Configuration Manual

MSc Research Project
Data Analytics

Amit Sahoo
Student ID: 18188851

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Amit Sahoo		
Student ID:	x18188851		
Programme:	Data Analytics	**Year:**	2019-2020
Module:	MSc Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	17 Aug 2020		
Project Title:	Classification of Wildfire Spread Severity using Machine Learning Algorithm		
Word Count:	1488	**Page Count:**	12

Signature: Amit Kumar Sahoo

Date: 17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Amit Sahoo
X18188851

1 Introduction

This document specifies detailed information about the hardware, software, and the programming languages used during the implementation of the research – 'Classification of Wildfire Spread Severity using Machine Learning Algorithm'

2 System Configuration

2.1 Hardware Configuration

- Processor : Intel(R) Core (TM) i7 -4600CPU @ 2.10GHz 2.70GHz
- Installed memory(RAM): 8.00 GB
- System type : 64-bit Operating System, x64-based processor
- Storage : 256 GB

2.2 Software configurations

Microsoft Excel 2016: Microsoft offered a spreadsheet tool that has been used to collect the data in the form of flat files as CSV (Comma Separated Values).

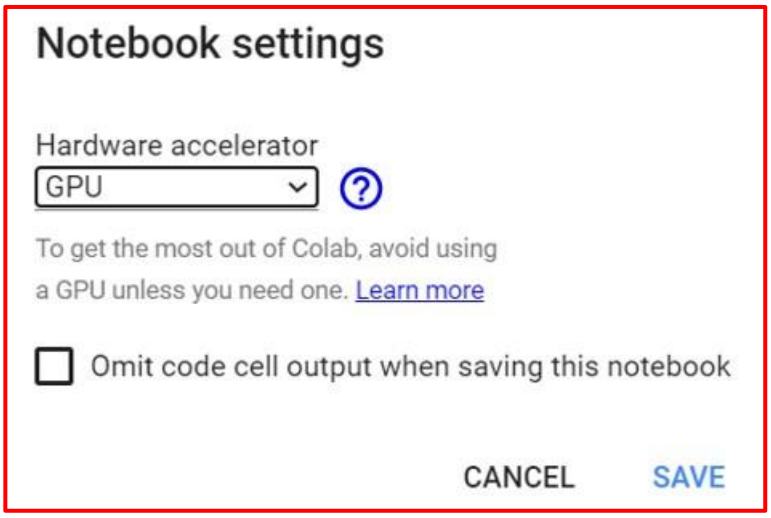
Jupyter Notebook: This is an open source web application where most of the data extraction, pre-processing and implementation are done using Python 3.7.4 version.

Spyder IDE: An open-source Integrated Development Environment (IDE) called 'Scientific Python Development Environment' (Spyder) has been used to develop python codes with 3.7.4 version.

RStudio: RStudio Desktop 1.3.1073 version has been used for explanatory data analysis and statistical analysis.

Google Colaboratory: This is a free cloud service by Google, also called as colab, which helps to develops python codes in the browser with no configuration, free access to GPUs (Shown in Figure 1) and easy sharing. This has been used to implement Neural Networks models and Deep Learning techniques.

Programming Language: Python 3.7.4 and R 4.0.2 have been used as the programming language for this research.


Notebook settings

Hardware accelerator: GPU

To get the most out of Colab, avoid using a GPU unless you need one. Learn more

☐ Omit code cell output when saving this notebook

Figure 1: GPU Setting in Google Colab

Tableau Desktop: Tableau desktop version 2020.2.4 has been used in the project for visualizing the data in a graphical manner

AWS EC2 Instances: A free tier account for Amazon Web Services (AWS) has been used to spin up t2.micro instances suitable for low to moderate performance with Mem- 1GB. The same is used to run python codes in Amazon cloud services.

3 Project Development

The implementation of this project has been done in both Python and R programming. Most statistical analysis has been done in R because it supports a wide variety of packages. However, model implementation on pre-processed data has been done in Python using sklearn and keras libraries.

3.1 Data Preparation

Data import has been done using the pandas (Dataframe) library. Data has been collected from several sources and imported into pandas dataframe. Finally, all the dataframes are merged to create the final dataset.

