Stat 154 Final Project Report

Authors: Ashritha Eswaran, Nathan Lam, Mitali Yadav

Part I: Introduction

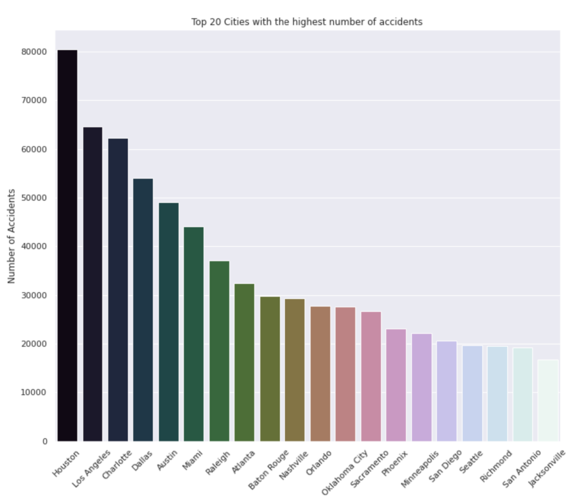
The dangers of driving is a struggle of urban life and usually the driver is at fault, but what about being in the wrong place at the wrong time? In this report, we try to model the severity of traffic accidents in the hope to identify what environmental factors significantly contribute to these accidents.

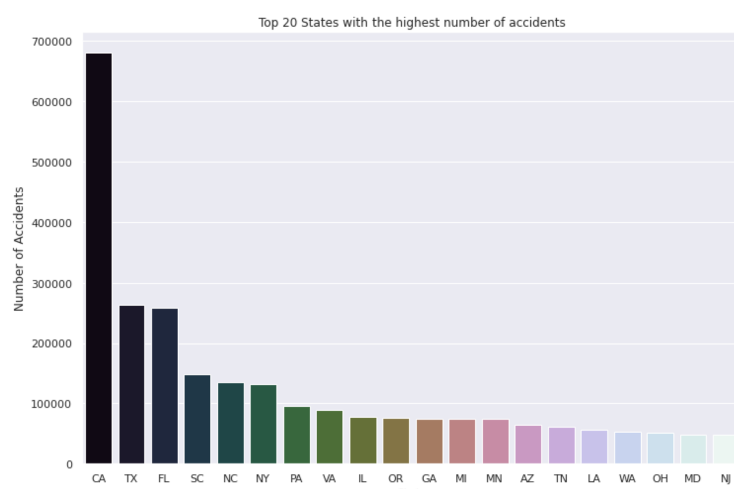
Part II: Background Information on Data

The data set, covering accident information from 2016 to 2020, contains 2,962,779 data points with 49 possible variables and features describing an accident's location and weather condition. The dataset needed some preprocessing as there were issues like unhelpful variables or missing values. In the process of data cleaning, we dropped variables that were deemed as unhelpful, created new and more useful columns from existing ones, filled in missing values using the variable's mean or median, and encoded categorical variables using integers. The list of variables used can be found in Table 1 in **Appendix A** describing each variable.

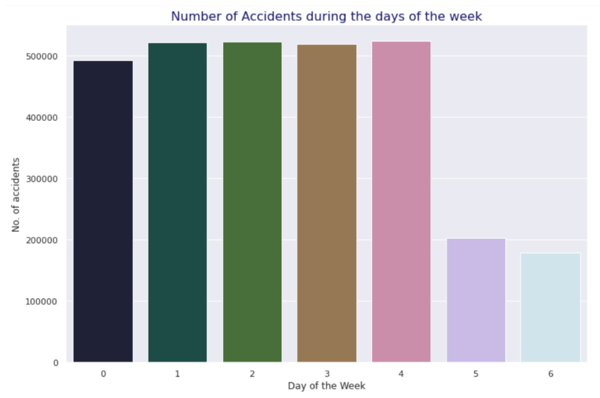
Part III: What We Learned about Data

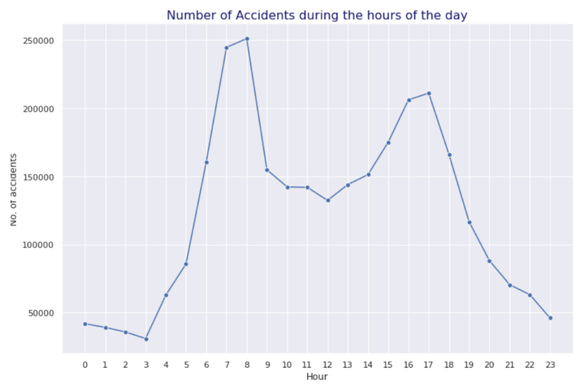
From our exploratory data analysis on the data, we discovered interesting features present in our data that could potentially be important in classifying accidents as severe or not. Out of the 49 variables, 23 of them contained some number of non-zero null values. However, we could not drop all of them, so we chose to drop those columns which had more than 45% of null values. In other instances, there were a small number of values missing for columns such as City, in which case we chose to drop the rows instead. By grouping the data by severity we found there were hotspots of accidents at certain locations and time. The top three most accident prone states are California, Texas, and Florida, but by city they were Houston, Los Angeles, and Charlotte.

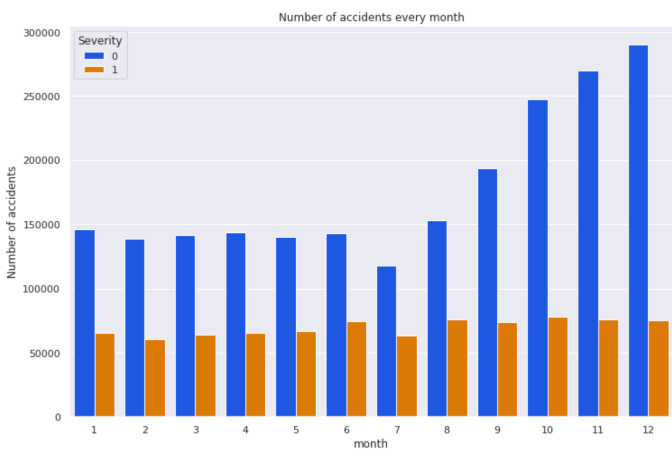




The patterns of an average person's activity can actually be observed. The typical times when someone drives to go to work and then back home, 8am and 6pm, paired with the weekdays being the most prone reflects the schedule someone would expect of an average person.







While the annual number of accidents has been increasing exponentially, this increase was not seen in the severity instances. At the same time, the number of more severe accidents (Severity=1) remained constant through the years, only the number of less severe accidents increased exponentially.

Part IV: Different Models We Considered

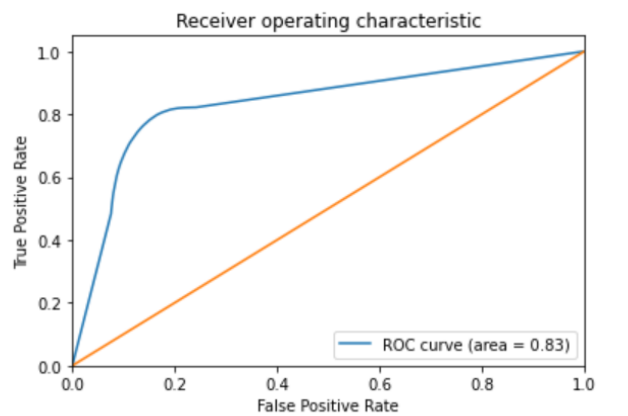
Collectively, we considered 12 unique models, varying from linear methods, tree methods, and neural networks. For the sake of compactness, three models will be discussed. The models are logistic regression, CART, and Random Forest.

Model 1: Logistic Regression

The features for Logistic Regression were found using forward and backward selection using BIC and Mallows' Cp as the criterion. The best model considered start longitude, county, state, crossing, stop, traffic signal, nautical twilight, year, and month. This was based on training 70% of the data and the remaining 30% was used to measure the accuracy of the model. The resulting validation accuracy was **70.3894%**. Immediately, it can be seen that the model did a poor job categorizing the data.

Model 2: Decision Trees

Rather than including numerous features (like in the Logistic Regression Model), we decided to focus on start latitude, start longitude, distance, and wind speed during the accident. The model was trained on 70% of the training data. The other 30% was used for validation purposes. Compared to Model 1, this model improved model accuracy significantly. The resulting validation accuracy was **83.60228%**. Since the training dataset included 2.9 million data points, we considered the decision tree model to be a decent model. The ROC Curve for this model can be seen to the left.

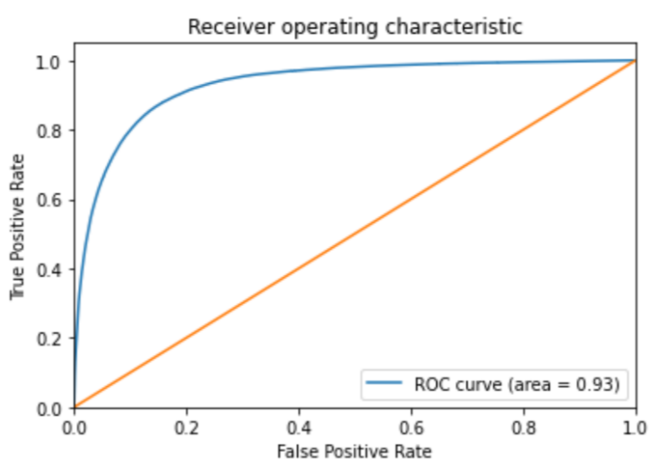


From the figure, the ROC Curve had an area of about 0.83. We were not satisfied with the results of the decision tree since the model accuracy could be improved, and the ROC curve showed us that the model had some overfitting issues.

Since decision trees have low bias and high variance, we decided to try a random forest model to address the issue of overfitting. Random forests are an ensemble method that combines several different decision trees to fit the model. Intuitively, we thought that random forests would improve our model accuracy further.

Model 3: Random Forests

For the random forest model, we only considered the start latitude, start longitude, and distance of the accident. The model was trained on 90% of the training data. The other 10% was used for validation purposes. Compared to Model 1 and 2, this model had the highest model accuracy and ROC curve characteristic. The resulting validation accuracy was **87.2422710%**. Because of this evaluation, we considered the random forest model to be an accurate model in classifying accidents on the scale of severity. The ROC Curve for the random forest model can be seen to the left.



From the figure, the ROC Curve had an area of about 0.93. We were satisfied with the results of the random forest since the model accuracy was relatively high, and the ROC curve indicated a good relationship between the false positive and true positive rates. All the other considered model accuracies are displayed in Table 2 in **Appendix A.**

Part V: Analysis of Results

From our three discussed models described in the previous section, we concluded that the geospatial features: start latitude and start longitude, as well as distance during the accident were the most important features in predicting accident severity. Intuitively, this makes sense because certain areas such as urban locations, particularly in the East and West Coast, had higher number of accidents and more severe accidents each year. As discussed in Part II, certain states and cities had higher frequency of accidents. Thus, the geographical location of the accidents is a key feature in predicting accident severity.

Part VI: What Went Well, What Didn't Work

The biggest issue working with the data was that there were a lot of data points. Running code would often freeze, crash, or take hours to run models due to this limitation. Our personal computers were not optimized in dealing with millions of data points. During the initial EDA stages, we chose to work with Google Colab so that we could conveniently share our code and recreate the results. To create the models, we used both R and Python. Using R was challenging, code would often take at least an hour to run and possibly fail due to not enough memory. In contrast, Python had a much better runtime, but still struggled to handle the size of the data set sometimes. In certain cases, we tried breaking up the data into chunks but found that it affected the accuracy of the model.

Part VII: Conclusion

Working from 49 variables, we utilized 3 to 8 features and considered 12 unique models. The most accurate model was a random forest focused on only 3 geospatial features with a model accuracy of 87.738%. Future attempts should utilize computers with better capability of handling large datasets. This will allow cross validation techniques to be more plausible. For future analysis, we hope to consider Natural Language Processing (NLP) for description analysis of accidents to determine if more accidents occur on a highway rather than on normal streets. Due to the lack of memory space on our local computers, we were unable to perform NLP on numerous data points. Given our limitations, we achieved the best models possible and hope to further improve our models to achieve a higher accuracy. Mainly because accident severity analysis is a very important issue in today's world.

Part VIII: Individual Contributions

Our group worked collectively during the data cleaning and data modeling portions. For the data cleaning part, Mitali analyzed the correlation between the potential variables and Severity variable, and performed Exploratory Data Analysis to get the large dataset ready for data modeling. As a group, each member explored various potential models and measured their accuracy in classifying unseen data. Nathan analyzed 2 different models for each of the following categories: Logistic, Linear, and LDA. Ashritha analyzed different SVM models with polynomial, linear, and rbf kernels. She also analyzed numerous decision trees, random forest classifiers, and bagging classifiers with different hyperparameters in each model. Mitali used Random Forest Classification, Decision Trees and also made an attempt to use k-nearest neighbors. She used as many columns as possible and hence had to use One Hot Encoding which further increased the size of her data which she then was able to reduce using PCA and tried out multiple combinations of hyperparameters. The report was written collectively by everyone.

