Arnav Bhakta
CSC630A
Dr. Zufelt
October 14, 2021

CSC630A: Midterm Reflection

**** Important Notes: I have included all updated/all work on my GitHub, which can be found here.**

Organization (3pts)

- Is your work toward this learning objective readable? (For example, are the cells of your notebook in the same order as the intended kernel order? Is your writing polished and self-edited?)
- Are explanations of code, visualizations, and other artifacts of work clearly and consistently associated with their artifact?
- Is your work polished?

Volume of work (3pts)

- Do you have work done in multiple formats toward this learning objective?
- How much work has been put towards each learning objective? *(Dr. Z might rephrase as, "Have you done what should be considered sufficient work toward this LO, given that this is a 600-level class?")*
- Have you incorporated aspects of this learning objective into many projects/assignments?
- Have you learned about multiple aspects/viewpoints about the LO's topics?

Analysis/Documentation (3pts)

- Do you present and substantiate compelling arguments toward this LO?
- Does your code documentation assist in providing *clarity* in your work toward your LO?
- Do your writing and your code documentation provide a *complete* explanation of your work toward this LO?

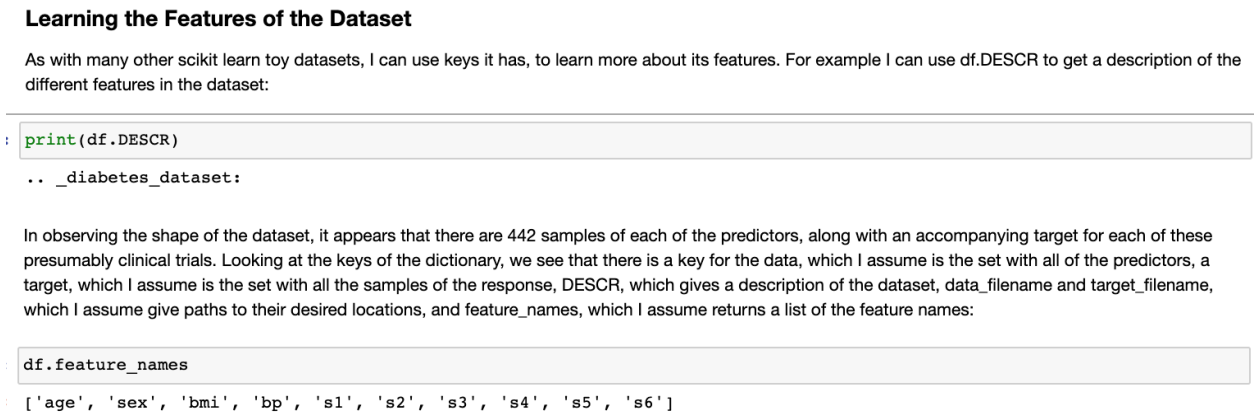
Progress (3pts)

- Do you have evidence that your skill level and understanding of this LO has improved?
- Have you leveraged *(and cited)* resources to expand your knowledge of this LO? Have you asked questions of your teachers/peers to do so?
- Have you looked at credible sources from both in and outside of class about this LO's topics?

Learning Objective 1: I can use the tools of the PyData stack to understand, interpret, and visualize datasets, including making arguments about its underlying distributions.

1. Organization: 3/3
2. Volume of Work: 3/3
3. Analysis/Documentation: 3/3
4. Progress: 3/3

In exploring the PyData stack, I have been very intentional in making my code very readable. For example, in working to understand sci-kit learn's diabetes dataset (my work can be found on my GitHub here), which I had previously never worked with and had very little intuition about, I made sure to make my thinking and work very clear and laid out in a methodological sequence of cells, such that if a complete stranger were to run the cells, they would have no issues doing so, and the code would execute flawlessly. Moreover, I included extra explanations in markdown cells in between each of the cells so that once again, if a complete stranger were to look at the notebook, they would easily be able to understand it:



Learning the Features of the Dataset

As with many other scikit learn toy datasets, I can use keys it has, to learn more about its features. For example I can use df.DESCR to get a description of the different features in the dataset:

```
print(df.DESCR)
```

```
.. _diabetes_dataset:
```

In observing the shape of the dataset, it appears that there are 442 samples of each of the predictors, along with an accompanying target for each of these presumably clinical trials. Looking at the keys of the dictionary, we see that there is a key for the data, which I assume is the set with all of the predictors, a target, which I assume is the set with all the samples of the response, DESCR, which gives a description of the dataset, data_filename and target_filename, which I assume give paths to their desired locations, and feature_names, which I assume returns a list of the feature names:

```
df.feature_names
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In looking at all of the explanations, it is evident that they have been edited to be clear and concise descriptions of what I am doing in the notebook.

Furthermore, when laying out the code in cells, explanations in markdown cells, and data visualizations, I made sure to place explanations strategically before the actual code, so that the reader wouldn't have to go into with a blind understanding, and follow it with any accompanying figures or tables that allowed one to get more insight into what exactly the code was doing:

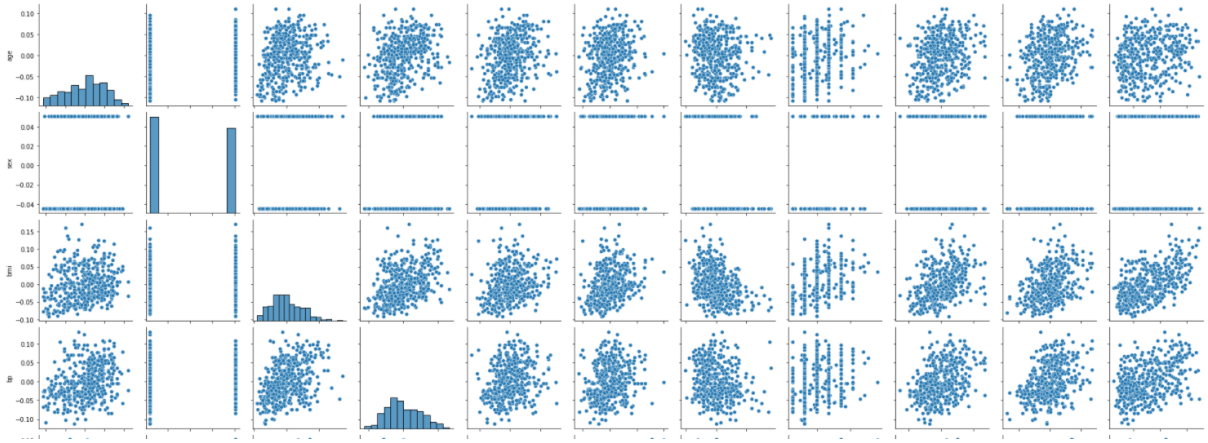
Graphing the Features

In this section, I will be using different plotting libraries to create histograms, scatterplots, heatmaps, etc. to get more statistical information about the relationship each of the features.

We begin by using sns's pairplot to create a figure that displays scatter plots between and histograms for all of the features.

```
]: sns.pairplot(df_pd)
```

```
]: <seaborn.axisgrid.PairGrid at 0x7fc2b97a0730>
```



As seen in the above figure, it appears that the sex feature is largely ordinal in nature, presumabely having the values -0.044642 and 0.050680 to represent either male or female. As seen from a histogram of its distribution, in the set, there are a roughly equal number of participants of each sex. Another variable with an interesting distribution is the s4 feature, or the total cholesterol feature. As opposed to having a more normal distribution, there appear to be 4-5 main total cholesterol measures that patients can have, with the majority of patients having near 0 measures of cholesterol.

Other than these two features, the rest seem to follow a more normal distribution, and for the most part, cluster together in a somewhat linear pattern, when scattered with the DPROG feature. Nonetheless, we can learn more about the each feature's relationship with each other, by creating a heatmap of correlations:

```
corr = df_pd.corr()
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(8,6)
plt.show()
```

As a result, I think it would be fair to say that in using the tools of the PyData stack, my work was polished, readable, and easily interpretable.

I am under the impression that in exploring the PyData stack, I have done a great deal of work towards the learning objective. Not only have I explored understanding, interpreting, and visualizing data sets in learning about sci-kit learn's diabetes dataset, but I have done so in

several other works I have done in this class, including but not limited to working with the Boston Housing Dataset (my work for which can be found here) and predicting Myers Briggs personality types (my work for which can be found here). In working with the Boston Housing Dataset, I once again began with a very minimal understanding of what exactly it was, its background, and what all of the features represented. As a result, I first began by trying to understand more background of the dataset, by using the keys of the dataset:

It appears that there are 13 features in the dataset, all of which are described below

```
print(df.DESCR)
```

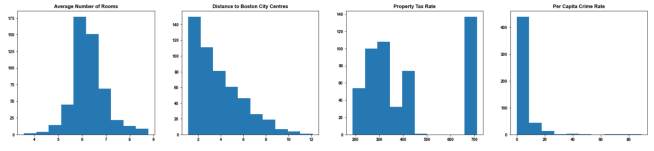
```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

Additionally, I worked to gain a deeper understanding of each of the feature's distributions, by creating histograms and scatter plots, to see if there were any underlying factors that affected the features, that I had not yet considered:

Graphing Histograms

1. The first histogram displays the average number of rooms per property, which follows an normal distribution. Suprisingly, the histogram peaks at roughly 6 rooms per property, which is quite a lot. Maybe a lot of these properties are mutli-family homes?
2. The second histogram displays how close each property is to a Boston City Employement Centre. THe Large majority of them are within 2 miles, indicating that a lot of the properties are near the city's center.
3. The third histogram displays the full-value property tax per 10,000 dollars. As can be observed, a lot of the properties have taxes within the 200-450 per 10,000 dollars range, with there being a larger secondary spike at around 700 per 10,000 dollars, which might be the subset of properties that cost more or are in prime locations in the city.
4. The fourth and final histogram shows the per capita crime rate per town, and as can be seen, the majority of the samples have very low crime rates in their area

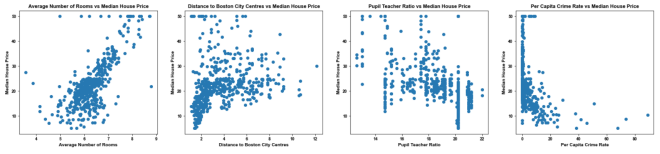
```
from matplotlib.pyplot import figure

plt.figure(num=None, figsize=(25,5))
plt.subplots_adjust(hspace=0.27, wspace=0.2)

font = {'weight' : 'bold', 'size': 9, 'family':'Arial'}
plt.rc('font', **font)


plt.subplot(1, 4, 1)
plt.hist(df_data['RM'])
plt.title('Average Number of Rooms', fontweight='bold', family='Arial', fontsize = 11)

plt.subplot(1, 4, 2)
plt.hist(df_data['DIS'])
plt.title('Distance to Boston City Centres', fontweight='bold', family='Arial', fontsize = 11)

plt.subplot(1, 4, 3)
plt.hist(df_data['TAX'])
plt.title('Property Tax Rate', fontweight='bold', family='Arial', fontsize = 11)

plt.subplot(1, 4, 4)
plt.hist(df_data['CRIM'])
plt.title('Per Capita Crime Rate', fontweight='bold', family='Arial', fontsize = 11)

plt.show()
```



Scatter Plots Between the Median House Price and Numerical Features Features

As can be seen from the below scatter plots, as few features have a stronger relationship with the Median House Price, namely the Average Number of Rooms, Distance to Boston City Centres, Per Capita Crime Rate, Percent Lower Status of the Town, and the Number of Properties Built Before 1940. I would assume that this is due to the fact that all of these features are related to the town the property is in, its commutability to different locations in the city, and the general size and age of the house, which all greatly affect the price at which a house is listed at.

```
plt.figure(num=None, figsize=(25,5))
plt.subplots_adjust(hspace=0.27, wspace=0.2)

plt.subplot(1, 4, 1)
plt.scatter(df_data['RM'], df_data['MEDV'])
plt.title('Average Number of Rooms vs Median House Price', fontweight='bold', family='Arial', fontsize = 11)
plt.xlabel('Average Number of Rooms', fontweight='bold', family='Arial', fontsize = 10)
plt.ylabel('Median House Price', fontweight='bold', family='Arial', fontsize = 10)

plt.subplot(1, 4, 2)
plt.scatter(df_data['DIS'], df_data['MEDV'])
plt.title('Distance to Boston City Centres vs Median House Price', fontweight='bold', family='Arial', fontsize = 11)
plt.xlabel('Distance to Boston City Centres', fontweight='bold', family='Arial', fontsize = 10)
plt.ylabel('Median House Price', fontweight='bold', family='Arial', fontsize = 10)

plt.subplot(1, 4, 3)
plt.scatter(df_data['PTRATIO'], df_data['MEDV'])
plt.title('Pupil Teacher Ratio vs Median House Price', fontweight='bold', family='Arial', fontsize = 11)
plt.xlabel('Pupil Teacher Ratio', fontweight='bold', family='Arial', fontsize = 10)
plt.ylabel('Median House Price', fontweight='bold', family='Arial', fontsize = 10)

plt.subplot(1, 4, 4)
plt.scatter(df_data['CRIM'], df_data['MEDV'])
plt.title('Per Capita Crime Rate vs Median House Price', fontweight='bold', family='Arial', fontsize = 11)
plt.xlabel('Per Capita Crime Rate', fontweight='bold', family='Arial', fontsize = 10)
plt.ylabel('Median House Price', fontweight='bold', family='Arial', fontsize = 10)

plt.show()
```



