Forecasting Parking Meter Transactions in San Diego

Stephen Reagin

Shiley-Marcos School of Engineering, University of San Diego

ADS 506 – Applied Time Series Analysis

Professor Dave Hurst, M.S.

December 11, 2023

Introduction

The City of San Diego manages approximately 5,700 metered parking spaces through the Office of the City Treasurer, which is responsible for the installation and maintenance of metered spaces as well as meter payment collections (City of San Diego, n.d.-b). Parking meter rates, length of stay limits, and hours of operations are designed to encourage turnover of limited parking spaces and to maximize space utilization. Enforcement of parking meter regulations is conducted through the San Diego Police Department, and the Economic Development Department acts as a liaison with the five local Community Parking Districts. This promotes local commerce because longer parking stays are financially disincentivized, allowing other commuters to use the parking space for engaging in nearby economic activity (Erickson, 2012).

Parking meter enforcement in San Diego was suspended in 2020 due to COVID-19 pandemic shutdowns and related citizen relief efforts. After the reinstatement of regular parking meter enforcement in February 2021, the number of daily parking meter transactions has declined from February 2021 to November 2023, which is the period of this study. It is currently unknown whether this is due to an increase in parking meter delinquency (i.e. more people parking without paying) or due to an actual change in parking meter utilization. If delinquencies are increasing, the City of San Diego likely requires more parking meter enforcement officers and/or a deeper investigation into the city's parking meter infrastructure and guiding philosophy.

Machine learning techniques such as time series modeling can assist the City in forecasting the expected number of parking meter transactions over a given 90-day horizon. If a trained model forecast is found to significantly deviate from the actual number of transactions, a case can be made that the City is not appropriately measuring its parking turnover nor its utilization target rate[1] of 85%, and further that

[1] According to the San Diego Municipal Code: "It is the intent of the City Council to establish a target utilization rate of 85 percent for all parking meters… based on well-accepted planning studies as well as the example of other municipalities... 85 percent is one of the most effective strategies for managing on-street parking and for recovering at least a portion of the estimated reasonable costs associated with parking and traffic control." (San Diego Municipal Code, 2016)

the City Treasury is losing revenue due to parking meter delinquency. With such tools, anomalous periods can be identified when parking meter transactions deviate significantly from expected values; thus deviations may signal that more enforcement is required, including the hiring of trained staff and increasing of Police Department budgets.

It is therefore the near-term goal of this research to build time series models best suited to forecast parking meter transactions over a 90-day and 180-day forecasting horizon. Success will be defined as a model which correctly forecasts 90 days of parking transactions volume within +/-5% of actual values and having RMSE less than 1685 (half a standard deviation).

Method

Motivation for Time Series Models

Time series models are especially well-suited to forecast parking meter activity into the near future, in part because these models can account for trends, seasons, cycles, and self-correlating behaviors that evolve over time. The forecasting horizon can be short or long, and forecasts can automatically roll-forward as new data comes in (Shmueli & Lichtendahl, pp. 20-22). The mandatory requirement is to utilize a variable or set of variables having an intrinsic temporal component, e.g. daily or weekly measurements. As described in the Dataset section, the parking meter transactions data and the external weather information used in this research both meet this requirement.

Dataset

The primary dataset comes directly from the City of San Diego Open Data Portal (2023), consisting of six CSV files with daily parking meter transactions from calendar January 2018 to November 2023. The secondary dataset comes from the National Oceanic and Atmospheric Administration (NOAA) free weather service API (National Oceanic and Atmospheric Administration, 2023), and it consists of daily precipitation and temperature average / minimum / maximum values. As described in the Data Preprocessing section, the itemized parking data points were transformed into

daily aggregates of transactions and dollar amounts and joined to the weather data points in a single data frame for analysis and modeling.

The final dataset has 1,030 observed days of data from January 31, 2021 to November 26, 2023, and consists of nine features:

- Date
- Precipitation (cm)
- Temperature (average, min, max, all in degrees Celsius)
- Number of daily transactions
- Sum of dollar amounts paid
- Binary indicator variables for Sundays and Holidays

Data Preprocessing

The original parking meter data is recorded as a new observation row for every parking meter and for every date. To forecast city-wide daily transactions these observations must be aggregated by date, which first requires pivoting the dataset so that each pole id is a unique column feature. After joining to the weather data (which ensures no days are skipped), null values are imputed with a 0 and the pole id transaction values are summed by date to create a daily aggregate of parking meter transactions. Most parking meters do not collect meter fares on Sundays and Holidays, so there are two binary indicator variables to flag Sundays and Holidays. The Holiday schedule consists of federal and state holidays, as well as special municipal occasions like parades and protests. When a public holiday falls on a weekend the indicator variable is set to the prior Friday or following Monday, in accordance with city practice.

As described in the Exploratory Data Analysis (EDA) section, the decision was made to remove all data values before January 31, 2021 due to COVID-19 pandemic shutdown relief efforts[2] affecting the regularity of data observations. Therefore, the final dataset consists of 1,030 daily observations of nine features as seen in Table 1. All data preprocessing steps in Python can be found in Appendix B.