Appendix A: Tables

Table 1: Variables Considered

Name	Data Type	Description
Severity	Categorical	Level of the traffic accident's severity. 1 = least severe, 4 = most severe
Start_Lat	Numeric	Starting Latitude of accident
Start_Lng	Numeric	Starting Longitude of accident
County	Categorical	County
State	Categorical	State
Crossing	Boolean	Signifies if a crosswalk is present
Stop	Boolean	Signifies if a stop sign is present
Traffic_Signal	Boolean	Signifies if a traffic signal is present
Nautical_Twilight	Boolean	Signifies if weather is nautical twilight
Year	Numeric	Year of accident
Month	Numeric	Month of accident
Hour	Numeric	Hour of accident
Wind_Speed.mph.	Numeric	Wind Speed at the time of accident
Distance.mi.	Numeric	Distance of accident

Table 2: Models Considered

Model	Variables	Accuracy
Logistic Regression	Start_Lng, County, State, Crossing, Junction, Stop, Traffic_Signal, Year, Month	0.70387
Logistic Regression	Start_Lng, County, State, Crossing, Stop, Traffic_Signal, Nautical_Twilight, Year, Month	0.70389
Linear Regression	Start_Lng, County, State, Crossing, Junction, Stop, Traffic_Signal, Year, Month	0.69684
Linear Regression	Start_Lng, County, State, Crossing, Stop, Traffic_Signal, Nautical_Twilight, Year, Month	0.69677
LDA	Start_Lng, County, State, Crossing, Junction, Stop, Traffic_Signal, Year, Month	0.70058
LDA	Start_Lng, County, State, Crossing, Stop, Traffic_Signal, Nautical_Twilight, Year, Month	0.70090
Decision Trees	Start_Lng, Start_Lat, Distance.mi. Wind_Speed.mph.	0.83602
Random Forest	Start_Lng, Start_Lat, Distance.mi.	0.87242
SVM (linear kernel)	Start_Lng, Start_Lat, Distance.mi.	0.71093
SVM (polynomial kernel)	Start_Lng, Start_Lat, Distance.mi.	0.71102
SVM (rbf kernel)	Start_Lng, Start_Lat, Distance.mi.	0.70253
Neural Network	Start_Lng, Start_Lat, Distance.mi.	0.79712

Project Stat 154

Nathan Lam

4/12/2021

```
library(dplyr) #data frame manipulation
library(glmnet) #logistic + penalized methods
library(caret) # cross validation
library(leaps) #cross validation
library(nnet) #multinomial logistic regression
library(MASS) #LDA
library(penalizedLDA)
```

```
start1 <- proc.time()
train <- read.csv('train.csv')
end1 <- proc.time()
print(end1-start1)

start2 <- proc.time()
train <- read.csv('train.csv')
end2 <- proc.time()
print(end2-start2)


#remove_train <- c(-1:-3,-6,-9:-14,-19:-23,-28,-34,-40,-43,-45)
#remove_test <- c(-1:-3,-5,-8:-13,-18:-22,-27,-33,-39,-42,-44)
remove <- c(-1:-4,-12:-14,-19:-23) #include more variables
Severity <- train$Severity

start3 <- proc.time()
train_set <- train[,remove]
end3 <- proc.time()
print(end3-start3)


train_set <- train[,c(-1,-2)]


head(train_cleaned3)
```

```
#clean raw train, test
start_clean <- proc.time()

for(i in 1:ncol(train_set)){if(any(is.na(train_set[,i]))){print(paste(i,names[i]))}}
#remove na in numeric columns
has_na <- c(5,6,12:16,18,19)
```

```r
for(n in has_na){
  train_set[is.na(train_set[,n]),n] <- mean(train_set[,n],na.rm=T)}
train_set$Humidity...[is.na(train_set$Humidity...)] <- round(mean(train_set$Humidity...,na.rm=T))


#encoding R-L as 1-0
train_set$Side[train_set$Side=='R'] <- 1
train_set$Side[train_set$Side=='L'] <- 0


#encoding things in general
names <- colnames(train_set)
for(i in 1:ncol(train_set)){print(paste(i,names[i],length(unique(train_set[,i]))/nrow(train_set)))}
#this for loop has takes almost an hour
for(m in 9:11){
  uniq <- unique(train_set[,m])
  for(h in 1:length(uniq)){
    if(h %% 100 == 0){
      print(uniq[h])
    }
    train_set[train_set[,m]==uniq[h],m] <- h
  }
}



#encoding T-F as 1-0
for(k in 21:33){
  train_set[train_set[,k]=='True',k] <- 1
  train_set[train_set[,k]=='False',k] <- 0
}


#Encoding day-night to 1-0
for(i in 34:37){
train_set[train_set[,i]=='',i] <- 1
train_set[train_set[,i]=='Day',i] <- 1
train_set[train_set[,i]=='Night',i] <- 0
}


#removing redundancies
train_set$Wind_Direction[train_set$Wind_Direction =="North"] <- "N"
train_set$Wind_Direction[train_set$Wind_Direction =="South"] <- "S"
train_set$Wind_Direction[train_set$Wind_Direction =="East"] <- "E"
train_set$Wind_Direction[train_set$Wind_Direction =="West"] <- "W"
train_set$Wind_Direction[train_set$Wind_Direction =="Calm"] <- "CALM"
train_set$Wind_Direction[train_set$Wind_Direction ==""] <- "CALM"
train_set$Wind_Direction[train_set$Wind_Direction =="Variable"] <- "VAR"


#encoding Wind direction as numbers
wind_dir <- unique(train_set$Wind_Direction)
```

```r
for(h in 1:length(wind_dir)){
  train_set$Wind_Direction[train_set$Wind_Direction==wind_dir[h]] <- h
}


#encoding wind condition
wind_cond <- unique(train_set$Weather_Condition)
for(h in 1:length(wind_cond)){
  train_set$Weather_Condition[train_set$Weather_Condition==wind_cond[h]] <- h
}


end_clean <- proc.time()
print(end_clean-start_clean)



Year <- substr(train_set$Start_Time,1,4)
Month <- substr(train_set$Start_Time,6,7)
Day <- substr(train_set$Start_Time,9,10)
Hour <- substr(train_set$Start_Time,12,13)


#names <- colnames(train_set)
#for(i in 1:ncol(train_set)){
#  print(paste(i,names[i],any(is.na(train_set[,i]))))
#}

train_cleaned2 <- cbind(train_set[,c(-1,-2,-5,-6)],Year,Month,Day,Hour)
train_cleaned3 <- cbind(Severity,train_cleaned2)

write.csv(train_cleaned3,"train_cleaned2.csv")
```

```r
#using penalized methods
#ran out of memory to run

sub_set <- sample(1:nrow(train_cleaned3),nrow(train_cleaned3)*0.7)
sub_train <- train_cleaned3[sub_set,]
sub_test <- train_cleaned3[-sub_set,]


lasso.cv <- cv.glmnet(as.matrix(sub_train),as.matrix(Severity[sub_set]),
                      type.measure="mse", family="gaussian", alpha=1,nfold=5,trace.it=T)

plot(lasso.cv)


lasso <- glmnet(as.matrix(sub_train),as.matrix(Severity[sub_set]),
                      type.measure="mse", family="gaussian", alpha=1,
                lambda = lasso.cv$lambda.1se)

summary(lasso)
```

```r
####

lasso.cv <- cv.glmnet(as.matrix(sub_train),as.matrix(Severity[sub_set]),
                      type.measure="mse", family="multinomial", alpha=1)

plot(lasso.cv)


lasso <- glmnet(as.matrix(sub_train),as.matrix(Severity[sub_set]),
                      type.measure="mse", family="multinomial", alpha=1,
                lambda = lasso$lambda.1se))

summary(lasso)
```

```r
#########Forward Selection#########
f_model <- regsubsets(x=as.matrix(sub_train[,c(-1,-2)]),y=as.factor(sub_train[,2]),
                      method = "forward", nbest = 1) %>% summary()

#extracting criterion
f_BIC <- f_model$bic
f_mallow_Cp <- f_model$cp

#picking best variables
f_BIC_picked <- f_model$which[which.min(f_BIC),]
f_cp_picked <- f_model$which[which.min(f_mallow_Cp),]

#printing picked variables
f_BIC_picked[f_BIC_picked == TRUE]
f_cp_picked[f_cp_picked == TRUE]


#########Backward Selection#########
b_model <- regsubsets(x=as.matrix(sub_train[,c(-1,-2)]),y=as.matrix(sub_train[,2]),
                      method = "backward", nbest = 1) %>% summary()

#extracting criterion
b_BIC <- b_model$bic
b_mallow_Cp <- b_model$cp

#picking best variables
b_BIC_picked <- b_model$which[which.min(b_BIC),]
b_cp_picked <- b_model$which[which.min(b_mallow_Cp),]

#printing picked variables
b_BIC_picked[b_BIC_picked == TRUE]
b_cp_picked[b_cp_picked == TRUE]
```

models selected (y using 60% of data)
linear dependencies found
f_BIC: intercept + Start_Lng + county + State + Crossing + Junction + Stop
+ Traffic_Signal + Year
f_mallow_cp: intercept + Start_Lng + county + State + Crossing + Junction + Stop
+ Traffic_Signal + Year

b__BIC: intercept + city
b__mallow__cp: intercept + start__lat + city + county + give__way + year

models selected (y using 70% of data)
linear dependencies found
f__BIC: intercept + Start__Lng + county + State + Crossing + Junction + Stop + Traffic__Signal + Year + Month
f__mallow__cp: intercept + Start__Lng + county + State + Crossing + Junction + Stop + Traffic__Signal + Year + Month
b__BIC: intercept + Start__Lng + county + State + Crossing + Stop + Traffic__Signal + Nautical__Twilight + Year + Month
b__mallow__cp: intercept + Start__Lng + county + State + Crossing + Stop + Traffic__Signal + Nautical__Twilight + Year + Month

```
#trying to CV to find RMSE
tc <- trainControl(method = "cv", number = 5)

model1_cv <- train(as.factor(sub_train[,1]) ~ Start_Lat + City + County +
                     Give_Way + No_Exit + Year + Month,
                   data = as.data.frame(sub_train),
                   method="glm",trControl=tc,family="multinomial")
model1_cv$results[,"RMSE"]

###

model2_cv <- train(as.factor(sub_train[,1]) ~ City,
                   data = as.data.frame(sub_train),
                   method="glm",trControl=tc,family="multinomial")
model2_cv$results[,"RMSE"]

###

model3_cv <- train(as.factor(sub_train[,1]) ~ Start_Lat + City + County + Give_Way + Year,
                   data = as.data.frame(sub_train),
                   method="glm",trControl=tc,family="multinomial")
model3_cv$results[,"RMSE"]

#could not cross validate
```

```
#not cross validating
#trained on 60% of the data

features <- c(4,7,8,18,20,24,25,30)
#model1 <- glmnet(as.matrix(sub_train[,features]),as.matrix(sub_train[,2]),
#family="multinomial")
model1a <- multinom(Severity ~ Start_Lng + County + State + Crossing + Junction +
                  Stop + Traffic_Signal + Year,data=sub_train)
model1_pred <- predict(model1a,newdata=sub_test[,features],"class")
paste('Model 1 MsE:',mean(model1_pred==sub_test[,2]))

###

features <- c(3,6,7,19,30)
#model2 <- glmnet(as.matrix(sub_train[,features]),as.matrix(sub_train[,2]),
#type.measure="mse", family="multinomial")
```

```r
model2a <- multinom(Severity ~ Start_Lat + City + County + Give_Way + Year,data=sub_train)
model2_pred <- predict(model2a,newdata=sub_test[,features])
paste('Model 2 MsE:',mean(model2_pred==sub_test[,2]))

###

features <- c(6,30)
#model3 <- glmnet(as.matrix(sub_train[,features]),as.matrix(sub_train[,2]),
#type.measure="mse", family="multinomial")
model3a <- multinom(Severity ~ City + Year,data=sub_train)
model3_pred <- predict(model3a,newx=sub_test[,features])
paste('Model 3 MsE:',mean(model3_pred==sub_test[,2]))
```

```r
#not cross validating
#trained on 70% of the data

features <- c(3,6,7,17,19,23,24,29,30)
#model1 <- glmnet(as.matrix(sub_train[,features]),as.matrix(sub_train[,2]), family="multinomial")
model1a <- multinom(Severity[sub_set] ~ Start_Lng + County + State + Crossing + Junction +
                 Stop + Traffic_Signal + Year + Month,
                 data=as.data.frame(sub_train2[sub_set,features]))
model1_pred <- predict(model1a,newdata=sub_train2[-sub_set,features],"class")
paste('Model 1 MsE:',mean(model1_pred==Severity[-sub_set])) #0.703872714140098 accuracy



model4 <- lm(Severity[sub_set] ~ Start_Lng + County + State + Crossing + Junction +
                 Stop + Traffic_Signal + Year + Month,
             data=as.data.frame(sub_train2[sub_set,features]))
model4_pred <- predict(model4,newdata=data.frame(sub_train2[-sub_set,features]))
paste('Model 4 MsE:',mean(round(model4_pred)==Severity[-sub_set])) #0.696835404586233 accuracy



LDA_model <- lda(Severity[sub_set] ~ Start_Lng + County + State + Crossing + Junction +
                 Stop + Traffic_Signal + Year + Month,
                 data=data.frame(sub_train2[sub_set,features]))
LDA_pred <- predict(LDA_model, data.frame(sub_train2[-sub_set,features]))
paste("LDA accuracy:",mean(LDA_pred$class == Severity[-sub_set])) #0.700576260584091 accuracy

###

features <- c(3,6,7,17,23,24,27,29,30)
#model2 <- glmnet(as.matrix(sub_train[,features]),as.matrix(sub_train[,2]),
#type.measure="mse", family="multinomial")
model2a <- multinom(Severity[sub_set] ~ Start_Lng + County + State + Crossing +
                 Stop + Traffic_Signal + Nautical_Twilight + Year + Month,
                 data=as.data.frame(sub_train2[sub_set,features]))
model2a_pred <- predict(model2a,newdata=sub_train2[-sub_set,features])
paste('Model 2 MsE:',mean(model2a_pred==Severity[-sub_set])) #0.703894090460086 accuracy
```

```r
model5 <- lm(Severity[sub_set] ~ Start_Lng + County + State + Crossing +
                 Stop + Traffic_Signal + Nautical_Twilight + Year + Month,
             data=as.data.frame(sub_train2[sub_set,features]))
model5_pred <- predict(model5,newdata=data.frame(sub_train2[-sub_set,features]))
paste('Model 5 MsE:',mean(round(model5_pred)==Severity[-sub_set])) #0.696765650278905 accuracy



LDA_model <- lda(Severity[sub_set] ~ Start_Lng + County + State + Crossing +
                 Stop + Traffic_Signal + Nautical_Twilight + Year + Month,
                 data=data.frame(sub_train2[sub_set,features]))
LDA_pred <- predict(LDA_model, data.frame(sub_train2[-sub_set,features]))
paste("LDA accuracy:",mean(LDA_pred$class == Severity[-sub_set])) #0.700904780870219 accuracy
```