Using the information I gained from these scatter plots and histograms, I worked to learn more about how each of the features were behaving and tried to create a basis for the features might be behaving the way they are and have the distribution they do, and then went about modifying the

dataset to account for these findings but removing features that I perceived as being unnecessary or that may skew the predictions of a model trained on this data, based on my interpretations and understandings of the features:

Slicing the Data Vertically

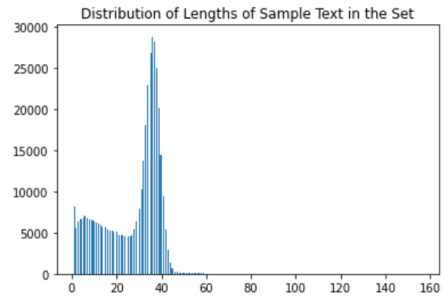
Based on the above scatter plots of all of the features vs the median house price, the features that I have decided the keep in the dataset, while removing the rest, are the Average Number of Rooms, Distance to Boston City Centres, Per Capita Crime Rate, Percent Lower Status of the Town, and the Number of Properties Built Before 1940, because these were the features that had the strongest relationship with the median house price. Thus, since we are trying to predict the median house price using the data we have, I believe that by dropping the remaining features, we will clear the dataset of features that are not as directly related to the median house price as the features kept in the dataset, and remove any loosely related data from the dataset, that the model might have latched on to otherwise and potentially skewed the accuracy of the median house price prediction.

```
df_data_filtered = df_data.iloc[:, 0:13]
df_data_filtered = df_data_filtered.drop(['ZN', 'INDUS', 'CHAS', 'NOX', 'RAD', 'TAX', 'PTRATIO', 'B'], axis=1)
df_data_filtered
```

	CRIM	RM	AGE	DIS	LSTAT
0	0.00632	6.575	65.2	4.0900	4.98
1	0.02731	6.421	78.9	4.9671	9.14
2	0.02729	7.185	61.1	4.9671	4.03
3	0.03237	6.998	45.8	6.0622	2.94
4	0.06905	7.147	54.2	6.0622	5.33
...	...	...	...	...	...
501	0.06263	6.593	69.1	2.4786	9.67
502	0.04527	6.120	76.7	2.2875	9.08
503	0.06076	6.976	91.0	2.1675	5.64
504	0.10959	6.794	89.3	2.3889	6.48
505	0.04741	6.030	80.8	2.5050	7.88

For my work with predicting Myers Briggs personality types, I employed similar strategies, except since I already had a very good understanding of the dataset, I instead applied this understanding to more in-depth applications, to optimize the data for training a model. For instance, when looking at the lengths of different sample texts that I would be used to train the model on, I saw that there was a clear spike where the majority of the texts' lengths fell in, and appropriately padded the samples to account for this:

```
lens = []
for i in df_user_flattened:
  sample_lens = i.split(" ")
  lens.append(len(sample_lens))

_ = plt.hist(lens, bins='auto')
plt.title("Distribution of Lengths of Sample Text in the Set")
plt.show()
```



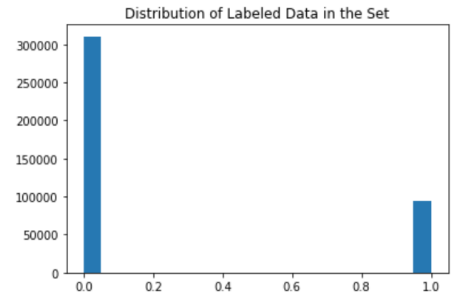
In observing the the distribution of the lengths of samples in the dataset, it can be seen that roughly 7.5% have a length close to 38 words. Given that the average tweet is roughly 55 words in length, we can assume that the majority of texts are very short in length and that when padded to a length of 150 words, the sequences will constist largely of 0s. Additionally, in looking at the distribution itself, we can see that it follows a roughly normal distribution, which reminds me of the Central Limit Theorem, and the fact that as we have more samples, as is the case in the current study, the distribution will be more likely to approach an average or most common frequency.

Furthermore, I investigated how the labels of the text were distributed by constructing a histogram and reasoned how to approach the data pre-processing based on these distributions:

Exploring the Data Distributions

Seeing as we have two specific features, it is important to understand more about their distributions by creating histograms, so that we can learn more about the features themselves. For the labels feature, we will primarily be looking at the split between samples labeled for Extroversion versus samples labeled for Introversion, allowing us to understand which is more prevalent in the current dataset. For the text feature, stored in the `NumPy` array `df_user_flattened`, we will be looking at the distribution of the lengths of each of the samples, in order to understand how much each sample will be padded on average, when padded or truncated to 150 words.

```
_ = plt.hist(df_label_flattened, bins='auto')
plt.title("Distribution of Labeled Data in the Set")
plt.show()
```



As can be seen from the above histogram, the large majority of labels in the dataset are for Extroversion (0) as opposed to Introversion (1) (roughly a 3 to 1 split). Though it is hard to decieve why this may be, a guess as to why, may be that as the text samples were taken from an online forum, where people are more inclined to be conversational, the majority of samples may have consequently exuded extroversive qualities. In discussing the implications of such an uneven split in the dataset, it could be argued that the set does not provide the model with enough introversive samples to truly be able to pick up on what characteristics of the text make in introversive in nature. However, it is also important to note that there are roughly 100k samples for texts labeled as being introversive, which should theoretically be enough for our model to learn what makes a text introvesive. Hence, we shall not alter the distributions of labels in the dataset.

In working through all of these projects, I worked toward properly documenting and explaining my code and decisions. As seen in all of the above screenshots, every piece of code I wrote has an accompanying explanation that dives deep into my decision process when coding. Additionally, in looking for missing data in the Boston Housing Dataset, which is a method that I had previously never used, I used a new methodology by writing the code – df_data.isnull().sum() – and followed it up with an analysis of what I had done: “Upon looking for null or N/A values in the dataset, it appears that none of the features have any such values and that all of their datums are of the type float, meaning that there is no missing data in the dataset … Looking at all of the types of the variables, it appears that they are all of the type floats, except for the CHAS variable that is instead an ordinal value for if the tract bounds the river (1) or not (0)”.

In considering the progress I have made on this learning objective, I would characterize it as being monumental. Prior to this class, I had very minimal experience working with NumPy, Pandas, Matplotlib, or Seaborn as I hadn’t done consistent data analysis of anysorts from before my Lower Year. Hence, given that I was able to work through the entire PyData stack in both the lab and other projects, I would say that I have improved immensely, as, as displayed in the above screenshots, I was able to take several steps necessary to learn about and characterize datasets, which I would not have been able to do before. In looking at resources I have used, I have consistently asked friends with more experience for help (William Yue, Michael Huang, Nathan Xiong, etc.), when I don’t understand something and refer to external sites and documentation when they cannot help me. For example, at the start of this class, I read through nearly the whole

w3schools python documentation and in working on my gradients project, I have accessed helpful resource whenever I don't understand something:

Understanding NumPy

- https://pynative.com/python-random-randrange/#:~:text=Use%20randrnage()%20to%20generate,4%2C%206%2C%208
- https://numpy.org/doc/stable/reference/generated/numpy.arange.html
- https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html

Understanding Dictionaries

- https://stackoverflow.com/questions/16819222/how-to-return-dictionary-keys-as-a-list-in-python

In addition, I have also been able to improve my understanding of analyzing data as a whole, by learning about important and necessary methods when learning more about your data, such as the central limit theorem.

As for what I think I can improve on moving forwards, I think something interesting would be to explore more methods by which to represent and explore the distributions of data. Thus far, the although I've used a large amount of methods to explore relationships between features and their distributions include histograms, scatter plots, correlation heatmaps, pairplots, and autoregression plots, I think it would be both fun and beneficial for me to explore more methods by which to visualize the data and argue for its underlying distributions, such as granger causality matrices, clustering techniques, similarity matrices, augmented dickey-fuller tests, and probability plots. Another exciting idea I have, although it is much more doubtful that it will happen, is curating my own dataset, and seeing how my understanding of the PyData stack affects the steps I take in deciding what features I include in the set and the main underlying distributions I try to create in the dataset.

Learning Objective 2: I can implement and describe the use of all aspects of the data modeling process.

1. Organization: 3/3
2. Volume of Work: 3/3
3. Analysis/Documentation: 3/3
4. Progress: 3/3

Similar to my work with the PyData stack, when carrying out all aspects of the data modeling process, I make sure to keep my code as organized, condensed, and understandable as possible. A clear example of this, is in working on the gradients project. For one, via this project, I have been able to learn a lot more about the data modeling process, that I had previously not known about, including how to model multidimensional functions using computational graphs and how one can take an input, and evaluate it with simple arithmetic or fit it to a more complex application, such as taking a gradient (my work can be found here). Furthermore, throughout all

of this, I have consistently kept my work very organized, by commenting my code such that even if there isn't very much or it isn't very hard to understand for someone with a lot coding experience, the average person can look at the code and immediately understand what is going on:

```
# getting the position of the current variable in our dictionary values
sort = sorted(list(values.keys))
pos = int(sort.index(self.name))

# creates the gradient which is an array of 0s, except for at the index of the current variable, at which it is 1
```

```
'''
I have chosen 7 as the number to define a handful of randomly generated
points, but any number can be chosen
'''

num_new_grad = 7

for i in range(num_new_grad):
    '''
    seeing as the x and y bounds of the above plot are (-8, 6) and
    (-8, 6), respectively, I have defined the x and y coordinates of
    our randomly generated test points to be integers within this range.
    '''

    x_random = np.random.randint(-8, 6)
    y_random = np.random.randint(-8, 6)
```

Additionally, with my work on Linear and Logistic regression, I was able to get a more robust understanding of gradient descent and different cost/loss functions, such as the cross-entropy function, which I worked on for homework, and residual sum of squares, how they functioned (please refer to my Canvas assignment on Linear and Logistic regression) (Please also refer to my screenshot on page 16, where I discuss bi-directional cross entropy loss functions more in-depth).

Outside of just class assignments meant to help me learn about data modelling, I have feverishly taken every opportunity possible to expand the breadth of my knowledge in the area. An example that sticks out to me is once again, my work with sci-kit learn's diabetes dataset (my work can be found on my GitHub here). As opposed to simply stopping with learning about the distribution of data and the PyData aspect of the assignment, I went ahead and researched and learned more about LSTMs and RNNs, by referencing several useful and reliable articles, such as:

- https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
- https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
- https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

and implemented them into my notebook. New and useful concepts that I was able to learn about by doing so, included the stochastic gradient descent optimization algorithm that LSTMs use in training, how learning rates are implemented into neural networks and their effect on training (please also refer to my screenshot on page 16, where I discuss learning rate once again), the

vanishing gradient problem that RNNs often face, and the back-propogation algorithm that LSTMs use to overcome this issue. Throughout this whole process, I discussed what I had learned in markdown cells, as to make my code and thought process clear and understandable:

Forecasting With the Dataset

Provided the relationships between features that we discussed in the previous section, I will now be exploring how well the predictor variables will be able to forecast the response or the DPROG feature. Although in the desciption of the dataset, it appears that Efron et al. used least angle regression, I will instead be using a Recurrent Neural Network (RNN), in specific a Long Short-Term Memory (LSTM) architecture to learn more about the algorithm and how well it is able to perform.

In doing some background research into LSTMs, I was able to discover that they use a stochastic gradient descent optimization algorithm in training. By having a set learning rate or rate at which each weight changes during each epoch of training, they are able to change the model based on the error gradient provided when the estimated coefficients are updated. In specific, the weights are updated using a back-propogation algorithm, which also helps in preventing the vanishing gradient problem. The vanishing gradient problem is an issue that RNNs face, as when more layers are added to the neural network, the gradient of the loss function approaches zeros, causing a severe amount of inaccuracy in a model. By using the back-propogation algorithm and maintaing a cell state, LSTMs avoid this problem, and allow for more accurate forecasts and pattern recognition [1], [2].

Additionally, in researching cost/loss functions of LSTMs, I discovered that in training, you are able to specify which type of loss function you want your model to be trained on. As a result, in building my LSTM model, I chose to use an MSE loss function, due to its ability to penalize outliers in a response set that is normally distributed around a mean, as seen in the description of the dataset and quadratically penalize predictions, such that large errors are penalized far harsher than small errors [3], [4]. The algorithm that MSE uses is as such:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $(y_i - \hat{y}_i)^2$ is the square of the difference of the actual and predicted values and n is the number of non-missing data points.

To begin building out LSTM model, I import the Keras library from TensorFlow, StandardScaler from sklearn to standardize the data, and train_test_split from sklearn to create our training and testing splits:

Then, using what I had learned, I went ahead and actually implemented the LSTM to see how well it was able to predict the DPROG feature of the dataset, by working with machine learning libraries I was very unfamiliar with, such as tensorflow, and learned and carried out the necessary data pre-processing steps necessary when predicting with a neural network, such as standardizing the data with StandardScaler, and splitting the training and testing sets, using train_test_split:

```
25]: import tensorflow as tf
     from tensorflow.python.keras.models import Sequential
     from tensorflow.python.keras.layers import Dense, LSTM
     from sklearn.preprocessing import MinMaxScaler
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
```

I then split the dataset into our response (y) and predictor sets (x) and split each set into a training (70%) and testing (30%) subset based on the percent of the dataset they are attributed:

```
26]: y = df_pd.iloc[:, 10].values
     x = df_pd.iloc[:, 0:10].values
```

```
27]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

Looking at the shape of the x_train set, we see that we were able to split the dataset, such that the training set it 70% of the original dataset (309 samples):

```
28]: x_train.shape
```

```
28]: (309, 10)
```

Next, I use StandardScaler to "standardize [the predictor] features by removing the mean and scaling to unit variance" [3].

