# City of Chicago Street Parking Ticket Assessment

## Table of Contents

1. Objective
2. Result
3. Future Development
4. Data Source
5. Deliverables
   1. Single Bash Script
   2. Exploratory Data Analysis (EDA)
   3. Traffic Citations Modeling ETL Pipeline
   4. Preprocessing and Encoding Categorical Variables
   5. Model Selection
   6. Training and Validation
6. Business Questions
7. Conclusion and Final Thoughts

## Objective

* Calculate the probability that a vehicle will be receiving a parking ticket during the staying period in one of the top 25 regions based on the City of Chicago local zoning and regions in this dataset based on the available and reliable features.
* Identifying the benefit of using Pandas vs. SQLite as the ETL pipeline’s main library

## Result

The probability of a vehicle to be receiving a parking ticket during the staying period in one of the top 25 regions based on the City of Chicago local zoning and regions is calculated as follows. In this dataset based on the available and reliable features about 92% of the vehicles are likely to receive a parking ticket.

## Future Development

As of end of November 2019, the items highlighted as follows are all completed. Some decisions need to be made.

* Data Ingestion Automation from the City of Chicago AWS Enterprise Account S3 Bucket.

The boto3 library is selected to download the data s3 bucket with .get\_object methods. The client is to decide the frequency of the data ingestion and the preferred schedule.

* Deploy the model server. The client is to decide what server/instance is needed to be dedicated to the task/schedule
* Post request and return the prediction in JSON format

Use the Flask and Server-side sessions and serializing with Pickle library which is aligned with the standard and best practice of containerization in the production environment. This enables the model to be readily available if/when production phase starts.

## Data Source

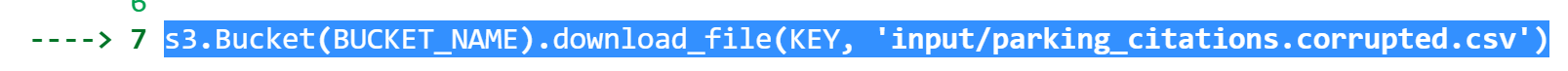
The available data source provided for the purpose of this submittal was ingested from the City of Chicago AWS Enterprise Account S3 Bucket located at the following address;

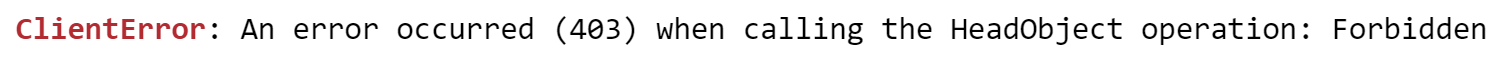
<https://s3-us-west-2.amazonaws.com/pcadsSection/parking_citations.corrupted.csv>

There were about 800 million rows available in this dataset with some vehicle manufacturers are more common than others.

Loading the data by utilizing boto3 client and resource to download the s3 bucket with .download\_file and .get\_object methods was facing a couple of unsuccessful attempt due to the following errors

Client Error: Forbidden and Signature Miss-match

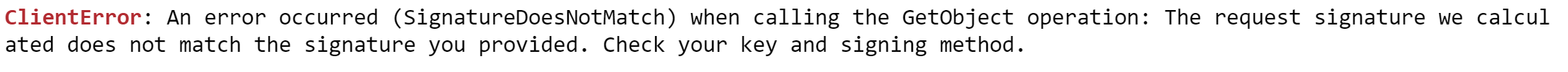




After resolving the issue with the miss-matching signature to access the AWS S3 Bucket the following error occurred. This error is identified due to the extremely large size of the dataset.

Client Error: Operation





The solution to resolve this error was to use “incremental loading” process. Please review the s3\_data\_ingest.py provided in the repository.

s3\_data\_ingest.py



## Deliverables

The main deliverables of this submittals are a single bash script, ETL pipeline, and a supervised machine learning model (KNN classification). The supervised machine learning model, K nearest neighbors was selected to predict the probability that a vehicle will be receiving a parking ticket during the staying period in one of the top 25 regions based on the City of Chicago local zoning and regions in this dataset based on the available and reliable features. The Uncorrupted half of the dataset was used to train, test, and validate the model, accuracy, and the overall hyper-parameter tuning.

The model also calculates the probability that a car is made by one of the top 25 manufacturers in this dataset based on the uncorrupted features. Some explanation about the quality of this model and feasibility of this task is also provided. The strengths and weaknesses of this model were discussed in section 5-3.

