Car Accident Severity Prediction for Seattle Road Accidents

1. Introduction

Traffic accidents are a significant source of deaths, injuries, property damage, and a major concern for public health and traffic safety. Accidents are also a major cause of traffic congestion and delay. Effective management of accident is crucial to mitigating accident impacts and improving traffic safety and transportation system efficiency.

Accurate predictions of severity can provide crucial information for emergency responders to evaluate the severity level of accidents, estimate the potential impacts, and implement efficient accident management procedures.

By recognizing the key factors that influence accident severity, the solution may be of great utility to various Government Departments/Authorities like SWDOT and Police. The results of analysis and modeling can be used by these Departments to take appropriate measures; such as early warning system to drivers; to reduce accident impact and thereby improve traffic safety. It is also useful to the Insurers in terms of reduced claims and better underwriting as well as rate making.

2. Data Understanding

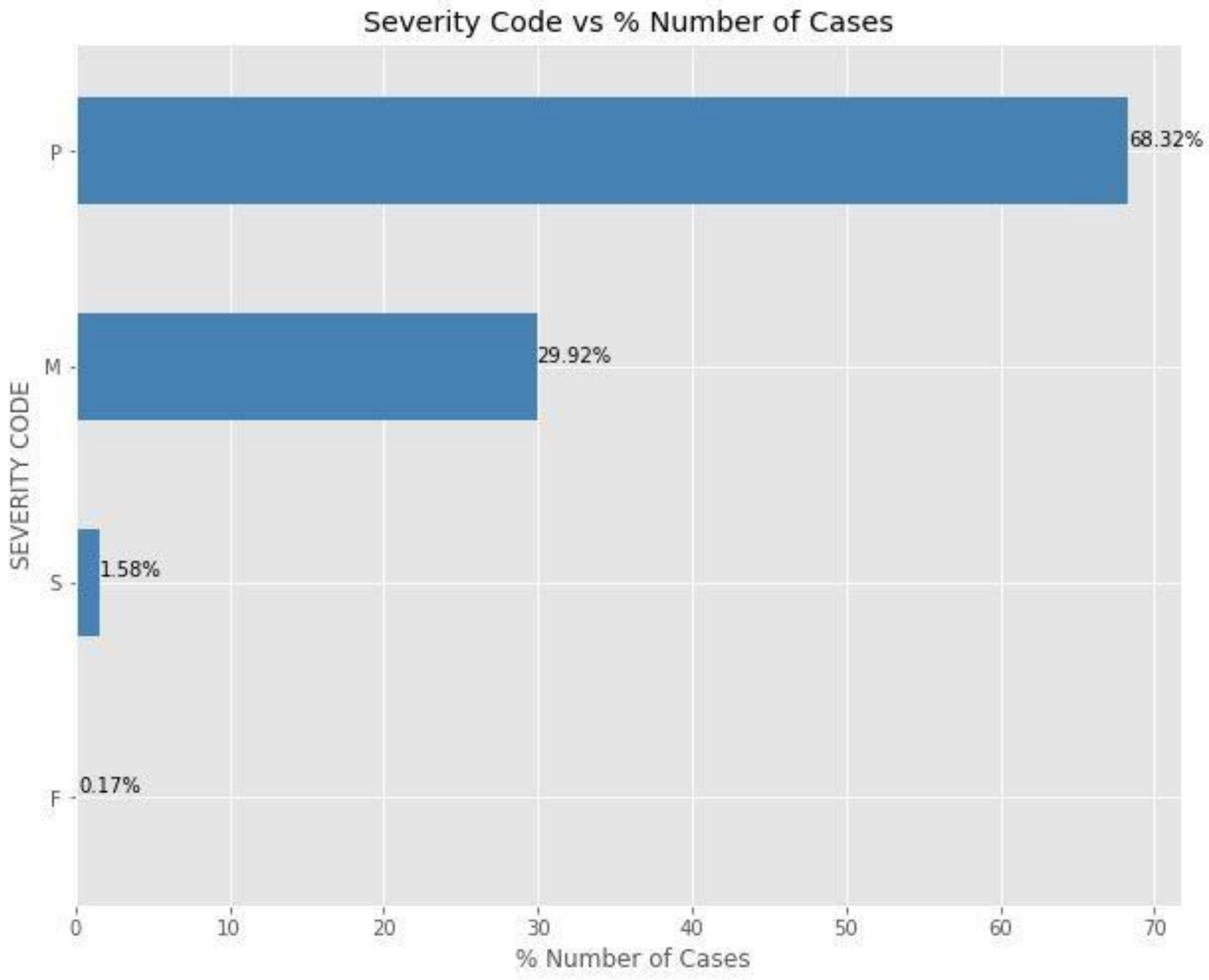
The **data** is obtained from City of Seattle Open Data Portal that is updated frequently. It contains all types of collisions from 2004 to Present. This data consists of 221,389 observations. Each with 40 different features, both numerical and categorical. A clear definition of each features is available in this **link.**

SEVERITYCODE feature is the target variable to predict the severity of car accidents. It is a categorical feature that is defined as 0, 1, 2, 2b, 3 to indicate the accident was of Unknown, Prop Damage only, Injury, Serious Injury or Fatality respectively.

3. Data Preparation

3.1 Data Clean Up

SEVERITYCODE "Unknown" is removed for a simple reason that we can't predict the unknown. Also to make it more descriptive, these codes are replaced with these labels **P**: Property Damage, **M**: Minor Injury, **S**: Serious Injury, **F**: Fatality.