```
29]: sc = StandardScaler()
     x_train = sc.fit_transform(x_train)
     x_test = sc.fit_transform(x_test)
```

I then convert x_train, x_test, y_train, and y_test to numpy arrays, such that operation can be easily performed on them, and reshape x_train and x_test to 3D arrays, to fit the requirements of the model.

```
30]: x_train, x_test, y_train, y_test = np.array(x_train), np.array(x_test), np.array(y_train), np.array(y_test)

     x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
     x_test = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Once again, along this whole process, I was including pieces of information that made what my code was doing easy to understand, in markdown cells.

Finally, I built the model itself, researching and discussing how to implement an LSTM:

Finally, I begin building the LSTM model itself. I start by initialising an RNN using Sequential() and then adding 4 LSTM layers with 50 neurons each, followed by an output layer that utilizes Dense for a full connection layer. Furthermore, in between each LSTM layer, I add a dropout layer using Dropout, to prevent the model from overfitting, by randomly setting neurons in the hidden layer to 0 in between each phase of training

```
from tensorflow.python.keras.layers import Dropout

model = Sequential()

model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 50))
model.add(Dropout(0.2))

model.add(Dense(units = 1))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 10, 50)            10400
_________________________________________________________________
dropout_4 (Dropout)          (None, 10, 50)            0
_________________________________________________________________
lstm_5 (LSTM)                (None, 10, 50)            20200
_________________________________________________________________
dropout_5 (Dropout)          (None, 10, 50)            0
_________________________________________________________________
lstm_6 (LSTM)                (None, 10, 50)            20200
_________________________________________________________________
dropout_6 (Dropout)          (None, 10, 50)            0
_________________________________________________________________
lstm_7 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_7 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
```

And though my model did not perform optimally, I took the opportunity to reflect on what I could do better for next time and how I can improve:

Using the model we just trained, I then see how well the model performs in predicting the test set, using RMSE, the square root of MSE, as a metric to measure accuracy:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where $(y_i - \hat{y}_i)^2$ is once again the square of the difference of the actual and predicted values and n is the number of non-missing data points.

```
predictions = model.predict(x_test)
rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
```

```
print('RMSE value: ' + str(rmse))
```

```
RMSE value: 110.38732230436831
```

As can be seen from the relatively high RMSE score, we can assume that the LSTM model we contructed did not provide the most accurate forecasts. Though this may indicate that the predictor features themselves may not be adequate to forecast the DPROG feature or the LSTM models are not optimal for forecasting on datasets of this type, I belive that the innacuracy stems from my own choices in training the model. With more time and a better understanding of how LSTMs and RNNs function, I believe that I will be able to optimize training by adding/removing more layers, incorporating more algorithms to bolster LSTM, or taking further steps in pre-processing data. Nonetheless, I was able to fully explore the Diabetes Dataset, learn about its different features, and work with modeling the data through this process, and satisfy the initial goals I had when I began analyzing the dataset.

Another instance of when I took on the challenge of the data modelling process, was in my Myers Briggs type prediction project (my work for which can be found here). Though this project was much more challenging, I still approached it head on, eager to learn more. In fact, by carrying out the project, I believe that I was truly able to go through the whole data modelling process, as I first began by carrying out a literature review, and seeing what methods had previously been used in similar classification tasks:

LSTM Model for Predicting Myers Briggs Personality Types

By Arnav Bhakta[1] and William Yue[1]

[1] **Phillips Academy Andover**

The authors would like to thank Patrick Chen, Michael Huang, and Ali Cy for their helpful input and advice in crafting this notebook.

In the current notebook, we begin laying the seeds for an LSTM model that will be used to predict Myers Briggs Type Indicators (MBTIs) from an inputted text. As a precursor to this notebook and an introduction to MBTIs and the dataset that we utilize in the current study, please see my previous notebook on the data pre-processing steps we took, to ensure that the data is optimized for the model it is being passed into.

As the purpose of this notebook is to begin to expirement with different models to see which ones provide the most accurate predictions, before settling on a particular framework and constructing a more complex architecture. We start with LSTM, because Hernandez and Knight had previously compared SimpleRNNs, GRUs, and LSTMs, to see which of the specified models performed most optimally when predicting MBTIs [1]. In specific, for the specific subtype of personality types that we are beginning our MBTI predictions with (Introversion and Extroversion) their LSTM achieved a 54.0% accuracy in classification [1]. Though these results themselves are very subpar, as LSTMs outperformed all other models tested, we would rather favor it, over the other tested models. Moreover, in comparing LSTMs ability to properly classify short texts (IMBD reviews, Douban comments, PTT posts, etc.), as is done in the current study, Wang et al. found that LSTMs consistently outperformed Naïve Bayes (NB) and Extreme Learning Machines (ELMs) [2]. Hence, we will begin by using LSTMs in our classification task, due to their accuracy in previous studies.

By doing so, I was able to learn more about which type of models perform best for the problem I was trying to solve, and ultimately settle on a bidirectional LSTM, due to the accuracy that it often displays in similar classification tasks. I then went ahead and cleaned and prepared the data for modeling, by appropriately splitting the text samples to remove sensitive or unnecessary information, tokenizing the data, sequencing the data, and then finally padding it. Once again, I made sure to document, narrate, and illustrate what I was doing in a clear and concise manner, in markdown cells:

Loading in the Data

Below, we load in the data and read the csv from My Drive, using Pandas, and split up the columns of the dataset into two distinct features: the text and labels. We then cast them to NumPy array from Pandas DataFrames, for ease of use later on.

```
df = pd.read_csv('/content/gdrive/My Drive/Peitho/0.csv')
df_text = np.array(df['text'])
df_label = np.array(df['label'])
```

We then create two lists: `df_user` and `personality_types`. `df_user` will hold all of the text in the dataset, after sensitive or hard to understand strings are removed. `personality_types` contains all of the 16 possible personality types one can have, as in the sample text, there are multiple instances of these personality types. So, we go ahead and remove them, as to optimize the training of our model, and ensure that it fully bases its predictions off of more natural language, as the different personality types are not necessarily elements of everday speech

```
df_user = []
personality_types = ['intj', 'intp', 'entp', 'entj', 'infj', 'infp', 'enfj', 'enfp', 'istj', 'isfj', 'estj', 'esfj', 'istp',
'isfp', 'estp', 'esfp']
```

In addition to removing all mentions of the different personality types, we also replace all instances of links in the text using the `re.sub()` method, as once again, links do not arise in everyday speech. After removing all such instances of sensitive or hard to understand text from the samples, we for one fix any spacing issues that may have arisen during the removing of these specific types of texts, but also, split the text up into the text of the different users, by splitting the text at all instances of `'|||'`, which serve to indicate a break in the person who is saying the speech. In doing so, we are able to have an individual person's speech matched with their specific label or personality type.

```
for i in df_text:
  text = i.lower()
  text = re.sub(r'http\S+', '', text)
  text = re.sub(r'https\S+', '', text)
  text = re.sub(r'@[A-Za-z0-9]+','', text)
  for j in personality_types:
    text = re.sub(j, '', text)
  while True:
    before_text = text
    text=text.replace('  ', ' ')
    if before_text == text:
      break
  df_user.append(np.array(text.split('|||')))
```

Tokenization

As mentioned above, we tokenize the text, by splitting it up into smaller units, as to optimize our model's ability to analyze the text and discover patterns within it. However, prior to doing so, we first go ahead and flatten the array, by reducing its dimensionality to a 1-dimensional array, in order to be able to tokenize the text, as currently, our text is within a series of nested arrays. We do this, be defining a new list, `df_user_flattened`, and looping through each nested array in the `df_user` array, which currently holds all the texts, and assigning each string of text within these nested arrays as new elements in our new 1-dimensional `df_user_flattened` list.

```
df_user_flattened = []
```

```
for i in df_user:
  for j in i:
    df_user_flattened.append(j)
```

We then convert `df_user_flattened` to a NumPy array for ease of use, and display its values. As we can see, as opposed to `df_user` which consisted of several arrays nested within each other, `df_user_flattened` contains only 1 array, meaning that we successfuly reduced the dimensionality of our data.

```
df_user_flattened = np.array(df_user_flattened)
display(df_user_flattened)
```

```
array(["' and moments sportscenter not top ten plays pranks",
       'what has been the most life-changing experience in your life?',
       ' on repeat for most of today.', ...,
       'i have seen it, and i agree. i did actually think that the first time i watched the movie, and from the beginning (or
when they got their powers) i kinda thought andrew would never work right with...',
       "ok so i have just watched underworld 4 (awakening) and must say it was a really good film, compared to the other films
out in the last few months anyway. i don't think it was as good as the first 3...",
       "i would never want to turn off my emotions. sometimes i hide them from the world, but i still need them for me.'"],
      dtype='<U874')
```

Next we go ahead and tokenize our data. In the current study, we do so using `Tokenizer` from `Keras`. `Tokenizer` takes in a few parameters, when tokenizing the data, which are as follows: num_words returns the ids of the `n` most commonly used words in the dataset, where `n` is the `vocab_size` we defined as `vocab_size = 4000`, `oov_token` is used to replace out of vocabulary words.

```
vocab_size = 4000

oov_tok = "<OOV>"

tokenizer = Tokenizer(num_words = vocab_size
                      ,oov_token= oov_tok
                      )
tokenizer.fit_on_texts(df_user_flattened)
word_index = tokenizer.word_index
```

Post-tokenization, we are able to call `word_index` from `tokenizer`, to get the most commonly used words in our dataset, which are:

```
word_index
```

```
{'<OOV>': 1,
 'i': 2,
```

Texts to Sequences

In order to be able to train our model using the available texts, we then use the `texts_to_sequences` method to convert our text to a sequence of integers. This is done by using the most frequent words that we found above, and replacing these words that the tokenizer knows, we integers, such that the model that we will build is able to interpret the text.

```
tokenized = tokenizer.texts_to_sequences(df_user_flattened)
```

```
tokenized
```

Padding

The final step that we take, is padding each of the samples, to be the same length. As seen above, the max number of words in the samples of texts that we are provided with is 156 words, so we pad all of the sequences to a length of 150, to ensure that all of the sequences have the same length. This is done using the `pad_sequences` method, which takes the tokenized and sequenced text that we just defined, and pads all of the sequences or truncates them to a length of 150, by adding on 0s until the sequence has a length of 150, or removing integers from the sequence, until the sequence has a length of 150.

```
padded = pad_sequences(tokenized, maxlen=150, padding='post', truncating='post')
```

As seen below, our padding was successful, and leaves us with an array of 405263 sequences of length 150, to pass into our model, so that we can predict MBTIs.

```
padded.shape
```

```
(405263, 150)
```

I then went ahead and built the bidirectional LSTM model, but before doing so, took the opportunity to learn more about how they worked and understand more about what my code was

doing, by going over topics such as embedding layers, bidirectional LSTMs, dropout layers, dense layers, and binary cross-entropy loss functions, and their purpose, function, and applications in machine learning. I did this by referring to several useful articles and papers, such as:

- https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
- https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/
- https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf
- https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
- https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/
- https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
- https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

And talking when classmates who are more experienced in these subject areas, such as Michael Huang. Per usual, I talked about what my code was doing and what I learned in markdown cells, to document and analyze my code, but also make the whole flow of the notebook more understandable, such that you can just open it up, run it without error, and understand what is going on:

Bidirectional LSTM Model

In the following section, we will be discussing the model utilized in this study. As we are simply testing the effectiveness of different models in classifying sample texts on their MBTIs, we do not begin by constructing an intricate and complex architecture, but instead begin with a very baseline model.

We start building out model, by defining a `Sequential` model using `Tensorflow`'s `Keras` : a stack of layers in a neural network, where each layer has one input tensor and one output tensor. The layers of the model as as follows: an embedding layer, a bidirectional LSTM, a dropout layer, a bidirectional LSTM, a dropout layer, a dense layer, a dropout layer, and a final dense layer, as outlined in the model summary.

Embedding Layers

Embedding layers are utilized in neural networks to handle word embedding, or a process by which words are represented via vectors. As opposed to more traditional bag-of-word encoding, which produce very sparce vectors, embedding provides very dense vectors, where each vector represents a single word. As a result, an embedding layer is often used as the first hidden layer in neural networks, when dealing with text data [3].

As an input, the layer requires an integer representation of the words, which we have already done via sequencing the tokenized text samples. It then requires 3 arguments, being the `input_dim` , the `output_dim` , and the `input_length` . The `input_dim` refers to the vocabulary size, or the the number of words that are represented in the sequenced data, that we defined while tokenizing the sample texts. The `output_dim` refers to the size of the vectors space that the words we pass in, will be embedded into. More generally, it sets the size of the output vector for each word. In the current study, we begin by setting the `output_dim` to 16, however, with tuning and testing, this value is subject to change. The final argument that we specify is the `input_length` , which refers to the number of words in each input sequence. During padding, we padded and truncated all samples to a length of 150 words, so this is the value we specify for this argument [3].

Bidirectional LSTM

The following explanation requires a bit of familiarity with LSTMs. If you don't have that, that's okay! You can refer to my explanation of LSTMs in this notebook and come back here.

Bidirectional LSTMs can be viewed as a subtypes of LSTMs that have been improved for classification tasks. It does so by training two LSTMs in parallel on the input data, as opposed to the traditional one, by having one be trained on the forward input sequence and the other being trained on the backward input sequence. This is done by simply duplicating the the first layer of the LSTM in the network, such that it results in there being two layers side by side and have better access to all of the input information that is provided [4], [5]. In doing so, it is able to split the state of the neurons, to get a more whole representation of the sample texts, as opposed to simply having a linear interpretation [4], [5]. From a more wholistic view, this can be explained as not being able to understand a sequence of words, until you know what their future context is, emphasizing the need to have a betterunderstanding of the "end" of the sequence [6].

