Computational Social Science Project #3

Enter your Name: Sofia Guo

Collaboration with: Marisa Tsai, Stacy Chen, Reiley Reed, Neena Albarus

Semester: Fall 2023

1. Introduction

Load data

In [1]:
```
#
# load libraries
# ------------
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_validate, cross_
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score, recall_score, preci
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier


# There are a few warnings that will appear that will not affect your analys
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Make sure to import other libraries that will be necessary for training mo
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(
R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Ke
rnel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(
R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(
R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Ke
rnel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(
R) AVX) instructions.
```

In [2]:
```
#
# read in "Inspections Data 2011-2013" csv data
# -----------
chicago_inspections_2011_to_2013 = pd.read_csv("data/Chicago Inspections 201
                                               low_memory=False)

#
# read in  "Inspections Data 2014_updated" csv data
# -----------
chicago_inspections_2014 = pd.read_csv("data/Chicago Inspections 2014_update
                                       low_memory=False)
```

In [3]:
```
# look at the inspections data
chicago_inspections_2011_to_2013.head()
```

Out[3]:

	Inspection_ID	Inspection_Date	DBA_Name	AKA_Name	License	Facility_Type	I
0	269961	2013-01-31	SEVEN STAR	SEVEN STAR	30790	Grocery Store	Ri (L
1	507211	2011-10-18	PANERA BREAD	PANERA BREAD	1475890	Restaurant	R (H
2	507212	2011-10-18	LITTLE QUIAPO RESTAURANT	LITTLE QUIAPO RESTAURANT	1740130	Restaurant	R (H
3	507216	2011-10-19	SERGIO'S TAQUERIA PIZZA INC.	SERGIO'S TAQUERIA PIZZA	1447363	Restaurant	R (H
4	507219	2011-10-20	TARGET STORE # T-2079	TARGET	1679459	Restaurant	Ri (Med

In [4]:
```
# drop column names related to geography, identification, and pass/fail flag
chicago_inspections_2011_to_2013.drop(columns = ['AKA_Name',
                                                 'License',
                                                 'Address',
                                                 'City',
                                                 'State',
                                                 'Zip',
                                                 'Latitude',
                                                 'Longitude',
                                                 'Location',
                                                 'ID',
                                                 'LICENSE_ID',
                                                 'LICENSE_TERM_START_DATE',
                                                 'LICENSE_TERM_EXPIRATION_DA
                                                 'LICENSE_STATUS',
                                                 'ACCOUNT_NUMBER',
                                                 'LEGAL_NAME',
                                                 'DOING_BUSINESS_AS_NAME',
                                                 'ADDRESS',
                                                 'CITY',
                                                 'STATE',
                                                 'ZIP_CODE',
                                                 'WARD',
                                                 'PRECINCT',
                                                 'LICENSE_CODE',
                                                 'BUSINESS_ACTIVITY_ID',
                                                 'BUSINESS_ACTIVITY',
                                                 'LICENSE_NUMBER',
                                                 'LATITUDE',
                                                 'LONGITUDE',
                                                 'pass_flag',
                                                 'fail_flag'],
                                      inplace = True)

# set index
chicago_inspections_2011_to_2013.set_index(['Inspection_ID', 'DBA_Name'], in
```

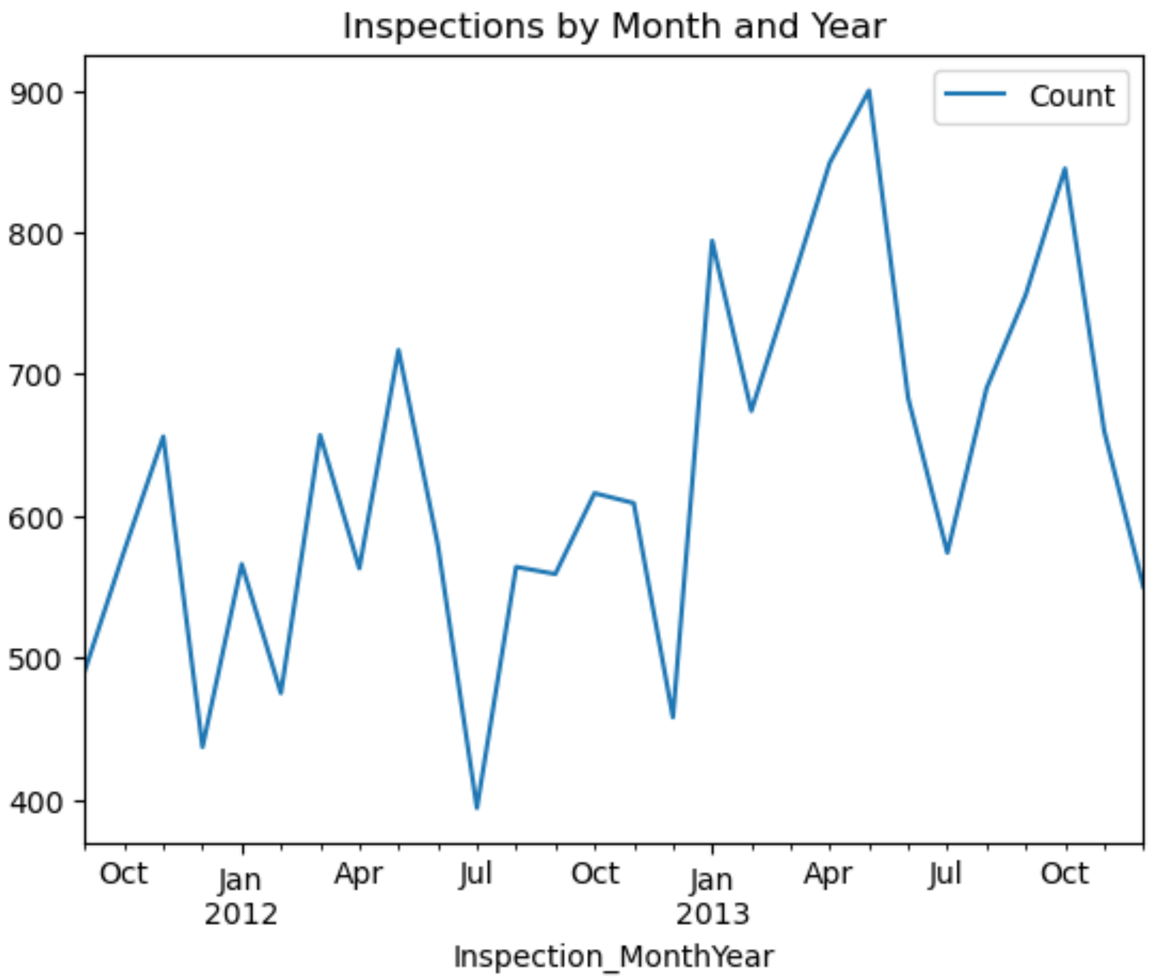
In [5]:
```
# convert the inspection date to a datetime format
chicago_inspections_2011_to_2013['Inspection_Date'] = pd.to_datetime(chicago
```

Visualization

Let's visualize what inspections look like over time.