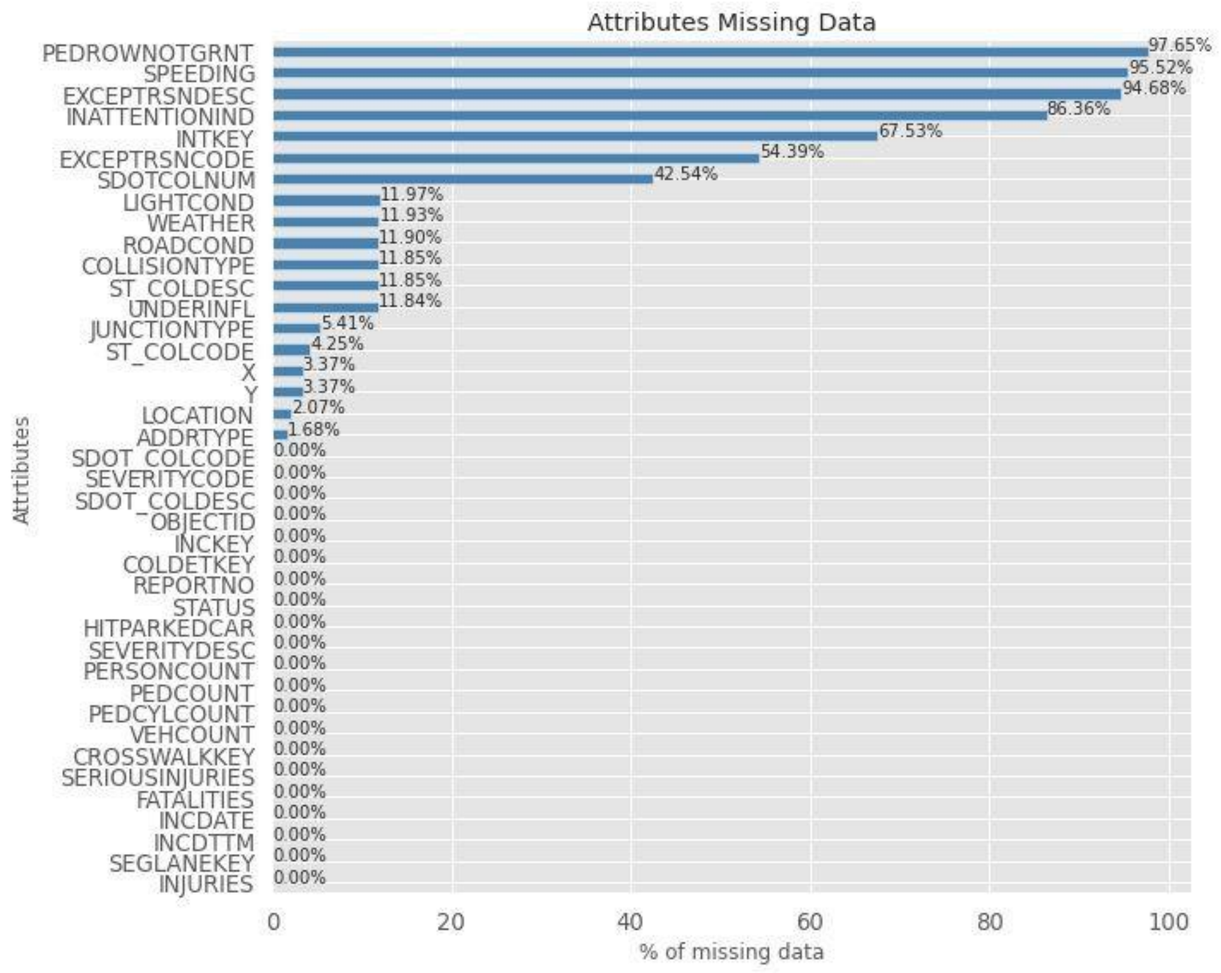


The plot above tells us that there is a extreme imbalance for the target variable.

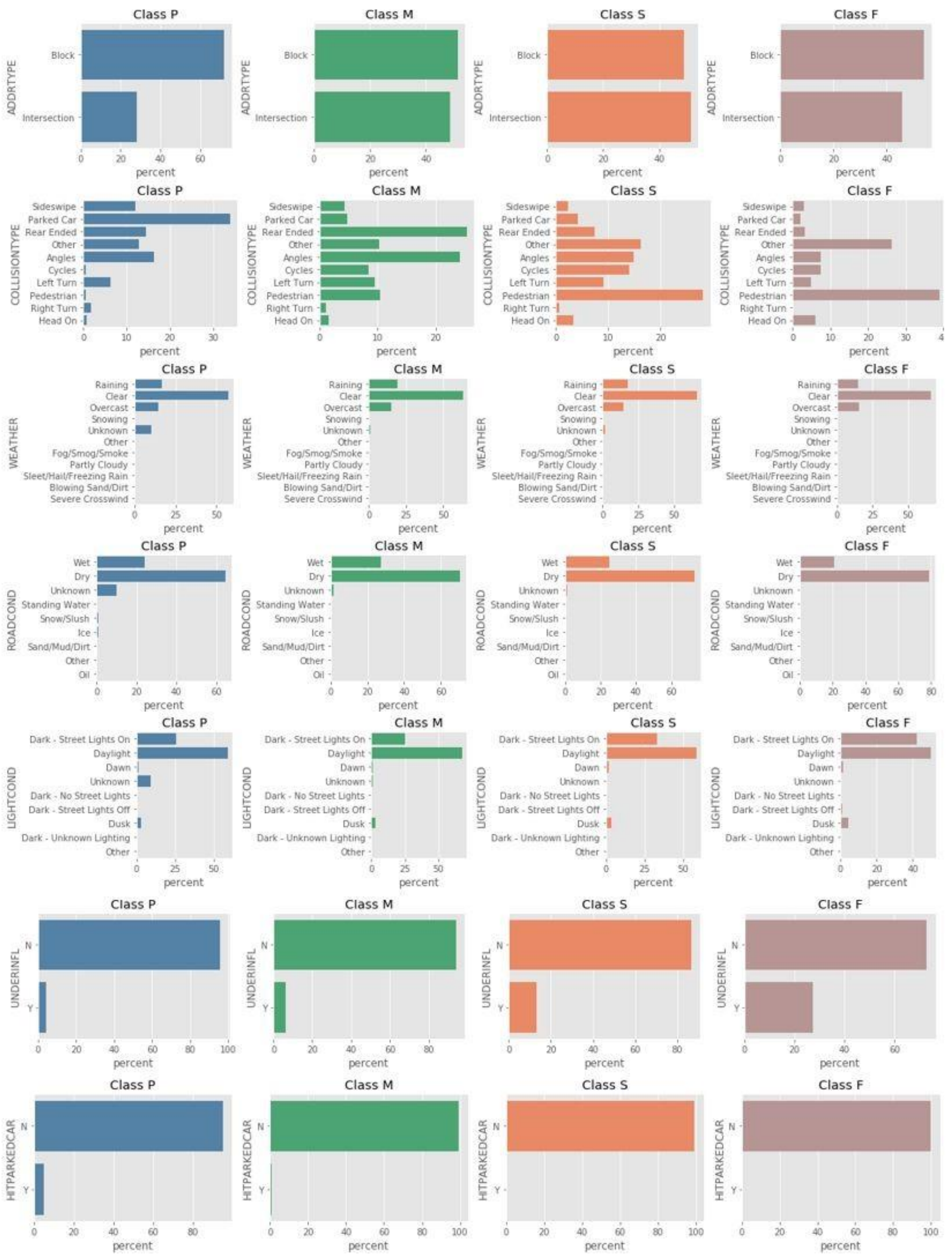
I remove features in dataset that may:

- have high percentage of missing data over 85%.
- irrelevant to accident severity such as ReportNo, Status.