Table 1

Sample Observations from the Final Dataset

	date	PRCP	TAVG	TMAX	TMIN	transactions	dollar_amount	isSunday	isHoliday
0	2021-01-31	0.0	12.7	20.0	7.2	210.0	83.20	1	0
1	2021-02-01	0.0	16.6	22.2	13.9	11008.0	12790.22	0	0
2	2021-02-02	0.0	16.8	19.4	14.4	11090.0	13520.16	0	0
3	2021-02-03	0.0	14.9	16.7	12.8	11658.0	13841.36	0	0
4	2021-02-04	0.0	15.2	17.2	11.7	11391.0	13908.27	0	0
...	...	...	...	...	...	...	...	...	...
1025	2023-11-22	0.0	17.2	25.6	10.0	17183.0	24783.07	0	0
1026	2023-11-23	0.0	16.2	20.0	10.6	181.0	46.20	0	1
1027	2023-11-24	0.3	17.4	18.9	12.8	13007.0	19177.49	0	0
1028	2023-11-25	0.0	15.0	19.4	10.0	15388.0	24139.33	0	0
1029	2023-11-26	0.0	13.9	21.7	7.8	190.0	48.75	1	0

1030 rows × 9 columns

Exploratory Data Analysis (EDA)

Figure 1 shows the volume of daily parking meter transactions from 2018 to 2023. The most notable detail is the significant change in parking meter activity as a direct result of COVID-19 relief efforts, including a stoppage of all parking meter fares in 2020. For the purposes of statistical analysis and time series modeling, the decision was made to remove all data before January 31, 2021. Starting from that date, there are noticeable seasonal dips from summer to winter and a steady decreasing trend in transactions volume. The decreasing trend in transactions volume is somewhat offset by the increasing trend in cost per transaction, as seen in Figure 2. This creates an overall flat trend in revenue.

[2] Parking meter enforcement was suspended by San Diego Mayor Faulconer as part of a broader citizen relief effort related to COVID-19 pandemic shutdowns from March 16, 2020 to October 1, 2020, and by his successor Mayor Todd Gloria from December 5, 2020 to February 8, 2021. See City of San Diego (2020) and City of San Diego (2021) for more information.

Figure 1

Daily Transactions Volume, 2018 – 2023

Figure 2

Weekly Moving Average of Cost per Transaction, Transactions Volume, and Total Revenue

Table 2 shows average descriptive statistics for transaction volume and dollar amounts for each day of the week. There are 14,685 average daily transactions with standard deviation of 7,341, or a daily average of 17,813 transactions if Sundays and Holidays are removed with standard deviation of 3,371.

Table 2

Descriptive Statistics of Parking Meter Activity by Weekday

Day	Average Transactions	Average Revenue	Cost per Transaction
Sunday	232	$81	$0.35
Monday	14,949	$20,687	$1.28
Tuesday	17,489	$24,527	$1.39
Wednesday	17,900	$25,301	$1.40
Thursday	17,502	$24,714	$1.37
Friday	18,173	$25,827	$1.38
Saturday	16,648	$24,924	$1.48

Figure 3 shows a weekly sum of daily transactions and there is a noticeable dip in activity for weeks with a Holiday, characterized by sharp downward spikes alongside the purple dots which represent Holidays. There is also an apparent association between large winter dips in transactions volume and a seasonal increase in precipitation (rainfall), but it remains an open question whether there is a causal factor linking these variables or if they both just happen to correlate with cold winter months. Figure 4 illustrates a potential linear relationship between average temperature and transactions volume.

Figure 3

Weekly Transactions Over Time, Alongside Weekly Precipitation and Holiday Flags,

Figure 4

Potential Linear Relationship Between Temperature and Transactions Volume

As described in the Time Series Models section for the ARIMA model, there are strong weekly patterns seen in the ACF of Figure 8, which means the dataset has a strong weekly seasonality component, most of which is accounted for by the lack of parking meter fares on Sunday. Figure 8 also shows a similar chart with Sundays removed, and the strong negative lag-1 autocorrelation indicates the difference between transactions on consecutive days is expected to swing up and down.

To recap the EDA section, multiple characteristics of the dataset were discovered: 1) There is a downward trend in transaction volume over time; 2) There are strong weekly seasonal patterns; 3) Holidays have a significant and predictable effect on meter transactions; and 4) There seems to be some association or potential relationship between meter transactions and weather patterns. Each of these characteristics was used as a feature or otherwise accounted for during the time series modeling phase.

Time Series Models

Multiple time series models were constructed to forecast parking meter transactions over a 180-day period, which includes a 90-day validation period (i.e. the actual answer is known and can be

compared against forecasts) and a 90-day future horizon forecast. All metrics are evaluated on forecasts over the validation period, and the additional future forecasts were generated both for visual inspections and to examine the behavior of longer-term forecasts. Each model uses a different mathematical framework with different philosophical assumptions, and model performance will help uncover attributes of the parking meter dataset in addition to generating forecasts.

There are two versions of every model: one with a dataset that includes all days, and one with a dataset that excludes Sundays and Holidays. The idea behind excluding Sundays and Holidays is to remove the known noise[3] arising from those days and hopefully strengthen the signal of other trends and seasons. However, this creates data gaps with missing dates and most time series methods do not handle gaps in dates very well, so the data is re-indexed to an ordered integer index variable, X={1,2,3,...,n}.

For all model figures in this section, Sundays and Holidays are the values near 0 which cause every left chart (using Training Set 1) to look like a sweeping curtain, and the right chart with Sundays and Holidays removed (Training Set 2) could possibly resemble something like a stock price.

The R code implementing these models is found in Appendix A. For simplicity of discussion, datasets with all days included are referred to as Dataset 1, Training Set 1, and Validation Set 1, and datasets with no Sundays or Holidays are referred to as Dataset 2, Training Set 2, and Validation Set 2.

Baseline Forecasts.

Baseline performance is established with three simple forecasting methods: Mean forecast, Naïve forecast, and Drift forecast. These simple models often serve as benchmarks to compare against more sophisticated forecasting models to ensure the new methods are better than simple alternatives.

The Mean forecast takes the arithmetic mean of the training data and forecasts the mean value into the forecast horizon. The Naïve forecast takes the very last value of the training set and forecasts that value into the forecast horizon. The Drift forecast is similar to a naïve forecast in that it starts with

[3] "Noise" is not *quite* the right word because these are extremely predictable outcomes—there's a Sunday every week!—but those Sunday and Holiday training points will end up generating noisy outcomes, as seen in the Results section.

the last value of the training data, except it adds a trendline based on the average change of the dataset, i.e. the slope of the difference between last observation and first observation (see Hyndman & Athanasopoulos, 2021, Ch 5.2: Some Simple Forecasting Methods).

