Data 602 Final Project: Fatal crashes in the District of Columbia

an explaination of the sttributes can be found here:
https://www.arcgis.com/sharing/rest/content/items/70392a096a8e431381f1f692aaa06afd/info/metadata/metadata.xml?format=default&output=html

```
import numpy as np
import pandas as pd
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
```

```
%matplotlib inline
```

```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

Introduction

The data that I am using for my final project was taken from the open data D.C web page. The data was taken from crashes in the district and recorded the time location, those involved, injuries and fatalities. The goal of my project is to create a classifier which can determine if a crash is fatal for anyone involved. This could be useful in finding locations that are particularly dangerous and re-working road signs or traffic patterns to make them safer.

Data exploration and cleaning

```
df = pd.read_csv('Crashes_in_DC.csv')
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:2718: DtypeWarning: Columns (4) have mixed
types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
           X          Y  ...  MINORINJURIESPASSENGER
UNKNOWNINJURIESPASSENGER
0 -77.007754  38.835717  ...                       0
0
1 -77.103886  38.939821  ...                       0
0
```

```
2 -77.031957  38.902531  ...                         0
0
3 -77.036076  38.933146  ...                         0
0
4 -77.006367  38.908292  ...                         0
0

[5 rows x 60 columns]
```

Many of the attributes indicate the location of the crash. To prevent multi-colinearity I will only use the X and Y coordinates provided. There are also many attributes such as ObjectID which provide no relevant information which I will remove.

```python
del df['X']
del df['Y']
del df['LATITUDE']
del df['LONGITUDE']
del df['REPORTDATE']
del df['OBJECTID']
del df['CRIMEID']
del df['CCN']
del df['ROUTEID']
del df['MEASURE']
del df['OFFSET']
del df['STREETSEGID']
del df['ROADWAYSEGID']
del df['TODATE']
del df['MARID']
del df['ADDRESS']
del df['EVENTID']
del df['MAR_SCORE']
del df['MAR_ADDRESS']
del df['NEARESTINTROUTEID']
del df['NEARESTINTSTREETNAME']
del df['LOCATIONERROR']
del df['LASTUPDATEDATE']
del df['MPDLATITUDE']
del df['MPDLONGITUDE']
del df['MPDGEOX']
del df['MPDGEOY']
del df['BLOCKKEY']
del df['SUBBLOCKKEY']
del df['WARD']
del df['FROMDATE']

df.head()
```

```
       XCOORD      YCOORD  ...  MINORINJURIESPASSENGER
UNKNOWNINJURIESPASSENGER
0  399314.444  129721.582  ...                       0
```

```
0
1  390980.291  141229.839  ...                    0
0
2  397228.190  137185.830  ...                    0
0
3  396884.752  140556.727  ...                    0
0
4  399489.348  137788.901  ...                    0
0

[5 rows x 29 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 259374 entries, 0 to 259373
Data columns (total 29 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   XCOORD                       259374 non-null  float64
 1   YCOORD                       259374 non-null  float64
 2   MAJORINJURIES_BICYCLIST      259374 non-null  int64
 3   MINORINJURIES_BICYCLIST      259374 non-null  int64
 4   UNKNOWNINJURIES_BICYCLIST    259374 non-null  int64
 5   FATAL_BICYCLIST              259374 non-null  int64
 6   MAJORINJURIES_DRIVER         259374 non-null  int64
 7   MINORINJURIES_DRIVER         259374 non-null  int64
 8   UNKNOWNINJURIES_DRIVER       259374 non-null  int64
 9   FATAL_DRIVER                 259374 non-null  int64
 10  MAJORINJURIES_PEDESTRIAN     259374 non-null  int64
 11  MINORINJURIES_PEDESTRIAN     259374 non-null  int64
 12  UNKNOWNINJURIES_PEDESTRIAN   259374 non-null  int64
 13  FATAL_PEDESTRIAN             259374 non-null  int64
 14  TOTAL_VEHICLES               259374 non-null  int64
 15  TOTAL_BICYCLES               259374 non-null  int64
 16  TOTAL_PEDESTRIANS            259374 non-null  int64
 17  PEDESTRIANSIMPAIRED          259374 non-null  int64
 18  BICYCLISTSIMPAIRED           259374 non-null  int64
 19  DRIVERSIMPAIRED              259374 non-null  int64
 20  TOTAL_TAXIS                  259374 non-null  int64
 21  TOTAL_GOVERNMENT             259374 non-null  int64
 22  SPEEDING_INVOLVED            259374 non-null  int64
 23  OFFINTERSECTION              259374 non-null  float64
 24  INTAPPROACHDIRECTION         259374 non-null  object
 25  FATALPASSENGER               259374 non-null  int64
 26  MAJORINJURIESPASSENGER       259374 non-null  int64
 27  MINORINJURIESPASSENGER       259374 non-null  int64
 28  UNKNOWNINJURIESPASSENGER     259374 non-null  int64
dtypes: float64(3), int64(25), object(1)
memory usage: 57.4+ MB
```

As we can see above there are several categorical features which will need to be dealt with. Finally, there is no need to distinguish between what kind of fatality/ injury so I will just sum those up and put them in their own respective features.

```
df['FATALITIES'] = df['FATAL_DRIVER'] + df['FATAL_BICYCLIST'] +
df['FATAL_PEDESTRIAN'] + df['FATALPASSENGER']
del df['FATAL_DRIVER']
del df['FATAL_BICYCLIST']
del df['FATAL_PEDESTRIAN']
del df['FATALPASSENGER']
df.head()
```

```
      XCOORD      YCOORD  ...  UNKNOWNINJURIESPASSENGER  FATALITIES
0  399314.444  129721.582  ...                         0           0
1  390980.291  141229.839  ...                         0           0
2  397228.190  137185.830  ...                         0           0
3  396884.752  140556.727  ...                         0           0
4  399489.348  137788.901  ...                         0           0

[5 rows x 26 columns]
```

Because we are only interested in wheather or not there was a fatality we want to convert the FATALITIES field into a 1 or 0.