- be derived from others features such as SEVERITYDESC, SEGLANEKEY. These derived features don't add value to dataset. It also create unnecessary correlation between features.
- be provided after the accidents has been reported and processed such as count of Fatality victims based on its severity. Post accidents features create strong bias into our model.

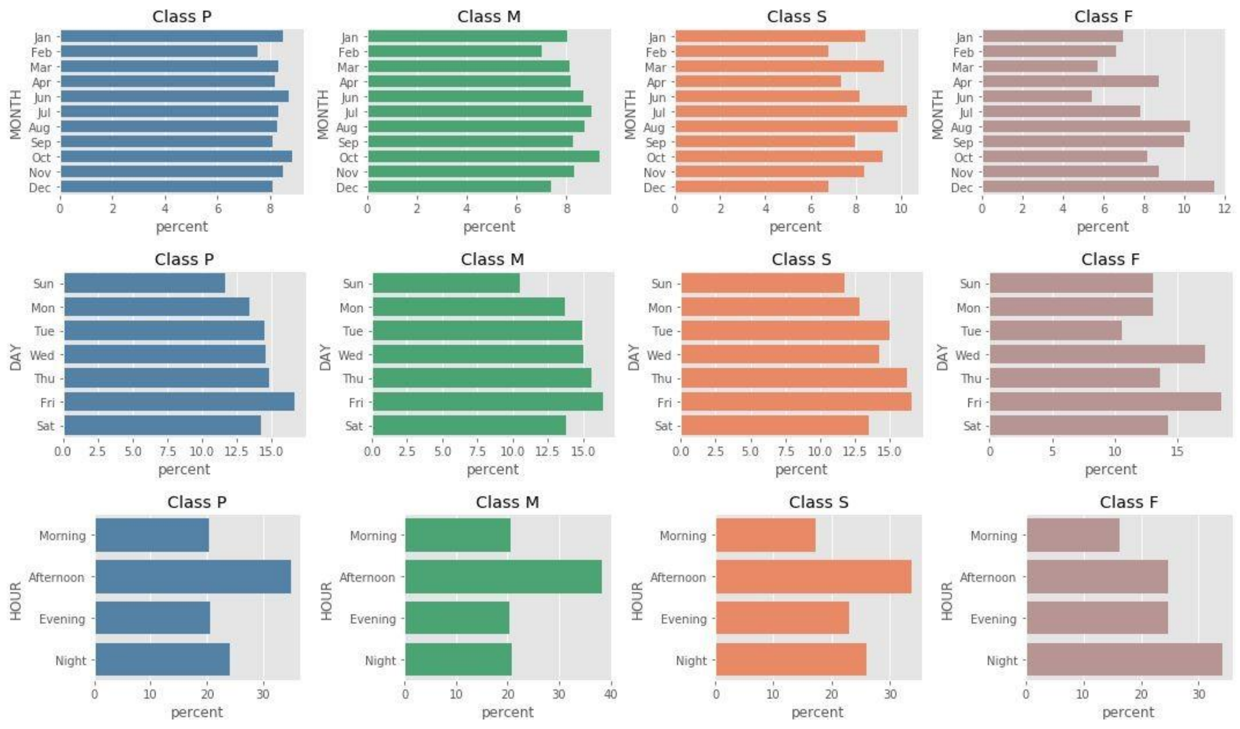


Each features are examined thoroughly for its consistency for every classes. Some features clearly stand out as a differentiator for a class but indistinguishable on other classes.

