was left as an open-ended question. This kind of data needs to be cleaned.

### Velocity

This dataset is time-based. It records contained here range from 2014 to 2015. There is one record that belongs to 2016 as well. The time interval between some records is minutes. For some other records, it is days, months, as well as years. The data spans from 2014 to 2015, and has one record that belongs to 2016.

### Value

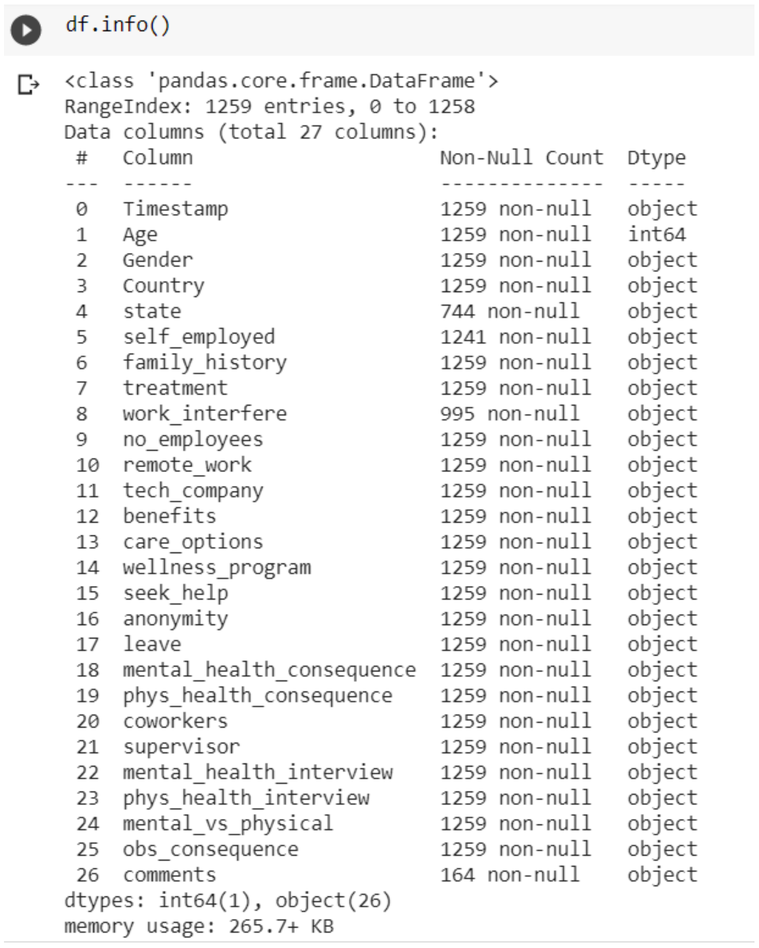
This dataset can be used to answer many question related to mental health. Some of them are listed below: [1]

1. What is the frequency of mental health illnesses by geographic location and how the attitudes of people living in different geographic regions vary when it comes to mental health?
2. What are some of the strongest predictors of mental health illnesses?
3. What are the attitudes of people towards mental health in the workplace?

# 2. Data Analysis

## Initial Analysis

The dataset was loaded in the Pandas Dataframe. It has 26 columns and 1259 rows. The columns and the numbers of values contained in each field are shown below:



```
df.info()
```

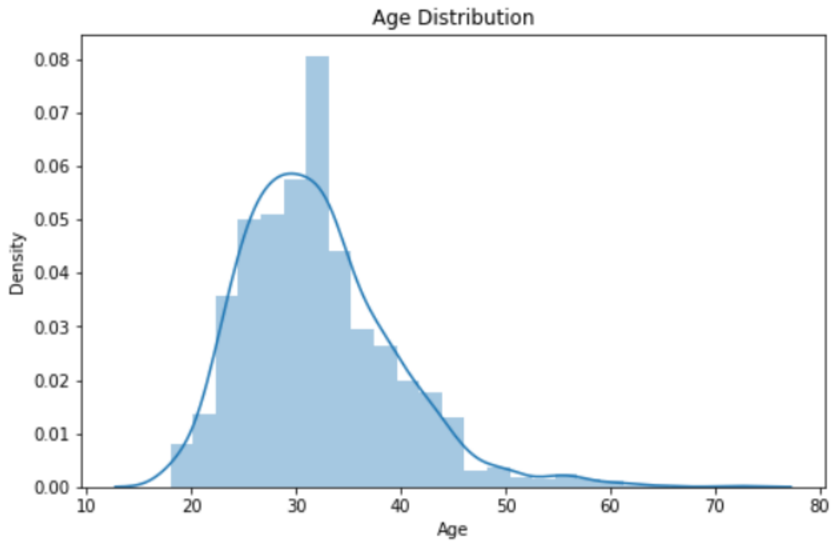
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Timestamp                  1259 non-null   object
 1   Age                        1259 non-null   int64 
 2   Gender                     1259 non-null   object
 3   Country                    1259 non-null   object
 4   state                      744 non-null    object
 5   self_employed              1241 non-null   object
 6   family_history             1259 non-null   object
 7   treatment                  1259 non-null   object
 8   work_interfere             995 non-null    object
 9   no_employees               1259 non-null   object
 10  remote_work                1259 non-null   object
 11  tech_company               1259 non-null   object
 12  benefits                   1259 non-null   object
 13  care_options               1259 non-null   object
 14  wellness_program           1259 non-null   object
 15  seek_help                  1259 non-null   object
 16  anonymity                  1259 non-null   object
 17  leave                      1259 non-null   object
 18  mental_health_consequence  1259 non-null   object
 19  phys_health_consequence    1259 non-null   object
 20  coworkers                  1259 non-null   object
 21  supervisor                 1259 non-null   object
 22  mental_health_interview    1259 non-null   object
 23  phys_health_interview      1259 non-null   object
 24  mental_vs_physical         1259 non-null   object
 25  obs_consequence            1259 non-null   object
 26  comments                   164 non-null    object
dtypes: int64(1), object(26)
memory usage: 265.7+ KB
```

For our analysis, we won't be using Timestamp, state, no_employees, and comments. Therefore, we dropped these columns from our dataframe.

The Age distribution plot depicts that Age has continuous values. Most of the age distributions were centered around 25 to 35.

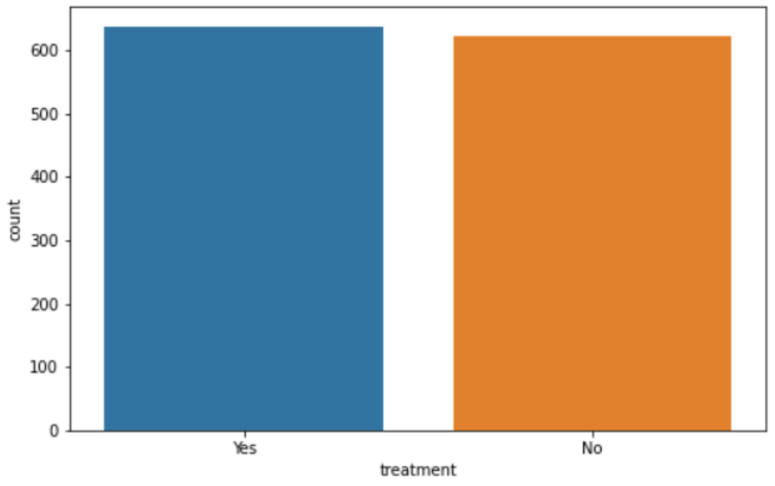
```
plt.figure(figsize=(8,5))
sns.distplot(df["Age"], bins=25)
plt.title("Age Distribution")
plt.xlabel("Age")
plt.show
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:25
  warnings.warn(msg, FutureWarning)
<function matplotlib.pyplot.show>
```



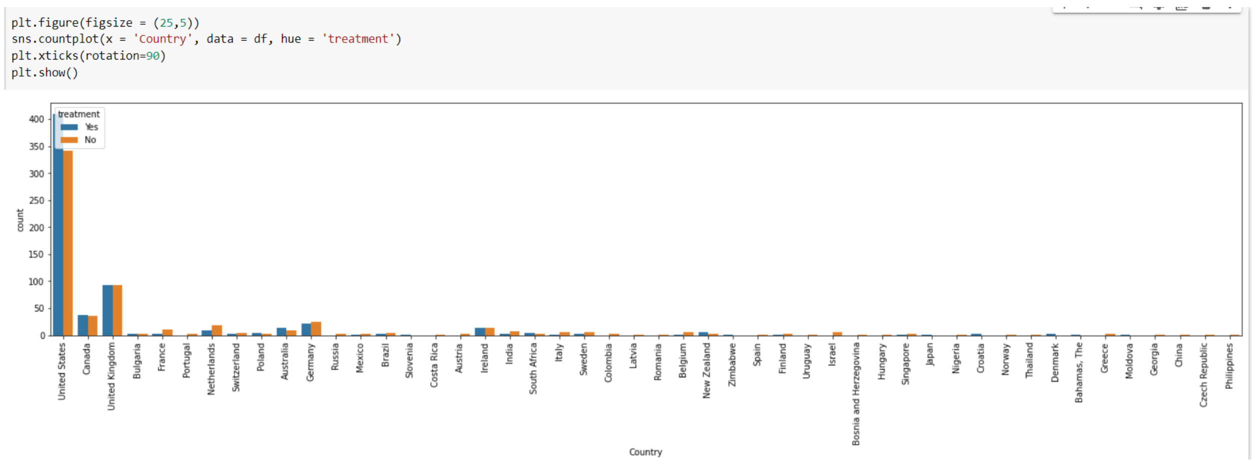
The treatment column has binary values. 637 respondents said that they had taken some kind of treatment for this mental health condition; 622 respondents said that they didn't go for any treatment.

