**check out the github repository**
https://github.com/teamoptimusai/datastorm2-initial

Team Name - “OptimizePrime”

## Data Analysis

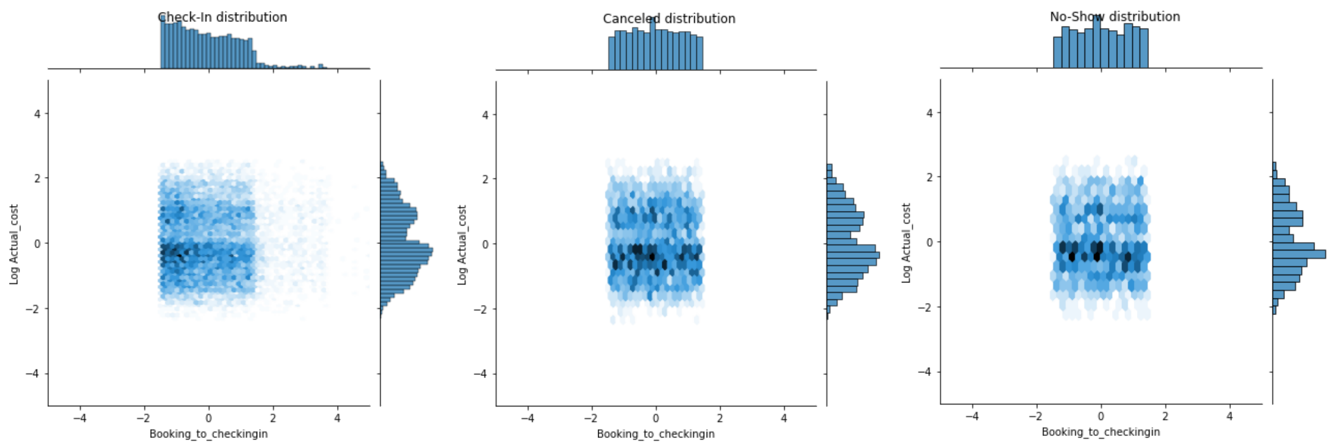
We have 3 files containing data from Hotel A for training, validation, and testing.  Instead of using the `'Hotel-A-train.csv'` as training data and `'Hotel-A-validation.csv'` as validation data, we combined both of them to create a single dataset (data_train). Then the feature columns were categorized into one-hot encoded, binary, or data type columns.

```
one_hot_encoded_lst = ['Ethnicity', 'Educational_Level',
      'Income', 'Country_region', 'Hotel_Type',
      'Meal_Type', 'Deposit_type', 'Booking_channel']

dates = ['Expected_checkin', 'Expected_checkout', 'Booking_date']

binary_cols = ['Gender', 'Visted_Previously', 'Previous_Cancellations',
      'Required_Car_Parking', 'Use_Promotion']
```

A quick check of Reservation Status showed that the classes are heavily imbalanced.

```
Dataset:
    Total: 30248
    Check-In: 22850 (75.54% of total)
    Canceled: 4875  (16.12% of total)
     No-Show: 2523  (8.34% of total)
```

Then we checked the distribution of features for each class. For the below graphs, we used the `"Log Actual Cost"` and `"Booking_to_checkingin"` features after normalizing.



Here you can see that Canceled and no-show samples contain a much higher rate of extreme values and that Date features are not normalized with the current scaler because there are few values.
According to our experience in dealing with imbalanced structured data, we came up with 2 solutions to deal with this issue.

1. Weighted Models
2. Oversampling

When trying out the weighted models, we used “balanced”, “balanced_subsample” and custom weights we have calculated based on the counts of each class. Following are the weights for each class we have calculated.

```
Weight for class 1: 0.66
Weight for class 2: 3.11
Weight for class 3: 6.02
```

After resampling,

```
resampled_features.shape
(43860, 44)
```

But in the end, only weighted model implementation is able to provide us better results.

