Grab x Microsoft Future Ready Skills Challenge 2020

Safety Challenge

Done by:
Team 2 - Chuan Hao, David, Sherisse

About Us

Chuan Hao

David

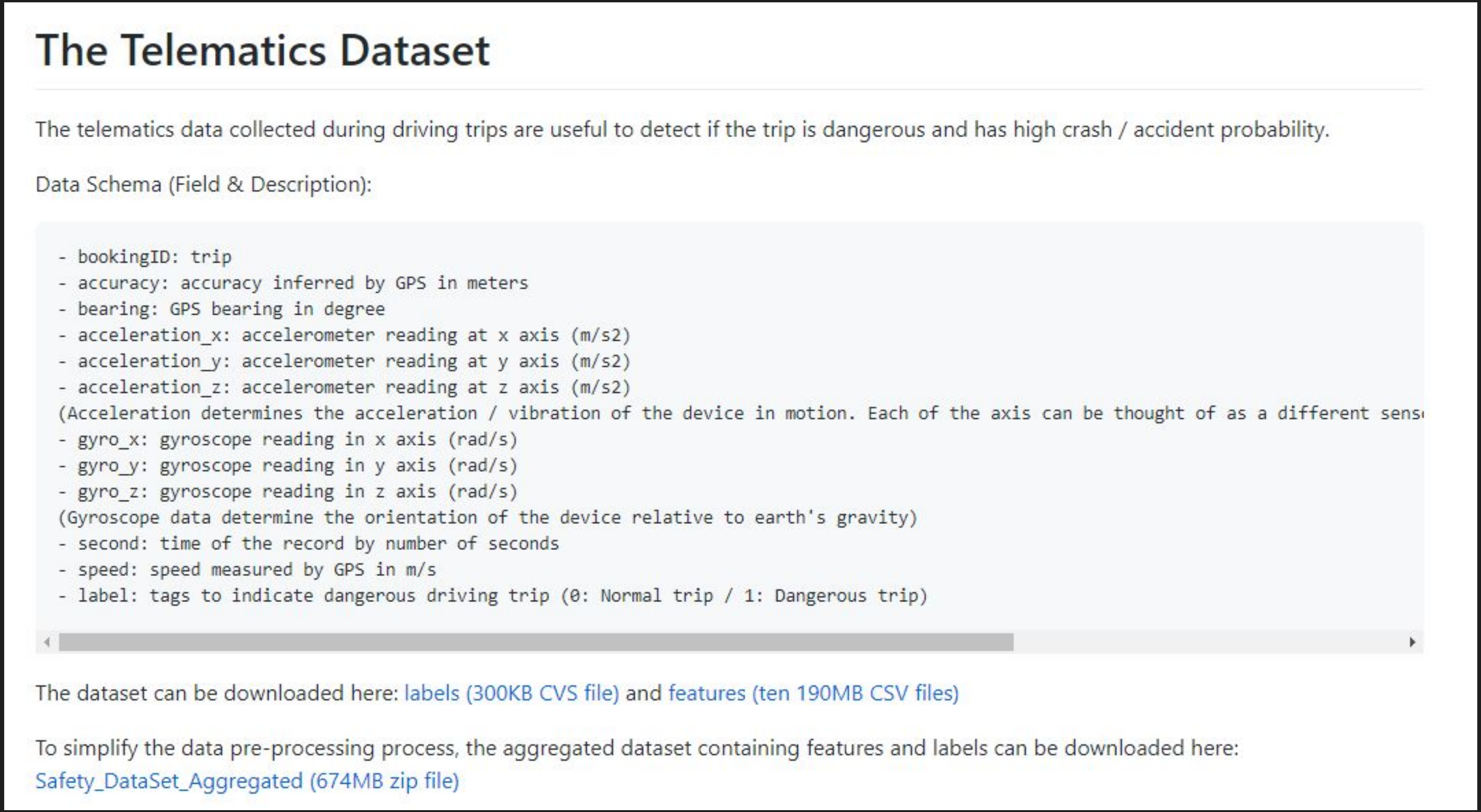
Sherisse

Dr Peter Leong
(Mentor)

Overview

a. Introduction to challenge
b. Preparing for the problem statement
c. Problem Statements
d. Overall Conclusion

Introduction to challenge



The Telematics Dataset

The telematics data collected during driving trips are useful to detect if the trip is dangerous and has high crash / accident probability.

Data Schema (Field & Description):

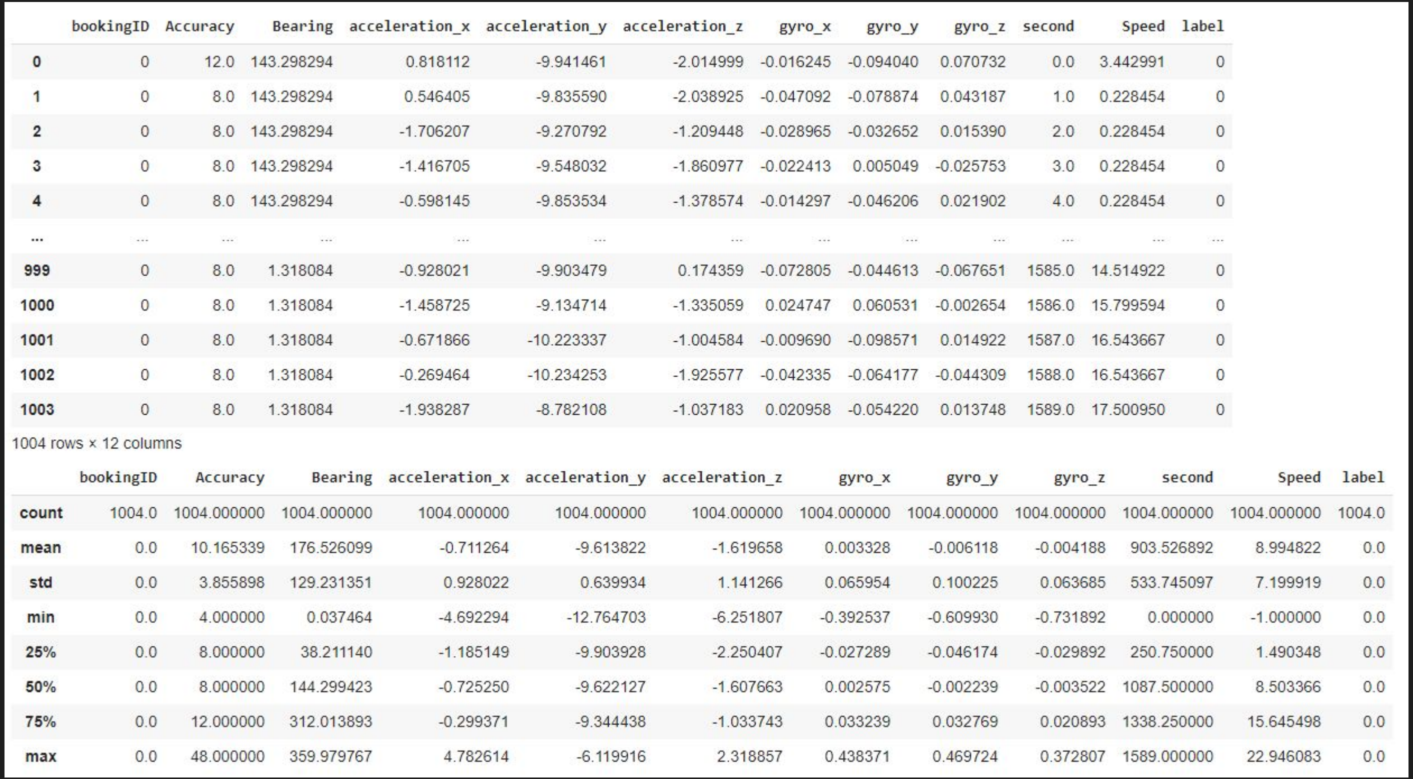
```
- bookingID: trip
- accuracy: accuracy inferred by GPS in meters
- bearing: GPS bearing in degree
- acceleration_x: accelerometer reading at x axis (m/s2)
- acceleration_y: accelerometer reading at y axis (m/s2)
- acceleration_z: accelerometer reading at z axis (m/s2)
(Acceleration determines the acceleration / vibration of the device in motion. Each of the axis can be thought of as a different sens
- gyro_x: gyroscope reading in x axis (rad/s)
- gyro_y: gyroscope reading in y axis (rad/s)
- gyro_z: gyroscope reading in z axis (rad/s)
(Gyroscope data determine the orientation of the device relative to earth's gravity)
- second: time of the record by number of seconds
- speed: speed measured by GPS in m/s
- label: tags to indicate dangerous driving trip (0: Normal trip / 1: Dangerous trip)
```

The dataset can be downloaded here: labels (300KB CVS file) and features (ten 190MB CSV files)

To simplify the data pre-processing process, the aggregated dataset containing features and labels can be downloaded here: Safety_DataSet_Aggregated (674MB zip file)

Exploratory Data Analysis

- Looking at the raw dataset



	bookingID	Accuracy	Bearing	acceleration_x	acceleration_y	acceleration_z	gyro_x	gyro_y	gyro_z	second	Speed	label
0	0	12.0	143.298294	0.818112	-9.941461	-2.014999	-0.016245	-0.094040	0.070732	0.0	3.442991	0
1	0	8.0	143.298294	0.546405	-9.835590	-2.038925	-0.047092	-0.078874	0.043187	1.0	0.228454	0
2	0	8.0	143.298294	-1.706207	-9.270792	-1.209448	-0.028965	-0.032652	0.015390	2.0	0.228454	0
3	0	8.0	143.298294	-1.416705	-9.548032	-1.860977	-0.022413	0.005049	-0.025753	3.0	0.228454	0
4	0	8.0	143.298294	-0.598145	-9.853534	-1.378574	-0.014297	-0.046206	0.021902	4.0	0.228454	0
...	...	...	...	...	...	...	...	...	...	...	...	...
999	0	8.0	1.318084	-0.928021	-9.903479	0.174359	-0.072805	-0.044613	-0.067651	1585.0	14.514922	0
1000	0	8.0	1.318084	-1.458725	-9.134714	-1.335059	0.024747	0.060531	-0.002654	1586.0	15.799594	0
1001	0	8.0	1.318084	-0.671866	-10.223337	-1.004584	-0.009690	-0.098571	0.014922	1587.0	16.543667	0
1002	0	8.0	1.318084	-0.269464	-10.234253	-1.925577	-0.042335	-0.064177	-0.044309	1588.0	16.543667	0
1003	0	8.0	1.318084	-1.938287	-8.782108	-1.037183	0.020958	-0.054220	0.013748	1589.0	17.500950	0