```
[31] plt.figure(figsize = (8,5))
     sns.countplot(x = 'treatment', data = df)
     plt.show()
```



The figure below shows whether individuals opted for treatment or not based on Country. In the US, about 420 people were/are undergoing treatment; however, about

340 are not. For Canada and UK, the Yes/No ratios are almost the same. However, in most European and Middle East and Asian countries – France, Portugal, Netherlands, Switzerland, Germany, Russia, Mexico, Brazil, Costa Rica, Austria, India, Italy, Sweden, Colombia, Latvia, Romania, Belgium, Spain, Finland, Uruguay, Israel, Bosnia and Herzegovinda, Hungary, Singapore, Nigeria, Norway, Thailand, Greece, Georgia, China, Czech Republic, and Phillippines, the people who did not undergo treatment (no responses) exceeded the people who did undergo a treatment (yes responses).



```
plt.figure(figsize = (25,5))
sns.countplot(x = 'Country', data = df, hue = 'treatment')
plt.xticks(rotation=90)
plt.show()
```

[Please refer to Treatment with Country.png in the root folder for enlarged image]

However, the important thing to note here is that the data is primarily focused on USA. In most cases, the number of respondents from each of these countries varies from 1 to 7 in most cases. Therefore, this data may not be an actual representative of whether people in the East are undergoing treatment for their mental health conditions or not. It needs more digging, and a greater sample size focused on countries other than the US.

```
df.Country.value_counts()
```

```
United States             751
United Kingdom            185
Canada                     72
Germany                    45
Ireland                    27
Netherlands                27
Australia                  21
France                     13
India                      10
New Zealand                 8
Switzerland                 7
Sweden                      7
Poland                      7
Italy                       7
Brazil                      6
South Africa                6
Belgium                     6
Israel                      5
Singapore                   4
Bulgaria                    4
Finland                     3
Austria                     3
Russia                      3
Mexico                      3
Greece                      2
Colombia                    2
Denmark                     2
Portugal                    2
Croatia                     2
Philippines                 1
China                       1
Spain                       1
Norway                      1
Romania                     1
Georgia                     1
Latvia                      1
Slovenia                    1
Nigeria                     1
Bosnia and Herzegovina      1
Zimbabwe                    1
Hungary                     1
Moldova                     1
Uruguay                     1
Thailand                    1
Bahamas, The                1
Costa Rica                  1
Japan                       1
Czech Republic              1
Name: Country, dtype: int64
```

## Dealing with Null Values

The self_employed column contains 18 null values and the work_interfere column contains 264 null values.

```
pd.options.display.max_rows = 200
df.isna().sum(axis = 0)
```

```
Age                         0
Gender                      0
Country                     0
self_employed              18
family_history              0
treatment                   0
work_interfere            264

remote_work                  0
tech_company                 0
benefits                     0
care_options                 0
wellness_program             0
seek_help                    0
anonymity                    0
leave                        0
mental_health_consequence    0
phys_health_consequence      0
coworkers                    0
supervisor                   0
mental_health_interview      0
phys_health_interview        0
mental_vs_physical           0
obs_consequence              0
dtype: int64
```

1. self_employed: The self employed column had 1241 records. 18 records were missing. This column contained binary values – Yes or No. To fill out the missing values, we used "Don't know" as the third value. Now, the self_employed column had 3 values – Yes, No, and Don't know.

```
[25] df.self_employed.value_counts()
```

```
     No            1095
     Yes            146
     Don't know      18
     Name: self_employed, dtype: int64
```

1095 people were not self-employed, 146 were self-employed, and we don't have any information about 18 people.

2. work_interfere: The work_interfere column had 995 values. 144 values were missing. This column records the degree of interference that a person's mental health condition had on their work. Values were recorded using 4 levels – Sometimes, Never, Rarely, and Often. For people who did not respond, we entered the value as "Don't same" (same thing as we did for self_employed). The updated values contained in this column are as under:

```
df.work_interfere.value_counts()
```

```
Sometimes     465
Don't know    264
Never         213
Rarely        173
Often         144
Name: work_interfere, dtype: int64
```

Now, the dataset doesn't have any null values.

```
pd.options.display.max_rows = 200
df.isna().sum(axis = 0)
```

```
Age                          0
Gender                       0
Country                      0
self_employed                0
family_history               0
treatment                    0
work_interfere               0
remote_work                  0
tech_company                 0
benefits                     0
care_options                 0
wellness_program             0
seek_help                    0
anonymity                    0
leave                        0
mental_health_consequence    0
phys_health_consequence      0
coworkers                    0
supervisor                   0
mental_health_interview      0
phys_health_interview        0
mental_vs_physical           0
obs_consequence              0
dtype: int64
```

## Dealing with Outliers

The Age column had lots of outliers. Some people had entered their age as a negative number, whereas somebody else said that they were 99999999 years old. Some people also claimed to be 5, 8, or 11 years old. This is not practical as the legal age for working in the US is 14 years. Thus, in terms of age, anything less than 14 or greater than 100 was considered to be an outlier.

```
print("Age Outliers \n")
for i in range(len(df['Age'])):
  if(df['Age'][i] < 14 or df['Age'][i] >100):
    print(df['Age'][i])
```

```
Age Outliers

-29
329
99999999999
-1726
5
8
11
-1
```

To remove outliers from the data, we replace the outliers by the maximum age in the dataset, which is 29.

```
for i in range(len(df['Age'])):
  if(df['Age'][i] < 14 or df['Age'][i] >100):
    df.replace(to_replace=df['Age'][i], value = 29, inplace=True)