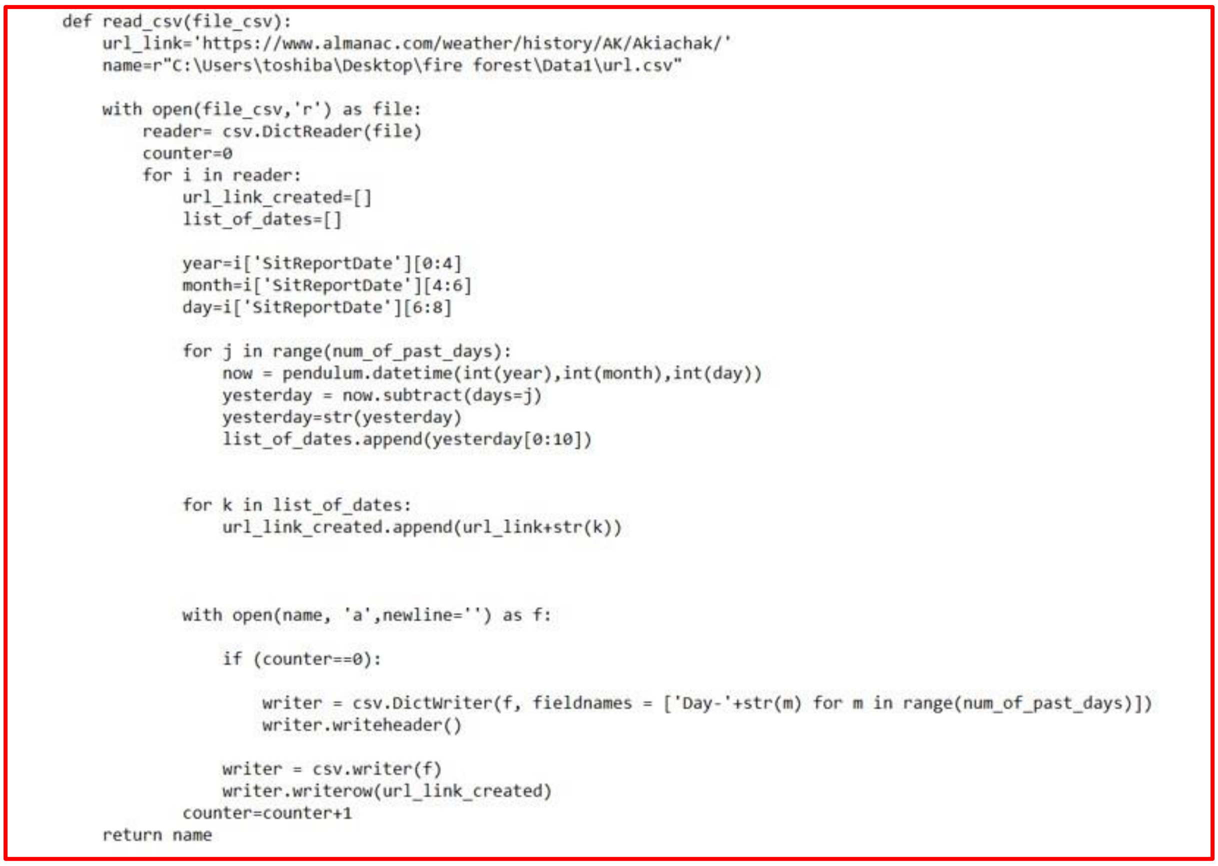
3.1.1 Fire Dataset

Fire dataset is present in the Alaska Interagency Coordination centre website[1], which has been directly imported into pandas dataframe.

1 https://fire.ak.blm.gov/predsvcs/weather.php

3.1.2 Weather Dataset

Historic weather data is present on a website, maintained by the USA government – The old Farmer's Almanac[2]. Based on the dates and location, URLs for the corresponding web pages are created and saved in a csv file (shown in Figure 2).

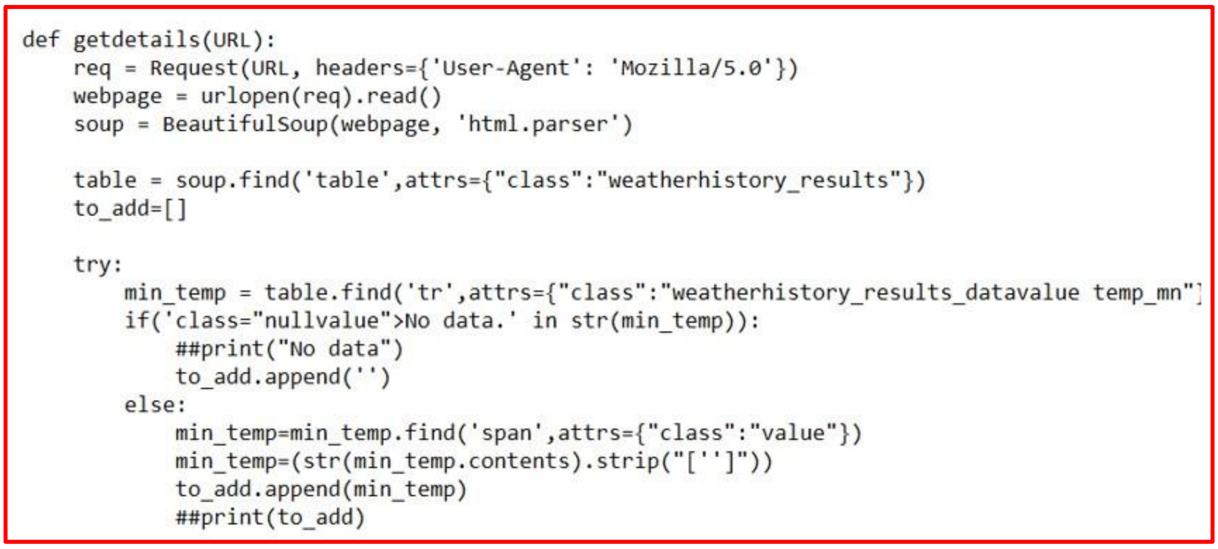


```
def read_csv(file_csv):
    url_link='https://www.almanac.com/weather/history/AK/Akiachak/'
    name=r"C:\Users\toshiba\Desktop\fire forest\Data1\url.csv"

    with open(file_csv,'r') as file:
        reader= csv.DictReader(file)
        counter=0
        for i in reader:
            url_link_created=[]
            list_of_dates=[]

            year=i['SitReportDate'][0:4]
            month=i['SitReportDate'][4:6]
            day=i['SitReportDate'][6:8]

            for j in range(num_of_past_days):
                now = pendulum.datetime(int(year),int(month),int(day))
                yesterday = now.subtract(days=j)
                yesterday=str(yesterday)
                list_of_dates.append(yesterday[0:10])


            for k in list_of_dates:
                url_link_created.append(url_link+str(k))



            with open(name, 'a',newline='') as f:

                if (counter==0):

                    writer = csv.DictWriter(f, fieldnames = ['Day-'+str(m) for m in range(num_of_past_days)])
                    writer.writeheader()

                writer = csv.writer(f)
                writer.writerow(url_link_created)
            counter=counter+1
    return name
```

Figure 2: Code Snippet for URL Creation

Onc the URLs are created, a python script is developed which would visit those URL webpages and collects the information and write it in the pandas dataframe. Figure 3 shows a sample snippet for collecting, only the temperatre field from the web page. Similar approach can be taken to collect all the weather details available on the website.