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
sns.set()
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [3]:

```
#uploading the training dataset
acc_train = pd.read_csv("/content/drive/MyDrive/final_project/train.csv")
acc_train.head()
```

Out[3]:

	ID	Source	TMC	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_
0	A-2017855	MapQuest	201.0	2	2018-07-19 20:30:23	2018-07-19 21:14:11	34.153896	-118.275482	I
1	A-3340193	Bing	NaN	2	2020-12-27 13:22:48	2020-12-27 15:02:42	40.261747	-75.250020	40.263
2	A-3274372	Bing	NaN	2	2020-12-19 20:27:52	2020-12-19 22:23:39	29.980875	-90.073829	29.981
3	A-2782559	Bing	NaN	3	2016-09-27 17:29:27	2016-09-27 23:29:27	39.018870	-77.102890	39.019
4	A-3722269	Bing	NaN	2	2020-02-11 19:22:00	2020-02-11 23:22:00	45.743940	-120.175670	45.743

In [4]:

```
acc_train.shape
```

Out[4]:

```
(2962779, 49)
```

In [5]:

```
acc_train.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%
TMC	1901648.0	208.348254	21.236047	200.00000	201.00000	201.00000
Severity	2962779.0	2.305150	0.533466	1.00000	2.00000	2.00000
Start_Lat	2962779.0	36.397790	4.964845	24.55527	33.52025	35.82550
Start_Lng	2962779.0	-95.466527	17.354510	-124.62380	-117.35660	-90.01895
End_Lat	1061131.0	36.898592	5.166983	24.57018	33.85400	37.35016
End_Lng	1061131.0	-98.597794	18.495270	-124.49780	-118.20730	-94.39028
Distance.mi.	2962779.0	0.336481	1.616485	0.00000	0.00000	0.00000
Number	1081417.0	6131.773288	12495.013510	1.00000	898.00000	2893.00000
Temperature.F.	2899590.0	61.488264	18.523924	-89.00000	49.00000	63.00000
Wind_Chill.F.	1634781.0	54.903686	22.720811	-89.00000	38.00000	58.00000
Humidity...	2895631.0	65.663194	22.735045	1.00000	49.00000	68.00000
Pressure.in.	2909032.0	29.693540	0.864541	0.00000	29.64000	29.93000
Visibility.mi.	2893495.0	9.112050	2.818845	0.00000	10.00000	10.00000
Wind_Speed.mph.	2626594.0	7.903770	5.339578	0.00000	4.60000	7.00000
Precipitation.in.	1516143.0	0.012379	0.163746	0.00000	0.00000	0.00000

In [6]:

```
#checking how many nulls in each column
nulls_df = acc_train.isnull().sum(axis = 0).to_frame()
nulls_df =  nulls_df.sort_values(by=0, ascending=False)
nulls_df[0] = round(nulls_df[0]/len(acc_train),3)
nulls_df
```

Out[6]:

	0
End_Lng	0.642
End_Lat	0.642
Number	0.635
Precipitation.in.	0.488
Wind_Chill.F.	0.448
TMC	0.358
Wind_Speed.mph.	0.113
Visibility.mi.	0.023
Weather_Condition	0.023
Humidity...	0.023
Temperature.F.	0.021
Wind_Direction	0.020
Pressure.in.	0.018
Weather_Timestamp	0.015
Airport_Code	0.002
Timezone	0.001
Zipcode	0.000
Sunrise_Sunset	0.000
Civil_Twilight	0.000
Nautical_Twilight	0.000
Astronomical_Twilight	0.000
City	0.000
Description	0.000
Country	0.000
Junction	0.000
Severity	0.000
Start_Time	0.000
End_Time	0.000
Turning_Loop	0.000
Traffic_Signal	0.000
Traffic_Calming	0.000
Stop	0.000
Station	0.000
Roundabout	0.000
Railway	0.000
No_Exit	0.000
Give_Way	0.000

	0
State	0.000
Crossing	0.000
Bump	0.000
Amenity	0.000
Start_Lat	0.000
Start_Lng	0.000
Distance.mi.	0.000
Street	0.000
Side	0.000
Source	0.000
County	0.000
ID	0.000

In [7]:

```
#dropping the following columns
cols_todrop = ['Number', 'Airport_Code' ,'End_Lat', 'End_Lng' ,'Weather_Timestamp' , 'T
MC' ,
                 'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight',
                 'Country','ID', 'Source','Timezone']
acc_train.drop(columns=cols_todrop, inplace=True)
```

In [8]:

```
#changing the severity column so that anything
acc_train['Severity'] = (acc_train['Severity'] > 2) * 1
```

In [9]:

```
#for the column city, it has 101 rows missing which is a small number compared to the t
otal number of rows and it is going to be crucial for analysis later.
#same goes for sunrise_sunset column and description
acc_train.dropna(subset=['Sunrise_Sunset','City','Description'],inplace=True)
```

In [10]:

```
#getting some additional columns which may/may not be used
acc_train['Start_Time'] = pd.to_datetime(acc_train['Start_Time'])
acc_train['End_Time'] = pd.to_datetime(acc_train['End_Time'])
acc_train['day_of_Week'] = acc_train['Start_Time'].dt.dayofweek
acc_train['month'] = pd.DatetimeIndex(acc_train['Start_Time']).month
acc_train['year'] = pd.DatetimeIndex(acc_train['Start_Time']).year
acc_train['Hour'] = acc_train['Start_Time'].dt.hour
```

In [11]:

```
#Finding the top 10 cities and top 10 states with the maximum number of accidents
city_df = acc_train[['City','State' , 'Severity']]
by_state = city_df.groupby('State').agg({'City' : 'count'}).sort_values(by='City',ascen
ding=False)
by_state.rename(columns={"City":"count"},inplace=True)
by_state
```

Out[11]:

	count
State	
CA	680772
TX	263389
FL	259261
SC	148563
NC	135593
NY	132605
PA	95472
VA	89540
IL	78036
OR	75787
GA	75090
MI	74219
MN	73800
AZ	65306
TN	61889
LA	56381
WA	52355
OH	51134
MD	48789
NJ	48421
OK	45451
UT	40910
AL	39960
CO	37651
MA	30820
MO	27950
IN	26857
CT	22813
NE	17615
KY	17226
WI	15025
IA	9803
RI	8999
NV	8429
KS	6821
NH	6012

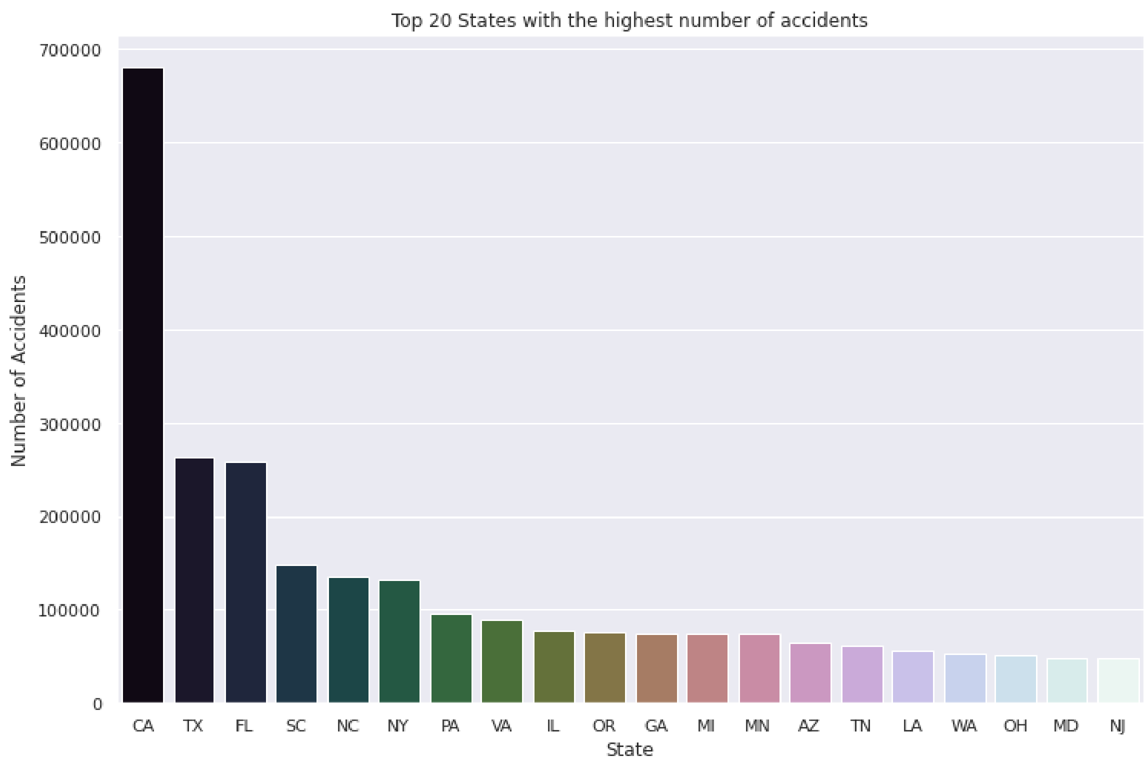
	count
State	
MS	5675
DE	4808
DC	4512
NM	4481
AR	3540
ID	2986
WV	2556
MT	2354
ME	1667
VT	503
WY	373
ND	314
SD	161

In [12]:

```
#viewing the top 20 states.
state_graph = by_state.iloc[0:20,:]
plt.figure(figsize=(12,8))
sns.barplot(data=state_graph, y='count',x=state_graph.index, palette='cubehelix')
plt.ylabel("Number of Accidents")
plt.title("Top 20 States with the highest number of accidents")
```

Out[12]:

```
Text(0.5, 1.0, 'Top 20 States with the highest number of accidents')
```



In [13]:

```
#finding the order of most accident-prone cities
stat_df = acc_train[['State' , 'City','Severity']].groupby('City').count()
stat_df = stat_df.rename(columns = {'Severity' : 'Count'}).drop(columns=['State'],inpla
ce=False)
by_city = stat_df.sort_values(by='Count' , ascending = False)
by_city
```

Out[13]:

	Count
City	
Houston	80475
Los Angeles	64674
Charlotte	62284
Dallas	53993
Austin	49097
...	...
Fort Laramie	1
Midway Park	1
Turon	1
Sigourney	1
Rockleigh	1

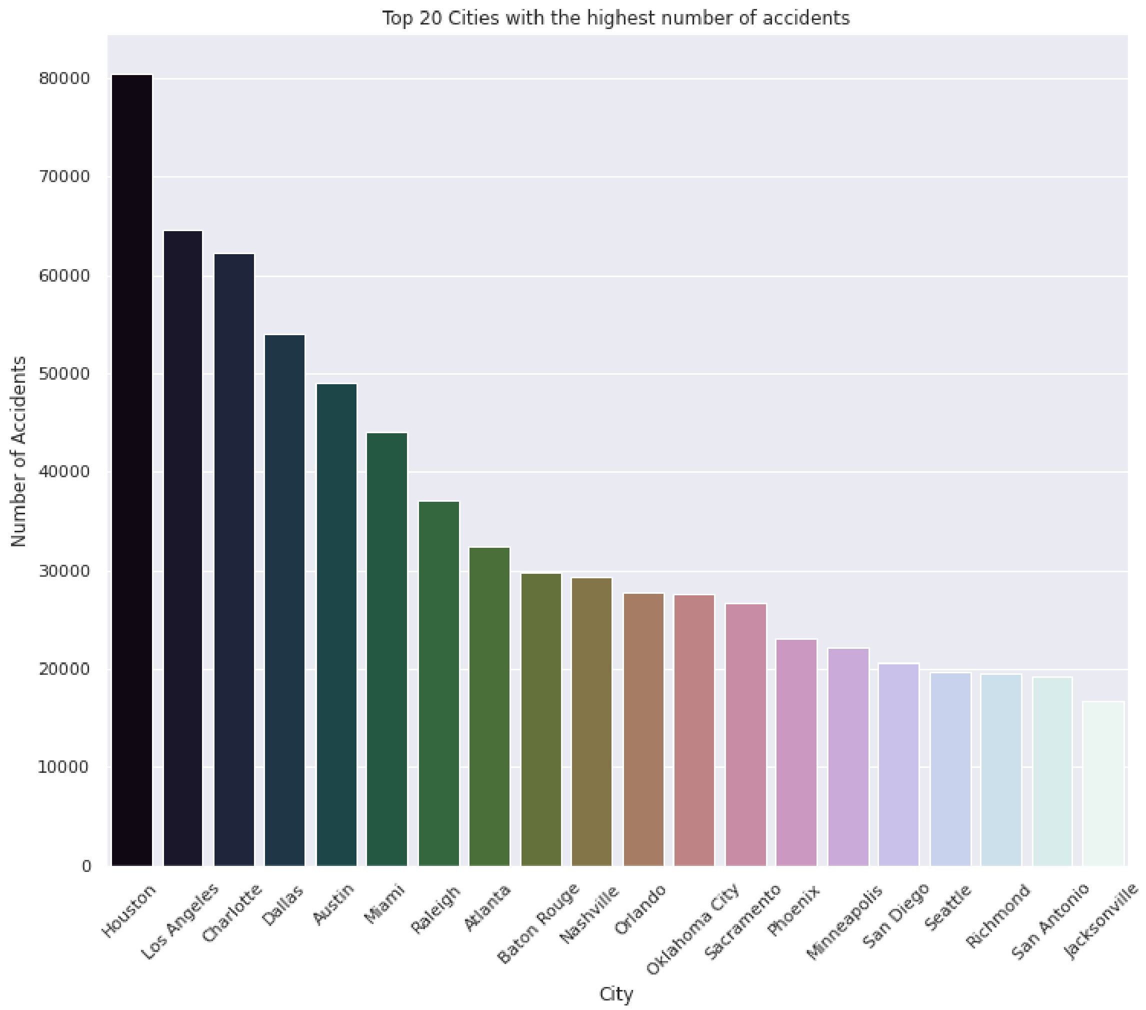
11766 rows × 1 columns

In [14]:

```
city_graph = by_city.iloc[0:20,:]
plt.figure(figsize=(12,10))
sns.barplot(data=city_graph, y='Count',x=city_graph.index, palette='cubehelix')
plt.xticks(rotation=45)
plt.ylabel("Number of Accidents")
plt.title("Top 20 Cities with the highest number of accidents")
```

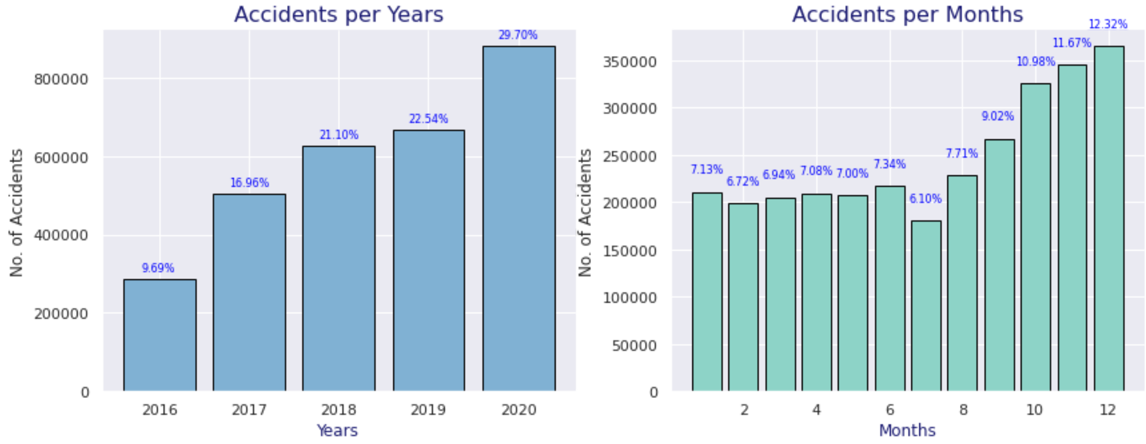
Out[14]:

```
Text(0.5, 1.0, 'Top 20 Cities with the highest number of accidents')
```



In [15]:

```
#plotting the frequencies of the accidents throughout different times
fig,(axis1,axis2) = plt.subplots(1,2,figsize=(14,5))
# plot for year

light_palette = sns.color_palette(palette='Set3')

year_color_map = [light_palette[4] for _ in range(5)]


year = acc_train['year'].value_counts()
years = axis1.bar(year.index.values , year, color=year_color_map , edgecolor = 'black')
axis1.spines[('top')].set_visible(False)
axis1.spines[('right')].set_visible(False)
axis1.set_xlabel("Years", fontdict = {'fontsize':12 , 'color':'MidnightBlue'} )
axis1.set_ylabel("No. of Accidents")
axis1.set_title('Accidents per Years', fontdict = {'fontsize':16 , 'color':'MidnightBlu
e'})

for p in axis1.patches :
    axis1.text(p.get_x() + p.get_width()/2,
            p.get_height() + 20000,
            '{:.2f}%'.format(p.get_height()/len(acc_train)*100),
            ha = "center",
            fontsize = 8, color='Blue')

#for month
month_color_map = [light_palette[0] for _ in range(5)]

month = acc_train['month'].value_counts()
months = axis2.bar(month.index.values , month, color=month_color_map , edgecolor = 'bla
ck')
axis2.spines[('top')].set_visible(False)
axis2.spines[('right')].set_visible(False)
axis2.set_xlabel("Months", fontdict = {'fontsize':12 , 'color':'MidnightBlue'} )
axis2.set_ylabel("No. of Accidents")
axis2.set_title('Accidents per Months', fontdict = {'fontsize':16 , 'color':'MidnightBl
ue'})

for p in axis2.patches :

    axis2.text(p.get_x() + p.get_width()/2,
            p.get_height() + 20000,
            '{:.2f}%'.format(p.get_height()/len(acc_train)*100),
            ha = "center",
            fontsize = 8, color='Blue')
```

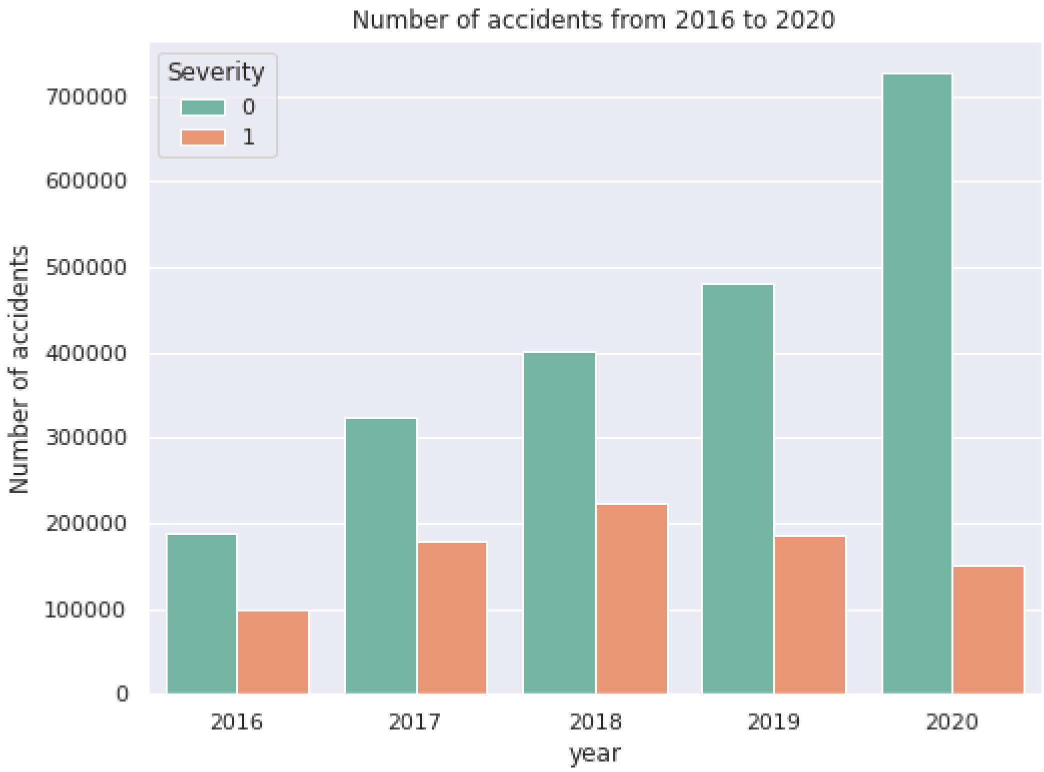


In [16]:

```
#checking growth in number of accidents for severity 0 and 1
plt.figure(figsize=(8,6))
sns.countplot(data=acc_train, x='year', hue='Severity', palette='Set2')
plt.ylabel('Number of accidents')
plt.title('Number of accidents from 2016 to 2020')
```

Out[16]:

```
Text(0.5, 1.0, 'Number of accidents from 2016 to 2020')
```

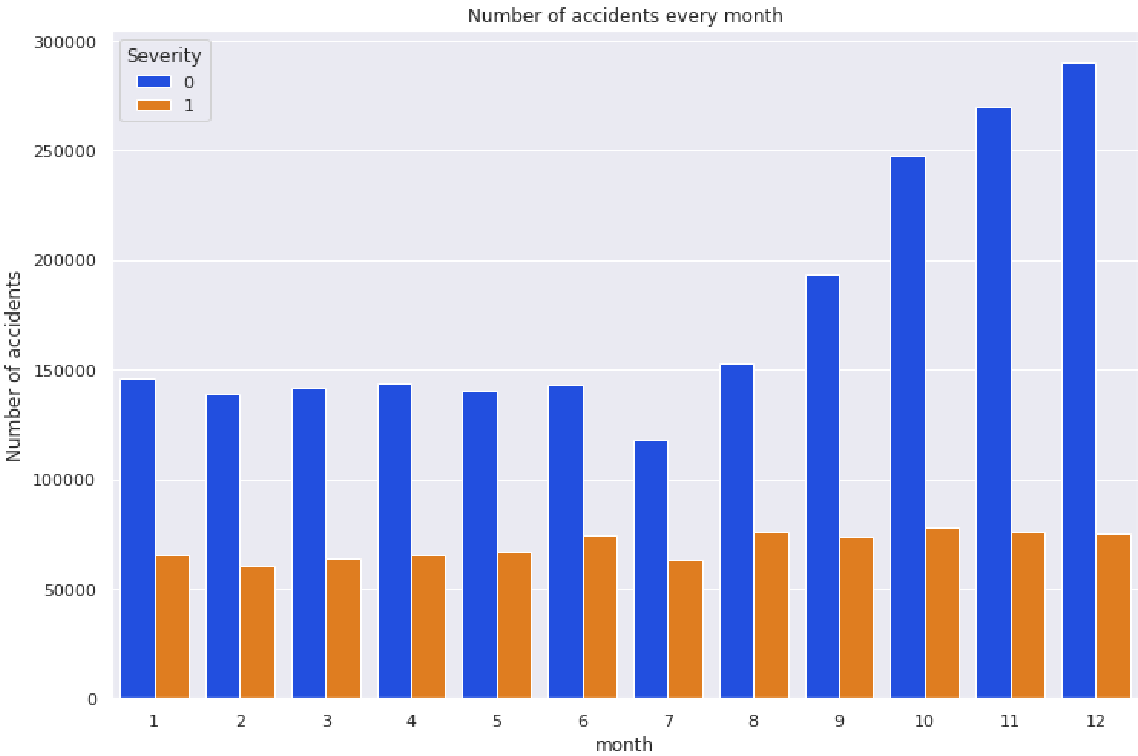


In [ ]:

```
#checking growth in number of accidents for severity 0 and 1
plt.figure(figsize=(12,8))
sns.countplot(data=acc_train, x='month', hue='Severity', palette='bright')
plt.ylabel('Number of accidents')
plt.title('Number of accidents every month')
```

Out[ ]:

```
Text(0.5, 1.0, 'Number of accidents every month')
```

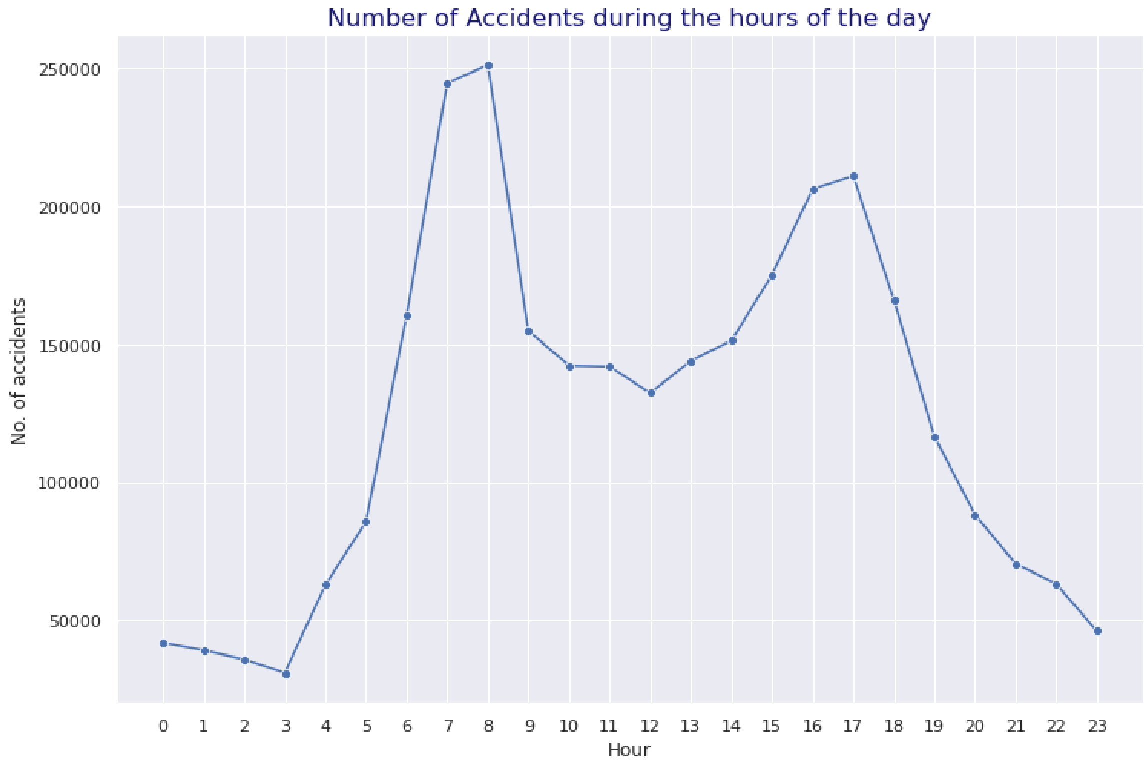


In [ ]:

```
#hours
hour = acc_train['Hour'].value_counts().to_frame()
hour_labels = np.arange(24)

plt.figure(figsize=(12,8))
sns.lineplot(
    data=hour,
    x=hour.index, y="Hour", marker='o', dashes=False
)
plt.xlabel("Hour")
plt.xticks(hour_labels)
plt.ylabel("No. of accidents")
plt.title("Number of Accidents during the hours of the day",fontdict = {'fontsize':16 ,
'color':'MidnightBlue'})
```

Out[ ]:

```
Text(0.5, 1.0, 'Number of Accidents during the hours of the day')
```



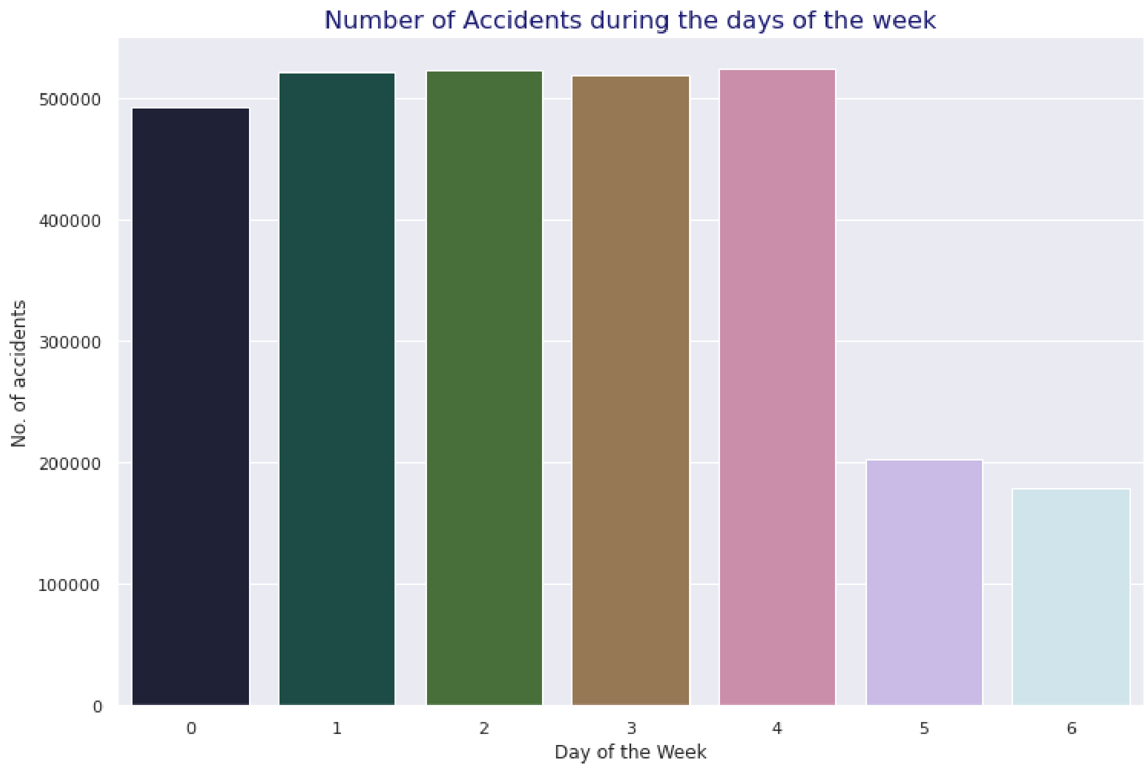
Here, we see the peaks in the morning (around 8am) and another peak around 5pm. This is close to the rush hours, especially during the weekdays.

In [ ]:

```
#days of the week
week_day = acc_train['day_of_Week'].value_counts().to_frame()
day_labels = np.arange(7)


plt.figure(figsize=(12,8))
sns.barplot(
    data=week_day,
    x=week_day.index, y="day_of_Week", palette='cubehelix')

plt.xlabel("Day of the Week")
plt.xticks(day_labels)
plt.ylabel("No. of accidents")
plt.title("Number of Accidents during the days of the week",fontdict = {'fontsize':16 ,
'color':'MidnightBlue'})
```

Out[ ]:

```
Text(0.5, 1.0, 'Number of Accidents during the days of the week')
```



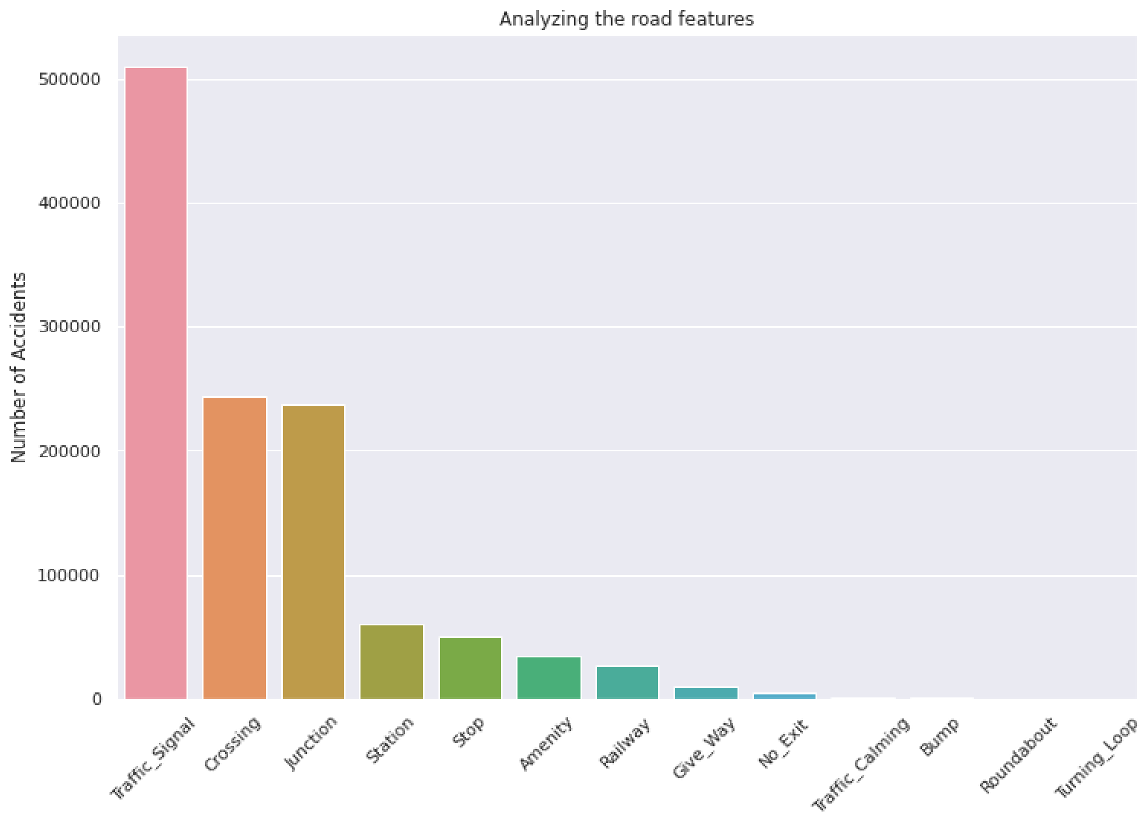
In [ ]:

In [19]:

```
#since there are so many road features, but not too many to perform PCA on, I'm going t
o try to analyze which ones are most important
road_features = acc_train[["Amenity", "Bump", "Crossing", "Give_Way", "Junction", "No_E
xit", "Railway", "Roundabout", "Station", "Stop", "Traffic_Calming", "Traffic_Signal",
"Turning_Loop"]]
road_df = road_features.sum().sort_values(ascending=False).to_frame().rename(columns={0
:'count'})
road_df

#plotting it on barplot
plt.figure(figsize=(12,8))
sns.barplot(data=road_df, x=road_df.index, y='count')
plt.xticks(rotation=45)
plt.ylabel("Number of Accidents")
plt.title("Analyzing the road features")
```

Out[19]:

```
Text(0.5, 1.0, 'Analyzing the road features')
```



So this shows that accidents usually happened in places where there was a traffic signal, with a crossing and a junction.

In [20]:

```
acc_train.nunique()
```

Out[20]:

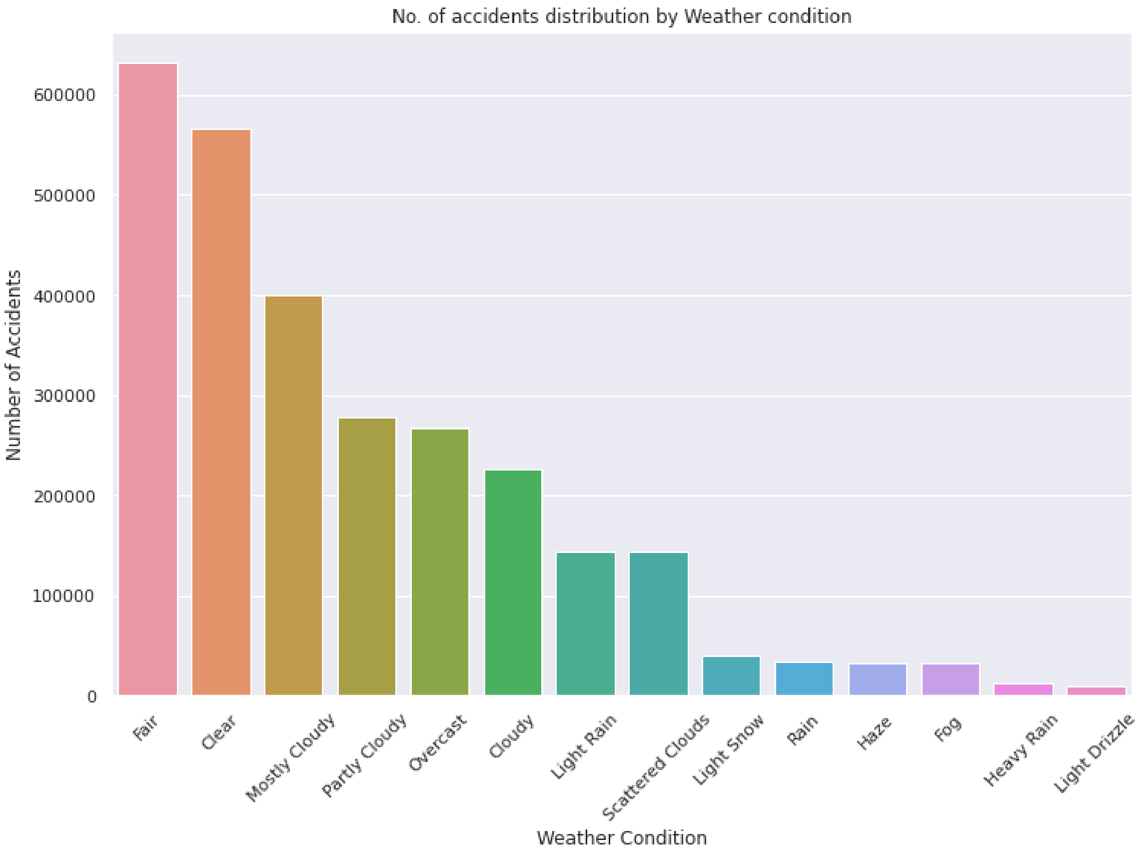
```
Severity                    2
Start_Time            2628784
End_Time              2707188
Start_Lat              798384
Start_Lng              749983
Distance.mi.            13061
Description           1655596
Street                 181216
Side                        2
City                    11766
County                   1730
State                      49
Zipcode                398476
Temperature.F.            826
Wind_Chill.F.             967
Humidity...               100
Pressure.in.             1024
Visibility.mi.             83
Wind_Direction             24
Wind_Speed.mph.           147
Precipitation.in.         250
Weather_Condition         127
Amenity                     2
Bump                        2
Crossing                    2
Give_Way                    2
Junction                    2
No_Exit                     2
Railway                     2
Roundabout                  2
Station                     2
Stop                        2
Traffic_Calming             2
Traffic_Signal              2
Turning_Loop                1
Sunrise_Sunset              2
day_of_Week                 7
month                      12
year                        5
Hour                       24
dtype: int64
```

In [21]:

```
weather_cond_df = acc_train['Weather_Condition'].value_counts(dropna=False)[0:15].to_fr
ame()

plt.figure(figsize=(12,8))
sns.barplot(data=weather_cond_df, x=weather_cond_df.index, y='Weather_Condition')
plt.xlabel("Weather Condition")
plt.ylabel("Number of Accidents")
plt.title("No. of accidents distribution by Weather condition")
plt.xticks(rotation=45)
```

Out[21]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13]),
 <a list of 14 Text major ticklabel objects>)
```



In [22]:

```
#number of null values per column
acc_train.isna().sum()
```

Out[22]:

```
Severity                  0
Start_Time                0
End_Time                  0
Start_Lat                 0
Start_Lng                 0
Distance.mi.              0
Description               0
Street                    0
Side                      0
City                      0
County                    0
State                     0
Zipcode                 933
Temperature.F.        63185
Wind_Chill.F.       1327951
Humidity...           67144
Pressure.in.          53742
Visibility.mi.        69279
Wind_Direction        58875
Wind_Speed.mph.      336173
Precipitation.in.   1446591
Weather_Condition     69140
Amenity                   0
Bump                      0
Crossing                  0
Give_Way                  0
Junction                  0
No_Exit                   0
Railway                   0
Roundabout                0
Station                   0
Stop                      0
Traffic_Calming           0
Traffic_Signal            0
Turning_Loop              0
Sunrise_Sunset            0
day_of_Week               0
month                     0
year                      0
Hour                      0
dtype: int64
```

In [23]:

```
#further dropping Precipitation and Wind_Chill with >1.3M missing values
acc_train.drop(columns=["Wind_Chill.F.","Precipitation.in."], inplace=True)
```

In [24]:

```
#taking a look at pressure and visibility
acc_train[["Pressure.in.", "Visibility.mi."]].describe().round(2)
```

Out[24]:

	Pressure.in.	Visibility.mi.
count	2908932.00	2893395.00
mean	29.69	9.11
std	0.86	2.82
min	0.00	0.00
25%	29.64	10.00
50%	29.93	10.00
75%	30.08	10.00
max	58.04	140.00