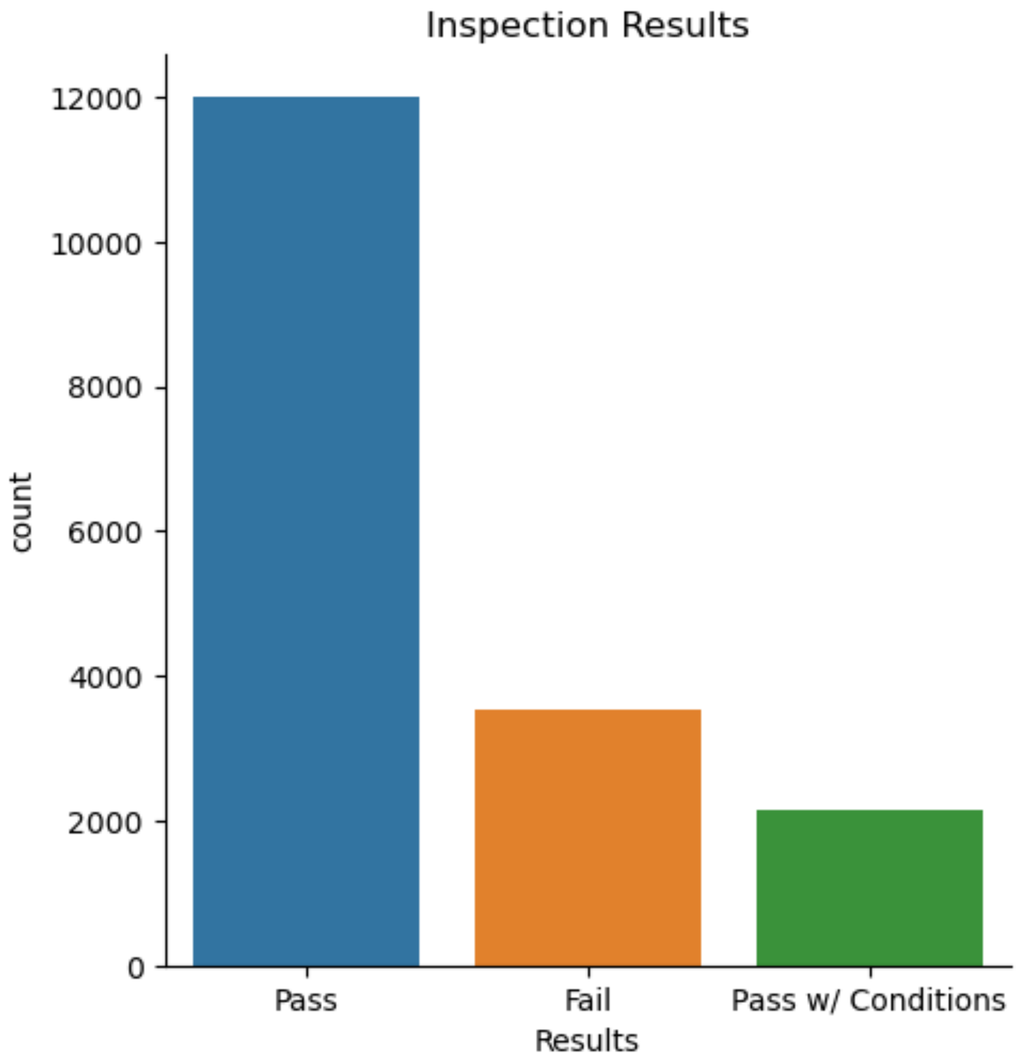
In [6]:
```
# visualize inspections over time
# -----------
chicago_inspections_2011_to_2013['Inspection_MonthYear'] = chicago_inspectio
counts_by_day = chicago_inspections_2011_to_2013.groupby('Inspection_MonthYe
counts_by_day.set_index(["Inspection_MonthYear"], inplace = True)
counts_by_day.plot(title = "Inspections by Month and Year")
```

Out[6]:
```
<Axes: title={'center': 'Inspections by Month and Year'}, xlabel='Inspection
_MonthYear'>
```



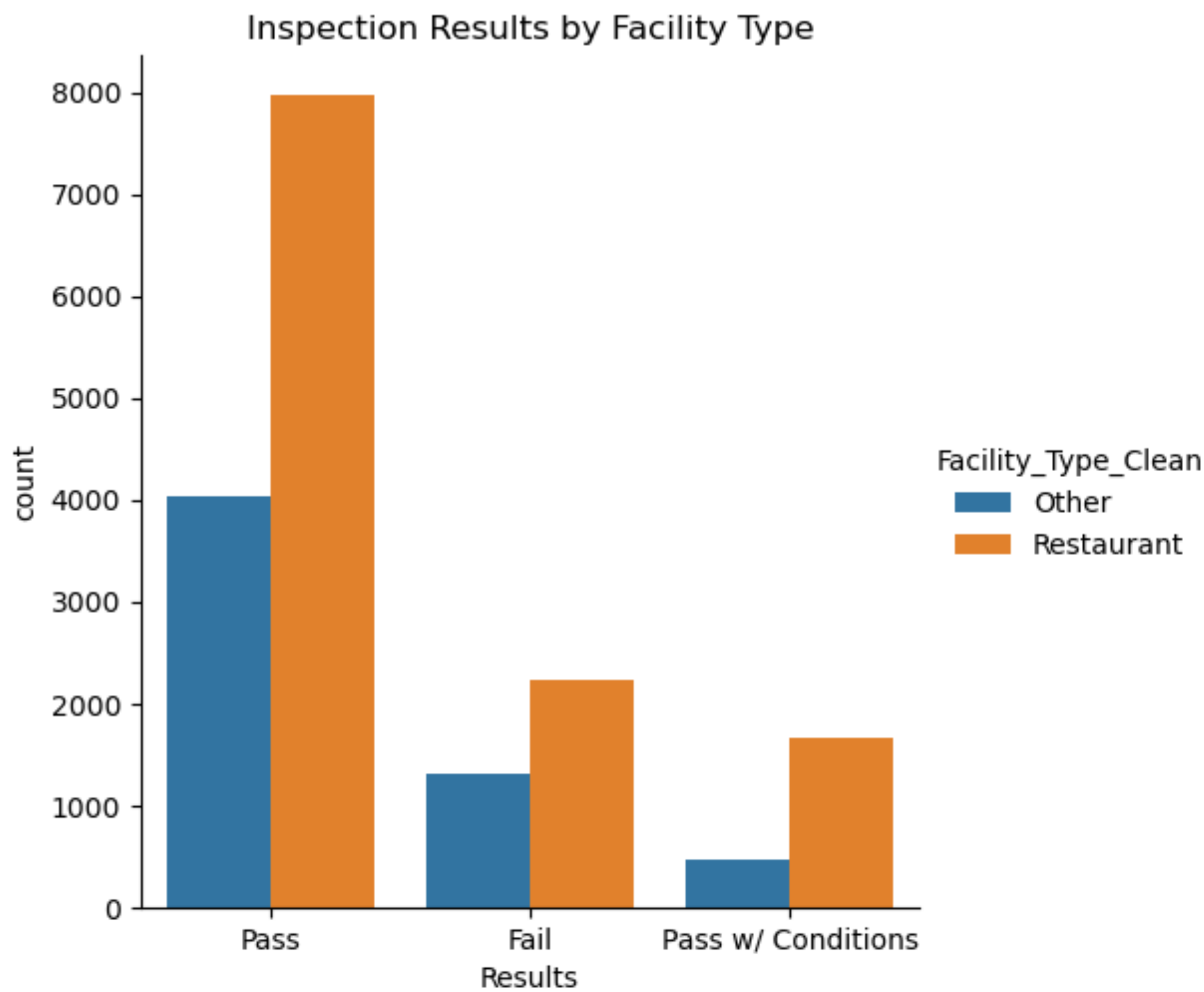
Let's visualize what the distribution of results looks like.

In [7]:
```
# view inspection results
# -----------
sns.catplot(data = chicago_inspections_2011_to_2013,
            x = "Results",
            kind = "count")

plt.title("Inspection Results")
plt.show()
```



What if we separate results by facility type?