[2] https://www.almanac.com/weather/history/AK

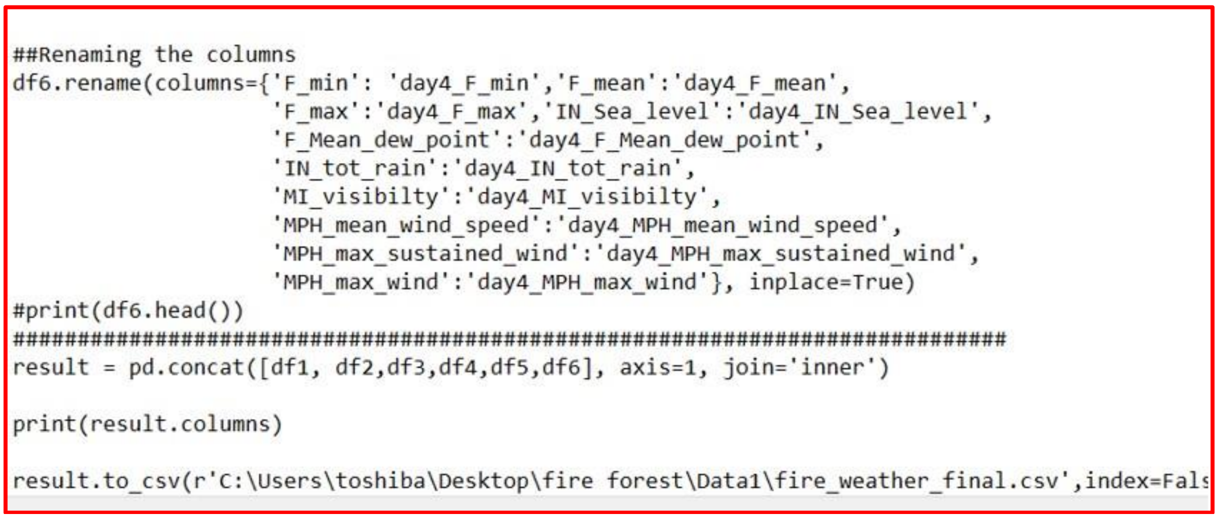


```
def getdetails(URL):
    req = Request(URL, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(req).read()
    soup = BeautifulSoup(webpage, 'html.parser')

    table = soup.find('table',attrs={"class":"weatherhistory_results"})
    to_add=[]

    try:
        min_temp = table.find('tr',attrs={"class":"weatherhistory_results_datavalue temp_mn"]
        if('class="nullvalue">No data.' in str(min_temp)):
            ##print("No data")
            to_add.append('')
        else:
            min_temp=min_temp.find('span',attrs={"class":"value"})
            min_temp=(str(min_temp.contents).strip("['']"))
            to_add.append(min_temp)
            ##print(to_add)
```

Figure 3: Code Snippet for Weather Data Collection

3.1.3 Merging and Initial Pre-processing

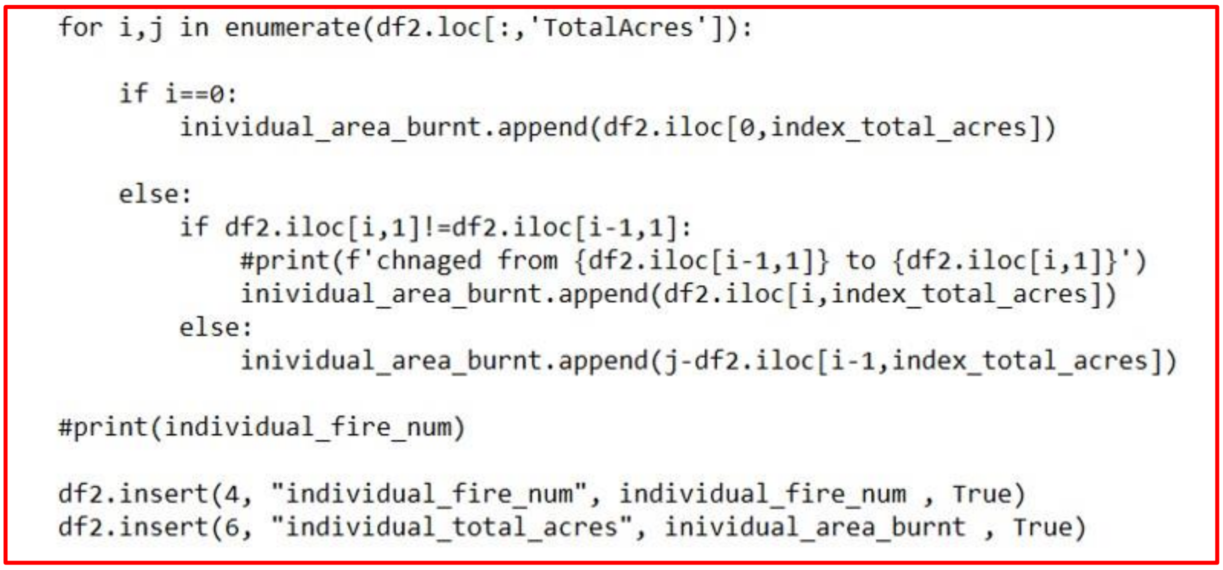
Once the data for five days have been collected, csv files containing the data are imported to pandas dataframe, columns names are modified for better understanding and finally data is merged. Figure 4 shows sample snippet.



```
##Renaming the columns
df6.rename(columns={'F_min': 'day4_F_min','F_mean':'day4_F_mean',
                    'F_max':'day4_F_max','IN_Sea_level':'day4_IN_Sea_level',
                    'F_Mean_dew_point':'day4_F_Mean_dew_point',
                    'IN_tot_rain':'day4_IN_tot_rain',
                    'MI_visibilty':'day4_MI_visibilty',
                    'MPH_mean_wind_speed':'day4_MPH_mean_wind_speed',
                    'MPH_max_sustained_wind':'day4_MPH_max_sustained_wind',
                    'MPH_max_wind':'day4_MPH_max_wind'}, inplace=True)
#print(df6.head())
##############################################################################
result = pd.concat([df1, df2,df3,df4,df5,df6], axis=1, join='inner')

print(result.columns)

result.to_csv(r'C:\Users\toshiba\Desktop\fire forest\Data1\fire_weather_final.csv',index=Fals
```