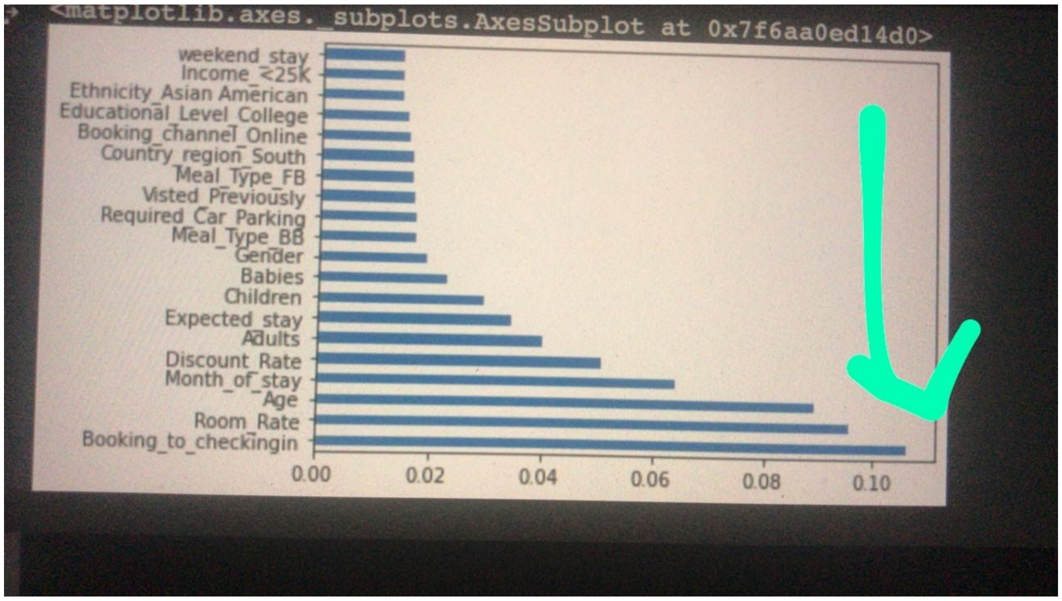
## Feature Engineering and Data Cleaning

When we looked at the data set we noticed 3 DateTime objects. Since these would not mean much in their original form we created 4 new columns

1. **Expected_Stay** - The Expecting time of staying. This was obtained by taking the difference between the 2 DateTime objects **Expected_checkout** and **Expected_checkcin**
2. **Booking_to_checkingin** - Since a user can easily forget the reservation if he/she had booked a way earlier than the actual stay we decided to create a column containing the number of dates between the booking date and the expected_checkin. This was obtained by taking the difference between the 2 DateTime objects **Expected_checkout** and **Booking_date**
3. **Month_of_Stay** - Holiday seasons affect the cancellation of reservations. Since we do not have enough data to consider the region where the hotel is located we could not deep dive into that conclusion. Therefore we used the **month of stay** of the reservation for a possible thing.

4. **Actual_Cost** - The cost would greatly affect the visitors with a tight budget so we created a new data field with the actual cost by multiplying the rate per day with the actual staying dates.

```
dates = ['Expected_checkin', 'Expected_checkout', 'Booking_date']
data_train['Expected_stay'] = (data_train[dates[1]] -
data_train[dates[0]]).dt.days
data_train['Booking_to_checkingin'] = (data_train[dates[0]] -
data_train[dates[2]]).dt.days
data_train['Month_of_stay'] = data_train[dates[0]].dt.month
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6aa0ed14d0>
```

## Trying out Different Types of Model

### Weighted Random Forest

After finding out that the random forest classifier performed well without weights then we tried out the weighted random forest with the following weights

```
Check-In: 0.66 | Canceled: 3.11 | No-Show: 6.02
```