In [8]:
```
# view inspection results by facility type (restaurant or not)
# -----------
sns.catplot(data = chicago_inspections_2011_to_2013,
            x = "Results",
            kind = "count",
            hue = 'Facility_Type_Clean')

plt.title("Inspection Results by Facility Type")
plt.show();
```



2. Data Preprocessing and Cleaning

In [9]:
```
# drop datetime info
# -----------
chicago_inspections_2011_to_2013 = chicago_inspections_2011_to_2013.dropna()
```

In [10]:
```
# process target
# -----------
y = chicago_inspections_2011_to_2013['Results']

# decide if you want to binarize the outcome variable
# -----------
# comment out the following lines of code if you don't want to binarize the
y = y.replace({'Pass w/ Conditions': 'Pass'})
lb_style = LabelBinarizer()
y = lb_style.fit_transform(y)

# recode 0s and 1s so 1s are "Fail"
y = np.where(y == 1, 0 ,1)


# process features
# -----------

# create feature dataset
X = chicago_inspections_2011_to_2013.drop(columns = ['Results',






# get dummies
X = pd.get_dummies(X)
```

In [11]:
```
# view feature datset
X.head()
```

Out[11]:

		criticalCount	seriousCount	minorCount	pastFail	pastCritical
Inspection_ID	**DBA_Name**					
269961	**SEVEN STAR**	0	0	2	0	0
507211	**PANERA BREAD**	0	0	3	0	0
507212	**LITTLE QUIAPO RESTAURANT**	0	2	6	0	0
507216	**SERGIO'S TAQUERIA PIZZA INC.**	0	0	6	0	0
507219	**TARGET STORE # T-2079**	0	2	6	0	0

In [12]:

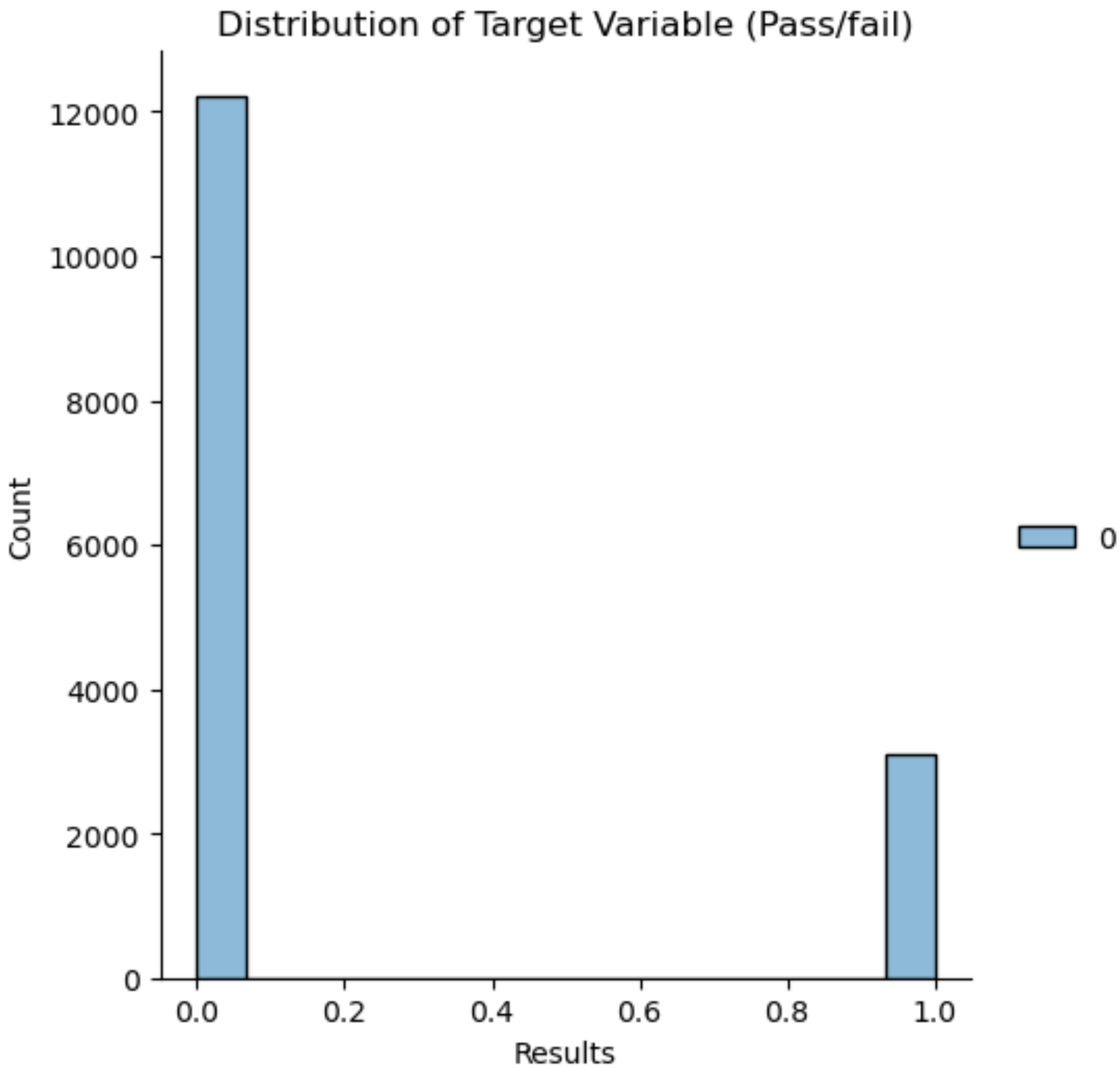
```
#proportion of fails
y.sum()/len(y)
```

Out[12]:

```
0.20161975050617204
```

In [13]:

```
# distribution plot of the outcome variable
sns.displot(y)    # notice the default is a histogram
plt.title("Distribution of Target Variable (Pass/fail)")
plt.xlabel('Results')
plt.ylabel('Count')
plt.show()
```



There is an imbalance in our data where there are around 80% passes and 20% fails, which could be a problem because a model would over predict the majority class more often.

In [14]:
```
#split our data into test/train
# Set seed
np.random.seed(10)

# training and test split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    train_size = .80,
                                                    test_size=0.20,
                                                    stratify=y)

# training and validation split
X_train, X_validate, y_train, y_validate = train_test_split(X_train,
                                                            y_train,
                                                            train_size = .75
                                                            test_size = .25,
                                                            stratify = y_tra
```

3. Fit Models

Now choose 3 different machine learning techniques and apply them below. Choose from one of the algorithms we have used in lab (e.g., logistic regression, random forests, `AdaBoost()`, `xgboost()`, `VotingClassifer()`, or `BART`).

Detail the basic logic and assumptions underlying each model, its pros/cons, and why it is a plausible choice for this problem. Also, be sure to do the following:

1. Import the appropriate library from sklearn
2. Set up a hyperparameter grid (check out our previous labs to see how to do this)
3. Find the best hyperparameters, and then fit your model (using either train/validation splits or cross-validation)

Model 1: Logistic regression

Logic: Logistic regression is used for binary classification tasks, predicting outcomes between two classes, which in this case are whether a site passes or fails a food inspection. It models the relationship between input features and the probability of a specific outcome. This method assumes that the data are relatively balanced between the two classes, so that the regression can learn from enough variation in the response variable (y = pass/fail).

Pros: Good for predicting binary responses coded 0 or 1, so it is suitable for this use.

Cons: May not be able to learn effectively from lack of balanced data, such that it could predict mostly zeroes with incorrect weights but still have a high accuracy score.

Reasoning: Given that our classification problem is between two outcomes (pass/fail), logistic regression could give a good baseline as a binary classifer which we can then compare to the performance of other classifers.