Figure 4: Merging of Datasets

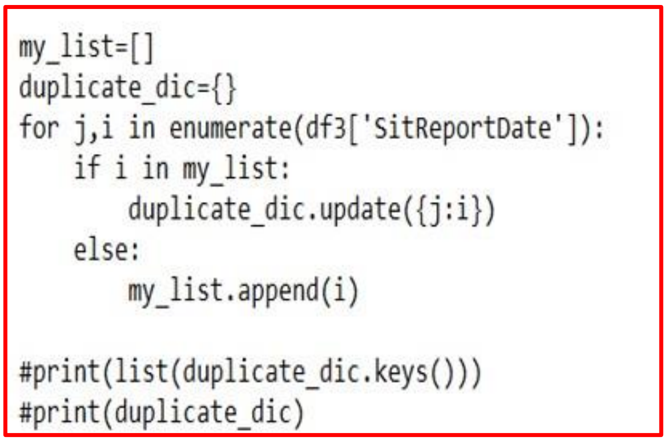
Once the final data is prepared, it requires a few initial pre-processing. Eg.- Number of fires and total area burnt is cumulative over each year, which is broken down to simpler form. Snippet for the same is shown in Figure 5.



```
for i,j in enumerate(df2.loc[:,'TotalAcres']):

    if i==0:
        inividual_area_burnt.append(df2.iloc[0,index_total_acres])

    else:
        if df2.iloc[i,1]!=df2.iloc[i-1,1]:
            #print(f'chnaged from {df2.iloc[i-1,1]} to {df2.iloc[i,1]}')
            inividual_area_burnt.append(df2.iloc[i,index_total_acres])
        else:
            inividual_area_burnt.append(j-df2.iloc[i-1,index_total_acres])

#print(individual_fire_num)

df2.insert(4, "individual_fire_num", individual_fire_num , True)
df2.insert(6, "individual_total_acres", inividual_area_burnt , True)
```

Figure 5: Conversion of Cumulative to Individual Numbers.

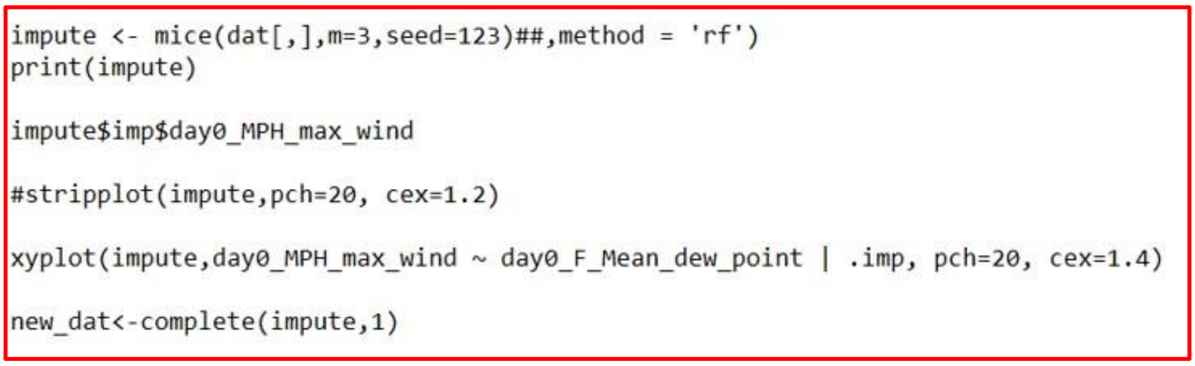
Duplicate records from the final dataset have been removed. Same is illustrated in Figure 6.



```
my_list=[]
duplicate_dic={}
for j,i in enumerate(df3['SitReportDate']):
    if i in my_list:
        duplicate_dic.update({j:i})
    else:
        my_list.append(i)

#print(list(duplicate_dic.keys()))
#print(duplicate_dic)
```

Figure 6: Removal of Duplicate Observations

Missing value imputation has been done with the help of mice package in R. Figure 7 depicts the code snippet for the same.



```
impute <- mice(dat[,],m=3,seed=123)##,method = 'rf')
print(impute)

impute$imp$day0_MPH_max_wind

#stripplot(impute,pch=20, cex=1.2)

xyplot(impute,day0_MPH_max_wind ~ day0_F_Mean_dew_point | .imp, pch=20, cex=1.4)

new_dat<-complete(impute,1)
```

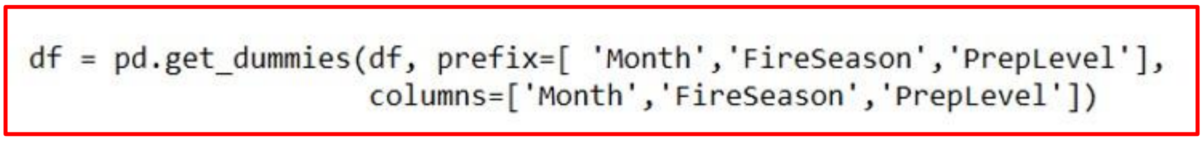
Figure 7: Missing value imputation through Mice

3.2 Feature Engineering

Feature engineering has been done, to improve the performance of the model, reduce the effect of large values over small and bring them on a common scale. Various Feature engineering technique used in the research are as follows :

3.2.1 One-hot Encoding

All the categorical values have been converted to binary encoded attributes using get_dummies() in pandas dataframe. The same is illustrated in Figure 8.

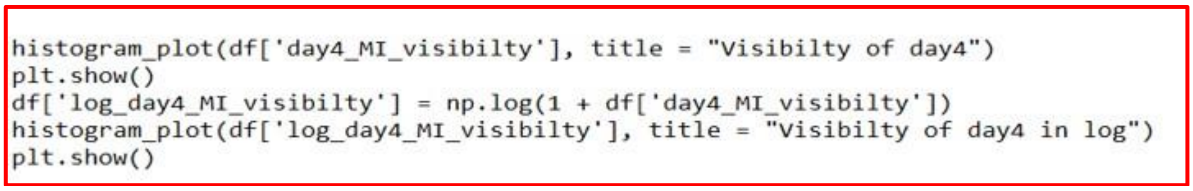


```
df = pd.get_dummies(df, prefix=[ 'Month','FireSeason','PrepLevel'],
                    columns=['Month','FireSeason','PrepLevel'])
```

Figure 8: Implementation of One-Hot Encoding

3.2.2 Logarithmic Transformation

Few of the variables are highly skewed, therefore the values are converted to logarithmic scale to achieve a normal distribution curve. The code snippet is shown in Figure 9.