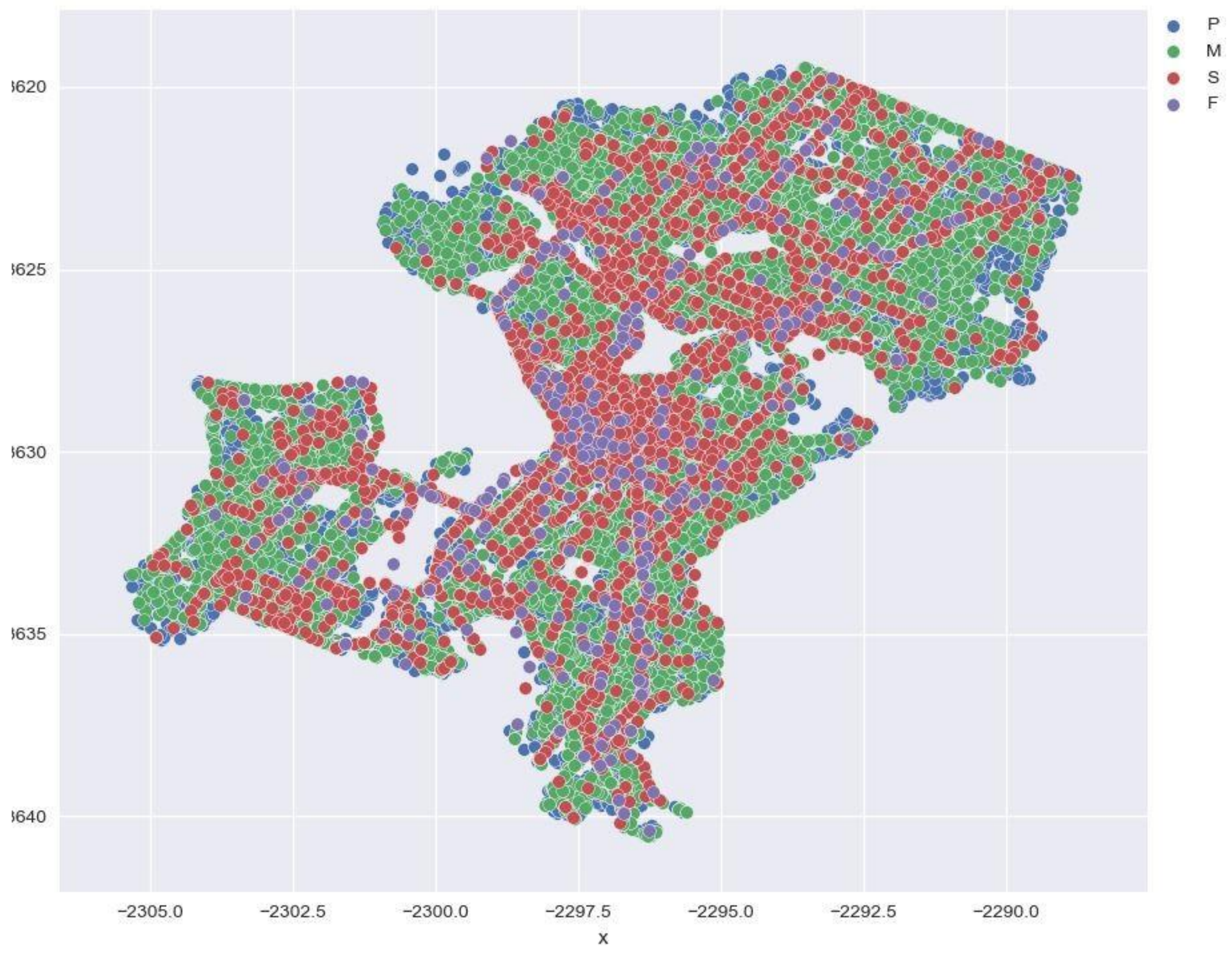

Class P
Class M
Class S
Class F
ADDRTYPE
Block
Intersection
COLLISIONTYPE
Sideswipe
Parked Car
Rear Ended
Other
Angles
Cycles
Left Turn
Pedestrian
Right Turn
Head On
WEATHER
Raining
Clear
Overcast
Snowing
Unknown
Other
Fog/Smog/Smoke
Partly Cloudy
Sleet/Hail/Freezing Rain
Blowing Sand/Dirt
Severe Crosswind
ROADCOND
Wet
Dry
Unknown
Standing Water
Snow/Slush
Ice
Sand/Mud/Dirt
Other
Oil
LIGHTCOND
Dark - Street Lights On
Daylight
Dawn
Unknown
Dark - No Street Lights
Dark - Street Lights Off
Dusk
Dark - Unknown Lighting
Other
UNDERINFL
N
Y
HITPARKEDCAR
N
Y
percent

3.2 Feature Engineering

The time and date of when the accident happens maybe contain some information. For example, Friday evening road traffic gets heavier increasing the probability of road accident. I extract MONTH, DAY, HOUR features from accident date and time. We can see that these three new features somewhat related to accident severity. For example, be in a traffic in Friday night December increases the chance of fatality accident.



Latitude and longitude feature are converted to Cartesian coordinate assuming the earth is sphere. From the scatter plot below we can see property damages accident is everywhere but fatality accidents are at certain location.



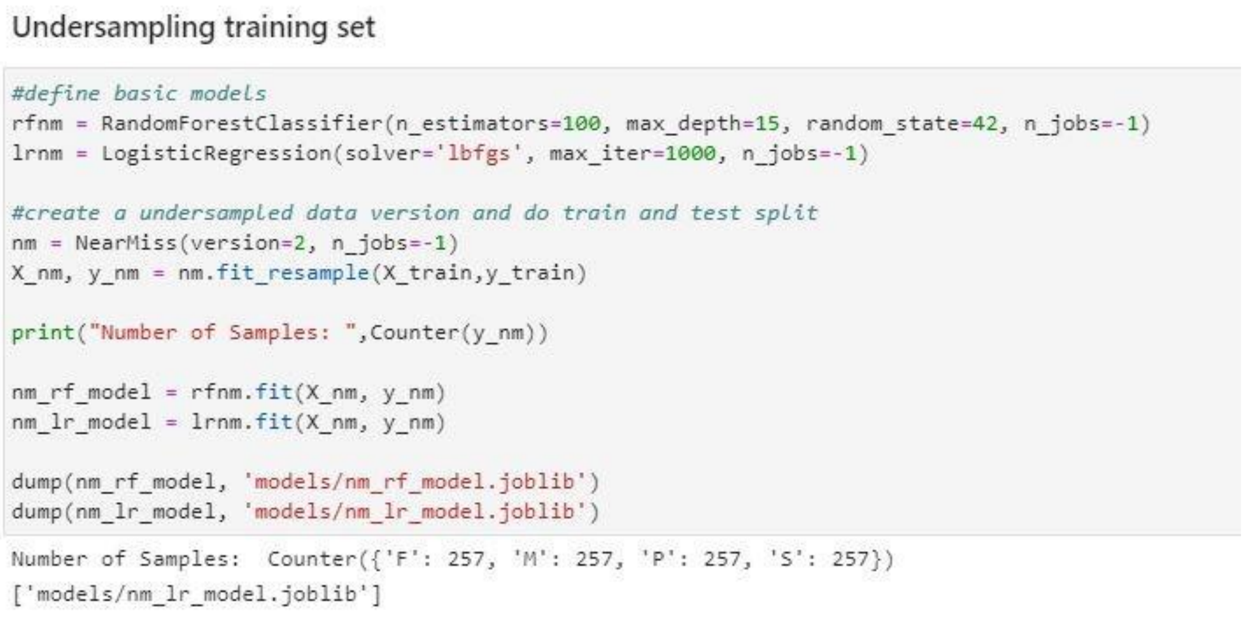
I decided to drop any samples that has nan values across the features. It slightly reduced number of sample in class P, the largest group.

After data clean up, feature engineering and hot encode categorical data, the clean dataset has 75 predictors with 189,434 samples.

4. Methodology

Logistic Regression and Random Forest models are used to solve the problem which is to predict the accident severity. I use the following techniques to handle the imbalance in the dataset:

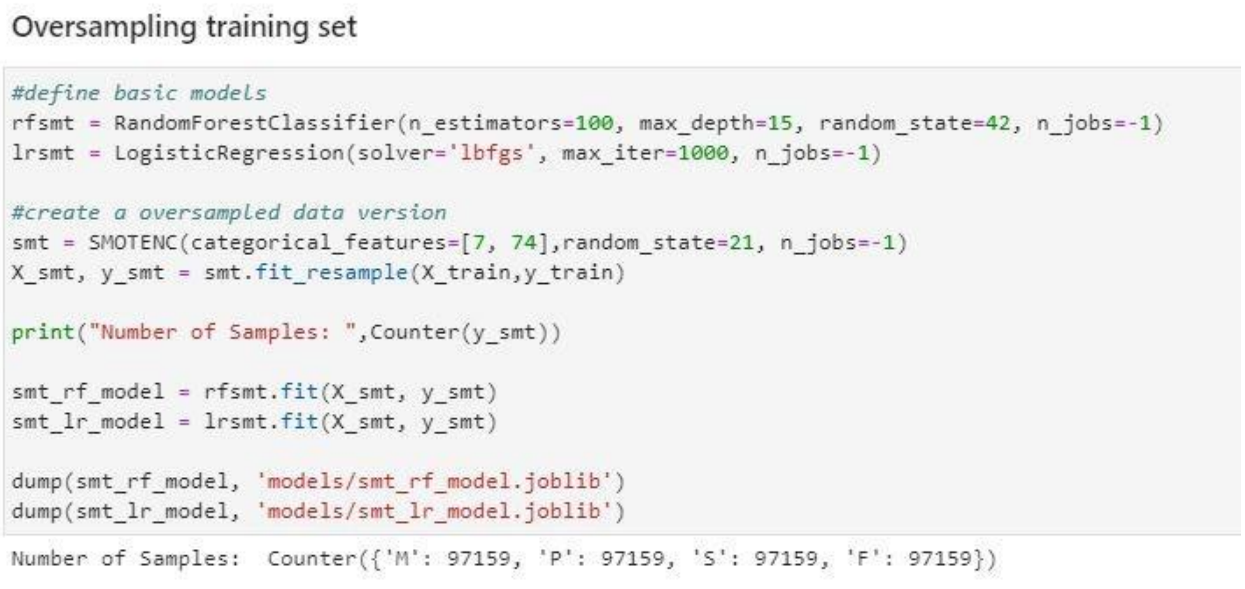
- Feed the models with **resampled train data after train_test_split** process. I use NearMiss algorithm for undersampling and SMOTENC algorithm for oversampling. Both are from imblearn-learn package