```
df['FATALITIES'] = df['FATALITIES'].where(df['FATALITIES'] < 1, 1)
df['FATALITIES'].value_counts()
```

```
0    258866
1       508
Name: FATALITIES, dtype: int64
```

```
df['INJURIES'] = df['MAJORINJURIES_BICYCLIST'] +
df['MINORINJURIES_BICYCLIST'] + df['UNKNOWNINJURIES_BICYCLIST'] +
df['MAJORINJURIES_DRIVER'] + df['MINORINJURIES_DRIVER'] +
df['UNKNOWNINJURIES_DRIVER'] + df['MAJORINJURIES_PEDESTRIAN'] +
df['MINORINJURIES_PEDESTRIAN'] + df['UNKNOWNINJURIES_PEDESTRIAN'] +
df['MAJORINJURIESPASSENGER'] + df['MINORINJURIESPASSENGER'] +
df['UNKNOWNINJURIESPASSENGER']
del df['MAJORINJURIES_BICYCLIST']
del df['MINORINJURIES_BICYCLIST']
del df['UNKNOWNINJURIES_BICYCLIST']
del df['MAJORINJURIES_DRIVER']
del df['MINORINJURIES_DRIVER']
del df['UNKNOWNINJURIES_DRIVER']
del df['MAJORINJURIES_PEDESTRIAN']
del df['MINORINJURIES_PEDESTRIAN']
del df['UNKNOWNINJURIES_PEDESTRIAN']
del df['MAJORINJURIESPASSENGER']
del df['MINORINJURIESPASSENGER']
del df['UNKNOWNINJURIESPASSENGER']
df.head()
```

```
       XCOORD      YCOORD  ...  FATALITIES  INJURIES
0  399314.444  129721.582  ...           0         0
1  390980.291  141229.839  ...           0         1
2  397228.190  137185.830  ...           0         1
3  396884.752  140556.727  ...           0         0
4  399489.348  137788.901  ...           0         0

[5 rows x 15 columns]
```

```
# one hot encode our categorical variable
df = pd.get_dummies(df , columns=['INTAPPROACHDIRECTION'])
```

Because our data is so unbalanced I want to undersample the data for later training.

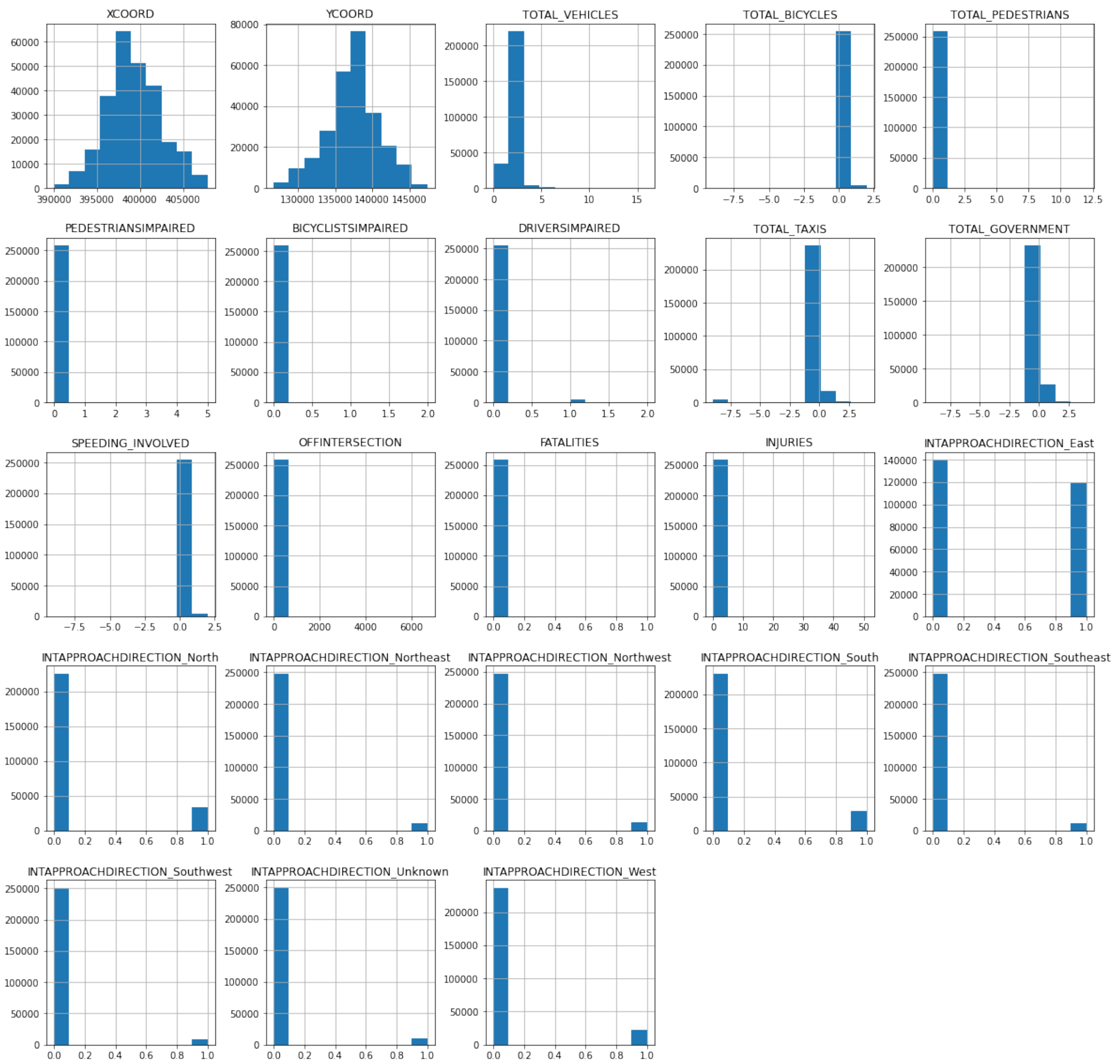
```
class_zero = df[df['FATALITIES'] == 0]
class_one = df[df['FATALITIES'] == 1]

class_zero = class_zero.sample(len(class_one))
df_undersample = pd.concat([class_one, class_zero])
df_undersample
```

```
            XCOORD  ...  INTAPPROACHDIRECTION_West
147     400638.752  ...                          0
1434    399813.992  ...                          0
1623    399244.809  ...                          0
1838    398087.686  ...                          0
2050    398299.387  ...                          0
...            ...  ...                        ...
228278  396642.978  ...                          0
15997   401944.400  ...                          0
194423  395102.153  ...                          0
165606  397325.903  ...                          1
128956  399070.097  ...                          0

[1016 rows x 23 columns]
```

```
df.hist(figsize=(20, 20))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6eee3d0>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6e93510>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6ec4550>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6e77a50>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6e27f50>],
       [<matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6dea490>,
        <matplotlib.axes._subplots.AxesSubplot object at
```