In this notebook, we utilize 2 bidirectional LSTMs as a baseline to test their effectiveness for our specific classification task. Each layer consists of 16 memory units, with the first hidden layer having a fully connected layer, that utilizes a sigmoid activation function to provide an adequate output.

Dropout Layer

Neural networks are often at risk of overfitting when training, which can have several adverse effects, including becoming accustomed to the noise of the training set, and consequently having low accuracy when predicting on new datasets. Dropout layers seek to prevent overfitting, by randomly ignoring a specificied number of layer outputs (in our case 10%), and allowing the layer to be viewed as a new layer, due to the differing number of nodes and changing connectivity to the input layer. As a result, each layer is seen as being different, meaning that there is a lesser amount of reliance on noise in the training set, when predicting the output vectors, and that there is the opportunity to correct any wrongdoings in previous layers. Thus, dropout layers allow us to place more trust in the fact that our model will be accurate and not overfit [7].

Due to its relaiability, we place dropout layers in between each of the layers we use in this model.

Dense Layer

Dense layers are deeply connected layers in a neural network, that recieves input from the neurons of previous layers, and uses matrix-vector multiplication to transform the vectors into a single vector of a specified size. This specific value is specified via the `units` parameter. As our model consists of two dense layers, the first vector outputted has a size of 8, while the second has a size of 1. In our model, we also specify activation functions for each dense layer, via the `activation` parameter. Activation functions are often used to create a non-linear relationship in the input neuron, such that the model can learn more complex relationships between input and output vectors [8]. The activation function we use for the first dense layer is a rectified linear unit (ReLU), which is a piecewise linear function, that outputs the input directly if the input is positive, or zero otherwise. In doing so, it allows model to learn faster and perform better. A sigmoid activation function is simply a logistic function, that that transforms the input into a value between 0 and 1 [9].

Learning Rate

The learning rate is a tuning paramater that specifies the step size at which weights are updated during training, while the loss function as it approaches its minimum. By doing so, it is responsible for how the model changes as a result of the estimated error it produces, each time its weights update, meaning that it is also in charge of how quickly the model adapts to the presented task. A large learning rate can cause the model to converge very quickly in the direction of an incorrect prediction, so for our model, we set a learning rate of 0.0001, so that the model spends enough time understanding trends in the input sample texts and updates accordingly [10].

Loss Function

A loss function can be viewed as providing a metric for how well your model fits the training set. More specifically, as we are tasked with a binary classification problem, the loss function we utilize is binary cross-entropy:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

where y is the label for the corresponding sample text, and $p(y)$ is the probability of the label being 1, for all of the labels in the set. More specifically, for each label of value 1, it adds the log probability of it being 1, $\log(p(y))$ to the loss, and adds the log probablity of the label being 0, $\log(1 - p(y))$, for each label of value 0. In doing so it is able to provide a justified predicted probability of the classifier being right, while also penalizing the model for inaccurate predictions [11].

I then went ahead and trained the model, while also discussing its training:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(4000, 16, input_length=150),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16, return_sequences=True)),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16)),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

learning_rate = 0.0001
model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate), metrics=['accuracy', 'ms
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 150, 16)           64000
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 150, 32)           4224
_________________________________________________________________
dropout_15 (Dropout)         (None, 150, 32)           0
_________________________________________________________________
bidirectional_11 (Bidirectio (None, 32)                6272
_________________________________________________________________
dropout_16 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 264
_________________________________________________________________
dropout_17 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 9
=================================================================
Total params: 74,769
Trainable params: 74,769
Non-trainable params: 0
_________________________________________________________________
```

After creating the model, we then train the model, by fitting it to the padded training texts and its corresponding labels. I train the model on 5 epochs – nowhere near to final number of epochs I would train the model on, due to the low learning rate – in order to get a general understanding of how well the bidirectional LSTM performs when predicting the MBTI types. Training for the model took roughly 6-7 hours.

```
model.fit(x=padded, y=np.asarray(ds_train_labels).astype('float32'), batch_size=8, epochs=5, validation_data=(padded_
```

```
Epoch 1/5
```

Finally, I discussed my results, which actually performed better than models in previously published papers (77% accuracy), by evaluating the model and reflecting on errors or false premises for which I was basing the models accuracy on. I also went ahead and analyzed this false premise, which I assumed to be that the model was just outputting 0 labels, by looking at the “positive” labels that the model outputted, and a distribution of the model’s outputs:

Results

The final step we take, is seeing how well the model performs when predicting on the testing set. We do this by using the `model.predict` function, and having it predicting on the padding sample texts, and comparing it to the actual labels for the corresponding texts. We can then compute metrics for the accuracy of the predictions, using the `model.evaluate` function.

In doing so, we see that in comparison to Hernandez and Knight's model, which recieved an accuracy of 54.0%, our baseline model greatly outperforms theirs with an accuracy of 76.504%. Though this result may seem to be very good compared to those of previous studies, something that seems off, is the fact that in our dataset we know that roughly 25% of the sample texts have a label of 1, while the remaining ~75% have a label of 0. This raises some suspicions due to the fact that we achieved a nearly 75% accuracy, which might implicate that our model might not be as accurate as we had initially assumed, and that it is just predicting 0s.

```
predictions = model.predict(padded_test)
```

```
print(np.mean(predictions))
```

```
0.22614066
```

```
labels_test = np.asarray(ds_test_labels).astype('float32')
```

```
model.evaluate(predictions, labels_test)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 150) for input KerasTensor(type_spec=TensorSpec(shape=(None, 150), dtype=tf.float32, name='embedding_5_input'), name='embedding_5_input', description="created by layer 'embedding_5_input'"), but it was called on an input with incompatible shape (None, 1).
2533/2533 [==============================] - 7s 2ms/step - loss: 0.5626 - accuracy: 0.7651 - mse: 0.1869 - mae: 0.4043 - mape: 244507056.0000

[0.5625528693199158,
 0.7651042938232422,
 0.18689122796058655,
 0.40433382987976074,
 244507056.0]
```

```
# check number of correct predictions
count = 0.0
correct = 0.0
for i in range(len(predictions)):
    predict = 0
    if predictions[i] < 0.5:
        predict = 0
    else:
        predict = 1
    if predict == labels_test[i]:
        correct += 1.0

    count += 1.0

print('test accuracy: ' + str(correct/count*100) +'%')
```

```
test accuracy: 76.50426264296202%
```

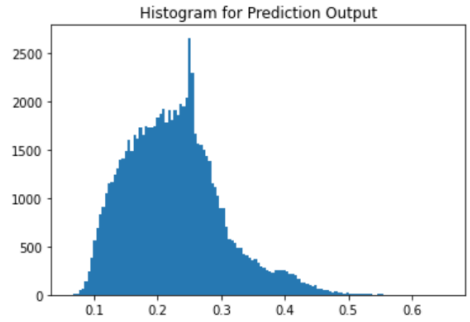
To test this hypothesis, we look at how many predictions the model actually outputted were 1s, and of these, how many were correct. As the model doesn't necessarily output integer label predictions 1 and 0, we take all predicted labels ≥ 0.5 to be a predicted label of 1, and predicted labels < 0.5 to be a predicted label of 0. In this manner, we discover that the model only outputted 125 positive label predictions, in comparison to the nearly 8000 predictions it made, of which only 60 of these predictions were correct. Accordingly, it is safe to say that the accuracy of our model was based on a false premise, and that more fine tuning to the model is necessary.

```
num_pos = 0
index_pos = []
index_correct_pos = []
for i in range(len(predictions)):
    if predictions[i] >= 0.5:
        num_pos += 1
        index_pos.append(i)
        if labels_test[i] == 1.0:
            index_correct_pos.append(i)

print('number of positive predictions: ' + str(num_pos))
print('\nindices of positive predictions:')
print(index_pos)
print('\nindices of correct positive predictions:')
print(index_correct_pos)
```

```
number of positive predictions: 125

indices of positive predictions:
[524, 698, 986, 1367, 2766, 3834, 5183, 5817, 5920, 7801, 7957, 9722, 10273, 10472, 12069, 14150, 15378, 16681, 1779
5, 17797, 17868, 18603, 19804, 20534, 20835, 21738, 21767, 22083, 22650, 22819, 22965, 23655, 25123, 28069, 28670, 28
731, 29747, 30038, 31044, 31954, 32388, 32669, 33716, 33966, 33976, 36077, 36294, 36935, 37216, 38093, 38231, 38922,
39228, 39645, 39902, 40392, 41708, 41935, 43202, 45416, 46262, 47024, 47206, 47322, 47798, 48050, 49093, 49362, 4958
3, 50414, 50517, 51056, 51280, 51866, 52005, 52233, 53121, 54471, 54909, 55677, 56169, 56598, 57158, 57345, 57727, 58
106, 58385, 58782, 59439, 59661, 59838, 59986, 60124, 60479, 61242, 61372, 62517, 63030, 64617, 65396, 66385, 66551,
66896, 68632, 69251, 69961, 70813, 71743, 72373, 72896, 73658, 73665, 74598, 74652, 74836, 75253, 77236, 78181, 7884
1, 78908, 78925, 79375, 79446, 80285, 80304]

indices of correct positive predictions:
[524, 986, 2766, 3834, 5817, 7801, 10273, 10472, 12069, 15378, 16681, 20835, 21738, 22819, 23655, 28069, 28670, 2873
1, 29747, 32669, 33966, 33976, 36077, 36935, 37216, 38922, 39902, 41935, 43202, 46262, 47024, 47322, 47798, 48050, 49
583, 50414, 50517, 51056, 52005, 54909, 55677, 56169, 56598, 57158, 58385, 59838, 61242, 61372, 65396, 66385, 66551,
69961, 71743, 72373, 72896, 73658, 74652, 75253, 78181, 80285]
```

Looking more in-depth at the distribution of the predicted label values via a histogram, we see nearly all of the predictions are < 0.5. In fact, the majority of the predicred labels recieved values < 0.3, indicating that the model had a tendency to predict high values of 0 labels.

```
_ = plt.hist(predictions, bins='auto')
plt.title("Histogram for Prediction Output")
plt.show()
```



I then wrapped up the notebook by analyzing where I went wrong and how I could improve:

In seeing these results, it is clear a more complex architecture is required to handle this classificaton task. Moreover, it suggests that bidirectional LSTMs may not be the best approach for handing this problem. Instead, we plan on testing the effectiveness of convolutional neural networks (CNNs), which have historically been know to perform optimally for classification tasks. We are considering the possibility of taking a combined approach using eXtreme gradient boosting (XGBoost) and CNNs, to pretrain the data via a decision tree-based method, before passing the outputs as inputs to a model consistent of CNNs. A final model we may consider in the future, is the use of transformers, due to their prevalence in handling textually based data.

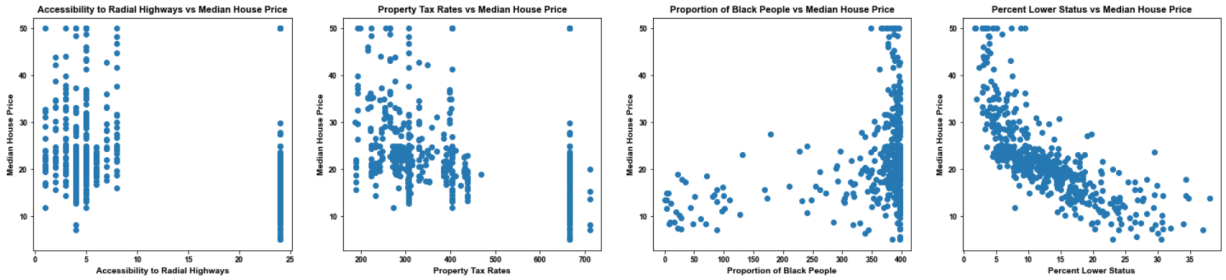
By discussing all of these examples of how I explored the data modeling process, I would say that without a doubt in my mind, I have made a great amount of progress. Prior to carrying out these projects, I had very minimal experiencing with data modeling, but by referring to outside resources, as well as my classmates, as outlined above, I was able to gain a greater intuition into cornerstones of data modeling, such as back propagation, computational graphs, Tensorflow, cross-entropy loss functions, neural networks, LSTMs, RNNs, activation functions, and more generally, applications of machine learning, which I had previously been very naive about.

Reflecting on how I can grow and learn more as the term progresses, I am excited to learn more about different neural networks, deep learning in specific, and hope to learn more about computer vision, as it is a subset of machine learning that I am very interested in, due to its ethical implications.

Learning Objective 3: I can use ethical reasoning to empower my data decisions, ensuring that the technical work that I do promotes equity and justice.

1. Organization: 3/3
2. Volume of Work: 2/3
3. Analysis/Documentation: 3/3
4. Progress: 2.6/3

At this point in the course, ethical reasoning is the area that I feel I've done the least amount of work in. Nonetheless, I have still been able to learn a lot about how I should integrate ethical reasoning into my decision making process, when dealing with data and technical work. One prominent example of this is my work with the Boston Housing Dataset. Even before we were asked to write a reflection on it, I was beginning to question why exactly certain racially motivated features such as the proportion of black people feature, were included in the dataset simply by constructing scatter plots and histograms. In analyzing the scatter plots and histograms, I saw that it had very little correlation to the median house price:



Consequently, I reasoned these features should not be included in the dataset at all, as "I believe that by dropping the remaining features, we will clear the dataset of features that are not as directly related to the median house price as the features kept in the dataset, and remove any loosely related data from the dataset, that the model might have latched on to otherwise and potentially skewed the accuracy of the median house price prediction". Once again, I included my explanations and thought process in markdown cells, to illustrate, the thought process behind my code, in a clear and concise manner, and analyze and document what I was doing:

Slicing the Data Vertically

Based on the above scatter plots of all of the features vs the median house price, the features that I have decided the keep in the dataset, while removing the rest, are the Average Number of Rooms, Distance to Boston City Centres, Per Capita Crime Rate, Percent Lower Status of the Town, and the Number of Properties Built Before 1940, because these were the features that had the strongest relationship with the median house price. Thus, since we are trying to predict the median house price using the data we have, I believe that by dropping the remaining features, we will clear the dataset of features that are not as directly related to the median house price as the features kept in the dataset, and remove any loosely related data from the dataset, that the model might have latched on to otherwise and potentially skewed the accuracy of the median house price prediction.