Figures 5

Simple Forecast Models and 180-day Forecasts on Daily Transactions

Linear Regression.

Linear Regression models are built on the assumption that a weighted sum of predictor variables can reliably fit a linear target variable in a multidimensional array. The predictor variables themselves do not need to be linear functions, and time series Linear Regression model takes advantage of this by incorporating Trend and Seasonality as weighted predictors along with other external variables (Shmueli & Lichtendahl, pp, 117-126, 154-158). The Linear Regression model for parking meter transactions assumes there is some linear relationship between parking meter transactions and daily weather patterns, and it incorporates trend, seasonality, precipitation, and temperature as predictor variables, as modeled by Equation 1.

Equation 1

Linear Regression Equation for Parking Meter Transactions

$$y_t = \beta_0 + \beta_1 trend + \beta_2 season + \beta_3 rain + \beta_4 Tavg + \beta_5 Tmin + \epsilon_t$$

This model cannot make predictions outside of the validation period because weather information is not known ahead of time. That is to say, the validation dataset provides predictor information for the 90-day forecast window, and forecasts cannot be extended indefinitely into the future without a knowledge of future weather data points. While these weather data points can be imputed based on historical records, as a practical matter the general rule is that any external data must be available at the time of prediction (Shmueli & Lichtendahl, p. 158).

Figure 6

Linear Regression Models and 90-day Forecasts on Daily Transactions

ETS.

Error, Trend, and Seasonality (ETS) models use exponential smoothing to take a weighted average of training data values and the weights decrease exponentially into the past, meaning more recent information carries greater importance when generating forecasts. The simplest ETS models require data that has no trend or seasonality, assumptions which are violated by the parking meter data, and a more advanced version such as Holt-Winters' exponential smoothing can account for those additional data characteristics.

In the fable library for implementing ETS models, the error type, trend type, and seasonal type can all be specified as Additive (A), Multiplicative (M), None (N), or as damped versions (Ad, Md). Those specifications can also be selected automatically, relying on the combination which fits best

according to minimizing some information criterion such as Akaike Information Criteria (AIC). See the "Exponential smoothing state space model" (n.d.) reference in the fable library to learn more.

For Training Set 1, the selected ETS parameters are (A,N,A):

- Error type: Additive
- Trend type: None
- Seasonal type: Additive

For Training Set 2, the selected ETS parameters are (A,N,N).

Figure 7

ETS Models and 180-day Forecasts on Daily Transactions

ARIMA.

AutoRegressive Integrated Moving Average (ARIMA) methods directly model the autocorrelation of series values and the autocorrelations of forecast errors. Here, "autoregression" means forecasting future values by using past values of the variable, indicating the forecast is a regression against itself; and "autocorrelation" is the degree of correlation the variable has with itself across successive time intervals (Taylor, n.d.).

The Figure 8 autocorrelation chart (ACF) for Training Set 1 shows very strong weekly autocorrelations (lag=7, 14, etc.). The ACF for Training Set 2 shows a strong negative lag-1 autocorrelation, meaning successive data values are expected to swing from high to low and vice versa.

Figure 8

Autocorrelation Function Charts for Training Set 1 and Training Set 2

There are three parameters (p,d,q) to specify for the non-seasonal components of ARIMA models, and three similar parameters (P,D,Q) to specify for the seasonal components:

- p = number of autoregressive terms
- d = number of integrated terms (i.e. how many times the series is differenced)
- q = number of moving average terms

Similar to ETS models, the fable library can automatically select (p,d,q) + (P,D,Q) parameters. For Training Set 1 the automatically selected model is ARIMA(0,0,2)(0,1,1). For Training Set 2 the automatically selected model is ARIMA(4,1,2) with no seasonal component. See the "Estimate an ARIMA model" (n.d.) reference in the fable library documentation to learn more.

Figures 9

ARIMA Models and 180-day Forecasts on Daily Transactions

STL.

Seasonal and Trend decomposition using Loess (STL) is not itself a predictive model but rather a method for estimating nonlinear relationships by decomposing a time series dataset into components, e.g. weekly season + annual season + moving average trend (Hyndman & Athanasopoulos, 2021, Ch. 3.6: STL Decomposition). The adjusted output is then fed into a predictive model and renormalized after predictions are made. Shmueli and Lichtendahl (2018) describe the STL prediction process:

> Once the time series is decomposed, the seasonal component is subtracted from the time series to obtain a deseasonalized series. We then use exponential smoothing or ARIMA to generate forecasts for the deseasonalized series. Finally, the resulting forecasted values are reseasonalized by adding a seasonal naïve forecast using the seasonal component. (p. 104)

The tsibble framework naturally integrates STL methods with ETS and ARIMA models, so that it's unnecessary to manually subtract the seasonal component before forecasting and then reseasonalize after forecasting. For Training Set 1 the decomposed seasons were specified as 7 days (one week) and 365 days (one year). For Training Set 2, which has no Sundays or holidays, the decomposed seasons were specified as 6 days (one week) and 294 days (one year).

Figure 10

Seasonal Decomposition of Daily Transactions Volume Over Time

STL + ETS.

After decomposition as described in the main STL section, the de-trended and de-seasonalized results were fed into an ETS model to automatically select ETS parameters. For Training Set 1 the automatically selected parameters are (A,Ad,A):

- Error type: Additive (A)
- Trend type: Damped additive (Ad)
- Seasonal type: Additive (A)

For training set 2 the automatically selected parameters are (A,Ad,N).