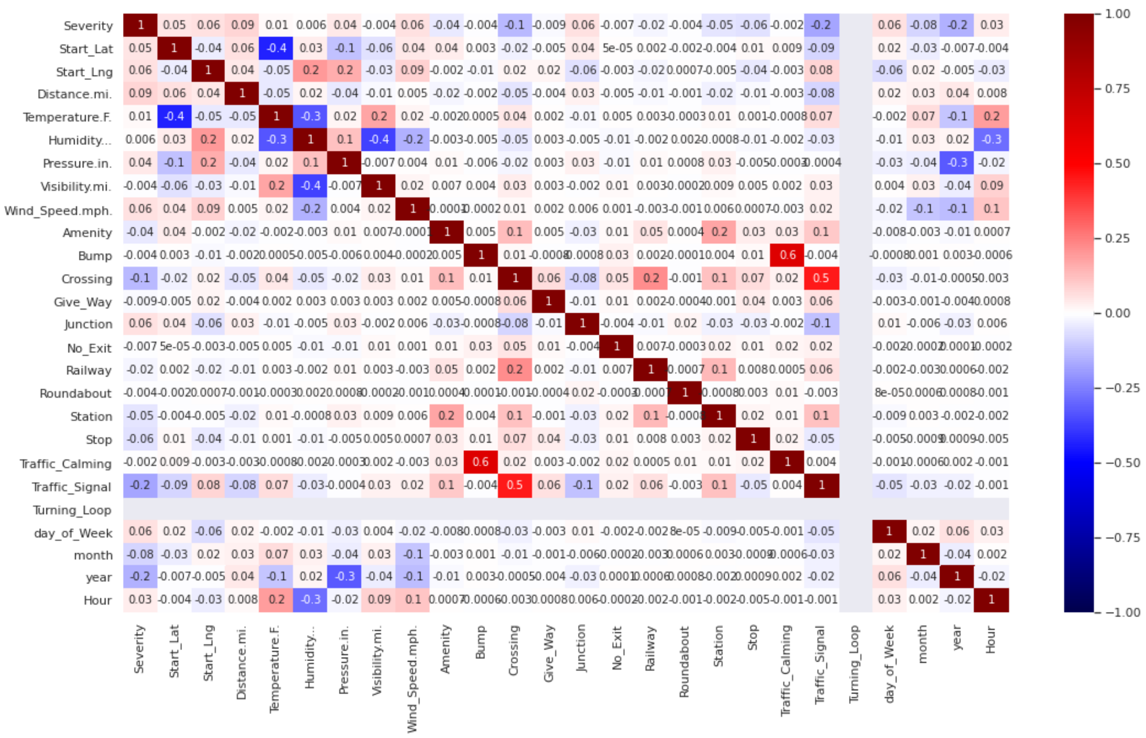
Data Preprocessing

In [27]:

```
#correlation matrix to determine if there are any highly related columns
correlation_mtx = acc_train.corr()

plt.figure(figsize=(18, 10))
sns.heatmap(correlation_mtx, vmin=-1, vmax=1, cmap="seismic", annot = True, fmt='.1g')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f97fefb11d0>
```



Windchill and Humidity seem to be highly correlation(negative correlation).\ Turning Loop seems to have a missing correlation which turns out are all False values.\

In [ ]:

```
acc_train["Description"].value_counts()
```

Out[ ]:

```
A crash has occurred causing no to minimum delays. Use caution.
1862
At I-15 - Accident.
1507
At I-5 - Accident.
1344
At I-405/San Diego Fwy - Accident.
1242
At I-605 - Accident.
1047

...
Lane blocked due to accident on PA-130 Sandy Creek Rd Northbound from PA-3
80 Frankstown Rd to Verona Rd.        1
Accident on I-5 Northbound before Exit 45 Poinsettia Ln. On the median.
1
Accident on US-29 Tryon St Westbound at Us-29 Access Rd.
1
Accident on 88th Ave Westbound at I-76.
1
Accident on CA-193 both ways at Kelsey Rd.
1
Name: Description, Length: 1655596, dtype: int64
```

In [ ]:

```
acc_train.columns
```

Out[ ]:

```
Index(['Severity', 'Start_Time', 'End_Time', 'Start_Lat', 'Start_Lng',
       'Distance.mi.', 'Description', 'Street', 'Side', 'City', 'County',
       'State', 'Zipcode', 'Temperature.F.', 'Humidity...', 'Pressure.i
n.',
       'Visibility.mi.', 'Wind_Direction', 'Wind_Speed.mph.',
       'Weather_Condition', 'Amenity', 'Bump', 'Crossing', 'Give_Way',
       'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
       'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunse
t',
       'day_of_Week', 'month', 'year', 'Hour'],
      dtype='object')
```

In [ ]:

```
drop_again = ["End_Time", "Turning_Loop",'Street', 'County', 'Zipcode','Start_Time','De
scription','Wind_Direction']
X = acc_train.drop(columns=drop_again, inplace=False)
```

In [ ]:

```
print("rows with duplicates:", len(X.index))
X.drop_duplicates(inplace=True)
print("rows after duplicates are dropped:", len(X.index))
```

```
rows with duplicates: 2962674
rows after duplicates are dropped: 2861522
```

In [ ]:

```
for i in X.columns:
    print(i)
    display(X[i].value_counts())
    display(len(X[i].value_counts().index))
```

```
Severity

0    2034106
1     827416
Name: Severity, dtype: int64

2

Start_Lat

37.808500    452
33.876290    410
33.941360    387
33.744980    375
33.781530    367
            ...
39.872078      1
29.711370      1
35.468370      1
45.779950      1
41.520110      1
Name: Start_Lat, Length: 798384, dtype: int64

798384

Start_Lng

-118.36830    626
-118.28060    602
-111.89100    578
-111.89130    555
-121.38290    548
             ...
-95.16788       1
-95.33212       1
-82.37925       1
-76.71649       1
-83.91406       1
Name: Start_Lng, Length: 749983, dtype: int64

749983

Distance.mi.

0.000     1849222
0.010      185463
0.020        5883
0.001        3841
0.008        2721
           ...
6.760           1
8.498           1
9.533           1
13.388          1
12.610          1
Name: Distance.mi., Length: 13061, dtype: int64

13061

Side
```

```
R    2343334
L     518188
Name: Side, dtype: int64
```

```
2
```

```
City
```

```
Houston           79236
Los Angeles       62533
Charlotte         60699
Dallas            52992
Austin            48369
                  ...
Counselor             1
Charleston AFB        1
Ventnor City          1
Tiro                  1
Wallington            1
Name: City, Length: 11766, dtype: int64
```

```
11766
```

```
State
```

```
CA    655363
TX    259189
FL    243043
SC    145433
NC    131731
NY    126628
PA     90435
VA     84366
IL     77119
GA     73426
MI     73003
OR     72330
MN     70913
AZ     63339
TN     60449
LA     54250
WA     51463
OH     50487
NJ     46282
MD     45376
OK     45259
UT     39478
AL     39082
CO     36779
MA     30298
MO     27274
IN     26317
CT     21626
NE     17542
KY     17059
WI     14953
IA      9377
RI      8865
NV      8316
KS      6620
NH      5963
MS      5526
DE      4613
NM      4422
DC      4181
AR      3210
ID      2622
WV      2405
MT      2149
ME      1660
VT       502
WY       370
ND       288
SD       141
Name: State, dtype: int64
```

```
49
```

```
Temperature.F.
```

```
 68.0     62468
 77.0     61222
 59.0     58493
 73.0     57054
 63.0     53913
          ...
 113.4        1
 111.7        1
 143.6        1
 167.0        1
-21.1         1
Name: Temperature.F., Length: 826, dtype: int64
```

```
826
```

```
Humidity...
```

```
100.0    115584
93.0     111138
90.0      65492
87.0      65062
89.0      54435
          ...
5.0        1581
4.0         868
3.0         217
2.0          67
1.0           9
Name: Humidity..., Length: 100, dtype: int64
```

```
100
```

```
Pressure.in.
```

```
29.96    54143
30.01    54001
29.99    53414
29.94    51922
30.04    51432
         ...
20.30        1
20.05        1
19.89        1
20.07        1
20.10        1
Name: Pressure.in., Length: 1024, dtype: int64
```

```
1024
```

```
Visibility.mi.
```

```
10.0    2225748
7.0       87391
9.0       75023
8.0       59906
5.0       56463
         ...
58.0          1
6.2           1
63.0          1
67.0          1
43.0          1
Name: Visibility.mi., Length: 83, dtype: int64
```

```
83
```

```
Wind_Speed.mph.
```

```
0.0      245610
4.6      151218
5.8      150100
3.5      141735
6.9      140161
          ...
211.0         1
214.0         1
105.0         1
232.0         1
77.1          1
Name: Wind_Speed.mph., Length: 147, dtype: int64
```

```
147
```

```
Weather_Condition
```

```
Fair                    587613
Clear                   562916
Mostly Cloudy           387759
Partly Cloudy           270262
Overcast                265814
                         ...
Blowing Sand                 1
Hail                         1
Light Fog                    1
Partial Fog / Windy          1
Light Sleet / Windy          1
Name: Weather_Condition, Length: 127, dtype: int64
```

```
127
```

```
Amenity
```

```
False    2827989
True       33533
Name: Amenity, dtype: int64
```

```
2
```

```
Bump
```

```
False    2860952
True         570
Name: Bump, dtype: int64
```

```
2
```

```
Crossing
```

```
False    2625408
True      236114
Name: Crossing, dtype: int64
```

```
2
```

```
Give_Way
```

```
False    2852483
True        9039
Name: Give_Way, dtype: int64
```

```
2
```

```
Junction
```

```
False    2633717
True      227805
Name: Junction, dtype: int64
```

```
2
```

```
No_Exit
```

```
False    2857543
True        3979
Name: No_Exit, dtype: int64
```

```
2
```

```
Railway
```

```
False    2835415
True       26107
Name: Railway, dtype: int64
```

```
2
```

```
Roundabout
```

```
False    2861350
True         172
Name: Roundabout, dtype: int64
```

```
2
```

```
Station
```

```
False    2804030
True       57492
Name: Station, dtype: int64
```

```
2
```

```
Stop
```

```
False    2812221
True       49301
Name: Stop, dtype: int64
```

```
2
```

```
Traffic_Calming
```

```
False    2860058
True        1464
Name: Traffic_Calming, dtype: int64
```

```
2
```

```
Traffic_Signal
```

```
False    2365989
True      495533
Name: Traffic_Signal, dtype: int64
```

```
2
```

```
Sunrise_Sunset
```

```
Day      1998429
Night     863093
Name: Sunrise_Sunset, dtype: int64
```

```
2
```

```
day_of_Week
```

```
4    508242
2    506441
1    505395
3    502531
0    476990
5    192008
6    169915
Name: day_of_Week, dtype: int64
```

```
7
```

```
month
```

```
12    333503
11    323433
10    310946
9     258921
8     227136
6     213374
1     209472
4     203471
5     202654
3     201599
2     197375
7     179638
Name: month, dtype: int64
```

```
12
```

```
year
```

```
2020    792602
2019    662625
2018    621325
2017    499246
2016    285724
Name: year, dtype: int64
```

```
5
```

```
Hour
```

```
8     247876
7     241209
17    205814
16    200750
15    169639
18    161249
6     157296
9     152426
14    145228
10    140042
11    139058
13    137093
12    127256
19    112616
20     84175
5      82998
21     66185
4      60068
22     58642
23     41257
0      37168
1      34022
2      31633
3      27822
Name: Hour, dtype: int64
```

```
24
```

Don't run anything under this until the modelling section

In [ ]:

```

```

In [ ]:

```
list_values = ['City','State','Weather_Condition','day_of_Week','month','year','Hour']
num_vals = [25, 25,15,6,12,5,24]
```

In [ ]:

```
for i in range(len(list_values)):
    col_name = list_values[i]
    num_cols = num_vals[i]
    col_df = pd.get_dummies(X[col_name],prefix=col_name, prefix_sep='_')
    col_df = col_df.iloc[:,0:num_cols]
    X = X.join(col_df)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-061c47e0a03f> in <module>()
      2     col_name = list_values[i]
      3     num_cols = num_vals[i]
----> 4     col_df = pd.get_dummies(X[col_name],prefix=col_name, prefix_sep='_')
      5     col_df = col_df.iloc[:,0:num_cols]
      6     X = X.join(col_df)

NameError: name 'pd' is not defined
```

In [ ]:

```
bool_vals = ['Amenity','Bump','Crossing','Give_Way',
               'Junction','No_Exit','Railway','Roundabout','Station','Stop','Traffic_Calming','Traffic_Signal',
             'Sunrise_Sunset',]
```

In [ ]:

```
X['Weather_Condition'].value_counts().to_frame().head(30)
```

Out[ ]:

	Weather_Condition
Fair	587613
Clear	562916
Mostly Cloudy	387759
Partly Cloudy	270262
Overcast	265814
Cloudy	211467
Scattered Clouds	142512
Light Rain	139644
Light Snow	39037
Rain	32918
Haze	31680
Fog	30231
Heavy Rain	12387
Light Drizzle	9835
Fair / Windy	7410
Smoke	5525
Snow	4399
Mostly Cloudy / Windy	4122
Cloudy / Windy	4067
Light Thunderstorms and Rain	3393
T-Storm	3362
Thunderstorm	3086
Thunder in the Vicinity	2844
Light Rain with Thunder	2722
Partly Cloudy / Windy	2493
Thunder	2323
Light Rain / Windy	2246
Patches of Fog	2073
Drizzle	2012
Mist	1894

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Stat154Project

May 12, 2021

[11]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

[12]:
```
#Load in Training Dataset
df = pd.read_csv('train.csv')
```

[13]:
```
#Changing Severity to Binary Labels
df['Severity'][df['Severity'] < 3] = 0
df['Severity'][df['Severity'] >= 3] = 1
```

```
<ipython-input-13-879673542eb3>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Severity'][df['Severity'] < 3] = 0
<ipython-input-13-879673542eb3>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Severity'][df['Severity'] >= 3] = 1
```