```
0x7fbea6d9f990>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6d55d90>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6d60f50>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6d21510>],
       [<matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6cfcd10>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6cbf250>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6c75750>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6c2ccd0>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6bee1d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6ba46d0>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6b5abd0>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6b1a110>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6ad2610>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6b08b10>],
       [<matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6ac9050>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6a81550>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea6a37a50>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea69ecf50>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7fbea69ae490>]],
      dtype=object)
```



Modeling

```
# Split data into training, validation, and testing sets where train
is 60% val is 20% and test is 20%

def train_val_test(df, test_size=0.2):
    """
    arguments:
    df: data frame to be split into training test and validation sets
    test_size: percent of dataframe to be allocated to test and
validation sets

    returns:
    X_train: Features to train model on
    X_val: Features of validation set
    X_test: Features of test set
```

```python
    y_train: classes to train on
    y_val: classes of validation set
    y_test: classes of test set
    """
    df.sample(frac=1).reset_index(drop=True)
    train, test = train_test_split(df, test_size=test_size)
    train, val = train_test_split(train, test_size=test_size)

    y_train = np.array(train.pop('FATALITIES'))
    y_val = np.array(val.pop('FATALITIES'))
    y_test = np.array(test.pop('FATALITIES'))

    X_train = np.array(train)
    X_val = np.array(val)
    X_test = np.array(test)
    scaler = StandardScaler()

    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)
    return X_train, X_val, X_test, y_train, y_val, y_test

from tensorflow.keras import regularizers

METRICS = [
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.Accuracy(name='accuracy')
]

def make_model(metrics=METRICS, learning_rate=0.005):
    """
    used to create a deep nn with keras
    attributes:
    metrics: List of metrics from keras to measure the performance of
the model
    learning_rate: the learning rate used for optimization

    returns the trained model
    """
    model = keras.Sequential([
        keras.layers.Dense(
            32,
            kernel_initializer='random_normal',
            activation='relu',
            input_shape=(X_train.shape[-1],),),
        keras.layers.Dropout(.2),
        keras.layers.Dense(1, activation='sigmoid')

    ])
```

```
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss=keras.losses.BinaryCrossentropy(),
        metrics=metrics)

    return model
```

```
X_train, X_val, X_test, y_train, y_val, y_test =
train_val_test(df_undersample, 0.2)
```

```
model = make_model()
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 32)                736

 dropout_6 (Dropout)         (None, 32)                0

 dense_13 (Dense)            (None, 1)                 33

=================================================================
Total params: 769
Trainable params: 769
Non-trainable params: 0
_________________________________________________________________
```

```
model = make_model(learning_rate=.05)
history = model.fit(
    X_train,
    y_train,
    batch_size=4096,
    epochs=100,
    validation_data=(X_val, y_val))
```

```
Epoch 1/100
1/1 [==============================] - 1s 1s/step - loss: 0.7042 -
precision: 0.4759 - recall: 0.6330 - accuracy: 2.1353e-04 - val_loss:
0.6066 - val_precision: 0.7222 - val_recall: 0.6265 - val_accuracy:
0.0000e+00
Epoch 2/100
1/1 [==============================] - 0s 28ms/step - loss: 0.6359 -
precision: 0.7178 - recall: 0.4503 - accuracy: 0.0000e+00 - val_loss:
0.5738 - val_precision: 0.7200 - val_recall: 0.6506 - val_accuracy:
0.0000e+00
Epoch 3/100
1/1 [==============================] - 0s 28ms/step - loss: 0.6092 -
precision: 0.7189 - recall: 0.5559 - accuracy: 0.0000e+00 - val_loss:
```

```
0.5586 - val_precision: 0.7237 - val_recall: 0.6627 - val_accuracy:
0.0000e+00
Epoch 4/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5921 -
precision: 0.7206 - recall: 0.5528 - accuracy: 0.0000e+00 - val_loss:
0.5525 - val_precision: 0.7571 - val_recall: 0.6386 - val_accuracy:
0.0000e+00
Epoch 5/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5807 -
precision: 0.7617 - recall: 0.5559 - accuracy: 0.0000e+00 - val_loss:
0.5500 - val_precision: 0.7571 - val_recall: 0.6386 - val_accuracy:
0.0000e+00
Epoch 6/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5640 -
precision: 0.7863 - recall: 0.5714 - accuracy: 0.0015 - val_loss:
0.5534 - val_precision: 0.7612 - val_recall: 0.6145 - val_accuracy:
0.0061
Epoch 7/100
1/1 [==============================] - 0s 28ms/step - loss: 0.5652 -
precision: 0.7897 - recall: 0.5714 - accuracy: 0.0000e+00 - val_loss:
0.5591 - val_precision: 0.7656 - val_recall: 0.5904 - val_accuracy:
0.0061
Epoch 8/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5635 -
precision: 0.7689 - recall: 0.5683 - accuracy: 0.0046 - val_loss:
0.5653 - val_precision: 0.7429 - val_recall: 0.6265 - val_accuracy:
0.0123
Epoch 9/100
1/1 [==============================] - 0s 28ms/step - loss: 0.5523 -
precision: 0.7430 - recall: 0.5745 - accuracy: 0.0031 - val_loss:
0.5721 - val_precision: 0.7465 - val_recall: 0.6386 - val_accuracy:
0.0123
Epoch 10/100
1/1 [==============================] - 0s 26ms/step - loss: 0.5451 -
precision: 0.7769 - recall: 0.6056 - accuracy: 0.0031 - val_loss:
0.5764 - val_precision: 0.7465 - val_recall: 0.6386 - val_accuracy:
0.0123
Epoch 11/100
1/1 [==============================] - 0s 25ms/step - loss: 0.5467 -
precision: 0.7725 - recall: 0.6118 - accuracy: 0.0062 - val_loss:
0.5790 - val_precision: 0.7794 - val_recall: 0.6386 - val_accuracy:
0.0061
Epoch 12/100
1/1 [==============================] - 0s 31ms/step - loss: 0.5385 -
precision: 0.7950 - recall: 0.5901 - accuracy: 0.0046 - val_loss:
0.5851 - val_precision: 0.7576 - val_recall: 0.6024 - val_accuracy:
0.0061
Epoch 13/100
1/1 [==============================] - 0s 26ms/step - loss: 0.5198 -
precision: 0.8063 - recall: 0.6335 - accuracy: 0.0077 - val_loss:
```