Figure 11

(STL + ETS) Models and 180-day Forecasts on Daily Transactions

STL + ARIMA.

After decomposition as described in the main STL section, the de-trended and de-seasonalized results were fed into an ARIMA model to automatically select trend parameters (p,d,q) and seasonal parameters (P,D,Q). For Training Set 1 the parameters are ARIMA(2,1,3)(0,0,1) and for Training Set 2 the parameters are ARIMA(2,1,4) with no seasonal component.

Figure 12

(STL + ARIMA) Models and 180-day Forecasts on Daily Transactions

Results

Proper evaluation across multiple models depends on useful metrics. Baseline performance for parking meter time series models is established in Table 3. For each model there are two datasets, and for each dataset there is a Training Set and 90-day Validation Set. Daily errors are defined as the daily difference between forecast value and actual validation value. The metrics are:

- **90-day deviation**: this is a custom metric, the sum of forecasted values divided by the sum of actual values over the 90-day forecast period. This metric smooths out performance fluctuations over a time window rather than using day-by-day error measurements.
- **Mean Error (ME)**: average of all daily errors.
- **Root Mean Squared Error (RMSE)**: square root of the mean of the sum of squared errors.

- **Mean Absolute Error (MAE)**: average of all absolute value daily errors.
- **ACF1**: autocorrelation of errors at lag-1, i.e. correlation between consecutive errors

Table 3

Performance of Simple Models to Capture Baseline Performance

Baseline Mean	90 day dev.	ME	RMSE	MAE	ACF1
Training Set 1	N/A	0	7423	5946	0.161
Training Set 2	N/A	0	3539	1802	0.798
Validation Set 1	-5.80%	-809	6427	4338	-0.0134
Validation Set 2	-4.62%	-791	1336	1051	0.441

Baseline Naïve	90 day dev.	ME	RMSE	MAE	ACF1
Training Set 1	N/A	18	9608	5898	-0.377
Training Set 2	N/A	9	2236	1106	-0.288
Validation Set 1	-20.19%	-2816	6970	3580	-0.0134
Validation Set 2	-3.20%	-547	1208	937	0.441

Baseline Drift	90 day dev.	ME	RMSE	MAE	ACF1
Training Set 1	N/A	0	9608	5894	-0.377
Training Set 2	N/A	0	2236	1104	-0.288
Validation Set 1	-25.94%	-3618	7386	3941	0.00278
Validation Set 2	-5.56%	-951	1530	1217	0.537

Time Series model results are shown in Table 4. Three models using Training Set 2 meet the success criteria of +/-5% forecasting of total transactions volume over a 90-day period and having RMSE less than 1685 (half a standard deviation), as do the Baseline Mean and Naïve models:

- Linear Regression
- STL + ETS
- STL + ARIMA

Models built on Training Set 2 tend to significantly outperform models built on Training Set 1. This is not surprising, since Sundays and holidays are an additional source of fluctuations which a model incorporates to make forecasts. But it's also the case that Training Set 1 models tend to overshoot Sunday forecasts and make predictions of negative parking meter transactions, which is obviously not relevant to the real world (unless the City starts paying people to park). Therefore, the remaining analysis of results will focus exclusively on Training Set 2 models.

Table 4

Measured Performance Results of Times Series Models

Linear Regression	90 day dev.	ME	RMSE	MAE	ACF1
Training Set 1	N/A	0	3110	1545	0.573
Training Set 2	N/A	0	2985	1431	0.786
Validation Set 1	1.48%	206	1670	858	0.144
Validation Set 2	0.73%	124	867	612	0.427
ARIMA	**90 day dev.**	**ME**	**RMSE**	**MAE**	**ACF1**
Training Set 1	N/A	48.9	3902	2107	0.0345
Training Set 2	N/A	15.8	2002	984	-0.012
Validation Set 1	-5.07%	-707	3039	981	0.117
Validation Set 2	-25.88%	-450	1157	887	0.438
ETS	**90 day dev.**	**ME**	**RMSE**	**MAE**	**ACF1**
Training Set 1	N/A	12.8	3741	1909	0.0929
Training Set 2	N/A	17.1	2067	1133	0.0941
Validation Set 1	-11.25%	-1569	3338	1625	0.12
Validation Set 2	-25.58%	-408	1152	879	0.441
STL + ETS	**90 day dev.**	**ME**	**RMSE**	**MAE**	**ACF1**
Training Set 1	N/A	-104	4682	1995	0.223
Training Set 2	N/A	-125	3401	1458	0.715
Validation Set 1	1.85%	3418	7518	4072	0.974
Validation Set 2	2.82%	483	1511	1278	0.257
STL + ARIMA	**90 day dev.**	**ME**	**RMSE**	**MAE**	**ACF1**
Training Set 1	N/A	-128	4646	1979	0.17
Training Set 2	N/A	-122	3351	1440	0.69
Validation Set 1	1.82%	253	4204	1641	-0.429
Validation Set 2	2.82%	483	1511	1274	0.256

Models built on Training Set 2 tend to significantly outperform models built on Training Set 1. This is not surprising, since Sundays and holidays are an additional source of fluctuations which a model incorporates to make forecasts. But it's also the case that Training Set 1 models tend to overshoot Sunday forecasts and make predictions of negative parking meter transactions, which is obviously not relevant to the real world (unless the City starts paying people to park). Therefore, the remaining analysis of results will focus exclusively on Training Set 2 models.

Discussion

The Linear Regression model has the best 90-day deviation of all models, and it also outperforms on every quantitative metric in addition to its forecast chart passing the "eye" test (meaning, it *looks* like a good fit in Figure 6). However, the Linear Regression model requires its predictors to be known ahead

of time, specifically meaning temperature and precipitation data, and daily weather information cannot be reliably forecasted beyond the next 10 days (Zhang et al., 2019). Thus, it can only be used for short-term periods and is better as a model to review historical deviations from expected parking meter transactions rather than forecasting future values. Also, while the weather data seems to be a useful indicator variable in the context of Linear Regression, it's unclear whether the association with parking meter transactions is causal or just correlative.