```
df_data_filtered = df_data.iloc[:, 0:13]
df_data_filtered = df_data_filtered.drop(['ZN', 'INDUS', 'CHAS', 'NOX', 'RAD', 'TAX', 'PTRATIO', 'B'], axis=1)
df_data_filtered
```

	CRIM	RM	AGE	DIS	LSTAT
0	0.00632	6.575	65.2	4.0900	4.98
1	0.02731	6.421	78.9	4.9671	9.14
2	0.02729	7.185	61.1	4.9671	4.03
3	0.03237	6.998	45.8	6.0622	2.94
4	0.06905	7.147	54.2	6.0622	5.33
...	...	...	...	...	...
501	0.06263	6.593	69.1	2.4786	9.67
502	0.04527	6.120	76.7	2.2875	9.08
503	0.06076	6.976	91.0	2.1675	5.64
504	0.10959	6.794	89.3	2.3889	6.48
505	0.04741	6.030	80.8	2.5050	7.88

506 rows × 5 columns

However, in the reflection, I was able to dive more deeply into the ethicality behind all of these features (my reflection can be found here). I reasoned how in exploring the dataset, "I was surprised and worried to see that the "B" feature in the dataset represented a function related to the proportion of black people in the town, thinking that I had simply misread the predictor's description. I was confused, to be honest, and tried to graph it against the median house value feature, but saw that there was very little correlation between them. I also tried to create a histogram for its values, but once again saw that there wasn't necessarily a clear distribution, confusing me even more as to why it was included in the dataset. Funnily enough, societal and historical discriminations are topics that I am also discussing in History and English class, so with the disgust carried over from these classes, I attributed this feature to the umbrella of being a racially motivated idea". Moreover, I discussed the implications of having features such as the LSTAT, CRIM, and PTRATIO features as part of a dataset, and talked about how creating a false stigma around a community can directly affect the livelihoods of its people.

Additionally, I explored the mathematical intuition behind the "B" column by reading interesting articles about problems with the Boston Housing Dataset, and how the authors "made the data non-invertible, but also made it so that you can't recover the original data, due to there being multiple x values for each y value, putting into question, the transformation and if it is purposefully transforming it to target certain racial groups". I also discussed the ethical issues behind this transformation, and how a practice that the researcher may be using, is "purposefully manipulating the data to have it such that they are discriminating against black people and specifically targeting them, by showing that they affect a house's price".

The final step I took in my reflection was discussing the impact of having datasets such as the Boston Housing Dataset be publicly available as toy datasets and how it can teach young

learners that using ethically wrong data to get better predictions, is okay. I also went ahead and looked at outside resources, such as:

- https://www.ichorstrategies.com/ideas-insights-blog/without-an-eye-on-racial-equity-fintech-cant-democratize-finance

To learn more about how different "quant and fintech companies … [use] very questionable datasets" that have a high racial bias, in order to optimize their own gains and profits. Throughout this reflection, I organized my thoughts into very clear and distinct points, such that it was easy to interpret and understand, and clearly reflected the deep analysis I did into the ethical and racial issues with the Boston Housing Dataset:

Arnav Bhakta

What is one discovery you made about this process, and what is one lingering question you have?

During this process, one thing I learned is how important it is to know what the dataset you are working with is. When working with data and different models in the past, to forecast outcomes, I've generally had a pretty good sense of what the dataset was, the relationship between features, and what they entail, primarily because I was directly involved in crafting the dataset and deciding which features to include. For this process, however, I was forced to take a step back and understand the relationship between features, by creating scatter plots, histograms, and getting statistics about the data, to get more insight into what the features presented. Moreover, I started this process with no idea about what the variables were, and it challenged my abilities to find more information about them, but also informed me of how significant it is to truly research your features, not only to decided if they are good predictors and if they should be kept in your dataset, but also to weigh the ethical background of the variables.

One lingering question I have is how to decide which variables to remove from the dataset when slicing it vertically. From my impression, I tried to remove variables that didn't necessarily have a strong correlation with the median house value feature, and reduce multicollinearity in the set, by removing variables that were highly correlated with each other. However, I'm not sure if there are further issues I should consider when cleaning my dataset and removing certain features. I also had a question on how to know where to slice the data horizontally. From past experiences, I know that when splitting a dataset for training and testing, it should be roughly around a 70/30 split respectively, or more or less, based on the situation, but in the case of the lab, I had no idea about where I should start with slicing the dataset horizontally, and what factors I should consider when doing so.

If you worked on it, what did you discoveries did you make about the columns? What was your emotional response to those discoveries?

While working with the Boston Housing Dataset, I had initially used the built-in DESCR key that the dataset provided, to get a better understanding of what features were in the dataset. Upon doing so, I was surprised and worried to see that the "B" feature in the dataset represented a function related to the proportion of black people in the town, thinking that I had simply misread the predictor's description. I was simply confused, to be honest, and tried to graph it against the median house value feature, but saw that there was very little correlation between them. I also tried to create a histogram for its values, but once again saw that there wasn't necessarily a clear distribution, confusing me even more as to why it was included in the dataset. Funnily enough, societal and historical discriminations are topics that I am also discussing in History and English class, so with the disgust carried over from these classes, I attributed this feature to the umbrella of being a racially motivated idea. Outside of this feature, however, there were other features that made me question why they were included and made me sick to my stomach, such as the LSTAT feature (percent of the lower status of the population), the CRIM features (per capita crime rate by town), and the PTRATIO features (pupil-teacher ratio). All of these features are things that can't necessarily be controlled, but I also know from previous discussions, that they create a false stigma about the community and affect the livelihoods of its people. This is unacceptable, and I was concerned to see that they were included as features of the dataset.

What do you believe is the author's argument about the "B" column, and what should we make of it?

I also didn't allow my thinking behind the ethical practices in data modeling and processing stop there, as working with sci-kit learn's diabetes dataset (my work can be found here) I made sure to take a moment on reflect on if all of the features in the dataset actually had significance to what I was doing, and that they were ethically, morally, and racially justified. Per usual, I made sure to clearly illustrate my thinking in a markdown cell, as to make the organization of my thoughts easy to understand, and outline analysis behind my thinking:

(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

As seen, all the predictors in this dataset are numerical predictors that are all factors that closely impact one's susceptibility to acquiring diabetes. Unlike other scikit learn toy datasets, namely the Boston Housing Dataset, the majority of these features appear to be ethical in nature, and are directly related to diabetes, without any convolution for discriminatory or racist practices.

Investigating the shape and the keys of the dictionary, we see:

In saying this, I believe it becomes apparent that in the work with data that I do, I always make sure to consider the ethical practices behind the data that I am using. Furthermore, as seen from the above figures, I always make sure that my thoughts are organized in a clear, concise, understandable manner, that make it such that the average reader understands what I am arguing for and that my documentation and analysis tell the full story behind my thinking and actions.

Analyzing my progress, I think that once again, my skills in this learning objective have increased greatly, because as I mentioned in my reflection, “even prior to this class, I had heard about the Boston Housing Data Set, and unknowingly used it, simply because of the richness of the data and how easy it is to access it”. I had never taken into consideration the ethical practices behind data and how sometimes data is racially motivated, but with the work I have done thus far, I have learned just how important thinking about these issues in practice are, and implemented them into my daily work. Additionally, I have gone outside of the class’s resources to find articles that discuss the effects of racially motivated data in fintech industries, and have had plenty of conversation with my classmates, such as William Yue and Carissa Yip about ethics in machine learning and how there are many instances including Facebook misidentifying black people as monkeys and a lack of access to certain resources, for people of specific classes, that continue to persist today.

This being said, I believe that there is still a lot of work that I can do for this learning objective. In specific, in my work, I haven’t necessarily focused any of my technical work on ethics. While I have integrated the idea of considering ethics while dealing with data, I have not actually done a project yet that dives into how machine learning models can be modified to put the ethics of it at risk. A few ideas I have for how I can start to focus projects on these ideas, is by looking at how computer vision models can be adversely affected (adversarial machine learning per se - looking at how intrinsic properties can be changed and how they affect the computer vision model). Additionally, I believe that I can still make a bit more progress, as there are a lot of very interesting articles that I am yet to read including about black-box machine learning and ethical problems with OpenAI’s algorithms.

Learning Objective 4: I can tell stories with data, both by discussing my process in shaping/manipulating/modeling it and the choices made to do so, and through making arguments about what my findings say about the world.

1. Organization: 3/3
2. Volume of Work: 3/3
3. Analysis/Documentation: 3/3
4. Progress: 2.7/3

Thinking about my storytelling process, I think a project that largely sticks out to me once again is the Myers Briggs Type Prediction project (my work for which can be found here, a subpart for data pre-processing which I will be discussing in the following reflection, can be found here). Throughout the notebooks that I used, I discussed nearly every aspect of the process I took when shaping, manipulating, and modeling the data and gave a detailed, concise, and understandable description for why I made the choice I did. Starting from the beginning, I walk the reader through what exactly Myers Briggs Type Indicators (MBTIs) are, the 16 different personality types that they indicate, and what can be learned from them. Additionally, I discuss what exactly we will be doing in the notebook and characteristics of the dataset, in a markdown cell, where each thing I discuss logically follow each other, and are clear and easy to understand:

Data Pre-Processing for Predicting Myers Briggs Types

By Arnav Bhakta[1] and William Yue[1]

[1] **Phillips Academy Andover**

The authors would like to thank Patrick Chen, Michael Huang, Ali Cy, and William H. Yue for their helpful input and advice in crafting this notebook.

In this notebook, we will be going through the data pre-processing steps that are necessary in order predict Myers Briggs Type Indicators (MBTI). To give a bit of an overview, MBTI is an "introspective self-report questionnaire indicating differing psychological preferences in how people perceive the world and make decisions" [1]. It "divides everyone into 16 distinct personality types across 4 axis:

- Introversion (I) – Extroversion (E)
- Intuition (N) – Sensing (S)
- Thinking (T) – Feeling (F)
- Judging (J) – Perceiving (P)"

and assigns everyone a label, based on which of the personality types they fulfill [2]. "For example, someone who prefers introversion, intuition, thinking and perceiving would be labelled an INTP in the MBTI system" [2]. In doing so, it is possible to get more of an overview of someone's personality, their preferences, and behaviors, and psychological perspective. Hence, in the current study, we look to levarage machine learning (ML) to correctly and accurately classify people's personalities or MBTIs, based on how it is that they "speak" and interact with others. The data is taken from the Kaggle (MBTI) Myers-Briggs Personality Type Dataset, which provides text of people interacting in a forum, as we presume from reading over samples in the dataset, which are then labeled with their corresponding personality type [2].

In the current notebook, we have seperated the presented dataset into a smaller set, consistent of two types of labeled data, Introversion (I) and Extroversion (E). Using the provided texts for each of these labels (Introversion being labeled a 1 and Extroversion being labeled a 0), we hope to be able to accurately predict each of these personality types.

Next, I discuss how the data is loaded into the notebook, such that the reader truly does understand each step of what I do with the data. As I used Google Colab for data pre-processing, due to the fact that I stored the data in Google Drive, I made sure to tell a clear and concise story to the reader about how I loaded in the data, mounted my drive, and imported the necessary libraries for the task that I had at hand, such that they could get a clear picture of what it was that I was trying to achieve and doing. As usual, my explanations were put into markdown cells, that were neat, organized, provided a good documentation and analysis for what I was doing:

```
print(__doc__)
```

```
Automatically created module for IPython interactive environment
```

Mounting Google Drive

As data pre-processing was done with Google Colab in Google Drive due to the ease of being able to store files on Drive, we must first mount our Drive, in order to access files on it.

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

Importing Libraries

We import the below libraries to help us with data pre-processing. Pandas is primarily used for loading in the data from a csv, and creating DataFrames. NumPy is primarily used for creating arrays and simple arithmetic. Regular expression operations (re) are primarily used for splitting up the samples and removing unwanted or implicative text from the dataset. Tokenize and Tokenizer are primarily used to split up sentences into smaller units or words called tokens, to helping to understand the text and build the model, by making it easier to understand the meaning of the text, by analyzing it as a sequence of words. The remaining libraries are used for importing and exporting data.

```
import pandas as pd
import numpy as np
import re
import tokenize
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from bs4 import BeautifulSoup
import requests
import json
```

Loading in the Data

Below, we load in the data and read the csv from My Drive, using Pandas, and split up the columns of the dataset into two distinct features: the text and labels. We then cast them to NumPy array from Pandas DataFrames, for ease of use later on.