Undersampling training set

```
#define basic models
rfnm = RandomForestClassifier(n_estimators=100, max_depth=15, random_state=42, n_jobs=-1)
lrnm = LogisticRegression(solver='lbfgs', max_iter=1000, n_jobs=-1)

#create a undersampled data version and do train and test split
nm = NearMiss(version=2, n_jobs=-1)
X_nm, y_nm = nm.fit_resample(X_train,y_train)

print("Number of Samples: ",Counter(y_nm))

nm_rf_model = rfnm.fit(X_nm, y_nm)
nm_lr_model = lrnm.fit(X_nm, y_nm)

dump(nm_rf_model, 'models/nm_rf_model.joblib')
dump(nm_lr_model, 'models/nm_lr_model.joblib')
```

```
Number of Samples:  Counter({'F': 257, 'M': 257, 'P': 257, 'S': 257})
['models/nm_lr_model.joblib']
```

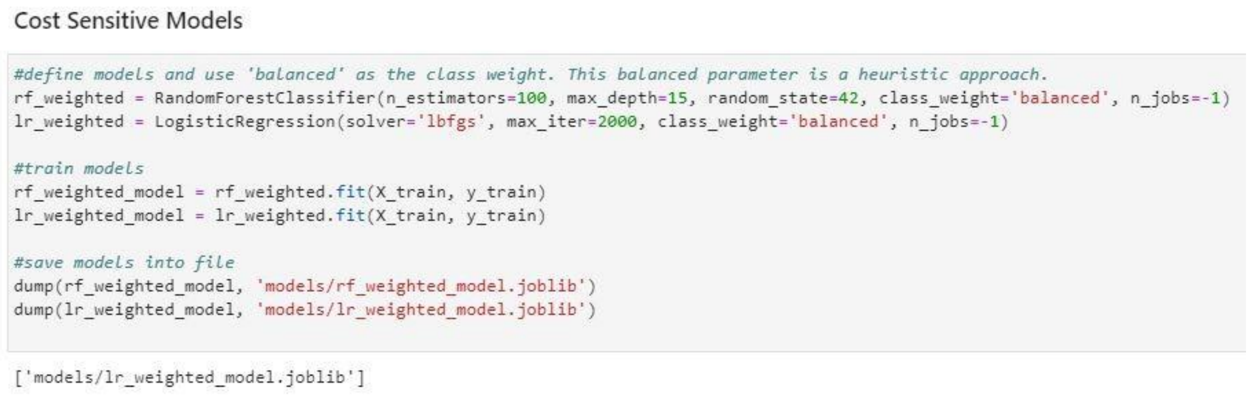


Oversampling training set

```
#define basic models
rfsmt = RandomForestClassifier(n_estimators=100, max_depth=15, random_state=42, n_jobs=-1)
lrsmt = LogisticRegression(solver='lbfgs', max_iter=1000, n_jobs=-1)

#create a oversampled data version
smt = SMOTENC(categorical_features=[7, 74],random_state=21, n_jobs=-1)
X_smt, y_smt = smt.fit_resample(X_train,y_train)

print("Number of Samples: ",Counter(y_smt))

smt_rf_model = rfsmt.fit(X_smt, y_smt)
smt_lr_model = lrsmt.fit(X_smt, y_smt)

dump(smt_rf_model, 'models/smt_rf_model.joblib')
dump(smt_lr_model, 'models/smt_lr_model.joblib')
```

```
Number of Samples:  Counter({'M': 97159, 'P': 97159, 'S': 97159, 'F': 97159})
```

- Cost Sensitive or weighted models. Both algorithms offers heuristic approach to handle imbalance dataset by applying weighting configuration during training. The weighting can penalize the model less for errors made on examples from the majority class and penalize the model more for errors made on examples from the minority class.


Cost Sensitive Models

```
#define models and use 'balanced' as the class weight. This balanced parameter is a heuristic approach.
rf_weighted = RandomForestClassifier(n_estimators=100, max_depth=15, random_state=42, class_weight='balanced', n_jobs=-1)
lr_weighted = LogisticRegression(solver='lbfgs', max_iter=2000, class_weight='balanced', n_jobs=-1)

#train models
rf_weighted_model = rf_weighted.fit(X_train, y_train)
lr_weighted_model = lr_weighted.fit(X_train, y_train)

#save models into file
dump(rf_weighted_model, 'models/rf_weighted_model.joblib')
dump(lr_weighted_model, 'models/lr_weighted_model.joblib')
```

```
['models/lr_weighted_model.joblib']
```

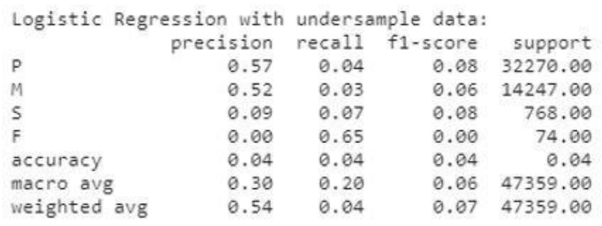
4. Evaluation

I use F1 score as the main metric to evaluate the performance of the different trained models. Accuracy is not best metric of choice when evaluating imbalanced datasets as it would bias toward majority class.

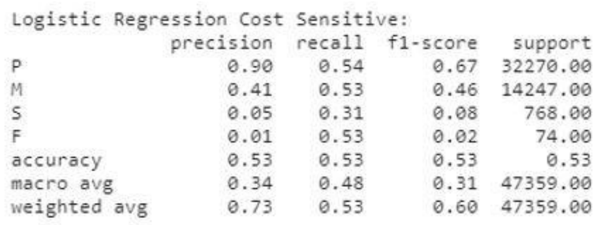
- Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- Recall: the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- F1 Score: the weighted average of precision and recall.

Each model is evaluated to predict classes from test data that has 47,359 samples.