### Single Bash Script

a single bash script named "run.sh" that will allow you to run the entire modeling pipeline:

run.sh can be found in the root directory of the repository. The following script is simply used in the bash script.

### Exploratory Data Analysis (EDA)

The repository contains the following Jupyter notebook(s) for exploratory / explanatory components. Comments are provided in the notebook(s): Traffic\_Citations-Analysis.py.

Some explanatory and analytical components are provided which lead to the preprocessing and better understanding the dataset.

Any additional instructions necessary to ensure that the team can reproduce this result or to direct their attention to the right places will be provided with the next submittals.

### Traffic **Citations** ETL Pipeline

The goal was defined to set up a reproduceable set of extraction, transformation, and loading operations.

The final deliverable has been prepared along with this document. The following git-hub repository hosts all component deliverables.

The link to the git-hub Repository: <https://github.com/BijanVafaei1992/TrafficCitations_ModelingPipeline2>

With cloning the repository, your team should be able to reproduce the pipeline locally from start to finish including loading the data from S3 and launching the model server.

All necessary python scripts for generating outputs can be found in Traffic\_Citations.py.

Traffic\_Citations.py

### Preprocessing and Encoding Categorical Variables

In this machine learning task, the make label of the corrupted parking citations dataset needs to be predicted/classified based on the uncorrupted features of the same dataset. This model calculates the probability that a car is made by one of the top 25 manufacturers.

Common make feature is defined as a binary variable;

Common make = 1 means vehicle is made by one of the top 25 manufacturers.

Missing Values:

In this machine learning projects the corrupted data of parking citations need to be preprocessed in order to be in the ideal format for producing the best performing model.

Some missing values were observed with the following statistics. For handling the missing values, the following items need to be mentioned. Removal, replacement, and imputing are used as follows.

|  |  |  |
| --- | --- | --- |
| **Columns** | **Number of Null Values** | **Percent** |
| Ticket number | 0 | 0.00% |
| Issue Date | 536 | <0.1% |
| Issue time | 2583 | <0.1% |
| Meter Id | 6456512 | 74.00% |
| Marked Time | 8435415 | 96.70% |
| RP State Plate | 765 | <0.1% |
| Plate Expiry Date | 794827 | 9.10% |
| VIN | 8709705 | 99.90% |
| Make | 4368470 | 50.60% |
| Body Style | 8890 | 0.10% |
| Color | 4115 | <0.1% |
| Location | 854 | <0.1% |
| Route | 65354 | 0.80% |
| Agency | 545 | 0.10% |
| Violation code | 0 | 0.00% |
| Violation Description | 872 | <0.1% |
| Fine amount | 6507 | <0.1% |
| Latitude | 3 | <0.1% |
| Longitude | 3 | <0.1% |

- Dropped the datapoint with Null value in the following 10 columns with number of null values less than 70'000. This Effects less than 2% of the dataset

['Issue time', 'RP State Plate', 'Body Style', 'Color', 'Location', 'Route', 'Agency', 'Violation Description', 'Fine amount', 'Latitude', 'Longitude']

- Removed the whole column for the following columns due to the high number of null value and extreme noises especially in 'VIN'

['Meter Id', 'Marked Time', 'VIN']

- Replaced the Plate Expiry Date ['Plate Expiry Date'] with the constant value of 202012 incorporating the assumption that state have 3 years plate expiry date program

Data Manipulation:

- Modify the Ticket number contains a letter D at the end of them which shows that those ticket numbers were deleted or meant to be deleted. Remove the letter D for the purpose of this analytics.

- TODO: with Issue Date we can identify day of week, weekend or weekdays, and holiday to expand

-TODO: with ‘Location’, ‘Route’, 'Latitude', and 'Longitude' features we can include some of the new geo-spatial analytics

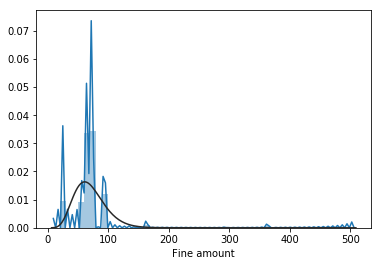
Encoding Categorical Variables:

There are often several transformational steps such as encoding categorical variables, feature scaling and normalization that need to be performed. Scikit-learn has built in functions for most of these commonly used transformations in the preprocessing package.

From sklearn.preprocessing, LabelEncoder is chosen for the ease of use and quick turnaround.

Normalization:

Fine Amount variable seems to fit the Gamma distribution Ranging from $10 to $505 with the mean of $70 and standard deviation of $32.