Incorporating seasonal decomposition (STL) significantly improves both ARIMA and ETS forecasting. This suggests two characteristics of parking meter behavior: 1) There is a reliable long-term trend in parking meter activity; and 2) There are reliable cycles/seasons operating on weekly and annual timescales. These are important details to help inform the mid-range and long-term workforce planning of parking meter enforcement staff. This finding also aligns with other results in Table 4 showing these models perform best when Sundays and Holidays are removed, because those days represent (weekly seasonal) times when little-to-no parking meter transactions are enforced by the City. In other words, datasets with fewer fluctuations tend to be easier to forecast.

The STL + ARIMA and STL + ETS models have very similar error metrics, and the forecast charts (Figures 11 and 12) also look nearly identical with minor deviations. It's possible that a deeper analysis could show mathematical convergence of the models with their respective parameters, or maybe both are accurately predicting an identical future. But it's also worth mentioning that both models forecast another January shutdown of parking meter transactions, which would only occur in real life because of regulatory actions rather than a predictable trend in commuter behavior.

Conclusion

Parking meter transactions can be reliably forecasted up to at least 90 days by multiple time series models based on different mathematical tools and philosophical frameworks. This suggests there are actual patterns in commuter behavior that can be captured by a data model to generate forecasts in

parking meter activity, and that significant deviations from that prediction could be a sign of widespread parking meter delinquency. Historical patterns can be used to identify trends and seasons, such that time of year and recent parking meter activity will inform an accurate predictive forecasting model.

The simple descriptive statistics may also provide direction for additional staffing; for example, Saturdays could have the most incentive for delinquency because of high average cost per transaction, plus a lackadaisical commuter attitude to not pay because "it's the weekend," and thus Saturdays may require more enforcement staffing. The time series models described here can help identify those periods of the year which require most staffing and provide evidential support for a part-time or seasonal parking enforcement staff. Salary for a Parking Enforcement Officer I or II ranges from $48K to $64K per year (City of San Diego, n.d.-a), so it may ultimately cost the City more than it recoups via fines and citations to have too many full-time enforcement officers on hand.

One potential issue is: the time series models may already be indirectly modeling a growing delinquency that currently exists. If so, then it does no good as a future delinquency indicator. It would be informative to incorporate knowledge of parking citations that have been issued over the period of this study to determine if there's any trend in parking meter violations. Another issue is: this research does not take into consideration the count of operating parking meters and how that number may have increased or decreased over time because the City does not directly publish that information.

A deeper analysis would ideally incorporate more geographic details and time granularity to make forecasts with more precision. For example, inclusion of hourly transaction data by pole id could be paired with the latitude and longitude of each pole, thus modeling peak times of the day and peak neighborhoods with the most transactions volume. A smaller, targeted parking enforcement staff could then be deployed with greater effectiveness. That said, such models would also need to account for periods of sparsity and the stochastic nature of parking meter transactions for individual meters.

This research represents the first step towards understanding the evolution of parking meter transactions over time, and the successful prediction models can be expanded into multiple lines of continuing research to provide a greater infrastructure planning benefit to the City of San Diego.

References

City of San Diego Job. (n.d.-a). *City of San Diego Job Opportunities Page, Job Descriptions.* City of San Diego. https://www.governmentjobs.com/careers/sandiego/classspecs?keywords=parking%20enforcement%20officer

City of San Diego. (n.d.-b). *Parking Meter Operations.* City of San Diego. https://www.sandiego.gov/parking/meterops

City of San Diego. (2020, August 31). *Mayor Faulconer Delays Full Parking Enforcement Until Oct. 1* [Press release]. https://www.sandiego.gov/sites/default/files/2020-8-31_parking_enforcement.pdf

City of San Diego. (2021, January 28). *City of San Diego to Restart Parking Enforcement Following Decision to Lift Regional Stay-at-Home Order* [Press release]. https://www.sandiego.gov/sites/default/files/2021-01-28_parking_enforcement.pdf

City of San Diego Open Data Portal. (2023). *Parking Meters Transactions* [Data set] https://data.sandiego.gov/datasets/parking-meters-transactions/

Erickson, A. (2012, April 3). *A Brief History of the Parking Meter.* Bloomberg. https://www.bloomberg.com/news/articles/2012-04-03/a-brief-history-of-the-parking-meter

Estimate an ARIMA model. (n.d.) In *fable library* reference documentation. https://fable.tidyverts.org/reference/ARIMA.html

Exponential smoothing state space model. (n.d.). In *fable library* reference documentation. https://fable.tidyverts.org/reference/ETS.html

Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp3/simple-methods.html

National Oceanic and Atmospheric Administration. (2023). *Climate Data Online: Web Services Documentation*. https://www.ncdc.noaa.gov/cdo-web/webservices/v2

San Diego Municipal Code. (2016, July 22). Article 6, Division 1: General Parking Regulations, Chapter 8: Traffic and Vehicles, San Diego Municipal Code § 86.0123 https://docs.sandiego.gov/municode/MuniCodeChapter08/Ch08Art06Division01.pdf

Shmueli, G. & Lichtendahl Jr., K.C. (2018). *Practical time series forecasting with R: A hands-on guide* (2nd ed.). Axelrod Schnall Publishers.