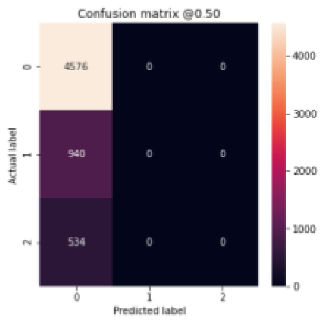
Using this we were able to get a much better macro average F1 Score (0.29 to 0.32)

```
RandomForest Classification_Custom_Weights : 0.7518595041322315
[[3598   54    8]
 [ 740   35    3]
 [ 383   13    6]]
```

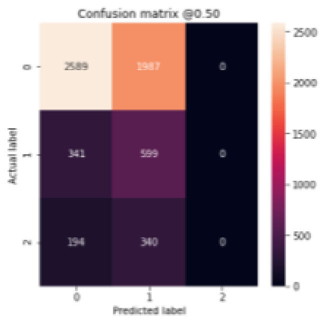
```
              precision    recall  f1-score   support

           0       0.76      0.98      0.86      3660
           1       0.34      0.04      0.08       778
           2       0.35      0.01      0.03       402

    accuracy                           0.75      4840
   macro avg       0.49      0.35      0.32      4840
weighted avg       0.66      0.75      0.66      4840
```

## Neural Networks

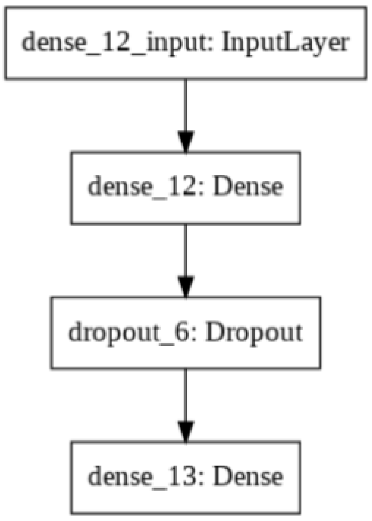
We implemented a simple small neural network and trained the model in 3 ways namely without weights, with weights, oversampled. In every situation when we checked the confusion matrix we realized that the Neural Network approach would be a better approach because all of them seem to underfit with the training dataset and completely ignores some classes.



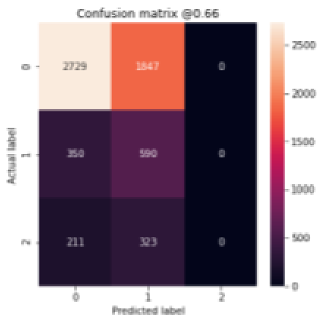
(a) Initial Model



(b) Weighted Model



Simple Neural Network



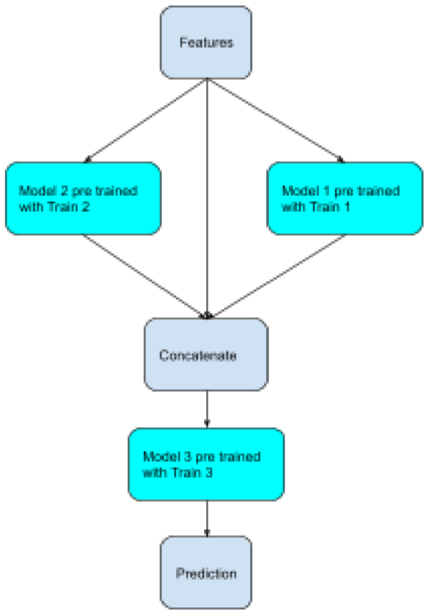
(c) Model Trained with Oversampled data

## Multi-Model Experiment

In Deep Learning, we sometimes use a method called Multi-Model Ensembling where we combine pertained huge neural networks parallelly and we concatenate the feature vectors coming from those Networks and then use a simple classifier (could be an ANN or a Tree-based classifier) on the concatenated feature vector for classification. In researches such as Image Classification, this method has been proven to work very well.