```
df = pd.read_csv('/content/gdrive/My Drive/Peitho/0.csv')
df_text = np.array(df['text'])
df_label = np.array(df['label'])
```

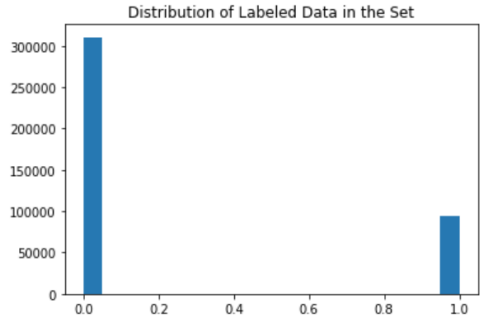
As can be seen above, I clearly explain every step that I take, and my reasoning behind it, so that the reader is not left out from any details.

As discussed in the above sections, I then give a very in-depth overview of tokenization, text to sequences, and the padding steps that were taken, in such a way that they clearly layout out for the reader, the decisions that were made when coding the proceeding steps, the reasoning and intuition behind what we did using both text and histograms, and provide documentation and analysis that explain exactly what our code is doing. In such a way, we are able to effectively storytell how we pre-process and handle the data, in a very organized manner, that follow step after step:

Exploring the Data Distributions

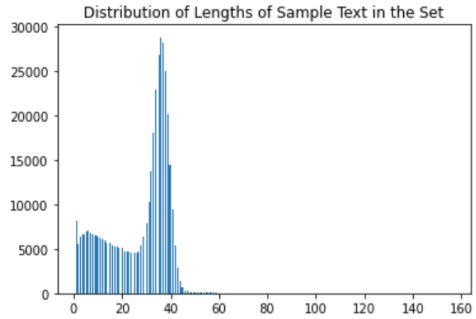
Seeing as we have two specific features, it is important to understand more about their distributions by creating histograms, so that we can learn more about the features themselves. For the labels feature, we will primarily be looking at the split between samples labeled for Extroversion versus samples labeled for Introversion, allowing us to understand which is more prevalent in the current dataset. For the text feature, stored in the `NumPy` array `df_user_flattened`, we will be looking at the distribution of the lengths of each of the samples, in order to understand how much each sample will be padded on average, when padded or truncated to 150 words.

```
_ = plt.hist(df_label_flattened, bins='auto')
plt.title("Distribution of Labeled Data in the Set")
plt.show()
```



As can be seen from the above histogram, the large majority of labels in the dataset are for Extroversion (0) as opposed to Introversion (1) (roughly a 3 to 1 split). Though it is hard to decieve why this may be, a guess as to why, may be that as the text samples were taken from an online forum, where people are more inclined to be conversational, the majority of samples may have consequently exuded extroversive qualities. In discussing the implications of such an uneven split in the dataset, it could be argued that the set does not provide the model with enough introversive samples to truly be able to pick up on what characteristics of the text make in introversive in nature. However, it is also important to note that there are roughly 100k samples for texts labeled as being introversive, which should theoretically be enough for our model to learn what makes a text introvesive. Hence, we shall not alter the distributions of labels in the dataset.

```
lens = []
for i in df_user_flattened:
  sample_lens = i.split(" ")
  lens.append(len(sample_lens))

_ = plt.hist(lens, bins='auto')
plt.title("Distribution of Lengths of Sample Text in the Set")
plt.show()
```



In observing the the distribution of the lengths of samples in the dataset, it can be seen that roughly 7.5% have a length close to 38 words. Given that the average tweet is roughly 55 words in length, we can assume that the majority of texts are very short in length and that when padded to a length of 150 words, the sequences will constist largely of 0s. Additionally, in looking at the distribution itself, we can see that it follows a roughly normal distribution, which reminds me of the Central Limit Theorem, and the fact that as we have more samples, as is the case in the current study, the distribution will be more likely to approach an average or most common frequency.

Tokenization

As mentioned above, we tokenize the text, by splitting it up into smaller units, as to optimize our model's ability to analyze the text and discover patterns within it. However, prior to doing so, we first go ahead and flatten the array, by reducing its dimensionality to a 1-dimensional array, in order to be able to tokenize the text, as currently, our text is within a series of nested arrays. We do this, be defining a new list, `df_user_flattened`, and looping through each nested array in the `df_user` array, which currently holds all the texts, and assigning each string of text within these nested arrays as new elements in our new 1-dimensional `df_user_flattened` list.

```
df_user_flattened = []
```

```
for i in df_user:
  for j in i:
    df_user_flattened.append(j)
```

We then convert `df_user_flattened` to a NumPy array for ease of use, and display its values. As we can see, as opposed to `df_user` which consisted of several arrays nested within each other, `df_user_flattened` contains only 1 array, meaning that we successfuly reduced the dimensionality of our data.

```
df_user_flattened = np.array(df_user_flattened)
display(df_user_flattened)
```

```
array(["' and moments sportscenter not top ten plays pranks",
       'what has been the most life-changing experience in your life?',
       ' on repeat for most of today.', ...,
       'i have seen it, and i agree. i did actually think that the first time i watched the movie, and from the begin
ning (or when they got their powers) i kinda thought andrew would never work right with...',
       "ok so i have just watched underworld 4 (awakening) and must say it was a really good film, compared to the ot
her films out in the last few months anyway. i don't think it was as good as the first 3...",
       "i would never want to turn off my emotions. sometimes i hide them from the world, but i still need them for m
e.'"],
      dtype='<U874')
```

Next we go ahead and tokenize our data. In the current study, we do so using `Tokenizer` from `Keras`. `Tokenizer` takes in a few parameters, when tokenizing the data, which are as follows: num_words returns the ids of the `n` most commonly used words in the dataset, where `n` is the `vocab_size` we defined as `vocab_size = 4000`, `oov_token` is used to replace out of vocabulary words.

```
vocab_size = 4000

oov_tok = "<OOV>"

tokenizer = Tokenizer(num_words = vocab_size
                      ,oov_token= oov_tok
                      )
tokenizer.fit_on_texts(df_user_flattened)
word_index = tokenizer.word_index
```

Post-tokenization, we are able to call `word_index` from `tokenizer`, to get the most commonly used words in our dataset, which are:

Texts to Sequences

In order to be able to train our model using the available texts, we then use the `texts_to_sequences` method to convert our text to a sequence of integers. This is done by using the most frequent words that we found above, and replacing these words that the tokenizer knows, we integers, such that the model that we will build is able to interpret the text.

```
tokenized = tokenizer.texts_to_sequences(df_user_flattened)
```

```
tokenized
```

Padding

The final step that we take, is padding each of the samples, to be the same length. As seen above, the max number of words in the samples of texts that we are provided with is 156 words, so we pad all of the sequences to a length of 150, to ensure that all of the sequences have the same length. This is done using the `pad_sequences` method, which takes the tokenized and sequenced text that we just defined, and pads all of the sequences or truncates them to a length of 150, by adding on 0s until the sequence has a length of 150, or removing integers from the sequence, until the sequence has a length of 150.

```
padded = pad_sequences(tokenized, maxlen=150, padding='post', truncating='post')
```

As seen below, our padding was successful, and leaves us with an array of 405263 sequences of length 150, to pass into our model, so that we can predict MBTIs.

```
padded.shape
```

```
(405263, 150)
```

Moving onto how I described how the data was manipulated, in describing how the training, validation, and test splits were taken, I was sure to first describe what exactly these splits were, so a novice to machine learning could understand what their purpose was, before then discussing how exactly each split was formed from the full set and the steps that were taken to ensure that the data was adequate to be passed into the model. In this manner, following the same intuition behind what I had done for the data pre-processing steps, I clearly articulated what exactly the purpose of what I was doing was and why I was doing it, so that it nicely fit into the story of building the model I was telling in a very clear sequential pattern, and provided an analysis, explanation, and documentation for the code I was running:

Training, Validation, and Test Splits

To train and properly test the accuracy of our model, by defining a training set, validation set, and test set. The training set is the set which used to fit the model and that the model uses to learn patterns in the text of specific categories. Consequently, the training set, consists of both the text samples and its corresponding labels, such that the model can learn how to properly classify texts. The validation set serves a sort a sort of test set, which we will see soon, to test the accuracy of the model while training on a subset of the data. The test set is usde to test how well the model performs when predicting the labels of text, by providing a set of text samples as inputs, having the model predict the corresponding labels of each sample, and comparing it to the actual labels, which were not provided for testing, but were rather kept to compare how well the predicted labels matched the actual labels.

We go about splitting up our training, validation, and test sets by attributing 60% of the dataset to the training set, 20% to the validation set, and 20% to the test set (this is done by taking horizontal slices of the dataset). This is done by combining the `df_user_flattened` array of text samples and the `df_label_flattened` array of the corresponding lables, into a single array, `ds` , where column 1 is the actual text samples, and column 2 is the labels that correspond to each of the respective labels. `ds` is then shuffled using `np.random.shuffle` to ensure that there is no specific pattern that the samples in dataset are distributed via, and that they are instead in a random order.

As can be seen from `ds_train` , such a process results in the creation of an array that consists of several arrays nested inside of it, which can be observed as containing the sample text in one column, and the label for this text in the other.

```
seed = 23
np.random.seed = seed
ds_df = pd.DataFrame()
ds_df['posts']=df_user_flattened
ds_df['labels'] = df_label_flattened
ds = ds_df.to_numpy()
np.random.shuffle(ds)
front_cutoff = int(ds.shape[0]*0.6)
mid_cutoff = int(ds.shape[0]*0.8)
ds_train, ds_val, ds_test = ds[:front_cutoff], ds[front_cutoff:mid_cutoff], ds[mid_cutoff:]
ds_train
```

```
array([['if i want to enjoy life more then what other types or aspects of type should i try to develop in myself? for
instance, should i somehow try to develop more s? in essence what i want is to break...',
```

Finally, in discussing how exactly the data was modeled, I once again didn't leave out any details, as to clearly illustrate the data's story I was trying to tell. Although it was tedious, I went through step by step and discussed each separate part of the bidirectional LSTM model that I built in a way that even the average person could understand what I was explaining. Moreover,

in looking at the actual Sequential model itself, I discussed each separate component in a meaningful order (being embedding layers, then bidirectional LSTMs, then dropout layers, then dense layers, then learning rate, and then finally the loss function) such that they all added onto what was being discussed. Hence, it is evident to me that the process I took in describing my process with the data was clear, organized, and understandable, and gave a justification, documentation, and analysis for the steps I was taking in my code, as to paint a clear and concise story of what I was doing. Once again, these descriptions were included in markdown cells:

Bidirectional LSTM Model

In the following section, we will be discussing the model utilized in this study. As we are simply testing the effectiveness of different models in classifying sample texts on their MBTIs, we do not begin by constructing an intricate and complex architecture, but instead begin with a very baseline model.

We start building out model, by defining a `Sequential` model using `Tensorflow`'s `Keras`: a stack of layers in a neural network, where each layer has one input tensor and one output tensor. The layers of the model as as follows: an embedding layer, a bidirectional LSTM, a dropout layer, a bidirectional LSTM, a dropout layer, a dense layer, a dropout layer, and a final dense layer, as outlined in the model summary.

Embedding Layers

Embedding layers are utilized in neural networks to handle word embedding, or a process by which words are represented via vectors. As opposed to more traditional bag-of-word encoding, which produce very sparce vectors, embedding provides very dense vectors, where each vector represents a single word. As a result, an embedding layer is often used as the first hidden layer in neural networks, when dealing with text data [3].

As an input, the layer requires an integer representation of the words, which we have already done via sequencing the tokenized text samples. It then requires 3 arguments, being the `input_dim`, the `output_dim`, and the `input_length`. The `input_dim` refers to the vocabulary size, or the the number of words that are represented in the sequenced data, that we defined while tokenizing the sample texts. The `output_dim` refers to the size of the vectors space that the words we pass in, will be embedded into. More generally, it sets the size of the output vector for each word. In the current study, we begin by setting the `output_dim` to 16, however, with tuning and testing, this value is subject to change. The final argument that we specify is the `input_length`, which refers to the number of words in each input sequence. During padding, we padded and truncated all samples to a length of 150 words, so this is the value we specify for this argument [3].

Bidirectional LSTM

The following explanation requires a bit of familiarity with LSTMs. If you don't have that, that's okay! You can refer to my explanation of LSTMs in this notebook and come back here.

Bidirectional LSTMs can be viewed as a subtypes of LSTMs that have been improved for classification tasks. It does so by training two LSTMs in parallel on the input data, as opposed to the traditional one, by having one be trained on the forward input sequence and the other being trained on the backward input sequence. This is done by simply duplicating the the first layer of the LSTM in the network, such that it results in there being two layers side by side and have better access to all of the input information that is provided [4], [5]. In doing so, it is able to split the state of the neurons, to get a more whole representation of the sample texts, as opposed to simply having a linear interpretation [4], [5]. From a more wholistic view, this can be explained as not being able to understand a sequence of words, until you know what their future context is, emphasizing the need to have a betterunderstanding of the "end" of the sequence [6].

In this notebook, we utilize 2 bidirectional LSTMs as a baseline to test their effectiveness for our specific classification task. Each layer consists of 16 memory units, with the first hidden layer having a fully connected layer, that utilizes a sigmoid activation function to provide an adequate output.

Dropout Layer

Neural networks are often at risk of overfitting when training, which can have several adverse effects, including becoming accustomed to the noise of the training set, and consequently having low accuracy when predicting on new datasets. Dropout layers seek to prevent overfitting, by randomly ignoring a specificied number of layer outputs (in our case 10%), and allowing the layer to be viewed as a new layer, due to the differing number of nodes and changing connectivity to the input layer. As a result, each layer is seen as being different, meaning that there is a lesser amount of reliance on noise in the training set, when predicting the output vectors, and that there is the opportunity to correct any wrongdoings in previous layers. Thus, dropout layers allow us to place more trust in the fact that our model will be accurate and not overfit [7].

Due to its relaiability, we place dropout layers in between each of the layers we use in this model.

Dense Layer

Dense layers are deeply connected layers in a neural network, that recieves input from the neurons of previous layers, and uses matrix-vector multiplication to transform the vectors into a single vector of a specified size. This specific value is specified via the `units` parameter. As our model consists of two dense layers, the first vector outputted has a size of 8, while the second has a size of 1. In our model, we also specify activation functions for each dense layer, via the `activation` parameter. Activation functions are often used to create a non-linear relationship in the input neuron, such that the model can learn more complex relationships between input and output vectors [8]. The activation function we use for the first dense layer is a rectified linear unit (ReLU), which is a piecewise linear function, that outputs the input directly if the input is positive, or zero otherwise. In doing so, it allows model to learn faster and perform better. A sigmoid activation function is simply a logistic function, that that transforms the input into a value between 0 and 1 [9].

Learning Rate

The learning rate is a tuning paramater that specifies the step size at which weights are updated during training, while the loss function as it approaches its minimum. By doing so, it is responsible for how the model changes as a result of the estimated error it produces, each time its weights update, meaning that it is also in charge of how quickly the model adapts to the presented task. A large learning rate can cause the model to converge very quickly in the direction of an incorrect prediction, so for our model, we set a learning rate of 0.0001, so that the model spends enough time understanding trends in the input sample texts and updates accordingly [10].

Loss Function

A loss function can be viewed as providing a metric for how well your model fits the training set. More specifically, as we are tasked with a binary classification problem, the loss function we utilize is binary cross-entropy:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

where y is the label for the corresponding sample text, and $p(y)$ is the probability of the label being 1, for all of the labels in the set. More specifically, for each label of value 1, it adds the log probability of it being 1, $\log(p(y))$ to the loss, and adds the log probablity of the label being 0, $\log(1 - p(y))$, for each label of value 0. In doing so it is able to provide a justified predicted probability of the classifier being right, while also penalizing the model for inaccurate predictions [11].

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(4000, 16, input_length=150),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16, return_sequences=True)),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16)),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

learning_rate = 0.0001
model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate), metrics=['accuracy', 'mse
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 150, 16)           64000
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 150, 32)           4224
_________________________________________________________________
dropout_15 (Dropout)         (None, 150, 32)           0
_________________________________________________________________
bidirectional_11 (Bidirectio (None, 32)                6272
_________________________________________________________________
dropout_16 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 264
_________________________________________________________________
dropout_17 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 9
=================================================================
Total params: 74,769
Trainable params: 74,769
Non-trainable params: 0
_________________________________________________________________
```

