Data Science in Car Accident Severity

1. Introduction/Business Problem.

Car accidents are a huge problem in our world. This project aims in analyzing "car accident severity" in terms of human fatality, traffic delay, property damage, or any other chance of fatality. In order to reduce car collisions in a community, a data science model must be trained to predict the severity of an accident. Taking consideration of the current weather, road and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful.

2. Data section

The dataset for this project is taken from a shared data link which is collected from Seattle SPOT Traffic Management Division. This is the shared data for Seattle city. The dataset is in the form of .CSV file. This includes all types of collisions. Collisions will display at the intersection or mid-block of a segment. The target label for the dataset is severity, which describes the fatality of an accident. The shared data has unbalanced labels. This dataset is updated weekly and is from 2004 to present. It contains information such as severity code, address type, location, collision type, weather, road condition, speeding, etc.,. There are 37 attributes in this dataset.

This Project for everyone who really care about the traffic records, especially in the transportation department.
This model is to improve the predictability of the accident severity and to reduce accidents in the future.

The result helps SHSP, DMV, stakeholders, insurance company, car manufacturers, and partnerships to allocate budget for education and enforcement to act on the result in order to achieve the goal of minimizing fatal/injury car crash.

The link for the dataset: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv

The link for the Metadata of the dataset: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Metadata.pdf

Some of the Metadata given with the dataset:

The target Data to be predicted under (**SEVERITYCODE** 1-prop damage 2-injury) label.
Other important variables include:

- **ADDRTYPE**: Collision address type: Alley, Block, Intersection
- **LOCATION**: Description of the general location of the collision
- **PERSONCOUNT**: The total number of people involved in the collision helps to identify severity involved

- **PEDCOUNT**: The number of pedestrians involved in the collision helps to identify severity involved
- **PEDCYLCOUNT**: The number of bicycles involved in the collision helps to identify severity involved
- **VEHCOUNT**: The number of vehicles involved in the collision helps to identify severity involved
- **JUNCTIONTYPE**: Category of junction at which collision took place helps to identify where most collisions occur
- **WEATHER**: A description of the weather conditions during the time of the collision
- **ROADCOND**: The condition of the road during the collision
- **LIGHTCOND**: The light conditions during the collision
- **SPEEDING**: Whether speeding was a factor in the collision (Y/N)
- **HITPARKEDCAR**: Whether the collision involved hitting a parked car

3. Methodology section

a. Data Analysis:

We have unbalanced dataset hence we must balance it. We should extract and convert the dataset into a proper format.

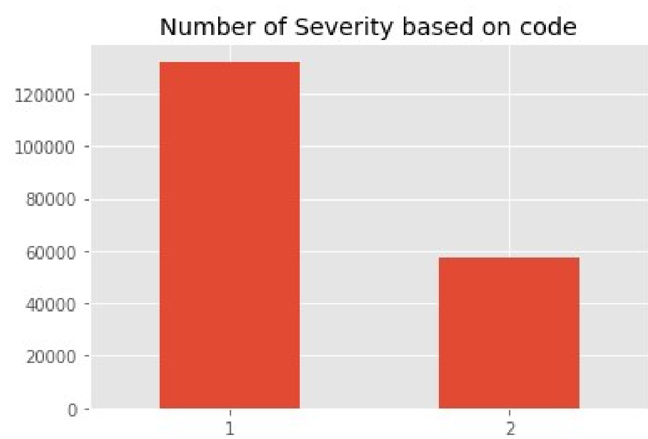
	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight

After dropping the unwanted columns and unknown numbers (NaN) the data types of the new columns in our data frame

```
SEVERITYCODE        int64
X                 float64
Y                 float64
ADDRTYPE           object
LOCATION           object
EXCEPTRSNCODE      object
EXCEPTRSNDESC      object
SEVERITYCODE.1      int64
SEVERITYDESC       object
COLLISIONTYPE      object
PERSONCOUNT         int64
PEDCOUNT            int64
PEDCYLCOUNT         int64
VEHCOUNT            int64
INCDATE            object
INCDTTM            object
JUNCTIONTYPE       object
INATTENTIONIND     object
UNDERINFL          object
WEATHER            object
ROADCOND           object
LIGHTCOND          object
SPEEDING           object
HITPARKEDCAR       object
dtype: object
```