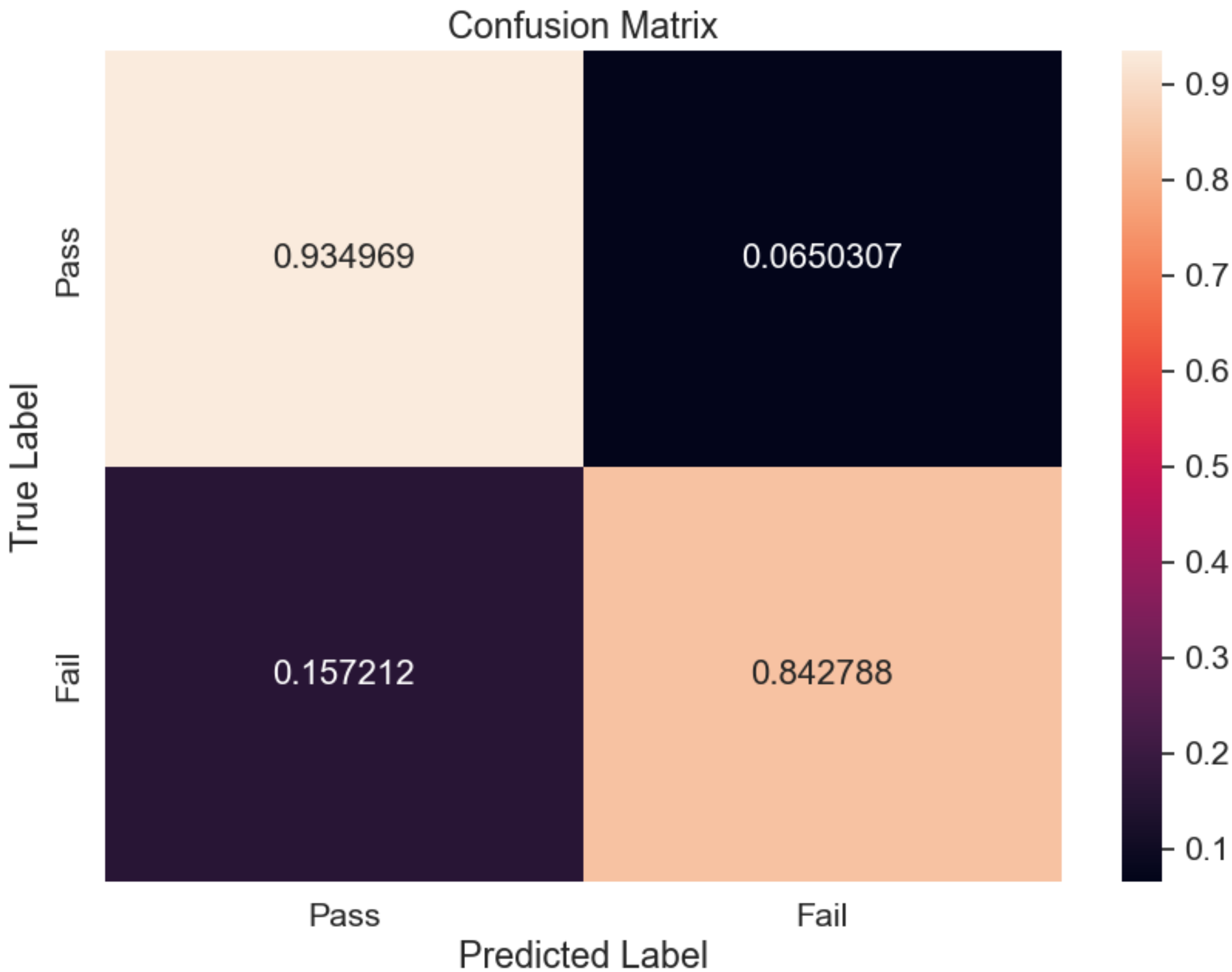
In [15]:
```
#Logistic regression code
# create a model
logit_reg = LogisticRegression()

# fit the model
logit_model = logit_reg.fit(X_train, y_train.ravel())

# predict on the validation data
y_pred = logit_model.predict(X_validate)
```

In [16]:
```
# extract the coefficents and create a dataframe for plotting
logit_data = pd.concat([pd.DataFrame(X.columns),
                        pd.DataFrame(np.transpose(logit_model.coef_))],
                       axis = 1)

logit_data.columns = ['Feature', 'Coefficient']
logit_data['abs_coef'] = abs(logit_data['Coefficient'])
```

In [17]:
```
# hyperparameter tuning
# ----------

# import libraries
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore')
from sklearn.metrics import accuracy_score

# set parameters
param_grid = {'penalty': ['l1', 'l2', 'elasticnet'],
             'C': np.arange(.1, 1, .1),
             'fit_intercept': [True, False],
             'solver': ['liblinear', 'saga']}


# execute the grid search and fit to training data
logit_grid = GridSearchCV(logit_model,
                          param_grid,
                          cv=2)
logit_grid.fit(X_train,
               y_train)

# choose best performing model
best_index = np.argmax(logit_grid.cv_results_["mean_test_score"])
best_logit_pred = logit_grid.best_estimator_.predict(X_validate)

# print results
print(logit_grid.cv_results_["params"][best_index])
print('Validation Accuracy', accuracy_score(best_logit_pred, y_validate))
```

```
{'C': 0.1, 'fit_intercept': True, 'penalty': 'elasticnet', 'solver': 'liblin
ear'}
Validation Accuracy 0.9163945133899413
```

In [18]:
```
# specify confusion matrix
cf_matrix = confusion_matrix(y_validate,
                             best_logit_pred,
                             normalize = "true")

# convert to dataframe
df_cm = pd.DataFrame(cf_matrix,
                     range(2),
                     range(2))

# label dataframe
df_cm = df_cm.rename(index=str, columns={0: "Pass", 1: "Fail"})
df_cm.index = ["Pass", "Fail"]

# plot
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm,
            annot=True,
            annot_kws={"size": 16},
            fmt='g')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Model 2: Random Forest

Logic: Random Forest is a type of decision tree method that constructs multiple "trees" of binary decisions that fit the response labels the best. Then, it uses a subset of features to grow each tree. The trees then make predictions, and the random forest takes a majority vote from the trees to determine the winner. Random forest is known as a "bagging" method. Bagging refers to bootstrap aggregating, where the multiple trees are created from random samples of the data with replacement. Random forest randomly assigns some features to use at each split during bagging so that overfitting is mitigated.

Pros: Can handle data that has more features than cases, is highly interpretable since the terminal nodes give the final label, and handles nonlinearities in the features; reduces overfitting by training on random samples with replacement

Cons: Risk of overfitting if there is not enough bootstrapping and high computational costs.

Reasoning: This is suitable for the classification task on hand because we have a lot of features to train on and want to know if RF can do better than a logistic regression.

In [19]:
```
# initialize a random forest classifier
# ----------
rf_classifier = RandomForestClassifier(
                        # specify parameters
                        n_estimators=100,              # specify the number c
                        criterion='gini',              # or you can use 'entr
                        max_depth=None,                # how deep tree nodes
                        min_samples_split=2,           # samples needed to sp
                        min_samples_leaf=1,            # samples needed for a
                        min_weight_fraction_leaf=0.0,  # weight of samples ne
                        max_features=None,             # number of features t
                        max_leaf_nodes=None,           # max nodes
                        min_impurity_decrease=1e-07,   # early stopping
                        random_state = 10)    # random seed