[14]:
```
#Filtering out non-useful columns
cols =␣
 ↪['ID','Source','TMC','End_Lat','End_Lng','Description','Number','Country','Airport_Code',
     ␣
 ↪'Weather_Timestamp','Timezone','Civil_Twilight','Nautical_Twilight','Astronomical_Twilight'
df = df.drop(cols, axis = 1)
```

[15]:
```
#Filtering out Subsets of NA Values per column
df = df.dropna(subset=['City'])
df = df.dropna(subset=['Temperature.F.'])
df = df.dropna(subset=['Humidity...'])
df = df.dropna(subset=['Pressure.in.'])
df = df.dropna(subset=['Visibility.mi.'])
```

```
df = df.dropna(subset=['Wind_Direction'])
df = df.dropna(subset=['Weather_Condition'])
```

[16]:
```
#getting some additional columns which may/may not be used
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['End_Time'] = pd.to_datetime(df['End_Time'])
df['day_of_Week'] = df['Start_Time'].dt.dayofweek
df['month'] = pd.DatetimeIndex(df['Start_Time']).month
df['year'] = pd.DatetimeIndex(df['Start_Time']).year
df['Hour'] = df['Start_Time'].dt.hour
```

[17]:
```
#further dropping Precipitation and Wind_Chill with >1.3M missing values
df.drop(columns=["Wind_Chill.F.","Precipitation.in."], inplace=True)
```

[18]:
```
#Dropping useless columns and duplicates
drop_again = ["End_Time", "Turning_Loop",'Street', 'County', 'Zipcode']
X = df.drop(columns=drop_again, inplace=False)
X.drop_duplicates(inplace=True)
```

[29]:
```
#Fill NA Value for Wind_Speed
X['Wind_Speed.mph.'] = X['Wind_Speed.mph.'].fillna((X['Wind_Speed.mph.']).
 ↪median())
```

[30]:
```
features = ['month','Severity','Start_Lat','Start_Lng','Distance.mi.
 ↪','Wind_Speed.mph.','Hour']
X = X[features]
y_train_full = X['Severity']
X_train_full = X.drop(['Severity'], axis=1)
```

[70]:
```
## Getting Test Set Ready for Predictions
x_test = pd.read_csv('test.csv')
ids = x_test['ID']
```

[71]:
```
#Filtering out non-useful columns
cols =␣
 ↪['ID','Source','TMC','End_Lat','End_Lng','Description','Number','Country','Airport_Code',
     ␣
 ↪'Weather_Timestamp','Timezone','Civil_Twilight','Nautical_Twilight','Astronomical_Twilight'
x_test = x_test.drop(cols, axis = 1)
x_test = x_test.drop(['City'],axis=1)
```

[72]:
```
#getting some additional columns which may/may not be used
x_test['Start_Time'] = pd.to_datetime(x_test['Start_Time'])
x_test['End_Time'] = pd.to_datetime(x_test['End_Time'])
x_test['day_of_Week'] = x_test['Start_Time'].dt.dayofweek
x_test['month'] = pd.DatetimeIndex(x_test['Start_Time']).month
x_test['year'] = pd.DatetimeIndex(x_test['Start_Time']).year
```

```
x_test['Hour'] = x_test['Start_Time'].dt.hour
```

[73]:
```
#further dropping Precipitation and Wind_Chill with >1.3M missing values
x_test.drop(columns=["Wind_Chill.F.","Precipitation.in."], inplace=True)
drop_again = ["End_Time", "Turning_Loop",'Street', 'County', 'Zipcode']
x_test = x_test.drop(columns=drop_again, inplace=False)
```

[74]:
```
features = ['month','Start_Lat','Start_Lng','Distance.mi.','Wind_Speed.mph.
↪','Hour']
x_test = x_test[features]
x_test['Wind_Speed.mph.'] = x_test['Wind_Speed.mph.'].
↪fillna((x_test['Wind_Speed.mph.']).median())
```

0.1 Model 1: Decision Tree (0.829 Accuracy on Kaggle) - 6 Features

[59]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score,auc, confusion_matrix
from matplotlib import pyplot
import matplotlib.pyplot as plt
```

[32]:
```
x_train, x_val, y_train, y_val = train_test_split(X_train_full, y_train_full,␣
↪test_size=0.30, random_state=42)
```

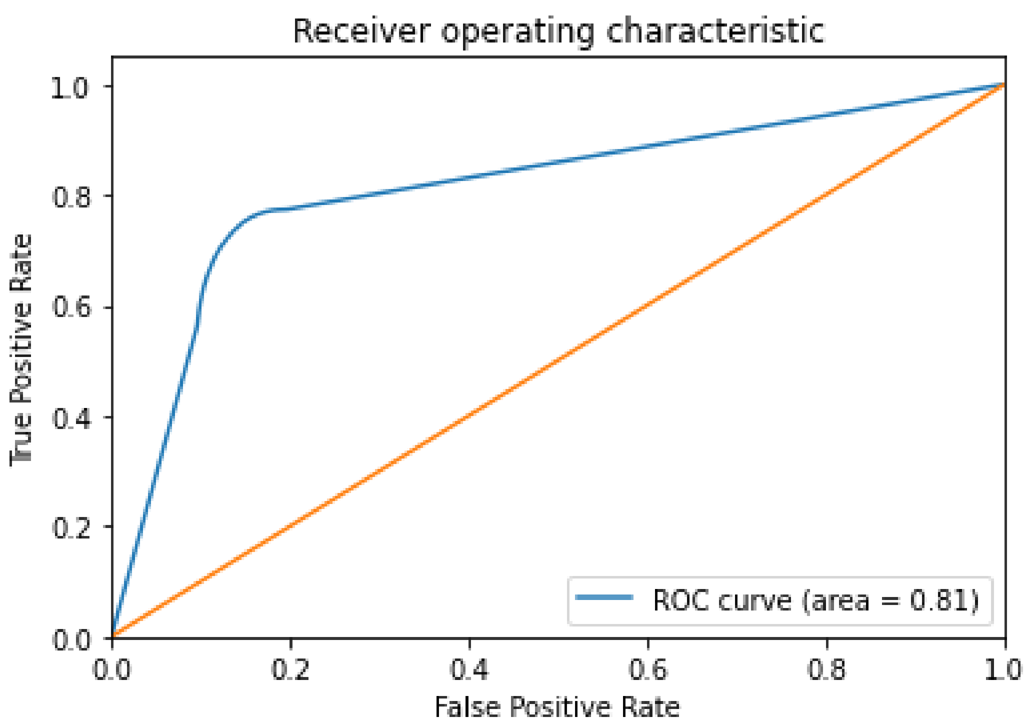
[33]:
```
dec_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 30,␣
↪random_state=40)

dec_tree.fit(x_train, y_train)

print("Train score:", dec_tree.score(x_train, y_train))
print("Validation score:", dec_tree.score(x_val, y_val))
```

```
Train score: 0.965190180838974
Validation score: 0.8278866161752448
```

[62]:
```
def get_auc_scores(clf, X_train, X_test, y_train, y_test):
    y_train_score = clf.predict_proba(X_train)[:, 1]
    y_test_score = clf.predict_proba(X_test)[:, 1]
    auc_train = roc_auc_score(y_train, y_train_score)
    auc_test = roc_auc_score(y_test, y_test_score)
    return y_test_score
def plot_roc_curve(y_test, y_test_score):
```

```
    fpr, tpr, _ = roc_curve(y_test, y_test_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, label= "ROC curve (area = %0.2f)" % roc_auc)
    plt.plot([0, 1], [0, 1])
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver operating characteristic")
    plt.legend(loc= "lower right")
    plt.show()
```

[63]:
```
scores = get_auc_scores(dec_tree, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



[40]:
```
#Predictions on test set
y_pred = dec_tree.predict(x_test)
```

[43]:
```
data = {'ID':ids,
       'Severity':y_pred}
y_predictions = pd.DataFrame(data)
```

[44]:
```
y_predictions.set_index('ID')
```

[44]:
```
          Severity
ID
A-1              0
A-5              0
A-7              0
A-14             0
A-22             0
…              …
A-4239383        0
A-4239389        0
A-4239400        0
A-4239402        0
A-4239404        0

[1269762 rows x 1 columns]
```

[455]:
```
y_predictions.to_csv('y_predictions.csv',index=False)
```

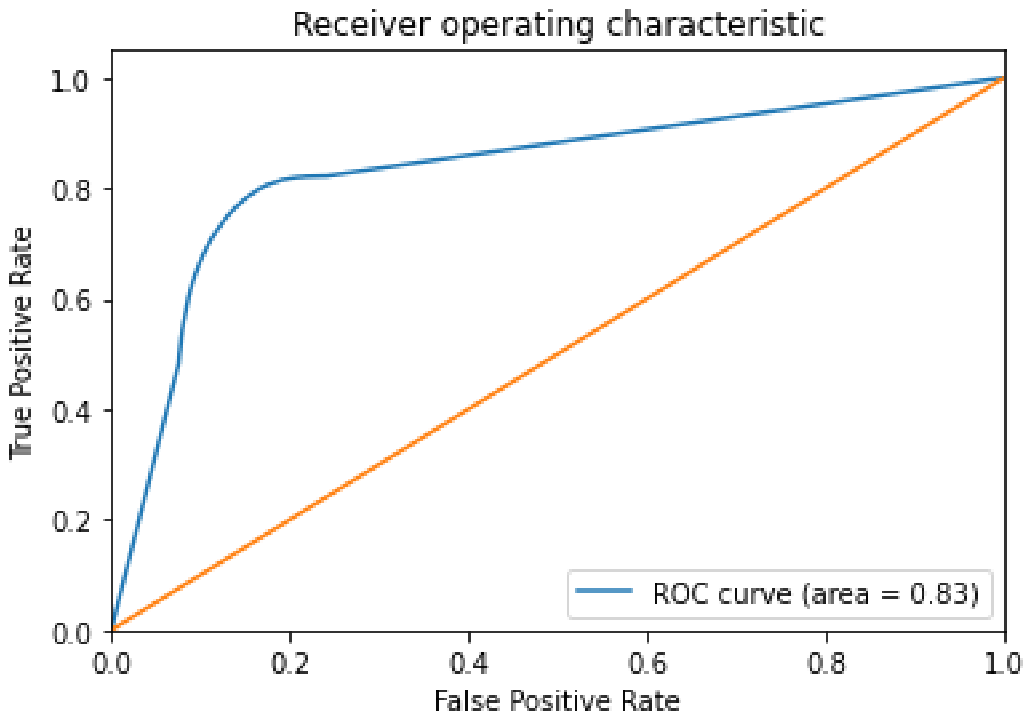
0.2 Model 2: Another Decision Tree (0.83393 Accuracy on Kaggle) - 4 Features

[65]:
```
X_m2 = X.drop(['month','Hour'], axis=1)
y_train_full2 = X_m2['Severity']
X_train_full2 = X_m2.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full2, y_train_full2,␣
↪test_size=0.30, random_state=42)
```

[67]:
```
dec_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 30,␣
↪random_state=40)
dec_tree.fit(x_train, y_train)

print("Train score:", dec_tree.score(x_train, y_train))
print("Validation score:", dec_tree.score(x_val, y_val))
```

```
Train score: 0.9444540675790769
Validation score: 0.8360228110889236
```

[68]:
```
scores = get_auc_scores(dec_tree, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



[79]:
```
#Getting predictions on test set
xtest2 = x_test.drop(['month','Hour'], axis=1)
y_pred2 = dec_tree.predict(xtest2)
```

[80]:
```
data = {'ID':ids,
       'Severity':y_pred2}
y_predictions = pd.DataFrame(data)
y_predictions.set_index('ID')
```

[80]:

	Severity
ID	
A-1	0
A-5	0
A-7	0
A-14	0
A-22	0
…	…
A-4239383	0
A-4239389	0
A-4239400	0
A-4239402	0
A-4239404	0

```
[1269762 rows x 1 columns]
```

[81]:
```
y_predictions.to_csv('y_predictions2.csv',index=False)
```

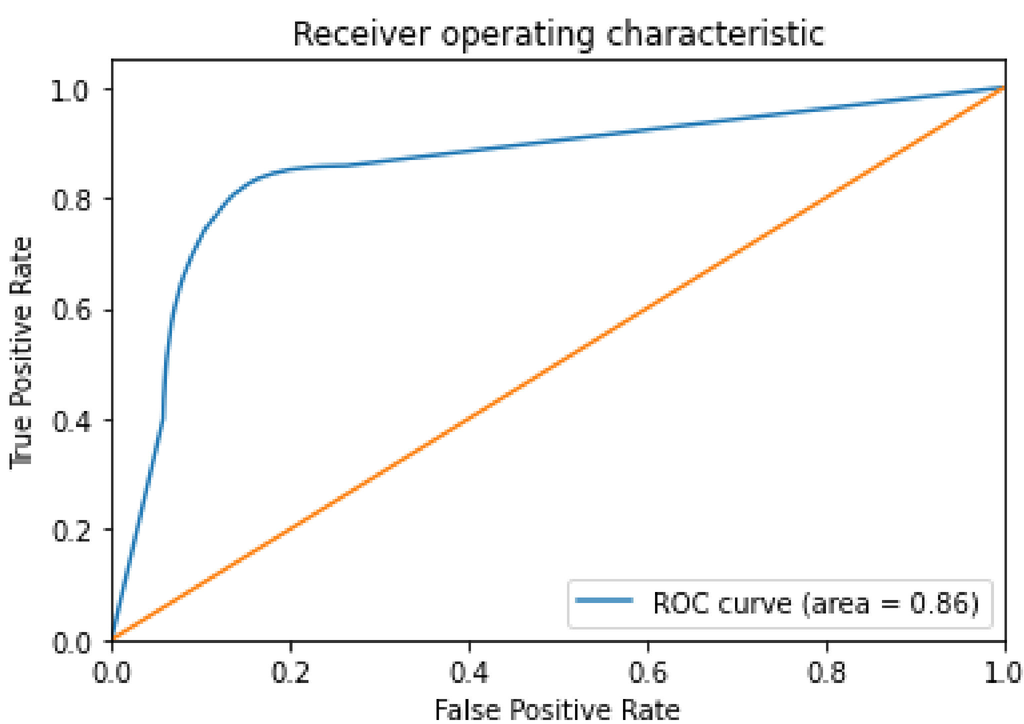
0.3 Model 3: Another Decision Tree (0.85271 Accuracy on Kaggle) - 3 Features

[82]:
```
X_m3 = X.drop(['month','Hour','Wind_Speed.mph.'], axis=1)
y_train_full3 = X_m3['Severity']
X_train_full3 = X_m3.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full3, y_train_full3,␣
↪test_size=0.30, random_state=42)
```

[83]:
```
dec_tree = DecisionTreeClassifier(criterion = 'gini', max_depth =␣
↪30,random_state=40)
dec_tree.fit(x_train, y_train)

print("Train score:", dec_tree.score(x_train, y_train))
print("Validation score:", dec_tree.score(x_val, y_val))
```

```
Train score: 0.9371928556706277
Validation score: 0.8511402546158874
```

[84]:
```
scores = get_auc_scores(dec_tree, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



[86]:
```
#Getting Predictions on test set
xtest3 = x_test.drop(['month','Wind_Speed.mph.','Hour'],axis=1)
y_pred3 = dec_tree.predict(xtest3)
```

[87]:
```
data = {'ID':ids,
        'Severity':y_pred3}
y_predictions = pd.DataFrame(data)
y_predictions.set_index('ID')
```

[87]:
```
           Severity
ID
A-1               0
A-5               0
A-7               0
A-14              0
A-22              0
…               …
A-4239383         1
A-4239389         0
A-4239400         0
A-4239402         0
A-4239404         0

[1269762 rows x 1 columns]
```

[88]:
```
y_predictions.to_csv('y_predictions3.csv',index=False)
```

0.4 Model 4: Random Forest (0.83864 Accuracy on Kaggle) - 5 Features

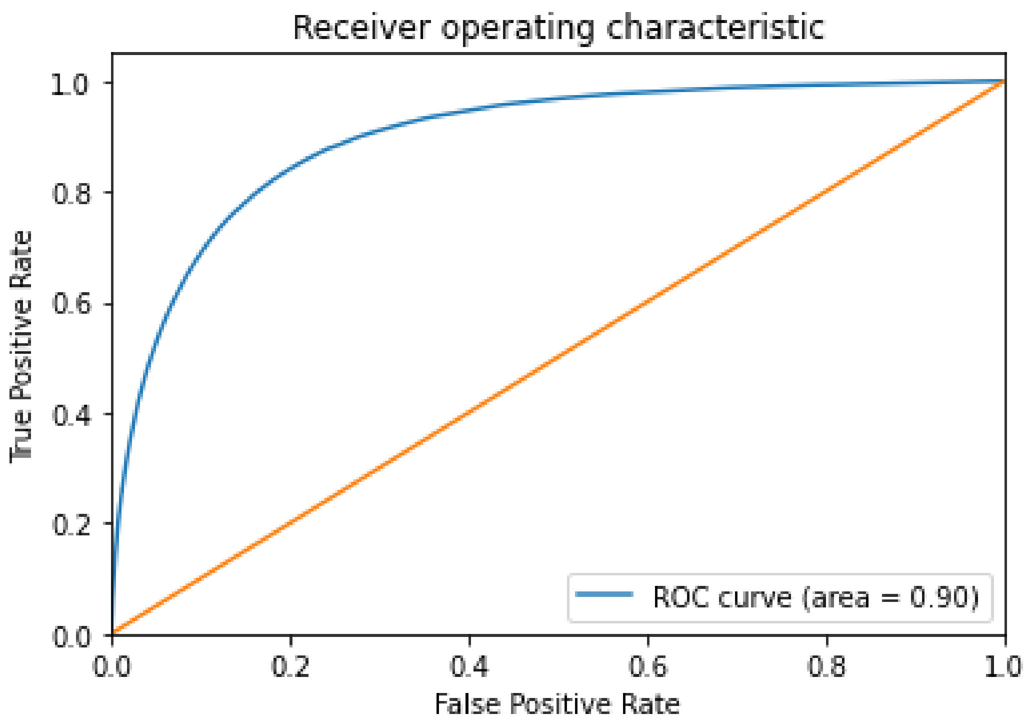
[90]:
```
X_m4 = X.drop(['Wind_Speed.mph.'],axis=1)
y_train_full4 = X_m4['Severity']
X_train_full4 = X_m4.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full4, y_train_full4,␣
↪test_size=0.20, random_state=42)
```

[91]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

dec_tree = Pipeline([('scaler',StandardScaler()),('rf',␣
↪RandomForestClassifier(criterion='entropy',n_estimators = 20, max_depth=30,
                                                        ␣
↪random_state=40))])

dec_tree.fit(x_train, y_train)
```

```
print("Train score:", dec_tree.score(x_train, y_train))
print("Validation score:", dec_tree.score(x_val, y_val))
```

```
Train score: 0.9791290941488965
Validation score: 0.8386529224458956
```

[92]:
```
scores = get_auc_scores(dec_tree, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



0.5 Model 5: Another Random Forest (0.87390 Accuracy on Kaggle) - 3 Features

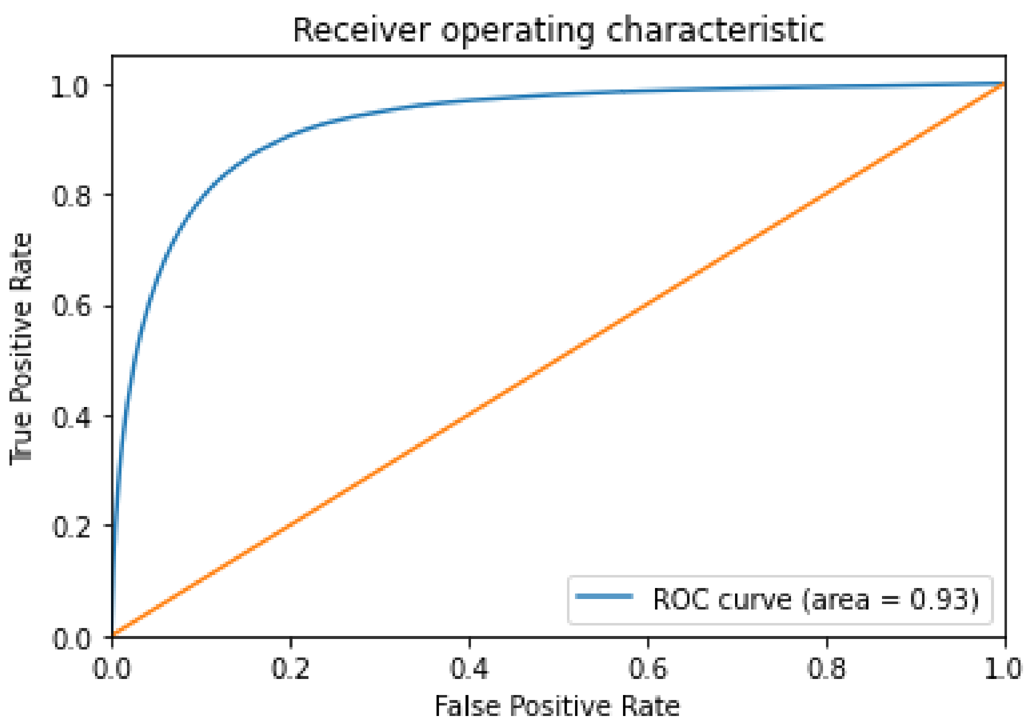
[93]:
```
X_m5 = X.drop(['month','Hour','Wind_Speed.mph.'],axis=1)
y_train_full5 = X_m5['Severity']
X_train_full5 = X_m5.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full5, y_train_full5,␣
↪test_size=0.2, random_state=42)
```

[94]:
```
rf = Pipeline([('scaler',StandardScaler()),('rf',␣
↪RandomForestClassifier(criterion='entropy',n_estimators = 35,max_depth=30,                                                                     ␣
↪random_state=40))])
```

```
rf.fit(x_train, y_train)

print("Train score:", rf.score(x_train, y_train))
print("Validation score:", rf.score(x_val, y_val))
```

```
Train score: 0.9397142674674541
Validation score: 0.869104286973573
```

[95]:
```
scores = get_auc_scores(rf, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



[104]:
```
xtest5 = x_test.drop(['month','Hour','Wind_Speed.mph.'],axis=1)
y_pred5 = dec_tree.predict(xtest5)
```

[106]:
```
data = {'ID':ids,
       'Severity':y_pred5}
y_predictions = pd.DataFrame(data)
y_predictions.set_index('ID')
y_predictions.to_csv('y_predictions5.csv',index=False)
```

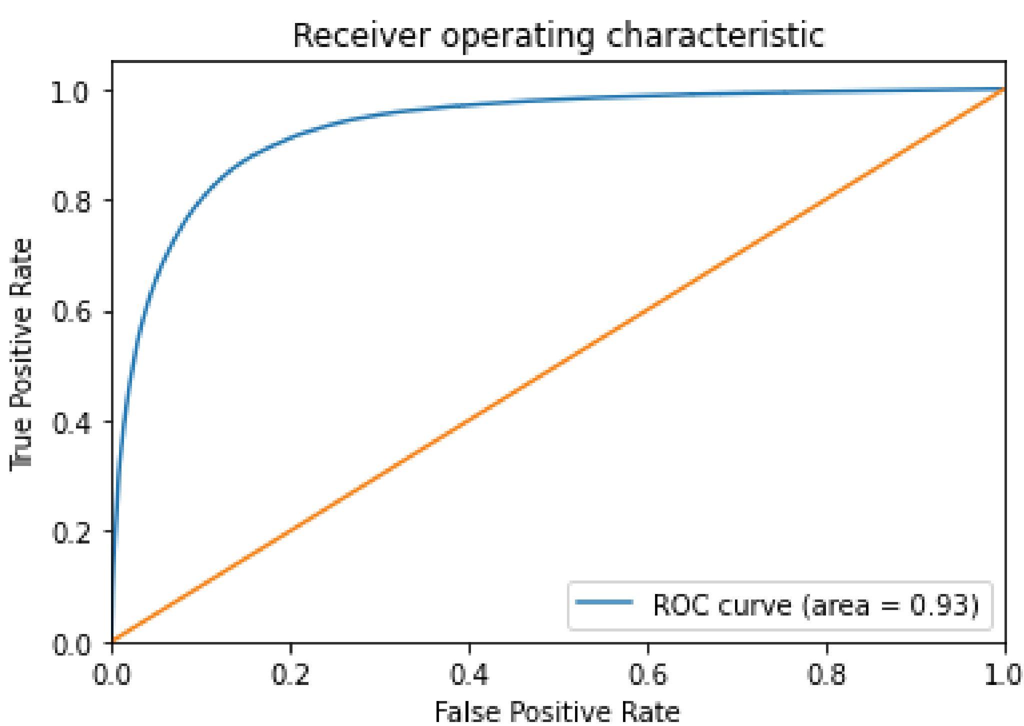
0.6 Model 7: Another Random Forest (0.87738 Accuracy on Kaggle)

[96]:
```
X_m7 = X.drop(['month','Hour','Wind_Speed.mph.'],axis=1)
y_train_full7 = X_m7['Severity']
X_train_full7 = X_m7.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full7, y_train_full7,␣
↪test_size=0.1, random_state=42)
```

[97]:
```
rf = Pipeline([('scaler',StandardScaler()),('rf',␣
↪RandomForestClassifier(bootstrap = True, criterion='gini',max_features =␣
↪'sqrt',
                                                                            ␣
↪min_samples_split = 3,n_estimators = 210,max_depth=35,
                                                                              ␣
↪random_state=40))])
rf.fit(x_train, y_train)

print("Train score:", rf.score(x_train, y_train))
print("Validation score:", rf.score(x_val, y_val))
```

```
Train score: 0.9478166940045486
Validation score: 0.8722471081367946
```

[98]:
```
scores = get_auc_scores(rf, x_train, x_val, y_train, y_val)
plot_roc_curve(y_val, scores)
```



[101]:
```
#xtest7 = x_test.drop(['month','Hour','Wind_Speed.mph.'],axis=1)
#y_pred7 = rf.predict(xtest7)
```

[51]:
```
data = {'ID':ids,
       'Severity':y_pred7}
y_predictions = pd.DataFrame(data)
y_predictions.set_index('ID')
y_predictions.to_csv('y_predictions7.csv',index=False)
```

0.6.1 Random Bagging Classifier - Not Good Results

[73]:
```
from sklearn.ensemble import BaggingClassifier
```

[125]:
```
X_m8 = X.drop(['month','Hour','Wind_Speed.mph.','Junction'],axis=1)
y_train_full8 = X_m8['Severity']
X_train_full8 = X_m8.drop(['Severity'], axis=1)
x_train, x_val, y_train, y_val = train_test_split(X_train_full8, y_train_full8,␣
 ↪test_size=0.1, random_state=42)
```

[126]:
```
be = DecisionTreeClassifier(max_depth=50, random_state=40)
dec_tree = Pipeline([('scaler',StandardScaler()),('rf',␣
 ↪BaggingClassifier(base_estimator = be,n_estimators=10, random_state=40))])
#parameters = [{"criterion": ["gini", "entropy"], "max_depth": [5, 10, 15, 30]}]
#grid = GridSearchCV(dec_tree, parameters, verbose=5, n_jobs=-1)
dec_tree.fit(x_train, y_train)


print("Train score:", dec_tree.score(x_train, y_train))
print("Validation score:", dec_tree.score(x_val, y_val))
```

```
Train score: 0.9602746770238481
Validation score: 0.8568716424138111
```

[ ]:
```
```