Capstone #2 Project Report:

Predicting San Francisco's 311 Service Requests

Pawandeep Jandir

Introduction

The City and County of San Francisco operates a 311 program. It is the official service for residents, visitors, and businesses to obtain information, report problems, or submit service requests to the city (non-emergencies only). This system is available 24 hours a day and is meant to be a fast and efficient way to interact with the local government. Requests can be made online, over the phone, through the mobile app, amongst other ways.

The Problem

The 311 program offers a glimpse into the local government and how it operates. The interactions represent, to some degree, the needs and desires of San Francisco. This presents an opportunity to ask the question: can we predict future interactions, specifically, service request volumes? Even further, do certain service requests come from certain areas or neighborhoods of San Francisco? By answering these questions we can try to optimize the government, via the 311 program. This would allow the city to be more proactive and responsive to the needs of people through predictive modeling. To help with the prediction and forecast, socio-economic and demographic data (from the US Census Bureau) of San Francisco will also be gathered.

The Client

The potential target for this project is the San Francisco local government. The analysis can help the city understand how certain city services are used and requested throughout different parts of the city. A predictive model providing count or volume data on future service requests can help the local government in becoming more effective for its citizens, visitors, and institutions. By becoming more efficient and proactive, the San Francisco local government can improve the city and make a large positive impact on its large and diverse communities.

The Data

The main data source is the 311 request data available at San Francisco's open data portal. In addition, San Francisco's socio-economic, demographic and other types of data is available from the US Census Bureau FactFinder website. Lastly, San Francisco's open data portal also provides a shapefile, in the form of a GeoJSON, outlining the city. This is needed during the data cleaning phase and will be discussed in further detail, below.

Additional sources of data that could be added to this project include more demographic or socio-economic variables. Variables related to crime could also be quite useful and improve the predictive models.

The Tools

The entirety of this project uses *python* and *JupyterNotebook*. Some of the python packages used are *numpy*, *pandas*, *BeautifulSoup*, *matplotlib*, *seaborn*, *scipy*, *statsmodels*, and *scikit-learn*. The entire project can be found on GitHub here. The code and scripts can be run completely out of the box, as the repository includes all related code and data. Reports, such as this one, can also be found in this online repo for further details and discussions.

Data Wrangling

This section details the data wrangling and cleaning of the various data stores to create a single cohesive dataset.

Acquisition

After identification of data sources, the next step is to obtain it somehow. For the 311 dataset, we can go to San Francisco's open data portal website for the 311 program. From here, it is as simple as downloading the dataset as a CSV file. Additional export methods are also available including JSON and XML. An API also exists which makes streaming the data a possibility as well. For the demographic data, we go to the FactFinder website maintained by the US Census Bureau. From here, we submit the correct location, City and County of San Francisco, and request a breakdown by Census Tract. We also specify the American Community Survey dataset. From here we select the relevant socio-economic and demographic data columns. See Tab. 1 for the full list of these variables.

Cleaning

First we read-in the 311 data with *pandas* which has over 2.8 million records. A warning is given about the last column of the input CSV file. It is a column with a string for a hyperlink to an associated image or picture to the service request. Not all of the records have this information, so the first thing we can do is simply drop this column and replace it with a boolean column showing wether or not there was this hyperlink string. The information contained in the links is potentially useful but is outside the scope of this project and can perhaps be investigated in another project. Next, because this is a time series dataset we want to ensure the time-based columns are recognized as such, so we convert them to the datetime format that *python* prefers.

We can see the dataset starts midway through 2008. Because we want to capture seasonality and fully model the data, we can throw out the records from this partial first year. Similarly, we can throw away data from this year (2018). We can question whether this is good idea; however, it is probably a good idea. Due to computational limits on my personal machine, this full dataset would take too long to process. The key component to this analysis is connecting this 311 data to the US Census Tract information. The US Census provides an API where the tract number is returned given a street address, which we do have in the 311 data. The problem is the time it takes to make this call. Even with batch submission of addresses, it takes around a half second to process each. Given the 2.8 million records, that would mean it would take over 16 entire days to complete the whole dataset! This limitation forces us to pare down the dataset to keep this time to more reasonable levels.

With this restriction in place, we can start slowly reducing the size of our dataset. First up, there are around 300,000 records with no given address: listed as "Not associated with a specific address." Along with the other handful of missing addresses, these records are dropped from the dataset. The API also has strict rules on the input addresses: they must have street numbers. This means intersections are not accepted and thus we also have to drop records that do not have any street number in the address. We also remove records whose address appears to have more commas than we generally expect from a normal street address. There are certainly ways to parse this kind of data column, but again since we are motivated to remove records, this is another negative filter for the data.

We can turn our attention to the actual types of requests in this dataset. The top sixteen categories cover nearly 90% of the entire dataset. Again, because we want to reduce the dataset, we can keep only the top eight. These eight category types represent about 74% of the dataset. The list below details these eight categories.

- Street and Sidewalk Cleaning
- Graffiti
- Abandoned Vehicle
- Homeless Concerns
- Encampments
- Sewer Issues
- Streetlights
- Damaged Property

Note that some of these are related, but are categorized separately. We will also analyze these correlations in full, later in this document. With this choice, we get to keep much of the dataset while still dropping records.

Then we can make sure all records in the dataset are actually in San Francisco. After a visual check, this is actually not the case. As mentioned earlier, we can get a shapefile containing the outline of the San Francisco. Using this GeoJSON file, we can cut out all records outside the city limits. After performing this step, we can plot where the requests have been made. This dot graph is shown below in Fig. 1.

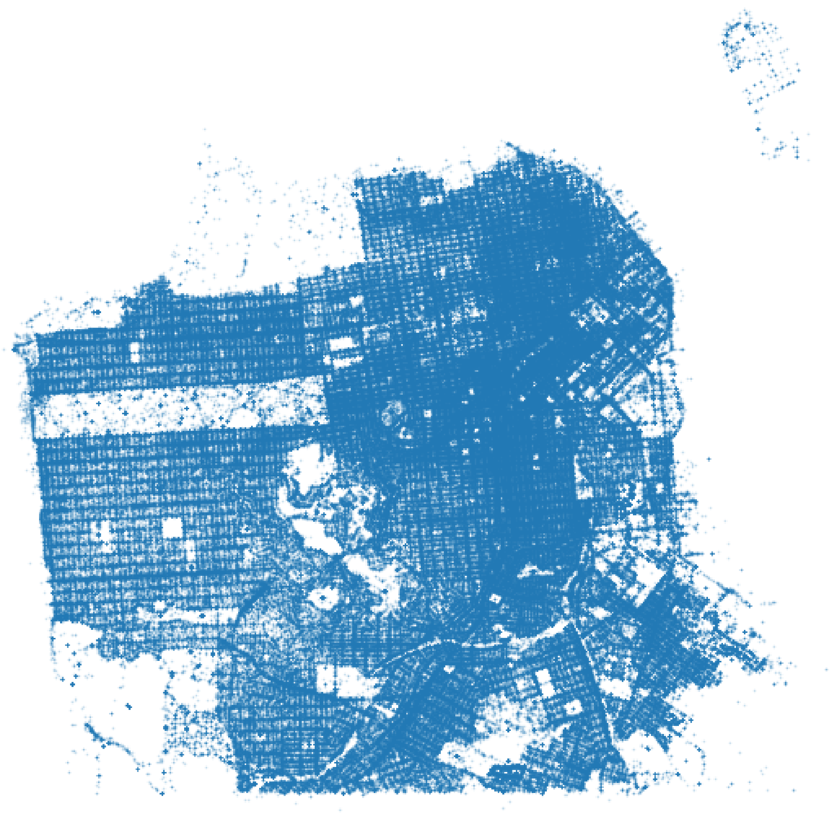

Figure 1: 311 service requests in San Francisco

This graphic gives us a nice outline of the city, including Golden Gate Park (the sparse bar in the upper left). Lastly, we look at the time component of the dataset again. It represents nine full years, from 2009 to 2017. By keeping only the last three years (2015 to 2017), we reduce the dataset to about 550,000 records. Three years of data should be enough to be able to capture possible seasonality and fit the data well. This also means we kept only around 20% of the original dataset, as we went from about 2,800,000 to 550,000 records.

Features

At the end of this process we have many variables or columns in our dataset. Many are not needed or otherwise are not within the scope of the analysis. Such columns include status notes, request details, and responsible agency. It should be noted that there may be textual information in these columns that might yield predictive information. However, this type of feature engineering is outside the goal of this project so these columns, amongst several others, are dropped from the dataset. The table below, Tab. 1, lists all kept columns with descriptions and any related notes.

Table 1: Full list of features in the cleaned dataset

Name	Description	Notes
Date	Date of the service request	-
Category	Service request type	Eight categories
Supervisor_District	SF Supervisor District	Twelve categories
Police_District	SF Police District	Ten categories
Latitude	Latitude of service request	-
Longitude	Longitude of service request	-
Source	Ways to make service request	Seven categories
Has_Media_Link	If media was included in service request	Boolean
Is_Closed	If service request is now closed	Boolean
Tract	Given US Census Tract	199 categories
Poverty	Poverty ratio in percent	Range from 0 to 100
Income	Median income in dollars	Range from 19,440 to 250,000
Unemployment	Unemployment ratio in percent	Range from 0 to 100
Population	Total population	Range from 89 to 13,057
Percent_Male	Percentage of male population	Range from 0 to 100
Percent_Pop_0_14	Ratio of population between 0 and 14 years old	Range from 0 to 1
Percent_Pop_15_24	Ratio of population between 15 and 24 years old	Range from 0 to 1
Percent_Pop_25_64	Ratio of population between 25 and 65 years old	Range from 0 to 1
Percent_Pop_65_up	Ratio of population over 65 years old	Range from 0 to 1
Percent_Pop_White	Ratio of population identifying as white	Range from 0 to 1
Percent_Pop_Black	Ratio of population identifying as black	Range from 0 to 1
Percent_Pop_Hispanic	Ratio of population identifying as hispanic	Range from 0 to 1
Percent_Pop_Asian	Ratio of population identifying as asian	Range from 0 to 1
Percent_Pop_Other	Ratio of population identifying as other	Range from 0 to 1

The full code detailing these data wrangling steps can be found on my GitHub. A link to this *JupyterNotebook* is given here.

Data Exploration

Initial and exploratory data analysis of the cleaned dataset is performed using various visuals and inferential statistics. The purpose is to gain some insight into how the time series component of the data is distributed and influenced by the various service request categories.

Visual Analysis

We can start exploring the data exploration process by looking at its time component. As we previously did, there are three years worth of service requests. How many requests are in each year? This is answered in Tab. 2 below.

Table 2: Request count per year

Year	Count
2015	131,522
2016	186,867
2017	227,737

The increasing trend is pretty clear. We can try to see what the dependence is on the month, regardless of the year. Fig. 2 shows this below.

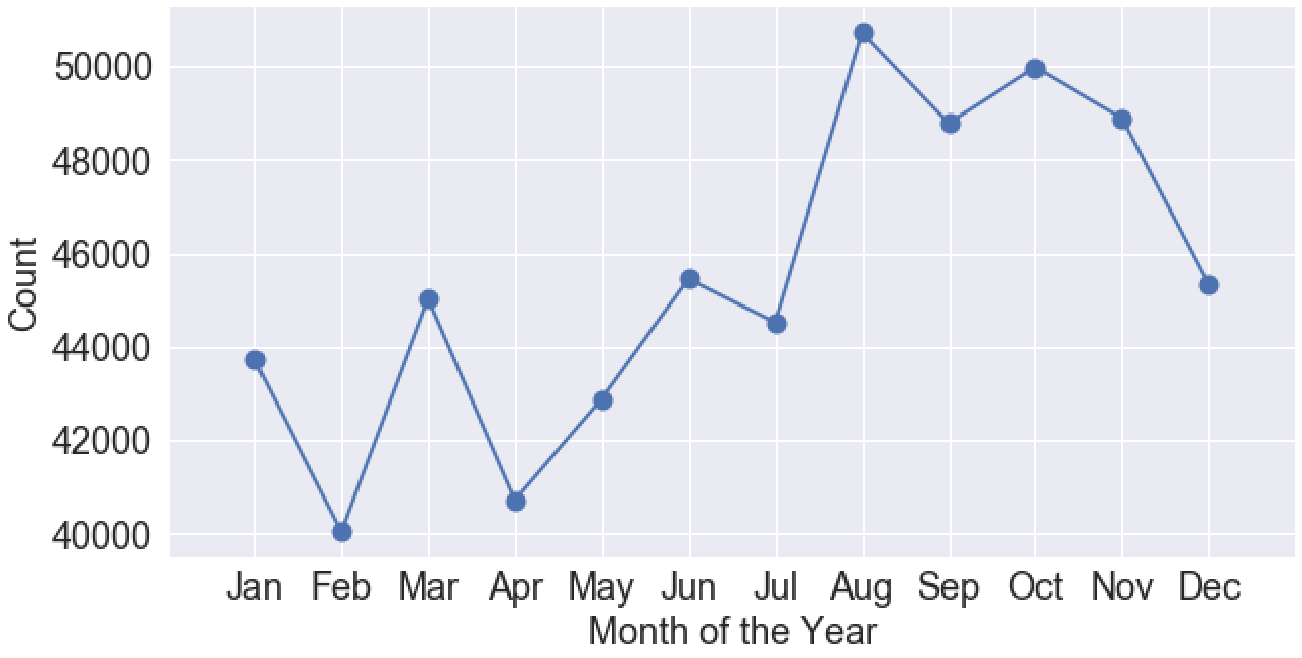

Figure 2: Request count per month

It is interesting to see that there is quite a bit of variation in any particular month. It is clear the back half of the year gets more requests, though it is not immediately clear why this is true. Perhaps there are PR campaigns or other advertisements for the 311 service during that time frame? At this point we can only speculate, though it is an intriguing result. We can also make a similar plot, in Fig. 3, for the days of the week to see how service requests fare across the seven days.

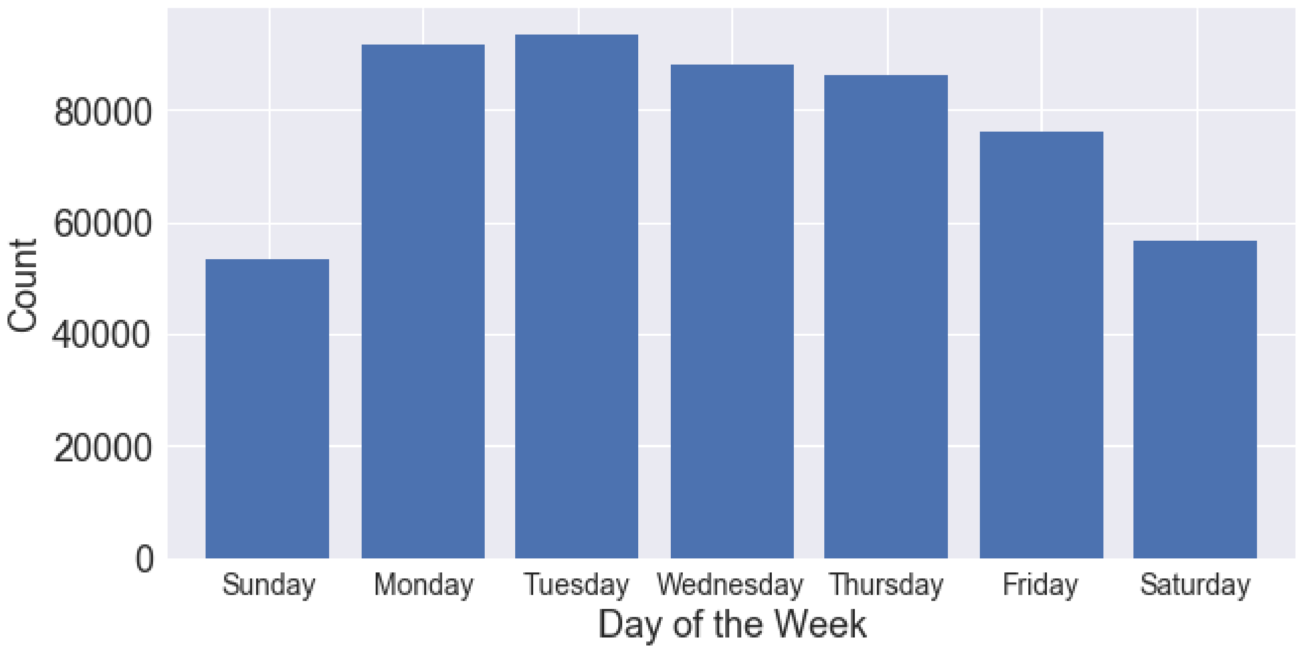

Figure 3: Request count per day of the week

This distribution also seems to make sense. There are less requests on the weekend, but is pretty constant within both weekends and week days, respectively.

Next, we can see how the different categories are distributed throughout the three years in the data. Here we up-sample the service requests to the order of weeks (from seconds) and count the requests. In other words, we resample per week and count the requests for each category during that time frame. Then we can plot this weekly count data as a time series for the eight categories. This plot is shown below in Fig. 4.

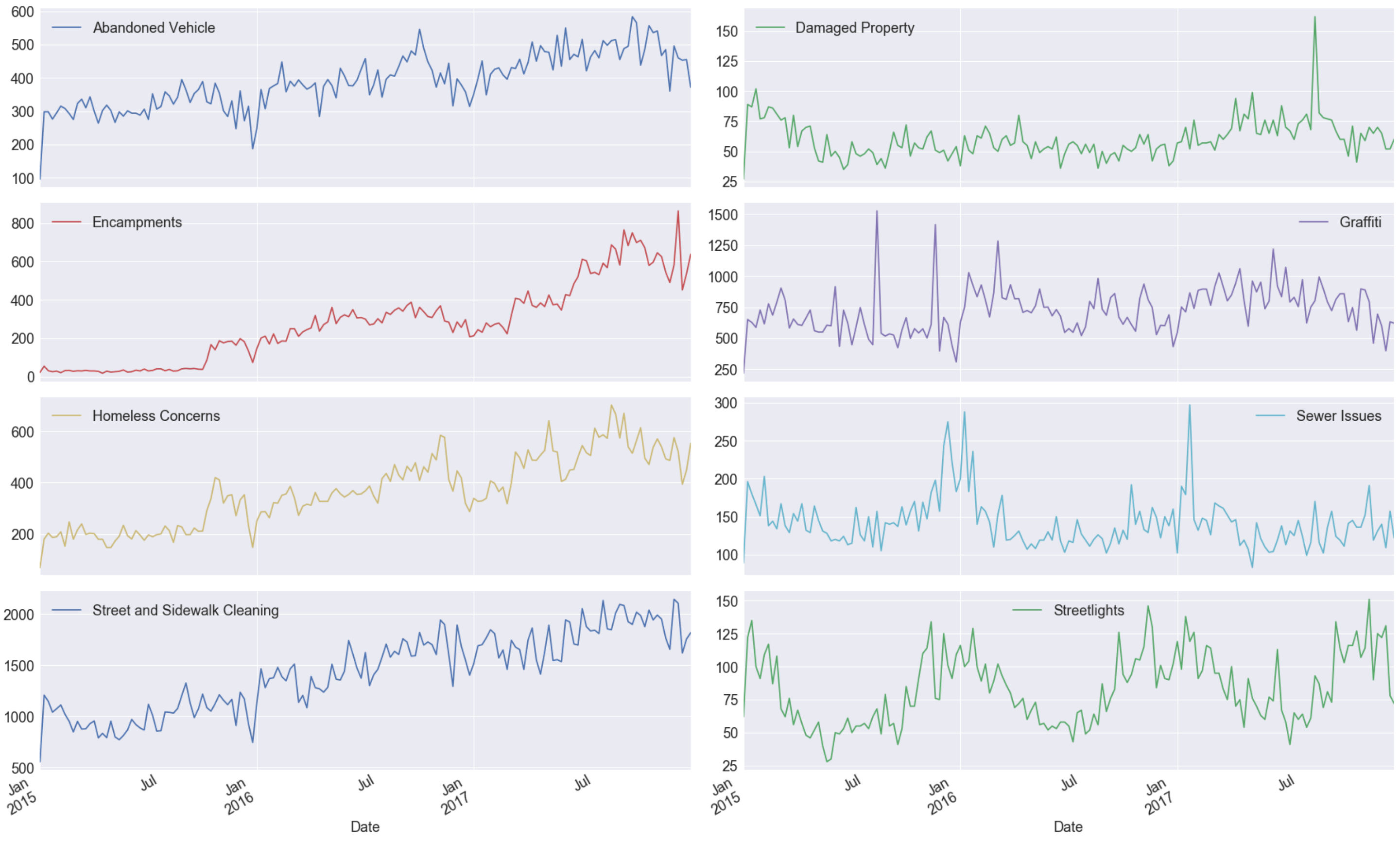

Figure 4: Weekly request count for each category

We can see how the eight categories have had requests over the past few years. Its interesting to see the large uptick in requests regarding Encampments starting in late 2015. We can presume that, coupled with the overall upward trend in Homeless Concerns, San Francisco is actively looking into this problem. There is also an obvious seasonal pattern in Streetlight requests, which we perhaps should have guessed beforehand. Lack of street lights is of course a much bigger issue in the darker, winter months.

Statistical Inference

We can also look at how the eight categories are related to each other. As we have already surmised, these categories are potentially related. Some of these are obvious like Abandoned Vehicle and Street and Sidewalk Cleaning. How much are they related though and can we quantify it? We can make a heatmap showing the Pearson correlation. This triangular heatmap is shown below, in the top panel of Fig. 5. This plot does indeed some large correlations. How significant are they though? We can calculate the $p-$values for the correlations and plot the results, in the bottom panel of Fig. 5, in the same way (i.e. a triangular heat map).