# Train the classifier using the training data
rf_model = rf_classifier.fit(X_train, y_train)
```

In [20]:
```
# specify cross-validation
# ----------
scores = cross_val_score(rf_classifier,
                         X_train,
                         y_train.ravel(), # Some algorithms will expect you
                         cv=5)
```

In [21]:
```
# calculate the average score across models
# ----------
scores.mean()
```

Out[21]:
```
0.9246684196154135
```

Model 3: Adaptive boosting (AdaBoost)

Logic: Boosting grows trees sequentially, using remaining prediction error (i.e. residuals) from prior tree; average predictions of resulting trees. AdaBoost increases the weight of cases that have higher error to improve prediction accuracy. It begins with a base model/tree and iteratively adjusts the weights of the incorrect classifications as the another model is trained on the previous model's outputs. Because the incorrect cases are weighted more, the new model prioritizes getting those correct on

Pros: Helps understand model accuracy differences across a hyperparameter; often better prediction than random forest.

Cons: Prone to overfitting because of the focus on higher error; not as efficient as using GridSearchCV

Reasoning: This classifier is supposed to perform better than random forest, and since we are testing RF as a potential model, we would like to see if this is in fact true.

In [22]:
```
# initialize an adaptive boosting classifer
# ----------
ada_classifier = AdaBoostClassifier(n_estimators=100)

# Train the classifier using the training data
ada_model = ada_classifier.fit(X_train, y_train)
```

In [23]:
```
# calculate accuracy using cross validation
# ----------
scores = cross_val_score(ada_classifier,  # specify classifier
                         X_train,               # specify features
                         y_train.ravel(),       # specify labels
                         cv = 5)           # specify 5-fold cross validation
```

In [24]:
```
# calculate mean score across models
# ----------
scores.mean()
```

Out[24]:
```
0.9184625308686218
```

Validation Metrics

Be sure to explain which of these metrics you would want to prioritize when conducting predictive auditing in this context and why.

Hint: Try writing a for loop to use `cross_val_score()` to check for accuracy, precision, recall and f1 across all of your models.

Explanation:

In [25]:
```
#Validate the models:

models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'AdaBoost': AdaBoostClassifier()
}

# Define the metrics as scorers
scorers = {
    'Accuracy': make_scorer(accuracy_score),
    'Precision': make_scorer(precision_score),
    'Recall': make_scorer(recall_score),
    'F1': make_scorer(f1_score)
}

# Perform cross-validation and calculate metrics for each model
for model_name, model in models.items():
    print(f"\n{model_name}:\n")

    # Perform cross-validation for each metric
    for metric_name, scorer in scorers.items():
        scores = cross_val_score(model, X_train, y_train, cv=5, scoring=scor
        avg_score = scores.mean()
        print(f"{metric_name}: {avg_score:.4f}")
```

```
Logistic Regression:

Accuracy: 0.9202
Precision: 0.7755
Recall: 0.8509
F1: 0.8112

Random Forest:

Accuracy: 0.9278
Precision: 0.7768
Recall: 0.9050
F1: 0.8333

AdaBoost:

Accuracy: 0.9180
Precision: 0.7840
Recall: 0.8196
F1: 0.8013
```

For this policy application, I think it would be most important to prioritize the F1 score because of our class imbalance (80% pass/20% fail), which would make our accuracy scores artifically high (when compared to a majority class classifier). The reason to use the F1 score is that it is the ratio of the product of precision and recall over the sum of precision and recall and therefore balances the tradeoff between false positives (precision) and false negatives (recall). The F1 score should be as close to 1 as possible, so for our models it looks like the random forest has the highest of the three models and would be the best one to use based on this metric.

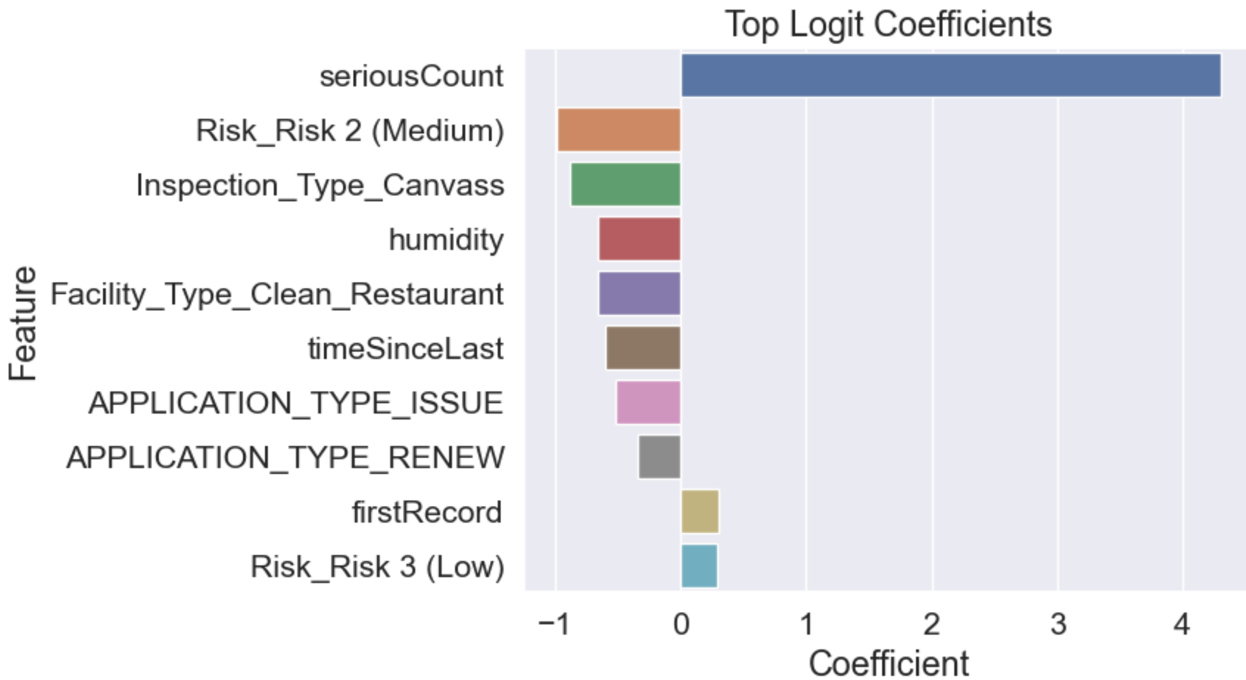
4. Policy Simulation

Interpretable Machine Learning

Use tools like coefficient plots or feature importance plots to investigate your models. Which features contribute to your predictions? Are there any additional features you wish you could incorporate that you don't have available in this analysis?

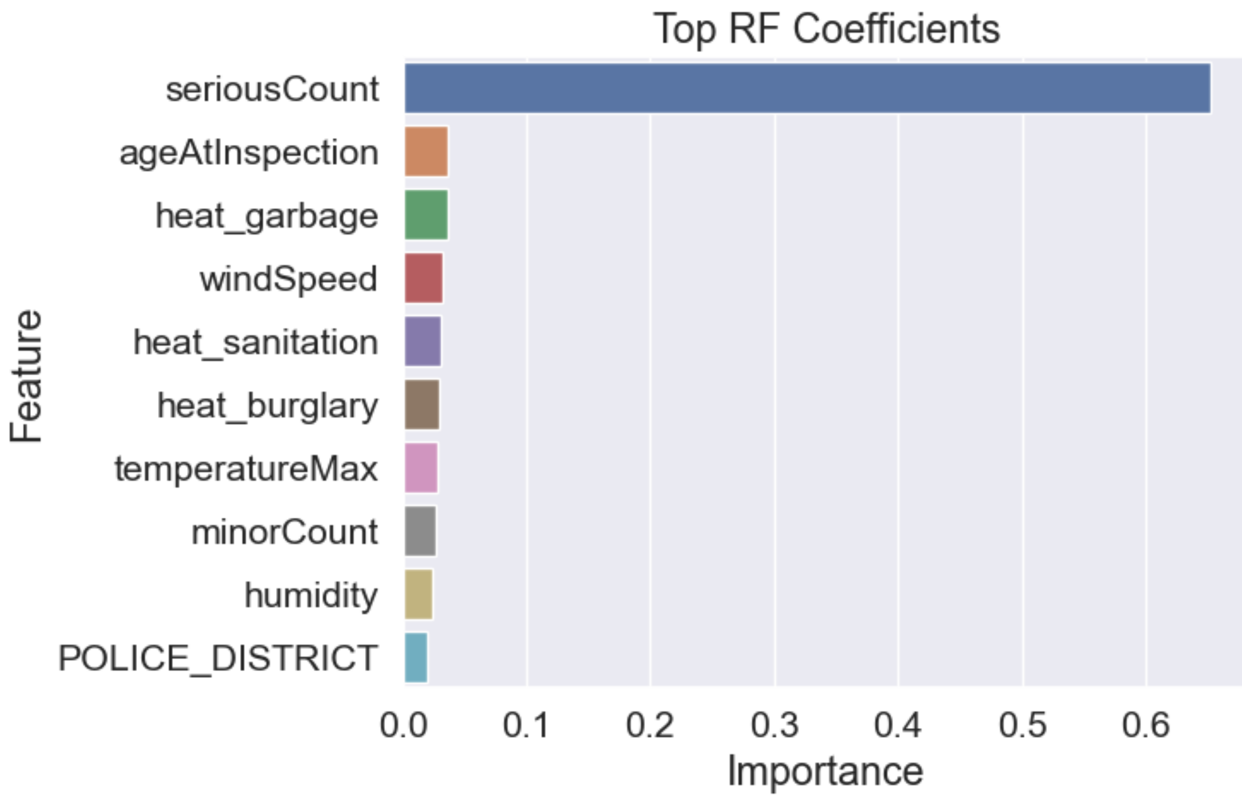
Hint: Use tools like feature importance plots and coefficient plots.

In [71]:
```
# plot
sns.barplot(x="Coefficient",
            y="Feature",
            data=logit_data.nlargest(10, 'abs_coef')).set_title("Top Logit C
plt.show()
```

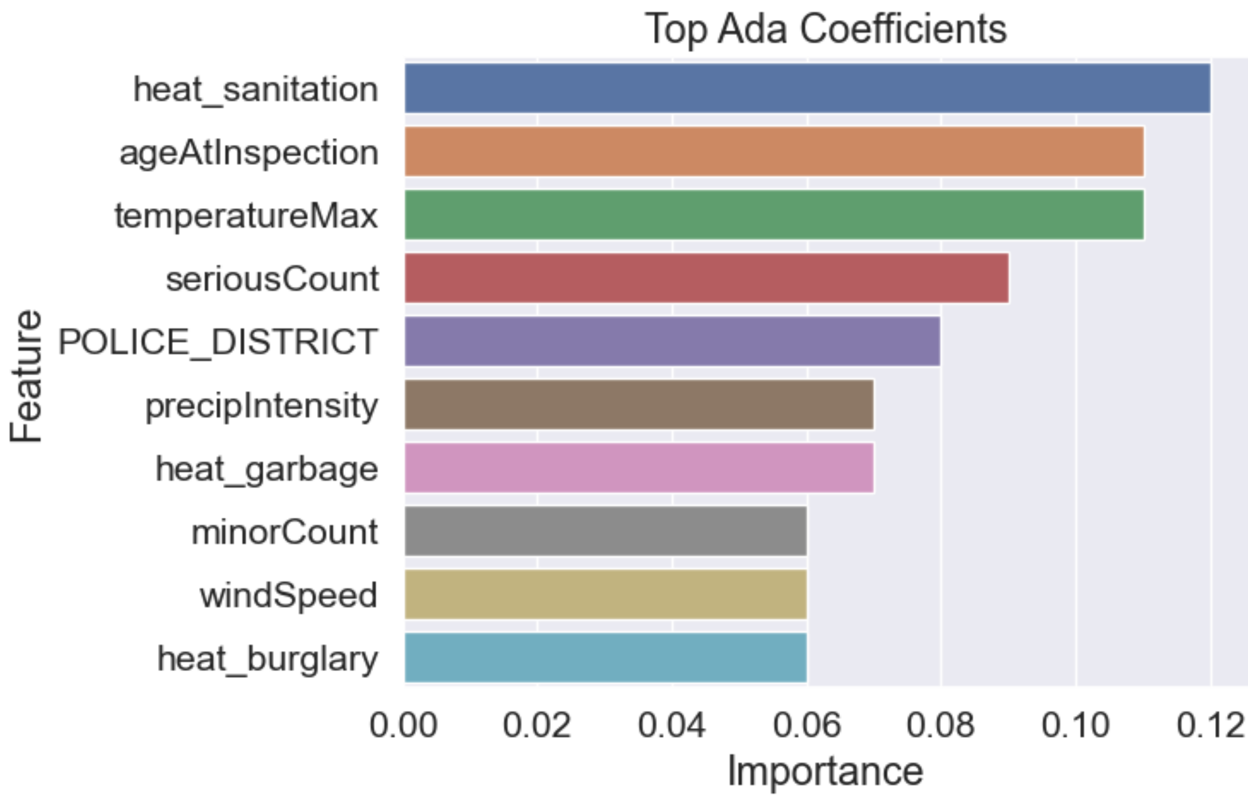


In [69]:

```
# import library
import seaborn as sns

# create feature importance dataframe
feat_importances = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(np.transp
feat_importances.columns = ["Feature", "Importance"]

# plot
sns.barplot(x = "Importance",
            y = "Feature",
            data = feat_importances.nlargest(10, 'Importance')).set_title("I
plt.show()
```



In [68]:
```
# import library
import seaborn as sns

# create feature importance dataframe
feat_importances = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(np.transp
feat_importances.columns = ["Feature", "Importance"]

# plot
sns.barplot(x = "Importance",
            y = "Feature",
            data = feat_importances.nlargest(10, 'Importance')).set_title("T
plt.show()
```



It looks like the most important features are different across the three models. Random forest and logit seem to think that seriousCount is the most important feature, while AdaBoost thinks that heat_sanitation, ageAtInspection, temperatureMax are all more important than seriousCount. I think that some other features to include would be who their main suppliers are (as I'm sure some of the facilities share wholesale distributors) for certain types of food, and whether the establishment serves raw foods (like grab and go salads or sushi than can be more prone to foodborne illnesses).

Prioritize Audits

Hint: Look up the `.predict()`, `.predict_proba()`, and `.sample()` methods. Then:

1. Choose one of your models (or train a new simplified model or ensemble!) to predict outcomes and probabilities.
2. Order your audits by their probability of detecting a "Fail" score
3. Plot your distribution of pass/fail among the first 1,000 observations in the dataset
4. Simulate random audits on the full chicago_2011_to_2013.csv dataset by picking 1,000 observations at random

In [85]:
```
#
# 1. Choose one of your models (or train a new simplified model or ensemble!
# -----------
# Get predicted values for the random forest model
rf_pred = rf_model.predict(X_test)