print("Age", "\t", "Number of People")
df.Age.value_counts()
```

## Building Correlation Matrices

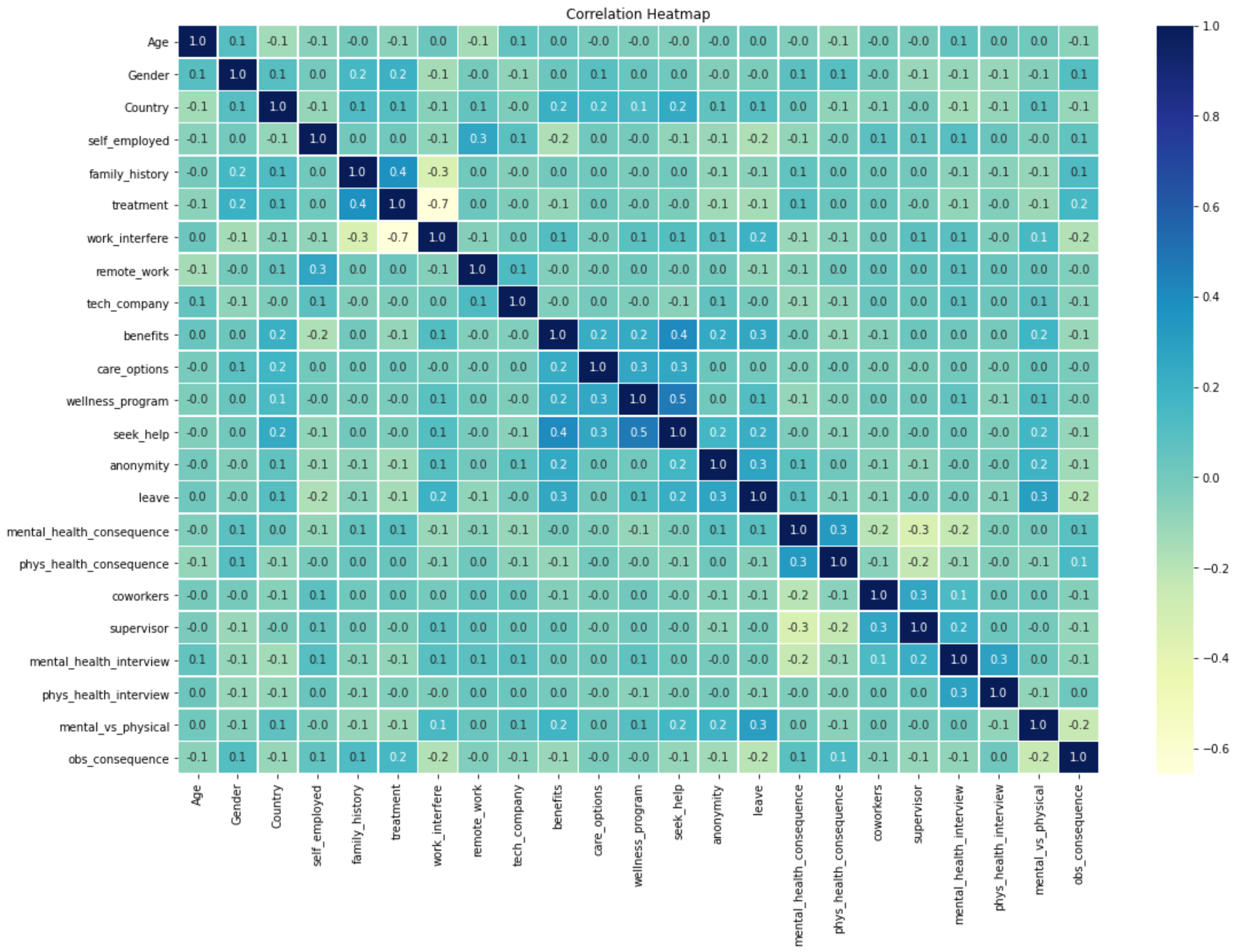
To find out the correlation between columns, we decided to build a correlation matrix. If two variables have a high positive correlation, the correlation box will have a very light shade. Similarly, if two variables have a high negative correlation, the correlation box will have a very dark shade.

Correlation matrices can only be built if the dataset has numeric values. In our case, we had string values. So, we had to convert all those values to numeric.

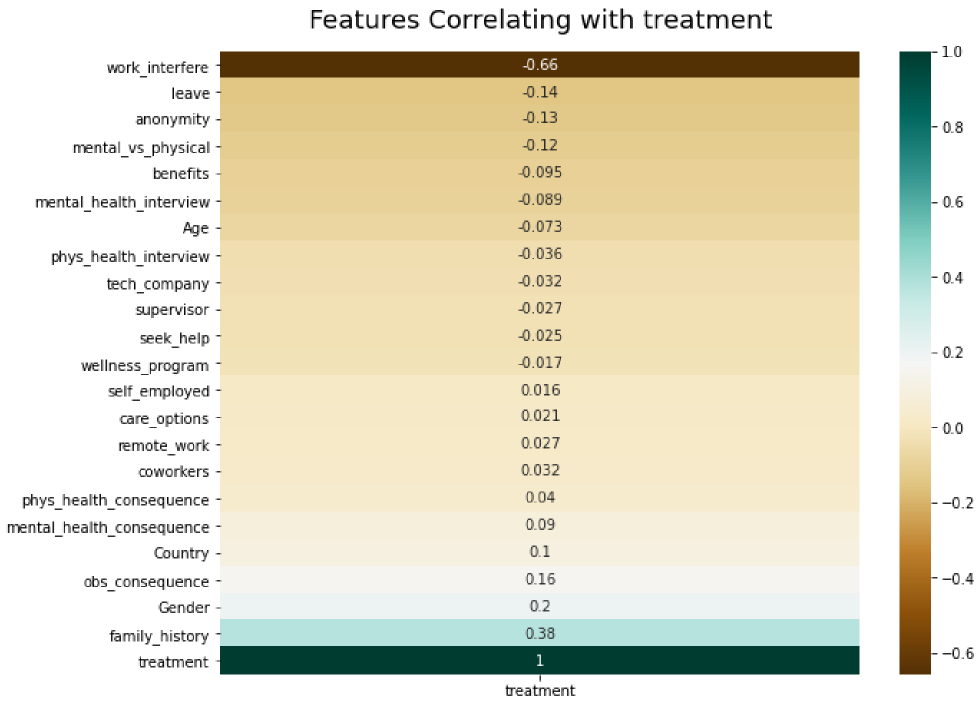
The original field value vs assigned integer value as follows:

| Actual Value | Assigned Value |
|---|---|
| Nonbinary | 0 |
| Female | 1 |
| Male | 2 |
| Don’t know, Not sure, Maybe | 0 |
| Yes | 1 |
| No | 2 |
| Never | 1 |
| Rarely | 2 |
| Sometimes | 3 |
| Often | 4 |
| Somewhat easy | 1 |
| Very Easy | 2 |
| Somewhat Difficult | 3 |
| Very Difficult | 4 |
| Some of them | 3 |
| United States, Canada | 1 |
| *Rest of the world | 2 |

After this, we converted the datatype of all the columns to integer and built the correlation heatmaps.



We are more interest in the treatment field. So here's a matrix that shows the features correlating with treatment.



Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. The close to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is. [2]

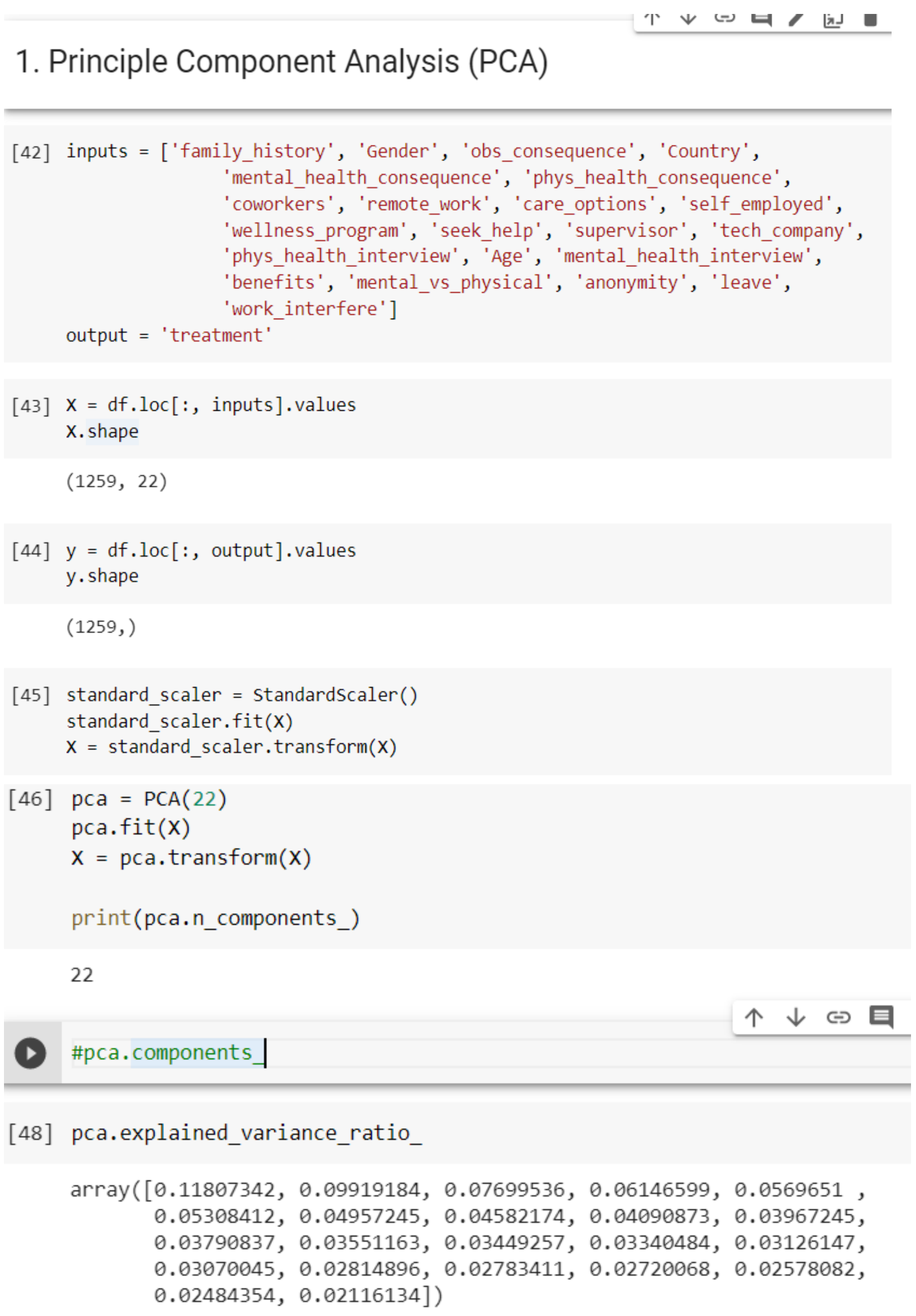
From the above figure, we can see that family_history has the highest positive correlation with treatment (0.38) and work interference has the highest negative correlation (-0.66). These values are still spaced from 1 and -1, i.e. if the score was more like -0.9 or +0.9, it would show a stronger correlation.

This leads us to our third phase, which is exploring the data to find out the principal components (using PCA) and exploring using Decision Trees.

# 3. Data Exploration

## 1. Principal Component Analysis

We started this phase by exploring the relative importance of the principal components and ended up getting some really interesting results. The methodology was to start with all the columns as inputs with treatment as output, standardize the values using standard scalar, apply PCA, and print out the variance ratio, and see which columns are necessary to explain a vast majority of the variance. This is exactly what we did.



## 1. Principle Component Analysis (PCA)

```
[42] inputs = ['family_history', 'Gender', 'obs_consequence', 'Country',
               'mental_health_consequence', 'phys_health_consequence',
               'coworkers', 'remote_work', 'care_options', 'self_employed',
               'wellness_program', 'seek_help', 'supervisor', 'tech_company',
               'phys_health_interview', 'Age', 'mental_health_interview',
               'benefits', 'mental_vs_physical', 'anonymity', 'leave',
               'work_interfere']
     output = 'treatment'
```

```
[43] X = df.loc[:, inputs].values
     X.shape

     (1259, 22)
```

```
[44] y = df.loc[:, output].values
     y.shape

     (1259,)
```

```
[45] standard_scaler = StandardScaler()
     standard_scaler.fit(X)
     X = standard_scaler.transform(X)
```

```
[46] pca = PCA(22)
     pca.fit(X)
     X = pca.transform(X)

     print(pca.n_components_)