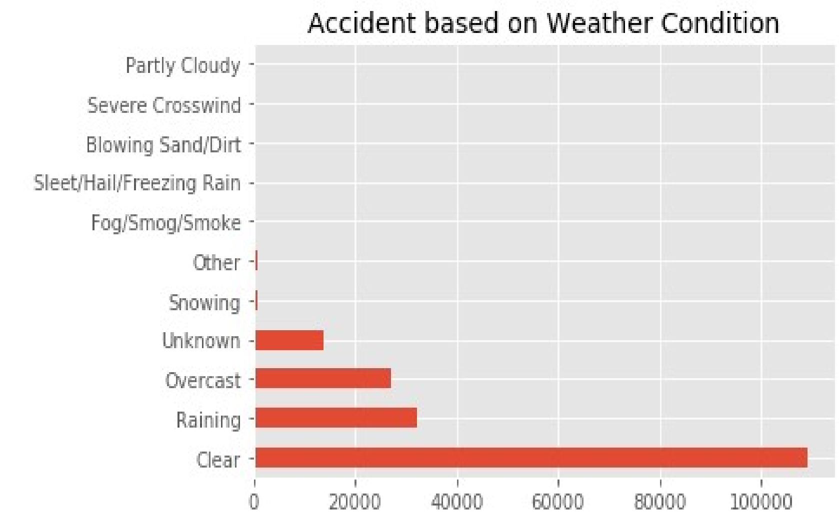
Balancing the Dataset

Our target variable SEVERITYCODE is only 42% balanced. In fact, severitycode in class 1 is nearly three times the size of class 2.

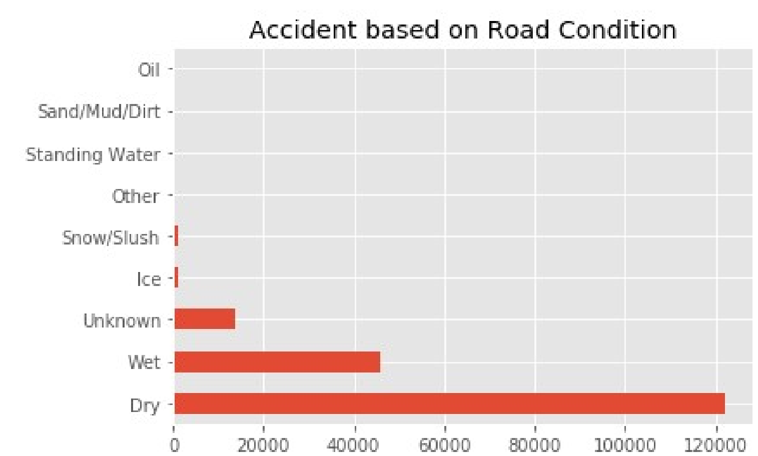


Calculating the total number of car accidents under different situations

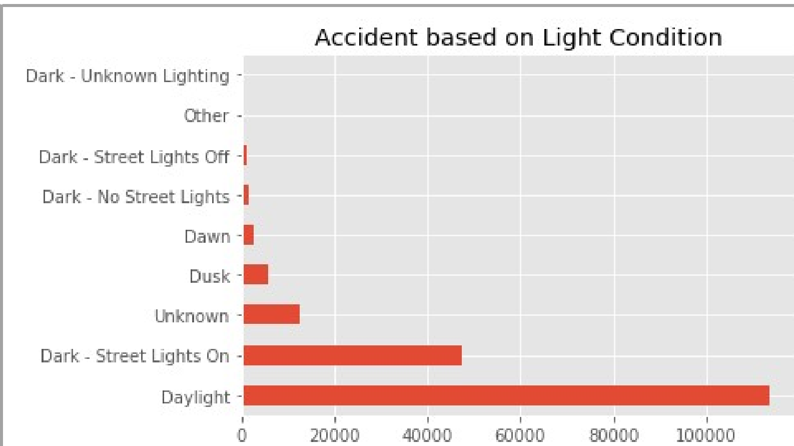
index	WEATHER
Clear	108959
Raining	32015
Overcast	27136
Unknown	13893
Snowing	894
Other	773
Fog/Smog/Smoke	553
Sleet/Hail/Freezing Rain	112
Blowing Sand/Dirt	50
Severe Crosswind	24
Partly Cloudy	5