# Using predict_proba() to predict probabilities for test data
proba_predictions = rf_classifier.predict_proba(X_test)
```

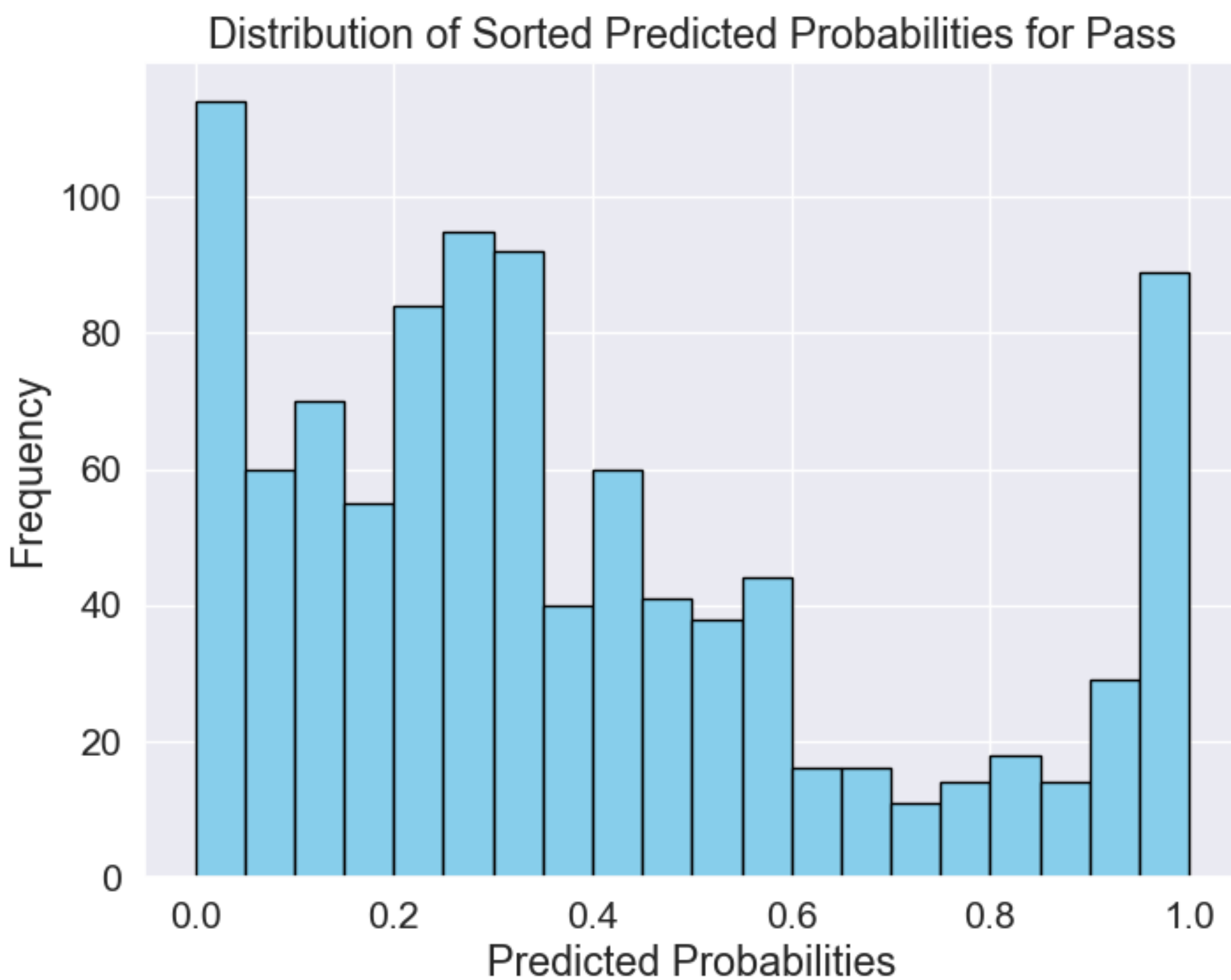
In [29]:
```
proba_predictions

#the first number in each row is the probability of a 0=Pass prediction;
# the second number is the probability of a 1=Fail prediction
```

Out[29]:
```
array([[1.  , 0.  ],
       [0.24, 0.76],
       [1.  , 0.  ],
       ...,
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.32, 0.68]])
```

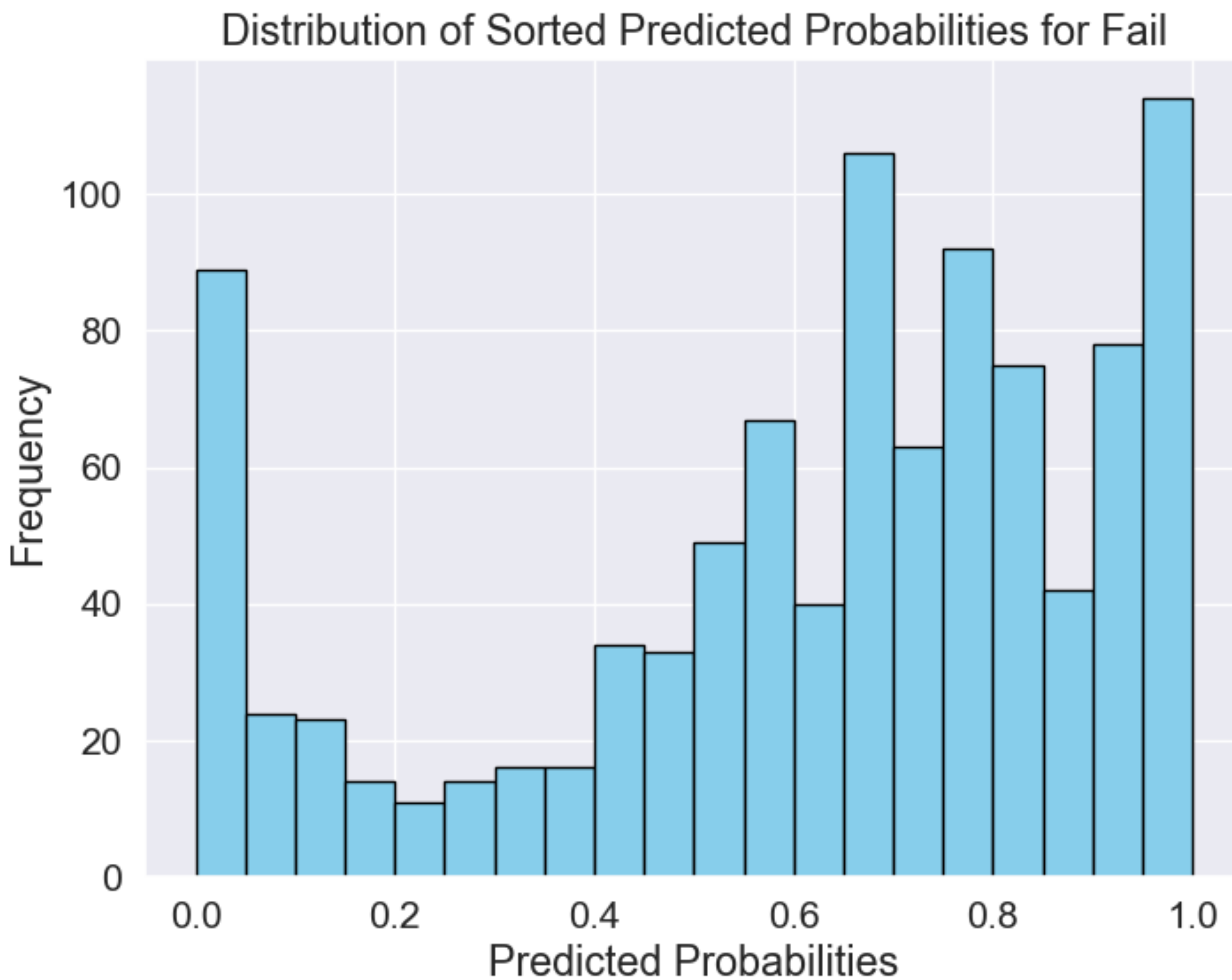
In [30]:
```
#
# 2. Order your audits by their probability of detecting a "Fail" score
# -----------
# Get the probabilities for class 1 (second column)
class_1_probabilities = proba_predictions[:, 1]

# Sort the indices based on the probabilities of class 1 in descending order
sorted_indices = np.argsort(class_1_probabilities)[::-1]