```
0.5916 - val_precision: 0.7727 - val_recall: 0.6145 - val_accuracy:
0.0061
Epoch 14/100
1/1 [==============================] - 0s 36ms/step - loss: 0.5170 -
precision: 0.8061 - recall: 0.6584 - accuracy: 0.0046 - val_loss:
0.5976 - val_precision: 0.7465 - val_recall: 0.6386 - val_accuracy:
0.0061
Epoch 15/100
1/1 [==============================] - 0s 25ms/step - loss: 0.5159 -
precision: 0.7985 - recall: 0.6646 - accuracy: 0.0062 - val_loss:
0.6010 - val_precision: 0.7260 - val_recall: 0.6386 - val_accuracy:
0.0061
Epoch 16/100
1/1 [==============================] - 0s 33ms/step - loss: 0.4971 -
precision: 0.8052 - recall: 0.6677 - accuracy: 0.0062 - val_loss:
0.5972 - val_precision: 0.7465 - val_recall: 0.6386 - val_accuracy:
0.0061
Epoch 17/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5182 -
precision: 0.8007 - recall: 0.6739 - accuracy: 0.0077 - val_loss:
0.5885 - val_precision: 0.7500 - val_recall: 0.6145 - val_accuracy:
0.0061
Epoch 18/100
1/1 [==============================] - 0s 28ms/step - loss: 0.5193 -
precision: 0.8071 - recall: 0.6366 - accuracy: 0.0077 - val_loss:
0.5904 - val_precision: 0.7324 - val_recall: 0.6265 - val_accuracy:
0.0061
Epoch 19/100
1/1 [==============================] - 0s 30ms/step - loss: 0.5038 -
precision: 0.7871 - recall: 0.6429 - accuracy: 0.0077 - val_loss:
0.5969 - val_precision: 0.6962 - val_recall: 0.6627 - val_accuracy:
0.0123
Epoch 20/100
1/1 [==============================] - 0s 39ms/step - loss: 0.4921 -
precision: 0.8030 - recall: 0.6708 - accuracy: 0.0092 - val_loss:
0.6038 - val_precision: 0.6914 - val_recall: 0.6747 - val_accuracy:
0.0123
Epoch 21/100
1/1 [==============================] - 0s 27ms/step - loss: 0.5041 -
precision: 0.7914 - recall: 0.6832 - accuracy: 0.0062 - val_loss:
0.6076 - val_precision: 0.6824 - val_recall: 0.6988 - val_accuracy:
0.0123
Epoch 22/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4944 -
precision: 0.7935 - recall: 0.6801 - accuracy: 0.0092 - val_loss:
0.6056 - val_precision: 0.7089 - val_recall: 0.6747 - val_accuracy:
0.0123
Epoch 23/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4798 -
precision: 0.8029 - recall: 0.6832 - accuracy: 0.0092 - val_loss:
```

```
0.6041 - val_precision: 0.7260 - val_recall: 0.6386 - val_accuracy:
0.0123
Epoch 24/100
1/1 [==============================] - 0s 26ms/step - loss: 0.5007 -
precision: 0.8060 - recall: 0.6708 - accuracy: 0.0062 - val_loss:
0.6041 - val_precision: 0.7260 - val_recall: 0.6386 - val_accuracy:
0.0123
Epoch 25/100
1/1 [==============================] - 0s 36ms/step - loss: 0.4716 -
precision: 0.8179 - recall: 0.7112 - accuracy: 0.0062 - val_loss:
0.6042 - val_precision: 0.7397 - val_recall: 0.6506 - val_accuracy:
0.0123
Epoch 26/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4923 -
precision: 0.7801 - recall: 0.6832 - accuracy: 0.0062 - val_loss:
0.6150 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0123
Epoch 27/100
1/1 [==============================] - 0s 30ms/step - loss: 0.4800 -
precision: 0.7943 - recall: 0.6957 - accuracy: 0.0077 - val_loss:
0.6227 - val_precision: 0.6951 - val_recall: 0.6867 - val_accuracy:
0.0184
Epoch 28/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4708 -
precision: 0.7808 - recall: 0.7081 - accuracy: 0.0092 - val_loss:
0.6264 - val_precision: 0.7089 - val_recall: 0.6747 - val_accuracy:
0.0184
Epoch 29/100
1/1 [==============================] - 0s 32ms/step - loss: 0.4877 -
precision: 0.7852 - recall: 0.6925 - accuracy: 0.0077 - val_loss:
0.6230 - val_precision: 0.7273 - val_recall: 0.6747 - val_accuracy:
0.0184
Epoch 30/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4673 -
precision: 0.8127 - recall: 0.7143 - accuracy: 0.0092 - val_loss:
0.6215 - val_precision: 0.7671 - val_recall: 0.6747 - val_accuracy:
0.0184
Epoch 31/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4793 -
precision: 0.8353 - recall: 0.6615 - accuracy: 0.0077 - val_loss:
0.6420 - val_precision: 0.7703 - val_recall: 0.6867 - val_accuracy:
0.0184
Epoch 32/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4593 -
precision: 0.8480 - recall: 0.6584 - accuracy: 0.0108 - val_loss:
0.6683 - val_precision: 0.7500 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 33/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4596 -
precision: 0.8085 - recall: 0.7081 - accuracy: 0.0108 - val_loss:
```

```
0.6840 - val_precision: 0.7308 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 34/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4398 -
precision: 0.8339 - recall: 0.7329 - accuracy: 0.0108 - val_loss:
0.6874 - val_precision: 0.7600 - val_recall: 0.6867 - val_accuracy:
0.0184
Epoch 35/100
1/1 [==============================] - 0s 33ms/step - loss: 0.4567 -
precision: 0.8235 - recall: 0.6957 - accuracy: 0.0108 - val_loss:
0.6885 - val_precision: 0.7403 - val_recall: 0.6867 - val_accuracy:
0.0184
Epoch 36/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4546 -
precision: 0.8485 - recall: 0.6957 - accuracy: 0.0108 - val_loss:
0.6952 - val_precision: 0.7342 - val_recall: 0.6988 - val_accuracy:
0.0184
Epoch 37/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4369 -
precision: 0.8450 - recall: 0.6770 - accuracy: 0.0123 - val_loss:
0.7126 - val_precision: 0.7215 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 38/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4463 -
precision: 0.8387 - recall: 0.7267 - accuracy: 0.0154 - val_loss:
0.7240 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 39/100
1/1 [==============================] - 0s 33ms/step - loss: 0.4285 -
precision: 0.8493 - recall: 0.7174 - accuracy: 0.0108 - val_loss:
0.7329 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 40/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4532 -
precision: 0.8285 - recall: 0.7050 - accuracy: 0.0123 - val_loss:
0.7381 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 41/100
1/1 [==============================] - 0s 25ms/step - loss: 0.4570 -
precision: 0.8134 - recall: 0.6770 - accuracy: 0.0092 - val_loss:
0.7392 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0245
Epoch 42/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4392 -
precision: 0.8321 - recall: 0.7236 - accuracy: 0.0169 - val_loss:
0.7414 - val_precision: 0.7089 - val_recall: 0.6747 - val_accuracy:
0.0184
Epoch 43/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4382 -
precision: 0.8370 - recall: 0.7174 - accuracy: 0.0108 - val_loss:
```