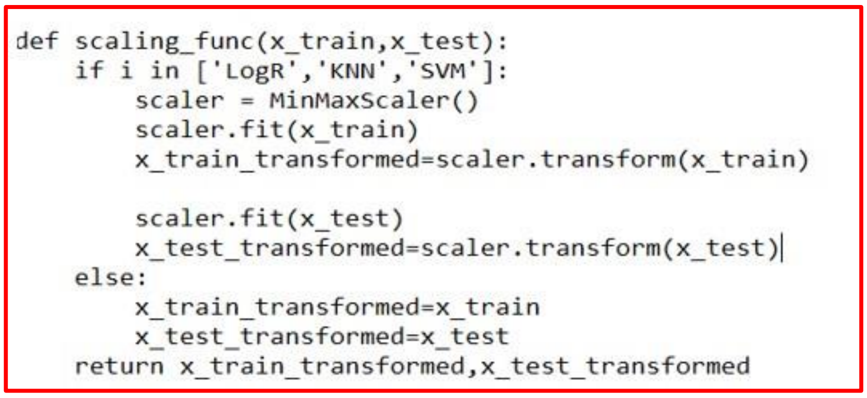


```
histogram_plot(df['day4_MI_visibilty'], title = "Visibilty of day4")
plt.show()
df['log_day4_MI_visibilty'] = np.log(1 + df['day4_MI_visibilty'])
histogram_plot(df['log_day4_MI_visibilty'], title = "Visibilty of day4 in log")
plt.show()
```

Figure 9: Logarithmic Transformation

3.2.3 Scaling of Data

All the data points have been brought to a common scale with a predefined class-MinMaxScaler in Python. The code snippet is shown in Figure 10.



```
def scaling_func(x_train,x_test):
    if i in ['LogR','KNN','SVM']:
        scaler = MinMaxScaler()
        scaler.fit(x_train)
        x_train_transformed=scaler.transform(x_train)

        scaler.fit(x_test)
        x_test_transformed=scaler.transform(x_test)
    else:
        x_train_transformed=x_train
        x_test_transformed=x_test
    return x_train_transformed,x_test_transformed
```

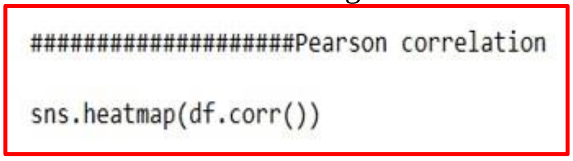
Figure 10: Scaling of Data using MinMaxScaler

3.3 Feature Selection

Five different feature selection methods have been tried which gave five different outputs. The output of all different feature selection is used to train a logistic regression and the method which has the highest accuracy has been used in the next steps.

3.3.1 Correlation Matrix

Snippet for correlation matrix has been shown in Figure 11.

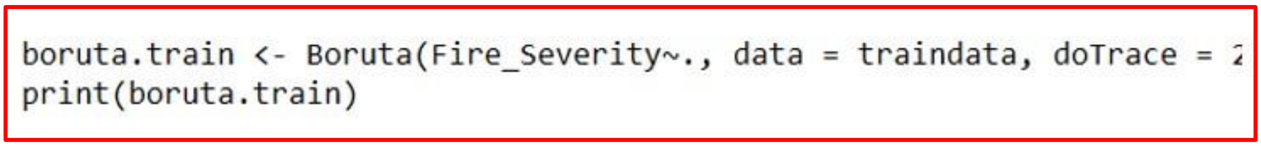


```
####################Pearson correlation

sns.heatmap(df.corr())
```

Figure 11: Correlation Matrix

3.3.2 Boruta Feature Selection

Boruta feature selection method in R has been implemented to find the most important features present in the dataset (snippet is shown in Figure 12).

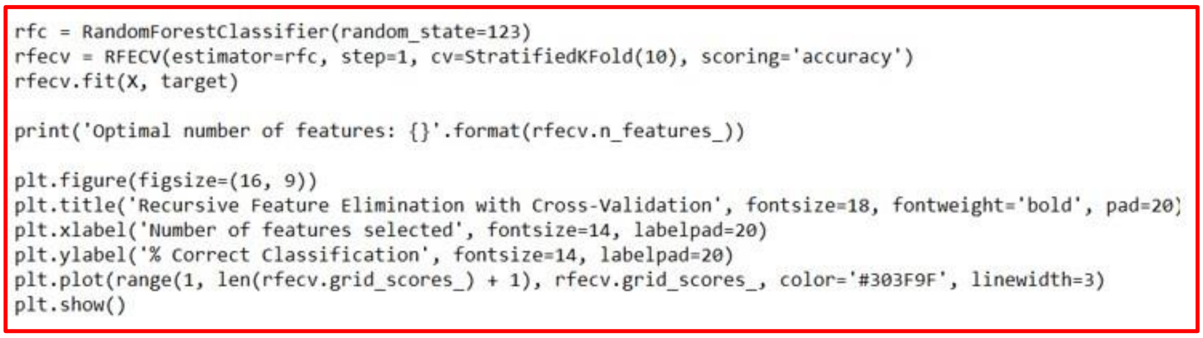


```
boruta.train <- Boruta(Fire_Severity~., data = traindata, doTrace = 2
print(boruta.train)
```

Figure 12: Boruta Feature Selection Method

3.3.3 Recursive Feature Elimination

RFE feature selection method is implemented in Python with Random forest as the base model. Important features are plotted in graph (shown in Figure 13).

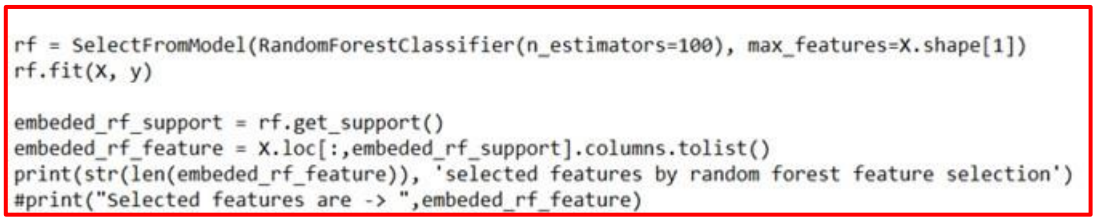


```
rfc = RandomForestClassifier(random_state=123)
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(10), scoring='accuracy')
rfecv.fit(X, target)

print('Optimal number of features: {}'.format(rfecv.n_features_))

plt.figure(figsize=(16, 9))
plt.title('Recursive Feature Elimination with Cross-Validation', fontsize=18, fontweight='bold', pad=20)
plt.xlabel('Number of features selected', fontsize=14, labelpad=20)
plt.ylabel('% Correct Classification', fontsize=14, labelpad=20)
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_, color='#303F9F', linewidth=3)
plt.show()
```