# Sort the proba_predictions array based on sorted indices
sorted_proba_predictions = proba_predictions[sorted_indices]

sorted_proba_predictions
```

Out[30]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

In [31]:
```
#
# 3. Plot your distribution of pass/fail among the first 1,000 observations
# -----------
# get the first 1,000 observations (indices 0 to 999)
num_observations = 1000
probabilities = sorted_proba_predictions[:num_observations, 0]  # Probabilit

# Plotting the distribution of sorted predicted probabilities
plt.figure(figsize=(8, 6))
plt.hist(probabilities, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Sorted Predicted Probabilities for Pass')
plt.xlabel('Predicted Probabilities')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



In [32]:
```
probabilities_1 = sorted_proba_predictions[:num_observations, 1]  # Probabil

# Plotting the distribution of sorted predicted probabilities
plt.figure(figsize=(8, 6))
plt.hist(probabilities_1, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Sorted Predicted Probabilities for Fail')
plt.xlabel('Predicted Probabilities')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



In [97]:
```
#
# 4. Simulate random audits on the full chicago_2011_to_2013.csv dataset by
# -----------

# Get the total number of observations in your dataset
total_observations = X.shape[0]

# Set the number of observations to audit (1000 in this case)
num_observations_to_audit = 1000

# Randomly select 1000 rows from the dataset without replacement
subset_data = X.sample(n=num_observations_to_audit, replace=False)
```

Predict on 2014 inspection data

Use your favorite model to make predictions based on the features using the "Chicago Inspection 2014_updated.csv" file. Treat this as you would a test dataset. This means you will have to format the features (including removing some features and getting dummies) and the label (binarize and recode) in the same way you did the training data. (Remember the "Results" column is your label). You will then compare your predictions with the actual.

In [34]:
```
# data processing

#
# read in  "Inspections Data 2014_updated" csv data
# -----------
chicago_inspections_2014 = pd.read_csv("data/Chicago Inspections 2014_update
                                        low_memory=False)
```

In [43]:
```
df2 = chicago_inspections_2014
```

In [44]:
```
df1 = chicago_inspections_2011_to_2013
```

In [46]:
```
common_columns = df1.columns.intersection(df2.columns)
common_columns
```

Out[46]:
```
Index(['Facility_Type', 'Risk', 'Inspection_Type', 'Results',
       'Facility_Type_Clean', 'criticalCount', 'seriousCount', 'minorCount',
       'pastFail', 'pastCritical', 'pastSerious', 'pastMinor', 'timeSinceLas
t',
       'firstRecord', 'WARD_PRECINCT', 'POLICE_DISTRICT',
       'LICENSE_DESCRIPTION', 'APPLICATION_TYPE', 'ageAtInspection',
       'consumption_on_premises_incidental_activity', 'tobacco',
       'package_goods', 'outdoor_patio', 'public_place_of_amusement',
       'limited_business_license', 'childrens_services_facility_license',
       'tavern', 'regulated_business_license', 'filling_station',
       'caterers_liquor_license', 'mobile_food_license', 'precipIntensity',
       'temperatureMax', 'windSpeed', 'humidity', 'heat_burglary',
       'heat_garbage', 'heat_sanitation', 'criticalFound'],
      dtype='object')
```

In [57]:
```
#select the columns that are relevant for the 2014 dataset
X_full = chicago_inspections_2014[['Risk', 'Inspection_Type', 'Results',
       'Facility_Type_Clean', 'criticalCount', 'seriousCount', 'minorCount',
       'pastFail', 'pastCritical', 'pastSerious', 'pastMinor', 'timeSinceLas
       'firstRecord', 'POLICE_DISTRICT', 'APPLICATION_TYPE', 'ageAtInspectic
       'consumption_on_premises_incidental_activity', 'tobacco',
       'package_goods', 'outdoor_patio', 'public_place_of_amusement',
       'limited_business_license', 'childrens_services_facility_license',
       'tavern', 'regulated_business_license', 'filling_station',
       'caterers_liquor_license', 'mobile_food_license', 'precipIntensity',
       'temperatureMax', 'windSpeed', 'humidity', 'heat_burglary',
       'heat_garbage', 'heat_sanitation', 'criticalFound']]

# drop datetime info
# -----------
X_full = X_full.dropna()
```

In [58]:
```
# drop missing vals
X14 = X_full.dropna()

# -----------
y14 = X14['Results']

# decide if you want to binarize the outcome variable
# -----------
# comment out the following lines of code if you don't want to binarize the
y14 = y14.replace({'Pass w/ Conditions': 'Pass'})
lb_style = LabelBinarizer()
y14 = lb_style.fit_transform(y14)

# recode 0s and 1s so 1s are "Fail"
y14 = np.where(y14 == 1, 0 ,1)


# process features
# -----------

# process features
X_test2014 = X14.drop(columns = ['Results'])
X_test2014 = pd.get_dummies(X_test2014)

# process target
y_test2014 = y14
```

In [59]:
```
# predict and compare using already trained rf_classifier
# -----------
# Get predicted values for the random forest model that we chose
rf14_pred=rf_model.predict(X_test2014)
```

In [60]:
```
#calculate accuracy score for rf14_pred
accuracy_score(y_test2014, rf14_pred)
```

Out[60]:
```
0.8958862366683595
```

In [64]:
```
#calculate precision
precision_score(y_test2014, rf14_pred)
```

Out[64]:
```
0.70917225950783
```

In [65]:
```
#calculate recall
recall_score(y_test2014,rf14_pred)
```

Out[65]:
```
0.8086734693877551
```

In [66]:
```
#calculate F1
f1_score(y_test2014,rf14_pred)
```

Out[66]:
```
0.7556615017878427
```

Given that the F1 score balances the precision and recall of the model, both which are important when considering our imbalanced data set, I think it is the right metric to evaluate our results with. The F1 using the 2014 test data is 0.7556, while the F1 from the original random forest model was 0.833. Comparing the recall scores, it looks like the original model had a higher score (0.8 vs 0.9), meaning that it was better able to minimize false negatives in the training set. Comparing the precision scores, it looks like the original model also had a higher score (0.77 vs 0.70), indicating that it was better able to minimize false positives. So, the lower F1 score is driven by a decrease in both the precision and recall of the model on 2014 test data.

5. Discussion Questions

1. Why do we need metrics beyond accuracy when using machine learning in the social sciences and public policy?

Accuracy measures the number of correct predictions over the number of total predictions made (the size of your test set). It can be misleading when your dataset has a high imbalance of labels such that the classifier has a high accuracy just by predicting the majority class in the training set (majority class classifer). Furthermore, misclassification errors can carry a high cost in the social sciences because of potential harms that may be enacted as a result; for example, in this project if a highly accurate classifier is replicating bias in the data collected on establishments that unfairly subject certain businesses to audits. Ideally we would like to have high scores across multiple metrics (e.g. precision being important in mental health treatment recommendations to avoid high false positives or negatives, and recall being important in child protection to avoid missing true positives).

1. Imagine that establishments learned about the algorithm being used to determine who gets audited and they started adjusting their behavior (and changing certain key features about themselves that were important for the prediction) to avoid detection. How could policymakers address this interplay between algorithmic decisionmaking and real world behavior?

Policymakers could try to select classifiers that use other methods such that those particular features no longer hold a lot of weight, but the accuracy/recall/precision of the classifier is still comparable. Alternatively, they could opt to just randomly audit instead because if establishments are successfully manipulating their features to avoid detection that would otherwise result in a fail status (low true positive rate), using classifiers with the same feature set would likely encode the similar weights on those key features that are being manipulated. Ideally, policymakers should be able to stay transparent about what their algorithm is doing without compromising the fidelity of the data being generated/collected - I think this may be possible by speaking generally about the model and its mechanism without specifying which features are weighted heavily and in what direction (of influencing probability of a pass/fail).