```
0.7362 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0184
Epoch 44/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4389 -
precision: 0.8095 - recall: 0.7391 - accuracy: 0.0169 - val_loss:
0.7289 - val_precision: 0.7051 - val_recall: 0.6627 - val_accuracy:
0.0184
Epoch 45/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4261 -
precision: 0.8789 - recall: 0.6988 - accuracy: 0.0154 - val_loss:
0.7397 - val_precision: 0.7051 - val_recall: 0.6627 - val_accuracy:
0.0245
Epoch 46/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4407 -
precision: 0.8521 - recall: 0.6801 - accuracy: 0.0139 - val_loss:
0.7596 - val_precision: 0.6951 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 47/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4441 -
precision: 0.8036 - recall: 0.6863 - accuracy: 0.0154 - val_loss:
0.7755 - val_precision: 0.6867 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 48/100
1/1 [==============================] - 0s 30ms/step - loss: 0.4517 -
precision: 0.8287 - recall: 0.7360 - accuracy: 0.0169 - val_loss:
0.7750 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 49/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4328 -
precision: 0.8261 - recall: 0.7081 - accuracy: 0.0200 - val_loss:
0.7667 - val_precision: 0.7215 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 50/100
1/1 [==============================] - 0s 50ms/step - loss: 0.4210 -
precision: 0.8603 - recall: 0.7267 - accuracy: 0.0185 - val_loss:
0.7642 - val_precision: 0.7143 - val_recall: 0.6627 - val_accuracy:
0.0245
Epoch 51/100
1/1 [==============================] - 0s 30ms/step - loss: 0.4042 -
precision: 0.8504 - recall: 0.7236 - accuracy: 0.0169 - val_loss:
0.7743 - val_precision: 0.7179 - val_recall: 0.6747 - val_accuracy:
0.0245
Epoch 52/100
1/1 [==============================] - 0s 34ms/step - loss: 0.4255 -
precision: 0.8278 - recall: 0.7019 - accuracy: 0.0185 - val_loss:
0.7852 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 53/100
1/1 [==============================] - 0s 34ms/step - loss: 0.4207 -
precision: 0.8214 - recall: 0.7143 - accuracy: 0.0185 - val_loss:
```