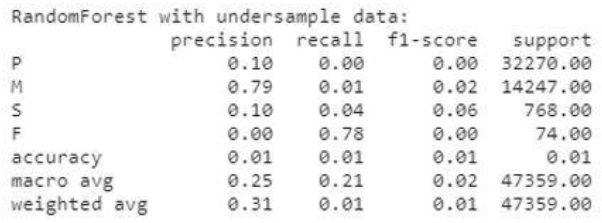
Classification Report



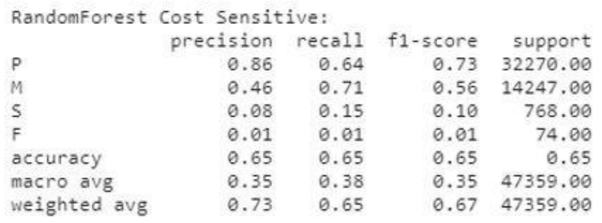
```
Logistic Regression with undersample data:
              precision    recall  f1-score   support
P                  0.57      0.04      0.08  32270.00
M                  0.52      0.03      0.06  14247.00
S                  0.09      0.07      0.08    768.00
F                  0.00      0.65      0.00     74.00
accuracy           0.04      0.04      0.04      0.04
macro avg          0.30      0.20      0.06  47359.00
weighted avg       0.54      0.04      0.07  47359.00
```



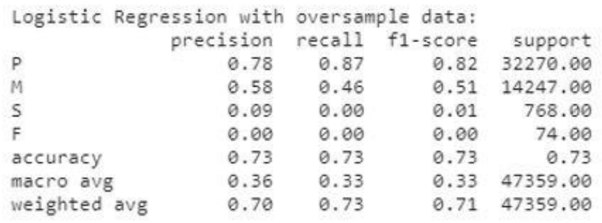
```
Logistic Regression Cost Sensitive:
              precision    recall  f1-score   support
P                  0.90      0.54      0.67  32270.00
M                  0.41      0.53      0.46  14247.00
S                  0.05      0.31      0.08    768.00
F                  0.01      0.53      0.02     74.00
accuracy           0.53      0.53      0.53      0.53
macro avg          0.34      0.48      0.31  47359.00
weighted avg       0.73      0.53      0.60  47359.00
```



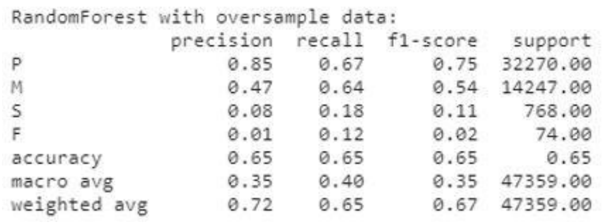
```
RandomForest with undersample data:
              precision    recall  f1-score   support
P                  0.10      0.00      0.00  32270.00
M                  0.79      0.01      0.02  14247.00
S                  0.10      0.04      0.06    768.00
F                  0.00      0.78      0.00     74.00
accuracy           0.01      0.01      0.01      0.01
macro avg          0.25      0.21      0.02  47359.00
weighted avg       0.31      0.01      0.01  47359.00
```



```
RandomForest Cost Sensitive:
              precision    recall  f1-score   support
P                  0.86      0.64      0.73  32270.00
M                  0.46      0.71      0.56  14247.00
S                  0.08      0.15      0.10    768.00
F                  0.01      0.01      0.01     74.00
accuracy           0.65      0.65      0.65      0.65
macro avg          0.35      0.38      0.35  47359.00
weighted avg       0.73      0.65      0.67  47359.00
```

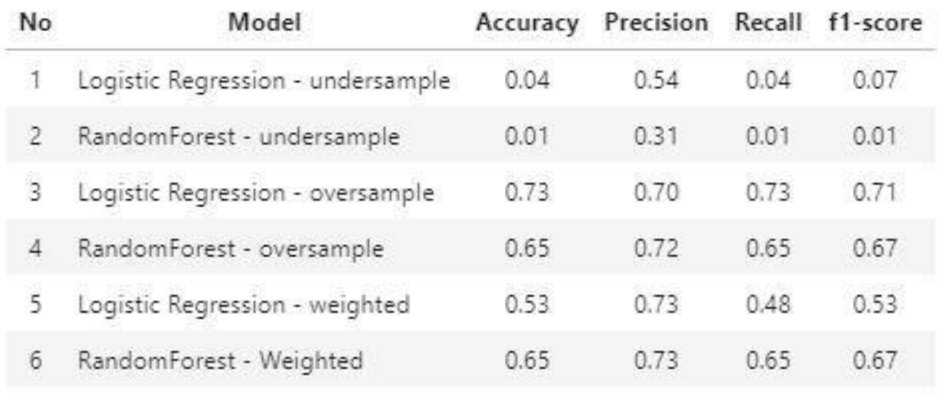


```
Logistic Regression with oversample data:
              precision    recall  f1-score   support
P                  0.78      0.87      0.82  32270.00
M                  0.58      0.46      0.51  14247.00
S                  0.09      0.00      0.01    768.00
F                  0.00      0.00      0.00     74.00
accuracy           0.73      0.73      0.73      0.73
macro avg          0.36      0.33      0.33  47359.00
weighted avg       0.70      0.73      0.71  47359.00
```



```
RandomForest with oversample data:
              precision    recall  f1-score   support
P                  0.85      0.67      0.75  32270.00
M                  0.47      0.64      0.54  14247.00
S                  0.08      0.18      0.11    768.00
F                  0.01      0.12      0.02     74.00
accuracy           0.65      0.65      0.65      0.65
macro avg          0.35      0.40      0.35  47359.00
weighted avg       0.72      0.65      0.67  47359.00
```

Models Metrics Summary



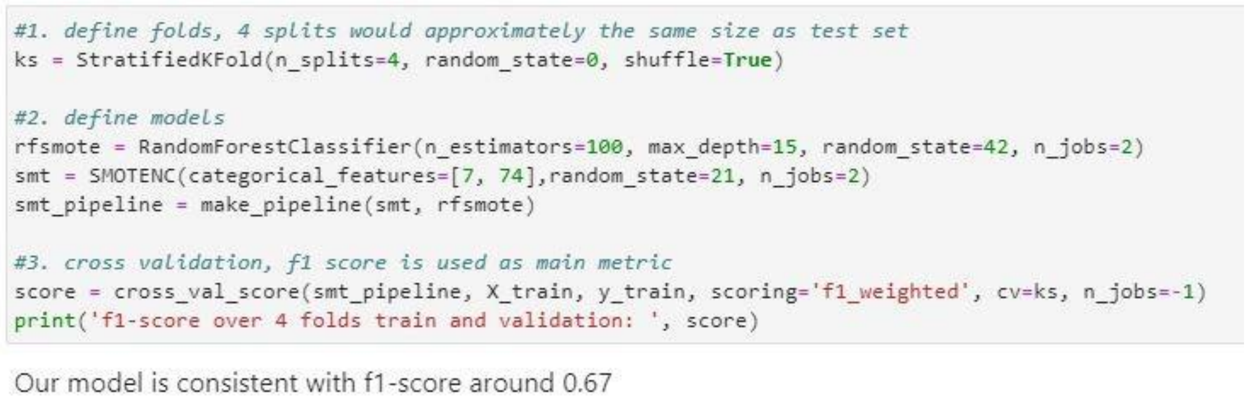
No	Model	Accuracy	Precision	Recall	f1-score
1	Logistic Regression - undersample	0.04	0.54	0.04	0.07
2	RandomForest - undersample	0.01	0.31	0.01	0.01
3	Logistic Regression - oversample	0.73	0.70	0.73	0.71
4	RandomForest - oversample	0.65	0.72	0.65	0.67
5	Logistic Regression - weighted	0.53	0.73	0.48	0.53
6	RandomForest - Weighted	0.65	0.73	0.65	0.67