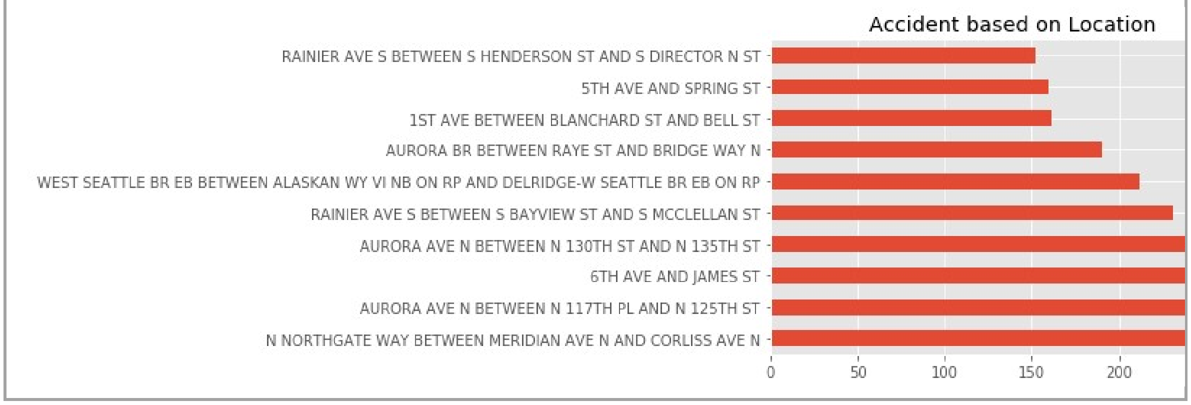
index	ROADCOND
Dry	122076
Wet	46064
Unknown	13839
Ice	1177
Snow/Slush	989
Other	117
Standing Water	102
Sand/Mud/Dirt	64
Oil	53

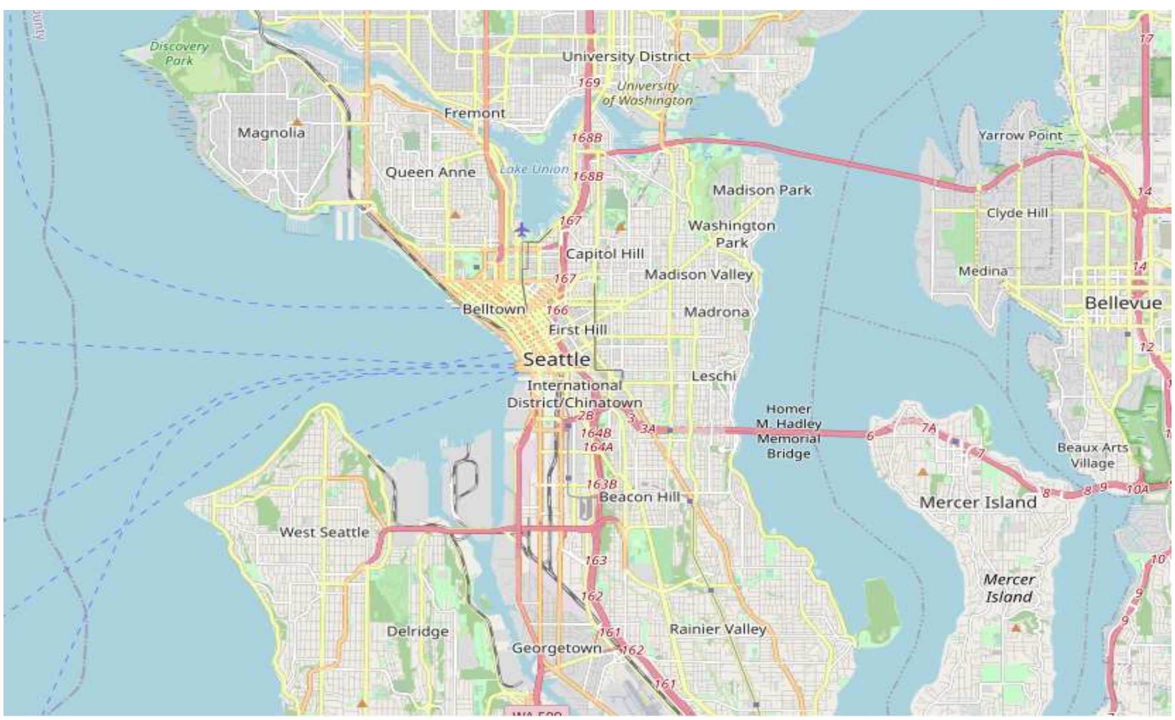


index	LIGHTCOND
Daylight	113582
Dark - Street Lights On	47314
Unknown	12432
Dusk	5775
Dawn	2422
Dark - No Street Lights	1451
Dark - Street Lights Off	1152
Other	188
Dark - Unknown Lighting	11