Figure 13: Recursive Feature Elimination

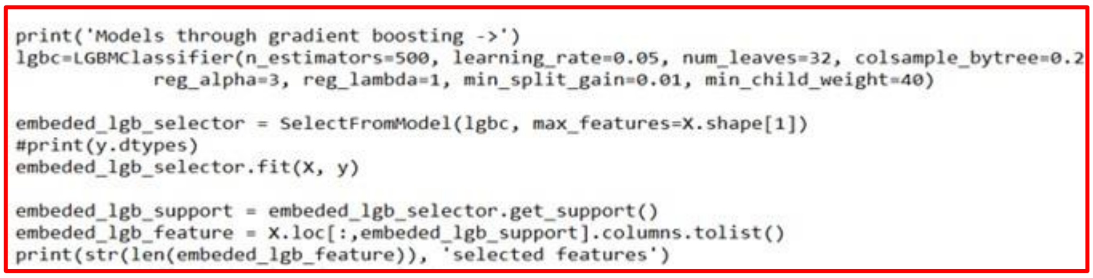
3.3.4 Random Forest Feature Selection

Embedded methods such as Random Forest Feature Selection and lightgbm Feature Selection has been implemented in python. Snippet for RF and LGBMclassifier is shown in Figure14 and Figure 15, respectively.



```
rf = SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=X.shape[1])
rf.fit(X, y)

embeded_rf_support = rf.get_support()
embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features by random forest feature selection')
#print("Selected features are -> ",embeded_rf_feature)
```

Figure 14: Random Forest Feature Selection

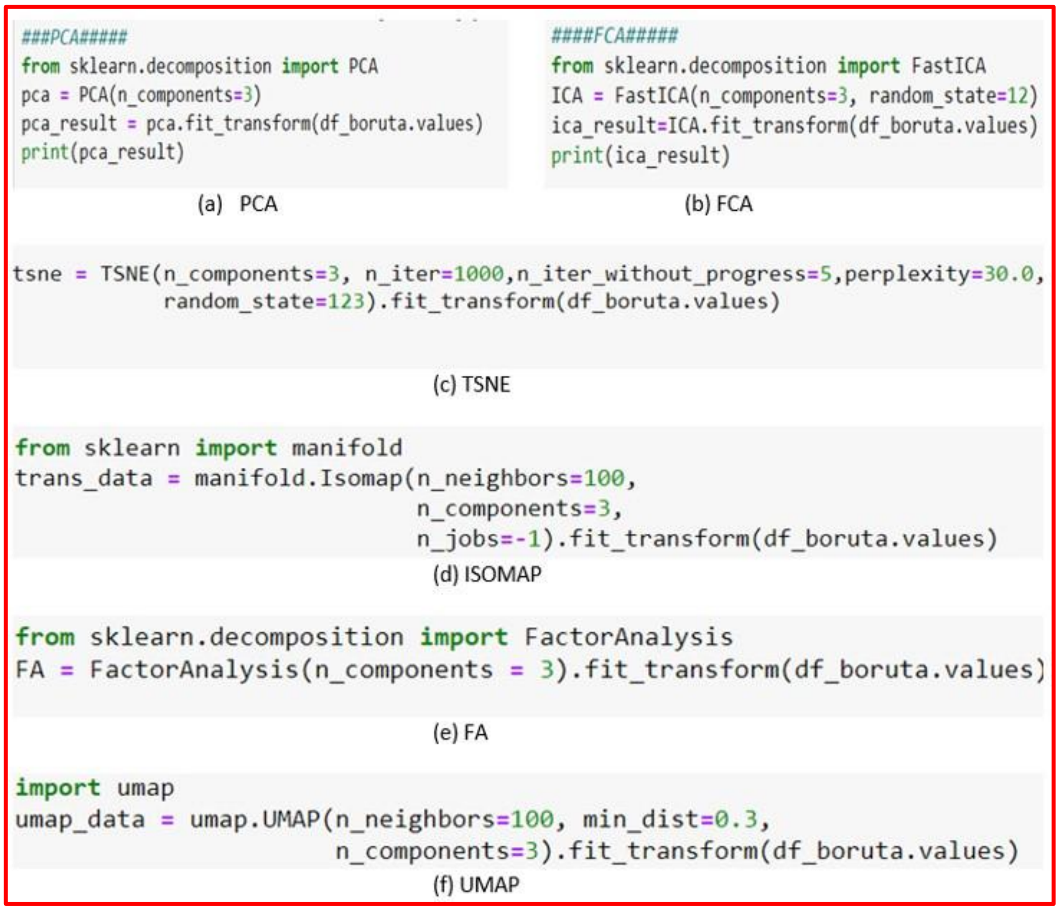


```
print('Models through gradient boosting ->')
lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=32, colsample_bytree=0.2
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)

embeded_lgb_selector = SelectFromModel(lgbc, max_features=X.shape[1])
#print(y.dtypes)
embeded_lgb_selector.fit(X, y)

embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = X.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features')
```

Figure 15: Lightgbm Feature Selection

3.4 Dimension Reduction

Several dimension reduction techniques have been tried and the outputs are fed to logistic regression model and the accuracy of the model is evaluated.



```
###PCA#####
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca_result = pca.fit_transform(df_boruta.values)
print(pca_result)
```

(a) PCA

```
####FCA#####
from sklearn.decomposition import FastICA
ICA = FastICA(n_components=3, random_state=12)
ica_result=ICA.fit_transform(df_boruta.values)
print(ica_result)
```

(b) FCA

```
tsne = TSNE(n_components=3, n_iter=1000,n_iter_without_progress=5,perplexity=30.0,
         random_state=123).fit_transform(df_boruta.values)
```

(c) TSNE

```
from sklearn import manifold
trans_data = manifold.Isomap(n_neighbors=100,
                             n_components=3,
                             n_jobs=-1).fit_transform(df_boruta.values)
```

(d) ISOMAP

```
from sklearn.decomposition import FactorAnalysis
FA = FactorAnalysis(n_components = 3).fit_transform(df_boruta.values)
```

(e) FA

```
import umap
umap_data = umap.UMAP(n_neighbors=100, min_dist=0.3,
                      n_components=3).fit_transform(df_boruta.values)
```

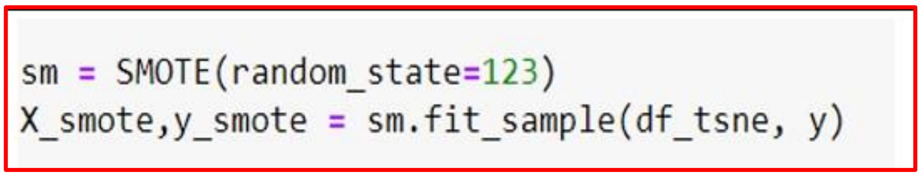
(f) UMAP

Figure 16: Dimension Techniques

The technique with the highest accuracy has been accepted as the final one. Code snippets of all the used techniques are shown in various sub-section in Figure 16.