Looking at the table above, it seems like Logistic Regression - oversample model is the best with f1 score 0.71. However, from the report we can see it failed to predict any sample from class F. Both models with undersample data are the worst. This is intuitively correct as trained models only have 257 samples.

Overall the **winner model** is **RandomForest - Oversample** that has f1 score 0.67. From the report, its recall score is 0.12 for class F. That means only 9 samples predicted correctly in class F.

4.1 Cross Validation - Winner Model

On the winner model, oversampling was done only one time over minority classes. We need to do cross validation on our model to verify its consistency. First we split the dataset into training and validation folds, oversampling, train classifier on training folds and then finally validate the classifier on remaining fold.

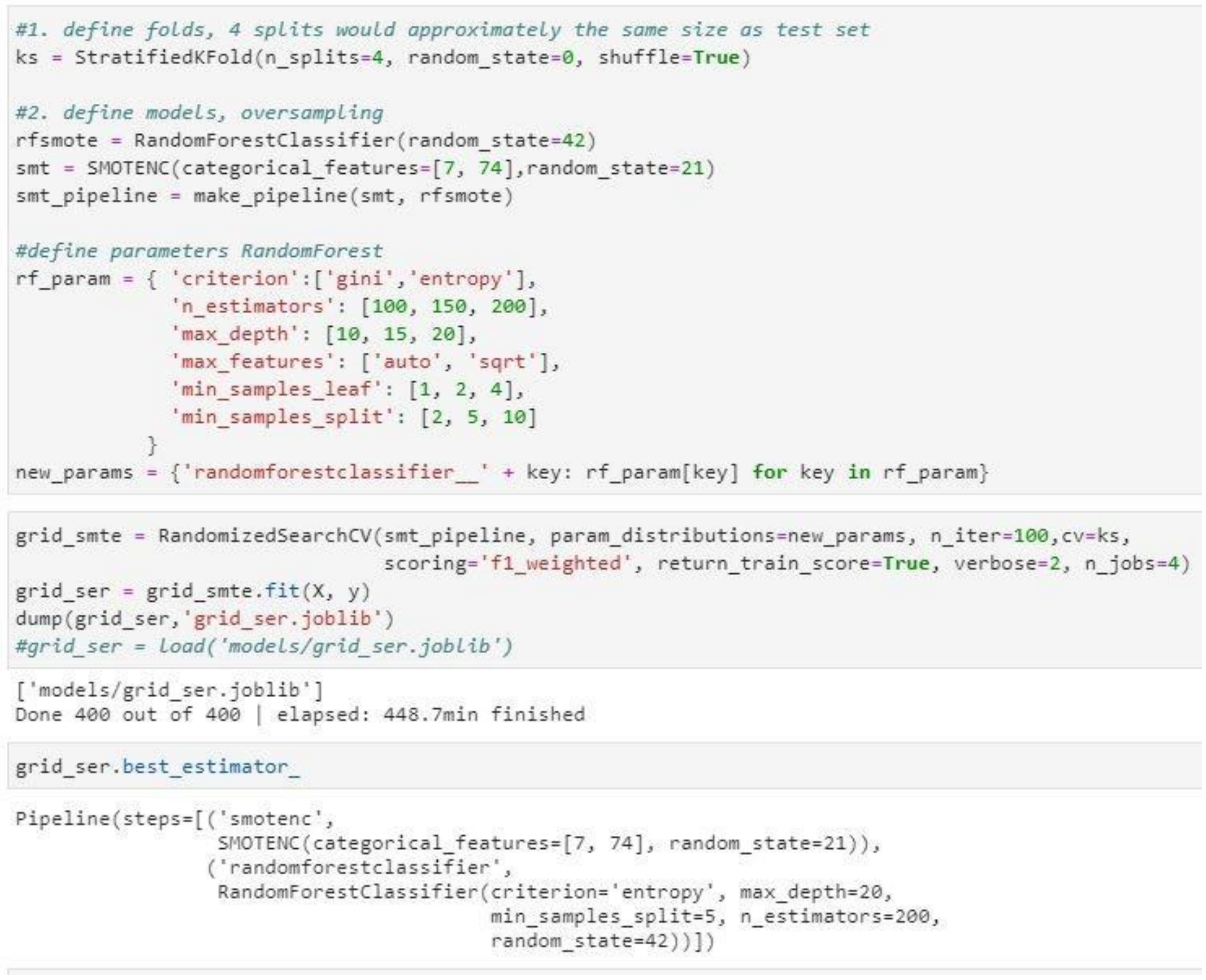


```python
#1. define folds, 4 splits would approximately the same size as test set
ks = StratifiedKFold(n_splits=4, random_state=0, shuffle=True)

#2. define models
rfsmote = RandomForestClassifier(n_estimators=100, max_depth=15, random_state=42, n_jobs=2)
smt = SMOTENC(categorical_features=[7, 74],random_state=21, n_jobs=2)
smt_pipeline = make_pipeline(smt, rfsmote)

#3. cross validation, f1 score is used as main metric
score = cross_val_score(smt_pipeline, X_train, y_train, scoring='f1_weighted', cv=ks, n_jobs=-1)
print('f1-score over 4 folds train and validation: ', score)
```