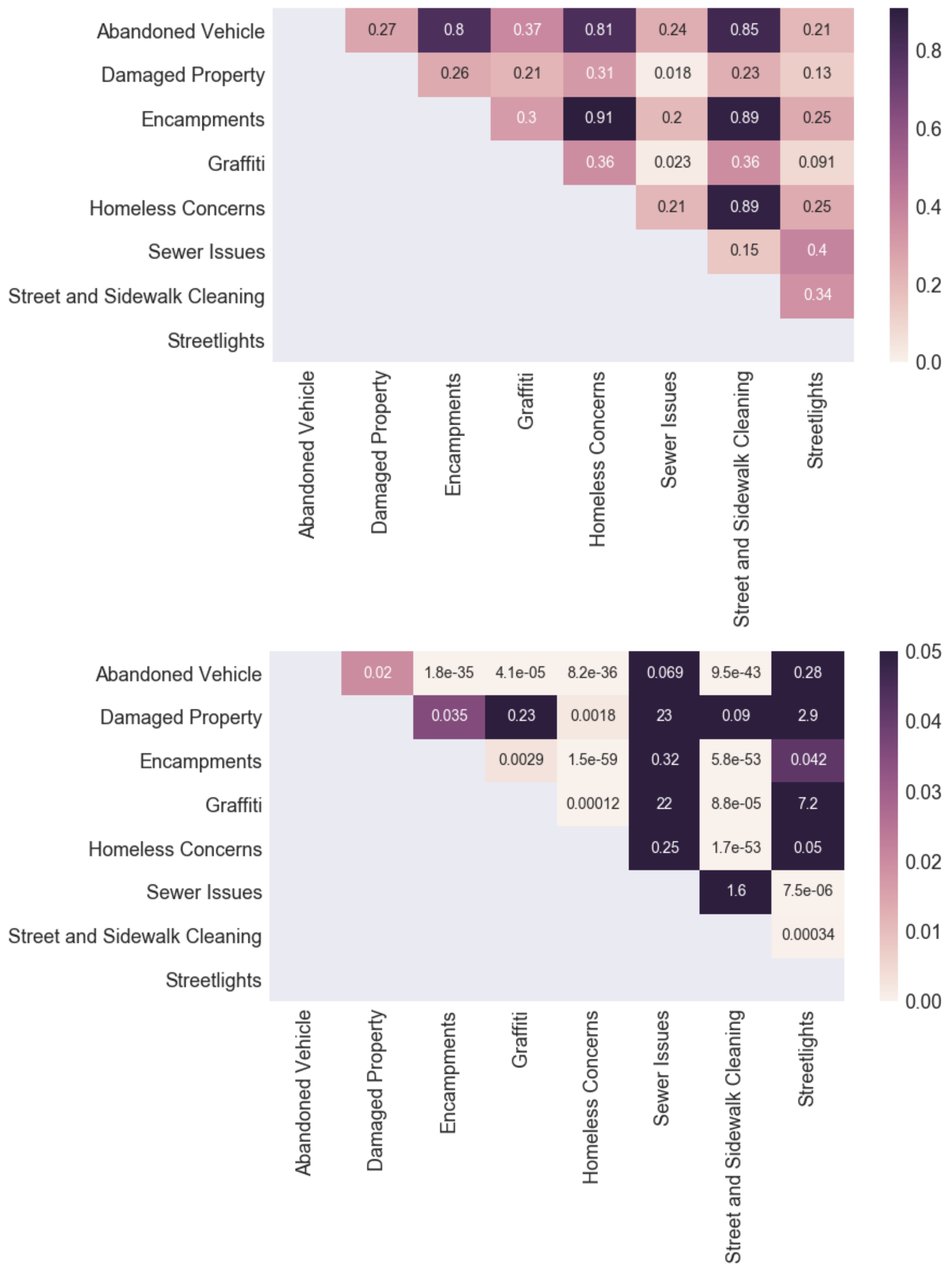

Figure 5: Correlations between weekly request counts for all eight categories. The top panel shows the Pearson correlation coefficient while the bottom panel shows the correlation coefficient's associated $p-$value.

By chance, any one of these comparisons can be very small (i.e. $p < 0.05$), potentially indicating significance where there might not actually be any. This is known as the multiple comparisons problem or the look-elsewhere effect in particle physics. This issue cannot be ignored. The simplest, and most conservative, technique to deal with this problem is to divide the threshold, or α level, by the number of comparisons made. Of course there are additional techniques to tackle this issue. However, we will stick with this simple solution, the Bonferroni correction. In this case then, there are 28 comparisons ($\frac{8*7}{2}$), so we will need to adjust with this number in mind. For visual purposes, we will actually multiply the $p-$values by 28, so that we can keep the usual α=0.05 threshold. The effect is the exact same though. In this plot the z-axis (or color bar) has an upper limit of 0.05 to show which correlations might be significant at the α=0.05 level. Note also that a lighter color indicates a small (adjusted) $p-$value, suggesting real significance of the correlation.

Even with our correction, we can see that some $p-$values are (very) small, indicating real significance and correlation between categories. In fact, over half of the correlations show significance. This makes some sense as many of these categories do share similarities in the underlying issue. For example, we can probably expect real correlation between Encampments and Homeless Concerns as they are related social issues, especially in San Francisco.

The full code detailing these data exploration steps can be found on my GitHub. A link to this $JupyterNotebook$ is given here.

Data Modeling

Preprocessing

After exploring and analyzing the dataset, we can move towards building a suitable model to predict the weekly counts of service requests. One of the first things we need to do at this point is preprocess the data so that it is standard, roughly, across all the features. Remember the purpose of the project is to use additional census, or demographic, data to help predict the 311 service request patterns. So while the majority of our cleaned dataset does have that information there is also additional information relating to the 311 service requests only. These extra columns are then discarded.

The majority of the census features are ratios, defined from 0 to 1. However, there are a few columns, Poverty, Unemployment, and Percent_Male, which are ratios, but range from 0 to 100. In addition, there are two columns, Population and Income, which are ratios and whose range is (somewhat) arbitrary. For this situation, we can use a scaling function to scale them from 0 to 1. We can do this with *scikit-learn*'s MaxAbsScaler function. This uses the maximum value of each feature to scale the entire feature. We want to make sure all these features are on equal footing, so we scale these appropriately to match the rest of the census features.

Model building

Now that we have a dataset ready to be modeled, let's map out our strategy. We want to model how 311 requests are made to San Francisco's local government. However, the data is granular, as it is recorded to the second. It would be difficult to predict a time series with that kind of fine granularity. Instead, we will first up-sample to a week and get the weekly count data. Then we can, for each census Tract and request Category, model the weekly count given the particular census data. Note also, we are surmising the additional census data will help the model. We will have to check this crucial portion of our assumption with the data as well.

Seasonal autoregressive integrated moving average models

We will use a Seasonal Autoregressive Integrated Moving Average with eXogenous variables (SARIMAX) to fit our data. Because the weekly count data is a simple time series, we can apply the well-known (S)ARIMA(X) algorithm, implemented in *statsmodels*, to our data. We use the Seasonal version to incorporate the likely seasonal patterns in the data. The census data features are treated as eXogenous variables and are input as such into the model. We will also optimize the SARIMAX model without the census data to give us an understanding of how much the extra information actually helps. The normal method to apply this type of model is to follow these steps:

1. Visualize the time series data
2. Stationarize the data
3. Find optimal AR and MA terms from the (Partial) Autocorrelation plots
4. Fit the model

This procedure is problematic for us because of the sheer number of time series data we will be creating. There are 8 Categories and 195 Tracts which implies 1,560 time series to fit, model and predict. Clearly determining the best way to optimize each time series cannot be done by hand. Instead we will loop over various parameters to find the optimal one. This is basically the same as hyper-parameter tuning a machine learning model, such as (regularized) linear regression. Similarly here, we will train using only a subset of the full data and test against the remaining data. We will use mean absolute error to score the different models and test them against each other. Like other machine learning processes, we will split a time series into a train and test set. The last 12 weeks worth of data are held out to test the model. This is done to validate a model on unseen data. It is also an attempt to be as unbiased as we can be when measuring a model's performance, in particular in relation to whether or not the exogenous variables were included in it.

An example model is shown in Fig. 6. Here the weekly count time series (red) is fit with (blue) and without (green) the census, or exogenous, variables. The actual fit is shown in the top panel while the forecast, or prediction, is shown in the bottom panel. The mean absolute error scores are given for each fitted model in the lower right of each panel. We can see in this case that including the census variables seems to result in a better model, as it scores better than not including them. However, as we will show later, this is not always the case.

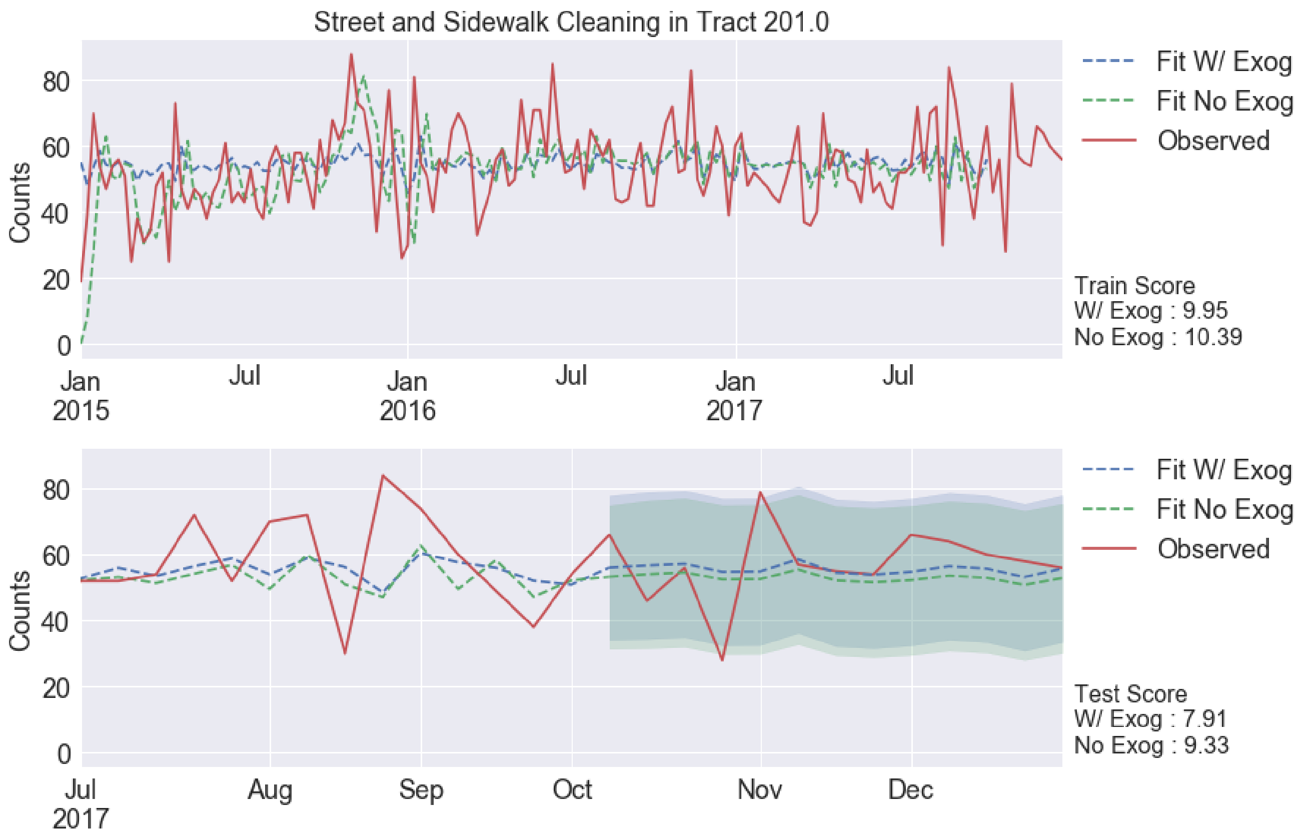

Figure 6: Model fit results (top panel) and forecast (bottom panel) for the Tract 201 and Street and Sidewalk Cleaning time series.

As a further check on the stability and validity of the model we can take a look at some diagnostics. These diagnostic plots are shown in Fig. 7. We can see that the residuals (top left panel) do not seem to have any particular order. This is good as patterns in the residual plot would imply the model has room for improvement. Similarly the correlogram (bottom right panel) shows no particular correlation between lagged versions of the residuals. The residuals also are decently normally distributed (top right and bottom left panels). It is not perfect but again does not show any alarming inconsistencies. Thus, we can be relatively happy that this model is well fit.

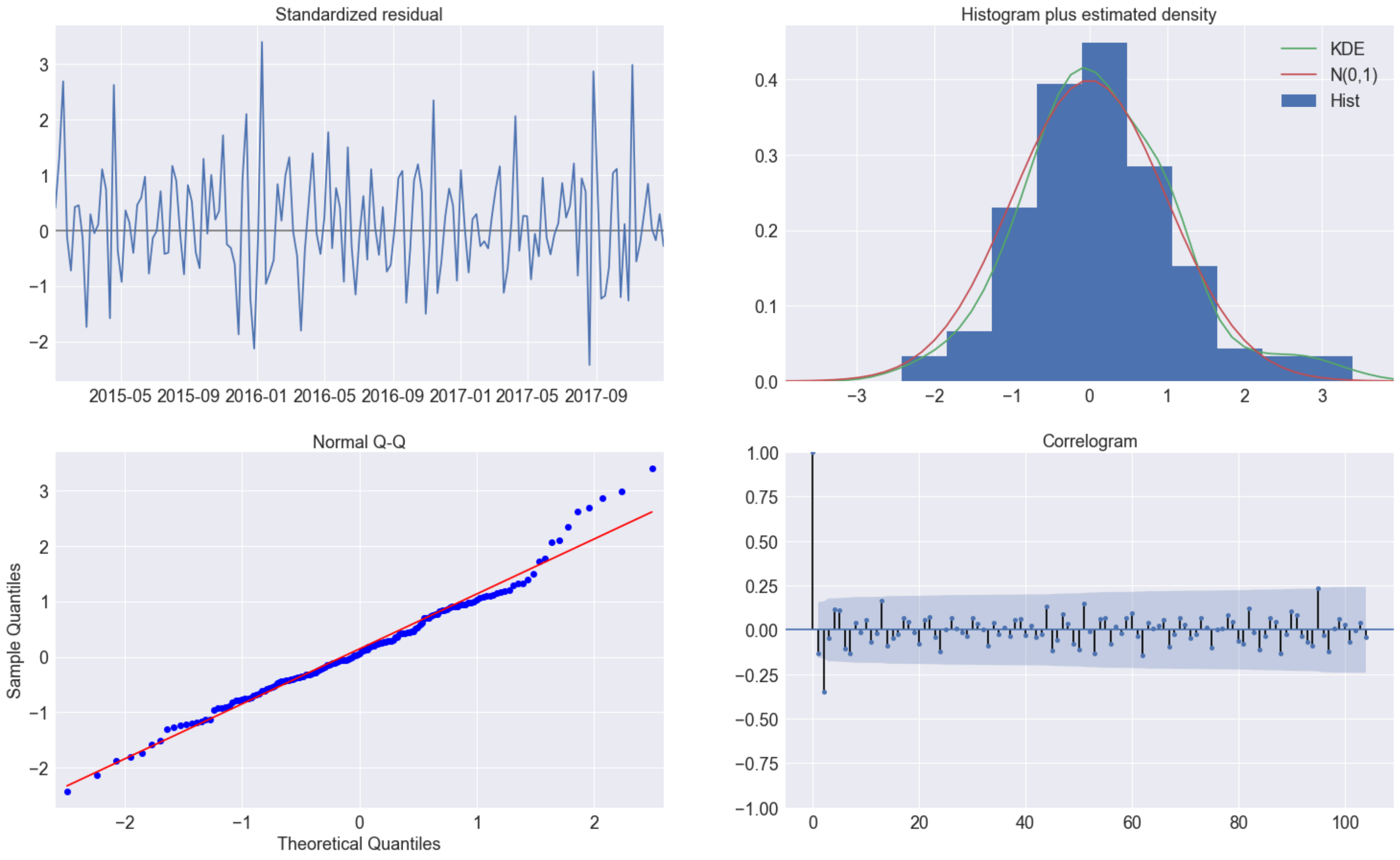

Figure 7: Model fit result diagnostics plots for the Tract 201 and Street and Sidewalk Cleaning time series. The top left panel is the standardized residuals over time, the top right panel is the histogram and estimated density of the standardized residuals with Normal reference density, the bottom left panel is a quantile-quantile plot with Normal reference line, and the bottom right panel is a correlogram.

Even though we are algorithmically choosing the best parameters without intervention, spot checks and inspections like these are still important. We should ensure there are not any surprises or other unexpected behavior in the model and its fit. Over- and under-fitting are both large concerns here since we would like to correctly fit the data for our many time series. Further we can, after further inspections, be satisfied that our model fitting is valid and works as intended.

Computational concerns

It is worth revisiting an important data wrangling choice made earlier in this project. As a reminder, a majority of the data was dropped during the wrangling and cleaning phase. This was done because the API call rate meant it would take over two weeks to get the US Census Tract for all 2.8 million records. Here, we have another two related issues.

First, many of the 1,560 time series, even after up-sampled from a second to a week, still have null entries. That is certain weeks have zero counts from no data. It becomes difficult to model such series when a majority of count values are missing. We can potentially overcome this by interpolating across the data. However, this adds an extra complication and source of bias in the analysis. This issue would be much less of a problem if we had not drastically reduced the dataset from the beginning. This means a re-do of this project would simply keep that data and not run into this problem, at least not as often.

Because of this problem though, we will only keep time series that are complete and non-null. This results in 310 time series, out of the total 1,560.

Second, there is yet another limiting computational cost. Here, instead of the problem coming from API calls, it is actual model fit run time. Running the hyper-tuning code locally is quite lengthy, taking about 20 to 25 minutes. This makes the former requirement a good thing as it helps alleviate the total computational runtime. In fact, the different time series are fit in batches and saved to local disk. This ensures access to the results anytime without having to re-do the calculations.

Full Results

After going through all the pertinent time series, we can take a look at how the models with the census variables ("Better") fared against the ones without ("Worse"). We can also determine which model is better, or if they are within about 0.1% of each other, call the two models equivalent to each other ("Tie"). This is shown in Tab. 3 below, as the percentage of the dataset broken down to which ones benefitted from the exogenous variables.

Table 3: Model results comparing fits with and without exogenous variables. The Better result refers to models including exogenous variables scoring higher than the models not including exogenous variables.

	Results [%]
Better	39.3
Tie	25.8
Worse	34.8

We can see that the models with census variables tend to lead to better fits more often than not, excluding ties. This is not unexpected but it is nice seeing that this idea of including relevant data increases the viability of a predictive model. We can also check the distribution amongst the eight Categories. This is done in Tab. 4, below. Here we again show the percentage, rounded to the nearest tenth of a percent, of the dataset broken down by Category service request type. In other words, each row will sum to 100% percentage, with rounding.

Table 4: Model results comparing fits with and without exogenous variables broken down by Category service request type. The Better result refers to models including exogenous variables scoring higher than the models not including exogenous variables.

	Results [%]		
Category	Better	Tie	Worse
Abandoned Vehicle	36.4	22.7	40.9
Damaged Property	50.0	12.5	37.5
Encampments	22.2	66.7	11.1
Graffiti	33.3	24.1	42.6
Homeless Concerns	52.2	26.1	21.7
Sewer Issues	51.7	03.4	44.8
Street and Sidewalk Cleaning	38.3	32.3	29.3
Streetlights	40.0	0.0	60.0

There is a similar distribution for each Category type as there is overall from Tab. 3. It is somewhat reassuring that this "better rate" is not particularly dependent on Category service request type. We do still have to be careful, of course, as we have intentionally limited our dataset because of computational concerns. Another more inclusive analysis should be better equipped to handle these types of questions.

The full code detailing this in-depth analysis and machine learning can be found on my GitHub. A link to this $JupyterNotebook$ is given here.

Conclusion

We were able to combine and clean different data sources to create a single cohesive dataset summarizing the 311 service requests to the local government of San Francisco. Because of computational concerns, we had to reduce the size of the dataset from around 2,800,000 million to about 550,000 records. We were then able to learn more about the service requests made to the local government of San Francisco. We summarized the time aspect of the data to understand the distribution of service requests across different time scales. It became clear there are some strong correlations between some of these request categories.

We then also showed time series fitting of service request counts, per week, for individual Category types in particular census Tracts is possible with an ARIMA based algorithm. Further, we added terms to the algorithm based on seasonality as well as exogenous, or additional, variables. This SARIMAX algorithm was ultimately the one used to fit the time series data. We then were able to show that including these additional variables, i.e. the US Census-based demographic and socio-economic data, allowed us to achieve better results more often than not.

Further work

There are many avenues of work to improve this project and analysis. One of the most glaring and obvious is to address the computational concerns and issues that have shown up. We can use cloud-based machines to both speed up computational calculations as well as complete them remotely. This would help in two ways. First, we could use up the whole dataset, consisting of 2,800,000 million records. This would of course allow more of the possible 1,560 time series to be fit, at least out of the box. Second, we could fit an individual time series more quickly, thereby also potentially expanding the hyper-parameter space to help ensure we find the global optimum model and not a locally optimum one.

We can also consider adding additional features to our dataset, the current list of course being shown in Tab. 1. We can assess expanding certain groups of features. For instance, we have the population of a census Tract divided among four age groups. This grouping can be extended to more age groups. Additional datasources can also be explored. Data related to crime could also be quite useful as another time series model that we could add as exogenous variables. Historical weather might also help to better understand when certain service requests are made. Other datasources could also be explored.

So far we only explored ARIMA-based algorithms to model and fit time series models. There are additional methods we can try as well. For instance, while not shown in this project, random forests were also evaluated. It showed some promise but more effort is needed to validate and understand the method. Other tree-based algorithms should perhaps also be examined, in addition to other algorithms such as Support Vector Machine or Neural Networks.