3.5 SMOTE-Oversampling

SMOTE oversampling method has been applied to balance the dataset. Code snippet has been shown in Figure 17.

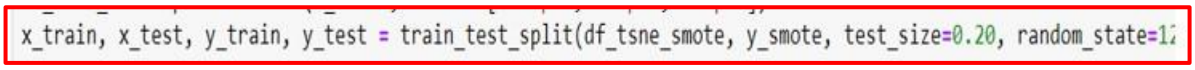


```
sm = SMOTE(random_state=123)
X_smote,y_smote = sm.fit_sample(df_tsne, y)
```

Figure 17: SMOTE Oversampling

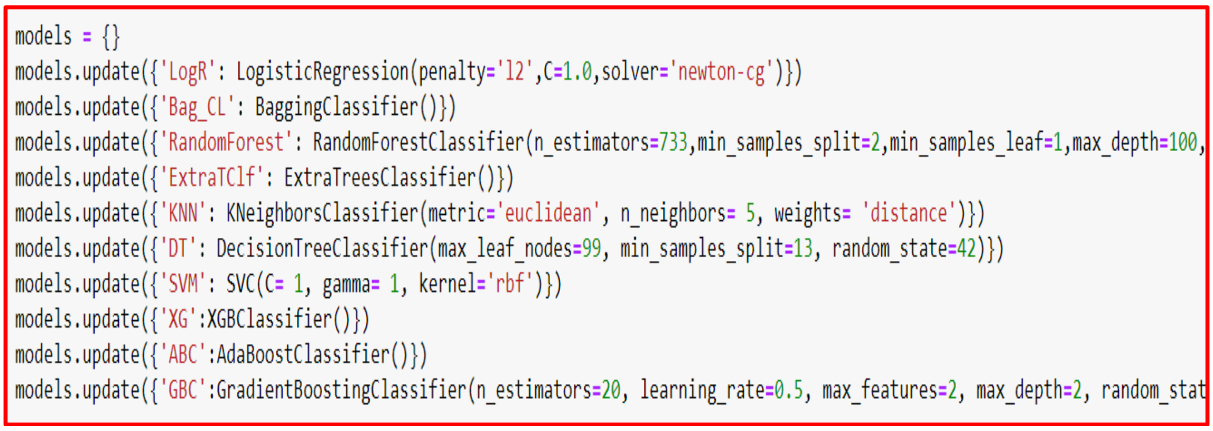
3.6 Data Modelling

Data has been divided into train and test data with 80:20 ratio (shown in Figure 18) and models are trained on the train data and evaluated on the remaining 20%of data. All ML models are stored in a python dictionary (with optimized parameters) which are applied and evaluated sequentially (shown in Figure 19).



```
x_train, x_test, y_train, y_test = train_test_split(df_tsne_smote, y_smote, test_size=0.20, random_state=1
```

Figure 18: Train- Test Split

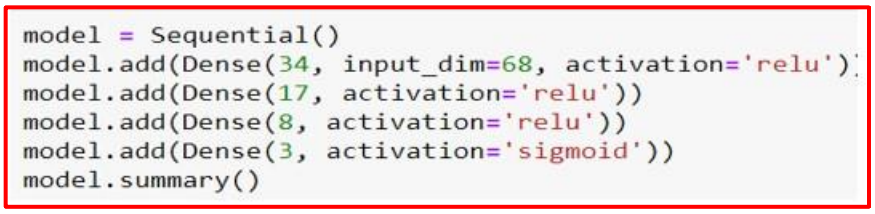


```
models = {}
models.update({'LogR': LogisticRegression(penalty='l2',C=1.0,solver='newton-cg')})
models.update({'Bag_CL': BaggingClassifier()})
models.update({'RandomForest': RandomForestClassifier(n_estimators=733,min_samples_split=2,min_samples_leaf=1,max_depth=100,
models.update({'ExtraTClf': ExtraTreesClassifier()})
models.update({'KNN': KNeighborsClassifier(metric='euclidean', n_neighbors= 5, weights= 'distance')})
models.update({'DT': DecisionTreeClassifier(max_leaf_nodes=99, min_samples_split=13, random_state=42)})
models.update({'SVM': SVC(C= 1, gamma= 1, kernel='rbf')})
models.update({'XG':XGBClassifier()})
models.update({'ABC':AdaBoostClassifier()})
models.update({'GBC':GradientBoostingClassifier(n_estimators=20, learning_rate=0.5, max_features=2, max_depth=2, random_stat
```

Figure 19: Applied Models

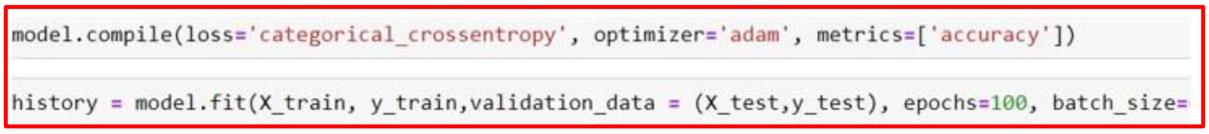
3.6.1 ANN

Google Colab has been used to run ANN and LSTM to leverage the use of GPUs. The code snippet for the model summary of ANN is shown in Figure 20. Model compile and model fit is shown in Figure 21.



```
model = Sequential()
model.add(Dense(34, input_dim=68, activation='relu'))
model.add(Dense(17, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='sigmoid'))
model.summary()
```