     22
```

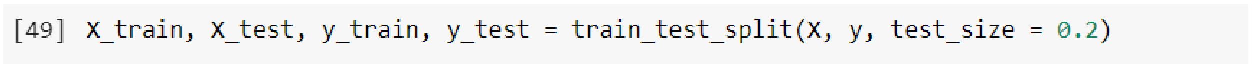
```
#pca.components_
```

```
[48] pca.explained_variance_ratio_

     array([0.11807342, 0.09919184, 0.07699536, 0.06146599, 0.0569651 ,
            0.05308412, 0.04957245, 0.04582174, 0.04090873, 0.03967245,
            0.03790837, 0.03551163, 0.03449257, 0.03340484, 0.03126147,
            0.03070045, 0.02814896, 0.02783411, 0.02720068, 0.02578082,
            0.02484354, 0.02116134])
```

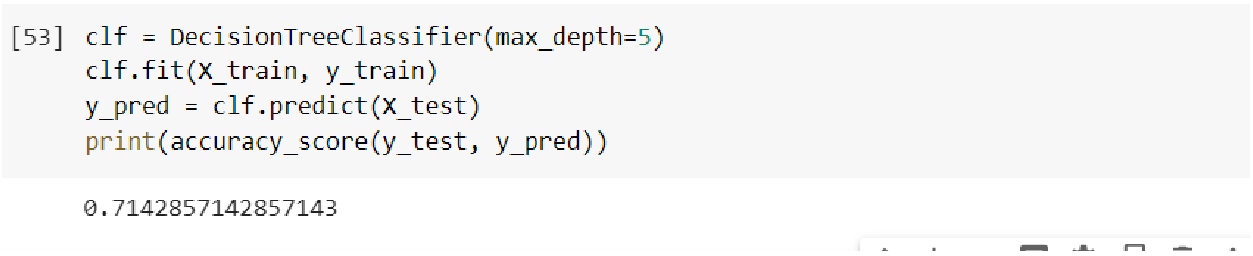
The number of components was initialized to 200. Interestingly, when we printed out the explained_variance_ratio_ , we found out that there is no component using which we can explain majority of the variance! The absolute maximum variance that can be explained using a single component is 11.8%. For other components, it is 9.91%, 7.69%,…., 2.11%. This leads us to conclude that there isn't any single component or a group of components that can be deemed as absolutely important or explaining a vast majority of the variance. The relative importance of all these fields is almost the same. So, change of plans! Instead of dropping irrelevant fields, we will keep all of them.

## 2. Decision Tree Classifier

Decision trees can be used to visually represent decisions and decision making. [3] To use Decision Tree Classifier, we started by splitting the dataset into training set and testing set. 80% records were used for training and 20% records for testing.



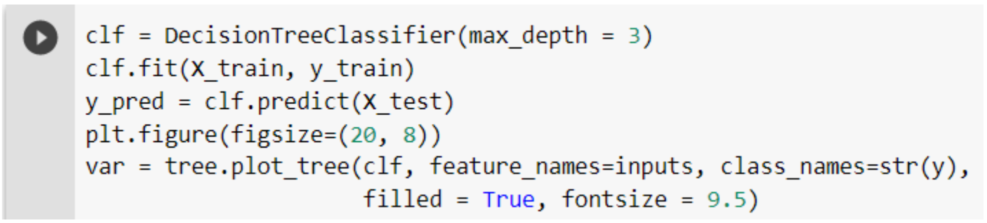
```
[49] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```



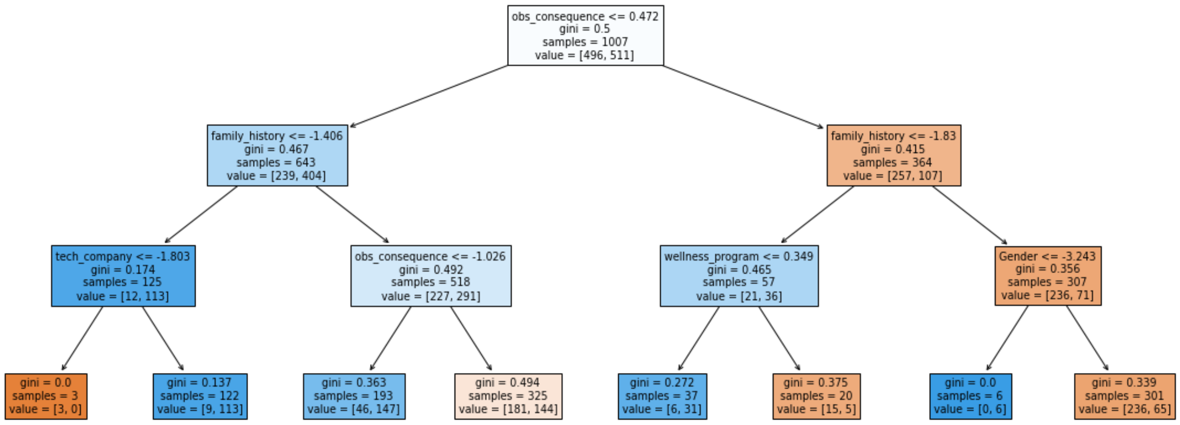
```
[53] clf = DecisionTreeClassifier(max_depth=5)
     clf.fit(X_train, y_train)
     y_pred = clf.predict(X_test)
     print(accuracy_score(y_test, y_pred))

     0.7142857142857143
```

The accuracy of Decision Tree classifier was 71.42%.

To understand the relationships between variables, we also included a visualization of the tree with a maximum depth of 3. The input and output fields were the same as PCA.



```
clf = DecisionTreeClassifier(max_depth = 3)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
plt.figure(figsize=(20, 8))
var = tree.plot_tree(clf, feature_names=inputs, class_names=str(y),
                     filled = True, fontsize = 9.5)
```



The first split was made using obs_consequence, which received a total number of samples equal to 1007. Out of this, 496 samples belonged to one class and 511 samples belonged to the second class (value = [496, 511]). The gini value for the root node is the highest. (0.5) As we go down, this value keeps decreasing. The root node checks if the value received is <= 0.472. If it is true, the values goes to the left node. If false, the value goes to the right node. This process is iterated over every single node.

Let's form a hypothesis and start testing it out!

# 4. Experimental Method

## Hypothesis

Our aim is to predict whether an individual will undergo treatment using attributes given in the dataset. We will verify the validity of our results using cross-validation. We will use Machine Learning classifiers such as Support Vector Classifiers, Decision Tree Classifiers, Random Forest Classifiers, Ensemble Methods such as Adaboost, Gradient Boost, etc. and use Multi-layer Perceptron.

## Feature Selection

While performing Principal Component Analysis and interpreting correlation heatmaps, we decided that we will use all 22 columns as our input. This is what our input and output sets look like:

```
inputs = ['family_history', 'Gender', 'obs_consequence', 'Country',
          'mental_health_consequence', 'phys_health_consequence',
          'coworkers', 'remote_work', 'care_options', 'self_employed',
          'wellness_program', 'seek_help', 'supervisor', 'tech_company',
          'phys_health_interview', 'Age', 'mental_health_interview',
          'benefits', 'mental_vs_physical', 'anonymity', 'leave',
          'work_interfere']
output = 'treatment'
```

```
X = df.loc[:, inputs].values
X.shape
```

```
(1259, 22)
```

```
y = df.loc[:, output].values
y.shape
```

```
(1259,)
```

We have 1259 records and 22 fields.

Let's start with Support Vector Classifiers.

## Feature Scaling

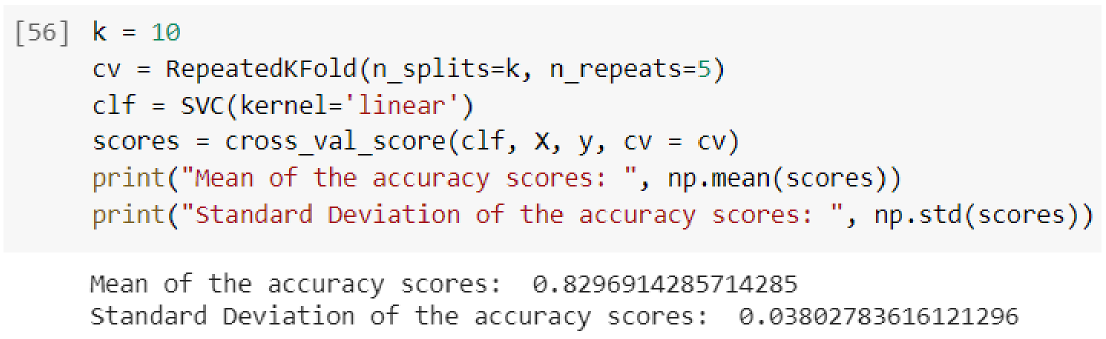
For feature scaling, we will use standard scalar. Standard scalar makes the mean of the distribution equal to 0. The values are scaled to a unit variance (+1 or -1). Also, unlike normalization, standardization does not have a boundary range. So, even if the data contains some outliers, they won't be affected by standardization. [4]

## 1. Support Vector Classifiers

SVC works well with small datasets. [5] It can be used for both – regression and classification tasks. This dataset had 1259 records and we were using only 22 columns. That's why we thought it's worth giving SVC a try!

**Iteration #1**

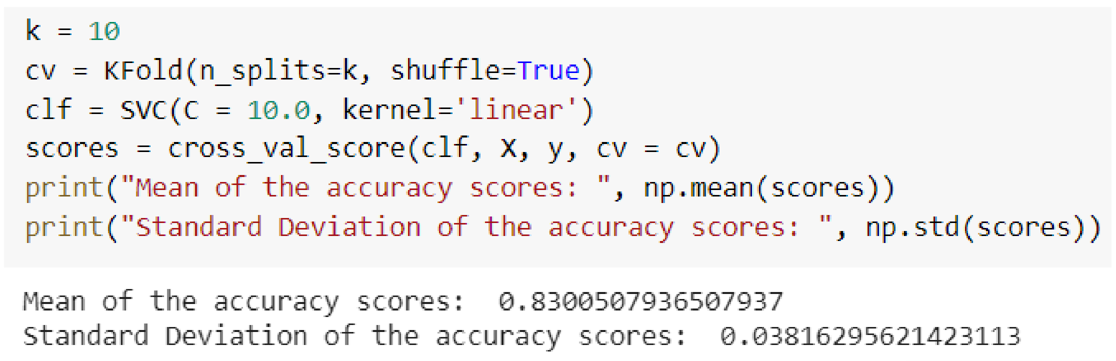
In the first iteration, we chose to use RepeatedKFold with 10 splits and 5 repeats. Repeated cross-validation is simply repeating cross-validation several times with the folds separated in a different way in each repetition. The model accuracy is computed after each cross-validation repetition. The scores from all iterations are averaged to find out the mean score. It is more robust than cross-validation. We selected the classifier as SVC as the kernel as linear. In this case, the model scored a mean accuracy of 82.96% and the standard deviation was 0.03.



```
[56] k = 10
     cv = RepeatedKFold(n_splits=k, n_repeats=5)
     clf = SVC(kernel='linear')
     scores = cross_val_score(clf, X, y, cv = cv)
     print("Mean of the accuracy scores: ", np.mean(scores))
     print("Standard Deviation of the accuracy scores: ", np.std(scores))

     Mean of the accuracy scores:  0.8296914285714285
     Standard Deviation of the accuracy scores:  0.03802783616121296
```

**Iteration #2**

In the second iteration, we simply chose KFold with 10 splits and shuffled records. If records are not shuffled, one might just get lucky and get a good result in one iteration even when the model is overfitting. This time, we tuned the hyperparameters of SVC by modifying C value and setting it equal to 10. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. [6] In this case, we got a slightly better result. The mean accuracy of the model was 83% but standard deviation continued to be the same i.e. 0.03.



```
k = 10
cv = KFold(n_splits=k, shuffle=True)
clf = SVC(C = 10.0, kernel='linear')
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))

Mean of the accuracy scores:  0.8300507936507937
Standard Deviation of the accuracy scores:  0.03816295621423113
```

**Iteration #3 [Best]**

In the third iteration, we chose KFold with 10 splits and shuffled records. However, this time we tried a lower value for C by setting it equal to 5. The mean accuracy here was 83.16% and the standard deviation also decreased to 0.02.

```
k = 10
cv = KFold(n_splits=k, shuffle=True)
clf = SVC(C = 5.0, kernel='linear')
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.831631746031746
Standard Deviation of the accuracy scores:  0.021691082005025765
```

## 2. Decision Tree Classifier

Like SVC, Decision trees can also be used as both regressors as well as classifiers. Decision trees are most commonly used for data mining for deriving a strategy to reach a particular goal. However, they are also widely used in Machine Learning. [3]

**Iteration #1**

In the first iteration, we chose LeaveOneOut() and trained our model on all data except one record. LeaveOneOut is basically KFold validation taken to its logical extreme with K=N-1 points in the dataset. This means that N separate times, the function is trained on all the data except for one point which is used as test data. [6] Because it is such a rigorous algorithm, generally, it produced more reliable results. However, it is a pretty expensive operation.