```
0.7955 - val_precision: 0.6867 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 54/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4323 -
precision: 0.8225 - recall: 0.7050 - accuracy: 0.0231 - val_loss:
0.7942 - val_precision: 0.6951 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 55/100
1/1 [==============================] - 0s 32ms/step - loss: 0.4213 -
precision: 0.8502 - recall: 0.7050 - accuracy: 0.0154 - val_loss:
0.7878 - val_precision: 0.6951 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 56/100
1/1 [==============================] - 0s 30ms/step - loss: 0.4203 -
precision: 0.8345 - recall: 0.7205 - accuracy: 0.0216 - val_loss:
0.7799 - val_precision: 0.6867 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 57/100
1/1 [==============================] - 0s 34ms/step - loss: 0.4188 -
precision: 0.8419 - recall: 0.7112 - accuracy: 0.0200 - val_loss:
0.7754 - val_precision: 0.7073 - val_recall: 0.6988 - val_accuracy:
0.0245
Epoch 58/100
1/1 [==============================] - 0s 33ms/step - loss: 0.4344 -
precision: 0.8496 - recall: 0.7019 - accuracy: 0.0231 - val_loss:
0.7842 - val_precision: 0.7024 - val_recall: 0.7108 - val_accuracy:
0.0245
Epoch 59/100
1/1 [==============================] - 0s 27ms/step - loss: 0.4056 -
precision: 0.8509 - recall: 0.7267 - accuracy: 0.0262 - val_loss:
0.7949 - val_precision: 0.6860 - val_recall: 0.7108 - val_accuracy:
0.0245
Epoch 60/100
1/1 [==============================] - 0s 32ms/step - loss: 0.4027 -
precision: 0.8430 - recall: 0.7671 - accuracy: 0.0216 - val_loss:
0.7914 - val_precision: 0.6860 - val_recall: 0.7108 - val_accuracy:
0.0245
Epoch 61/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4084 -
precision: 0.8339 - recall: 0.7640 - accuracy: 0.0262 - val_loss:
0.7680 - val_precision: 0.7160 - val_recall: 0.6988 - val_accuracy:
0.0245
Epoch 62/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4125 -
precision: 0.8303 - recall: 0.7143 - accuracy: 0.0247 - val_loss:
0.7544 - val_precision: 0.7733 - val_recall: 0.6988 - val_accuracy:
0.0245
Epoch 63/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4070 -
precision: 0.8513 - recall: 0.7112 - accuracy: 0.0231 - val_loss:
```

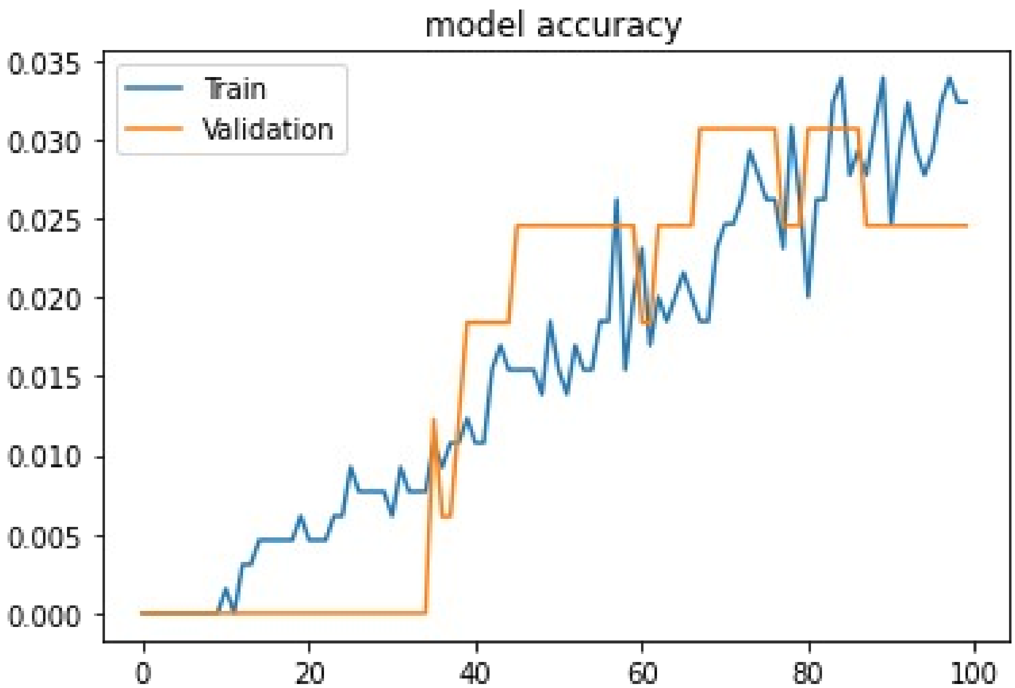
```
0.7793 - val_precision: 0.7746 - val_recall: 0.6627 - val_accuracy:
0.0245
Epoch 64/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4274 -
precision: 0.8405 - recall: 0.6708 - accuracy: 0.0262 - val_loss:
0.8181 - val_precision: 0.7534 - val_recall: 0.6627 - val_accuracy:
0.0245
Epoch 65/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4111 -
precision: 0.8467 - recall: 0.7205 - accuracy: 0.0293 - val_loss:
0.8649 - val_precision: 0.6914 - val_recall: 0.6747 - val_accuracy:
0.0307
Epoch 66/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4045 -
precision: 0.8299 - recall: 0.7578 - accuracy: 0.0277 - val_loss:
0.8963 - val_precision: 0.6667 - val_recall: 0.6988 - val_accuracy:
0.0307
Epoch 67/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4049 -
precision: 0.8299 - recall: 0.7578 - accuracy: 0.0431 - val_loss:
0.9000 - val_precision: 0.6905 - val_recall: 0.6988 - val_accuracy:
0.0307
Epoch 68/100
1/1 [==============================] - 0s 28ms/step - loss: 0.4109 -
precision: 0.8475 - recall: 0.7422 - accuracy: 0.0308 - val_loss:
0.8915 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0245
Epoch 69/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4069 -
precision: 0.8788 - recall: 0.7205 - accuracy: 0.0308 - val_loss:
0.9039 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0307
Epoch 70/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4103 -
precision: 0.8491 - recall: 0.6988 - accuracy: 0.0277 - val_loss:
0.9198 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0307
Epoch 71/100
1/1 [==============================] - 0s 33ms/step - loss: 0.4061 -
precision: 0.8556 - recall: 0.7174 - accuracy: 0.0324 - val_loss:
0.9350 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0368
Epoch 72/100
1/1 [==============================] - 0s 37ms/step - loss: 0.3916 -
precision: 0.8689 - recall: 0.7205 - accuracy: 0.0339 - val_loss:
0.9566 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0429
Epoch 73/100
1/1 [==============================] - 0s 46ms/step - loss: 0.3968 -
precision: 0.8504 - recall: 0.7236 - accuracy: 0.0416 - val_loss:
```