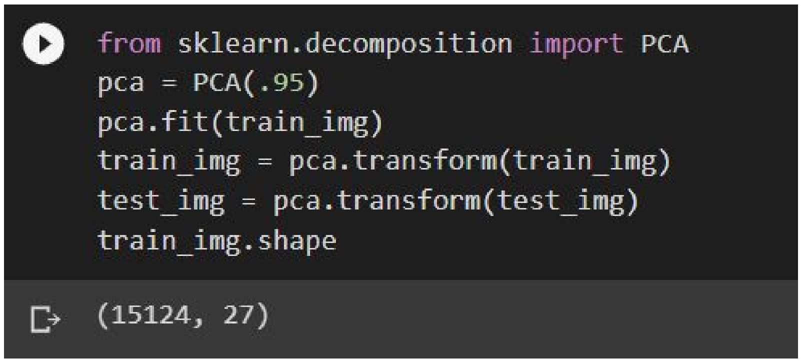
So we used the same principle and made a multi-model classifier but by replacing Tree-based classifiers instead of Deep Neural Networks. As tree-based classifiers cannot produce feature vectors as outputs we have to just stick with their current outputs.

For this experiment, we divided the dataset into 4 parts namely (train1, train2, train3, Val) and them as follows.


Features
Model 2 pre trained with Train 2
Model 1 pre trained with Train 1
Concatenate
Model 3 pre trained with Train 3
Prediction

## PCA Dimensionality reduction test

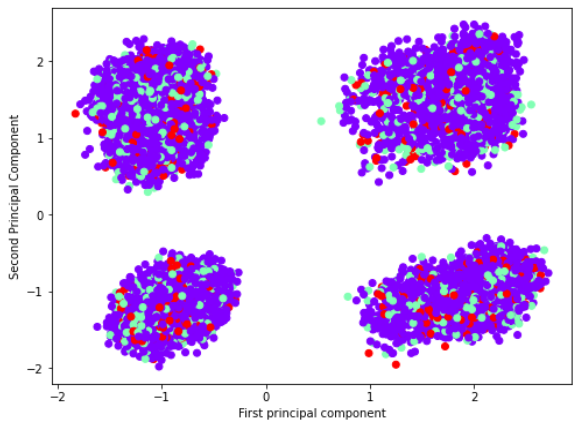
Since the original data consisted of 44 columns and to run the training model smoother we implemented principal component analysis to reduce the number of dimensions. Any columns with more than 95% of covariance will be used to calculate a new dimension.

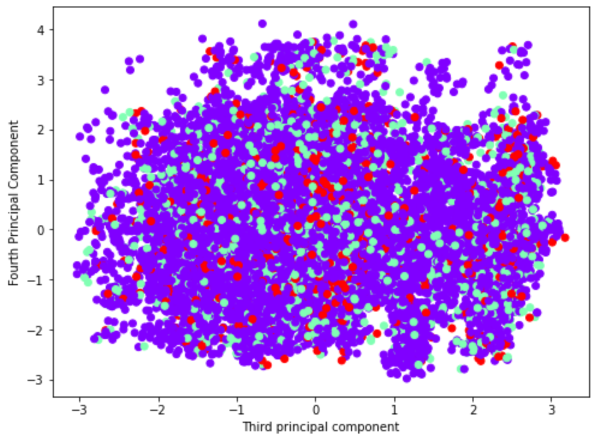


```
from sklearn.decomposition import PCA
pca = PCA(.95)
pca.fit(train_img)
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
train_img.shape
```

```
(15124, 27)
```

As you can see here the dimensions have been reduced up to 27. How the data influenced under different principal components can be seen below.