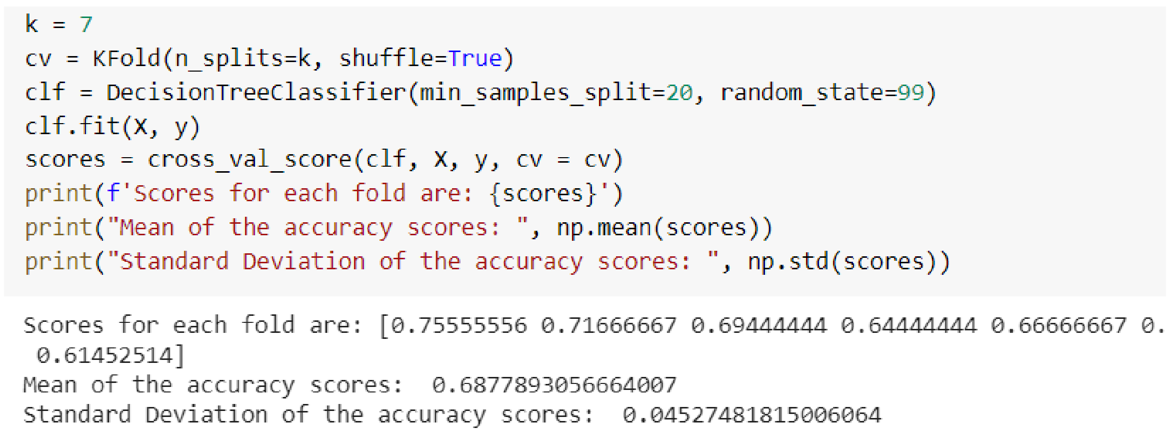
```
cv = LeaveOneOut()
clf = DecisionTreeClassifier()
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv=cv, n_jobs=-1)

print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.6640190627482129
Standard Deviation of the accuracy scores:  0.47233224223548176
```

With Decision Trees and LeaveOneOut, the mean accuracy of our model dropped down to 66.40% with a standard deviation of 0.47. LeaveOneOut didn't seem to perform very well with this dataset. Plus, it took a lot of time for computation.

**Iteration #2**

In the second iteration, we used KFold with 7 splits and shuffled records.



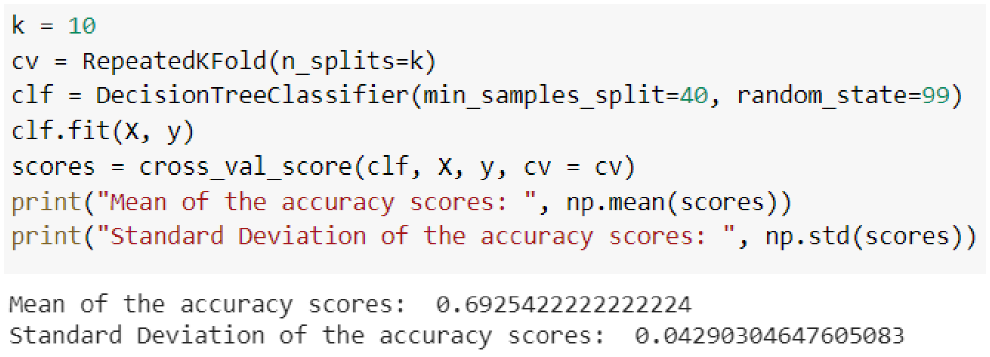
```
k = 7
cv = KFold(n_splits=k, shuffle=True)
clf = DecisionTreeClassifier(min_samples_split=20, random_state=99)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print(f'Scores for each fold are: {scores}')
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Scores for each fold are: [0.75555556 0.71666667 0.69444444 0.64444444 0.66666667 0.
 0.61452514]
Mean of the accuracy scores:  0.6877893056664007
Standard Deviation of the accuracy scores:  0.04527481815006064
```

Although the model accuracy wasn't very good either in this case, it was still better than the previous iteration. We managed to get a mean accuracy of 68.77% and standard deviation of 0.45.

**Iteration #3 [Best]**

In the third iteration, we used Repeated K-Fold with 10 splits. Here, we changed the hyperparameters of the DecisionTreeClassifier by increasing the min_sample_split from 20 to 40. This resulted in a slightly better accuracy of 0.69% with a standard deviation of 0.04.



```
k = 10
cv = RepeatedKFold(n_splits=k)
clf = DecisionTreeClassifier(min_samples_split=40, random_state=99)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.6925422222222224
Standard Deviation of the accuracy scores:  0.04290304647605083
```

Overall, Decision Tree Classifier produced some mediocre results on this dataset.

## 3. Random Forest

Random forest is an ensemble method that decides where to split based on a random selection of features. An ensemble method is a method that combines several base models to produce one optimal predictive model. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, thereby producing a more accurate predictor. [7]

For Random Forest, we conduct a number of iterations. The most important ones are described below:

**Iteration #1 [Best]**

In the first iteration, we used 13 KFold Splits with shuffled records. We used RandomForestClassifer with random_state = 99 and no other parameter. The mean model accuracy came out to be 81.88% with a standard deviation of 0.03.

```
k = 13
cv = KFold(n_splits=k, shuffle=True)
clf = RandomForestClassifier(random_state=99)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.8188606925720328
Standard Deviation of the accuracy scores:  0.03934989517140384
```

**Iteration #2**

In this iteration, we used grid search to determine which parameters would work best for our RandomForestClassifier. It took more than 30 minutes to run a single Grid Search on Random Forest, so we had to limit our choices.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
clf = RandomForestClassifier(random_state=99)
param_grid = {
    'max_depth' : [10, 12],
    'n_estimators': [300, 500],
    'criterion' :['gini','entropy']
}
clf_gridSearch = GridSearchCV(estimator=clf, param_grid=param_grid, cv= cv)
clf_gridSearch.fit(X_train, y_train)
```

```
clf_gridSearch.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 500}
```

The grid search chose entropy over gini, 500 estimators, and max_depth = 10. We chose these parameters tested our model RepeatedKFold with 13 splits.

```
k = 13
cv = RepeatedKFold(n_splits=k)
clf = RandomForestClassifier(n_estimators = 500, criterion='entropy',
                             random_state = 99, max_depth=10)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.8097839016653449
Standard Deviation of the accuracy scores:  0.035004770808031 05
```

Turns out, with these parameters, the accuracy was even less than Iteration 1. (Mean accuracy = 80.97%, standard deviation = 0.03).

**Iteration #3**

For this iteration, we chose the same hyperparameters as the previous one, except for n_estimation which were set to 600 this time. Also, rather than selecting repeatedKFold, we chose KFold with 13 splits and shuffled records. This time, the accuracy was 81.01% and standard deviation was 0.03.

```
k = 13
cv = KFold(n_splits=k, shuffle=True)
clf = RandomForestClassifier(n_estimators = 600, criterion='entropy',
                             random_state = 99, max_depth=10)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.8101787602431934
Standard Deviation of the accuracy scores:  0.03554975102361522
```

## 4. Adaboost

Adaboost is another ensemble method, which, in some cases, is less susceptible to overfitting than other algorithms. The individual learners can be weak, but as long as the performance of each weak learner is better than random guessing, the final model can converge to a **strong learner** (a learner not influenced by outliers and with a great generalization power, in order to have strong performances on uknown data). [8]

**Iteration #1**

We started by using AdaboostClassifier simply without including any hyperparameters and used RepeatedKFold with 13 splits. The accuracy in this case was 78%, with a standard deviation of 0.04.