Figure 20: Model Summary of ANN

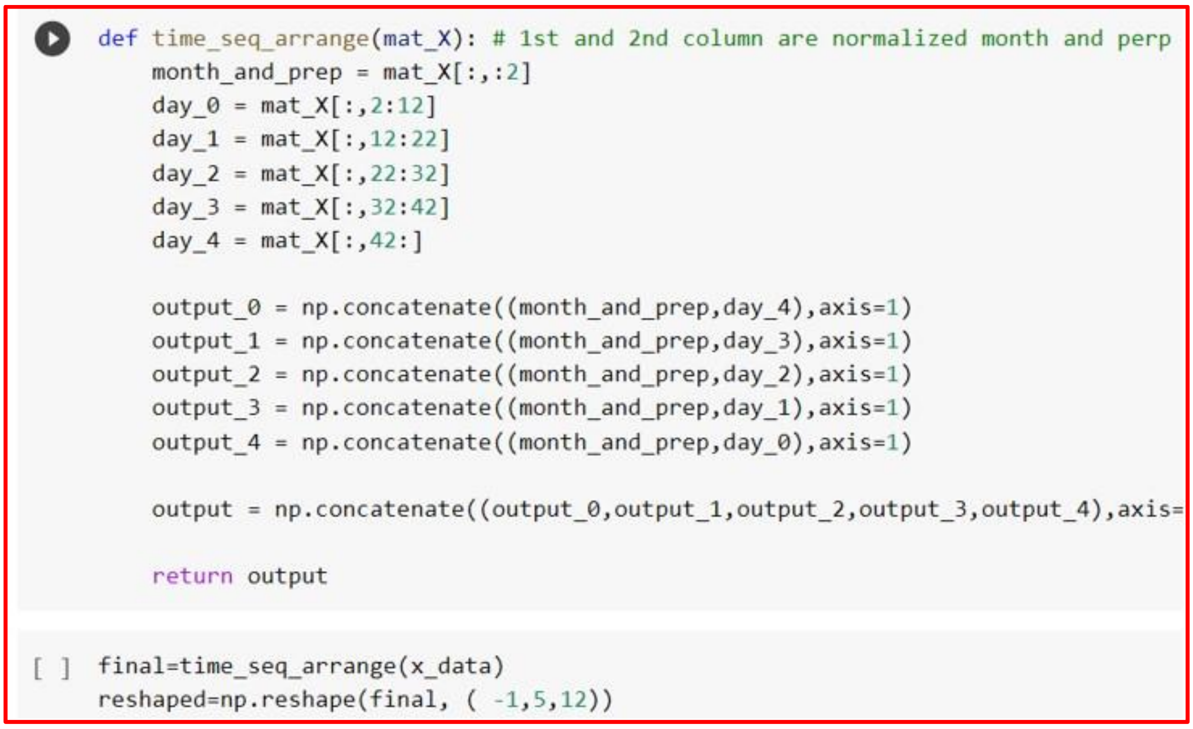


```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, y_train,validation_data = (X_test,y_test), epochs=100, batch_size=
```

Figure 21: Model Compile and Fit

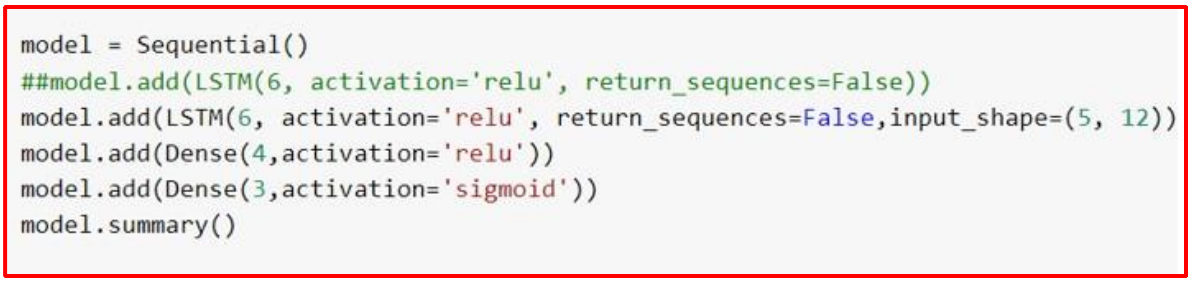
3.6.2 LSTM

Data is converted into a sequence of fixed length and reshaped (shown in Figure 21) before applying LSTM for sequence classification. Model Summary, Compile/fit is shown in Figure 22 and Figure 23, respectively.



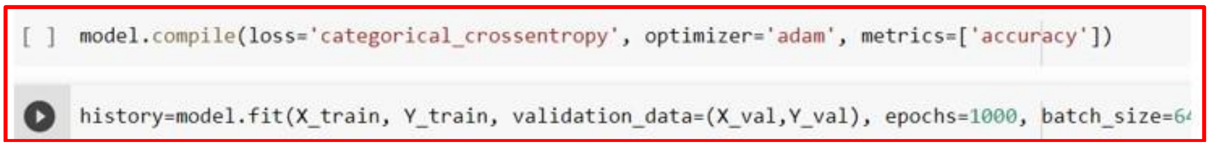
```
def time_seq_arrange(mat_X): # 1st and 2nd column are normalized month and perp
    month_and_prep = mat_X[:,:2]
    day_0 = mat_X[:,2:12]
    day_1 = mat_X[:,12:22]
    day_2 = mat_X[:,22:32]
    day_3 = mat_X[:,32:42]
    day_4 = mat_X[:,42:]

    output_0 = np.concatenate((month_and_prep,day_4),axis=1)
    output_1 = np.concatenate((month_and_prep,day_3),axis=1)
    output_2 = np.concatenate((month_and_prep,day_2),axis=1)
    output_3 = np.concatenate((month_and_prep,day_1),axis=1)
    output_4 = np.concatenate((month_and_prep,day_0),axis=1)

    output = np.concatenate((output_0,output_1,output_2,output_3,output_4),axis=

    return output

[ ] final=time_seq_arrange(x_data)
    reshaped=np.reshape(final, ( -1,5,12))
```

Figure 22: Function for Sequence Creation



```
model = Sequential()
##model.add(LSTM(6, activation='relu', return_sequences=False))
model.add(LSTM(6, activation='relu', return_sequences=False,input_shape=(5, 12))
model.add(Dense(4,activation='relu'))
model.add(Dense(3,activation='sigmoid'))
model.summary()
```

Figure 23: Model Summary of LSTM



```
[ ] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    history=model.fit(X_train, Y_train, validation_data=(X_val,Y_val), epochs=1000, batch_size=6
```

Figure 24: Model Compile and Fit