index	LOCATION
N NORTHGATE WAY BETWEEN MERIDIAN AVE N AND COR...	265
AURORA AVE N BETWEEN N 117TH PL AND N 125TH ST	254
6TH AVE AND JAMES ST	252
AURORA AVE N BETWEEN N 130TH ST AND N 135TH ST	239
RAINIER AVE S BETWEEN S BAYVIEW ST AND S MCCLE...	231
WEST SEATTLE BR EB BETWEEN ALASKAN WY VI NB ON...	212
AURORA BR BETWEEN RAYE ST AND BRIDGE WAY N	190
1ST AVE BETWEEN BLANCHARD ST AND BELL ST	161
5TH AVE AND SPRING ST	160
RAINIER AVE S BETWEEN S HENDERSON ST AND S DIR...	152





We will use the following models:

a. **K-Nearest Neighbor (KNN)**
   i. KNN will predict the severity code of an outcome by finding the most like data point within k distance.

b. **Decision Tree**
   i. A decision tree model will give the layout of all possible outcomes, so the model predicts all the different consequences of a decision. The decision tree observes all possible outcomes of different weather conditions.

c. **Logistic Regression**
   i. As the dataset only has two severity code outcomes, the model will only predict one of those two classes. This makes the data binary, which is perfect to use with logistic regression.

Initialization

Normalize the dataset

```
In [68]:  X = np.asarray(df[['WEATHER_ID', 'ROAD_ID','LIGHT_ID']] .values)
          X[0:5]

Out[68]: array([[3, 2, 1],
                [2, 2, 2],
                [3, 1, 1],
                [1, 1, 1],
                [2, 2, 1]], dtype=int64)
```

```
In [69]:  y = df['SEVERITYCODE'].values
          y[0:5]

Out[69]: array([2, 1, 1, 1, 2], dtype=int64)
```

Train/Test Split

We will use 30% of our data for testing and 70% for training.

```
In [70]:  X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
          X[0:5]

Out[70]: array([[ 1.71174445,  1.26999435, -0.56022029],
                [ 0.53070702,  1.26999435,  0.6752644 ],
                [ 1.71174445, -0.56446027, -0.56022029],
                [-0.65033041, -0.56446027, -0.56022029],
                [ 0.53070702,  1.26999435, -0.56022029]])
```

```
yhat = neigh.predict(X_test)
yhat[0:5]

: array([1, 1, 1, 1, 1], dtype=int64)
```

```
In [71]:  from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
          print ('Train set:', X_train.shape,  y_train.shape)
          print ('Test set:', X_test.shape,  y_test.shape)

          Train set: (133364, 3) (133364,)
          Test set: (33341, 3) (33341,)
```

KNN

In [72]:
```
from sklearn.neighbors import KNeighborsClassifier

k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[72]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

In [74]:
```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.6706232566509702
Test set Accuracy:  0.6708856962898533
```

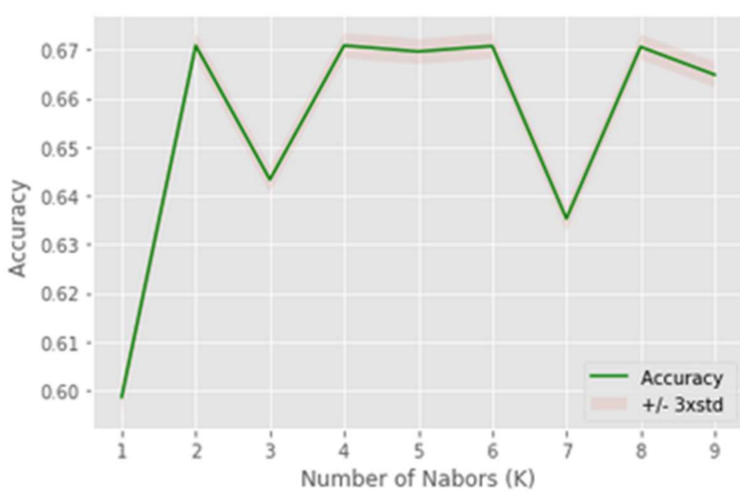
In [75]:
```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Out[75]:
```
array([0.59863231, 0.6708857 , 0.64332204, 0.6708857 , 0.66965598,
       0.67079572, 0.63531388, 0.67061576, 0.66488708])
```