Taylor, S. (n.d.). *Autocorrelation*. Corporate Finance Institute. https://corporatefinanceinstitute.com/resources/data-science/autocorrelation/#

Zhang, F., Sun, Y. Q., Magnusson, L., Buizza, R., Lin, S., Chen, J., & Emanuel, K. (2019). *What Is the Predictability Limit of Midlatitude Weather?* Journal of the Atmospheric Sciences, 76(4), 1077-1091. https://doi.org/10.1175/JAS-D-18-0269.1

Appendix A: Final Report Models

Stephen Haslett
2022-06-30

This file contains R code to create parking meter activity using various time series models:

- Linear Regression
- ETS
- ETS decomposition (stuctting)
- ARIMA (Autoregressive integrated moving average)
- STL + ARIMA (seasonal decomposition with ARIMA)
- Prophet (Facebook)
  - Base forecast
  - Prophet boxplot
  - CV forecast

The data is divided into two sections which correspond with two datasets:

- All days (include)
- Sundays and holidays excluded

```
setwd("/Users/shaslett/Desktop/PhD/SBD_PhD_Final_project/Final_project/Datasets")
```

```
library(fixtures)
library(tidyverse)
library(ggplot2)
library(ggfortify)
library(dplyr)
library(tsibble)
library(fable)
library(fabletools)
library(feasts)
library(tsibbledata)
library(lubridate)
library(ggseasonplot)
library(ggtsdiag)
library(forecast)
library(tseries)
library(urca)
library(fpp2)
library(fpp3)
library(tsibble)
library(tsibbledata)
library(tsibble)
library(tsibbledata)
library(tsibble)
library(tsibbledata
```

```
DFA_ETS = DFA_ETS(T,T[['Consumption']],
                  ETSkind=ETS.ANN,
                  ETSseasonal=ETS.ADD,
                  ETSdamped=False,
                  seasonal=365,
                  ma=365,
                  verbose=True)
DFA_ETS.fit(optimized=True,use_brute=True,use_boxcox=False,disp=0)
print(DFA_ETS.summary())
DFA_ETS.forecast(steps=365,alpha=0.05).plot(ax=ax,label='ETS',legend=True)
```

```
ETS Model (ANN) 100-day forecast training set 2
```

```
statsmodels.ETS(T[['Consumption']],seasonal=365,initialization_method='heuristic')
```

```
dfa = y_t.dropna()
```

```
statsmodels.ETS(dfa)
```

```
fit = statsmodels.fit()
print(fit.summary())
statsmodels.tsa.exponential_smoothing.ExponentialSmoothing.HoltWinters.HoltWintersResults.summary(fit)
```

```
statsmodels.ETS(dfa,trend='add',seasonal='add').fit().summary()
```

```
fit = statsmodels.ETS(dfa,trend='add',seasonal='add').fit()
print(fit.summary())
statsmodels.tsa.exponential_smoothing.ExponentialSmoothing.ExponentialSmoothingResults.summary(fit)
```

STL + ETS

Decomposition of the All Days dataset

```
T.Consumption.ewm(span=365).mean().plot(ax=ax,legend=True,label='Consumption')
T.Consumption.ewm(span=365).mean().diff().plot(ax=ax,legend=True,label='Consumption')
```


Decomposition of the No Sundays, No Holidays dataset

```
T.Consumption.ewm(span=365).mean().plot(ax=ax,legend=True,label='Consumption')
T.Consumption.ewm(span=365).mean().diff().plot(ax=ax,legend=True,label='Consumption')
```

STL + ETS - All Days

```
DFA_ETS = statsmodels.DFA.DFA(ETSkind=ETS.ANN,seasonal=365,trend=0,resamp=False,interp=False)
DFA_ETS.fit(y=T[['Consumption']],cutoff=T[:'2006-12-31'].shape[0],nlag=365,verbose=True)
print(DFA_ETS)
```

```
statsmodels.DFA.DFA(ETSkind=ETS.ANN,seasonal=365,trend=0,resamp=False,interp=False)
```

```
DFA_ETS = statsmodels.DFA.DFA(ETSkind=ETS.ANN,seasonal=365,trend=0,resamp=False,interp=False)
DFA_ETS.fit(y=T[['Consumption']],cutoff=T[:'2006-12-31'].shape[0],nlag=365,verbose=True)
DFA_ETS.forecast(steps=365,alpha=0.05).plot(ax=ax,legend=True,label='ETS')
```

```
accuracy(fit)
```

```
## # A tibble: 3 × 10
##   .model .type          ME  RMSE   MAE  MPE  MAPE  MASE RMSSE   ACF1
##   <chr>  <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 mean   Training -5.70e-12 3539. 1803.  -Inf   Inf 1.43  1.58 0.798
## 2 naive  Training  8.69e+ 0 2236. 1106.  -Inf   Inf 1     1    -0.288
## 3 drift  Training  4.67e-13 2236. 1104.  -Inf   Inf 0.998 1.00 -0.288
```

```
accuracy(fc[1:90],, valid_nonseqmial_df)
```

```
## # A tibble: 3 × 10
##   .model .type   ME  RMSE   MAE  MPE  MAPE  MASE RMSSE  ACF1
##   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 mean   Test  -791. 1336. 1051. -5.06  6.45  NaN   NaN 0.441
```

```
sum(fc5.mean[1:90]) / sum(valid_nonseqmial_df$transactions)
```

```
## [1] 1.04237
```

```
accuracy(fc[181:270],, valid_nonseqmial_df)
```

```
## # A tibble: 3 × 10
##   .model .type   ME  RMSE   MAE  MPE  MAPE  MASE RMSSE  ACF1
##   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 mean   Test  -547. 1208.  937. -2.63  5.73  NaN   NaN 0.441
```

```
sum(fc5.mean[181:270]) / sum(valid_nonseqmial_df$transactions)
```

```
## [1] 1.03197
```

```
accuracy(fc[361:450],, valid_nonseqmial_df)
```

```
## # A tibble: 3 × 10
##   .model .type   ME  RMSE   MAE  MPE  MAPE  MASE RMSSE  ACF1
##   <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 drift  Test  -951. 1530. 1217. -6.04  7.46  NaN   NaN 0.537
```

```
sum(fc5.mean[361:450]) / sum(valid_nonseqmial_df$transactions)
```

```
## [1] 1.055602
```