```
k = 13
cv = RepeatedKFold(n_splits=k)
clf = AdaBoostClassifier()
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.7803264604810997
Standard Deviation of the accuracy scores:  0.041555174723170 26
```

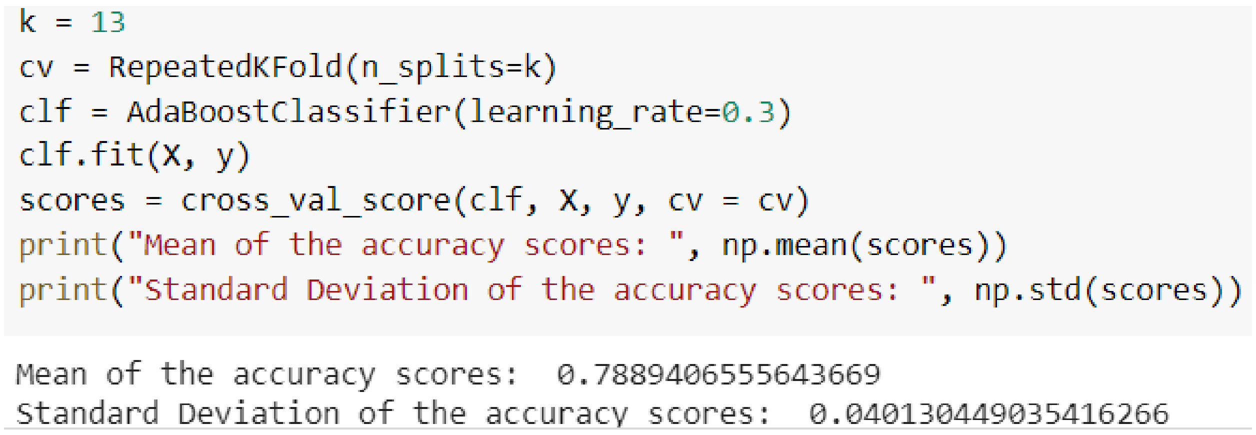
**Iteration #2 [Best]**
In the second iteration, we used grid search and tried out different learning rates. After testing out the best learning rate, we found out that there was just a slight increase in accuracy : 78.89% with standard deviation = 0.04.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
clf = AdaBoostClassifier()
param_grid = {
    'learning_rate': [0.1, 0.5, 0.2, 0.3]
}
clf_gridSearch = GridSearchCV(estimator=clf, param_grid=param_grid, cv= cv)
clf_gridSearch.fit(X_train, y_train)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0, n_estimators=50,
                                          random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 0.5, 0.2, 0.3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
clf_gridSearch.best_params_
```

```
{'learning_rate': 0.3}
```



```
k = 13
cv = RepeatedKFold(n_splits=k)
clf = AdaBoostClassifier(learning_rate=0.3)
clf.fit(X, y)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.7889406555643669
Standard Deviation of the accuracy scores:  0.040130449035416266
```

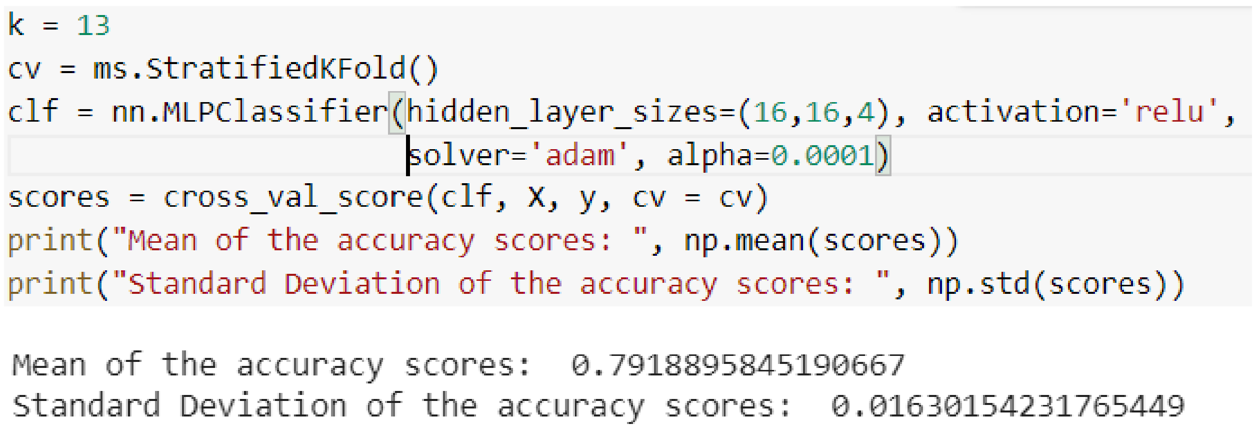
## 5. Multi-Layer Perceptron

MLPs can be used for tabular datasets and regression and classification tasks. [9] They are very flexible, and they are generally used to learn some mapping from inputs to outputs. For our model, we are performing a classification task on a tabular dataset. That's why we chose MLP.

For MLP, we ran several iterations. The most important ones are listed below:

**Iteration #1**

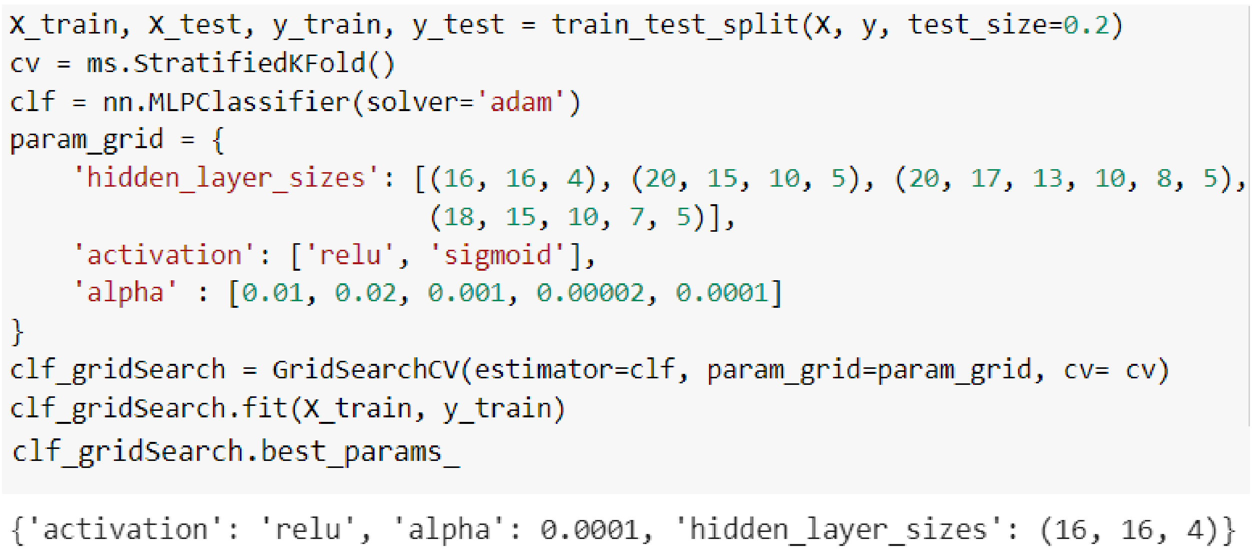
Iteration 1 was based on trial and error. We used stratified k-fold and got the accuracy of 79.18% with a standard deviation of 0.016.



```
k = 13
cv = ms.StratifiedKFold()
clf = nn.MLPClassifier(hidden_layer_sizes=(16,16,4), activation='relu',
                       solver='adam', alpha=0.0001)
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

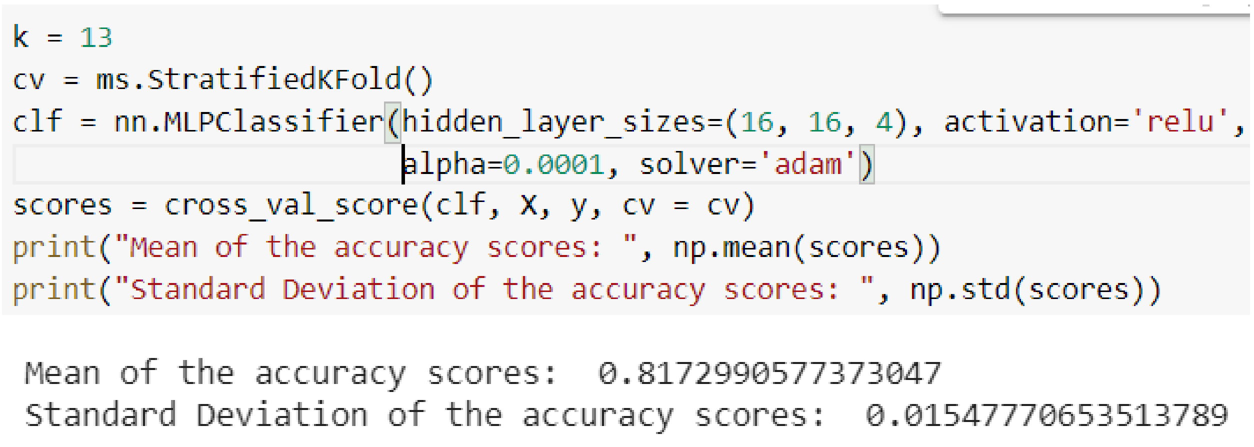
```
Mean of the accuracy scores:  0.7918895845190667
Standard Deviation of the accuracy scores:  0.01630154231765449
```

**Iteration #2 [Best]**
We used a Grid Search to test different hyperparameter values, such as different values for hidden layers, activation functions such as sigmoid and reLU, and different alpha values.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
cv = ms.StratifiedKFold()
clf = nn.MLPClassifier(solver='adam')
param_grid = {
    'hidden_layer_sizes': [(16, 16, 4), (20, 15, 10, 5), (20, 17, 13, 10, 8, 5),
                           (18, 15, 10, 7, 5)],
    'activation': ['relu', 'sigmoid'],
    'alpha' : [0.01, 0.02, 0.001, 0.00002, 0.0001]
}
clf_gridSearch = GridSearchCV(estimator=clf, param_grid=param_grid, cv= cv)
clf_gridSearch.fit(X_train, y_train)
clf_gridSearch.best_params_
```

```
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (16, 16, 4)}
```



```
k = 13
cv = ms.StratifiedKFold()
clf = nn.MLPClassifier(hidden_layer_sizes=(16, 16, 4), activation='relu',
                       alpha=0.0001, solver='adam')
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.8172990577373047
Standard Deviation of the accuracy scores:  0.01547770653513789
```

The mean accuracy increased from 79% to 81.7%. Standard deviation decreased from 0.016 to 0.015.

6. Repeated KFold vs Stratified KFold

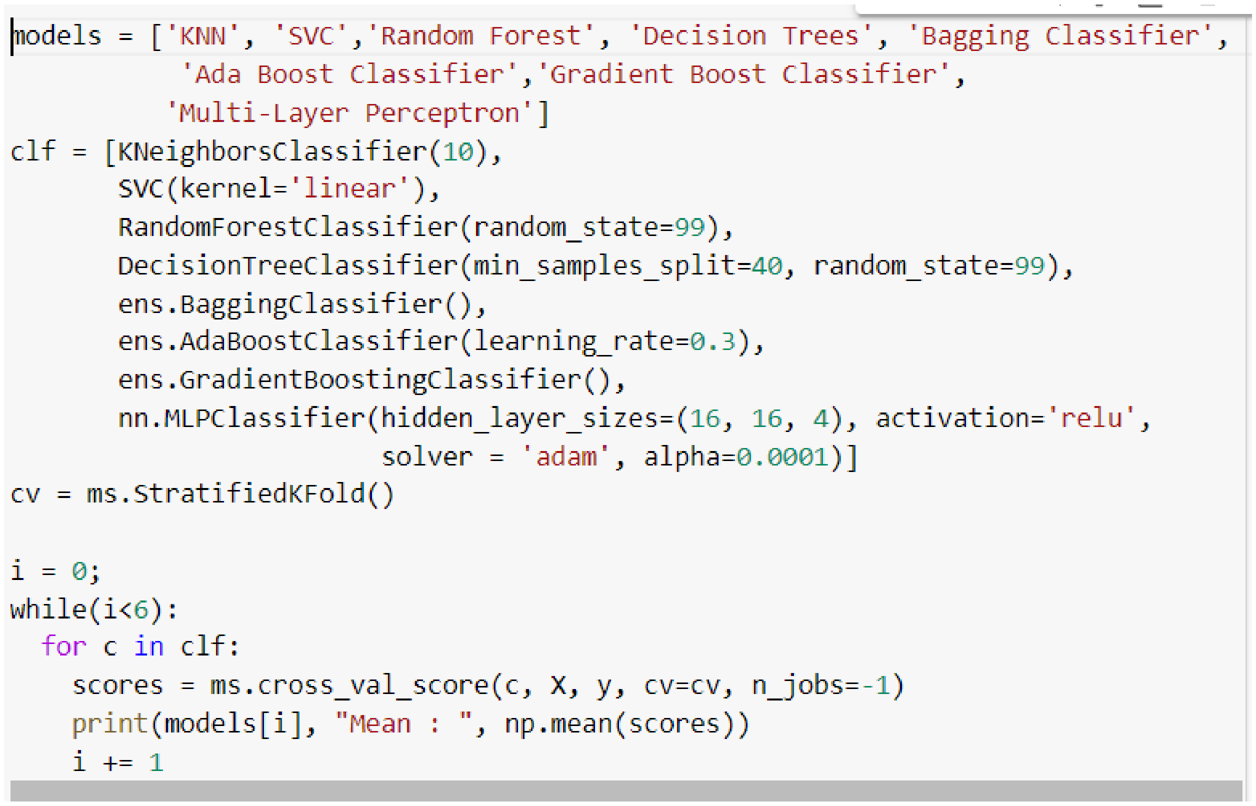
Here, we compared the mean accuracy is all the above models, plus KNN, Bagging Classifier, and GradientBoost, and ultimately found that Stratified KFold had better overall results in comparison to RepeatedKFold.

## Repeated K Fold

```
models = ['KNN', 'SVC','Random Forest', 'Decision Trees', 'Bagging Classifier',
           'Ada Boost Classifier','Gradient Boost Classifier',
          'Multi-Layer Perceptron']
clf = [KNeighborsClassifier(10),
       SVC(kernel='linear'),
       RandomForestClassifier(random_state=99),
       DecisionTreeClassifier(min_samples_split=40, random_state=99),
       ens.BaggingClassifier(),
       ens.AdaBoostClassifier(learning_rate=0.3),
       ens.GradientBoostingClassifier(),
       nn.MLPClassifier(hidden_layer_sizes=(16, 16, 4), activation='relu',
                        solver = 'adam', alpha=0.0001)]

k=13
cv = RepeatedKFold(n_splits=k)

i = 0;
while(i<6):
  for c in clf:
    scores = ms.cross_val_score(c, X, y, cv=cv, n_jobs=-1)
    print(models[i], "Mean : ", np.mean(scores))
    i += 1
```

```
KNN Mean :  0.777447627544277
SVC Mean :  0.8281076526566217
Random Forest Mean :  0.8061896973301611
Decision Trees Mean :  0.6868003238170765
Bagging Classifier Mean :  0.7617144131641553
Ada Boost Classifier Mean :  0.7937376090404441
Gradient Boost Classifier Mean :  0.7983189598202484
Multi-Layer Perceptron Mean :  0.7903870935765265
```

Stratified K Fold



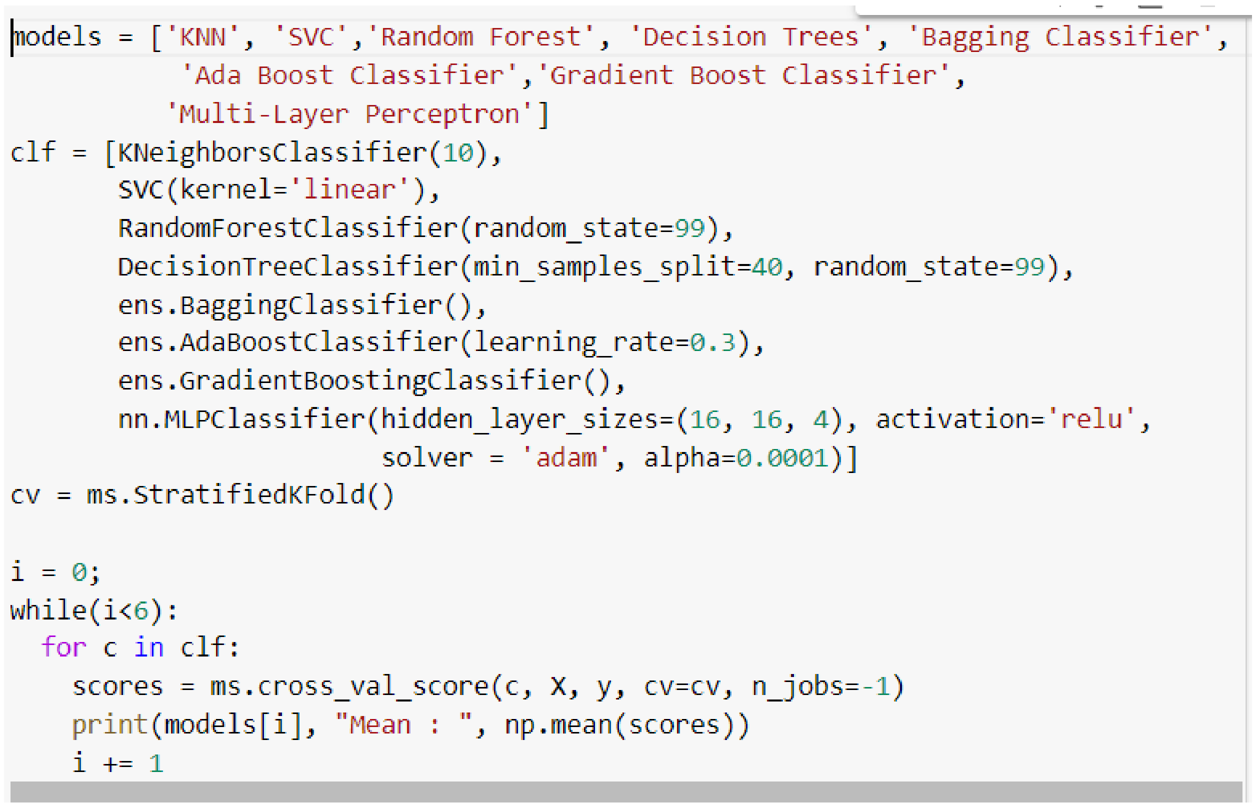
```
models = ['KNN', 'SVC','Random Forest', 'Decision Trees', 'Bagging Classifier',
           'Ada Boost Classifier','Gradient Boost Classifier',
          'Multi-Layer Perceptron']
clf = [KNeighborsClassifier(10),
       SVC(kernel='linear'),
       RandomForestClassifier(random_state=99),
       DecisionTreeClassifier(min_samples_split=40, random_state=99),
       ens.BaggingClassifier(),
       ens.AdaBoostClassifier(learning_rate=0.3),
       ens.GradientBoostingClassifier(),
       nn.MLPClassifier(hidden_layer_sizes=(16, 16, 4), activation='relu',
                        solver = 'adam', alpha=0.0001)]
cv = ms.StratifiedKFold()

i = 0;
while(i<6):
  for c in clf:
    scores = ms.cross_val_score(c, X, y, cv=cv, n_jobs=-1)
    print(models[i], "Mean : ", np.mean(scores))
    i += 1
```

```
KNN Mean :  0.7751881363435148
SVC Mean :  0.8284038449377095
Random Forest Mean :  0.798222981091507
Decision Trees Mean :  0.6775121735281098
Bagging Classifier Mean :  0.760096123442737
Ada Boost Classifier Mean :  0.7934452665528363
Gradient Boost Classifier Mean :  0.7902833111996459
Multi-Layer Perceptron Mean :  0.7982324669575667
```

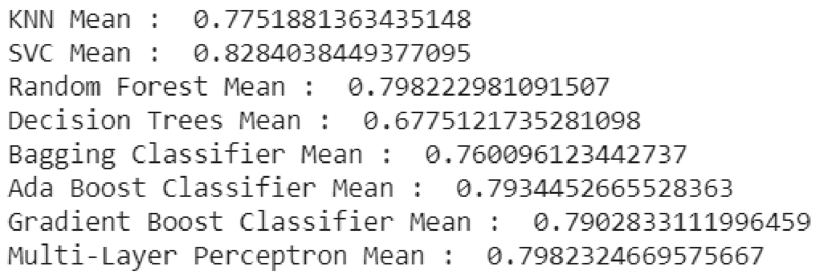
An in-depth analysis of the above screenshots is presented in the results section. Stratified KFold model is ultimately chosen.

## 5. Results and Analysis

In this step, we chose KNN, Bagging Trees, and Gradient Boost in addition to the models described above. For the classifiers that were included in the experiments section, we chose the best hyperparameters that produced the greatest accuracy in each.



```
models = ['KNN', 'SVC','Random Forest', 'Decision Trees', 'Bagging Classifier',
           'Ada Boost Classifier','Gradient Boost Classifier',
          'Multi-Layer Perceptron']
clf = [KNeighborsClassifier(10),
       SVC(kernel='linear'),
       RandomForestClassifier(random_state=99),
       DecisionTreeClassifier(min_samples_split=40, random_state=99),
       ens.BaggingClassifier(),
       ens.AdaBoostClassifier(learning_rate=0.3),
       ens.GradientBoostingClassifier(),
       nn.MLPClassifier(hidden_layer_sizes=(16, 16, 4), activation='relu',
                        solver = 'adam', alpha=0.0001)]
cv = ms.StratifiedKFold()

i = 0;
while(i<6):
  for c in clf:
    scores = ms.cross_val_score(c, X, y, cv=cv, n_jobs=-1)
    print(models[i], "Mean : ", np.mean(scores))
    i += 1
```



```
KNN Mean :  0.7751881363435148
SVC Mean :  0.8284038449377095
Random Forest Mean :  0.798222981091507
Decision Trees Mean :  0.6775121735281098
Bagging Classifier Mean :  0.760096123442737
Ada Boost Classifier Mean :  0.7934452665528363
Gradient Boost Classifier Mean :  0.7902833111996459
Multi-Layer Perceptron Mean :  0.7982324669575667
```

As seen before, Stratified K Fold performed better than Repeated K Fold cross validation. This is because the former ensures that the number of instances per class is equal and that they have a uniform distribution. It returns stratified folds.

Comparing the performance of individual classifiers, we found that SVC had the best performance, with an accuracy of 80.28%. This was followed by Multi-Layer Perceptron and Random Forest with accuracy equal to 79.8%. Ensemble methods including Adaboost and Gradient boost also performed at par with an accuracy of about 79.3 and 79% respectively. KNN followed, with an accuracy of 77.5%. Decision trees demonstrated the worst performance with an accuracy of 67.75%.

SVC works comparatively well when two or more classes have a clear margin of separation. Additionally, it works better on small sized datasets. This can be one of the reasons why SVC performed best on our model. This also implies that our data could have classes that have a clear margin of separation.

MLP uses the concept of hidden layers. Here, the number of perceptrons in layer 1 is multiplied by the number of perceptrons in layer 2, which is multiplied by layer 3, so on and so forth. This can lead to a very high number of dimensions and computations which can be one of the reasons why MLP did not score the highest.

Ensemble methods combine several base models to produce one optimal model. Decision tree is a base model whereas Random Forests, Adaboost, and Gradient Boost are all built on top of a base model. This is why these ensemble methods outperformed Decision Trees.

KNN also works well with small number of input variables. As the number of input variables increase, the performance of KNN starts to decrease. We didn't have as many inputs in our model – just 22! Being a simple method, KNN couldn't manage to outperform Ensemble Methods or MLP, but it did leave Decision trees behind with a whooping difference of 10%, which is pretty good.

Since SVC had the best performance, we will choose SVC as the final algorithm.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

clf = SVC(C = 5.0, kernel='linear')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Confusion matrix for SVC:

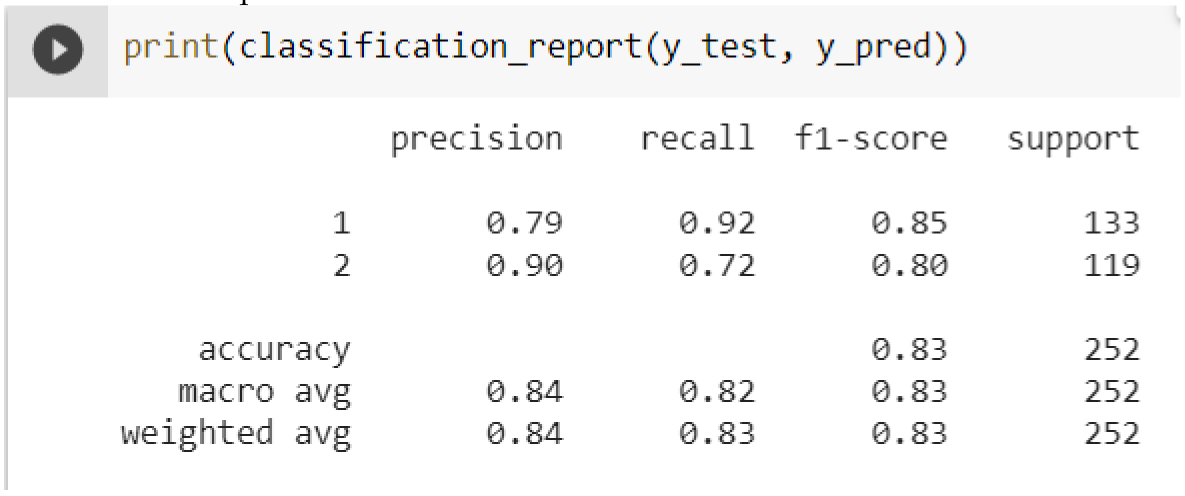
```
print(confusion_matrix(y_test, y_pred))
```

```
[[123  10]
 [ 33  86]]
```

True Positives = 123
False Positives = 10

False Negatives = 33
True Negatives = 86

Classification Report

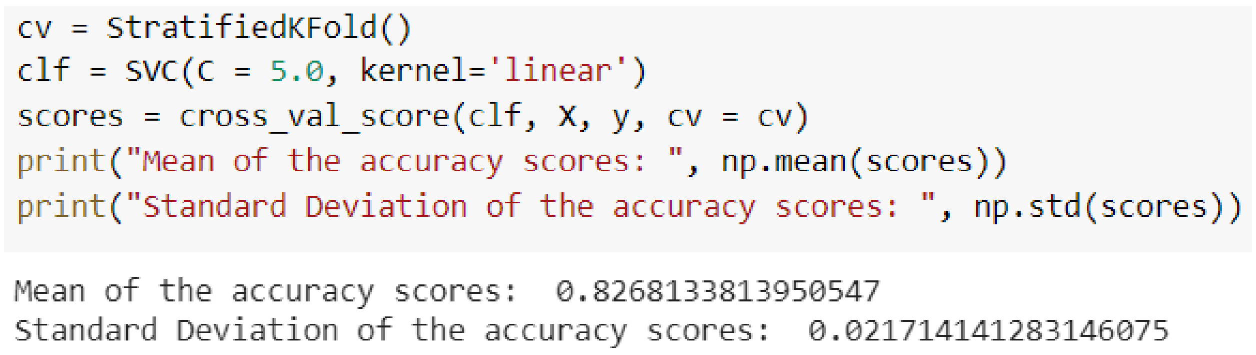


```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.79      0.92      0.85       133
           2       0.90      0.72      0.80       119

    accuracy                           0.83       252
   macro avg       0.84      0.82      0.83       252
weighted avg       0.84      0.83      0.83       252
```

Model accuracy is 83%. The precision is 90%, recall is 72%, and f1 score is 80%.

Stratified K Fold Cross Validation



```
cv = StratifiedKFold()
clf = SVC(C = 5.0, kernel='linear')
scores = cross_val_score(clf, X, y, cv = cv)
print("Mean of the accuracy scores: ", np.mean(scores))
print("Standard Deviation of the accuracy scores: ", np.std(scores))
```

```
Mean of the accuracy scores:  0.8268133813950547
Standard Deviation of the accuracy scores:  0.021714141283146075
```

In the last step, we verified our results using Stratified K Fold Validation. The mean accuracy is 82.68%, which is very close to what we got in the last step.

For practical applications, this Machine Learning model can be used to predict if a person will undergo treatment or not, based on their behavioral aspects. For instance, whether their mental illness is interfering with their work, or if they have a family history of mental illnesses, or if they are self-employed...many such things can be used to predict if they are (will) undergoing any kind of treatment or not. Mental health is very important, and this is an issue that cannot be swept under the rug. It is crucial that we try our best to help those who need it the most. After all, today's youth is tomorrow's future. And today's youth is affected by this. So, for a better, healthier future, we need this.

## References

[1] "Mental Health in Tech Survey," Open Sourcing Mental Illness Ltd, 2014. [Online]. Available: https://www.kaggle.com/osmi/mental-health-in-tech-survey.

[2] Shabuki, "StackExchange," 14 Feb 2019. [Online]. Available: https://stats.stackexchange.com/questions/392517/how-can-one-interpret-a-heat-map-plot#:~:text=Correlation%20ranges%20from%20%2D1%20to,the%20stronger%20this%20relationship%20is..

[3] P. Gupta, "Decision Trees in Machine Learning," 17 May 2017. [Online]. Available: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052.

[4] A. Uoadhyay, "StandardScaler and Normalization with code and graph," 13 June 2020. [Online]. Available: https://medium.com/analytics-vidhya/standardscaler-and-normalization-with-code-and-graph-ba220025c054.

[5] A. Yadav, "SUPPORT VECTOR MACHINES(SVM)," 20 Oct 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589.

[6] M. Shivers, "What is the influence of C in SVMs with linear kernel?," [Online]. Available: https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel.

[7] "Cross Validation," [Online]. Available: https://www.cs.cmu.edu/~schneide/tut5/node42.html.

[8] E. Lutins, "Ensemble Methods in Machine Learning: What are They and Why Use Them?," 1 Aug 2017. [Online]. Available: https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f.

[9] "How Adaboost works," [Online]. Available: https://iq.opengenus.org/adaboost/.

[10] J. Brownlee, "When to Use MLP, CNN, and RNN Neural Networks," 23 July 2018. [Online]. Available: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/.