Normalization was performed on the Fine amount to transforms the feature by scaling each to a given range between Min and Max amount.

This transformation scales and translates the feature individually such that it is in the given range on the training set, e.g. between zero and one.

### Model Selection

Given the nature of problem and the explained setup, K\_ Nearest Neighbors (K-NN) from the family of supervised machine learning algorithms was selected. K-NN is going to use our labeled variables to predict the Make values that were accidentally deleted.

The K-NN algorithm is a robust classifier which is often used as a benchmark for other classifiers. Below is the list of reasons to choose K-NN machine learning algorithm:

Strength/Advantages of K-NN Model:

- Easy to use and intuitive:

K-NN algorithm is very simple to understand and equally easy to implement.

- No underlying assumptions:

K-NN is a non-parametric algorithm which means there are no assumptions to be met to implement K-NN. It does not make any assumptions about the probability distribution of the input. Parametric models like linear regression has lots of assumptions to be met by data before it can be implemented which is not the case with K-NN. This is useful for applications with input properties that are unknown and therefore makes k-NN more robust than algorithms that are parametric.

- Quick respond to changes in input:

K-NN employs lazy learning, and memory-based approach which generalize during testing. This allows it to

change during real-time use. It allows the algorithm to respond quickly to changes in the input during real-time use.

- Very easy to implement for multi-class problem:

Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.

Weaknesses/Disadvantages of K-NN Model:

* KNN requires expensive pre-evaluating to optime the hyper-parameters (number of nearest neighborhoods). The training and testing of each instance are relatively computationally intensive. Specialized algorithms and heuristics exist for specific problems and distance functions, which can mitigate this issue. This is problematic for datasets with many attributes. When the number of instances is much larger than the number of attributes, a R-tree can be used to store instances, allowing for fast neighbor identification.
* K-NN has higher sensitiveness to noisy or irrelevant attributes, which can result in less meaningful distance numbers. Scaling and/or feature selection are typically used in combination with kNN to mitigate this issue. Sensitiveness to very unbalanced datasets, where most entities belong to one or a few classes, and infrequent classes are therefore often dominated in most neighborhoods. This can be alleviated through balanced sampling of the more popular classes in the training stage, possibly coupled with ensembles.

Overall, for the purpose of this task, It is decided to utilize K-NN model

“from sklearn.neighbors import KNeighborsClassifier”.

### Training and Validation

Commonly used splitting the data around 20%-80% between testing and training approach was utilized for this supervised learning. The dataset was split into a training data and test data using scikit learn library.

“from sklearn.model\_selection import train\_test\_split”

Training Dataset: The sample of data used to fit the model

Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

Cross Validation (CV)

To evaluate the performance of the machine learning model, cross validation was performed on data. With that we can say weather our model is Under-fitting/Over-fitting/Well generalized.

Cross validation (CV) is one of the techniques used to test the effectiveness of a machine learning models, it is also a re-sampling procedure used to evaluate a model if we have a limited data.

Based on previous experience, K-fold Cross Validation, the most popular method of cross validation was introduced to avoid over fitting and general biases. The K-fold Cross Validation approach was also checked instead of train-test\_split method. Cross validation avoided over fitting and yielded higher accuracy on average.

Model Server Deliverable:

Deploy the local server (that will only be used locally) to allow a user to submit a corrupted row in some format and receive the probability of a top 25 'Make' in return.

The server should:

- expect a json of features. Ex: {'Color':'GR', 'Latitude':63453.0} etc.

{

"Parking\_Citations": [

{

"Ticket number": 1234567890,

"Issue time": "700",

"RP State Plate": "CA",

"Plate Expiry Date": "20180506",

"Body Style": "TR",

"Color": "BK",

"Location": "4TH/STATE",

"Route": "CM96",

"Agency": 1,

"Violation code":208,

"Violation Description":204,

"Fine amount":93,

"Latitude":99999,

"Longitude":99999

}

]

}

- work with a post request

Local host:

<http://localhost:3000/post/parking_citations>

- return a json with a "prediction" field and the appropriate model output value

Expository / Exploratory Analysis:

Model explanation / analysis should be delivered in a Jupyter notebook:

Comments are provided in the following jupyter notebooks:

Traffic\_Citations-Analysis.ipynb

1. **Section 2: Business Questions**

Pandas and SQLite libraries were used to calculate some statistics on this dataset are as follows

An SQLite database was set up to do a few comparisons with pandas. Both run time and ease of use are considered

This analysis is conducted on the uncorrupted component of the data only.

The runtime comparison between Pandas and Sqlite libraries is provided in the following Jupyter notebooks:

Sqlite\_Pandas\_Compare.ipynb

Runtime for Pandas performing the task a. was about 1 second whereas the same task took about 9.5 second using SQLite.

A thorough analysis among SQLite, SQLite memory, and pandas was conducted that can be found at the link of below;

<https://blog.thedataincubator.com/2018/05/sqlite-vs-pandas-performance-benchmarks/#targetText=pandas%20scales%20with%20the%20data,but%20this%20was%20the%20closest>

Results

SQLite is faster in selecting and filtering. Pandas is faster in loading, joining, and aggregating (groug by)

Overall, Pandas library seems to be optimized for group-by operations, where it performs well facing larger dataset. Moreover, Pandas scales with the data, up to just under 0.5 seconds for approximately 10 million records. Pandas is faster for group-by computation of a mean and sum, load data from disk, and join data.

The use cases for Pandas library has been exponentially increased over the course of last few years where the detail documentations, debugging, and error handling helps data scientists and engineers to move forward quicker with their problems in hand. Pandas provide higher ease of use in terms of syntax and intuitiveness.

pandas is a data analysis toolkit, a general purpose programming language. However, SQL is a domain-specific language for querying relational data (usually in a relational database management system which SQLite, MySQL, Oracle, SQL Server, PostgreSQL etc. are examples).

SQL implies a higher standard in working with data in an RDBMS, and database domain knowledge, and provide an

Faster learning curve.

The comparison of SQLite vs Pandas on the following tasks are also provided in the Jupyter notebook Sqlite\_Pandas\_Compare.ipynb:

1. calculate top 25 most common 'makes'

# Runtime:

#Pandas: 0.66 ± 0.07 seconds (faster)

#SQLite: 3.72 ± 2.4 seconds

# Result: Pandas handle the Group By and Aggregation task more efficiently and it is more widely used with greater level of

# details in supporting documentations

# Pandas perform more efficiently facing bigger datasets.

# Padas is easier to code

# Winner based on the run time and ease of use is Pandas

1. calculate most common 'Color' for each 'Make'

# Runtime:

#Pandas: 0.36 ± 0.026 second

# SQLite: Unknown

Result: Using Pandas.series.mode method with aggregation is a simple way to conclude this task

Black, White, Gray are most common

My experience with SQLite was not suitable for this task as it needed to use nested query u using ranks and frequently. Some of the methods were not supported by SQLite3

SQLite needs longer time to build up the code for this task

pd.series is useful method especially in facing variable with multi-modes.

pd.series.mode always returns a series. pandas.series has a comprehansive set of methods

which make it very compatible with agg and apply, especially when re-constructing the group by output.

It is fast and as easy as one line of code to build.

It creates a useful set of capabilities with no need to encode as an extra step around Categorical

Values Mean, Mode, Min, Max, and many more methods are readily available.

Winner: Pandas

1. find the first ticket issued for each 'Make'

# Runtime:

#Pandas: 1.38 ± 0.018 second

# SQLite: Unknown

Result: Using Pandas.series.min method with aggregation and creating the Issue Datetime is a simple way to conclude this task

Choose either pandas to answer the following question:

"Is an out-of-state license plate more likely to be expired at the time of receiving a ticket than an in-state license plate?"

First the binary feature of ‘Out of State’ was created assuming the state of Illinois was the host of this project;

Out of State: 0 if the state is CA

Out of State: 1 Otherwise

Then the Boolean for ‘Is Expired’ was introduced, if the ticket Issue Date was after Plate Expiry Date

Yes, it is more likely that a vehicle parked in Region to have an expired license plate at the time of receiving a parking ticket.

TODO: is the difference is statistically significant?

Out of 1,111,820 instances (1,042,974 + 68,846) which have expired license plate, 68,846 are from Out-of-State. Out of 3.7 million In-State vehicles (3,723,823), 28% of them have expired license plate (1,042,974).

Out of 237,174 vehicles registered Out-of-State, 29% of them have expired license (68,846);

The rest of Out-of-State vehicles (71%) had valid license plate (168,328)

|  |  |  |  |
| --- | --- | --- | --- |
| In Region? | Number Vehicles with Expired License Plate | Total Number of Vehicles | Percentage of Vehicle with Expired License Plate |
| No | 1,042,974 | 3,723,823 | 28.01% |
| Yes | 68,846 | 237,174 | 29.03% |
| Total Number of Vehicles | 1,111,820 | 3,960,997 | 28.07% |