Las Vegas Restaurant Inspections

1. Introduction

1.1 Problem Statement

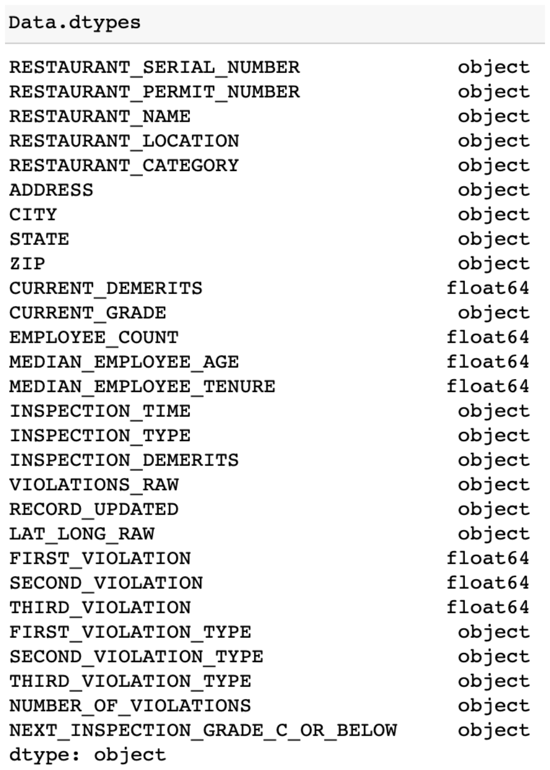
The main goal of this Capstone project is to explore the possibility of building a classification model to predict the outcome of a restaurant's next inspection based on the provided data of the previous inspection in Las Vegas. The subtasks consist of 1) to analyze the provided information and interpret all the information; 2) to select important features and perform data cleaning and preprocessing; 3) to find the best classifier to create a model and predict the outcomes.

1.2 Key Stakeholders

A stakeholder can be defined generally as any individuals, end-users, or organizations who have an interest in the food regulation, in which restaurant inspection is one of the key components. It is important to take into consideration the views and concerns of relevant stakeholders in policymaking. The stakeholders can come from consumers, enterprises of food business, and food regulatory and enforcement agencies

2. Data Analysis

The data types of each feature are checked:



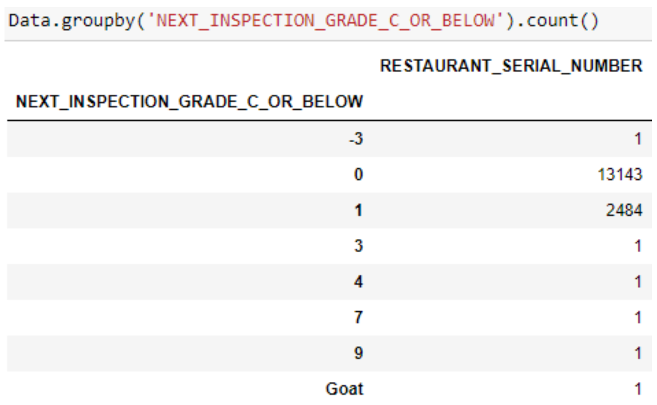
```
Data.dtypes

RESTAURANT_SERIAL_NUMBER             object
RESTAURANT_PERMIT_NUMBER             object
RESTAURANT_NAME                      object
RESTAURANT_LOCATION                  object
RESTAURANT_CATEGORY                  object
ADDRESS                              object
CITY                                 object
STATE                                object
ZIP                                  object
CURRENT_DEMERITS                    float64
CURRENT_GRADE                        object
EMPLOYEE_COUNT                      float64
MEDIAN_EMPLOYEE_AGE                 float64
MEDIAN_EMPLOYEE_TENURE              float64
INSPECTION_TIME                      object
INSPECTION_TYPE                      object
INSPECTION_DEMERITS                  object
VIOLATIONS_RAW                       object
RECORD_UPDATED                       object
LAT_LONG_RAW                         object
FIRST_VIOLATION                     float64
SECOND_VIOLATION                    float64
THIRD_VIOLATION                     float64
FIRST_VIOLATION_TYPE                 object
SECOND_VIOLATION_TYPE                object
THIRD_VIOLATION_TYPE                 object
NUMBER_OF_VIOLATIONS                 object
NEXT_INSPECTION_GRADE_C_OR_BELOW     object
dtype: object
```

The features contained in the dataset can be classified into the following categories:

- Info (10): RESTAURANT_SERIAL_NUMBER, RESTAURANT_PERMIT_NUMBER, RESTAURANT_NAME, RESTAURANT_LOCATION, ADDRESS, CITY, STATE, ZIP, LAT_LONG_RAW, VIOLATIONS_RAW
- Time (2): INSPECTION_TIME, RECORD_UPDATED.
- Categorical (6): RESTAURANT_CATEGORY, CURRENT_GRADE, INSPECTION_TYPE, FIRST_VIOLATION_TYPE, SECOND_VIOLATION_TYPE, THIRD_VIOLATION_TYPE.
- Target (1): NEXT_INSPECTION_GRADE_C_OR_BELOW (Have noise).
- Continous(9): CURRENT_DEMERITS, EMPLOYEE_COUNT, MEDIAN_EMPLOYEE_AGE, MEDIAN_EMPLOYEE_TENURE, INSPECTION_DEMERITS(NA inside), FIRST_VIOLATION, SECOND_VIOLATION, THIRD_VIOLATION, NUMBER_OF_VIOLATIONS (NA inside).

2.1 Target feature

The target feature ('NEXT_INSPECTION_GRADE_C_OR_BELOW') is first examined.



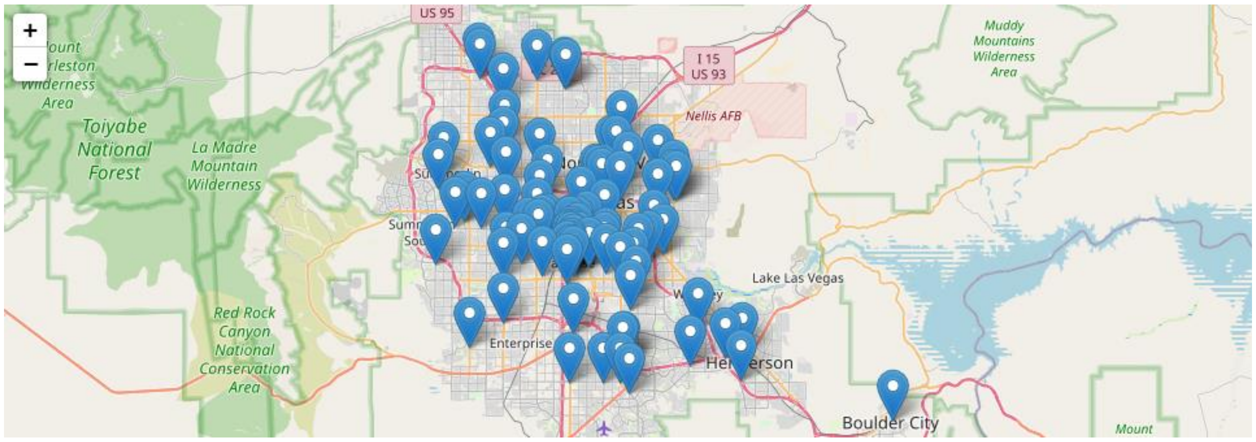
```
Data.groupby('NEXT_INSPECTION_GRADE_C_OR_BELOW').count()
```

NEXT_INSPECTION_GRADE_C_OR_BELOW	RESTAURANT_SERIAL_NUMBER
-3	1
0	13143
1	2484
3	1
4	1
7	1
9	1
Goat	1

The results show that target feature has outliers, including Goat, 9, 7, 4, 3, -3, and null. For target feature, 0 means Next inspection Grade not C or below C; 1 means Next inspection Grade is C or below C.

2.2 Location feature

100 restaurants are randomly selected from the list and presented their distribution on the map using folium function.

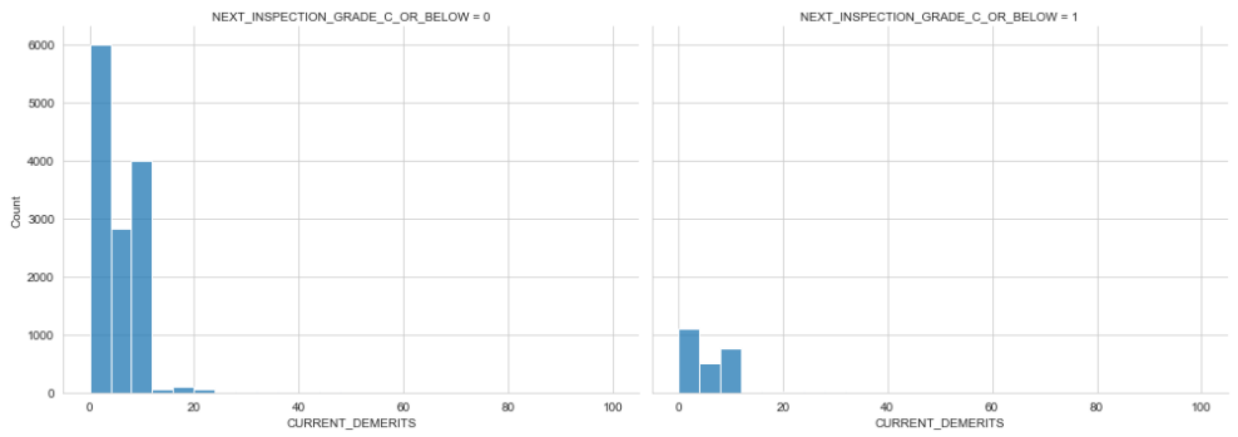

US 95
I 15
US 93
Nellis AFB
Muddy Mountains Wilderness Area
Toiyabe National Forest
La Madre Mountain Wilderness
Red Rock Canyon National Conservation Area
Enterprise
Lake Las Vegas
Henderson
Boulder City

2.3 Continuous features

All the continuous features are inspected and cleaned. Here, the key continuous features are visualized in terms of the target variable.

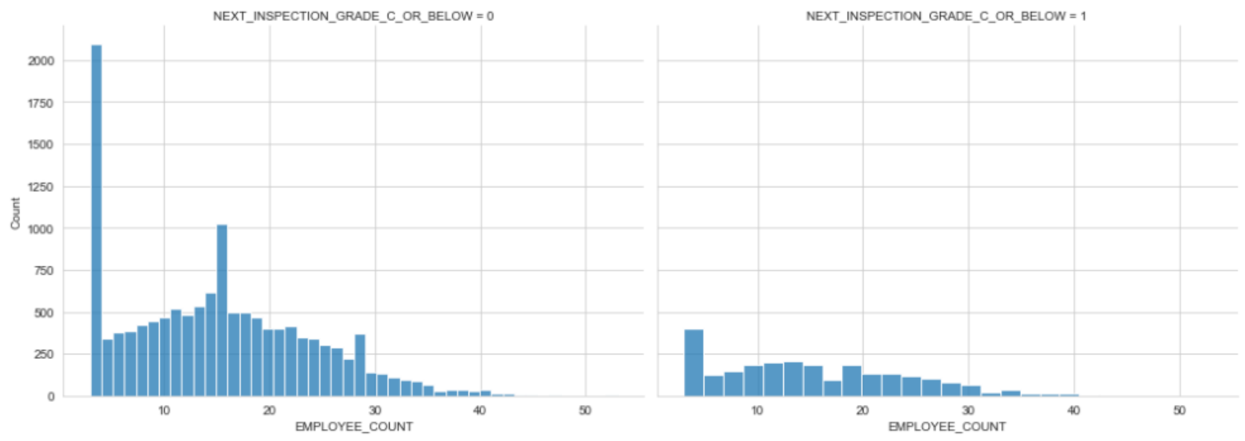
2.3.1 'CURRENT_DEMERITS'

There are negative and large numbers in ‘Current_Demerits’, which doesn’t make sense. So I use a threshold of 0 and 100 to exclude outliers and then filled NaN with the median.

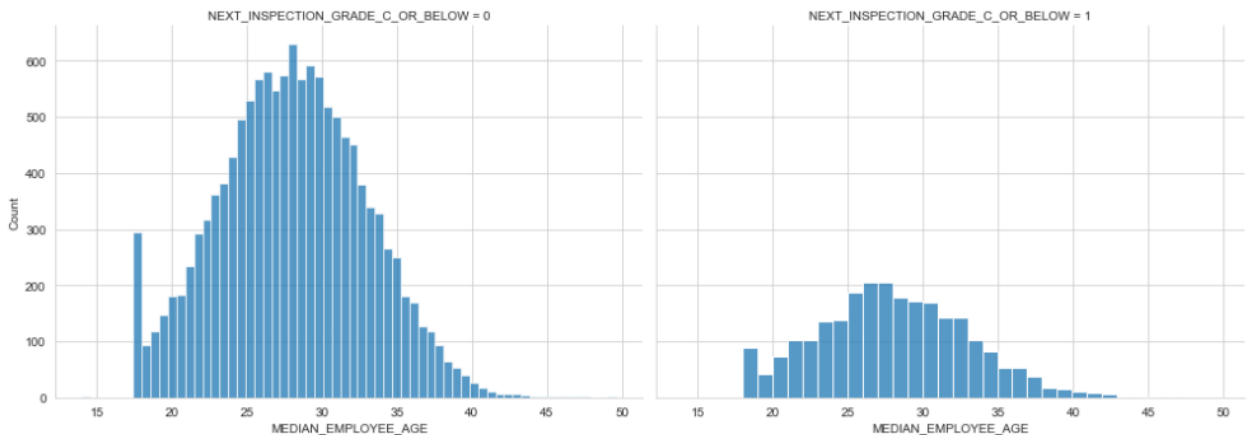


2.3.2 'EMPLOYEE_COUNT'

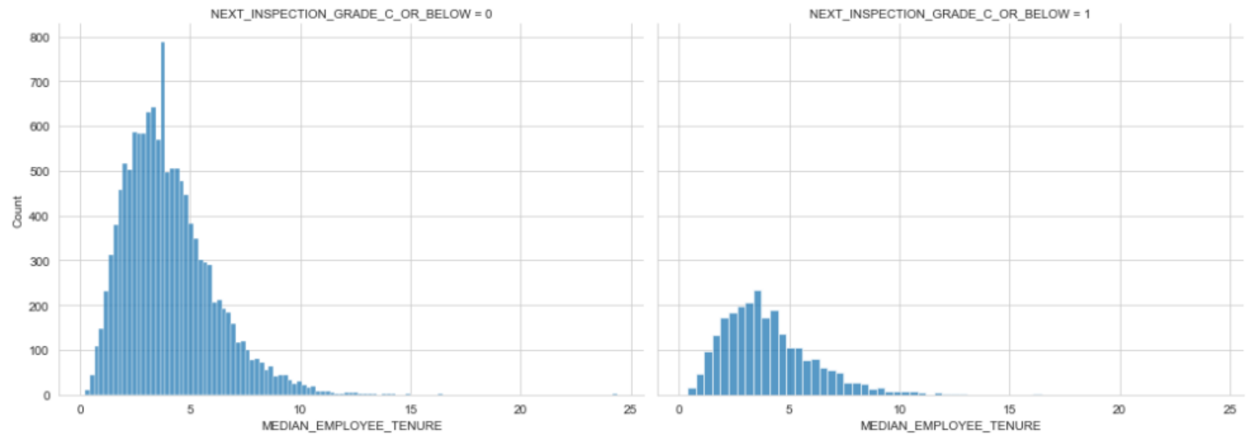
There are negative (-7.0) and large numbers (111447) in 'EMPLOYEE_COUNT'. So I excluded outliers and then filled NaN with the median.



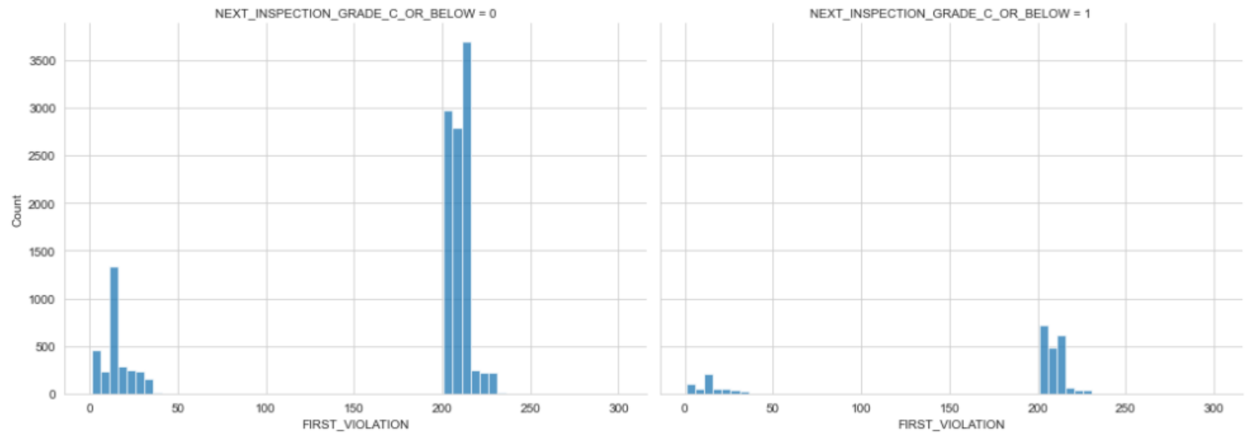
2.3.3 'MEDIAN_EMPLOYEE_AGE'



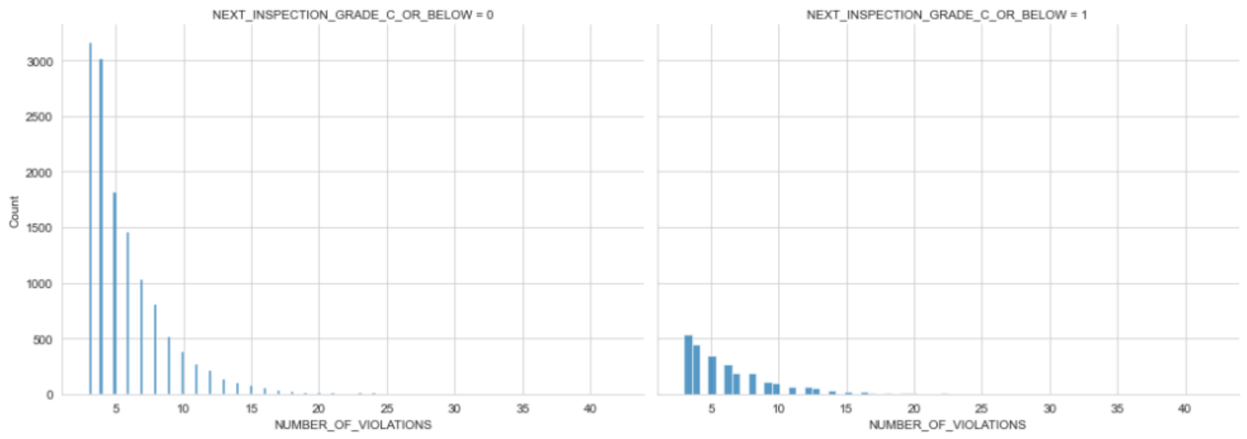
2.3.4 'MEDIAN_EMPLOYEE_TENURE'



2.3.5 'FIRST_VIOLATION'



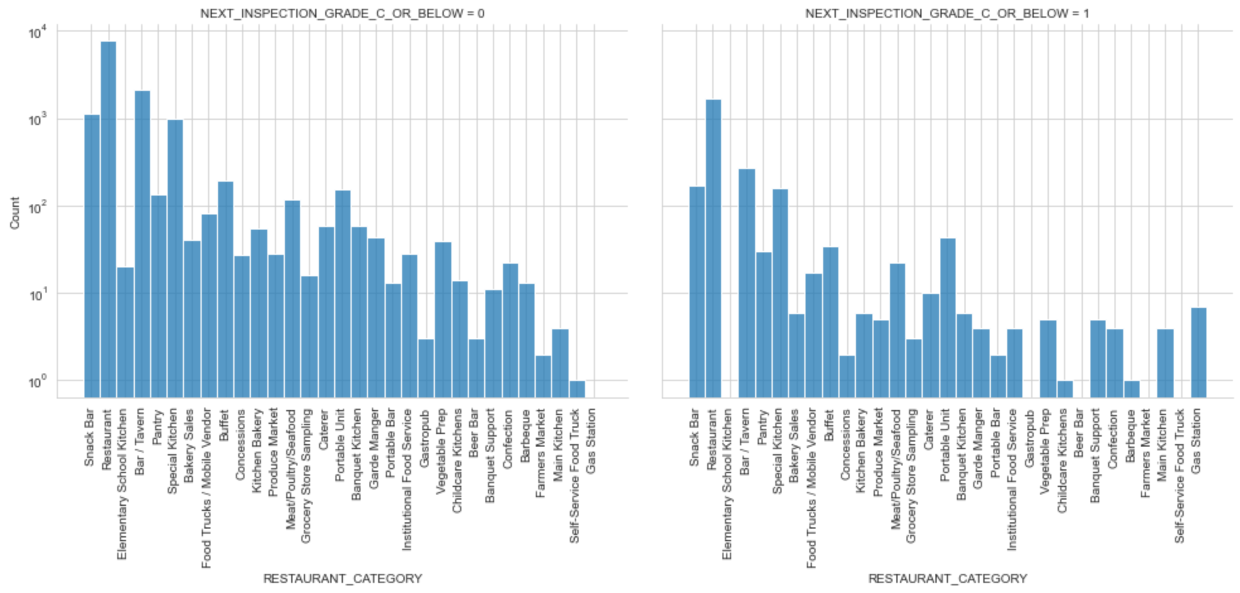
2.3.6 'NUMBER_OF_VIOLATIONS'



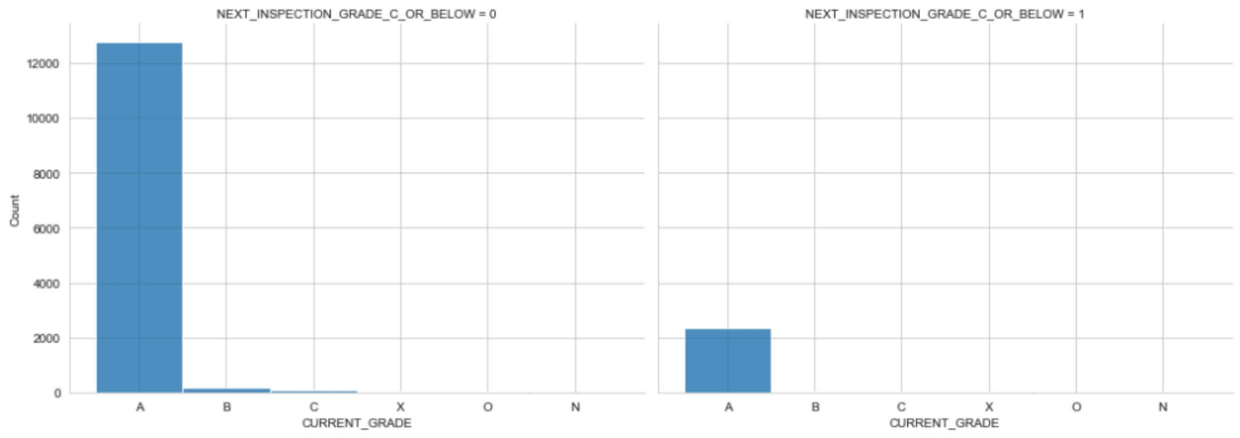
2.4 Categorical Columns

All the categorical features are inspected and cleaned. Here, the key categorical features are visualized in terms of the target variable.

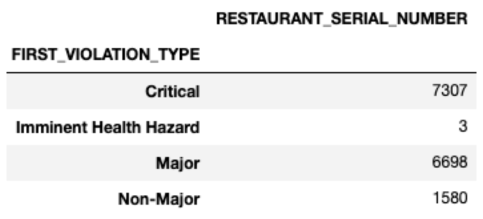
2.4.1 'RESTAURANT_CATEGORY'



2.4.2 'CURRENT_GRADE'

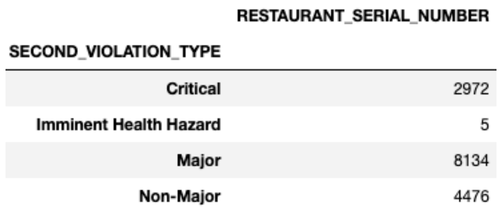


2.4.3 'FIRST_VIOLATION_TYPE'



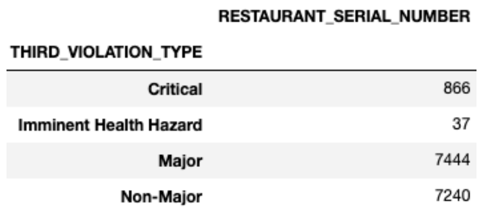
	RESTAURANT_SERIAL_NUMBER
FIRST_VIOLATION_TYPE	
Critical	7307
Imminent Health Hazard	3
Major	6698
Non-Major	1580

2.4.4 'SECOND_VIOLATION_TYPE'



SECOND_VIOLATION_TYPE	RESTAURANT_SERIAL_NUMBER
Critical	2972
Imminent Health Hazard	5
Major	8134
Non-Major	4476

2.4.5 'THIRD_VIOLATION_TYPE'



THIRD_VIOLATION_TYPE	RESTAURANT_SERIAL_NUMBER
Critical	866
Imminent Health Hazard	37
Major	7444
Non-Major	7240

3. Featuring Engineering

17 features are selected to train the model, which are:

continuous_features = ['LAT', 'LONG', 'CURRENT_DEMERITS', 'EMPLOYEE_COUNT', 'MEDIAN_EMPLOYEE_AGE', 'MEDIAN_EMPLOYEE_TENURE', 'INSPECTION_DEMERITS', 'FIRST_VIOLATION', 'SECOND_VIOLATION', 'THIRD_VIOLATION', 'NUMBER_OF_VIOLATIONS']

categorical_features = ['CURRENT_GRADE','RESTAURANT_CATEGORY', 'INSPECTION_TYPE', 'FIRST_VIOLATION_TYPE', 'SECOND_VIOLATION_TYPE', 'THIRD_VIOLATION_TYPE']

The target feature is ['NEXT_INSPECTION_GRADE_C_OR_BELOW']

Before proceeding to the modeling, featuring engineering is need for selecting, manipulating, and transforming raw data into features that can be used in supervised learning. Machine learning models require all input and output variables to be numeric. Therefore, for categorical data, we must encode it to numbers before we can fit and evaluate a model. In this project, the number of categories is quite large so that OrdinalEncoder can be more effective. After applying OrdinalEncoder to categorical feature:

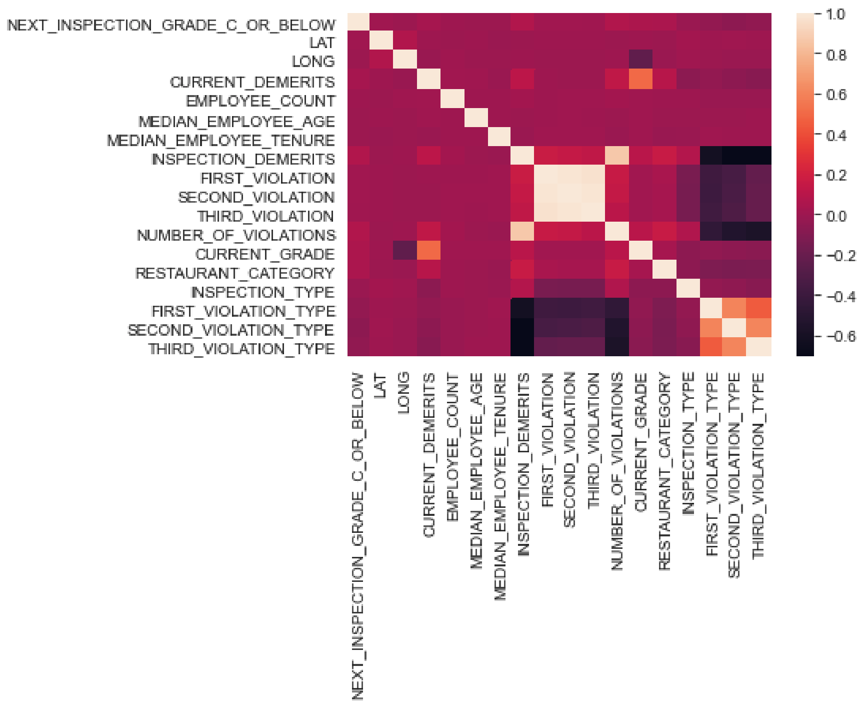
```
OrdinalEncoder().fit(Features_cate).categories_
```

```
[array(['A', 'B', 'C', 'N', 'O', 'X'], dtype=object),
 array(['Bakery Sales', 'Banquet Kitchen', 'Banquet Support',
        'Bar / Tavern', 'Barbeque', 'Beer Bar', 'Buffet', 'Caterer',
        'Childcare Kitchens', 'Concessions', 'Confection',
        'Elementary School Kitchen', 'Farmers Market',
        'Food Trucks / Mobile Vendor', 'Garde Manger', 'Gas Station',
        'Gastropub', 'Grocery Store Sampling',
        'Institutional Food Service', 'Kitchen Bakery', 'Main Kitchen',
        'Meat/Poultry/Seafood', 'Pantry', 'Portable Bar', 'Portable Unit',
        'Produce Market', 'Restaurant', 'Self-Service Food Truck',
        'Snack Bar', 'Special Kitchen', 'Vegetable Prep'], dtype=object),
 array(['Re-inspection', 'Routine Inspection'], dtype=object),
 array(['Critical', 'Imminent Health Hazard', 'Major', 'Non-Major'],
       dtype=object),
 array(['Critical', 'Imminent Health Hazard', 'Major', 'Non-Major'],
       dtype=object),
 array(['Critical', 'Imminent Health Hazard', 'Major', 'Non-Major'],
       dtype=object)]
```

```
Features_cate.iloc[:,:] = OrdinalEncoder().fit_transform(Features_cate.iloc[:,:])
Features_cate.head()
```

	CURRENT_GRADE	RESTAURANT_CATEGORY	INSPECTION_TYPE	FIRST_VIOLATION_TYPE	SECOND_VIOLATION_TYPE	THIRD_VIOLATION_TYPE
0	0.0	28.0	1.0	0.0	2.0	1.0
1	0.0	26.0	1.0	2.0	2.0	3.0
2	0.0	26.0	1.0	0.0	2.0	3.0
3	0.0	11.0	1.0	2.0	3.0	3.0
4	0.0	3.0	1.0	2.0	3.0	3.0

Investigating the correlation between class and each feature can help us to select the best features. The results show that there are both positive and negative correlations.



Machine learning algorithms like Linear Regression and Gaussian Naive Bayes assume the numerical variables have a Gaussian probability distribution. That's why we need data transformation. Standard Scaler doesn't have predetermined range, which is the first option for most cases.

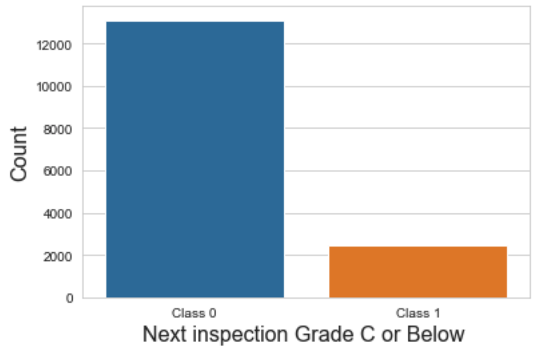
```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

Features_norm = Features.copy()
Features_norm.iloc[:,:] = scaler.fit_transform(Features.iloc[:,:])
```

```
Features_norm.head()
```

NUMBER_OF_VIOLATIONS	CURRENT_GRADE	RESTAURANT_CATEGORY	INSPECTION_TYPE	FIRST_VIOLATION_TYPE	SECOND_VIOLATION_TYPE
-0.575101	-0.138519	0.673370	0.242396	-1.029635	0.092611
-0.242259	-0.138519	0.452037	0.242396	0.740109	0.092611
-0.907943	-0.138519	0.452037	0.242396	-1.029635	0.092611
-0.907943	-0.138519	-1.207958	0.242396	0.740109	1.072608
-0.242259	-0.138519	-2.093288	0.242396	0.740109	1.072608

Before using this data in the model, we need to pay attention to the distribution of the class NEXT_INSPECTION_GRADE_C_OR_BELOW. Counting the number of zeros and ones, we can find that we have imbalanced data.



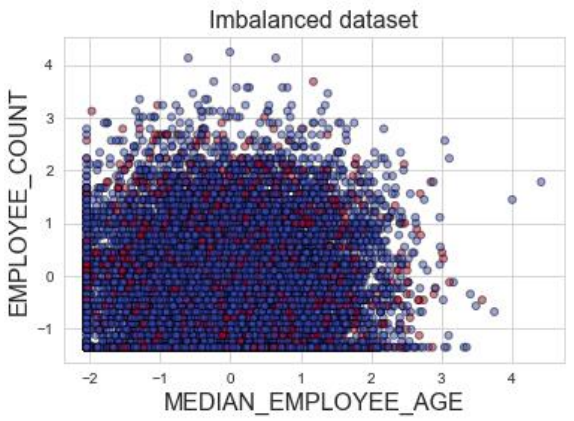
The dataset presents highly imbalanced. We can use Synthetic Minority Oversampling Technique (SMOTE) to resample the data

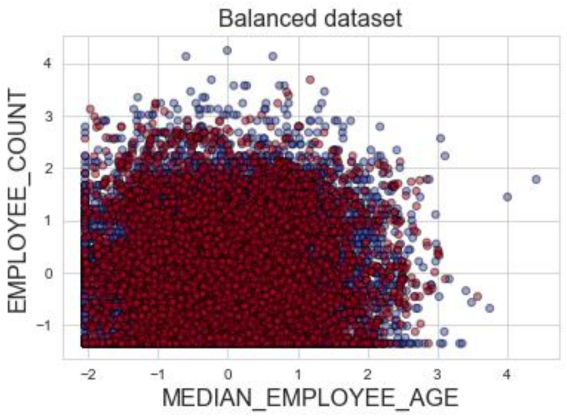
```
import imblearn
from numpy import mean
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(random_state=32)
X_smote, y_smote = smote.fit_resample(X_train, y_train)

print('Original dataset shape', Counter(y_train['NEXT_INSPECTION_GRADE_C_OR_BELOW']))
print('Resample dataset shape', Counter(y_smote['NEXT_INSPECTION_GRADE_C_OR_BELOW']))
```

```
Original dataset shape Counter({0: 10481, 1: 1988})
Resample dataset shape Counter({0: 10481, 1: 10481})
```

After applying SMOTE, as shown below, the dataset becomes balanced.

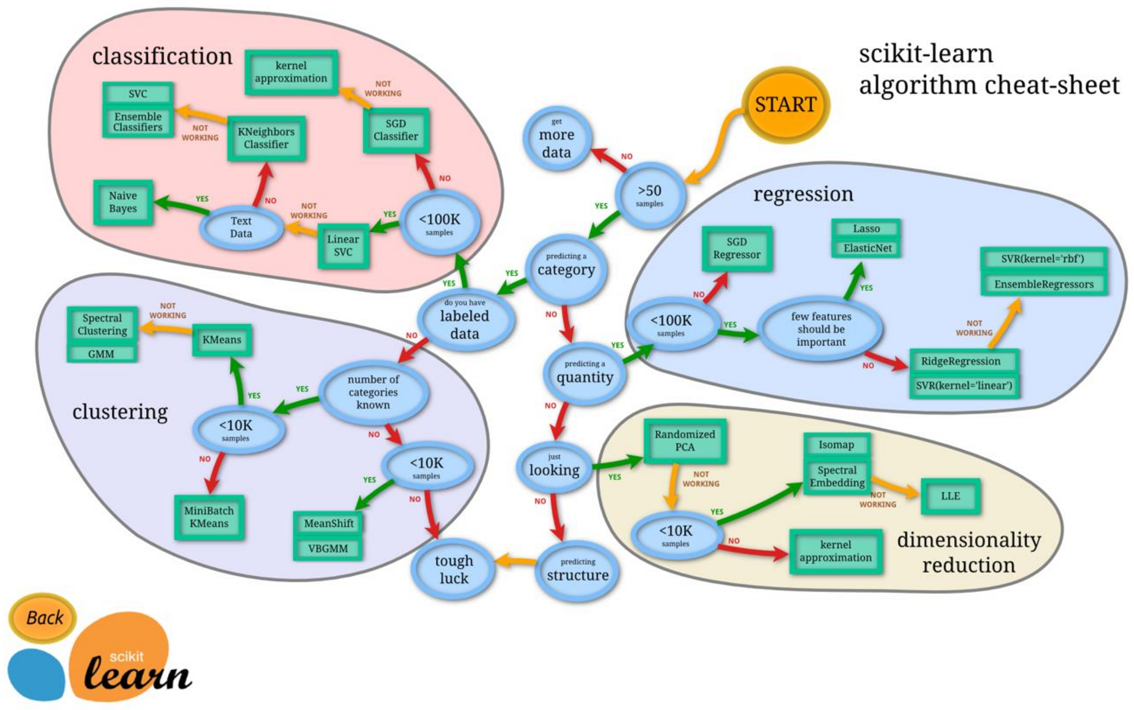




4. Modeling

Now we are ready to train a model and predict the required solution. As shown below, scikit-learn provides multiple predictive modeling algorithms to the claissification problem. The objective of this project is to identify the relationship between the target variable (Inspection grade below C or not) with other variables or features using supervised learning and classification models. The following 7 models are selected to be tested:

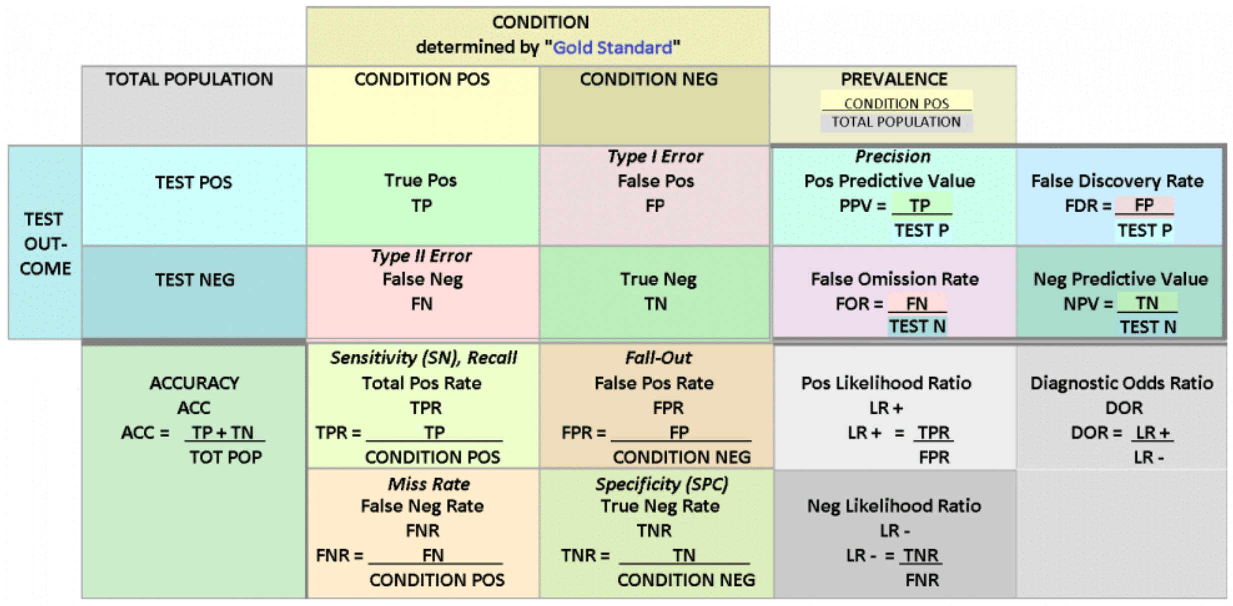
- Logistic Regression
- Decision Tree
- Random Forest
- KNN or k-Nearest Neighbors
- Naive Bayes classifier
- Gradient Boosting Classifier
- Support Vector Machines



Machine Learning Map (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

The entire dataset is split into train and test sets. To create the best model, we need to train multiple classifiers on the train set first and then examine the model on the test set. To evaluate their results, multiple performance evaluation metrics are considered:

- Accuracy: Percentage of correct prediction.
- Recall: True Positive / (True Positive + False Negative)
- Precision: True Positive / (True Positive + False Positive)
- F1-Score: The harmonic mean of precision (PRE) and recall (REC)



		CONDITION determined by "Gold Standard"			
	TOTAL POPULATION	CONDITION POS	CONDITION NEG	PREVALENCE $\frac{\text{CONDITION POS}}{\text{TOTAL POPULATION}}$	
TEST OUT-COME	TEST POS	True Pos TP	*Type I Error* False Pos FP	*Precision* Pos Predictive Value PPV = $\frac{\text{TP}}{\text{TEST P}}$	False Discovery Rate FDR = $\frac{\text{FP}}{\text{TEST P}}$
	TEST NEG	*Type II Error* False Neg FN	True Neg TN	False Omission Rate FOR = $\frac{\text{FN}}{\text{TEST N}}$	Neg Predictive Value NPV = $\frac{\text{TN}}{\text{TEST N}}$
	ACCURACY ACC ACC = $\frac{\text{TP + TN}}{\text{TOT POP}}$	*Sensitivity (SN), Recall* Total Pos Rate TPR TPR = $\frac{\text{TP}}{\text{CONDITION POS}}$	*Fall-Out* False Pos Rate FPR FPR = $\frac{\text{FP}}{\text{CONDITION NEG}}$	Pos Likelihood Ratio LR + LR + = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic Odds Ratio DOR DOR = $\frac{\text{LR +}}{\text{LR -}}$
		Miss Rate False Neg Rate FNR FNR = $\frac{\text{FN}}{\text{CONDITION POS}}$	*Specificity (SPC)* True Neg Rate TNR TNR = $\frac{\text{TN}}{\text{CONDITION NEG}}$	Neg Likelihood Ratio LR - LR - = $\frac{\text{TNR}}{\text{FNR}}$	

Confusiong matrix (https://www.unite.ai/what-is-a-confusion-matrix/)

A hypertable is established to track the model performance, which contains the following parameters and hyperparameters, as shown in the table below.

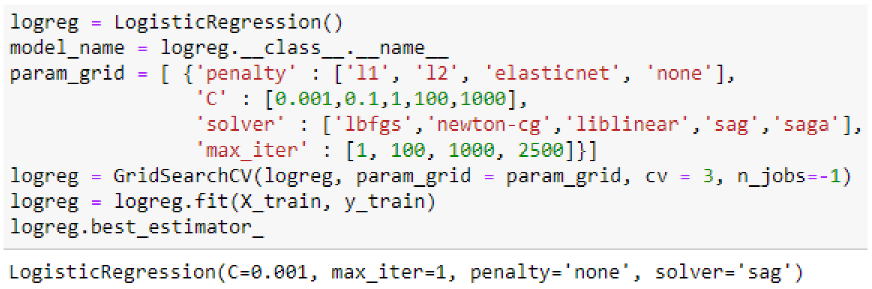
Index	model	f1_train	f1_val	accuracy_score_train	accuracy_score_val	best_estimator

The techniques of K-Fold cross-validation and GridSearch are applied for over/under-fitting and hyperparameter tuning.

4.1 Logistic Regression

Logistic Regression is a classification technique used in machine learning. It uses a logistic function to model the dependent variable. The dependent variable is dichotomous in nature, i.e. there could only be two possible classes (eg.: either the cancer is malignant or not). As a result, this technique is used while dealing with binary data. Logistic Regression has some advantages: 1) Logistic regression is easier to implement, interpret, and very efficient to train. It is very fast at classifying unknown records; 2) It performs well when the dataset is linearly separable; 3) It can interpret model coefficients as indicators of feature importance.

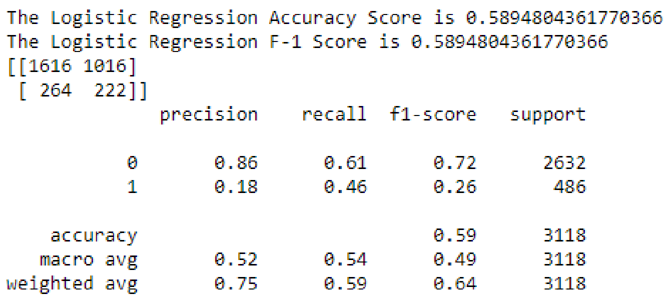
The hyperparameters used in logistic regression are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
logreg = LogisticRegression()
model_name = logreg.__class__.__name__
param_grid = [ {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
                'C' : [0.001,0.1,1,100,1000],
                'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
                'max_iter' : [1, 100, 1000, 2500]}]
logreg = GridSearchCV(logreg, param_grid = param_grid, cv = 3, n_jobs=-1)
logreg = logreg.fit(X_train, y_train)
logreg.best_estimator_
```

```
LogisticRegression(C=0.001, max_iter=1, penalty='none', solver='sag')
```

The fitting accuracy, F-1 score, and confusion matrix for logistic regression are shown as follows:

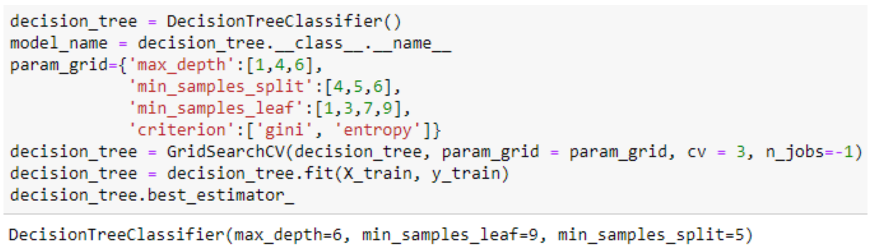


```
The Logistic Regression Accuracy Score is 0.5894804361770366
The Logistic Regression F-1 Score is 0.5894804361770366
[[1616 1016]
 [ 264  222]]
              precision    recall  f1-score   support

           0       0.86      0.61      0.72      2632
           1       0.18      0.46      0.26       486

    accuracy                           0.59      3118
   macro avg       0.52      0.54      0.49      3118
weighted avg       0.75      0.59      0.64      3118
```

4.2 Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. A Decision Tree Classifier functions by breaking down a dataset into smaller and smaller subsets based on different criteria. Different sorting criteria will be used to divide the dataset, with the number of examples getting smaller with every division.

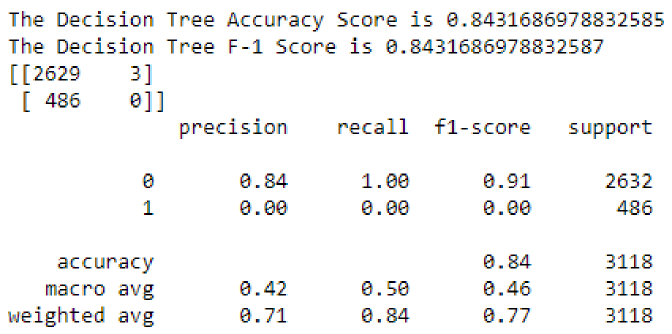
The hyperparameters used in decision tree are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
decision_tree = DecisionTreeClassifier()
model_name = decision_tree.__class__.__name__
param_grid={'max_depth':[1,4,6],
            'min_samples_split':[4,5,6],
            'min_samples_leaf':[1,3,7,9],
            'criterion':['gini', 'entropy']}
decision_tree = GridSearchCV(decision_tree, param_grid = param_grid, cv = 3, n_jobs=-1)
decision_tree = decision_tree.fit(X_train, y_train)
decision_tree.best_estimator_
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=9, min_samples_split=5)
```

The fitting accuracy, F-1 score, and confusion matrix for decision tree are shown as follows:

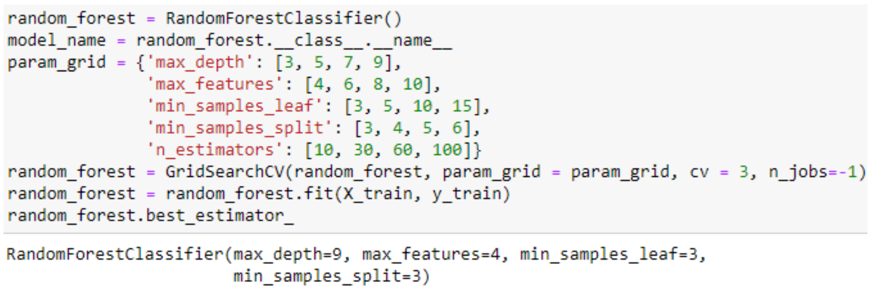


```
The Decision Tree Accuracy Score is 0.8431686978832585
The Decision Tree F-1 Score is 0.8431686978832587
[[2629    3]
 [ 486    0]]
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      2632
           1       0.00      0.00      0.00       486

    accuracy                           0.84      3118
   macro avg       0.42      0.50      0.46      3118
weighted avg       0.71      0.84      0.77      3118
```

4.3 Random Forest

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

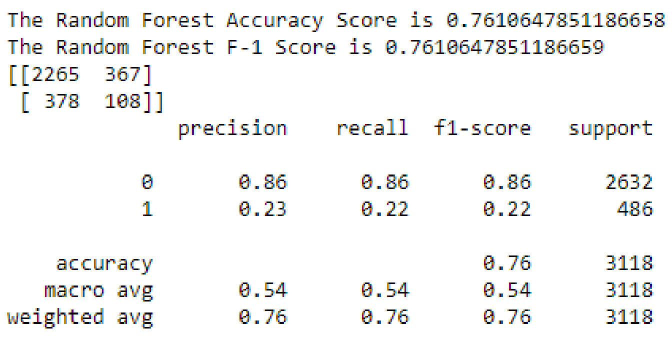
The hyperparameters used in random forest are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
random_forest = RandomForestClassifier()
model_name = random_forest.__class__.__name__
param_grid = {'max_depth': [3, 5, 7, 9],
              'max_features': [4, 6, 8, 10],
              'min_samples_leaf': [3, 5, 10, 15],
              'min_samples_split': [3, 4, 5, 6],
              'n_estimators': [10, 30, 60, 100]}
random_forest = GridSearchCV(random_forest, param_grid = param_grid, cv = 3, n_jobs=-1)
random_forest = random_forest.fit(X_train, y_train)
random_forest.best_estimator_
```

```
RandomForestClassifier(max_depth=9, max_features=4, min_samples_leaf=3,
                       min_samples_split=3)
```

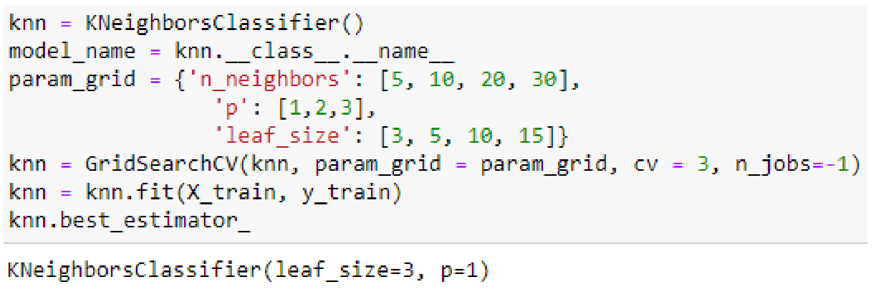
The fitting accuracy, F-1 score, and confusion matrix for random forest are shown as follows:



```
The Random Forest Accuracy Score is 0.7610647851186658
The Random Forest F-1 Score is 0.7610647851186659
[[2265  367]
 [ 378  108]]
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      2632
           1       0.23      0.22      0.22       486

    accuracy                           0.76      3118
   macro avg       0.54      0.54      0.54      3118
weighted avg       0.76      0.76      0.76      3118
```

4.4 K Nearest Neighbours

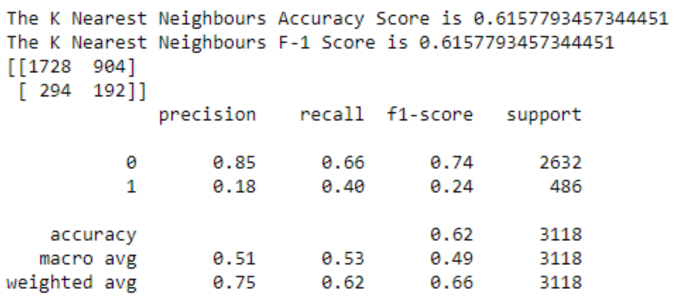
The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. To select the right K for the data, the hyperparameters used in KNN are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
knn = KNeighborsClassifier()
model_name = knn.__class__.__name__
param_grid = {'n_neighbors': [5, 10, 20, 30],
                'p': [1,2,3],
                'leaf_size': [3, 5, 10, 15]}
knn = GridSearchCV(knn, param_grid = param_grid, cv = 3, n_jobs=-1)
knn = knn.fit(X_train, y_train)
knn.best_estimator_
```

```
KNeighborsClassifier(leaf_size=3, p=1)
```

The fitting accuracy, F-1 score, and confusion matrix for KNN are shown as follows:

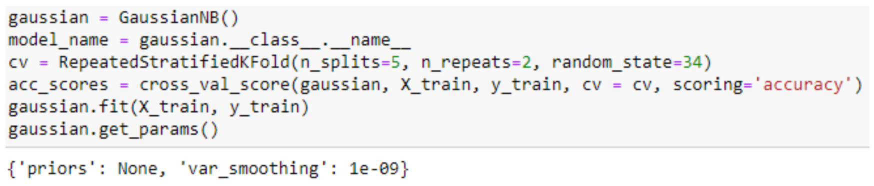


```
The K Nearest Neighbours Accuracy Score is 0.6157793457344451
The K Nearest Neighbours F-1 Score is 0.6157793457344451
[[1728  904]
 [ 294  192]]
              precision    recall  f1-score   support

           0       0.85      0.66      0.74      2632
           1       0.18      0.40      0.24       486

    accuracy                           0.62      3118
   macro avg       0.51      0.53      0.49      3118
weighted avg       0.75      0.62      0.66      3118
```

4.5 Naive Bayes

Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).

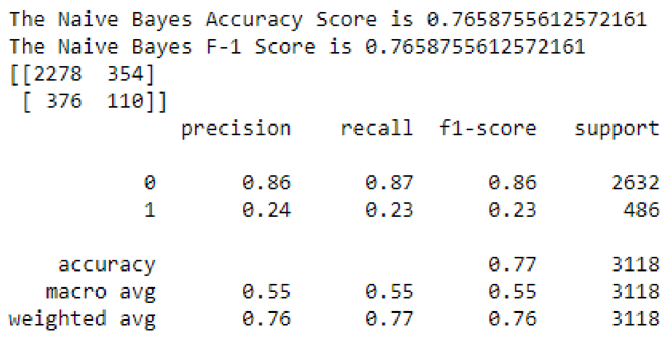
The hyperparameters used in Naive Bayes are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
gaussian = GaussianNB()
model_name = gaussian.__class__.__name__
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=34)
acc_scores = cross_val_score(gaussian, X_train, y_train, cv = cv, scoring='accuracy')
gaussian.fit(X_train, y_train)
gaussian.get_params()
```

```
{'priors': None, 'var_smoothing': 1e-09}
```

The fitting accuracy, F-1 score, and confusion matrix for Naive Bayes are shown as follows:

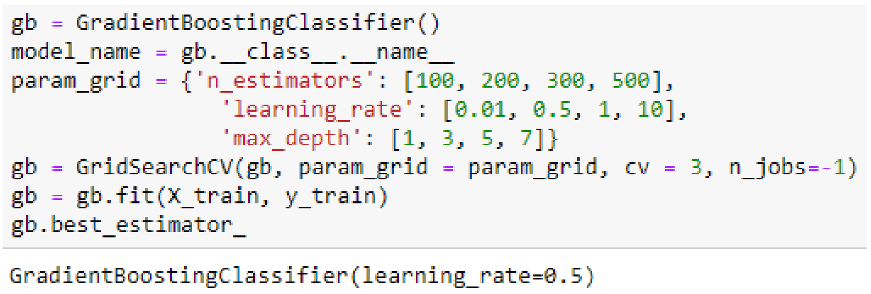


```
The Naive Bayes Accuracy Score is 0.7658755612572161
The Naive Bayes F-1 Score is 0.7658755612572161
[[2278  354]
 [ 376  110]]
              precision    recall  f1-score   support

           0       0.86      0.87      0.86      2632
           1       0.24      0.23      0.23       486

    accuracy                           0.77      3118
   macro avg       0.55      0.55      0.55      3118
weighted avg       0.76      0.77      0.76      3118
```

4.6 Gradient Boosting Classifier

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

The hyperparameters used in gradient boosting are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.



```
gb = GradientBoostingClassifier()
model_name = gb.__class__.__name__
param_grid = {'n_estimators': [100, 200, 300, 500],
              'learning_rate': [0.01, 0.5, 1, 10],
              'max_depth': [1, 3, 5, 7]}
gb = GridSearchCV(gb, param_grid = param_grid, cv = 3, n_jobs=-1)
gb = gb.fit(X_train, y_train)
gb.best_estimator_
```

```
GradientBoostingClassifier(learning_rate=0.5)
```

The fitting accuracy, F-1 score, and confusion matrix for gradient boosting classifier are shown as follows:

```
The Gradient Boosting Accuracy Score is 0.8277742142398974
The Gradient Boosting F-1 Score is 0.8277742142398974
[[2561   71]
 [ 466   20]]
              precision    recall  f1-score   support

           0       0.85      0.97      0.91      2632
           1       0.22      0.04      0.07       486

    accuracy                           0.83      3118
   macro avg       0.53      0.51      0.49      3118
weighted avg       0.75      0.83      0.77      3118
```

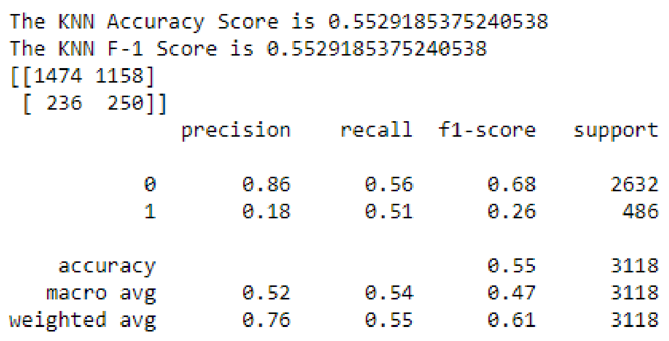
4.7 Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. The advantages of support vector machines are: 1) Effective in high dimensional spaces; 2) Still effective in cases where number of dimensions is greater than the number of samples; 3) Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

The hyperparameters used in support vector machine are tested in the following ranges/values. The obtained best estimator is also presented in the following figure.

```
svc = SVC()
model_name = svc.__class__.__name__
param_grid = [{'kernel': ['rbf'], 'gamma': [1e-2, 1e-3, 1e-4],
                     'C': [0.01,0.1,10]},
              {'kernel': ['sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4],
                     'C': [0.01,0.1,10]},
              {'kernel': ['linear'], 'C': [0.01,0.1,10]}]
svc = GridSearchCV(svc, param_grid = param_grid, cv = 3, n_jobs=-1)
svc = svc.fit(X_train, y_train)
svc.best_estimator_

SVC(C=10, gamma=0.01)
```

The fitting accuracy, F-1 score, and confusion matrix for gradient boosting classifier are shown as follows:



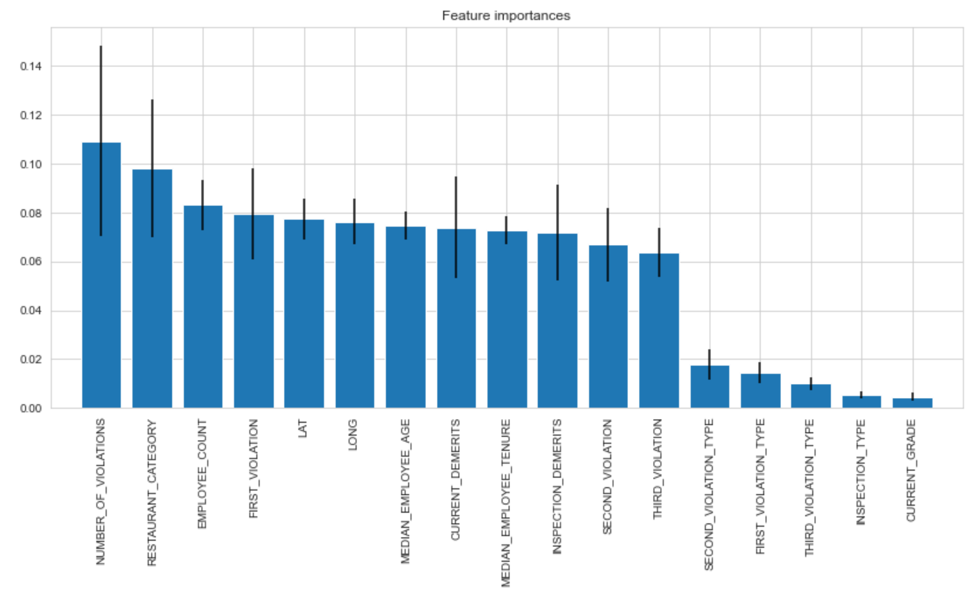
```
The KNN Accuracy Score is 0.5529185375240538
The KNN F-1 Score is 0.5529185375240538
[[1474 1158]
 [ 236  250]]
              precision    recall  f1-score   support

           0       0.86      0.56      0.68      2632
           1       0.18      0.51      0.26       486

    accuracy                           0.55      3118
   macro avg       0.52      0.54      0.47      3118
weighted avg       0.76      0.55      0.61      3118
```

4.8 Best classifier

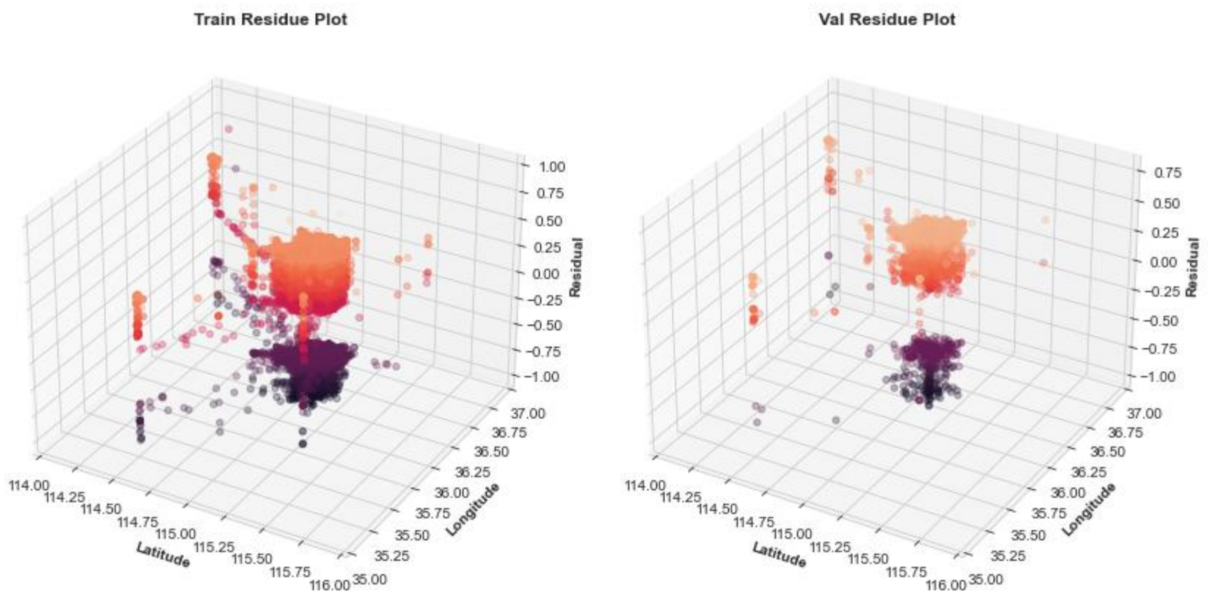
The results of each model are concluded and shown in the following table, which is ordered by the F-1 score of the test set. As shown in the table, the Decision Tree Classifier obtained the best match.

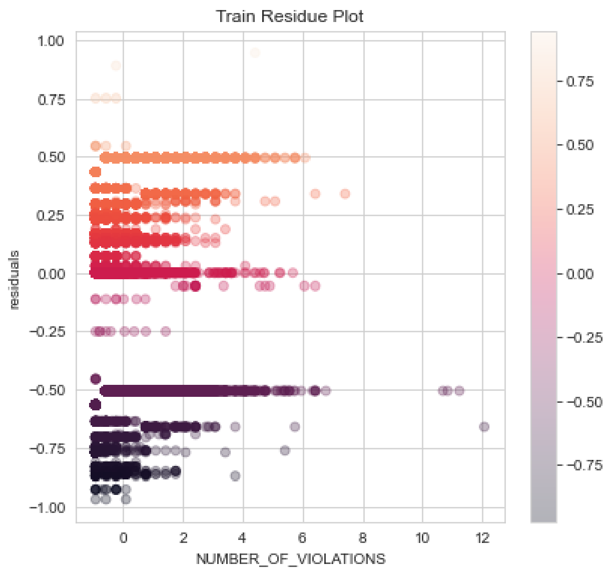
model	f1 train	f1 val	Accuracy score train	Accuracy score val	best_estimator
Decision Tree Classifier	0.642973	0.843169	0.642973	0.843169	DecisionTreeClassifier(max_depth=6, min_samples_leaf=9, min_samples_split=5)
Gradient Boosting Classifier	0.905686	0.827774	0.905686	0.827774	GradientBoostingClassifier(learning_rate=0.5)
Gaussian NB	0.529243	0.765876	0.529243	0.765876	{'priors': None, 'var_smoothing': 1e-09}
Random Forest Classifier	0.826639	0.761065	0.826639	0.761065	RandomForestClassifier(max_depth=9, max_features=4, min_samples_leaf=3, min_samples_split=3)
K Neighbors Classifier	0.872913	0.615779	0.872913	0.615779	KNeighborsClassifier(leaf_size=3, p=1)
Logistic Regression	0.544032	0.58948	0.544032	0.58948	LogisticRegression(C=0.001, max_iter=1, penalty='none', solver='sag')
SVC	0.594791	0.552919	0.594791	0.552919	SVC(C=10, gamma=0.01)

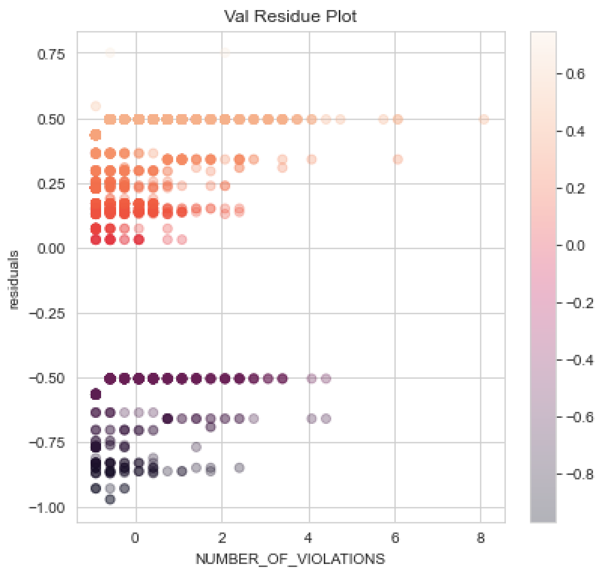
Random forest is able to provide feature importance. As shown in the figure below, the 'NUMBER_OF_VIOLATIONS' is the most important feature.



The residuals of the probabilities ('NEXT_INSPECTION_GRADE_C_OR_BELOW' = 1) predicted by the best model, Decision Tree, are visualized in terms of the location and 'NUMBER_OF_VIOLATIONS', as shown below. The residual plot shows a random pattern, indicating a good fit for the classification model.







5. Future Works and Recommendations

This project investigated the data sets of restaurant inspections in Las Vegas and then built classification models to predict the restaurant's next inspection below or above C. Through the project, the model development cycle goes through various important stages, including data collection, data cleaning, data analysis, featuring engineering and model building. Based on the provided dataset, information, and results, Decision Tree is the optimum model for this dataset with hyperparameters of max_depth = 6, min_samples_leaf = 9, and min_samples_split = 5.

For future work to achieve a better prediction, adding more data for class 1 to avoid imbalanced datasets is needed. In addition, feature importance results show that features like number of violations, restaurant category, employee count, first violation, and locations are important to predict the next inspection. Therefore, implementing further featuring engineering to create new features out of existing ones can improve the accuracy of the model.