After creating the model, we then train the model, by fitting it to the padded training texts and its corresponding labels. I train the model on 5 epochs – nowhere near to final number of epochs I would train the model on, due to the low learning rate – in order to get a general understanding of how well the bidirectional LSTM performs when predicting the MBTI types. Training for the model took roughly 6-7 hours.

```
model.fit(x=padded, y=np.asarray(ds_train_labels).astype('float32'), batch_size=8, epochs=5, validation_data=(padded_v
```

Furthermore, I didn't just stop here, but I discussed the real world implications of the story I was telling, and flaws that may have occurred within it. For example, I conducted a literature review, and looked at how LSTMs have performed in previous studies that focused on predicting MBTIs. In doing so, I was able to discuss how my model performed substantially better in accuracy (~76.5%) compared to previous models (~54.0%). Additionally, I was able to discuss how my model performed in actuality, by discussing the distributions of the models predicted labels. By doing so, I was able to highlight critical flaws with how our LSTM performed and the false confidence that was placed in its accuracy. Moreover, this offered me the opportunity to analyze how we could improve our model and compare the methodologies we used, compared to other real world methods, such as CNNs. Thus, in providing a detailed storytelling of how I handled the data, I was also able to create a grounds to discuss the real world implications of what I did and how they compared to current machine learning understandings. Per usual, this description was provided in markdown cells, in a manner than followed a logical order and clearly documented and analyzed the code that was written and its importance:

Results

The final step we take, is seeing how well the model performs when predicting on the testing set. We do this by using the `model.predict` function, and having it predicting on the padding sample texts, and comparing it to the actual labels for the corresponding texts. We can then compute metrics for the accuracy of the predictions, using the `model.evaluate` function.

In doing so, we see that in comparison to Hernandez and Knight's model, which recieved an accuracy of 54.0%, our baseline model greatly outperforms theirs with an accuracy of 76.504%. Though this result may seem to be very good compared to those of previous studies, something that seems off, is the fact that in our dataset we know that roughly 25% of the sample texts have a label of 1, while the remaining ~75% have a label of 0. This raises some suspicions due to the fact that we achieved a nearly 75% accuracy, which might implicate that our model might not be as accurate as we had initially assumed, and that it is just predicting 0s.

```
predictions = model.predict(padded_test)
```

```
print(np.mean(predictions))
```

```
0.22614066
```

```
labels_test = np.asarray(ds_test_labels).astype('float32')
```

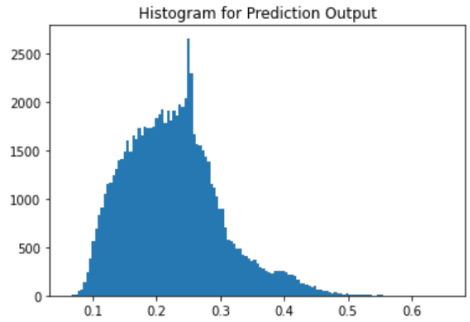
```
model.evaluate(predictions, labels_test)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 150) for input KerasTensor(type_spec=TensorSpec(shape=(None, 150), dtype=tf.float32, name='embedding_5_input'), name='embedding_5_input', description="created by layer 'embedding_5_input'"), but it was called on an input with incompatible shape (None, 1).
2533/2533 [==============================] - 7s 2ms/step - loss: 0.5626 - accuracy: 0.7651 - mse: 0.1869 - mae: 0.4043 - mape: 244507056.0000
```

```
[0.5625528693199158,
 0.7651042938232422,
 0.18689122796058655,
 0.40433382987976074,
 244507056.0]
```

Looking more in-depth at the distribution of the predicted label values via a histogram, we see nearly all of the predictions are < 0.5. In fact, the majority of the predicred labels recieved values < 0.3, indicating that the model had a tendency to predict high values of 0 labels.

```
_ = plt.hist(predictions, bins='auto')
plt.title("Histogram for Prediction Output")
plt.show()
```



In seeing these results, it is clear a more complex architecture is required to handle this classificaton task. Moreover, it suggests that bidirectional LSTMs may not be the best approach for handing this problem. Instead, we plan on testing the effectiveness of convolutional neural networks (CNNs), which have historically been know to perform optimally for classification tasks. We are considering the possibility of taking a combined approach using eXtreme gradient boosting (XGBoost) and CNNs, to pretrain the data via a decision tree-based method, before passing the outputs as inputs to a model consistent of CNNs. A final model we may consider in the future, is the use of transformers, due to their prevalence in handling textually based data.

However, this isn't the only project in which I clearly told a story of how I handled the data. My whole notebook on sci-kit learn's diabetes dataset (my work can be found here) gives a clear description of the steps I used when deciding how to handle my data, as for one, I started with nearly no idea of how the data was distributed and what the features were, and as a result, was able to give a clear description behind my reasoning for the steps I took in pre-processing, manipulation, and modeling. Consequently, it is fair to say that when handling data, I make sure

tell a clear story, that outlines why I make the decisions I do, and the implications these decisions have on my code (the next 6 pages are photos of justification):

Investigating the Diabetes Dataset

In this notebook, I will be exploring the intrinsic properties of the scikit learn Diabetes Dataset. In specific, as I've never seen this dataset before, I'll be trying to understand and interpret it using the skills I have, and visualize the data to get a better idea of its distributions and any of its underlying characters. Moreover, I hope to understand the applications of the dataset, and any methods that would be appropriate for making predictions using the provided data.

Importing the appropriate libraries

In exploring this dataset, I import pandas for data manipulation and analysis, numpy for mathematical operations and linear algebra, matplotlib.pyplot and seaborn for plotting, and sklearn for the Diabetes Dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.datasets import load_diabetes
```

Loading the Diabetes Dataset Using Pandas

```
df = load_diabetes()
```

Learning the Features of the Dataset

As with many other scikit learn toy datasets, I can use keys it has, to learn more about its features. For example I can use df.DESCR to get a description of the different features in the dataset:

```
print(df.DESCR)
```

```
.. _diabetes_dataset:
```

As seen, all the predictors in this dataset are numerical predictors that are all factors that closely impact one's susceptibility to acquiring diabetes. Unlike other scikit learn toy datasets, namely the Boston Housing Dataset, the majority of these features appear to be ethical in nature, and are directly related to diabetes, without any convolution for discriminatory or racist practices.

Investigating the shape and the keys of the dictionary, we see:

```
print("The dataset's shape: " + str(df.data.shape))
print("The dataset's keys: " + str(df.keys()))
```

```
The dataset's shape: (442, 10)
The dataset's keys: dict_keys(['data', 'target', 'frame', 'DESCR', 'feature_names', 'data_filename', 'target_filenam
e'])
```

In observing the shape of the dataset, it appears that there are 442 samples of each of the predictors, along with an accompanying target for each of these presumably clinical trials. Looking at the keys of the dictionary, we see that there is a key for the data, which I assume is the set with all of the predictors, a target, which I assume is the set with all the samples of the response, DESCR, which gives a description of the dataset, data_filename and target_filename, which I assume give paths to their desired locations, and feature_names, which I assume returns a list of the feature names:

```
df.feature_names
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

As seen below, is it relatively hard to visualize the data, using numpy arrays:

```
df.data
```

```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
```

Consequently, we will create a DataFrame out of the both the data and the target, to get a better idea of what exactly they look like:

```
df_pd = pd.DataFrame(df.data, columns = df.feature_names)
```

The above code only creates a DataFrame out of the predictor variables, so we will create a new column for the response feature:

```
df_pd['DPROG'] = df.target
```

```
df_pd.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	DPROG
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

As seen above, we were successfully able to create a much more understandable DataFrame to visualize the dataset. Moreover, using in-built functions of pandas, we can learn more about the distribution of features in the dataset:

```
df_pd.describe()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	DPRO
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	442.0000
mean	-3.639623e-16	1.309912e-16	-8.013951e-16	1.289818e-16	-9.042540e-17	1.301121e-16	-4.563971e-16	3.863174e-16	-3.848103e-16	-3.398488e-16	152.1334
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	77.0930
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123996e-01	-1.267807e-01	-1.156131e-01	-1.023071e-01	-7.639450e-02	-1.260974e-01	-1.377672e-01	25.0000
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665645e-02	-3.424784e-02	-3.035840e-02	-3.511716e-02	-3.949338e-02	-3.324879e-02	-3.317903e-02	87.0000
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670611e-03	-4.320866e-03	-3.819065e-03	-6.584468e-03	-2.592262e-03	-1.947634e-03	-1.077698e-03	140.5000
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564384e-02	2.835801e-02	2.984439e-02	2.931150e-02	3.430886e-02	3.243323e-02	2.791705e-02	211.5000
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320442e-01	1.539137e-01	1.987880e-01	1.811791e-01	1.852344e-01	1.335990e-01	1.356118e-01	346.0000

```
df_pd.isnull().sum()
```

```
age      0
sex      0
bmi      0
bp       0
s1       0
s2       0
s3       0
s4       0
s5       0
s6       0
DPROG    0
dtype: int64
```

Using the pandas methods, we see that there are no invalid observations in the dataset, and that the mean and distribution of the numerical predictors are relatively evenly distributed. Interestingly though, they all have the same Standard Deviation, which indicated to me, that among the features, there are relively no levels of variance.

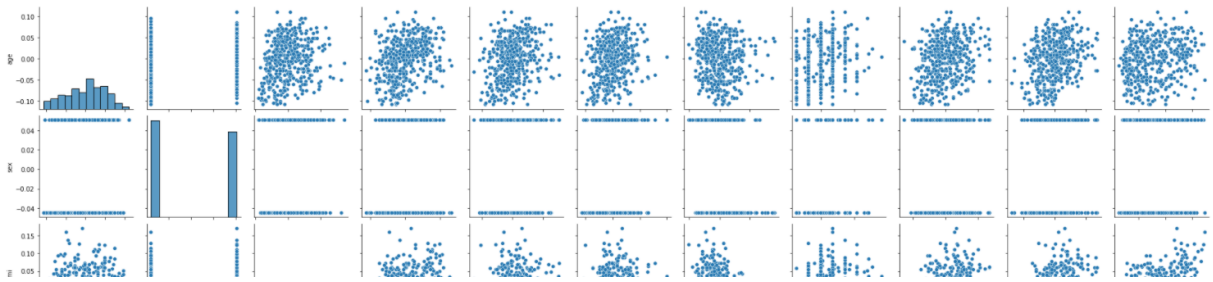
Graphing the Features

In this section, I will be using different plotting libraries to create histograms, scatterplots, heatmaps, etc. to get more statistical information about the relationship each of the features.