1004 rows × 12 columns

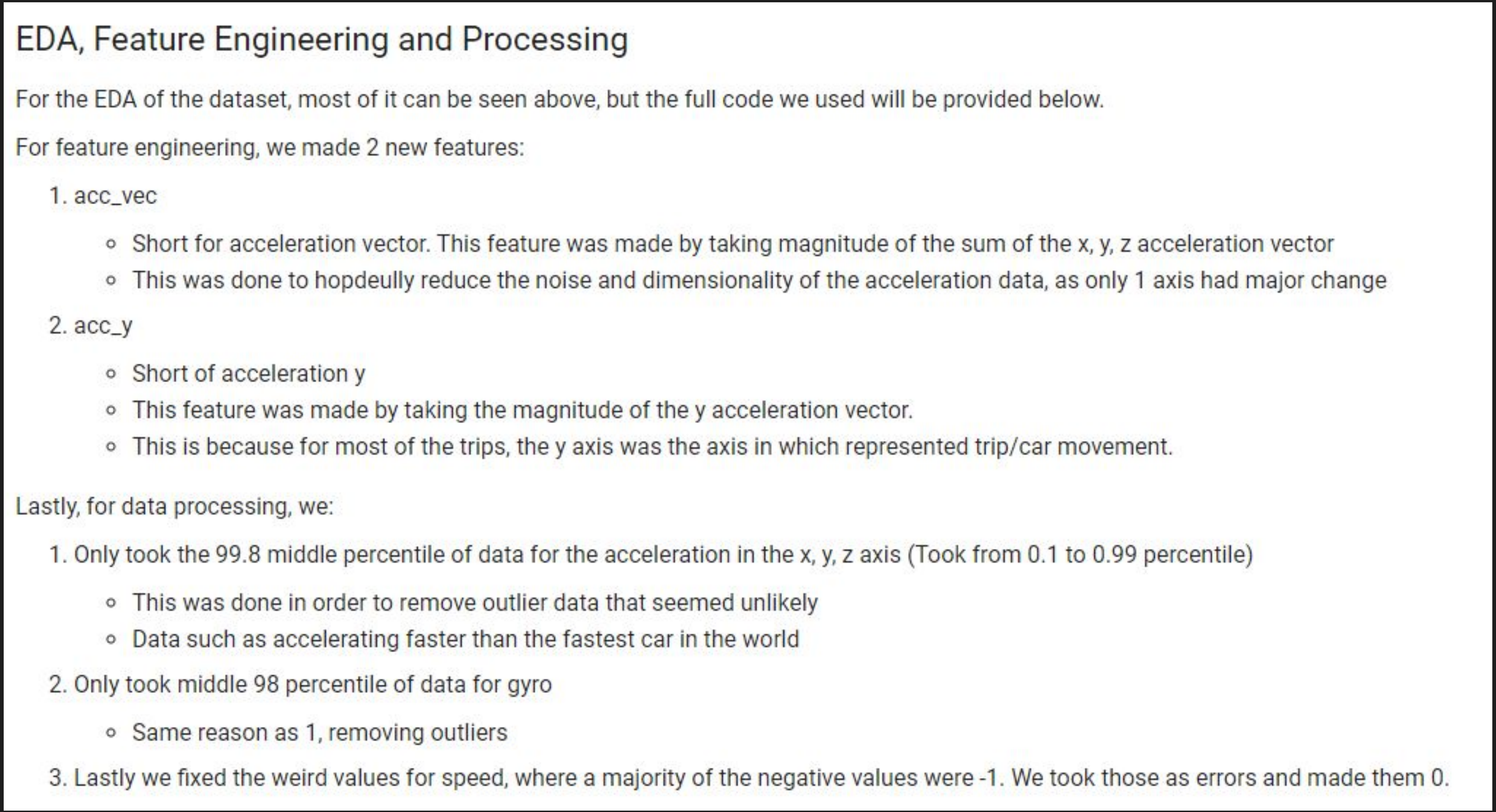
	bookingID	Accuracy	Bearing	acceleration_x	acceleration_y	acceleration_z	gyro_x	gyro_y	gyro_z	second	Speed	label
count	1004.0	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.000000	1004.0
mean	0.0	10.165339	176.526099	-0.711264	-9.613822	-1.619658	0.003328	-0.006118	-0.004188	903.526892	8.994822	0.0
std	0.0	3.855898	129.231351	0.928022	0.639934	1.141266	0.065954	0.100225	0.063685	533.745097	7.199919	0.0
min	0.0	4.000000	0.037464	-4.692294	-12.764703	-6.251807	-0.392537	-0.609930	-0.731892	0.000000	-1.000000	0.0
25%	0.0	8.000000	38.211140	-1.185149	-9.903928	-2.250407	-0.027289	-0.046174	-0.029892	250.750000	1.490348	0.0
50%	0.0	8.000000	144.299423	-0.725250	-9.622127	-1.607663	0.002575	-0.002239	-0.003522	1087.500000	8.503366	0.0
75%	0.0	12.000000	312.013893	-0.299371	-9.344438	-1.033743	0.033239	0.032769	0.020893	1338.250000	15.645498	0.0
max	0.0	48.000000	359.979767	4.782614	-6.119916	2.318857	0.438371	0.469724	0.372807	1589.000000	22.946083	0.0

Problem Statements

1. Which feature has a high impact on indicating dangerous trips?
2. How does the driving behaviour change over time (second) for normal trips and dangerous trips? What are the behaviour difference?
3. Given the telematics data for new trips, derive a model to detect if the trip is a dangerous trip.

Preparing for the problem statement

Exploratory Data Analysis



EDA, Feature Engineering and Processing

For the EDA of the dataset, most of it can be seen above, but the full code we used will be provided below.

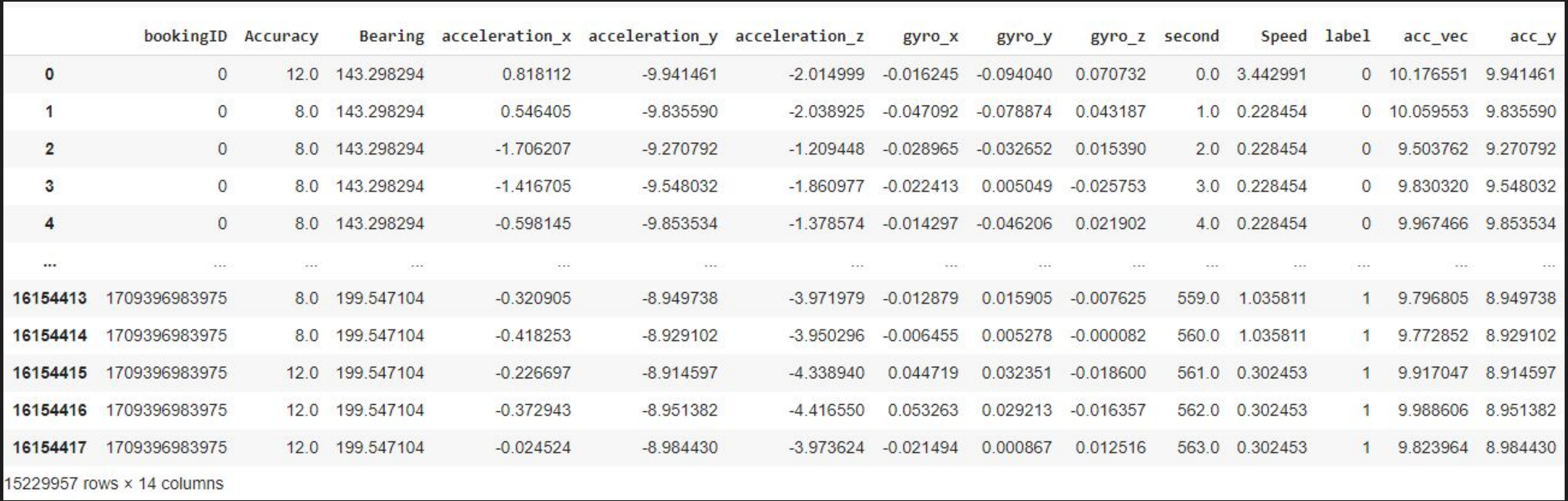
For feature engineering, we made 2 new features:

1. acc_vec
   - Short for acceleration vector. This feature was made by taking magnitude of the sum of the x, y, z acceleration vector
   - This was done to hopdeully reduce the noise and dimensionality of the acceleration data, as only 1 axis had major change
2. acc_y
   - Short of acceleration y
   - This feature was made by taking the magnitude of the y acceleration vector.
   - This is because for most of the trips, the y axis was the axis in which represented trip/car movement.

Lastly, for data processing, we:

1. Only took the 99.8 middle percentile of data for the acceleration in the x, y, z axis (Took from 0.1 to 0.99 percentile)
   - This was done in order to remove outlier data that seemed unlikely
   - Data such as accelerating faster than the fastest car in the world
2. Only took middle 98 percentile of data for gyro
   - Same reason as 1, removing outliers
3. Lastly we fixed the weird values for speed, where a majority of the negative values were -1. We took those as errors and made them 0.