Decision Tree Classifier

In [93]:
```
DTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
DTree
```

Out[93]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [94]:
```
DTree.fit(X_trainset,y_trainset)
```

Out[94]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

Logestic Regression

In [81]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[81]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

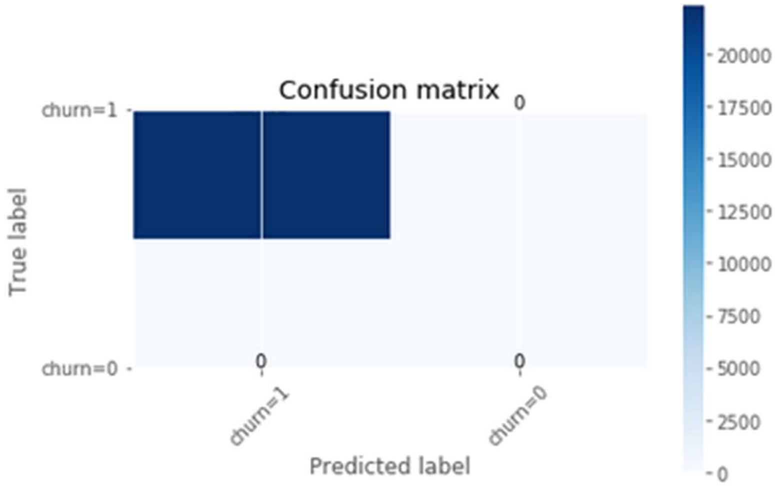
In [82]:
```
yhat = LR.predict(X_test)
yhat
```

Out[82]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [83]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[83]:
```
array([[0.66258059, 0.33741941],
       [0.66258059, 0.33741941],
       [0.67875559, 0.32124441],
       ...,
       [0.67840944, 0.32159056],
       [0.66660727, 0.33339273],
       [0.66091408, 0.33908592]])
```

```
Confusion matrix, without normalization
[[22383     0]
 [    0     0]]
```



4. Results section

The accuracy of the three models

KNN

```
In [77]:  from sklearn.metrics import f1_score
          f1_score(y_test, yhat, average='weighted')

Out[77]:  0.5484192629328207

In [78]:  from sklearn.metrics import jaccard_similarity_score
          jaccard_similarity_score(y_test, yhat)

Out[78]:  0.6648870759725263
```

Decision Tree Classifier

```
In [103]:  from sklearn.metrics import f1_score
           f1_score(y_test, yhat, average='weighted')

Out[103]:  0.5393189496069195

In [104]:  from sklearn.metrics import jaccard_similarity_score
           jaccard_similarity_score(y_test, yhat)

Out[104]:  0.6713355928136528
```

Logestic Regression

```
In [105]:  from sklearn.metrics import jaccard_similarity_score
           jaccard_similarity_score(y_test, yhat)

Out[105]:  0.6713355928136528

In [106]:  from sklearn.metrics import f1_score
           f1_score(y_test, yhat, average='weighted')

Out[106]:  0.5393189496069195

In [107]:  from sklearn.metrics import log_loss
           log_loss(y_test, yhat_prob)

Out[107]:  0.6327685830156192
```

5. Discussion section

After analyzing and cleaning the data, it is fed through three ML models namely K-Nearest Neighbor also known as KNN, Decision Tree and Logistic Regression. The evaluation metrics for these three models that test the accuracy of the models are the Jaccard index, f-1 score and logloss for logistic regression.

6. Conclusion section

What is the best model? The quality of the model shouldn't be only measured by accuracy rate, but the simplicity, easily understandable by decision maker and convenience to implement do matter the most

Logistic regression showed good performance than the other two models and can be the best model to implement for the reduction of fatal/ injury accident. There is no perfect solution, only a solution that is good enough for the intended purpose. This purpose can – and in many cases should– grow in complexity and sophistication as the results prove more and more useful and provide a feedback loop on how to improve themselves.

Thanks for reading!