We begin by using sns's pairplot to create a figure that displays scatter plots between and histograms for all of the features.

```
sns.pairplot(df_pd)
```

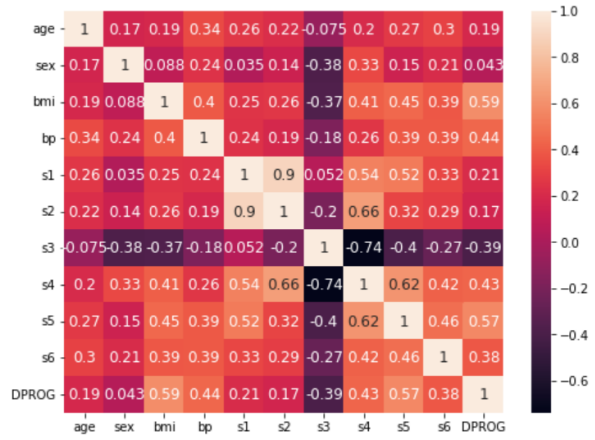
```
<seaborn.axisgrid.PairGrid at 0x7fc2b97a0730>
```



As seen in the above figure, it appears that the sex feature is largely ordinal in nature, presumabely having the values -0.044642 and 0.050680 to represent either male or female. As seen from a histogram of its distribution, in the set, there are a roughly equal number of participants of each sex. Another variable with an interesting distribution is the s4 feature, or the total cholesterol feature. As opposed to having a more normal distribution, there appear to be 4-5 main total cholesterol measures that patients can have, with the majority of patients having near 0 measures of cholesterol.

Other than these two features, the rest seem to follow a more normal distribution, and for the most part, cluster together in a somewhat linear pattern, when scattered with the DPROG feature. Nonetheless, we can learn more about the each feature's relationship with each other, by creating a heatmap of correlations:

```
corr = df_pd.corr()
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(8,6)
plt.show()
```



As the correlation heatmap displays, the majority of features have relatively low degrees of correlation with the DPROG feature, with the highest correlation being 0.59, and the lowest being 0.043. In observing this dataset, this comes as a surprise to me, as in looking at all of the predictor variables, the majority of them have historically been known to affect one's susceptibility to diabetes. Nonetheless, their correlation scores with the DPROG feature, indicate that they may not be as good of predictors as I have initially assumed.

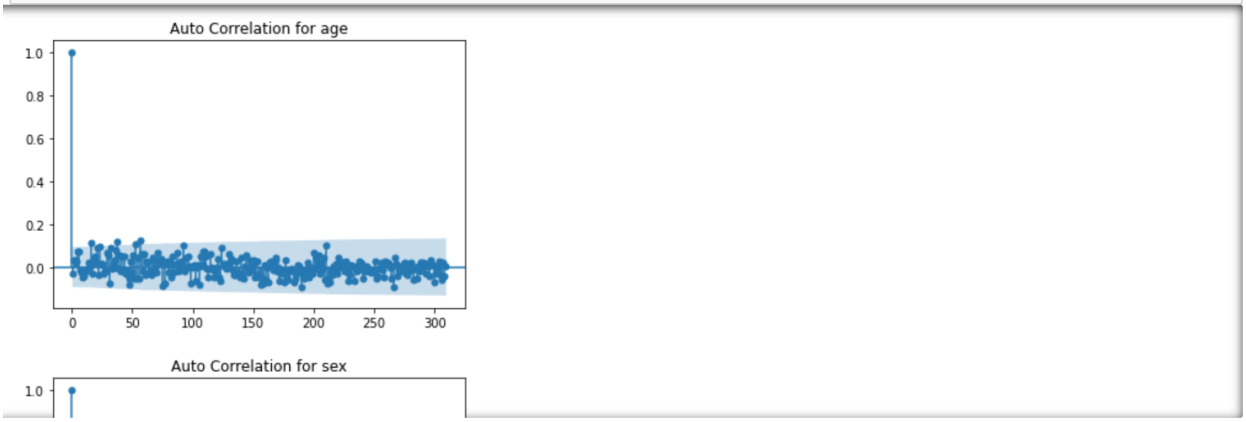
Furthermore, the correlation between the predictor set also comes as a surprise to me, as features I had assumed would by highly correlated with each other, such as s1 and s4 (0.54 correlation score), simply aren't. Though this comes as a surprise, it also has its upside, as in traditional machine learning algorithms, high degrees of multicollinearity in the training set isn't necessarily favorable.

As the purpose of this dataset is to accurately predict the DPROG feature, I also explore statistical features of the dataset, using the statsmodel library:

```
import statsmodels as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

I begin by plotting the autocorrelation for each variable. As we will splitting the dataset into training and testing splits later on, I will look at the autocorrelation for the training set in specific, which I will take at the first 70% of the set (442*0.7 = 309 lags):

```
for i in df_pd:
  plot_acf(df_pd[i], lags = 309)
  plt.title('Auto Correlation for %s' % i)
  plt.show()
```



In inspecting the above autocorrelation plots, it appears that nearly every variable has autocorrelation values fall within the 95% confidence interval (the light blue highlighted area) by the 2nd lag. Such distribution of autocorrelation values for each feature, indicates to me that within the variable themselves, there is once again a very low level of correlation, which is optimal for forecasting with them.

Forecasting With the Dataset

Provided the relationships between features that we discussed in the previous section, I will now be exploring how well the predictor variables will be able to forecast the response or the DPROG feature. Although in the desciption of the dataset, it appears that Efron et al. used least angle regression, I will instead be using a Recurrent Neural Network (RNN), in specific a Long Short-Term Memory (LSTM) architecture to learn more about the algorithm and how well it is able to perform.

In doing some background research into LSTMs, I was able to discover that they use a stochastic gradient descent optimization algorithm in training. By having a set learning rate or rate at which each weight changes during each epoch of training, they are able to change the model based on the error gradient provided when the estimated coefficients are updated. In specific, the weights are updated using a back-propogation algorithm, which also helps in preventing the vanishing gradient problem. The vanishing gradient problem is an issue that RNNs face, as when more layers are added to the neural network, the gradient of the loss function approaches zeros, causing a severe amount of inaccuracy in a model. By using the back-propogation algorithm and maintaing a cell state, LSTMs avoid this problem, and allow for more accurate forecasts and pattern recognition [1], [2].

Additionally, in researching cost/loss functions of LSTMs, I discovered that in training, you are able to specify which type of loss function you want your model to be trained on. As a result, in building my LSTM model, I chose to use an MSE loss function, due to its ability to penalize outliers in a response set that is normally distributed around a mean, as seen in the description of the dataset and quadratically penalize predictions, such that large errors are penalized far harsher than small errors [3], [4]. The algorithm that MSE uses is as such:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $(y_i - \hat{y}_i)^2$ is the square of the difference of the actual and predicted values and n is the number of non-missing data points.

To begin building out LSTM model, I import the Keras library from TensorFlow, StandardScaler from sklearn to standardize the data, and train_test_split from sklearn to create our training and testing splits:

:
```
import tensorflow as tf
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

I then split the dataset into our response (y) and predictor sets (x) and split each set into a training (70%) and testing (30%) subset based on the percent of the dataset they are attributed:

Looking at the shape of the x_train set, we see that we were able to split the dataset, such that the training set it 70% of the original dataset (309 samples):

```
x_train.shape
```

```
(309, 10)
```

Next, I use StandardScaler to "standardize [the predictor] features by removing the mean and scaling to unit variance" [3].

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

I then convert x_train, x_test, y_train, and y_test to numpy arrays, such that operation can be easily performed on them, and reshape x_train and x_test to 3D arrays, to fit the requirements of the model.

```
x_train, x_test, y_train, y_test = np.array(x_train), np.array(x_test), np.array(y_train), np.array(y_test)

x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Finally, I begin building the LSTM model itself. I start by initialising an RNN using Sequential() and then adding 4 LSTM layers with 50 neurons each, followed by an output layer that utilizes Dense for a full connection layer. Furthermore, in between each LSTM layer, I add a dropout layer using Dropout, to prevent the model from overfitting, by randomly setting neurons in the hidden layer to 0 in between each phase of training

```
from tensorflow.python.keras.layers import Dropout

model = Sequential()

model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 50))
model.add(Dropout(0.2))

model.add(Dense(units = 1))
```

```
model.summary()
```

Hence, looking at my work for this learning objective, I think it is very apparent my storytelling of the data that I use is very methodical and well laid out, and that the steps I take to document and analyze my code and doings, whether it be via visualizations or text, all serve a very specific purpose of make my decisions making process clear and understandable. In addition, the volume of work that I have done in this area is extremely large and takes several different perspectives on this learning objective.

Thinking about my progress in this area, I say that I have progressed tremendously, as in looking back at my first notebook (work available here) I didn't even comment, forget describing what I was doing. Hence, I believe that by getting to the state I am at now, where I am able to clearly describe each action that I am taking in my notebook, regardless of how tedious it may be, is a great improvement. Additionally, as seen by the people credited at the top of my Myers Briggs Type Prediction notebook:

The authors would like to thank Patrick Chen, Michael Huang, and Ali Cy for their helpful input and advice in crafting this notebook.

I am not afraid to go to my peers for help on how to best handle my data, and greatly value their feedback, as to optimize my performance. Nonetheless, an area that I feel that I still need to improve in, is referring to more sources of more creative and fun ways in which to handle your data, as currently, I have only referred and cited a few paper:

- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6839354.pdf
- https://aclanthology.org/O18-1021.pdf

For which I have drawn inspiration from for how to handle my data, but I am yet to expand the breadth of my knowledge in this area, by referring to a greater amount of sources regarding this topic. Consequently, this is also a goal of mine moving forwards: explore more innovative methods by which to handle data, that take an unique spin, but also enhance performance.

Overall grade: (12+12+10.6+11.7)/48 = 46.3/48 = 96.45%

Additional Questions:

- Is the course more or less what you expected before taking it? If not, what is noticeably different from your initial expectations?

In all honesty, this course is very different from what I have expected it to be. For one, I was under the impression that this course would focus a lot more on the machine learning aspect of the course, by learning different algorithms and such, but I've been surprised by how much of an emphasis has been placed on learning Python and data analysis, as opposed to more popular ML techniques. Additionally, something that I actually like a lot, that I was not expecting, is the amount of freedom you have to do as you choose and explore different avenues, as I often characterize myself as a pretty curious person, so by cultivating an environment where the freedom to explore is promoted, it personally allows me to learn more.

- What challenges have you faced in dealing with the sometimes daunting amount of material we've been learning? I haven't given you explicit instructions for how to, for example, take notes, save/organize sample code snippets, research additional sources of explanations/examples for topics, or test out your thinking in your own code. How have you been faring in these "being a student" tasks, and how might you improve at them as the course continues?

I think the biggest thing has been time management and adjusting to the sheer amount of work that this class demands. In combination with college apps and all of my other classes, I've had a tough time this term systematically staying on top of all of my work, so keeping a schedule that allows me to get my work done in a systematic manner has been awfully difficult. Outside of this, the only thing I've generally been confused about, is how to properly take notes in class, due to the fact that we move so quickly, so I've been trying to figure out how best to balance listening and taking notes during class.

For how I can improve on these tasks moving forward, I think for time management, using a calendar can definitely help. I just recently started using Google Calendar, and it has helped tremendously, so I anticipate that as I become more accustomed to it, I will have an easier time staying on top of my work. As for how I can approach my problem with note taking, I think I can more generally just figure out how to consolidate my notes, so that I only focus on necessary and important details.

- We discussed at the beginning of the term that all of this material is freely available online, and so the only "real" reason we have for doing this work here is (that it's organized into assignments, and) the community of learners around you, and my role as your teacher. How have you leveraged these human resources, for better or worse thus far, and how do you hope to improve upon this as the course continues?

I have definitely leveraged these human resources over the course of this class. For example, in learning more about neural networks and machine learning techniques, I have constantly bugged Michael Huang and Ali Cy for their advice on how I should optimize my architectures. Additionally, my experiencing working on the gradients project has been very collaborative, working with William Yue, Nathan Xiong, Andrew Wen, and Davin Jeong, who have all helped me get a better understanding of what exactly I'm supposed to be doing and more generally helped me improve as a programmer.

- What are your initial thoughts about how you'd like to spend the rest of the term, post Gradients Project? This is highly non-binding, but I just want you to do some brainstorming at this point. Some examples include:
    - One thing I definitely want to do is become more accustomed with PyTorch and learn more about what exactly the difference between it and Tensorflow is.
    - I also want to learn more about computer vision, adversarial computer vision in specific, and how it can be applied to solve real world problems in fields such as medicine or in applications to defense strategies.
    - Additionally, I want to dive deeper into ethical issues with deep learning, and how we can get a better grip over preventing racial biases in prediction modeling.
    - Another idea I had was working on some sort of game bot, perhaps a tetris or chess bot. I see 2 specific approaches I can take for these ideas: one being an RL approach, which I know is relatively hard to do but very educational, and two being supervised learning, where I think the largest difficulty would be getting an adequate amount of training data.
    - A final idea I had is creating a sort of grammarly, except it is able to autograde papers, tests, etc. I don't think this is feasible but I think it would be interesting to see what types of approaches I can use to tackle this task.
- Obviously, doing any one of these necessitates doing some of the others as well, but having a focusing idea can really help you have a sense of where you want to take the second half of the term. Which of these feels *most* important to you?

I think the area that has intrigued me for the longest time is adversarial machine learning, so hopefully I can integrate that into the work that I do during the rest of this term. I also know that Michael Huang has a particular interest in computer vision, so maybe we could collaborate on a project in this domain. I've also been discussing making a game bot with William Yue for quite a long time now, so it would be particularly exciting to tackle this project.

- *(Optional)* Anything else you'd like me to know?

This is very subjective, but in thinking about the test, I believe that it's not truly a representation of my understanding of Python. I want to be clear that I'm not trying to make an excuse for my performance on the test, however, through the above reflection and the work I've done, I hope I've actively shown that I'm trying to improve in my Python skills and understanding of machine learning techniques!