Our model is consistent with f1-score around 0.67

4.2 RandomizedSearchCV - Hyperparameter Tuning Winner Model

The winner model is further investigated to see if its performance can be increased by tuning some hyperparameter in the model. The process is almost the same with cross validation mentioned previously. The difference is combination of parameters on each train fold are tested to find the best f1 score. This whole process took 448.7 minutes in a single quad core machine.



```
#1. define folds, 4 splits would approximately the same size as test set
ks = StratifiedKFold(n_splits=4, random_state=0, shuffle=True)

#2. define models, oversampling
rfsmote = RandomForestClassifier(random_state=42)
smt = SMOTENC(categorical_features=[7, 74],random_state=21)
smt_pipeline = make_pipeline(smt, rfsmote)

#define parameters RandomForest
rf_param = { 'criterion':['gini','entropy'],
             'n_estimators': [100, 150, 200],
             'max_depth': [10, 15, 20],
             'max_features': ['auto', 'sqrt'],
             'min_samples_leaf': [1, 2, 4],
             'min_samples_split': [2, 5, 10]
           }
new_params = {'randomforestclassifier__' + key: rf_param[key] for key in rf_param}
```

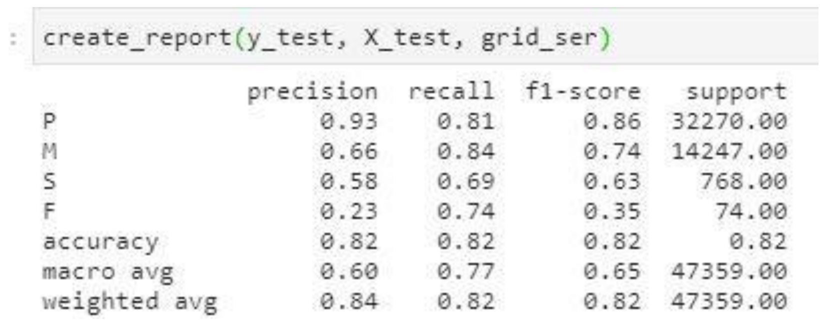
```
grid_smte = RandomizedSearchCV(smt_pipeline, param_distributions=new_params, n_iter=100,cv=ks,
                               scoring='f1_weighted', return_train_score=True, verbose=2, n_jobs=4)
grid_ser = grid_smte.fit(X, y)
dump(grid_ser,'grid_ser.joblib')
#grid_ser = load('models/grid_ser.joblib')
```

```
['models/grid_ser.joblib']
Done 400 out of 400 | elapsed: 448.7min finished
```

```
grid_ser.best_estimator_
```

```
Pipeline(steps=[('smotenc',
                 SMOTENC(categorical_features=[7, 74], random_state=21)),
                ('randomforestclassifier',
                 RandomForestClassifier(criterion='entropy', max_depth=20,
                                        min_samples_split=5, n_estimators=200,
                                        random_state=42))])
```

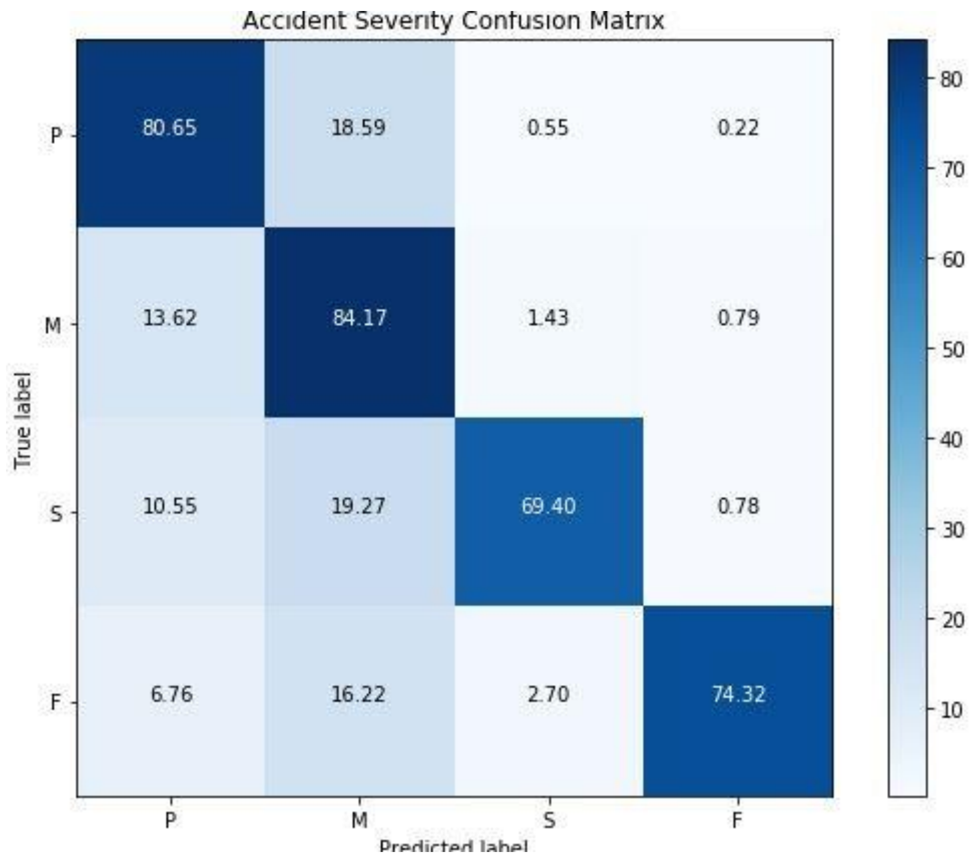
Classification report with tuned parameters:



```
create_report(y_test, X_test, grid_ser)
```

```
              precision    recall  f1-score   support
P                  0.93      0.81      0.86  32270.00
M                  0.66      0.84      0.74  14247.00
S                  0.58      0.69      0.63    768.00
F                  0.23      0.74      0.35     74.00
accuracy           0.82      0.82      0.82      0.82
macro avg          0.60      0.77      0.65  47359.00
weighted avg       0.84      0.82      0.82  47359.00
```

Confusion matrix with tuned parameters:



With tuned parameters, the model overall score were improved, specifically F1 score increased from 0.67 to 0.82.

5. Discussion

The clean dataset has 75 features that mostly categorical. Majority of the features are somewhat interrelated to each other. That said, no features left behind...A tuned parameter RandomForest fed with oversampled data can predict 74% true positive on 74 samples out of 47,359 samples in class F, a 0.15% of test set population. It's quite impressive.

6. Conclusion

Car accident severity prediction is a classic problem. Many academic papers have been published on this matter with different approaches statistical, machine learning, and deep learning. In this study, I use machine learning approach to analyze the relationship between accident severity with given set of features. The challenge is real life data always imbalance. Synthetic Minority Oversampling Technique (SMOTE) is used to solve imbalance data and followed by tuned RandomForest algorithm performs well.

7. References

- **https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html**
- **https://towardsdatascience.com/predicting-crash-severity-for-nz-road-accidents-6214117e73fb**
- **https://imbalanced-learn.readthedocs.io/en/stable/index.html**
- Intensive search in **stackoverflow.com**