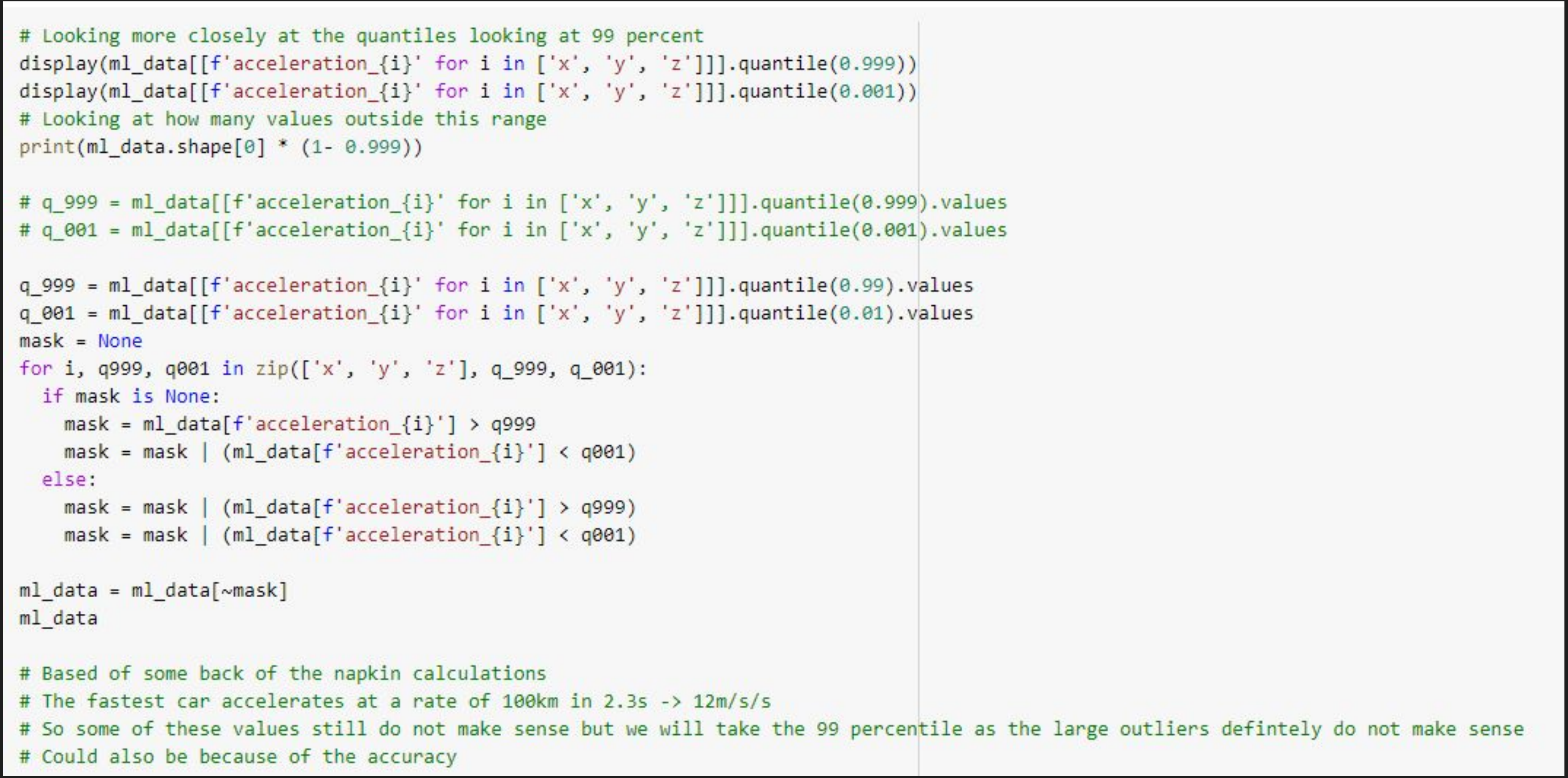
Feature engineering



	bookingID	Accuracy	Bearing	acceleration_x	acceleration_y	acceleration_z	gyro_x	gyro_y	gyro_z	second	Speed	label	acc_vec	acc_y
0	0	12.0	143.298294	0.818112	-9.941461	-2.014999	-0.016245	-0.094040	0.070732	0.0	3.442991	0	10.176551	9.941461
1	0	8.0	143.298294	0.546405	-9.835590	-2.038925	-0.047092	-0.078874	0.043187	1.0	0.228454	0	10.059553	9.835590
2	0	8.0	143.298294	-1.706207	-9.270792	-1.209448	-0.028965	-0.032652	0.015390	2.0	0.228454	0	9.503762	9.270792
3	0	8.0	143.298294	-1.416705	-9.548032	-1.860977	-0.022413	0.005049	-0.025753	3.0	0.228454	0	9.830320	9.548032
4	0	8.0	143.298294	-0.598145	-9.853534	-1.378574	-0.014297	-0.046206	0.021902	4.0	0.228454	0	9.967466	9.853534
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
16154413	1709396983975	8.0	199.547104	-0.320905	-8.949738	-3.971979	-0.012879	0.015905	-0.007625	559.0	1.035811	1	9.796805	8.949738
16154414	1709396983975	8.0	199.547104	-0.418253	-8.929102	-3.950296	-0.006455	0.005278	-0.000082	560.0	1.035811	1	9.772852	8.929102
16154415	1709396983975	12.0	199.547104	-0.226697	-8.914597	-4.338940	0.044719	0.032351	-0.018600	561.0	0.302453	1	9.917047	8.914597
16154416	1709396983975	12.0	199.547104	-0.372943	-8.951382	-4.416550	0.053263	0.029213	-0.016357	562.0	0.302453	1	9.988606	8.951382
16154417	1709396983975	12.0	199.547104	-0.024524	-8.984430	-3.973624	-0.021494	0.000867	0.012516	563.0	0.302453	1	9.823964	8.984430

15229957 rows × 14 columns

Removing outliers



```
# Looking more closely at the quantiles looking at 99 percent
display(ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.999))
display(ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.001))
# Looking at how many values outside this range
print(ml_data.shape[0] * (1- 0.999))

# q_999 = ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.999).values
# q_001 = ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.001).values

q_999 = ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.99).values
q_001 = ml_data[[f'acceleration_{i}' for i in ['x', 'y', 'z']]].quantile(0.01).values
mask = None
for i, q999, q001 in zip(['x', 'y', 'z'], q_999, q_001):
  if mask is None:
    mask = ml_data[f'acceleration_{i}'] > q999
    mask = mask | (ml_data[f'acceleration_{i}'] < q001)
  else:
    mask = mask | (ml_data[f'acceleration_{i}'] > q999)
    mask = mask | (ml_data[f'acceleration_{i}'] < q001)

ml_data = ml_data[~mask]
ml_data

# Based of some back of the napkin calculations
# The fastest car accelerates at a rate of 100km in 2.3s -> 12m/s/s
# So some of these values still do not make sense but we will take the 99 percentile as the large outliers defintely do not make sense
# Could also be because of the accuracy
```

Safety

Problem Statements:

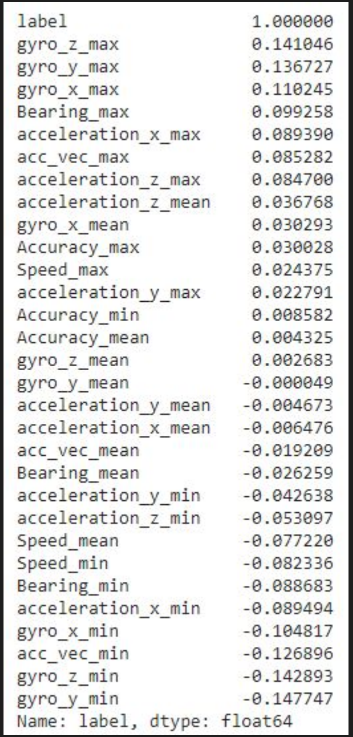
1. Which feature has a high impact on indicating dangerous trips?
2. How does the driving behaviour change over time (second) for normal trips and dangerous trips? What are the behaviour difference?
3. Given the telematics data for new trips, derive a model to detect if the trip is a dangerous trip.

Problem Statement 1

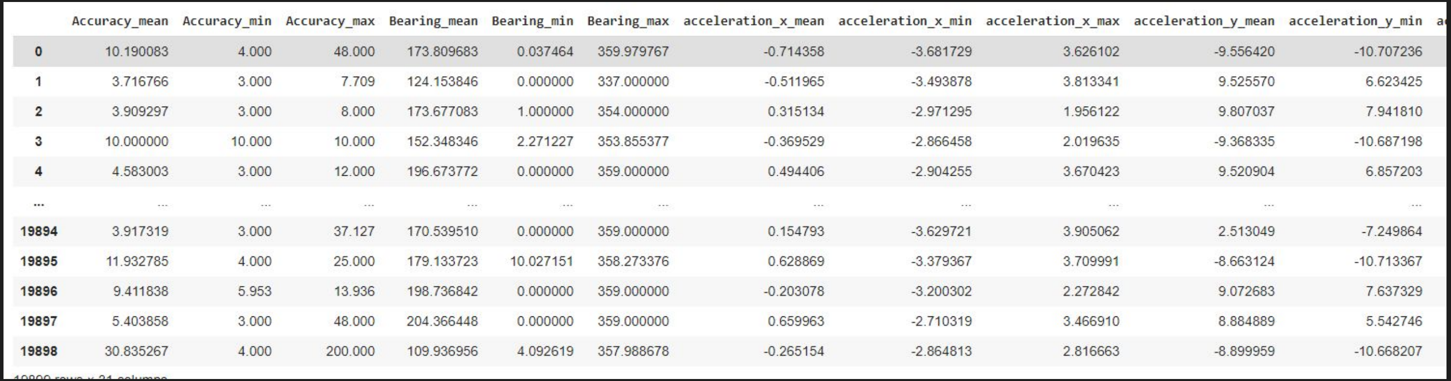
Which feature has a high impact on indicating dangerous trips?

Looking at statistical features

- Statistical Features



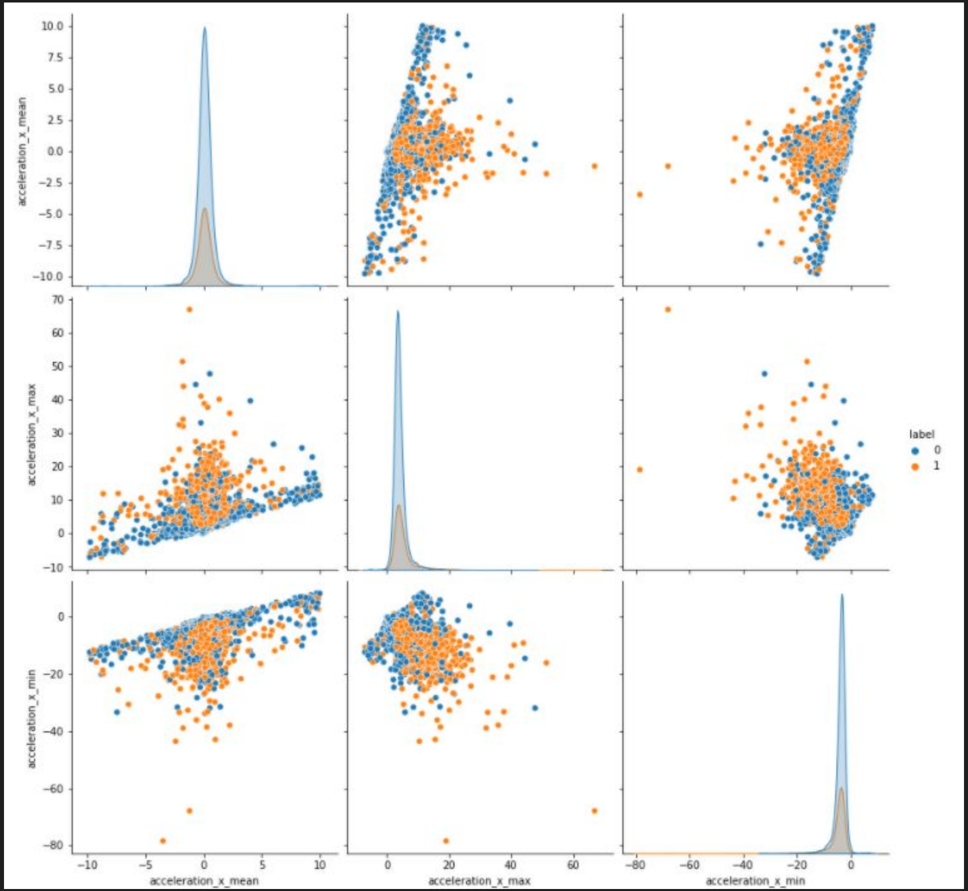
```
label                  1.000000
gyro_z_max             0.141046
gyro_y_max             0.136727
gyro_x_max             0.110245
Bearing_max            0.099258
acceleration_x_max     0.089390
acc_vec_max            0.085282
acceleration_z_max     0.084700
acceleration_z_mean    0.036768
gyro_x_mean            0.030293
Accuracy_max           0.030028
Speed_max              0.024375
acceleration_y_max     0.022791
Accuracy_min           0.008582
Accuracy_mean          0.004325
gyro_z_mean            0.002683
gyro_y_mean           -0.000049
acceleration_y_mean   -0.004673
acceleration_x_mean   -0.006476
acc_vec_mean          -0.019209
Bearing_mean          -0.026259
acceleration_y_min    -0.042638
acceleration_z_min    -0.053097
Speed_mean            -0.077220
Speed_min             -0.082336
Bearing_min           -0.088683
acceleration_x_min    -0.089494
gyro_x_min            -0.104817
acc_vec_min           -0.126896
gyro_z_min            -0.142893
gyro_y_min            -0.147747
Name: label, dtype: float64
```



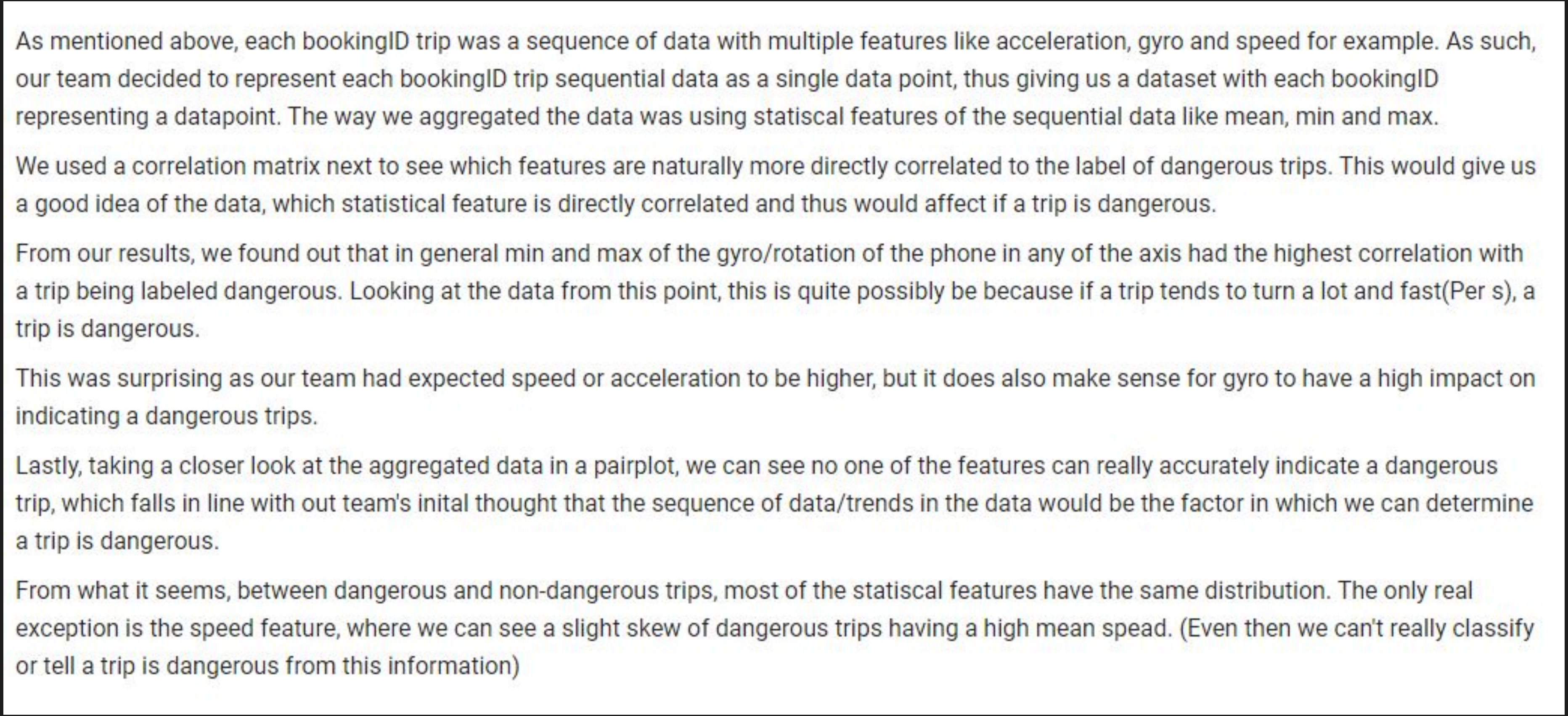
	Accuracy_mean	Accuracy_min	Accuracy_max	Bearing_mean	Bearing_min	Bearing_max	acceleration_x_mean	acceleration_x_min	acceleration_x_max	acceleration_y_mean	acceleration_y_min
0	10.190083	4.000	48.000	173.809683	0.037464	359.979767	-0.714358	-3.681729	3.626102	-9.556420	-10.707236
1	3.716766	3.000	7.709	124.153846	0.000000	337.000000	-0.511965	-3.493878	3.813341	9.525570	6.623425
2	3.909297	3.000	8.000	173.677083	1.000000	354.000000	0.315134	-2.971295	1.956122	9.807037	7.941810
3	10.000000	10.000	10.000	152.348346	2.271227	353.855377	-0.369529	-2.866458	2.019635	-9.368335	-10.687198
4	4.583003	3.000	12.000	196.673772	0.000000	359.000000	0.494406	-2.904255	3.670423	9.520904	6.857203
...	...	...	...	...	...	...	...	...	...	...	...
19894	3.917319	3.000	37.127	170.539510	0.000000	359.000000	0.154793	-3.629721	3.905062	2.513049	-7.249864
19895	11.932785	4.000	25.000	179.133723	10.027151	358.273376	0.628869	-3.379367	3.709991	-8.663124	-10.713367
19896	9.411838	5.953	13.936	198.736842	0.000000	359.000000	-0.203078	-3.200302	2.272842	9.072683	7.637329
19897	5.403858	3.000	48.000	204.366448	0.000000	359.000000	0.659963	-2.710319	3.466910	8.884889	5.542746
19898	30.835267	4.000	200.000	109.936956	4.092619	357.988678	-0.265154	-2.864813	2.816663	-8.899959	-10.668207

19899 rows × 31 columns

Taking a closer look at the distributions



Taking a closer look at the distributions



As mentioned above, each bookingID trip was a sequence of data with multiple features like acceleration, gyro and speed for example. As such, our team decided to represent each bookingID trip sequential data as a single data point, thus giving us a dataset with each bookingID representing a datapoint. The way we aggregated the data was using statiscal features of the sequential data like mean, min and max.

We used a correlation matrix next to see which features are naturally more directly correlated to the label of dangerous trips. This would give us a good idea of the data, which statistical feature is directly correlated and thus would affect if a trip is dangerous.

From our results, we found out that in general min and max of the gyro/rotation of the phone in any of the axis had the highest correlation with a trip being labeled dangerous. Looking at the data from this point, this is quite possibly be because if a trip tends to turn a lot and fast(Per s), a trip is dangerous.

This was surprising as our team had expected speed or acceleration to be higher, but it does also make sense for gyro to have a high impact on indicating a dangerous trips.

Lastly, taking a closer look at the aggregated data in a pairplot, we can see no one of the features can really accurately indicate a dangerous trip, which falls in line with out team's inital thought that the sequence of data/trends in the data would be the factor in which we can determine a trip is dangerous.

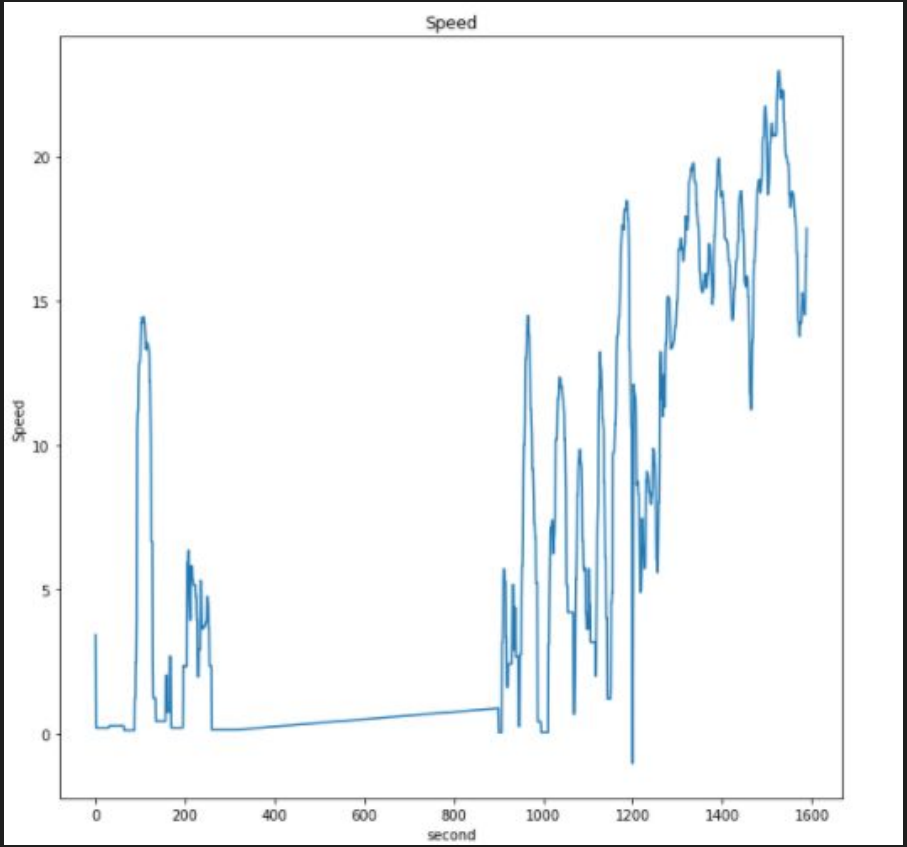
From what it seems, between dangerous and non-dangerous trips, most of the statiscal features have the same distribution. The only real exception is the speed feature, where we can see a slight skew of dangerous trips having a high mean spead. (Even then we can't really classify or tell a trip is dangerous from this information)

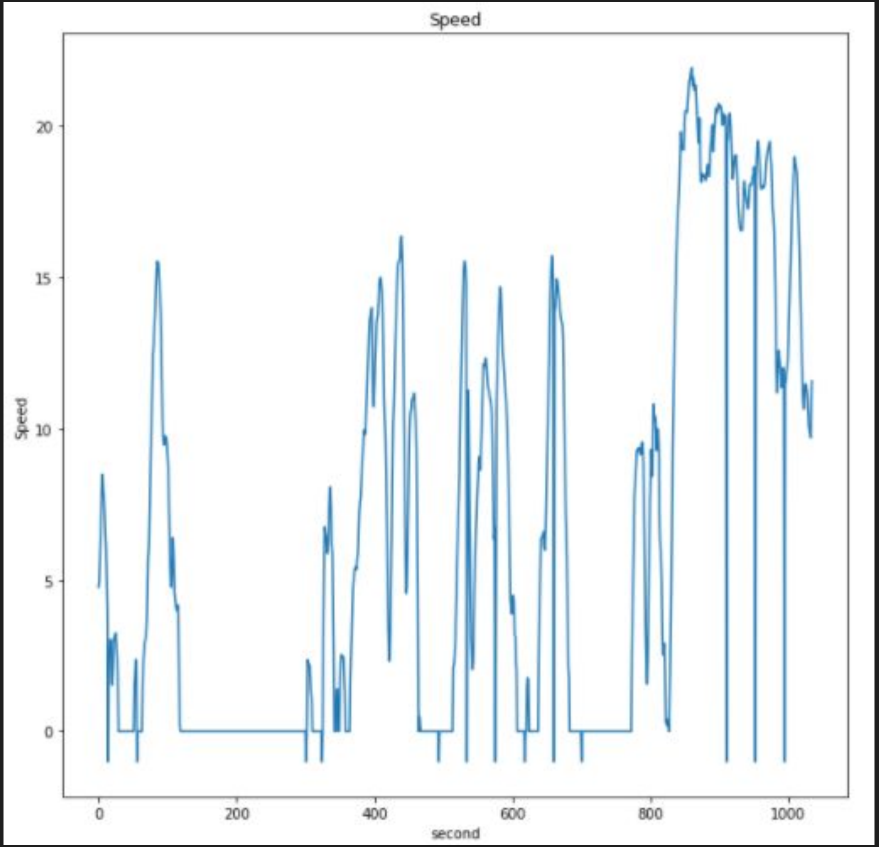
Problem Statement 2

How does the driving behaviour change over time (second) for normal trips and dangerous trips? What are the behaviour difference?

Safe

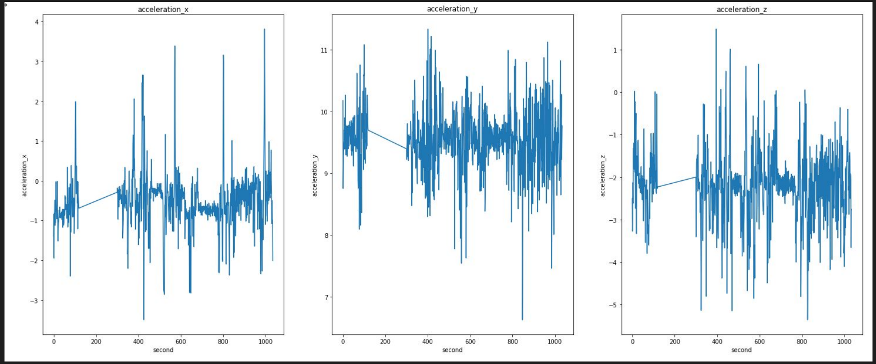
Dangerous



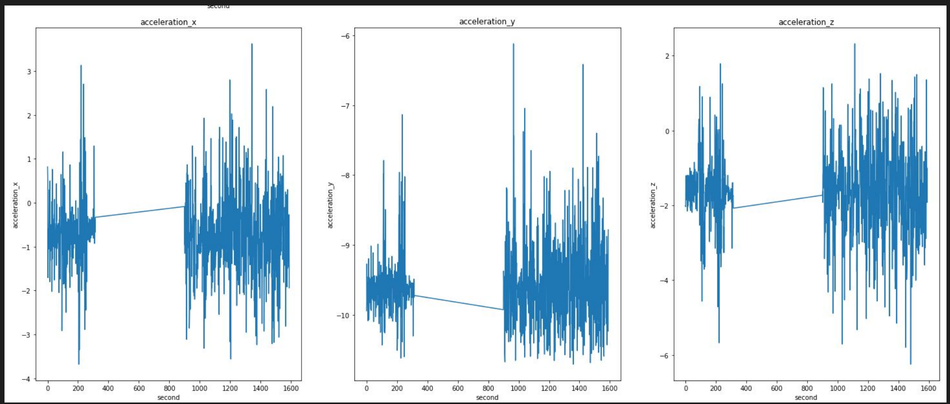


Other features

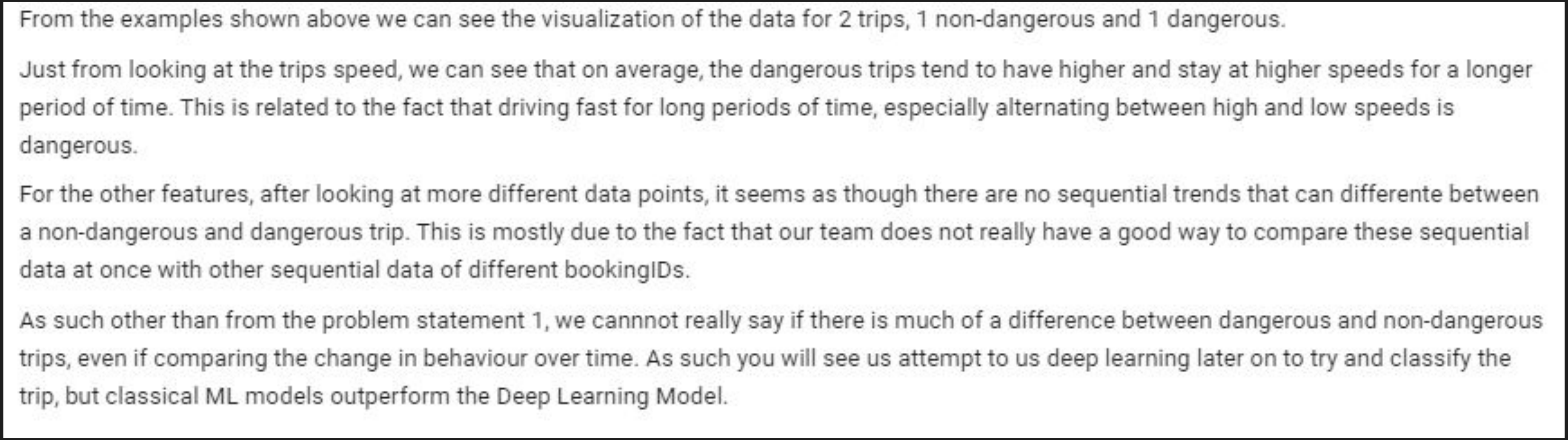
Safe



Dangerous



Conclusion



From the examples shown above we can see the visualization of the data for 2 trips, 1 non-dangerous and 1 dangerous.

Just from looking at the trips speed, we can see that on average, the dangerous trips tend to have higher and stay at higher speeds for a longer period of time. This is related to the fact that driving fast for long periods of time, especially alternating between high and low speeds is dangerous.

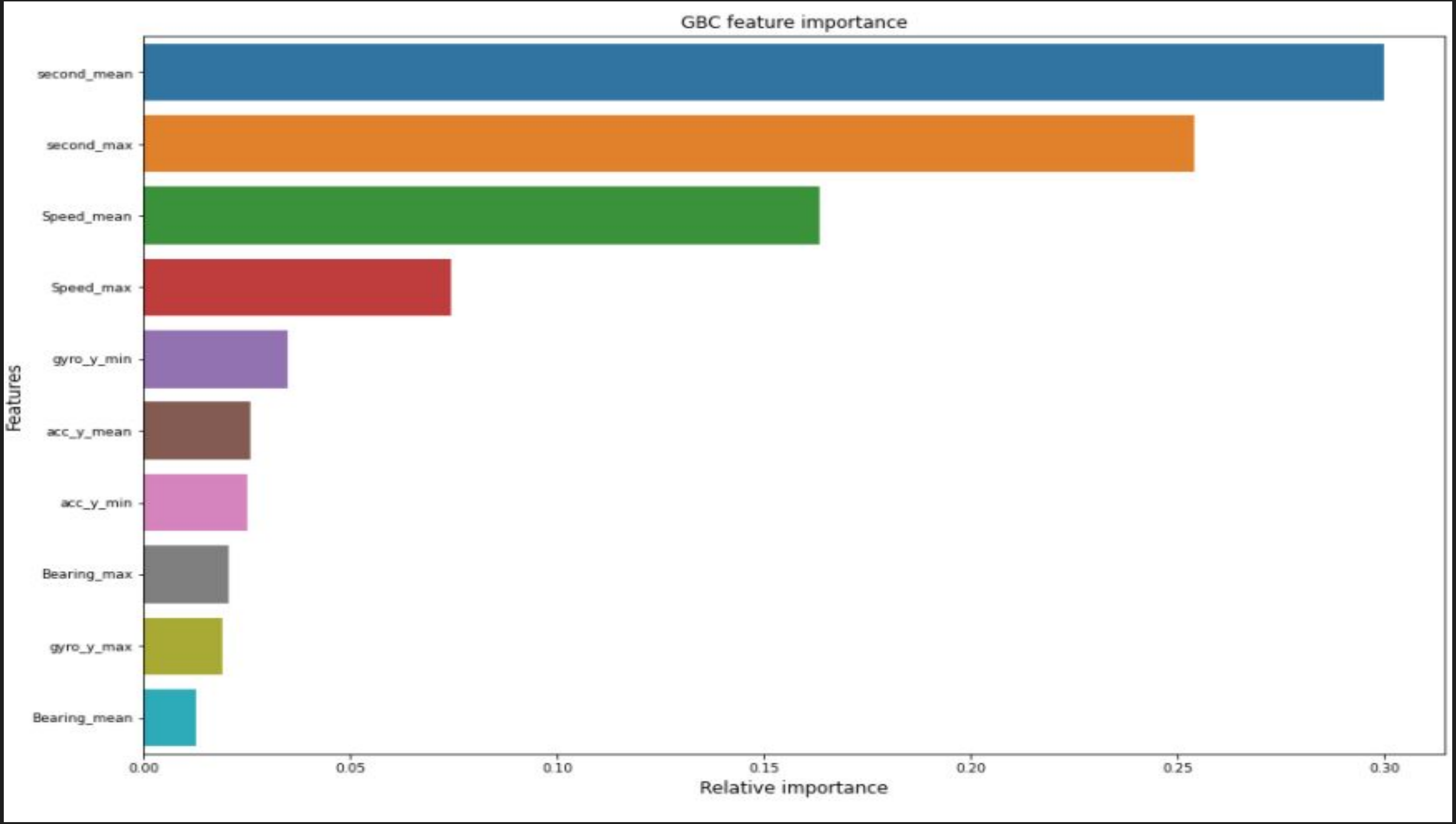
For the other features, after looking at more different data points, it seems as though there are no sequential trends that can differente between a non-dangerous and dangerous trip. This is mostly due to the fact that our team does not really have a good way to compare these sequential data at once with other sequential data of different bookingIDs.

As such other than from the problem statement 1, we cannnot really say if there is much of a difference between dangerous and non-dangerous trips, even if comparing the change in behaviour over time. As such you will see us attempt to us deep learning later on to try and classify the trip, but classical ML models outperform the Deep Learning Model.

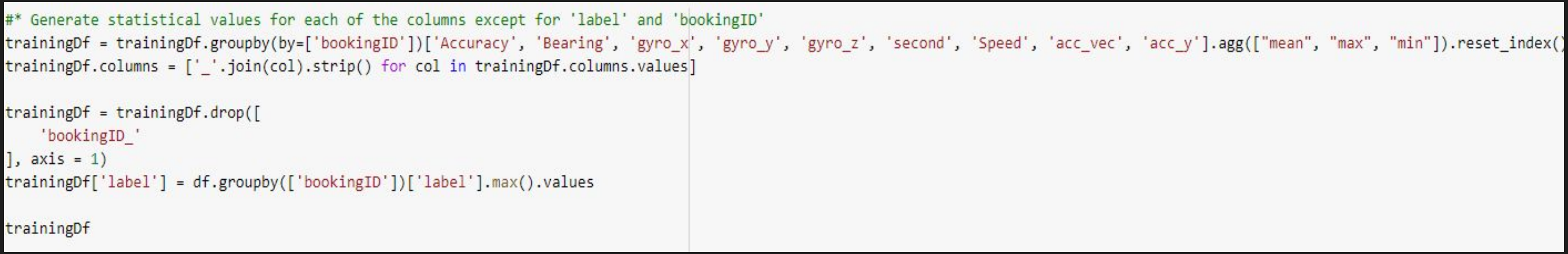
Problem Statement 3

Given the telematics data for new trips, derive a model to detect if the trip is a dangerous trip.

Classical Classification Model - Feature Importance

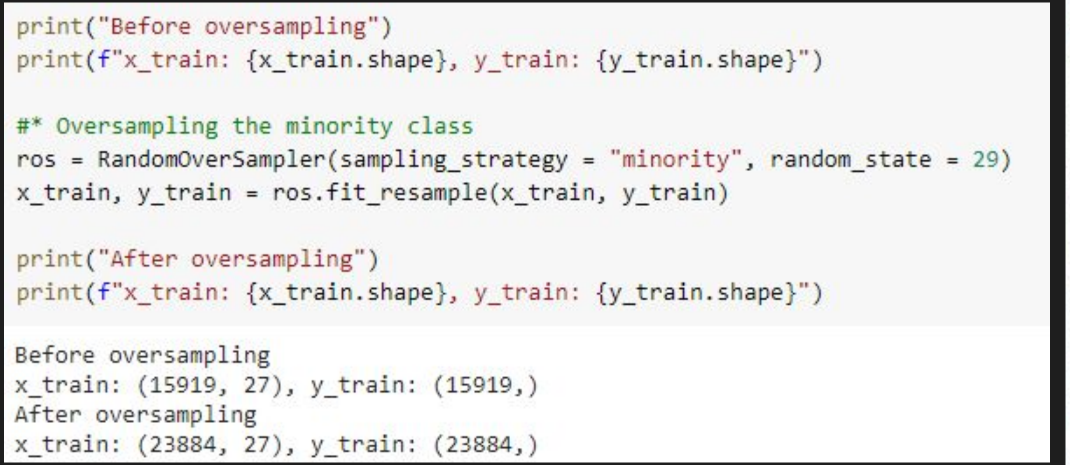


Classical Classification Model - Feature Engineering, Data Preparation



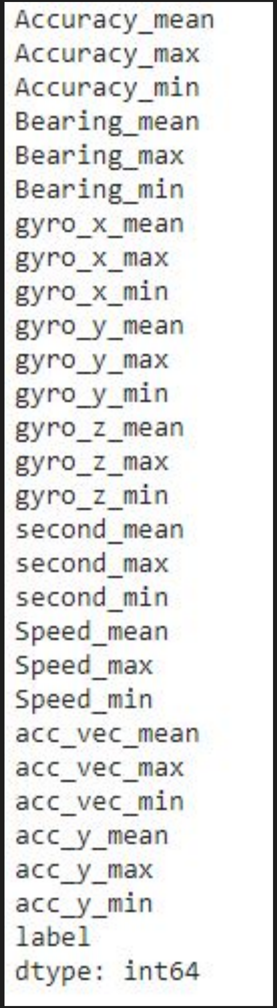
```
#* Generate statistical values for each of the columns except for 'label' and 'bookingID'
trainingDf = trainingDf.groupby(by=['bookingID'])['Accuracy', 'Bearing', 'gyro_x', 'gyro_y', 'gyro_z', 'second', 'Speed', 'acc_vec', 'acc_y'].agg(["mean", "max", "min"]).reset_index(
trainingDf.columns = ['_'.join(col).strip() for col in trainingDf.columns.values]

trainingDf = trainingDf.drop([
    'bookingID_'
], axis = 1)
trainingDf['label'] = df.groupby(['bookingID'])['label'].max().values

trainingDf
```

- Aggregated values for each column into 'mean', 'max' and 'min'
- Due to an overbalance in the dataset for the number of data for each label, we oversampled the label with the smaller number of data rows.



```
print("Before oversampling")
print(f"x_train: {x_train.shape}, y_train: {y_train.shape}")

#* Oversampling the minority class
ros = RandomOverSampler(sampling_strategy = "minority", random_state = 29)
x_train, y_train = ros.fit_resample(x_train, y_train)

print("After oversampling")
print(f"x_train: {x_train.shape}, y_train: {y_train.shape}")
```

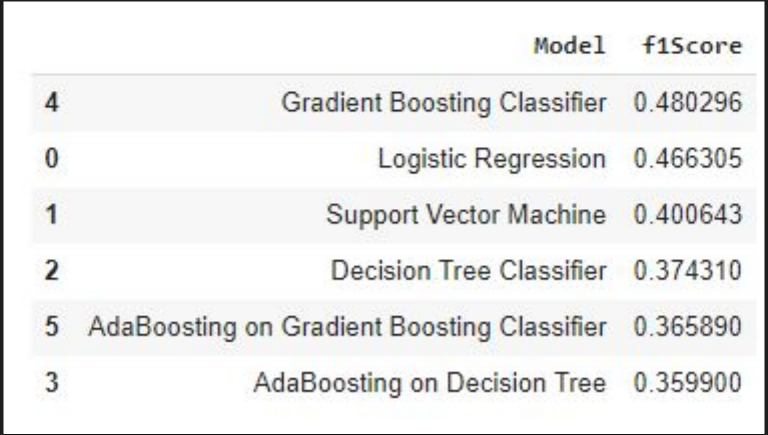
```
Before oversampling
x_train: (15919, 27), y_train: (15919,)
After oversampling
x_train: (23884, 27), y_train: (23884,)
```



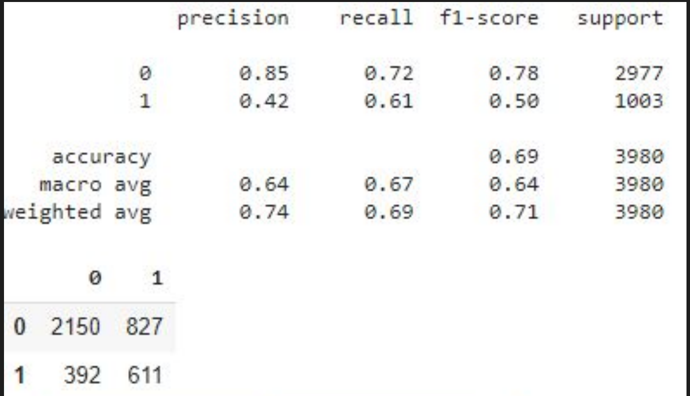
```
Accuracy_mean
Accuracy_max
Accuracy_min
Bearing_mean
Bearing_max
Bearing_min
gyro_x_mean
gyro_x_max
gyro_x_min
gyro_y_mean
gyro_y_max
gyro_y_min
gyro_z_mean
gyro_z_max
gyro_z_min
second_mean
second_max
second_min
Speed_mean
Speed_max
Speed_min
acc_vec_mean
acc_vec_max
acc_vec_min
acc_y_mean
acc_y_max
acc_y_min
label
dtype: int64
```

Classical Classification Model - Model training (Model selection)

- Used a variety of Classification models like Gradient Boosting Classifier, Support Vector Machine and Logistic Regression
- Decided on using Gradient Boosting Classifier in the end as it resulted in the best f1 score out of all the models

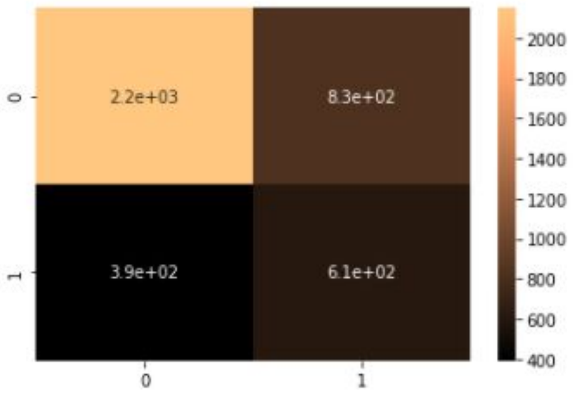


	Model	f1Score
4	Gradient Boosting Classifier	0.480296
0	Logistic Regression	0.466305
1	Support Vector Machine	0.400643
2	Decision Tree Classifier	0.374310
5	AdaBoosting on Gradient Boosting Classifier	0.365890
3	AdaBoosting on Decision Tree	0.359900



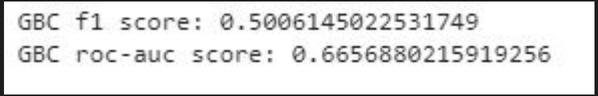
```
              precision    recall  f1-score   support

           0       0.85      0.72      0.78      2977
           1       0.42      0.61      0.50      1003

    accuracy                           0.69      3980
   macro avg       0.64      0.67      0.64      3980
weighted avg       0.74      0.69      0.71      3980
```

	0	1
0	2150	827
1	392	611

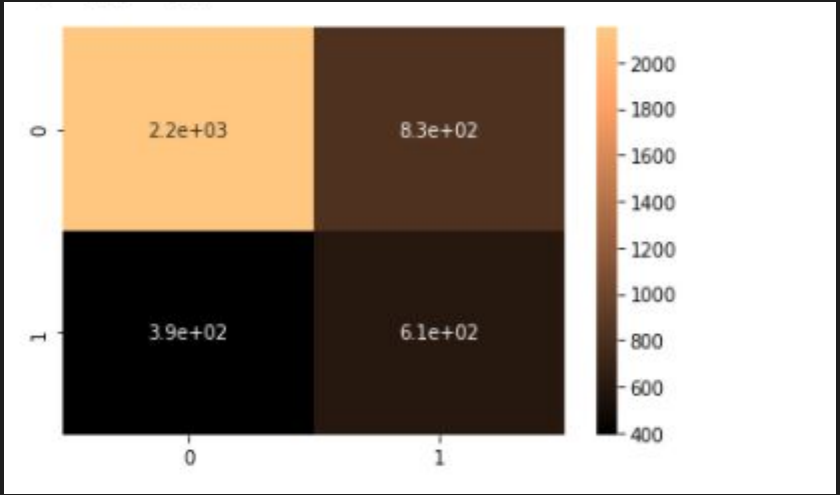


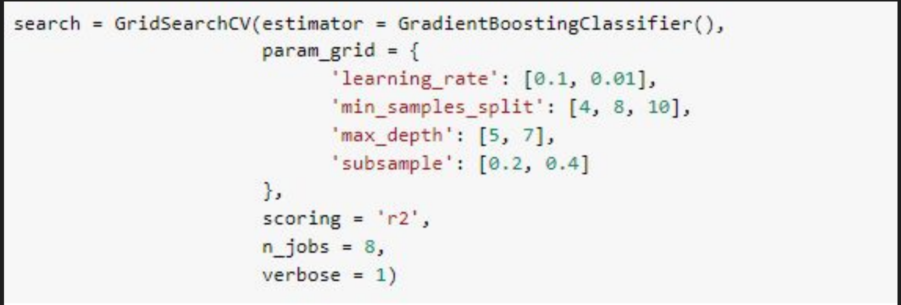
Classical Classification Model - Model training (Hyperparameter tuning)

- Performed GridSearchCV on the GBC model
- Performed some manual tuning based on best params obtained from GridSearchCV

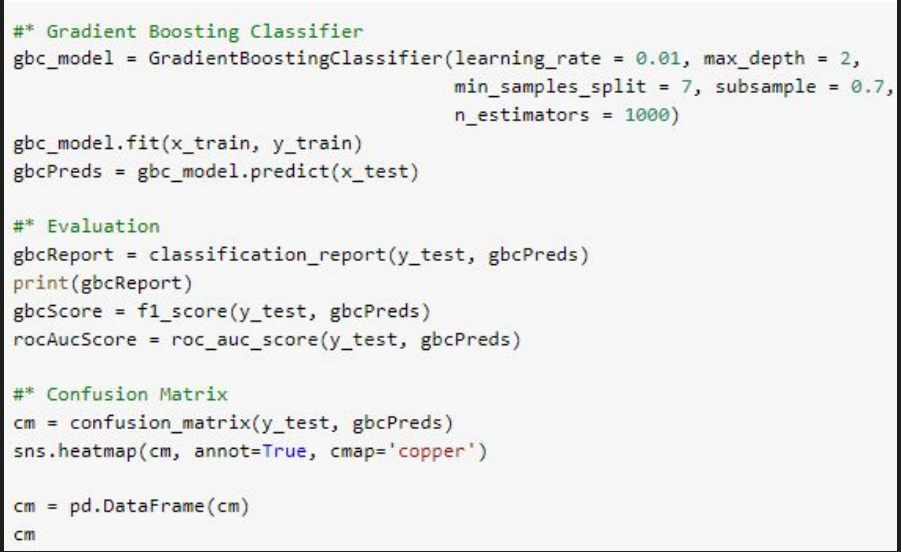


```
GBC f1 score: 0.5006145022531749
GBC roc-auc score: 0.6656880215919256
```



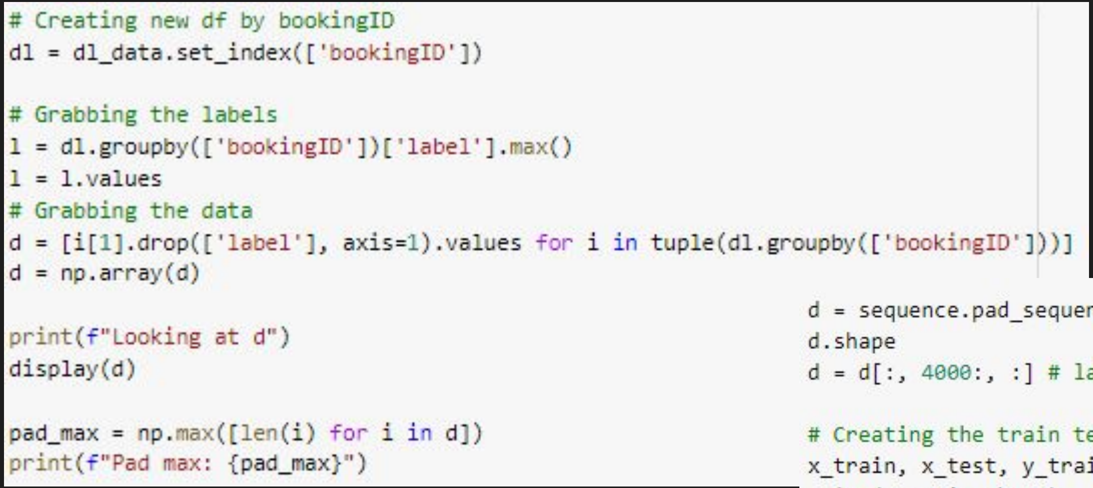


```
search = GridSearchCV(estimator = GradientBoostingClassifier(),
                      param_grid = {
                          'learning_rate': [0.1, 0.01],
                          'min_samples_split': [4, 8, 10],
                          'max_depth': [5, 7],
                          'subsample': [0.2, 0.4]
                      },
                      scoring = 'r2',
                      n_jobs = 8,
                      verbose = 1)
```