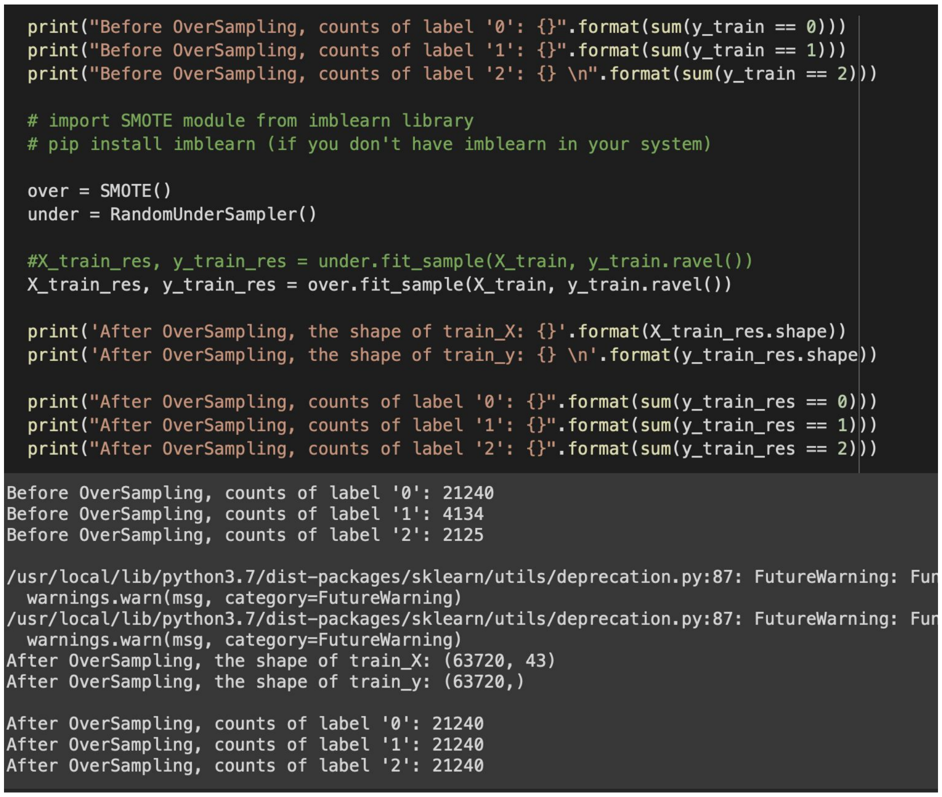




## SMOTE algorithm for Imbalanced

Since the original training table was highly imbalanced, the necessity of dataset balancing was reasonable.

1. First undersampling the dataset and then oversampling the whole dataset
2. Oversampling every minority class to the majority class



```
print("Before OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '2': {} \n".format(sum(y_train == 2)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)

over = SMOTE()
under = RandomUnderSampler()

#X_train_res, y_train_res = under.fit_sample(X_train, y_train.ravel())
X_train_res, y_train_res = over.fit_sample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '2': {}".format(sum(y_train_res == 2)))
```

```
Before OverSampling, counts of label '0': 21240
Before OverSampling, counts of label '1': 4134
Before OverSampling, counts of label '2': 2125

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fur
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fur
  warnings.warn(msg, category=FutureWarning)
After OverSampling, the shape of train_X: (63720, 43)
After OverSampling, the shape of train_y: (63720,)

After OverSampling, counts of label '0': 21240
After OverSampling, counts of label '1': 21240
After OverSampling, counts of label '2': 21240
```

But undersampling + oversampling models were lower than the just oversampling method but both methods were not an improvement than the stated method.

## Business Implementation

As you can see when we consider the key features that drive the model, we can see that the Discount rate, Month of stay, Age, Room rate, and Time duration from Booking date to Checking Date affects most. Therefore when we consider customers who got a discount rate tend to act according to their reservation. People tend to forget about the reservation when they have a very long period between the booking and the checking date. So we have to make sure that they are alarmed frequently about their reservation.

**At the end of all of these experiments, our conclusion is that the training, validation and test datasets come from different distributions. Due to that, the performance in the validation set does not guarantee the same performance in the test set.**

**check out the github repository**
**https://github.com/teamoptimusai/datastorm2-initial/tree/master/Final_Implementation**