```
0.9767 - val_precision: 0.6905 - val_recall: 0.6988 - val_accuracy:
0.0429
Epoch 74/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4085 -
precision: 0.8213 - recall: 0.7422 - accuracy: 0.0370 - val_loss:
0.9886 - val_precision: 0.6905 - val_recall: 0.6988 - val_accuracy:
0.0368
Epoch 75/100
1/1 [==============================] - 0s 31ms/step - loss: 0.3977 -
precision: 0.8245 - recall: 0.7733 - accuracy: 0.0462 - val_loss:
0.9837 - val_precision: 0.6905 - val_recall: 0.6988 - val_accuracy:
0.0368
Epoch 76/100
1/1 [==============================] - 0s 26ms/step - loss: 0.3991 -
precision: 0.8419 - recall: 0.7609 - accuracy: 0.0431 - val_loss:
0.9688 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0368
Epoch 77/100
1/1 [==============================] - 0s 32ms/step - loss: 0.4019 -
precision: 0.8451 - recall: 0.7453 - accuracy: 0.0416 - val_loss:
0.9563 - val_precision: 0.7273 - val_recall: 0.6747 - val_accuracy:
0.0368
Epoch 78/100
1/1 [==============================] - 0s 28ms/step - loss: 0.3957 -
precision: 0.8740 - recall: 0.7112 - accuracy: 0.0370 - val_loss:
0.9555 - val_precision: 0.7467 - val_recall: 0.6747 - val_accuracy:
0.0368
Epoch 79/100
1/1 [==============================] - 0s 29ms/step - loss: 0.4048 -
precision: 0.8819 - recall: 0.6957 - accuracy: 0.0416 - val_loss:
0.9795 - val_precision: 0.7089 - val_recall: 0.6747 - val_accuracy:
0.0429
Epoch 80/100
1/1 [==============================] - 0s 31ms/step - loss: 0.4263 -
precision: 0.8212 - recall: 0.6988 - accuracy: 0.0462 - val_loss:
1.0044 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0429
Epoch 81/100
1/1 [==============================] - 0s 30ms/step - loss: 0.4033 -
precision: 0.8333 - recall: 0.7609 - accuracy: 0.0431 - val_loss:
1.0096 - val_precision: 0.6905 - val_recall: 0.6988 - val_accuracy:
0.0429
Epoch 82/100
1/1 [==============================] - 0s 26ms/step - loss: 0.3952 -
precision: 0.8300 - recall: 0.7733 - accuracy: 0.0370 - val_loss:
0.9906 - val_precision: 0.6786 - val_recall: 0.6867 - val_accuracy:
0.0491
Epoch 83/100
1/1 [==============================] - 0s 41ms/step - loss: 0.3881 -
precision: 0.8587 - recall: 0.7360 - accuracy: 0.0370 - val_loss:
```

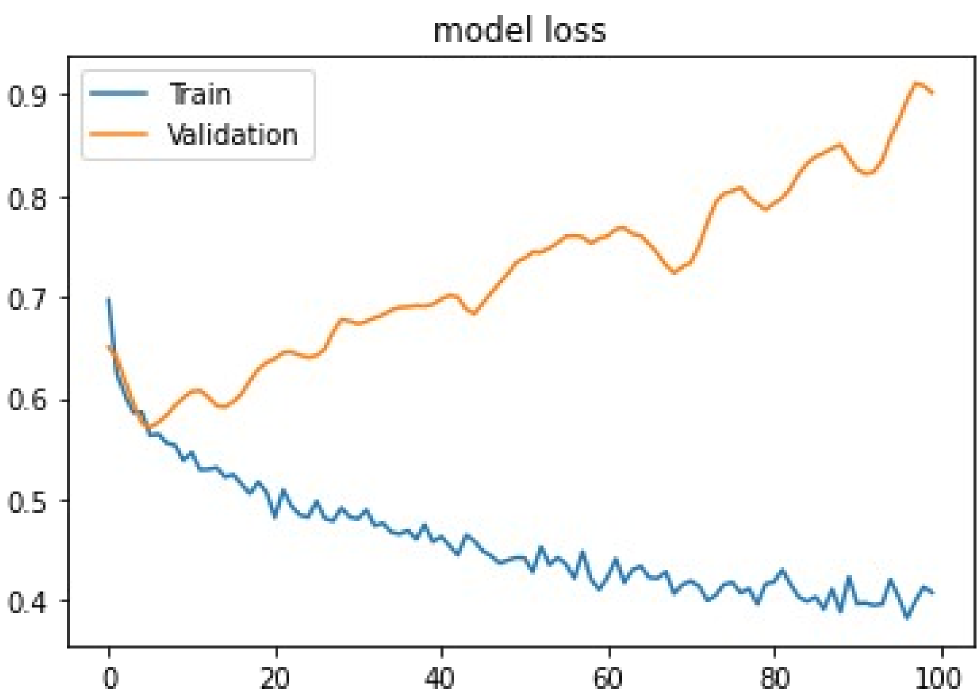
```
0.9684 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0552
Epoch 84/100
1/1 [==============================] - 0s 43ms/step - loss: 0.3883 -
precision: 0.8798 - recall: 0.7050 - accuracy: 0.0431 - val_loss:
0.9559 - val_precision: 0.7179 - val_recall: 0.6747 - val_accuracy:
0.0613
Epoch 85/100
1/1 [==============================] - 0s 30ms/step - loss: 0.3992 -
precision: 0.8956 - recall: 0.6925 - accuracy: 0.0493 - val_loss:
0.9727 - val_precision: 0.7125 - val_recall: 0.6867 - val_accuracy:
0.0798
Epoch 86/100
1/1 [==============================] - 0s 29ms/step - loss: 0.3972 -
precision: 0.8636 - recall: 0.7081 - accuracy: 0.0539 - val_loss:
0.9917 - val_precision: 0.7037 - val_recall: 0.6867 - val_accuracy:
0.0859
Epoch 87/100
1/1 [==============================] - 0s 37ms/step - loss: 0.3927 -
precision: 0.8592 - recall: 0.7578 - accuracy: 0.0555 - val_loss:
1.0005 - val_precision: 0.7160 - val_recall: 0.6988 - val_accuracy:
0.0859
Epoch 88/100
1/1 [==============================] - 0s 38ms/step - loss: 0.3899 -
precision: 0.8576 - recall: 0.7671 - accuracy: 0.0586 - val_loss:
1.0028 - val_precision: 0.7073 - val_recall: 0.6988 - val_accuracy:
0.0920
Epoch 89/100
1/1 [==============================] - 0s 34ms/step - loss: 0.3872 -
precision: 0.8773 - recall: 0.7329 - accuracy: 0.0508 - val_loss:
1.0055 - val_precision: 0.7215 - val_recall: 0.6867 - val_accuracy:
0.0920
Epoch 90/100
1/1 [==============================] - 0s 31ms/step - loss: 0.3730 -
precision: 0.8537 - recall: 0.7795 - accuracy: 0.0586 - val_loss:
1.0057 - val_precision: 0.7308 - val_recall: 0.6867 - val_accuracy:
0.0982
Epoch 91/100
1/1 [==============================] - 0s 28ms/step - loss: 0.3663 -
precision: 0.8864 - recall: 0.7267 - accuracy: 0.0586 - val_loss:
1.0154 - val_precision: 0.7273 - val_recall: 0.6747 - val_accuracy:
0.0982
Epoch 92/100
1/1 [==============================] - 0s 32ms/step - loss: 0.3725 -
precision: 0.8597 - recall: 0.7422 - accuracy: 0.0616 - val_loss:
1.0375 - val_precision: 0.7000 - val_recall: 0.6747 - val_accuracy:
0.0982
Epoch 93/100
1/1 [==============================] - 0s 30ms/step - loss: 0.3816 -
precision: 0.8696 - recall: 0.7453 - accuracy: 0.0539 - val_loss:
```

```
1.0582 - val_precision: 0.7051 - val_recall: 0.6627 - val_accuracy:
0.1043
Epoch 94/100
1/1 [==============================] - 0s 36ms/step - loss: 0.3863 -
precision: 0.8278 - recall: 0.7764 - accuracy: 0.0601 - val_loss:
1.0580 - val_precision: 0.7051 - val_recall: 0.6627 - val_accuracy:
0.1043
Epoch 95/100
1/1 [==============================] - 0s 52ms/step - loss: 0.3657 -
precision: 0.8780 - recall: 0.7826 - accuracy: 0.0647 - val_loss:
1.0573 - val_precision: 0.7237 - val_recall: 0.6627 - val_accuracy:
0.1043
Epoch 96/100
1/1 [==============================] - 0s 29ms/step - loss: 0.3806 -
precision: 0.8592 - recall: 0.7578 - accuracy: 0.0632 - val_loss:
1.0478 - val_precision: 0.7500 - val_recall: 0.6506 - val_accuracy:
0.1043
Epoch 97/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3892 -
precision: 0.8812 - recall: 0.7143 - accuracy: 0.0601 - val_loss:
1.0489 - val_precision: 0.7500 - val_recall: 0.6506 - val_accuracy:
0.1043
Epoch 98/100
1/1 [==============================] - 0s 38ms/step - loss: 0.3719 -
precision: 0.8815 - recall: 0.7391 - accuracy: 0.0539 - val_loss:
1.0611 - val_precision: 0.7432 - val_recall: 0.6627 - val_accuracy:
0.1043
Epoch 99/100
1/1 [==============================] - 0s 41ms/step - loss: 0.3801 -
precision: 0.8817 - recall: 0.7640 - accuracy: 0.0616 - val_loss:
1.0627 - val_precision: 0.7143 - val_recall: 0.6627 - val_accuracy:
0.0982
Epoch 100/100
1/1 [==============================] - 0s 37ms/step - loss: 0.3608 -
precision: 0.8562 - recall: 0.7764 - accuracy: 0.0586 - val_loss:
1.0511 - val_precision: 0.7179 - val_recall: 0.6747 - val_accuracy:
0.0982
```

