Stanford University

STATS 202: Data Mining and Analysis

Study of the Efficacy of an Experimental Drug on Schizophrenia Treatment

Authors' Names:
Arie N. Arya
Lam P. Bach

Authors' Email:
ana4718@stanford.edu
riddle95@stanford.edu

Group Instructor: *Linh Tran*

Final Project Report

August 13, 2020

Contents

1 Abstract **3**

2 Treatment effect **4**
2.1 Data Pre-processing . . . 4
2.2 Statistical Testing . . . 4
2.3 Comparison with Existing Medical Treatment . . . 5

3 Patient segmentation **6**
3.1 Naive Clustering Implementation and its Issues . . . 6
3.2 PCA and K-means Clustering . . . 6
3.2.1 Data Pre-processing . . . 6
3.2.2 Observing Correlation . . . 7
3.2.3 Performing PCA . . . 7
3.2.4 K-means Clustering . . . 7
3.2.5 Analyzing the Clusters and Segmenting the Population . . . 7

4 Forecasting **9**
4.1 Data Pre-processing . . . 9
4.2 Creating and Defining RNN Model . . . 10
4.3 Training the RNN Model . . . 11
4.4 RNN Model Performance . . . 11

5 Binary classification **12**
5.1 Data Pre-processing . . . 12
5.2 Logistic Regression . . . 12
5.3 Random Forests . . . 13
5.4 Gradient Boosting . . . 13

6 Appendix **14**
6.1 Figures and Diagrams . . . 14
6.2 Part 1 Code: Treatment Effect . . . 22
6.3 Part 2 Code: Patient Segmentation . . . 24
6.4 Part 3 Code: Forecasting . . . 28
6.5 Part 4 Code: Binary Classification . . . 32

References **34**

1 Abstract

The objective of this report is to analyze the efficacy of an anonymized drug treatment on treating patients with schizophrenia, and to predict the progress and outcome of similar patients throughout the course of the treatment program. This report will use both R and Python to conduct statistical analysis and modelling.

The patient studies are conducted by randomly splitting patients into "Treatment" and "Control" groups, where treatment groups are given the experimental drug for treating the disorder, whilst the control group is given the standard and accepted medical treatment in order to compare its effectiveness.

The measure used to determine the severity of patients with schizophrenia is the PANSS (Positive and Negative Syndrome Scale). This is a numeric scale which measures 30 related symptoms in a patient, each with a severity score of between 1-7; 7 being the most extreme [3]. The 30 symptom scores measured are shown below.

Positive scale

7 Items, (minimum score = 7, maximum score = 49)

- Delusions
- Conceptual disorganization
- Hallucinations
- Excitement
- Grandiosity
- Suspiciousness/persecution
- Hostility

Negative scale

7 Items, (minimum score = 7, maximum score = 49)

- Blunted affect
- Emotional withdrawal
- Poor rapport
- Passive/apathetic social withdrawal
- Difficulty in abstract thinking
- Lack of spontaneity and flow of conversation
- Stereotyped thinking

General Psychopathology scale

16 Items, (minimum score = 16, maximum score = 112)

- Somatic concern
- Anxiety
- Guilt feelings
- Tension
- Mannerisms and posturing
- Depression
- Motor retardation
- Uncooperativeness
- Unusual thought content
- Disorientation
- Poor attention
- Lack of judgment and insight
- Disturbance of volition
- Poor impulse control
- Preoccupation
- Active social avoidance

PANSS Total score minimum = 30, maximum = 210

In conjunction with this analyzing the drug's efficacy, as the PANSS measure conducted are oftentimes contradictory and inconsistent (i.e. one measure may indicate psychosis where another indicates normality), some readings will need to be flagged for re-evaluation. This report will discuss methods to determine whether or not a particular PANSS interview session should be investigated for its reliability.

This report will be broken up into four main sections:

- Treatment Effect
- Patient Segmentation
- Forecasting
- Binary Classification

Treatment Effect analyzes the overall effectiveness and efficacy of the anonymized drug in treating patients with schizophrenia. **Patient Segmentation** attempts to find and categorize similar patients with the disorder, and observe the reactions of these different groups to the drug. **Forecasting** looks at different models to make projections of the overall PANSS score of patients on their 18th week of treatment. **Binary Classification** analyzes patterns in erroneous PANSS assessment sessions in an attempt to reliably automate the process of flagging inconsistent results for re-evaluation.

2 Treatment effect

This section will use R to analyze the overall performance of the anonymized drug treatment on treating patients with schizophrenia by observing their PANSS Score over the course of their treatment program. Here, the patients will be segmented into their respective "Control" and "Treatment" group to compare the effectiveness of the drug to the accepted standard medication. The code for this section can be observed in section 6.2 of the Appendix.

In order to reduce the effect of data snooping / dredging, we have decided to conduct a null hypothesis test a-priority to analyzing or inspecting the data-set. This test will determine if there exists a statistical relationship of the anonymized drug on treating schizophrenia over a period of time.

In particular, the null hypothesis test will be carried out with the predictor "$VisitDay$" on patients in the **treatment group**, and a linear model will be fitted. The general equation for this linear model is:

$$PANSS\_Total = \beta_0 + \beta_1 VisitDay \tag{1}$$

The corresponding null hypothesis is then given as:

$$H_0 : \beta_1 = 0 \tag{2}$$

With the alternative hypothesis being:

$$H_a : \beta_1 \neq 0 \tag{3}$$

Should the drug be ineffective in treating schizophrenia, the p-value of this predictor should remain above the typical threshold 0.05, indicating the PANSS score remains relatively constant in the course of the program (i.e. treatment has no effect). Moreover, should the null hypothesis be rejected, further analysis will be performed to see whether or not the PANSS score worsens throughout the program, which will determine the overall treatment effect of the drug.

2.1 Data Pre-processing

In order to conduct the statistical test, the data from all studies A to D will be merged into a combined study. This is necessary to reduce the overall bias of the data-set as it becomes more representative of the whole population, including the study of patients from different countries. This is achieved by the code segment below:

```
df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
df = rbind(df_A, df_B, df_C, df_D)
```

Next, as we would like to conduct statistical inference on patients in the treatment group, the control group patients must be neglected.

```
df = filter(df, TxGroup == "Treatment")
```

Selecting only the predictor "$VisitDay$" and the response "$PANSS\_Total$", an initial scatter-plot is created to observe the general relationship of the data. This is displayed in Figure 1.

By observation of the plot, it is clear that there is a consistent and general drop in the PANSS score of the treatment group patients throughout the treatment program.

2.2 Statistical Testing

A linear model shown in equation 1 can be fitted to the processed data to conduct the null hypothesis test defined in the beginning of this section. This is achieved by the code below:

```
visitDay = df$VisitDay
panssTotal = df$PANSS_Total
linear_fit = lm(panssTotal~visitDay)
print(summary(linear_fit))
```

The summary obtained from the fit is shown below:

```
Call:
lm(formula = panssTotal ~ visitDay)

Residuals:
    Min      1Q  Median      3Q     Max
-47.280  -9.730  -1.047   9.665  59.965

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 82.436125   0.210871   390.9   <2e-16 ***
visitDay    -0.119758   0.001565   -76.5   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.27 on 10291 degrees of freedom
Multiple R-squared:  0.3625,    Adjusted R-squared:  0.3625
F-statistic:  5853 on 1 and 10291 DF,  p-value: < 2.2e-16
```

By observation of the p-value of the "$VisitDay$" variable, as it is considerably lower than the threshold 0.05, we can safely **reject null hypothesis**. This indicates that the PANSS score of the patients in the treatment group does vary throughout the treatment program.

Further, the coefficient β_1 of the "$VisitDay$" variable is obtained as -0.119758. This indicates that on average, the PANSS score of patients in the treatment group reduces by 0.199758 every day, which is a desirable treatment effect.

The plot of the linear model is shown in Figure 2. Additionally, plots of all symptoms of treatment patients throughout the program is shown in Figure 3. This shows a general reduction and improvement in the symptom score (which ranges from 1-7) of the patient.

In general, it is observed that the symptom scores reduce (improves) throughout the course of the program, indicating a positive effect of the treatment drug in treating the patients.

2.3 Comparison with Existing Medical Treatment

Although the findings thus far suggests that the drug is effective in treating the patients, it is also important to make comparison with existing and accepted medication to compare their performance. Using the same method as seen previously, we can select patients under the **control group**, and construct a linear model. The result is shown in Figure 4.
The corresponding intercept and gradient obtained in this linear model is:

```
(Intercept)    visitDay
 83.9494583  -0.1269148
```

As can be seen, the accepted standard medication for treating schizophrenia on average sees an improvement in patients' PANSS score of 0.1269 per day, compared to 0.1198 for the anonymized drug. In conclusion, although the anonymized drug has an overall positive effect on treating patients with schizophrenia, if the interpretation of the term *effectiveness* refers to a possible replacement of standard medication with the experimental drug, then evidence from this analysis does not strongly support this case.

3 Patient segmentation

This section explores techniques and methods to segment the population of schizophrenia patients into subgroups in order to study how the proposed treatment affects them. In particular, the clustering method used in this section is the k-means clustering algorithm, where the optimum k value is obtained by the elbow point method. Further, the clustering will only be performed on the baseline patient data, i.e. their corresponding data on the first day (day 0) of visit, as this captures their condition prior to any treatment effect. The code for this section can be observed in section 6.3 of the Appendix.

3.1 Naive Clustering Implementation and its Issues

A very naive implementation of a clustering method is to look at all 30 PANSS symptoms and applying k-means clustering directly. However, the underlying issue with such a method is its high dimension. As the dimension of our problem grows, the data becomes more and more sparsely spread in space, and as a result k-means becomes less effective at distinguishing and thus grouping similar points (especially through the use of Euclidean distance). This is the so-called the *curse of dimensional* problem.

Alternatively then, a possible clustering method would be to look at the three categories of the PANSS score separately; that is, the **Positive, Negative, and the General Psychopathology Scale**. First, after totalling all of the columns corresponding to each of the three symptom categories, a general 3D scatter-plot is obtained, shown in Figure 5.Afterwards, the elbow method is performed, and an optimum k value of 5 is obtained as the elbow point (though in this case, the elbow point itself is rather ambiguous). This is shown in Figure 6.Performing k-means at a $k = 5$ gives us 5 clusters, and is illustrated in Figure 7 and Figure 8.

However, the key drawback in this method is that it assumes that the symptoms in each category are highly correlated, and can therefore be generalized as their joint sum. This, as will be explained in the next section, is in fact not a sensible assumption. By combining all symptoms into their corresponding categories, information about those individual symptoms are lost, and therefore meaningful clusters that may have arisen between specific symptoms from the different categories are also not possible. For this reason, the approach that is most reliable is to first perform PCA (Principal Component Analysis) before performing k-means clustering.

3.2 PCA and K-means Clustering

The general idea behind PCA is to reduce the dimensional of our data-set by combining highly correlated features and projecting the result into a lower dimension. This is because removing highly correlated features tend to sustain the information in our system whilst simultaneously reducing its dimensions, and therefore our k-means algorithm will suffer less from the curse of dimensional problem.

3.2.1 Data Pre-processing

Firstly, the data from all studies A-E are combined so as to reduce the overall bias of our analysis.

```
df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
df_E = LoadData(paste(DATA_PATH,"/Study_E.csv", sep=""))
df = rbind(df_A, df_B, df_C, df_D, df_E)
```

Next, scale the dataframe to 0 mean and unit variance and select only the rows corresponding to VisitDay 0 as well as the columns which corresponds to the 30 symptom scores,

```
# Select rows with day of visit 0
df = filter(df, VisitDay == 0)
# Select columns with symptom scores
df = subset(df, select = -c(Study, Country, SiteID, RaterID, AssessmentID, TxGroup,
     LeadStatus, VisitDay, PANSS_Total))
# Scale the dataframe
df = data.frame(scale(df))
```

3.2.2 Observing Correlation

After pre-processing the data, we can immediately observe the correlation between each of the 30 symptom scores. The resulting illustration of the correlation matrix is shown in Figure 9.

As can be seen, some of the symptoms are very highly correlated to one another (e.g. N1 and N4), and thus jointly contributes little additional information to our overall data-set. The purpose of PCA then is to obtain the highly uncorrelated principal components out of our features, which can then be used to perform clustering.

3.2.3 Performing PCA

Applying PCA to our pre-processed data-set using the *prcomp* function from the *stats* package in R, we can observe each principal components and the proportion of the total variance of our data-set that they represent. This is shown in Figure 10.

For this analysis, we have chosen the principal components which represents slightly above 60% of the total variance of our original data-set. This is achieved by the code below:

```
# Choose the number of PC that reaches 60% of total variance
pc <- prcomp(df)
var_list = summary(pc)$importance[2,]
pc_length = 0
cur_var_proportion = 0
for(i in 1:length(var_list)){
    cur_var_proportion = cur_var_proportion + var_list[i]
    if(cur_var_proportion >= 0.60){
      pc_length = i
      break
    }
}
```

From our data-set, the number of principal components which represent just over 60% of the total variance of the original data-set is 9. The plot of the correlation matrix between all 9 of these principal components is shown in Figure 11.

By observing the above correlation matrix, we can see that there is very little linear correlation between each principal components, meaning it retains a high degree of information whilst minimizing its dimensional.

3.2.4 K-means Clustering

Next, after applying PCA, we can apply k-means clustering. First, an elbow method is used to determine an optimum k-value. The illustration of this method can be observed in Figure 12. Though the elbow point again is quite ambiguous, a k value of 5 is quite appropriate as the gradient drops off marginally.
Finally, applying k-means clustering with a k value of 5, we obtain 5 overall clusters as illustrated in Figure 13.

3.2.5 Analyzing the Clusters and Segmenting the Population

Note, PCA alone is useful for allowing us to capture meaningful clusters, but the value of its centroid or its points is insubstantial since our original feature space has been transformed. The clusters obtained from this method uniquely assigns each *PatientID* to a specific cluster. Using these Patient ID's, we can observe the 30 symptom scores of each patient within each cluster. The mean value of each of the 30 symptom scores in each cluster is summarized in Figure 14. Note that the data was originally scaled to 0 mean and unit variance. Hence, a mean symptom score of around 0.8 is considered high, a mean symptom score of 0 is considered moderate, and a mean symptom score of -0.8 is considered low.
From this result, we can extract interesting properties in each cluster.

- **Cluster 1:** fairly high P values, fairly low N values, moderate G values.
- **Cluster 2:** relatively high P values, moderate N values, moderate G values.
- **Cluster 3:** fairly low P values, fairly low N values, fairly low G values.
- **Cluster 4:** moderate P values, very high N values, high G values.

- **Cluster 5:** very low P values, moderate N values, fairly low G values.

Through these generalizations of the clusters, we can make a general segmentation of the population:

- **Cluster 1 Patients:** suffer from a general distortion of normal functions, little diminution of normal functions, and a moderate degree of general psychosis
- **Cluster 2 Patients :** suffer from a clear distortion of normal functions (i.e. hallucinations and delusions), fair loss of normal functions, and a moderate degree of general psychosis
- **Cluster 3 Patients:** General lack of distortion and diminution of normal functions, and general signs of normality
- **Cluster 4 Patients:** slight distortion of normal functions, but a clear loss of normal functions and strong evidence of general psychosis
- **Cluster 5 Patients:** Very little signs of general psychosis or distortion of normal functions, but some degree of diminution in normal functions.

Out of the five clusters, cluster 2 and cluster 4 are generally patients who suffer severely from schizophrenia, whereas cluster 3 and cluster 5 have patients whose severity is only slight to moderate.

Of course, more meaningful interpretations of the clusters can be made by individually inspecting each of the 30 symptom scores and how they may relate between patients in real life. However, this would likely require a separate study on its own.

4 Forecasting

Kaggle username: Arie Arya, Public Leaderboard Rank: 12

This section concerns the forecasting / projection of the 18-th week PANSS score of patients in study E. For this particular problem, we have decided to use RNN (Recurrent Neural Network) with Python (using the Keras library) in order to predict the 18-th week PANSS score of different patients. RNN is a class of neural networks typically used to predict sequential or time series data, e.g. predicting stock prices in the next 24 hours, through the use of the LSTM (Long Short-Term Memory) architecture [2]. As with previous sections, modelling is done on the combined study A-D to reduce the overall bias of the problem.

4.1 Data Pre-processing

Before creating and training the RNN model, the raw data must be pre-processed to properly feed into the neural network.

Firstly, feature selection must be performed to choose the most relevant features for the problem. For this particular model, the features *AssessmentID*, *SiteID*, *Study*, *RaterID*, and *LeadStatus* are neglected as it unnecessarily increases the dimension of the problem without adding significant information gain to the model.

Additionally, the dataframe must be normalized to give the same weighting / significance to each feature and allow faster convergence for the gradient descent process on the neural network backpropagation stage. From testing the model and optimizing the RMSE, it can be seen that minmax normalization is preferred over standardization.

```
df_scaled = (inp_df-inp_df.min())/(inp_df.max()-inp_df.min())
```

Finally, the final feature transformation that needs to be performed is to apply one-hot-encoding to the categorical features *TxGroup* and *Country*. This allows the categorical features to be represented by multiple separate feature columns for each category, e.g. "*Treatment*" and "*Control*" for TxGroup, and "*USA*", "*Russia*", "*India*", etc for Country. A patient who falls into one of the categories of the categorical feature will have a 1 in the new respective one-hot-encoded column, whilst a 0 everywhere else.

```
# Perform One-Hot Encoding on Country and TxGroup
label_encoder = LabelEncoder()
country_encoded = label_encoder.fit_transform(combined_df['Country'].to_numpy())
txgroup_encoded = label_encoder.fit_transform(combined_df['TxGroup'].to_numpy())
country_ohc = to_categorical(country_encoded)
txgroup_ohc = to_categorical(txgroup_encoded)
merged_array = np.hstack((country_ohc, txgroup_ohc))
```

Next, the input (features) and output (response) must be separated from the main data. Here, the input features includes all individual symptom scores P1, P2, ..., G16, as well as *VisitDay* and *TxGroup*. The output responses are the 30 symptom scores P1, P2, ..., G16 at the end of the patient assessment (presumed to be the 18-th week values). The final 18-th week PANSS score is then the sum of the output symptom scores. The input to the RNN model is a 3-dimensional matrix, where each observation is a 2-dimensional numpy array which represents the time-series sequence of a patient's symptom scores. Note, for training the model, only patients who has reached the 18-th week mark (at least day 126) are used, whilst the others are neglected. An example of an input observation and its corresponding 2-d numpy representation is shown in the figure below.

Patient X Observation

VisitDay	P1	P2	P3	...	G13	G14	G15	G16
0	7	6	6	...	5	4	7	6
32	5	5	6	...	3	5	6	5
...	...	...	...	...	...	...	...	...
164	3	2	2	...	1	3	2	4
212	2	1	2	...	1	3	3	3

Input NumPy Matrix

$$\begin{bmatrix} 0 & 7 & 6 & 6 & \dots & 5 & 4 & 7 & 6 \\ 32 & 5 & 5 & 6 & \dots & 3 & 5 & 6 & 5 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 164 & 3 & 2 & 2 & \dots & 1 & 3 & 2 & 4 \end{bmatrix}$$

Output Array

$$\begin{bmatrix} 2 & 1 & 2 & \dots & 1 & 3 & 3 & 3 \end{bmatrix}$$

Note at this stage in the actual code, the numpy matrix would have been normalized and also contains the new one-hot-encoded features, but this is neglected here for convenience of explanation and demonstration. This input-output structure, along with the LSTM RNN model, allows the neural network to learn the sequential pattern behind the patients' symptom scores throughout the treatment program, as well as learning the pattern of a particular patient's day of visit to the clinic.

In addition to this, zero-padding must be performed due to the varying number of PANSS clinical visits of each patient, which results in inconsistent numpy matrix dimensions. For example, one patient may have done sixteen PANSS interviews, whilst another patient may have only done five. This is accomplished by simply filling an array of 0's at the beginning of the input matrix.

4.2 Creating and Defining RNN Model

The RNN model is created using the Keras library in Python. It is made with a Sequential model with an input layer, a Bidirectional LSTM layer with a tanh (hyperbolic tangent) activation function, as well as an output dense layer made of 30 neurons (i.e. the 30 symptom scores). *Tanh* proved to perform better (smaller test error) than the more common activation function *ReLU (Rectified Linear Unit)* and suffers little from the vanishing gradient problem as our neural network is not sufficiently deep, and hence still converges quickly. A bidirectional layer comprises of two hidden layers, a backward and a forward layer, whose output is combined and sent to the next layer (in this case a dense output layer). In essence, it allows our input to be run in two ways, from the past to the future as well as the future to the past in order to extract more specific features of our sequential data [1]. Figure 15 shows the general bidirectional RNN architecture implemented in this model.

```
def createRNN(x_train, y_train, epoch):
    model = Sequential()
    model.add(Bidirectional(LSTM(30, activation='tanh'), input_shape=(x_train.shape[1:])))
    # 30 PANSS symptom scores, no activation function
    model.add(Dense(y_train.shape[1]))
    model.compile(loss="mse", optimizer="adam", metrics=[
                  tf.keras.metrics.RootMeanSquaredError()])
    model.fit(x_train, y_train, epochs=epoch, batch_size=5, verbose=1, validation_split=0.1)
    return model
```

4.3 Training the RNN Model

The RNN model, as mentioned previously, is trained using the patient data whose visit day is at least 126 (i.e. they have potentially reached their 18-th week assessment). Each observation is a 2-dimensional matrix whose rows represent subsequent PANSS assessment, allowing the model to learn patterns in its sequence. The output is the 30 individual PANSS symptom scores at the end of the 18-th week assessment.

```
model = createRNN(x_mtx, y_mtx, 80)
ypred = model.predict(x_applied_mtx)
```

Here, training is performed with an epoch of 80 (to prevent overfitting), and a batch size of 5 (observe createRNN function previously), meaning the model will look at 5 input observations before updating its parameters through back-propagation.

4.4 RNN Model Performance

After obtaining the corresponding predictions of the 18-th week symptom scores of patients in study E, any scores that are less than 1 or greater than 7 are rounded to 1 and 7 respectively. Afterwards, the PANSS scores are summed and rounded to give the final PANSS score prediction on the 18-th week. The result *ypred* is then saved and stored on a csv file.

```
ypred[ypred < 1] = 1
ypred[ypred > 7] = 7
ypred = np.sum(ypred, axis=1)
ypred = np.round(ypred, 0)
y_result = pd.DataFrame(data=ypred, columns=["PANSS_Total"])
y_result.to_csv(r'result.csv', index=False)
```

The overall model performs relatively well on the study E patient data, giving an RMSE result of 6.29518. The primary issue in using an RNN model for this scenario is the missing and inconsistent time-spaced data. The day of visit in each observation sequence is different, and therefore the RNN model will find it more difficult to extract patterns in the sequence. e.g. patient X may see a significant drop in PANSS score between assessments than patient Y, but this may be because patient X skipped many weeks before another assessment, whilst patient Y attends weekly assessments. In addition, the existence of inconsistent PANSS assessment that is either flagged / assigned to CS (as opposed to passed) makes the training data more noisy, and hence the prediction may also diverge more from the true PANSS score.

5 Binary classification

Kaggle username: Arie Arya, Public Leaderboard Rank: 20

This section concerns the classification of patient assessments in study E using R into either the *passed* or *flagged / assign to CS* categories, respectively represented by class 0 and class 1 in the model. In particular, three common classification models is experimented upon: Logistic Regression, Random Forests, and Gradient Boosting. For each classification method, a probability of a patient assessment being classified to class 1 (flagged / assign to CS) is obtained. For the purposes of evaluation and optimization of the models, given this probability is greater than 0.5, then a class of 1 is assigned to the assessment, otherwise a class of 0. This will then be compared to the true class in the train or test set to evaluate its performance.

5.1 Data Pre-processing

Data pre-processing for binary classification in R is relatively straightforward. Firstly, the four studies must be merged into one to reduce the bias of our model. Additionally, the *LeadStatus* column categories must be transformed, where categories "*AssigntoCS*" and "*Flagged*" are combined into class 1, and category "*Passed*" into class 0. Lastly, feature selection must be performed to select relevant features for this problem, and the categorical feature column *TxGroup* must be changed into factor type to allow it to be treated as a categorical feature in the model.

```
# Set Flagged or CS as 1, Passed to 0
df$LeadStatus[df$LeadStatus == "Assign to CS"] = 1
df$LeadStatus[df$LeadStatus == "Flagged"] = 1
df$LeadStatus[df$LeadStatus == "Passed"] = 0
df = subset(df, select=-c(Study, PatientID, Country, SiteID, RaterID, AssessmentID))
df$TxGroup = as.factor(df$TxGroup)
```

In addition, the data is subsequently split into training and test sets at a 80:20 ratio for model evaluation and optimization.

```
smp_size = 0.8*nrow(df)
train_index = sample(nrow(df), smp_size)
data.train = df[train_index, ]
data.test = df[-train_index, ]
```

5.2 Logistic Regression

Logistic Regression is a classification model obtained by passing a linear model through a sigmoid function, thus bounding its value between 0 and 1. This bound allows for convenient representation of the probability of a certain class in a binary classification problem, shown in the equation below:

$$Pr = \frac{e^{\beta_0+\beta_1 x+\ldots}}{1+e^{\beta_0+\beta_1 x+\ldots}} \tag{4}$$

Here, the β coefficients are optimized through the training process using the glm() function in R.

```
# Fit using Logistic Regression
log_fit = glm(as.factor(LeadStatus) ~., data=data.train, family = binomial)
log_prob = predict(log_fit, data.test, type="response")
log_pred = ifelse(log_prob > 0.5, 1, 0)
print(table(data.test$LeadStatus, log_pred))
log_acc = sum(data.test$LeadStatus == log_pred)/nrow(data.test)
print(paste("Logistic Regression accuracy: ", log_acc))
```

The corresponding test accuracy, i.e. the proportion of correctly predicted classes, obtained using Logistic Regression by setting the prediction to the most probable class is **0.76563245823389**. This performs comparatively worse than both Random Forest and Gradient Boosting.

5.3 Random Forests

Random forest is a classification algorithm which utilizes a form of bagging on uncorrelated trees. This is achieved by selecting only a certain number of features to consider in each split of the bagged decision tree (i.e. the hyper-parameter *mtry* in R), hence making it unlikely that any two bagged decision trees are exactly the same and are therefore uncorrelated. Random forest reduces the tendency of decision trees from over-fitting to the training data.

Firstly, it is important to optimize the mtry hyper-parameter of the random forest. Here, we can iterate mtry over the total number of feature columns and observe the test accuracy of our decision tree.

```
# Plots the test accuracy against mtry for Random Forest
num_pred = ncol(data.train)-1
test_accuracy = 1:num_pred
for(i in 1:num_pred){
    print(paste("iteration: ", i, "/ ", num_pred))
    rf_fit = randomForest(as.factor(LeadStatus) ~., data=data.train,
        mtry = i, ntree=1000)
    rf_prob = predict(rf_fit, data.test, ntree=1000, type="prob")
    rf_prob = rf_prob[,2]
    rf_pred = ifelse(rf_prob > 0.5, 1, 0)
    rf_acc = sum(data.test$LeadStatus == rf_pred)/nrow(data.test)
    print(paste("rf accuracy: ", rf_acc))
    test_accuracy[i] = rf_acc
}
plot(1:num_pred, test_accuracy, type="b", xlab="mtry", ylab="Test Accuracy")
```

Figure 16 shows the resulting plot of test accuracy against the mtry values. As can be observed, an mtry value of 7 provides the highest test accuracy, and is therefore used for the final RF model. The corresponding Random Forest test accuracy obtained by setting the prediction to the most probable outcome is **0.839379474940334**. Interestingly, even as the accuracy is significantly higher than that of Gradient Boosting (next section), its log-loss performance on the study E patient data is considerably worse. This may indicate that the probability values may possibly be more skewed and inaccurate.

5.4 Gradient Boosting

Gradient boosting performs most optimally on study E patient data compared to Logistic Regression and Random Forest. Gradient boosting is an algorithm which creates a strong prediction model with notably lower bias from an ensemble of weak learners, most typically decision trees. Each subsequent weak learner is obtained through fitting on the residual of the ensemble of the previous weak learners, and the final model is the sum of the weak learners and its corresponding learning rates. For R, gradient boosting can be performed using the gbm function.

```
# Fit using Boosting
boost_fit = gbm(LeadStatus ~., data=data.train, cv.folds=10, bag.fraction = 0.75,
                distribution="bernoulli", n.trees=1000, shrinkage=0.05)
boost_prob = predict(boost_fit, data.test, n.trees = 1000, type = "response")
boost_pred = ifelse(boost_prob > 0.5, 1, 0)
print(table(data.test$LeadStatus, boost_pred))
boost_acc = sum(data.test$LeadStatus == boost_pred)/nrow(data.test)
print(paste("Boost accuracy: ", boost_acc))
```

Here, the distribution is kept as Bernoulli to restrict the output between 0 and 1, and the shrinkage parameter is kept relatively low to allow the model to learn more slowly, hence capturing more specific features of the distribution. Upon training and evaluating the model on study A-D patients, the corresponding test accuracy is obtained as **0.781837708830549**. Though this test accuracy is notably lower than that of random forest, its log-loss performance on study E patients shows considerable improvement (0.61643), suggesting its output probabilities is more accurate and thus more closely resembling the true data distribution.

6 Appendix

6.1 Figures and Diagrams

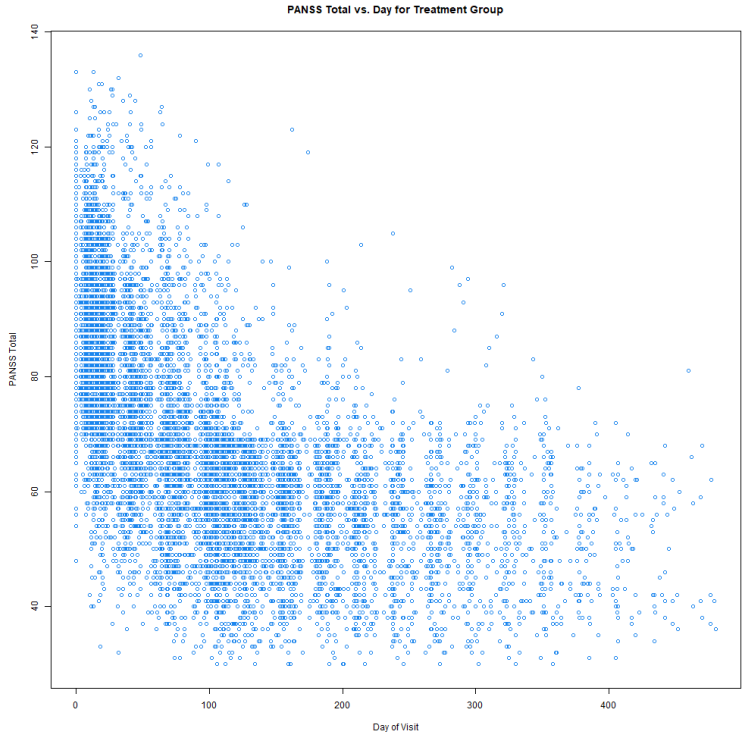

Figure 1: Patients' PANSS score in the treatment group throughout the program

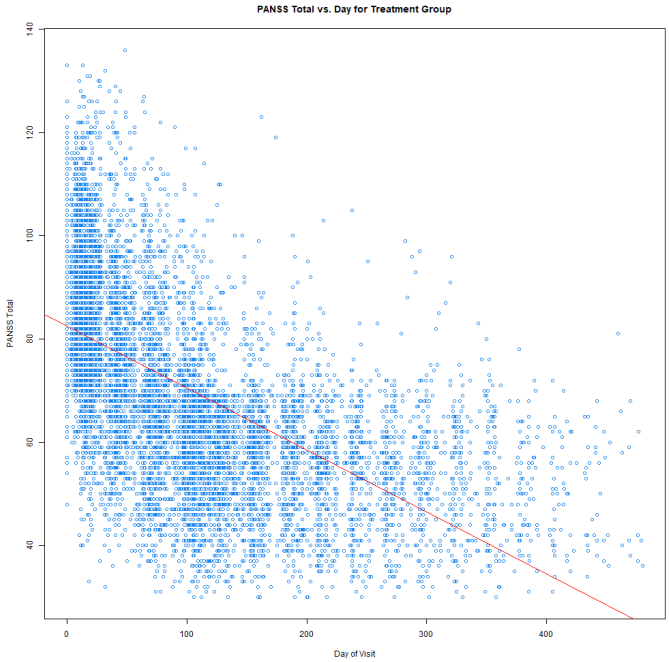

Figure 2: Linear Fit on pre-processed data

Figure 3: Patients' 30 symptom scores in the treatment group throughout the program

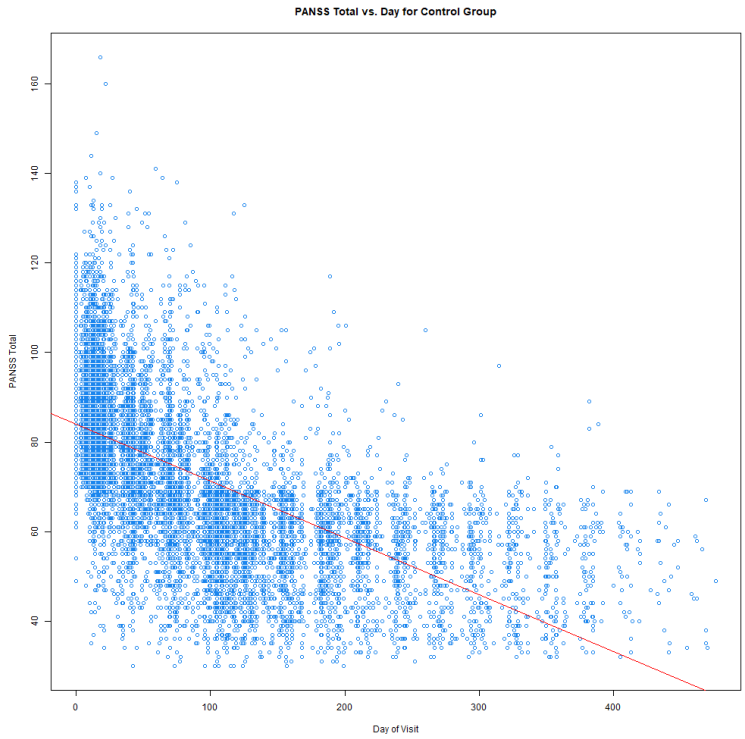

Figure 4: Linear Fit on Control Group Patients

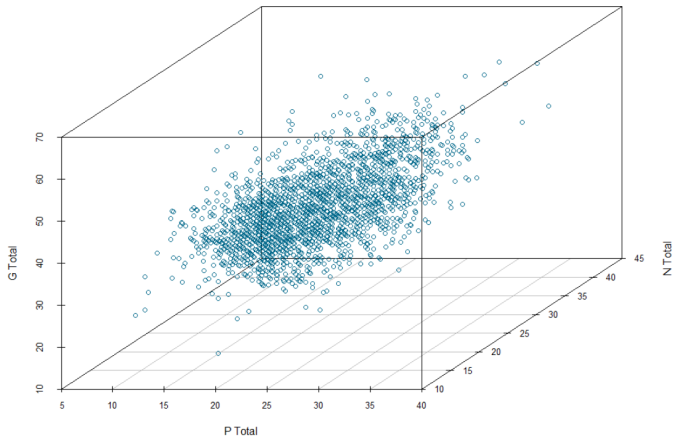

Figure 5: Scatterplot of patient's P, N, and G scores

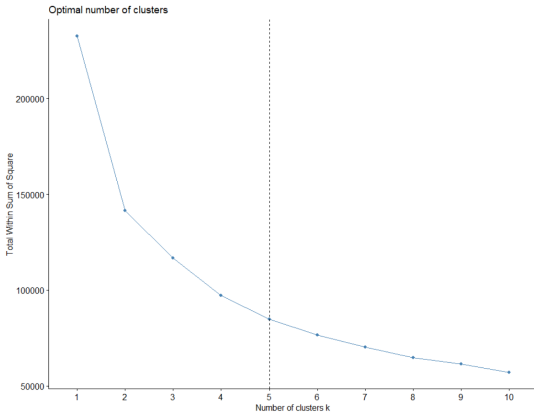

Figure 6: Elbow Method on the 3D dataset

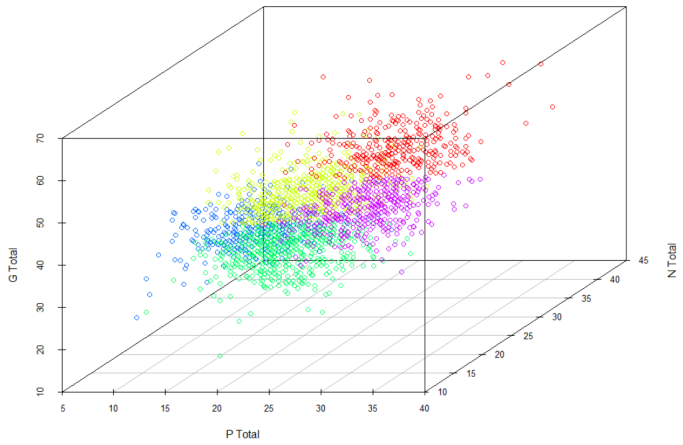

Figure 7: Clusterplot of 3D dataset

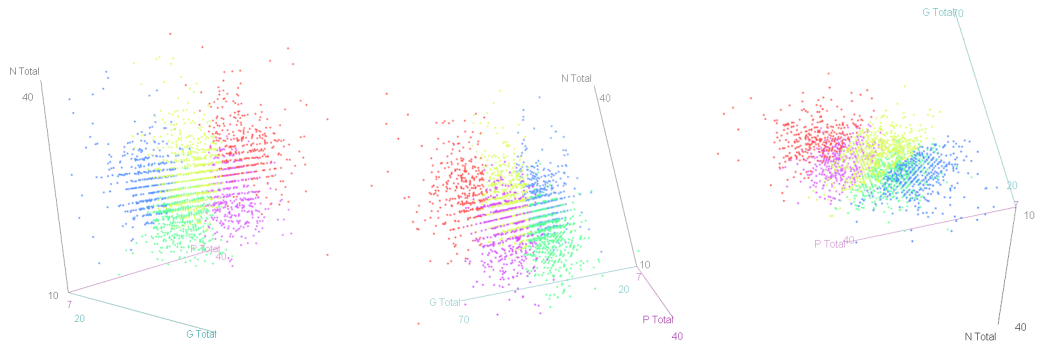

Figure 8: Different visualizations of the 3D Clusterplot

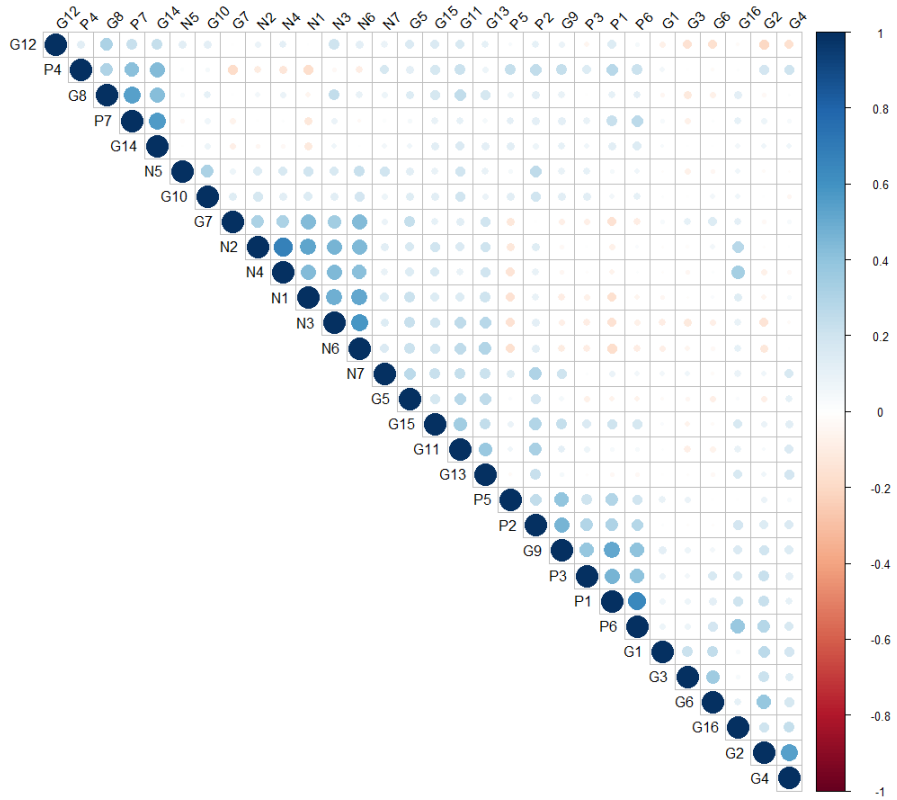

Figure 9: Correlation Matrix between 30 PANSS symptoms

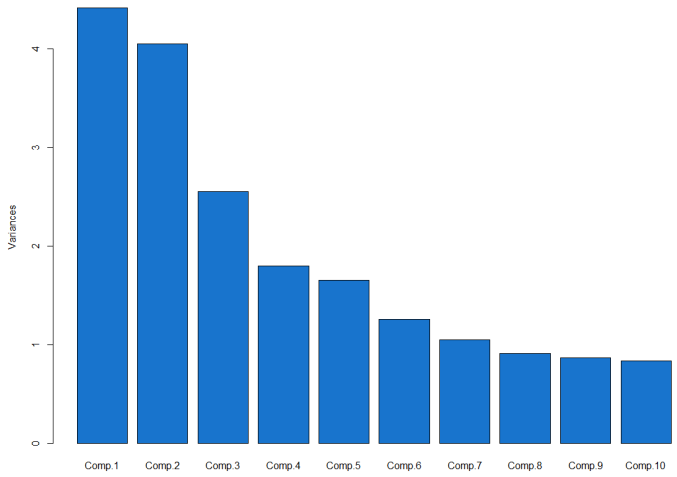

Figure 10: Principal Components and their Proportion of Total Variance

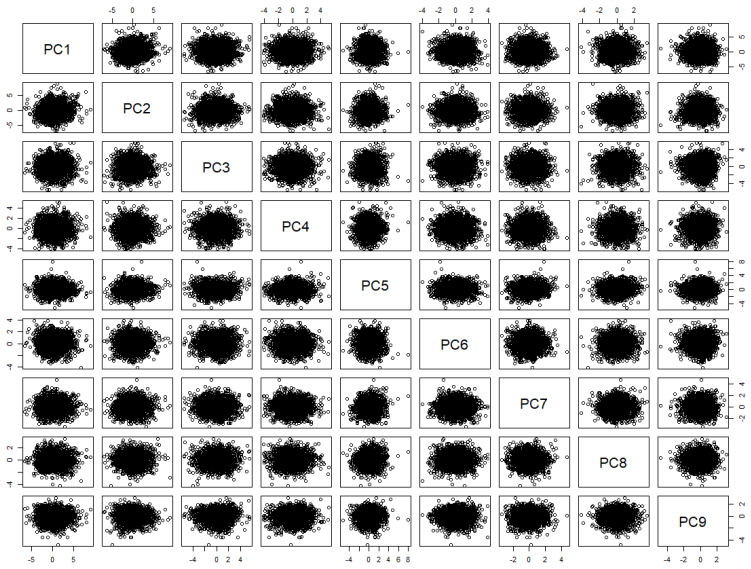

Figure 11: Correlation Matrix of the first 9 Principal Components

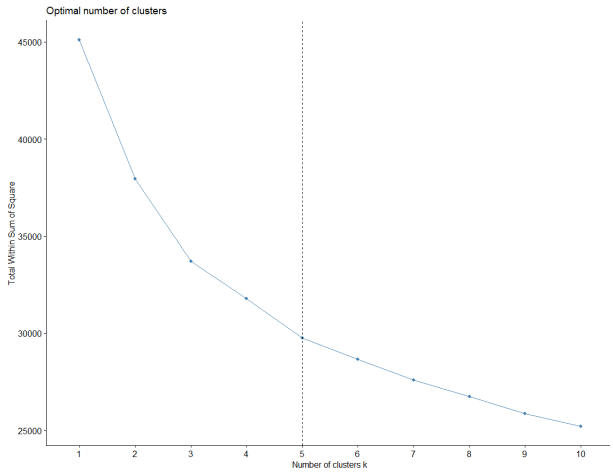

Figure 12: Elbow Method on PCA dataset

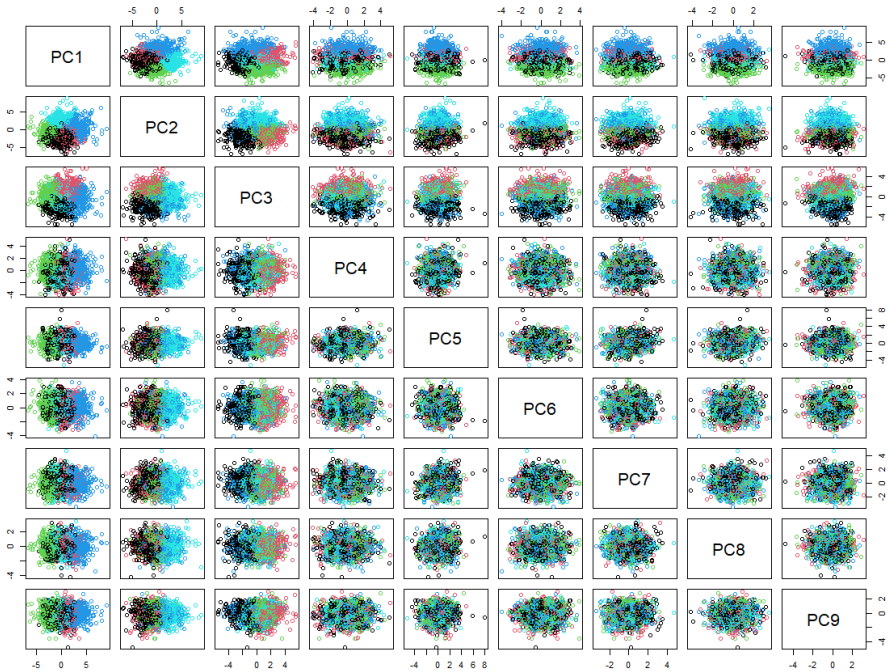

Figure 13: Cluster Illustration from the 9 Principal Components

	P1	P2	P3	P4	P5	P6
C1 Mean	0.55085027	0.1030059	0.03282209	0.7014167	0.52704003	0.2767536
C2 Mean	0.67831128	0.7685820	0.80378923	0.1251413	0.35630824	0.8224101
C3 Mean	-0.21242323	-0.7076123	-0.08475754	-0.2076490	-0.14265233	-0.2820382
C4 Mean	0.08630077	0.4980728	0.03043778	0.4706159	-0.05202001	0.1454371
C5 Mean	-0.84305378	-0.3739988	-0.69375176	-0.6962222	-0.51201172	-0.7520569

	P7	N1	N2	N3	N4	N5
C1 Mean	0.7389268	-0.8047276	-0.7055892	-0.3773909	-0.6124692	-0.16044582
C2 Mean	-0.1132336	0.1692330	0.2771232	-0.1582904	0.1826200	0.18278542
C3 Mean	-0.2777446	-0.5314340	-0.5698765	-0.5840922	-0.5435981	-0.44476724
C4 Mean	0.7126840	0.7163404	0.7811113	1.0498964	0.7787858	0.56495407
C5 Mean	-0.5896107	0.4820706	0.3113217	0.3219504	0.3078970	0.03207225

	N6	N7	G1	G2	G3	G4
C1 Mean	-0.6658893	0.02753082	-0.23381693	-0.28061071	-0.3557350	-0.3642067
C2 Mean	-0.0471629	0.17703819	0.34203451	0.70644479	0.4166495	0.5105578
C3 Mean	-0.5508850	-0.39093592	0.16648081	0.10283069	0.2531448	0.0101764
C4 Mean	1.0164575	0.46942315	-0.08945388	-0.06055149	-0.2518976	0.2020062
C5 Mean	0.4078274	-0.08609343	-0.26502702	-0.52060035	-0.2318664	-0.3590886

	G5	G6	G7	G8	G9	G10
C1 Mean	-0.16546385	-0.4804749	-0.708734257	0.6383755	0.3396916	-0.1494117
C2 Mean	-0.02207843	0.5708239	-0.004561449	-0.2712908	0.8307194	0.2138493
C3 Mean	-0.50797315	0.1873952	-0.326208525	-0.3943803	-0.3783002	-0.3583216
C4 Mean	0.72620709	-0.2242544	0.650716599	0.9891127	0.1173691	0.5861043
C5 Mean	0.18328898	-0.2371770	0.406350594	-0.4354295	-0.6856705	-0.1071868

	G11	G12	G13	G14	G15	G16
C1 Mean	0.02384380	0.6605488	-0.1623634	0.6790082	-0.03455756	-0.3435870
C2 Mean	0.11676574	-0.3018744	0.0436878	-0.1383781	0.36753579	0.6334983
C3 Mean	-0.56233672	-0.5142019	-0.5050471	-0.1825223	-0.50508680	-0.3972126
C4 Mean	0.82471351	0.6353377	0.7447138	0.6659888	0.71954670	0.4053889
C5 Mean	-0.08419235	-0.0672039	0.1055651	-0.5925485	-0.26416718	-0.1993226

Figure 14: Mean symptom scores of each cluster C1, C2, C3, C4, and C5

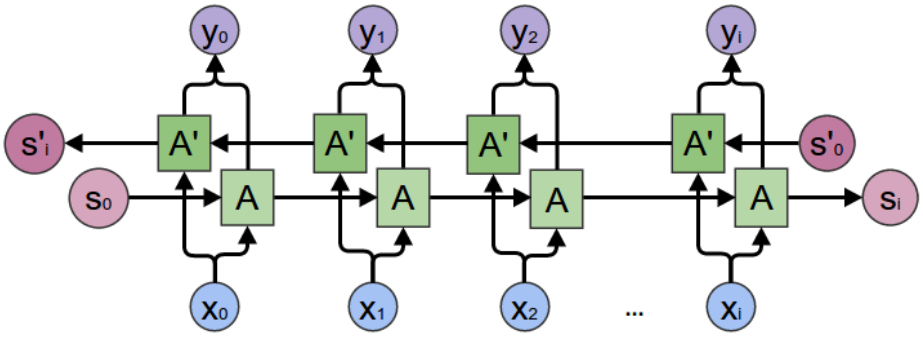

Figure 15: Bidirectional RNN Architecture

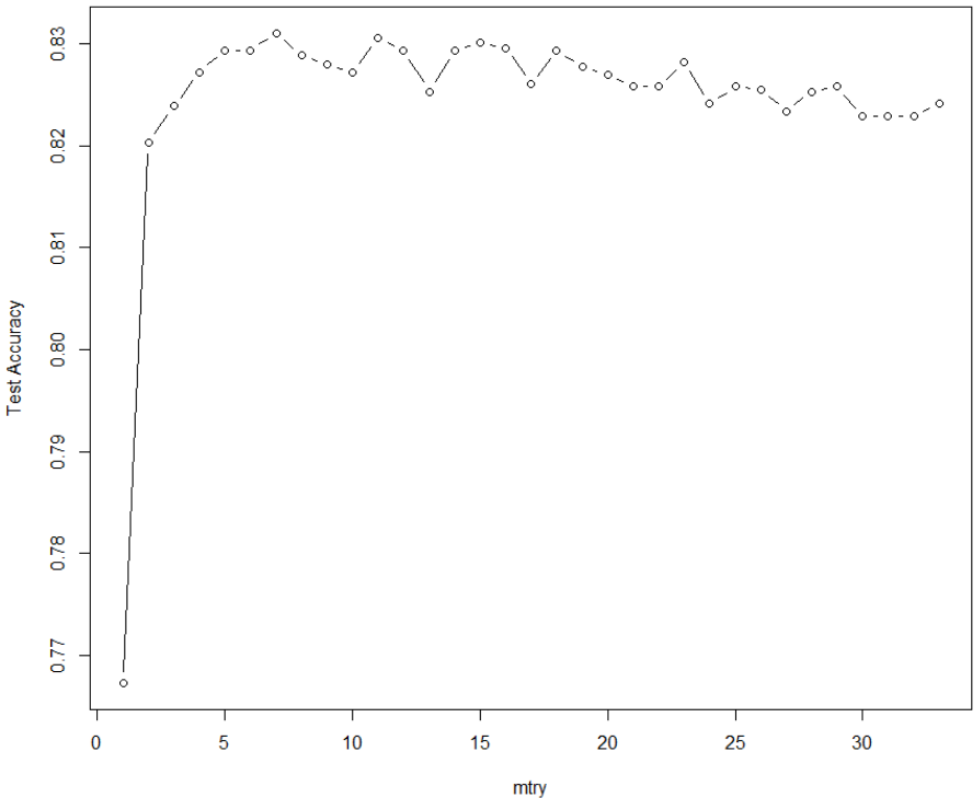

Figure 16: Random Forest Test Accuracy vs. mtry

6.2 Part 1 Code: Treatment Effect

```
suppressPackageStartupMessages({
  library(logging)
  library(tidyverse)
})

#' The width and height of exported plots.
FIGSIZE = c(1000, 1000)

DATA_PATH = "..."

LoadData <- function(data_path, verbose=TRUE) {
  stopifnot(file.exists(data_path))
  if (verbose)
    loginfo(sprintf("\tLoading␣%s", data_path))
  data <- read.csv(data_path)
  return (data)
}

Part1TreatmentGroup <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)

  # Select only the patients in the treatment group
  df = filter(df, TxGroup == "Treatment")

  # Select the useful columns
  df = df[c("VisitDay", "PANSS_Total")]
  # print(head(df, 5))

  # Print out the Scatterplot
  visitDay = df$VisitDay
  panssTotal = df$PANSS_Total
  png(paste(DATA_PATH, "/Part␣1/Part1Treatment.png", sep=""), width=FIGSIZE[1],
                height=FIGSIZE[2])
  plot(x = visitDay, y = panssTotal, main="PANSS␣Total␣vs.␣Day␣for␣Treatment␣Group",
                xlab="Day␣of␣Visit", ylab="PANSS␣Total", col="dodgerblue2")

  # Obtain a Linear Fit
  linear_fit = lm(panssTotal~visitDay)
  print(summary(linear_fit))
  abline(linear_fit, col="red")
  dev.off()

  # Print the gradient of the Regression Line
  print(coef(linear_fit))
}

Part1ControlGroup <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)
  print(head(df, 5))

  # Select only the patients in the control group
```

```
  df = filter(df, TxGroup == "Control")

  # Select the useful columns
  df = df[c("VisitDay", "PANSS_Total")]
  print(head(df, 5))

  # Print out the Scatterplot
  visitDay = df$VisitDay
  panssTotal = df$PANSS_Total
  png(paste(DATA_PATH, "/Part 1/Part1Control.png", sep=""), width=FIGSIZE[1],
                height=FIGSIZE[2])
  plot(x = visitDay, y = panssTotal, main="PANSS Total vs. Day for Control Group",
                xlab="Day of Visit", ylab="PANSS Total", col="dodgerblue2")

  # Obtain a Linear Fit
  linear_fit = lm(panssTotal~visitDay)
  print(summary(linear_fit))
  abline(linear_fit, col="red")
  dev.off()

  # Print the gradient of the Regression Line
  print(coef(linear_fit))
}

Part1TreatmentGroupCombinedPlots <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)

  # Select only the patients in the treatment group
  df = filter(df, TxGroup == "Treatment")

  # Select the corresponding columns
  visitDay = df$VisitDay

  # Loop over the 30 symptoms
  png(paste(DATA_PATH, "/Part 1/Part1TreatmentCombinedPlots.png", sep=""), width=FIGSIZE[1],
                    height=FIGSIZE[2])
  par(mfrow=c(5, 6))
  for (i in 0:29){
    symptom_score = df[,9+i]
    plot(x = visitDay, y = symptom_score, xlab="Day of Visit", ylab=colnames(df[9+i]),
                    col="dodgerblue2", las=1)
  }
  dev.off()
}
```

6.3 Part 2 Code: Patient Segmentation

```
suppressPackageStartupMessages({
  library(logging)
  library(tidyverse)
  library(factoextra)
  library(cluster)
  library("scatterplot3d")
  library(car)
  library(rgl)
  library(FactoMineR)
  library(corrplot)
})

#' The width and height of exported plots.
FIGSIZE = c(1000, 1000)

DATA_PATH = "..."

LoadData <- function(data_path, verbose=TRUE) {
  stopifnot(file.exists(data_path))
  if (verbose)
    loginfo(sprintf("\tLoading␣%s", data_path))
  data <- read.csv(data_path)
  return (data)
}

# Segmentation by 30 PANSS symptoms
Part2SegmentationBy30Symptoms <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df_E = LoadData(paste(DATA_PATH,"/Study_E.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)

  # Select only readings with day of visit 0
  df = filter(df, VisitDay == 0)

  # Select the useful columns
  df = subset(df, select = -c(Study, Country, SiteID, RaterID, AssessmentID,
                    TxGroup, LeadStatus, VisitDay, PANSS_Total))

  # Remove duplicate rows with the same PatientID (multiple assessments on same day)
  df = df[!duplicated(df$PatientID), ]
  rownames(df) <- df$PatientID
  df$PatientID <- NULL
  df = scale(df)

  # Find optimum k with Elbow Method
  km.elbow = fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 4,
                    linetype = 2)
  print(km.elbow)

  # Find optimum k with Silhouette Method
  km.silh = fviz_nbclust(df, kmeans, method = "silhouette")
  print(km.silh)

  # Let us use k = 2. Iterate
  opt.km_result = kmeans(df, 2, nstart = 25)
```

```
  min_wss = as.numeric(unlist(opt.km_result[5]))
  print(typeof(min_wss))
  for(i in 1:20){
    curr.km_result = kmeans(df, 2, nstart = 25)
    curr_wss = as.numeric(unlist(curr.km_result[5]))
    if(curr_wss < min_wss){
      opt.km_result = curr.km_result
      min_wss = curr_wss
    }
  }
  print(opt.km_result)

  fviz_cluster(opt.km_result, data = df)
}

# Segmentation by 3 PANSS Categories
Part2SegmentationBy3Categories <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df_E = LoadData(paste(DATA_PATH,"/Study_E.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D, df_E)

  # Select only readings with day of visit 0
  df = filter(df, VisitDay == 0)

  # Obtain columns based on sum of symptoms in each category
  P_Total <- df$P1+df$P2+df$P3+df$P4+df$P5+df$P6+df$P7
  N_Total <- df$N1+df$N2+df$N3+df$N4+df$N5+df$N6+df$N7
  G_Total <- df$G1+df$G2+df$G3+df$G4+df$G5+df$G6+df$G7
                +df$G8+df$G9+df$G10+df$G11+df$G12+df$G13+df$G15+df$G16

  df = subset(df, select = c(PatientID))

  symptom_df = data.frame(P_Total, N_Total, G_Total)
  df = cbind(df, symptom_df)

  # Remove duplicate rows with the same PatientID (multiple assessments on same day)
  df = df[!duplicated(df$PatientID), ]
  rownames(df) <- df$PatientID
  df$PatientID <- NULL
  # df = data.frame(scale(df))

  # Plot out 3d scatterplot
  scatterplot3d(x=df$P_Total, y=df$N_Total, z=df$G_Total, xlab="P␣Total", ylab="N␣Total",
                zlab="G␣Total", color="deepskyblue4")

  # Find optimum k with Elbow Method
  km.elbow = fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 5,
                linetype = 2)
  print(km.elbow)

  # Let us use k = 5 clusters
  opt.km_result = kmeans(df, 5, nstart = 25)
  min_wss = as.numeric(unlist(opt.km_result[5]))

  for(i in 1:20){
    curr.km_result = kmeans(df, 5, nstart = 25)
    curr_wss = as.numeric(unlist(curr.km_result[5]))
    if(curr_wss < min_wss){
```

```
      opt.km_result = curr.km_result
      min_wss = curr_wss
    }
  }

  df$cluster = factor(opt.km_result$cluster)
  scatterplot3d(x=df$P_Total, y=df$N_Total, z=df$G_Total, xlab="P Total", ylab="N Total",
                zlab="G Total", color=rainbow(5)[opt.km_result$cluster])

  scatter3d(x=df$P_Total, y=df$N_Total, z=df$G_Total, surface=FALSE,
            point.col=rainbow(5)[opt.km_result$cluster], xlab="P Total",
            ylab="N Total", zlab="G Total")

  print(opt.km_result[2])
}

# Segmentation by PCA Analysis and k-means
Part2SegmentationByPCA <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df_E = LoadData(paste(DATA_PATH,"/Study_E.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)

  # Select only readings with day of visit 0
  df = filter(df, VisitDay == 0)

  # Select the columns with the symptoms
  df = subset(df, select = -c(Study, Country, SiteID, RaterID, AssessmentID,
                TxGroup, LeadStatus, VisitDay, PANSS_Total))

  # Remove duplicate rows with the same PatientID (multiple assessments on same day)
  df = df[!duplicated(df$PatientID), ]
  rownames(df) <- df$PatientID
  df$PatientID <- NULL
  df = data.frame(scale(df))

  # Plot the correlation matrix
  corr_mtx = cor(df)
  corrplot(corr_mtx, type = "upper", order = "hclust",
           tl.col = "black", tl.srt = 45)

  # Obtain Principle Components
  pc <- princomp(df)
  plot(pc, col="dodgerblue3")
  print(summary(pc))

  # Choose the number of PC that reaches 60% of total variance
  pc <- prcomp(df)
  var_list = summary(pc)$importance[2,]
  print(var_list)
  pc_length = 0
  cur_var_prop = 0
  for(i in 1:length(var_list)){
    cur_var_prop = cur_var_prop + var_list[i]
    if(cur_var_prop >= 0.60){
      pc_length = i
      break
    }
  }
```

```
  print(pc_length)

  # Select the first i principle components to represent more than 60% variance
  sel_comp = data.frame(pc$x[,1:i])
  plot(sel_comp)

  # Find optimum k with Elbow Method
  km.elbow = fviz_nbclust(sel_comp, kmeans, method = "wss") + geom_vline(xintercept = 5,
                linetype = 2)
  print(km.elbow)

  # use k = 5 clusters
  km_result = kmeans(df, 5, nstart = 25)
  plot(sel_comp, col=km_result$clust)

  # Split into separate dataframes corresponding to cluster
  cluster_df = data.frame(km_result$cluster)
  cl1_df = filter(cluster_df, km_result.cluster == 1)
  cl2_df = filter(cluster_df, km_result.cluster == 2)
  cl3_df = filter(cluster_df, km_result.cluster == 3)
  cl4_df = filter(cluster_df, km_result.cluster == 4)
  cl5_df = filter(cluster_df, km_result.cluster == 5)

  # Filter into main dataframe to obtain the symptom scores
  df_1 <- subset(df, rownames(df) %in% rownames(cl1_df))
  df_2 <- subset(df, rownames(df) %in% rownames(cl2_df))
  df_3 <- subset(df, rownames(df) %in% rownames(cl3_df))
  df_4 <- subset(df, rownames(df) %in% rownames(cl4_df))
  df_5 <- subset(df, rownames(df) %in% rownames(cl5_df))

  # Obtain the means of all symptom scores in each cluster
  res.means_df = rbind(sapply(df_1, mean), sapply(df_2, mean), sapply(df_3, mean),
                 sapply(df_4, mean), sapply(df_5, mean))
  rownames(res.means_df) <- c("C1␣Mean","C2␣Mean","C3␣Mean","C4␣Mean","C5␣Mean")
  print(res.means_df)
}
```

6.4 Part 3 Code: Forecasting

```
import csv
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional
import itertools
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from numpy import mean, sqrt, square
from tensorflow.keras.metrics import RootMeanSquaredError
import matplotlib.pyplot as plt

# Splits the training data and the applied (study E) data and performs one-hot-encoding
def init_data_split(train_df, applied_df):
    train_df = train_df.drop_duplicates(
        subset=["PatientID", "VisitDay"], keep='first')
    applied_df = applied_df.drop_duplicates(
        subset=["PatientID", "VisitDay"], keep='first')
    cut_index = len(train_df)
    combined_df = pd.concat([train_df, applied_df])

    # Perform One-Hot Encoding on Country and TxGroup
    label_encoder = LabelEncoder()
    country_encoded = label_encoder.fit_transform(combined_df['Country'].to_numpy())
    txgroup_encoded = label_encoder.fit_transform(combined_df['TxGroup'].to_numpy())

    country_ohc = to_categorical(country_encoded)
    txgroup_ohc = to_categorical(txgroup_encoded)

    merged_array = np.hstack((country_ohc, txgroup_ohc))
    df_ohc = pd.DataFrame(data=merged_array)

    combined_df = combined_df.drop(['Study', 'SiteID', 'RaterID', 'AssessmentID',
                  'PANSS_Total', 'Country', 'TxGroup'], axis=1)

    combined_df = pd.concat([combined_df.reset_index(drop=True),
                    df_ohc.reset_index(drop=True)], axis=1)

    return (combined_df, cut_index)

# Calculates the maximum number of occurences of the same patient
def count_max_occur(df):
    # First calculate the largest number of occurences of the same patient
    cur_patient_id = df.iloc[0, 2]
    cur_count = 0
    max_count = 0
    max_id = cur_patient_id
    for i in range(0, len(df)):
        if df.iloc[i, 0] == cur_patient_id:
            cur_count += 1
        else:
            if max_count < cur_count:
                max_count = cur_count
                max_id = cur_patient_id
            cur_patient_id = df.iloc[i, 0]
            cur_count = 1
```

```
    return max_count, max_id

# Obtains the output (response) of the training set
def extract_y(df):
    y_mtx = []
    cur_patient_id = df.iloc[0, 0]
    for i in range(0, len(df)):
        print("preprocessing iteration ", i, "/", len(df), " [1/2]")
        if i != len(df)-1:
            if df.iloc[i+1, 0] != cur_patient_id:
                y_mtx.append(df.iloc[i, 2:32].tolist())
                cur_patient_id = df.iloc[i+1, 0]
        else:
            y_mtx.append(df.iloc[i, 2:32].tolist())

    y_mtx = np.array(y_mtx)
    return (y_mtx)

# Obtains the input (features) of the training set
def extract_x(inp_df, max_count, cut_index):
    patient_id = inp_df['PatientID']
    # Minmax normalization
    inp_df = inp_df.drop(['PatientID'], axis=1)
    df_scaled = (inp_df-inp_df.min())/(inp_df.max()-inp_df.min())
    inp_df = pd.concat([patient_id, df_scaled], axis=1)
    x_train_df = inp_df[:cut_index]
    x_applied_df = inp_df[cut_index:]
    print(x_train_df.to_numpy().shape)
    print(x_applied_df.to_numpy().shape)
    x_combined = [x_train_df, x_applied_df]
    x_result = []
    for df in x_combined:
        x_mtx = []
        cur_x_mtx = []
        cur_patient_id = df.iloc[0, 0]
        cur_iter = 0
        row_length = len(df.drop(['PatientID'], axis=1).iloc[0, ].tolist())
        for i in range(0, len(df)):
            print("preprocessing iteration ", i, "/", len(df),  " [2/2]")
            cur_iter += 1
            if i != len(df)-1:
                # checks if next patient in list is still the same patient
                if df.iloc[i+1, 0] != cur_patient_id:
                    # Pad with 0's if there are empty rows
                    while(cur_iter <= max_count):
                        masked_row = [0]*row_length
                        cur_x_mtx.insert(0, masked_row)
                        cur_iter += 1
                    cur_iter = 0
                    x_mtx.append(cur_x_mtx)
                    cur_x_mtx = []
                    cur_patient_id = df.iloc[i+1, 0]
                else:
                    if(cur_iter <= max_count):
                        cur_x_mtx.append(
                            df.drop(['PatientID'], axis=1).iloc[i, ].tolist())
            else:
                # Pad with 0's if there are empty rows
                while(cur_iter <= max_count):
                    masked_row = [0]*row_length
                    cur_x_mtx.insert(0, masked_row)
```

```
                    cur_iter += 1
                cur_iter = 0
                x_mtx.append(cur_x_mtx)

        # convert to numpy array
        x_mtx = np.array(x_mtx)
        x_result.append(x_mtx)

    # returns x and y matrices
    return (x_result[0], x_result[1])

# Create RNN with one Bidirectional Layer and one Dense Layer
def createRNN(x_train, y_train, epoch):
    model = Sequential()
    model.add(Bidirectional(LSTM(30, activation='tanh'), input_shape=(x_train.shape[1:])))
    # 30 PANSS symptom scores, no activation function
    model.add(Dense(y_train.shape[1]))
    model.compile(loss="mse", optimizer="adam", metrics=[
                tf.keras.metrics.RootMeanSquaredError()])
    model.fit(x_train, y_train, epochs=epoch, batch_size=5, verbose=1, validation_split=0.1)
    return model

def main():
    # Read files into a Dataframe
    df_A = pd.read_csv('Study_A.csv')
    df_B = pd.read_csv('Study_B.csv')
    df_C = pd.read_csv('Study_C.csv')
    df_D = pd.read_csv('Study_D.csv')
    df_E = pd.read_csv('Study_E.csv')

    # Merge dataframes A to D together
    df = pd.concat([df_A, df_B, df_C, df_D])
    df.to_csv(r'CombinedStudy.csv', index=False)
    # print(df.head(5))

    # First filter the training model to include patients who finished 18 weeks of study
    train_df = df[df.LeadStatus == 'Passed']
    train_df = train_df[train_df.PatientID.isin(train_df[train_df["VisitDay"] >
                    120]["PatientID"].values.tolist())]
    train_df = train_df.drop(['LeadStatus'], axis=1)
    combined_df, cut_index = init_data_split(train_df, df_E)
    max_count, max_id = count_max_occur(combined_df)

    try:
        # Attempt to load the numpy array
        print("Attempting to open train numpy file...")
        with open('preproc_data.npy', 'rb') as f:
            x_applied_mtx = np.load(f)
            x_mtx = np.load(f)
            y_mtx = np.load(f)
    except:
        # Preprocess data and save it
        print("No train numpy file found, preprocessing data...")
        y_mtx = extract_y(combined_df[:cut_index])
        x_mtx, x_applied_mtx = extract_x(combined_df, max_count, cut_index)
        with open('preproc_data.npy', 'wb') as f:
            np.save(f, x_applied_mtx)
            np.save(f, x_mtx)
            np.save(f, y_mtx)

    x_train, x_test, y_train, y_test = train_test_split(x_mtx, y_mtx, test_size=0.2,
```

```
                    random_state=21314)

    # Obtain test MSE from study A-D
    y_test = np.sum(y_test, axis=1)
    model = createRNN(x_train, y_train, 80)
    ypred = model.predict(x_test)
    ypred[ypred < 1] = 1
    ypred[ypred > 7] = 7
    ypred = np.sum(ypred, axis=1)
    ypred = np.round(ypred, 0)
    print("RMSE error: ", np.sqrt(np.mean((y_test-ypred)**2)))

    # Obtain the prediction on study E
    model = createRNN(x_mtx, y_mtx, 80)
    ypred = model.predict(x_applied_mtx)
    ypred[ypred < 1] = 1
    ypred[ypred > 7] = 7
    ypred = np.sum(ypred, axis=1)
    ypred = np.round(ypred, 0)
    print(ypred)
    print(ypred.shape)
    y_result = pd.DataFrame(data=ypred, columns=["PANSS_Total"])
    y_result.to_csv(r'result.csv', index=False)

if __name__ == "__main__":
    main()
```

6.5 Part 4 Code: Binary Classification

```
suppressPackageStartupMessages({
  library(logging)
  library(tree)
  library(gbm)
  library(glmnet)
  library(randomForest)
  library(class)
  library(data.table)
  library(mltools)
  library(MLmetrics)
})

FIGSIZE = c(1000, 1000)

DATA_PATH = "..."

LoadData <- function(data_path, verbose=TRUE) {
  stopifnot(file.exists(data_path))
  data <- read.csv(data_path)
  return (data)
}

Part4 <- function(){
  df_A = LoadData(paste(DATA_PATH,"/Study_A.csv", sep=""))
  df_B = LoadData(paste(DATA_PATH,"/Study_B.csv", sep=""))
  df_C = LoadData(paste(DATA_PATH,"/Study_C.csv", sep=""))
  df_D = LoadData(paste(DATA_PATH,"/Study_D.csv", sep=""))
  df_E = LoadData(paste(DATA_PATH,"/Study_E.csv", sep=""))
  df = rbind(df_A, df_B, df_C, df_D)

  # Set Flagged or CS as 1, Passed to 0
  df$LeadStatus[df$LeadStatus == "Assign␣to␣CS"] = 1
  df$LeadStatus[df$LeadStatus == "Flagged"] = 1
  df$LeadStatus[df$LeadStatus == "Passed"] = 0
  df = subset(df, select=-c(Study, PatientID, Country, SiteID, RaterID, AssessmentID))
  df$TxGroup = as.factor(df$TxGroup)
  df_E = subset(df_E, select=-c(Study, PatientID, SiteID, Country, RaterID, AssessmentID))
  df_E$TxGroup = as.factor(df_E$TxGroup)
  df_E$LeadStatus = replicate(nrow(df_E), -1)

  smp_size = 0.8*nrow(df)
  train_index = sample(nrow(df), smp_size)

  data.train = df[train_index, ]
  data.test = df[-train_index, ]

  # Fit using Boosting
  boost_fit = gbm(LeadStatus ~., data=data.train, cv.folds=10, bag.fraction = 0.75,
                  distribution="bernoulli", n.trees=1000, shrinkage=0.05)
  boost_prob = predict(boost_fit, data.test, n.trees = 1000, type = "response")
  boost_pred = ifelse(boost_prob > 0.5, 1, 0)
  print(table(data.test$LeadStatus, boost_pred))
  boost_acc = sum(data.test$LeadStatus == boost_pred)/nrow(data.test)
  print(paste("Boost␣accuracy:␣", boost_acc))

  # Fit using Logistic Regression
  log_fit = glm(as.factor(LeadStatus) ~., data=data.train, family = binomial)
  log_prob = predict(log_fit, data.test, type="response")
  log_pred = ifelse(log_prob > 0.5, 1, 0)
```

```
  print(table(data.test$LeadStatus, log_pred))
  log_acc = sum(data.test$LeadStatus == log_pred)/nrow(data.test)
  print(paste("Logistic Regression accuracy: ", log_acc))

  # Plots the test accuracy against mtry for Random Forest
  num_pred = ncol(data.train)-1
  test_accuracy = 1:num_pred
  for(i in 1:num_pred){
    print(paste("iteration: ", i, "/ ", num_pred))
    rf_fit = randomForest(as.factor(LeadStatus) ~., data=data.train, mtry = i, ntree=1000)
    rf_prob = predict(rf_fit, data.test, ntree=1000, type="prob")
    rf_prob = rf_prob[,2]
    rf_pred = ifelse(rf_prob > 0.5, 1, 0)
    rf_acc = sum(data.test$LeadStatus == rf_pred)/nrow(data.test)
    print(paste("rf accuracy: ", rf_acc))
    test_accuracy[i] = rf_acc
  }
  plot(1:num_pred, test_accuracy, type="b", xlab="mtry", ylab="Test Accuracy")

  # Fit using Random Forest
  rf_fit = randomForest(as.factor(LeadStatus) ~., data=data.train, ntree=1000, max.depth=5,
                min.node.size=10, splitrule=Gini, replace=TRUE, sampsize=nrow(data.train),
                importance=TRUE, mtry=7)
  rf_prob = predict(rf_fit, data.test, ntree=1000, type="prob")
  rf_prob = rf_prob[,2]
  rf_pred = ifelse(rf_prob > 0.5, 1, 0)
  rf_acc = sum(data.test$LeadStatus == rf_pred)/nrow(data.test)
  print(paste("Random Forest accuracy: ", rf_acc))

  # Boosting Final Fit on study E
  fin_boost_fit = gbm(LeadStatus ~., data=df, cv.folds=10, bag.fraction = 0.75,
                distribution="bernoulli", n.trees=1000, shrinkage=0.05)
  fin_boost_prob = predict(fin_boost_fit, df_E, n.trees = 1000, type = "response")
  write.csv(fin_boost_prob, file = paste(DATA_PATH, "/Part 4/boost_result.csv", sep=""),
                row.names=FALSE)

  # Random Forest Final Fit on study E
  fin_rf_fit = randomForest(as.factor(LeadStatus) ~., data=df, ntree=1000, max.depth=5,
                min.node.size=10, splitrule=Gini, replace=TRUE, sampsize=nrow(data.train),
                importance=TRUE)
  fin_rf_prob = predict(fin_rf_fit, df_E, ntree=1000, type="prob")
  fin_rf_prob = fin_rf_prob[,2]
  write.csv(fin_rf_prob, file = paste(DATA_PATH, "/Part 4/rf_result.csv", sep=""),
                row.names=FALSE)
}
```

References

[1] Bidirectional Recurrent Neural Network. *https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks*, *author = Wikipedia*.

[2] Loukas,S. LSTM Time-Series Forecasting: Predicting Stock Prices Using An LSTM Model. *https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f*.

[3] Wikipedia. Positive and Negative Syndrome Scale. *https://en.wikipedia.org/wiki/Positive_and_Negative_Syndrome_Scale*.