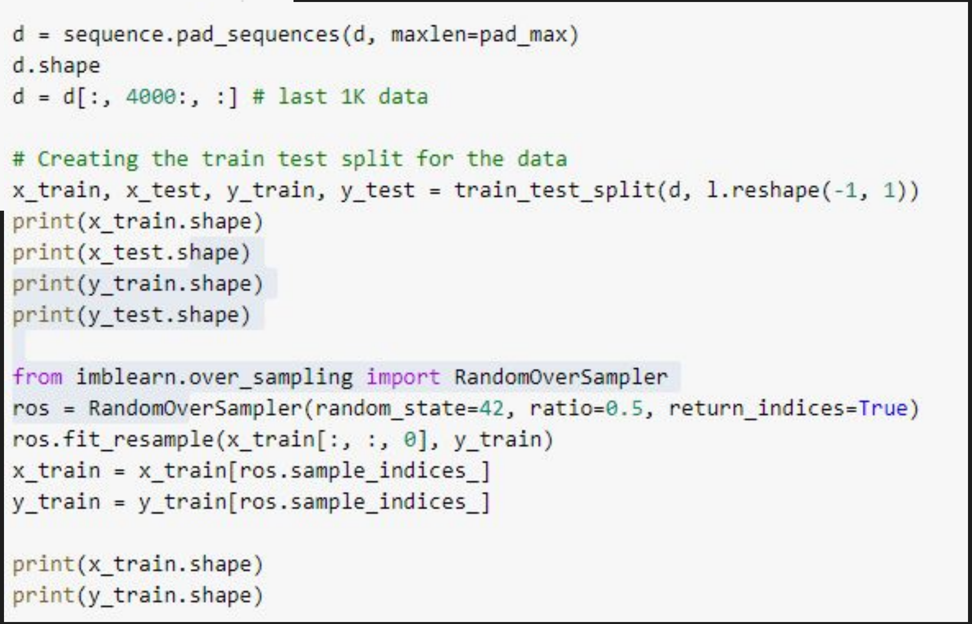


```
#* Gradient Boosting Classifier
gbc_model = GradientBoostingClassifier(learning_rate = 0.01, max_depth = 2,
                                       min_samples_split = 7, subsample = 0.7,
                                       n_estimators = 1000)
gbc_model.fit(x_train, y_train)
gbcPreds = gbc_model.predict(x_test)

#* Evaluation
gbcReport = classification_report(y_test, gbcPreds)
print(gbcReport)
gbcScore = f1_score(y_test, gbcPreds)
rocAucScore = roc_auc_score(y_test, gbcPreds)

#* Confusion Matrix
cm = confusion_matrix(y_test, gbcPreds)
sns.heatmap(cm, annot=True, cmap='copper')

cm = pd.DataFrame(cm)
cm
```

Data Preparation for Deep Learning

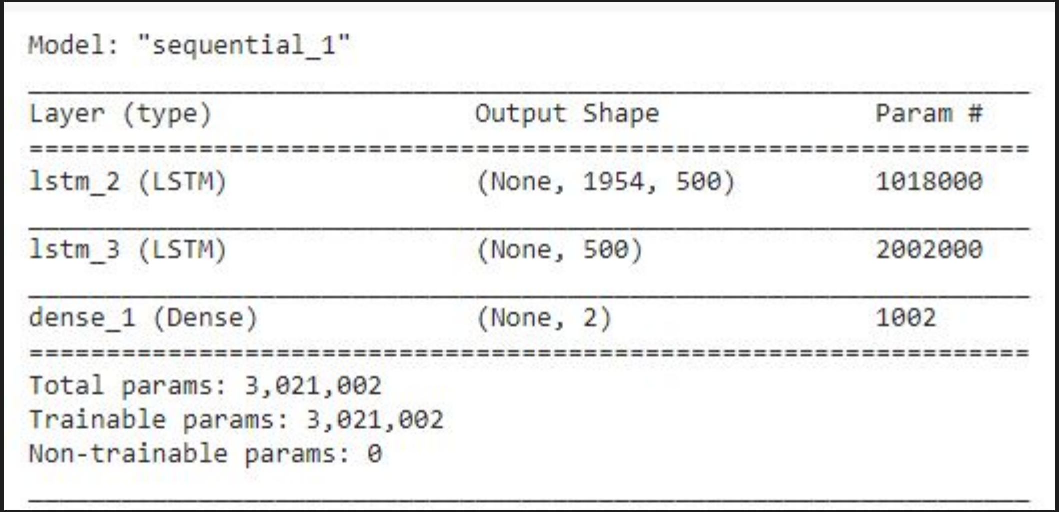


```python
# Creating new df by bookingID
dl = dl_data.set_index(['bookingID'])

# Grabbing the labels
l = dl.groupby(['bookingID'])['label'].max()
l = l.values
# Grabbing the data
d = [i[1].drop(['label'], axis=1).values for i in tuple(dl.groupby(['bookingID']))]
d = np.array(d)

print(f"Looking at d")
display(d)

pad_max = np.max([len(i) for i in d])
print(f"Pad max: {pad_max}")
```

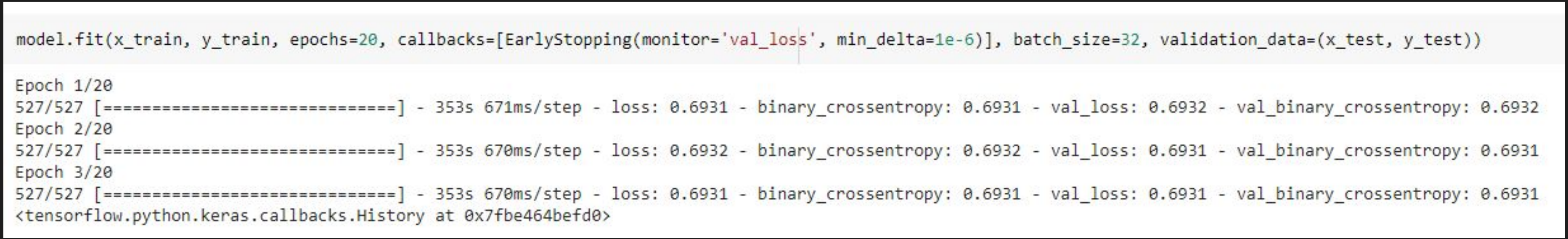


```python
d = sequence.pad_sequences(d, maxlen=pad_max)
d.shape
d = d[:, 4000:, :] # last 1K data

# Creating the train test split for the data
x_train, x_test, y_train, y_test = train_test_split(d, l.reshape(-1, 1))
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42, ratio=0.5, return_indices=True)
ros.fit_resample(x_train[:, :, 0], y_train)
x_train = x_train[ros.sample_indices_]
y_train = y_train[ros.sample_indices_]

print(x_train.shape)
print(y_train.shape)
```

LSTM Model



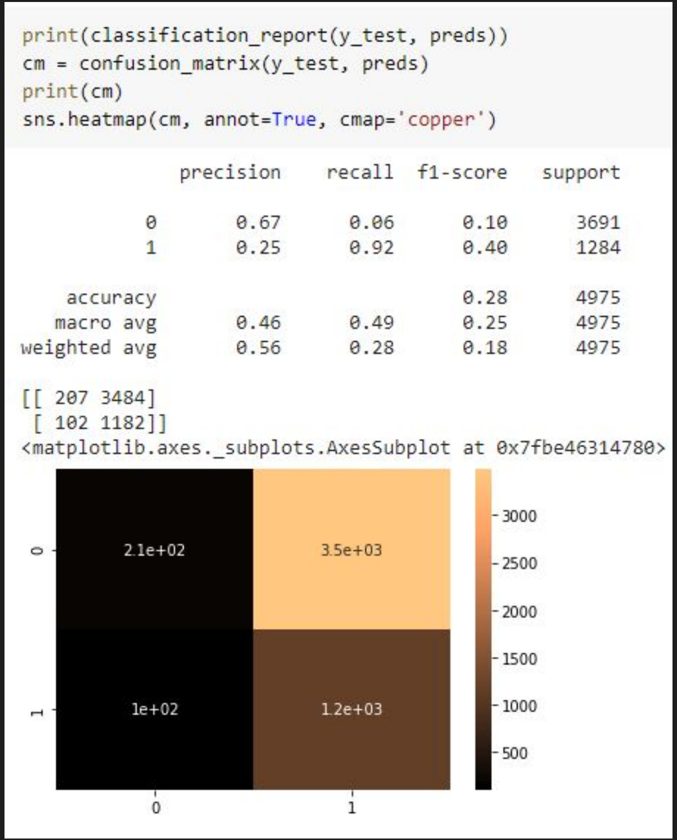
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 1954, 500)         1018000
_________________________________________________________________
lstm_3 (LSTM)                (None, 500)               2002000
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1002
=================================================================
Total params: 3,021,002
Trainable params: 3,021,002
Non-trainable params: 0
_________________________________________________________________
```



```
model.fit(x_train, y_train, epochs=20, callbacks=[EarlyStopping(monitor='val_loss', min_delta=1e-6)], batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/20
527/527 [==============================] - 353s 671ms/step - loss: 0.6931 - binary_crossentropy: 0.6931 - val_loss: 0.6932 - val_binary_crossentropy: 0.6932
Epoch 2/20
527/527 [==============================] - 353s 670ms/step - loss: 0.6932 - binary_crossentropy: 0.6932 - val_loss: 0.6931 - val_binary_crossentropy: 0.6931
Epoch 3/20
527/527 [==============================] - 353s 670ms/step - loss: 0.6931 - binary_crossentropy: 0.6931 - val_loss: 0.6931 - val_binary_crossentropy: 0.6931
<tensorflow.python.keras.callbacks.History at 0x7fbe464befd0>
```

Results for Deep Learning model

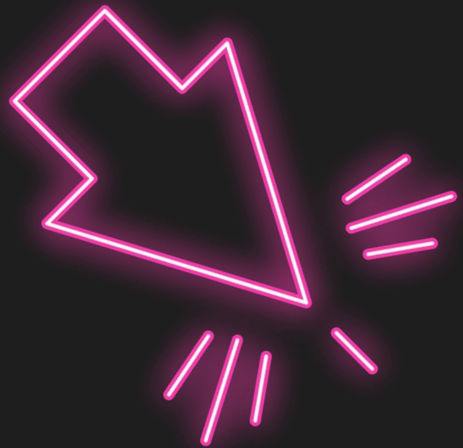


```
print(classification_report(y_test, preds))
cm = confusion_matrix(y_test, preds)
print(cm)
sns.heatmap(cm, annot=True, cmap='copper')
```

```
              precision    recall  f1-score   support

           0       0.67      0.06      0.10      3691
           1       0.25      0.92      0.40      1284

    accuracy                           0.28      4975
   macro avg       0.46      0.49      0.25      4975
weighted avg       0.56      0.28      0.18      4975

[[ 207 3484]
 [ 102 1182]]
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe46314780>
```

Overall conclusion

- In truth, we were pressed for time for this challenge
  - It was much harder to both understand and tackle
  - However, with more time, we are confident we could try out more techniques and strategies
- There are much more details and explanations in the main notebook in the github repo.
- If you want to find out what specific code we used, you can check out the development notebooks in their respective folders as well



Thanks for listening!

Our Githubs:

@chuanhao01

@David-The-Programmer

@sherissetjw

The github repo: https://github.com/chuanhao01/MicrosoftxGrab_FRSP2020_Safety_Team_2