```python
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



I was unable to reduce over fitting with this model, Even with a balanced data set the accuracy was just too low for this to be a realistic model.

```
def grid_classifier(model, params):
    """
    used to fit multiple classifiers quickly so that I can test them
with grid search
    attributes:
    model: What model to train
    params: dictionary containing parameters and values to test with
grid search
    returns:
    nothing, but prints best estimator and its score
    """
    cv = GridSearchCV(clf,parameters,cv=5)
    cv.fit(X_train, y_train)
    print(cv.best_estimator_)
    print(cv.best_score_)
```

The following models were chosen following the sklearn estimator flow chart.

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV

clf = svm.SVC()
parameters = {
    "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],
    "C":[0.001, 0.01, 0.1, 10]
}
grid_classifier(clf, parameters)
```

```
SVC(C=0.01, kernel='linear')
0.6579487179487179
```

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
parameters = {
    "max_depth": [2, 10, 100, 1000, None],
    "n_estimators": [1, 10, 100, 1000]
}
grid_classifier(clf, parameters)
```

```
RandomForestClassifier(max_depth=2, n_estimators=1000)
0.6887418008348241
```

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0)
parameters = {
    "C": [0.01, 0.1, 1, 10, 100]
```

```
    }
grid_classifier(clf, parameters)
```

```
LogisticRegression(C=10, random_state=0)
0.6594156231365534
```

```
clf = RandomForestClassifier(max_depth=1000, n_estimators=10)
clf.fit(X_train, y_train)
clf.score(X_val, y_val)
```

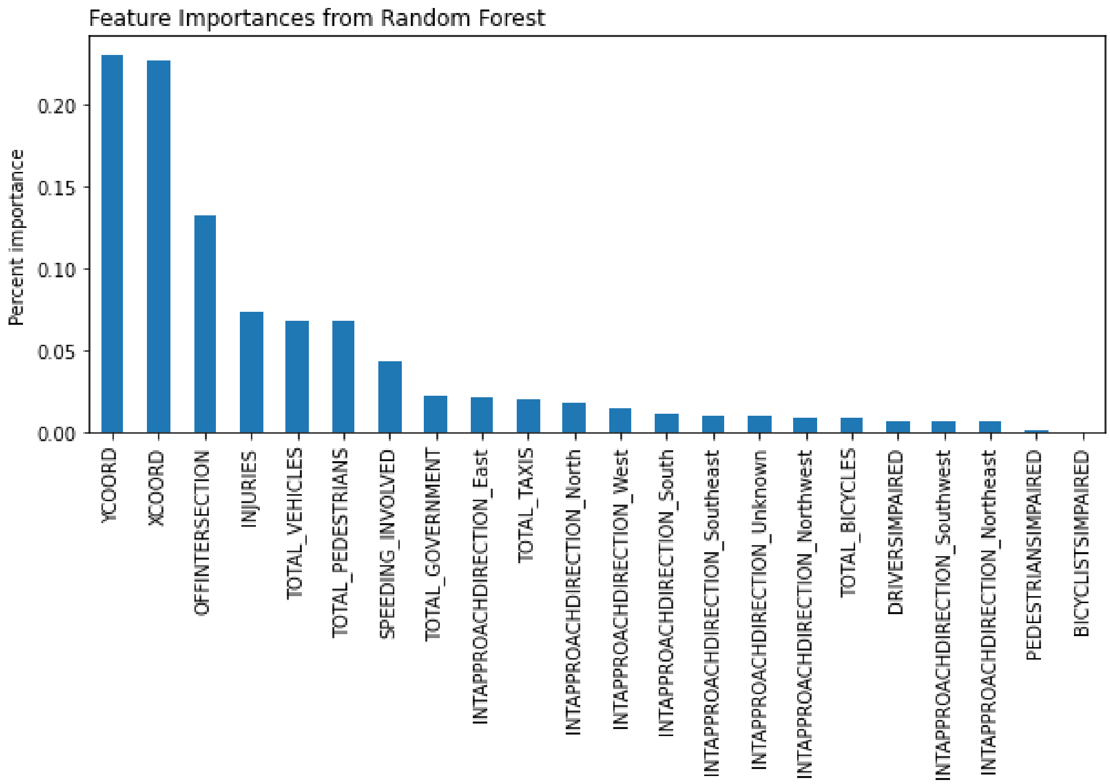
```
0.6380368098159509
```

Results

```
from sklearn.metrics import classification_report
y_val_pred = clf.predict(X_val)
print(classification_report(y_val, y_val_pred))
```

```
              precision    recall  f1-score   support

           0       0.60      0.73      0.66        78
           1       0.69      0.55      0.61        85

    accuracy                           0.64       163
   macro avg       0.65      0.64      0.64       163
weighted avg       0.65      0.64      0.64       163
```

```
impt = clf.feature_importances_

feature_names = df.columns.values.tolist()
feature_names.remove('FATALITIES')
tick_locations = [*range(0, len(feature_names), 1)]
impt = list(zip(feature_names, impt))
impt = pd.DataFrame(impt)
impt.columns = ['feature', 'importance']
impt = impt.sort_values(by='importance', ascending=False)

impt.plot.bar(figsize=(10,4))
plt.title('Feature Importances from Random Forest', loc='left')
plt.xlabel('')
plt.ylabel('Percent importance')
plt.xticks(tick_locations, impt.feature)
plt.legend().remove()
plt.show()
```



The most important conclusion we can draw from our analysis is that the location of the crash plays the largest role in whether an accident will be fatal or not. This is striking because location is more imprtant even than whether or not the driver was impaired.

Next Steps

Because we can see that the most relevant features to whether or not some one died in an accident was where that accident took place the first thing I would do is try to find out where the fatal accidents are taking place. Then a survey can be done to try and determine why these areas are so lethal.