Additional visuals

Weekly Transactions, Holidays, Precipitation

```
library(xts)

num_transactions.ts <- ts(transactions_df$transactions, start=c(2020,295), freq=365)

#creating time series vectors
total_transactions.xts <- xts(transactions_df$transactions, order.by = transactions_df$date)
isHoliday.xts <- xts(transactions_df$isHoliday, order.by = transactions_df$date)
precip.xts <- xts(transactions_df$PRCP, order.by = transactions_df$date)
tavg.xts <- xts(transactions_df$TAVG, order.by = transactions_df$date)


weekly_transactions.xts <- apply.weekly(total_transactions.xts, sum)
weekly_precip.xts <- apply.weekly(precip.xts, sum)
weekly_tavg.xts <- apply.weekly(tavg.xts, mean)

#weekly transactions with holidays
plot.xts(weekly_transactions.xts, main='Weekly Transactions over Time',ylim=c(0,150000))
```

```
lines(weekly_precip.xts * 1000 + 20000, col='blue', lty=2, lwd=1)
```

```
points(isHoliday.xts * 100000-1, pch = 20, col='purple')
```

```
addLegend("topleft",c('Weekly transactions', 'Rainfall', 'Holiday'), lty=c(1,2,3),
          col=c('black','blue','purple'), lwd=c(1,1,2))
```

Relationship with Average Temperature

```
transactions_tsibble |> ggplot(aes(x=TAVG, y=transactions)) + geom_point() +
  geom_smooth(method = 'lm', se=FALSE) + labs(title='Transactions as a function of Temperature (C)')
```

```
nonseqmial_df |> ggplot(aes(x=TAVG, y=transactions)) + geom_point() +
  geom_smooth(method = 'lm', se=FALSE) + labs(title = 'Transactions as a function of Temperature (Sundays & Holid
ays removed)')
```

ACF and Difference plots

ACF - All Days

```
train_transactions_tsibble |> ACF(difference(transactions), lag_max = 180) |> autoplot() +
  labs(title = 'ACF for Training Set 1')
```

```
train_transactions_tsibble |> autoplot(difference(transactions, lag=7)) +
  labs(title = "Training Set 1, lag-7 difference")
```

ACF - No Sundays, No Holidays

```
train_nonseqmial_df |> ACF(difference(transactions), lag_max = 180) |> autoplot() +
  labs(title = 'ACF for Training Set 2')
```

```
train_nonseqmial_df |> autoplot(difference(transactions, lag=1)) +
  labs(title = "Training Set 1, lag-7 difference")
```

Appendix B

Transforming Parking Meter CSV files

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
pd.set_option('display.max_rows', 5)
```

```
for i in range(0, len(df_2011.ts)):
    df_2011.loc[i, 'parking_meter_id'] = df_2011.loc[i, 'meter_id'].split('-')[0]
    df_2011.loc[i, 'meter_id'] = df_2011.loc[i, 'meter_id'].split('-')[1]
    df_2011.loc[i, 'ts'] = df_2011.loc[i, 'ts'].split('.')[0]
    df_2011.loc[i, 'ts_date'] = df_2011.loc[i, 'ts'].split(' ')[0]
    df_2011.loc[i, 'ts_time'] = df_2011.loc[i, 'ts'].split(' ')[1]
    df_2011.loc[i, 'ts_year'] = df_2011.loc[i, 'ts'].split('-')[0]
    df_2011.loc[i, 'ts_month'] = df_2011.loc[i, 'ts'].split('-')[1]
    df_2011.loc[i, 'ts_day'] = df_2011.loc[i, 'ts'].split('-')[2]
    df_2011.loc[i, 'ts_hour'] = df_2011.loc[i, 'ts'].split(':')[0]
    df_2011.loc[i, 'ts_min'] = df_2011.loc[i, 'ts'].split(':')[1]
    df_2011.loc[i, 'ts_sec'] = df_2011.loc[i, 'ts'].split(':')[2]
    df_2011.loc[i, 'ts_year_month'] = df_2011.loc[i, 'ts_year'] + '-' + df_2011.loc[i, 'ts_month']
    df_2011.loc[i, 'ts_date_time'] = df_2011.loc[i, 'ts_date'] + ' ' + df_2011.loc[i, 'ts_time']
    df_2011.loc[i, 'ts_year_int'] = int(df_2011.loc[i, 'ts_year'])
    df_2011.loc[i, 'ts_month_int'] = int(df_2011.loc[i, 'ts_month'])
    df_2011.loc[i, 'ts_day_int'] = int(df_2011.loc[i, 'ts_day'])
    df_2011.loc[i, 'ts_hour_int'] = int(df_2011.loc[i, 'ts_hour'])
    df_2011.loc[i, 'ts_min_int'] = int(df_2011.loc[i, 'ts_min'])
    df_2011.loc[i, 'ts_sec_int'] = int(df_2011.loc[i, 'ts_sec'])
    df_2011.loc[i, 'ts_date_time_int'] = int(df_2011.loc[i, 'ts_date_time'].replace('-', '').replace(':', '').replace(' ', ''))
    df_2011.loc[i, 'ts_year_month_int'] = int(df_2011.loc[i, 'ts_year_month'].replace('-', ''))
```

In [38]:
```
#sns.histplot(percapita_df['total'],bins=100)
#plt.title("I don't think this is correct")
```

Rolling averages

In [39]:
```
plt.figure(figsize=(12,3))
sns.lineplot(x=transactions_df['date'],y=transactions_df['transactions'].rolling(7,center=True).mean())
ax2 = sns.scatterplot(x=transactions_df['date'], y = transactions_df['isHoliday'] * 20000)

plt.xticks(rotation=60);
```

In [40]:
```
plt.figure(figsize=(12,3))
ax1 = sns.lineplot(x=amount_df['date'],y=amount_df['transactions'].rolling(1, center=True).mean())
#ax2 = sns.scatterplot(x=amount_df['date'], y = amount_df['isHoliday'] * 2500000)
plt.xticks(rotation=60);
```

In [41]:
```
#plt.figure(figsize=(12,3))
#sns.lineplot(x=amount_df['date'],y=amount_df['total'].rolling(7).mean())
#plt.xticks(rotation=60);
```

Differencing

In [42]:
```
plt.figure(figsize=(12,3))
sns.lineplot( x=transactions_df['date'], y= transactions_df['transactions'].diff(7))
```

Out[42]:
```
<Axes: xlabel='date', ylabel='transactions'>
```

In [43]:
```
plt.figure(figsize=(12,3))
sns.lineplot( x=amount_df['date'], y= amount_df['transactions'].diff(7))
```

Out[43]:
```
<Axes: xlabel='date', ylabel='transactions'>
```

In [ ]:

In [44]:
```
transactions_df.to_csv('../datasets/daily_total_transactions.csv')

transactions_df
```

Out[44]:

	date	PRCP	TAVG	TMAX	TMIN	transactions	dollar_amount	isSunday	isHoliday
0	2021-01-31	0.0	12.7	20.0	7.2	210.0	83.20	1	0
1	2021-02-01	0.0	16.6	22.2	13.9	11008.0	12790.22	0	0
2	2021-02-02	0.0	16.8	19.4	14.4	11090.0	13520.16	0	0
3	2021-02-03	0.0	14.9	16.7	12.8	11658.0	13841.36	0	0
4	2021-02-04	0.0	15.2	17.2	11.7	11391.0	13908.27	0	0
...	...	...	...	...	...	...	...	...	...
1025	2023-11-22	0.0	17.2	25.6	10.0	17183.0	24783.07	0	0
1026	2023-11-23	0.0	16.2	20.0	10.6	181.0	46.20	0	1
1027	2023-11-24	0.3	17.4	18.9	12.8	13007.0	19177.49	0	0
1028	2023-11-25	0.0	15.0	19.4	10.0	15388.0	24139.33	0	0
1029	2023-11-26	0.0	13.9	21.7	7.8	190.0	48.75	1	0

1030 rows × 9 columns

In [45]:
```
amount_df.to_csv('../datasets/daily_total_amount.csv')

amount_df
```

Out[45]:

	date	PRCP	TAVG	TMAX	TMIN	transactions	isSunday	isHoliday
0	2021-01-31	0.0	12.7	20.0	7.2	83.20	1	0
1	2021-02-01	0.0	16.6	22.2	13.9	12790.22	0	0
2	2021-02-02	0.0	16.8	19.4	14.4	13520.16	0	0
3	2021-02-03	0.0	14.9	16.7	12.8	13841.36	0	0
4	2021-02-04	0.0	15.2	17.2	11.7	13908.27	0	0
...	...	...	...	...	...	...	...	...
1025	2023-11-22	0.0	17.2	25.6	10.0	24783.07	0	0
1026	2023-11-23	0.0	16.2	20.0	10.6	46.20	0	1
1027	2023-11-24	0.3	17.4	18.9	12.8	19177.49	0	0
1028	2023-11-25	0.0	15.0	19.4	10.0	24139.33	0	0
1029	2023-11-26	0.0	13.9	21.7	7.8	48.75	1	0

1030 rows × 8 columns

In [ ]:

In [46]:
```
amount_df['avg'] = (amount_df['transactions'] / transactions_df['transactions'])
transactions_df['avg'] = (amount_df['transactions'] / transactions_df['transactions'])
```

In [47]:
```
temp_df = transactions_df[(transactions_df['isSunday'] == 0) & (transactions_df['isHoliday']==0)]
```

In [48]:
```
fig, axes = plt.subplots(3,1, figsize=(12,6), sharex=True)
fig.suptitle("Weekly moving average: Cost per Transaction / Daily Transactions / Daily Revenue")
labels=["Avg cost", "Transactions","Revenue"]
axes[0].set_ylabel('Average Cost')
axes[1].set_ylabel(labels[1])
axes[2].set_ylabel(labels[2])

axes[0].axvline(pd.to_datetime("2022-01-01"), color='lightgray', linestyle='dotted')
axes[1].axvline(pd.to_datetime("2022-01-01"), color='lightgray', linestyle='dotted')
axes[2].axvline(pd.to_datetime("2022-01-01"), color='lightgray', linestyle='dotted')

axes[0].axvline(pd.to_datetime("2023-01-01"), color='lightgray', linestyle='dotted')
axes[1].axvline(pd.to_datetime("2023-01-01"), color='lightgray', linestyle='dotted')
axes[2].axvline(pd.to_datetime("2023-01-01"), color='lightgray', linestyle='dotted')

ax1 = sns.lineplot(x=temp_df['date'], y=temp_df['avg'].rolling(6).mean(), ax=axes[0], color='orange')
ax2 = sns.lineplot(x=temp_df['date'], y=temp_df['transactions'].rolling(6).mean(), ax=axes[1])
ax3 = sns.lineplot(x=temp_df['date'], y=temp_df['dollar_amount'].rolling(6).mean(), ax=axes[2], color='green')
plt.xticks(rotation=60)
#fig.axvline(pd.to_datetime("2022-01-01"))
plt.xlabel("Date");
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [54]:
```
np.std(transactions_df['transactions'])
```

Out[54]:
```
7341.041817404638
```

In [ ]: