Analysis of California Cannabis Market

Final Project for CS M148 at UCLA

Nicholas Darci-Maher

UID #504924547

December 1, 2021

Executive Summary

The California cannabis market is a fast-growing industry evolving quickly over time under the influence of multiple complex factors. Some marijuana brands make more money than others, but what is it that sets those brands apart from the rest?

In this report, I document an analysis of a large dataset representing monthly sales data and brand characteristics across thousands of cannabis brands in California dating back to 2018. To generate this dataset, I merged information from multiple sources, matching brand sales metrics to brand characteristics which were aggregated from product-level data.

First, I uncovered correlations and substructure in the data using raw correlations and dimensionality reduction techniques. This information is important for highlighting important features off the bat and flagging potential collinearities in the selected features.

Next, I applied a series of machine learning-based prediction models to the data, with the goal of predicting future sales data based on past sales and aggregate brand characteristics. I applied a standard linear regression, a random forest regressor, an L1-regularized LASSO regression, an ElasticNet regression fit on the dimension-reduced data, and a neural network.

I found that the neural network performed the best out of these methods, and was able to make relatively accurate predictions for what the future sales of a brand would look like based on past sales of every brand in the market and the types of products sold by that brand.

I also found that historical sales were by far the strongest predictor of future sales, but some brand characteristics were more important than others, namely the proportion of “flower” products sold by the brand and whether or not the brand sold its products through generic vendors.

This analysis still leaves much discovery to future study, but indicates the ability of machine learning models to predict cannabis sales, and identifies some important factors that can contribute to a brand’s success in the market.

Introduction

Since California's legalization of recreational marijuana in 2016, a massive cannabis industry has developed in the state. In 2021 alone, BDSA predicts that total cannabis sales will top $17 million. Legalization has also brought the advent of new technologies and experimentation to marijuana—shoppers at a typical dispensary can choose from marijuana in its traditional bud form ("flower"), edibles, vapes, extracts, and even marijuana-infused drinks. This is a profitable and fast-growing field, and many companies are already becoming wealthy from the sale of marijuana. However, the forces that drive this new and dynamic market are still largely unstudied.

A businessperson in the marijuana sector might wonder whether they should focus on high-end luxury products or simpler, cheaper products. They might wonder whether they should be picky about the stores that sell their product, or get it onto as many shelves as possible. Importantly, they might wonder how much sales volume they should expect for the rest of the year based on how their products have been selling in recent months.

Here, I present a comprehensive analysis of monthly sales data in the cannabis industry in California that attempts to address these kinds of questions. I apply a host of regression and machine learning-based methods to predict future sales based on historical sales data and brand characteristics, and I examine the results of these models to determine important factors contributing to a cannabis brand's financial success.

Methodology

Generate time series features

To augment features for each brand that would leverage the time-dependent nature of the data, I first merged the Average Retail Price (ARP), units, and sales information into a single dataframe, using brand name and month as the key. Next, I generated time-dependent features. For ARP, units, and sales, I generated both a rolling average and a previous month column. Each of these features was calculated independently for each brand, and the results were concatenated together.

The rolling average took the mean of the previous 3 time points recorded for a given brand, allowing one or two time points to be used in the calculation when there were only so

many rows above the row being calculated. The previous month column took the values from only the previous month. I defined "previous" as the most recent time points, regardless of the interval between time points, meaning I assumed that in most cases there was month-level data in the table for a continuous stretch of months. After performing this augmentation, each brand's earliest time point had a missing value for the rolling average and previous month columns. I dropped these rows from the data.

Aggregate product-level details to brand-level details

To condense the "BrandDetails" table from information on each individual product to information on each brand, I performed feature aggregation. First, I manually selected a subset of the original features in the data based on low missingness and relevance to my research question. Of these selected features, I had some numerical features and some categorical features. For each numerical feature, I took the mean of that feature across all the products offered by a given brand to calculate the brand's score for that feature. For each categorical feature, I calculated the proportions of the top 5 most frequent values for the feature, out of the non-missing values in that feature. This, along with an 'other' column representing the proportion of other values besides the top 5, meant that each categorical feature was converted to 6 aggregated features per brand.

Augment features

To highlight interactions between key features, I generated two feature crosses that were used downstream. First, I crossed mean Tetrahydrocannabinol (THC) content with mean mood effect. This interaction term was intended to highlight brands that had a high concentration of potent marijuana products, assuming that a product with high total THC and a mood-specific effect would be on the stronger side. Second, I crossed mean generic vendor with flower proportion. This was meant to capture brands that "keep it simple," selling marijuana in the way it's been consumed for the longest amount of time and getting it into as many storefronts as possible.

Prepare dataset for modeling

After preparing the time-related data and the details for each brand, I merged these two tables to create one matrix of all my features and all my observations. Each row represented a brand at a unique time point, measured by month.

I used two strategies for handling missing data. Features that were missing a substantial amount of observations (approximately more than 5,000) I dropped from the dataframe entirely. Once these were removed, approximately 3,000 samples had missing data in one or more of the remaining features. These observations I dropped from the dataframe. After this, I was left with a matrix of 20,313 observations across 36 features.

All 36 of these features were numeric, and I scaled each of them to have mean zero and unit variance.

From this final cleaned matrix, I extracted my response variable, total sales, and generated my design matrix by removing total sales as well as some features with high collinearity.

Correlate design matrix features

I performed pairwise Pearson correlation between every feature in the design matrix, along with the response variable. The correlations were visualized in a heatmap.

Perform dimensionality reduction on the data

Before fitting any models, I ran Principal Components Analysis (PCA) on the scaled and centered data to reduce the dimensionality of my dataset. Based on the variance-explained elbow plot, I decided that 10 PCs was sufficient to capture the vast majority of variance in the data.

Train models to predict total sales in future months

To come up with a model that could predict future sales given historical sales data, I experimented with a variety of models. I trained 5 models to predict future sales: a standard linear regression, a random forest regressor, an L1-regularized LASSO regression, an ElasticNet regression fit on the first 10 PCs, and a neural network.

In each model, I followed the same training method. I fit the model to a design matrix with paired labels from a training set, which represented historical data. Then, I predicted the labels of a test set, given only the design matrix. The test set represented future data and was from time points later than all time points included in the training data. Finally, I evaluated the performance of each model by comparing the true test set labels to the predicted test set labels.

Cross-validate model performance

Each model was evaluated using a time series-specific 5-fold cross validation. Each fold included a training set of historical data, with dates up to a certain cutoff point, and a test set of future data, with dates after the cutoff point. The test set size was kept constant across folds. In the random forest, ElasticNet, and neural network, a grid search of input parameters was conducted to find the optimal combination. The metric used to compare parameters was negative mean squared error, and the final cross-validation output metric was the mean negative mean squared error across folds.

Select features most relevant to prediction

To find the features that contributed most to the prediction, I examined the p-values and coefficients from the standard linear regression and the LASSO regression. Features with high coefficients and low p-values were taken to be the most important, and features with low (or 0) coefficients were taken to be the least important.

Results

Summary of methodology

To generate predictive models that would forecast future brand sales, I wrangled time-dependent sales data and brand metadata into a matrix of important features. I cleaned this feature data and scaled it, to avoid bias due to differences in variance across the features. I explored the structure of the data using pairwise correlations and PCA. I then leveraged a toolkit of regression and machine learning-based methods to predict future sales based on historical sales and brand characteristics. From these models, I gained an understanding of the most important factors contributing to the monetary success of a brand. Finally, I chose the most effective model based on comparative cross-validation.

Certain features are highly correlated with total sales

As my first look into the structure of the data, I examined the correlations between features and the response variable (Figure 1). The strongest positive correlations with total sales were some of my time series features: the rolling averages and previous month data for total sales and total units ($r > 0.85$). Among the strongest negative correlations with total sales were the proportion of accessories, proportion of topicals, and proportion of ingestibles, though these

negative correlations were weak (-0.1 < r). Interestingly, both the generic vendor feature and the cross between generic vendor and flower proportion were highly correlated with total sales compared to other brand-level metadata (r = 0.31 and 0.72, respectively).

Principal components capture variation in total sales

To take a deeper look at the substructure of the data, I performed PCA on the design matrix. The first PC captured over 21% of the variance in the data, and the first 10 PCs combined captured over 99% of the variance in the data (Figure 2). To explore how effective the PCs were in stratifying the data along total sales, I experimented with different 2-dimensional visualizations of various combinations of PCs. I found that PC4 and PC5 effectively separate time points where brands were making a lot of money from time points where brands were making less money (Figure 3).

Future sales are effectively predicted by a neural network

After learning about the structure of the data I was working with, I reasoned that future brand sales could be predicted by my features, which included historical sales data and brand-level characteristics.

I attempted this prediction with five different models: a standard linear regression, a random forest regressor, an L1-regularized LASSO regression, an ElasticNet regression fit on the first 10 PCs, and a neural network. After performing 5-fold cross validation and a grid search for optimal parameters (on relevant models), I found that the neural network predicted future sales with the lowest error (Table 1). However, the random forest, standard linear regression, and lasso regression performed very similarly to the neural net (Figure 4).

Historical sales dominate in predictive power

Based on the coefficients from the LASSO regression, the rolling average sales feature is by far the most predictive of future sales (Figure 5). Of the remaining features, the cross between generic vendor and flower proportion is the most important, though interestingly, this feature has a negative coefficient. This may be an artifact of the scaling process, though, because the raw feature cross is positively correlated with total sales. The rolling average ARP feature also has a strong negative coefficient in the LASSO regression.

Discussion

In this project, I leveraged monthly sales data and brand metadata in an effort to discover key factors at play in marijuana sales in California.

I found that a neural network model can predict future sales to a moderate degree of confidence. The error metric achieved with my model could have been lower, but this is a good starting point for future efforts.

Additionally, I found that while the past sales data was the biggest driver in predicting future sales, brand characteristics also made an impact. My results suggest that where profit is concerned, simplicity wins—relative to other brand characteristics, a strong predictor of future sales was the feature representing brands that sold products through generic vendors and focused on flower. Also, selling expensive products might not pay off, seeing as a brand's past mean ARP was negatively associated with future sales.

My approach had some limitations that could be improved upon in the future. Judging by my correlation heatmap, there was high collinearity in my feature set. This caused the variances of my standard linear model predictions to be quite high. This issue prompted me to try the random forest regression, because ensemble methods are effective in reducing the variance of a predictor by aggregating across many instances of the simple model. I was also prompted to try the LASSO regression, because regularization can zero out collinear features if they are not contributing enough to the prediction. However, I was not able to improve much on the error metric from the standard linear model with these more complex methods.

Also, there could have been a better way of choosing and cleaning the features. For features with high missingness, I decided to drop them entirely to avoid introducing bias from whatever imputation method I chose. For features with low missingness, I dropped the samples with missing values for the same reason. If an effective imputation method could be devised for those features, the sample size of this analysis could be increased. It's possible there is some important information that I lost by dropping those observations. Additionally, there are many other ways to represent the categorical data than the top-5-proportion method that I used. I reasoned that the composition of product categories that a brand sold would give a sense of the that company's profile and approach to sales. However, it may have been a less noisy measurement to measure, for example, the presence of edibles/inhaleables/etc., rather than measuring the proportion of products falling into each of these categories.

Conclusion

In this project, I sought to predict future sales based on historical sales data, and to determine the key factors contributing to a brand's success on the California marijuana market. I was able to consolidate the available information into a complete dataset representing sales over time across thousands of brands, along with several characteristics of each brand. I was then able to analyze this data to find trends in the market.

I made two main findings: first, that the future sales volume of a marijuana brand can be well predicted by a neural net trained on the historical sales data and details about the products the brand sells. Second, that the model is driven by historical sales data, with brand details serving as fine-tune adjustments to the predicted outcome. Of these details, simplicity and ubiquity are likely to improve the sales of a brand's products.

project3_analysis

December 1, 2021

1 Analysis

This notebook includes the end-to-end data analysis pipeline for predicting sales in the California cannabis market.

Done for Project 3 in CS M148 at UCLA.

[ ]:
```
import pandas as pd
import numpy as np

from collections import Counter

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import TimeSeriesSplit, cross_val_score,␣
↪GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPRegressor
import sklearn.metrics as metrics

import statsmodels.api as sm

from matplotlib import pyplot as plt
import seaborn as sns
```

1.1 Define data pipeline functions

[ ]:
```
def mergeTimeData(arp, totsales, totunits):
    '''Merges raw time-level data into a single clean dataframe'''
    # rename columns to match
    arp.rename(columns = {'Brands': 'Brand',
                          'vs. Prior Period': 'ARP_vsPrior'}, inplace = True)
    totsales.rename(columns = {'Total Sales ($)': 'totSales'}, inplace = True)
    totunits.rename(columns = {'Brands': 'Brand',
                               'Total Units': 'totUnits',
```

```
                              'vs. Prior Period': 'totUnits_vsPrior'}, inplace =␣
↪True)
   # combine dataframes
   monthlev = arp.merge(totsales, how = 'outer', on = ['Brand', 'Months']).
↪merge(totunits, how = 'outer', on = ['Brand', 'Months'])
   # remove rows with no information in any of the important columns
   monthlev = monthlev.dropna(subset = ['ARP', 'totSales', 'totUnits'], how =␣
↪'all').reset_index(drop = True)
   # cast columns to correct type
   # clean out commas in totUnits and totSales
   monthlev.loc[:,'totUnits'] = [x.replace(',', '') for x in monthlev.totUnits]
   monthlev.loc[:, 'totSales'] = [x.replace(',', '') if type(x) == str else np.
↪NaN for x in monthlev.totSales ]
   monthlev = monthlev.astype({'Brand': 'str', 'ARP': 'float', 'ARP_vsPrior':␣
↪'float',
                               'totSales': 'float', 'totUnits': 'float',␣
↪'totUnits_vsPrior': 'float'})
   monthlev.Months = pd.to_datetime(monthlev.Months)

   # compute a rolling average of sales for this brand (average of last␣
↪WINDOWSIZE months' sales)
   WINDOWSIZE = 3
   monthlev_roll = pd.DataFrame()
   timecols = ['totSales', 'totUnits', 'ARP']
   brands = monthlev.loc[:,'Brand'].unique()
   for b in brands:
       mli = monthlev.loc[monthlev['Brand'] == b, :].sort_values('Months')
       for col in timecols:
           # compute rolling average column
           rollcolname = 'rolling_avg_' + col
           mli.loc[:,rollcolname] = mli.loc[:,col].rolling(WINDOWSIZE,␣
↪min_periods = 1).mean()
           # shift the column so the rolling average is only from previous␣
↪WINDOWSIZE months
           newroll = list(mli.loc[:,rollcolname])[0:-1]
           newroll.insert(0, np.NaN)
           mli.loc[:,rollcolname] = newroll
           # compute previous month column
           prevcolname = 'prev_month_' + col
           prevcol = list(mli.loc[:,col][0:-1])
           prevcol.insert(0, np.NaN)
           mli.loc[:,prevcolname] = prevcol
       monthlev_roll = pd.concat([monthlev_roll, mli], axis = 0, sort = False)

   # remove initial time point for each brand; that info is now encoded in the␣
↪second time point, and they won't have a rolling average
```

```
    monthlev_roll_clean = monthlev_roll.dropna(subset = ['rolling_avg_' + c for␣
↪c in timecols] + ['prev_month_' + c for c in timecols])
    monthlev_roll_clean = monthlev_roll_clean.drop(['ARP_vsPrior',␣
↪'totUnits_vsPrior'], axis = 1)

    return monthlev_roll_clean
```

[ ]:
```
def topX(column, x):
    '''helper for aggregateDetails: get the X most frequent values in each␣
↪column we're doing topX on'''
    sortedcounts = sorted(Counter(column.dropna()).items(),
                          key = lambda item: item[1], reverse = True)
    return([pair[0] for pair in sortedcounts[0:x]])

def topXproportion(column, topXvals):
    '''helper for aggregateDetails: get the proportions of the top X values in␣
↪brand's subset of a feature'''
    thislen = len(column.dropna()) # get proportion of non-null values that are␣
↪val
    if(thislen):
        props = [sum(column == val) / thislen for val in topXvals]
        props.append(1 - sum(props)) # add an other column
    else:
        # return NA if column has none of the most common values
        props = [np.NaN] * (len(topXvals) + 1)
    return(props)

def aggregateDetails(details):
    '''Condense product-level details to aggregate statistics per brand'''
    # clean up column names
    details.rename(columns = {'Category L1': 'cat1', 'Category L2': 'cat2',␣
↪'Category L3': 'cat3', 'Category L4': 'cat4', 'Category L5': 'cat5',
                'Product Description': 'product', 'Total Sales ($)':␣
↪'totSales_2021', 'Total Units': 'totUnits_2021', 'ARP': 'ARP_2021',
                'Items Per Pack': 'items_per_pack', 'Item Weight':␣
↪'item_weight', 'Total THC': 'tot_THC', 'Total CBD': 'tot_CBD', 'Contains␣
↪CBD': 'cont_CBD',
                'Pax Filter': 'pax_filter', 'Is Flavored': 'is_flavored', 'Mood␣
↪Effect': 'mood_effect', 'Generic Vendor': 'generic_vendor',
                'Generic Items': 'generic_items', '$5 Price Increment':␣
↪'price_inc_5dollar'}, inplace = True)

    # select the features decided on in develop pipeline
    keepfeat = ['cat1', 'cat2', 'cat3', 'Brand', 'product', 'items_per_pack',
        'tot_THC', 'tot_CBD', 'cont_CBD', 'mood_effect', 'generic_vendor',␣
↪'generic_items']
```

```
    details_clean = details.loc[:,keepfeat]

    # convert binary columns to numeric
    binarycols = ['cont_CBD', 'mood_effect', 'generic_vendor', 'generic_items']
    details_clean.loc[:,'cont_CBD'] = details_clean.loc[:,'cont_CBD'].
↪map({'Contains CBD': 1, 'THC Only': 0})
    details_clean.loc[:,'mood_effect'] = details_clean.loc[:,'mood_effect'].
↪map({'Mood Specific': 1, 'Not Mood Specific': 0})
    details_clean.loc[:,'generic_vendor'] = details_clean.loc[:
↪,'generic_vendor'].map({'Generic Vendors': 1, 'Non-Generic Vendors': 0})
    details_clean.loc[:,'generic_items'] = details_clean.loc[:,'generic_items'].
↪map({'Generic Items': 1, 'Non-Generic Items': 0})
    details_clean.loc[:,binarycols] = details_clean.loc[:,binarycols].apply(pd.
↪to_numeric)

    # convert remaining numeric columns to numeric
    makenumeric = ['tot_THC', 'tot_CBD']
    details_clean.loc[:,'tot_THC'] = [x.replace(',', '') for x in details_clean.
↪loc[:,'tot_THC']]
    details_clean.loc[:,'tot_CBD'] = [x.replace(',', '') for x in details_clean.
↪loc[:,'tot_CBD']]
    details_clean.loc[:,makenumeric] = details_clean.loc[:,makenumeric].
↪apply(pd.to_numeric)

    # group on brand and aggregate details
    # define columns I want to do top X proportion aggregation on
    topXcols = ['cat1', 'cat2', 'cat3']
    # group details on brand name
    details_grouponbrand = details_clean.groupby(['Brand'])
    NTOP = 5
    topX_percol_dict = {}
    for col in topXcols:
        topX_percol_dict[col] = topX(details_clean[col], NTOP)
    # run aggregation specific to each column
    details_agg = details_grouponbrand.agg({'cat1': lambda c: topXproportion(c,␣
↪topX_percol_dict['cat1']),
                                            'cat2': lambda c: topXproportion(c,␣
↪topX_percol_dict['cat2']),
                                            'cat3': lambda c: topXproportion(c,␣
↪topX_percol_dict['cat3']),
                                            'items_per_pack': 'mean',
                                            'tot_THC': 'mean',
                                            'tot_CBD': 'mean',
                                            'cont_CBD': 'mean',
                                            'mood_effect': 'mean',
                                            'generic_vendor': 'mean',
```

```
                                                'generic_items': 'mean'})

    # convert columns of lists into separate columns
    notTopXcols = list(set(details_agg.columns) - set(topXcols))
    dta = details_agg.loc[:,notTopXcols]
    for col in topXcols:
        thisdf = pd.DataFrame(details_agg[col].tolist(), columns =␣
↪topX_percol_dict[col] + ['other_' + col])
        dta = pd.concat([dta.reset_index(drop = True), thisdf.reset_index(drop␣
↪= True)], axis = 1)
    dta.loc[:,'Brand'] = details_agg.index

    # add some augmented features
    dta.loc[:,'THC_cross_mood'] = dta.loc[:,'tot_THC'] * dta.loc[:
↪,'mood_effect']
    dta.loc[:,'generic_vendor_cross_flower'] = dta.loc[:,'generic_vendor'] *␣
↪dta.loc[:,'Flower']

    return(dta)
```

[ ]:
```
def combineTimeAndDetails(monthlev_roll_clean, dta):
    '''Combine merged time data and brand-aggregated details into a single␣
↪clean dataframe '''
    # merge on brand
    cookies = monthlev_roll_clean.merge(dta, on = 'Brand', how = 'left')
    # remove the ~3k rows with NA remaining
    cookies = cookies.dropna()
    return cookies
```

1.2 Import data and put it through the pipeline

[ ]:
```
arp = pd.read_csv('data/BrandAverageRetailPrice.csv')
totsales = pd.read_csv('data/BrandTotalSales.csv')
totunits = pd.read_csv('data/BrandTotalUnits.csv')
details = pd.read_csv('data/BrandDetails.csv')
```

[ ]:
```
cookies = combineTimeAndDetails(mergeTimeData(arp, totsales, totunits),
                                aggregateDetails(details))
cookies
```

[ ]:
```
                  Brand     Months        ARP      totSales    totUnits  \
0          #BlackSeries 2021-01-01  13.611428   9739.423400  715.532838
1          #BlackSeries 2021-02-01  11.873182   9102.802187  766.669135
2      101 Cannabis Co. 2020-01-01  34.134929  11790.663567  345.413448
3      101 Cannabis Co. 2020-02-01  29.091388  20266.761007  696.658431
4      101 Cannabis Co. 2020-03-01  32.293498  30465.470533  943.393328
…                  …         …         …           …          …
```

```
23633          Zkittlez 2021-03-01  39.352005  30241.899130  768.497040
23634          Zkittlez 2021-04-01  39.387355  35209.055568  893.917749
23635          Zkittlez 2021-05-01  40.463240  25006.899159  618.015240
23636          Zkittlez 2021-06-01  38.295832  15835.402614  413.501985
23637          Zkittlez 2021-07-01  51.224714   1041.125100   20.324664

       rolling_avg_totSales  prev_month_totSales  rolling_avg_totUnits  \
0              25352.135918         25352.135918           1616.339004
1              17545.779659          9739.423400           1165.935921
2               4465.040321          4465.040321            131.067720
3               8127.851944         11790.663567            238.240584
4              12174.154965         20266.761007            391.046533
…                     …                    …                     …
23633           6116.791592          3848.649409            111.146389
23634          14584.506270         30241.899130            331.358823
23635          23099.868036         35209.055568            574.473890
23636          30152.617952         25006.899159            760.143343
23637          25350.452447         15835.402614            641.811658

       prev_month_totUnits  rolling_avg_ARP  …  Sublinguals  other_cat2  \
0              1616.339004        15.684913  …          0.0         0.0
1               715.532838        14.648170  …          0.0         0.0
2               131.067720        34.066667  …          0.0         0.0
3               345.413448        34.100798  …          0.0         0.0
4               696.658431        32.430995  …          0.0         0.0
…                      …              …  …  …          …           …
23633            61.006881        55.554205  …          0.0         0.0
23634           768.497040        53.717689  …          0.0         0.0
23635           893.917749        47.274952  …          0.0         0.0
23636           618.015240        39.734200  …          0.0         0.0
23637           413.501985        39.382142  …          0.0         0.0

       Vape  Dabbable Concentrates  Pre-Rolled  Hybrid  Infused Pre-Rolled  \
0       0.0                0.000000    0.000000     0.5            0.000000
1       0.0                0.000000    0.000000     0.5            0.000000
2       0.0                0.935065    0.000000     0.0            0.064935
3       0.0                0.935065    0.000000     0.0            0.064935
4       0.0                0.935065    0.000000     0.0            0.064935
…     …                     …          …     …                 …
23633   0.0                0.533333    0.466667     0.0            0.000000
23634   0.0                0.533333    0.466667     0.0            0.000000
23635   0.0                0.533333    0.466667     0.0            0.000000
23636   0.0                0.533333    0.466667     0.0            0.000000
23637   0.0                0.533333    0.466667     0.0            0.000000

       other_cat3  THC_cross_mood  generic_vendor_cross_flower
0             0.5             0.0                          0.0
```

```
1              0.5              0.0                          0.0
2              0.0              0.0                          0.0
3              0.0              0.0                          0.0
4              0.0              0.0                          0.0
…              …                …                            …
23633          0.0              0.0                          0.0
23634          0.0              0.0                          0.0
23635          0.0              0.0                          0.0
23636          0.0              0.0                          0.0
23637          0.0              0.0                          0.0

[20313 rows x 38 columns]
```

1.2.1 Split off labels and define scaling method

[ ]:
```
# needs to be sorted on month to do time series CV later
cookies = cookies.sort_values(by = 'Months')
```

[ ]:
```
cookies_y = cookies.loc[:,'totSales']
cookies_x = cookies.drop(['Brand', 'Months', 'totSales', 'totUnits',
                          'prev_month_totSales', 'rolling_avg_totUnits',
                          'prev_month_totUnits', 'ARP', 'prev_month_ARP'], axis =␣
 ↪1)

numcols = cookies_x.columns

print(cookies_x.shape)
print(cookies_y.shape)

# pipeline is just standardscaler, everything is numeric
pipe = StandardScaler()
```

```
(20313, 29)
(20313,)
```

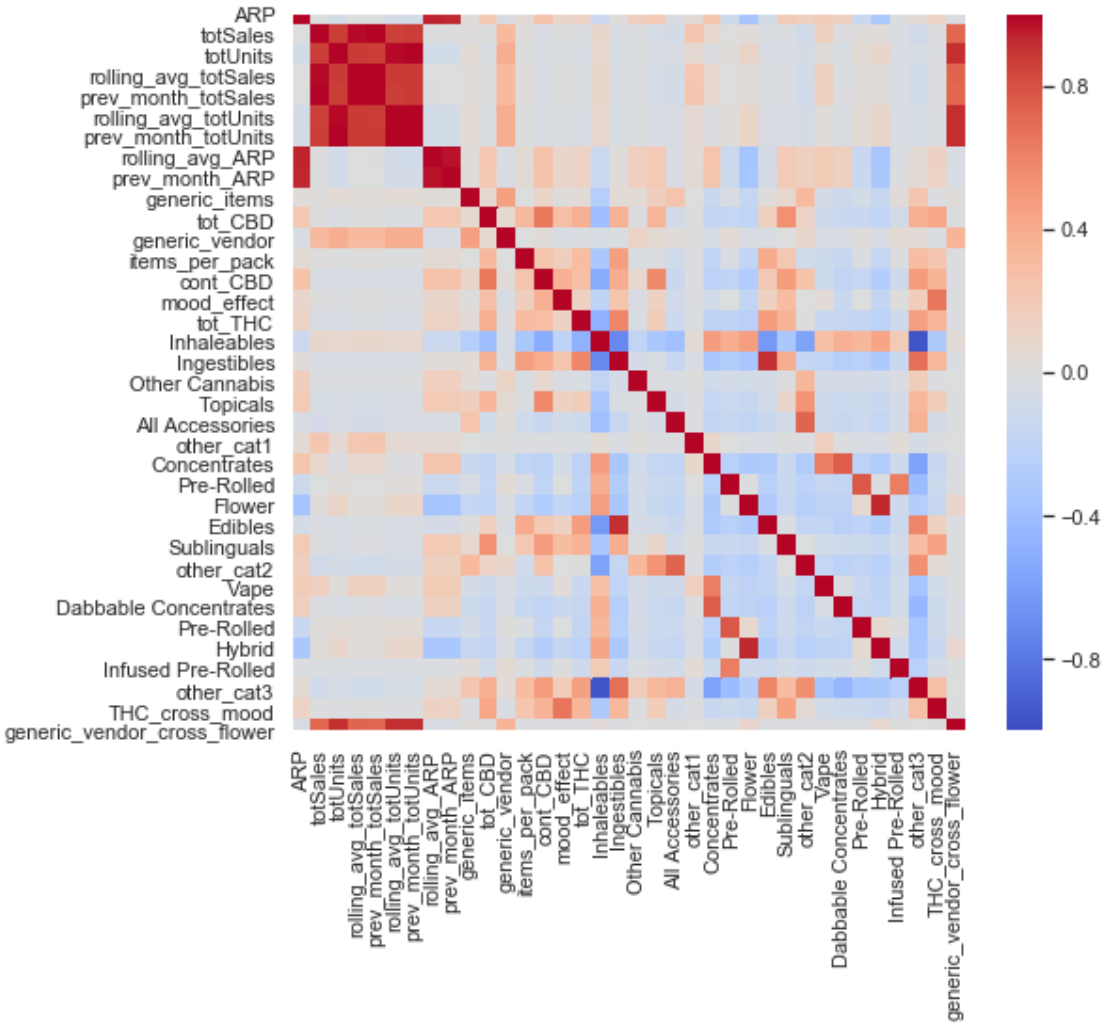
1.3 Explore the cleaned, merged dataset with some basic statistics

Figure 1

[ ]:
```
# correlate all the features with each other
cortable = cookies.corr(method = 'pearson')

sns.set(rc = {'figure.figsize':(8, 7)})
sns.heatmap(cortable, vmin = -1, vmax = 1, cmap = 'coolwarm',
            xticklabels = True, yticklabels = True)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb026373d10>
```



[ ]:
```
# find features most highly correlated with sales/units
print(cortable.loc[:,'totSales'].sort_values().head(10))
print(cortable.loc[:,'totSales'].sort_values().tail(10))

print(cortable.loc[:,'totUnits'].sort_values().head())
print(cortable.loc[:,'totUnits'].sort_values().tail())
```

```
other_cat2           -0.087804
other_cat3           -0.086252
All Accessories      -0.065689
Topicals             -0.051829
Ingestibles          -0.035176
cont_CBD             -0.033893
tot_THC              -0.029145
Edibles              -0.027805
Infused Pre-Rolled   -0.026497
```

```
Sublinguals          -0.023020
Name: totSales, dtype: float64
Vape                           0.153533
other_cat1                     0.240451
generic_vendor                 0.313528
generic_vendor_cross_flower    0.719664
rolling_avg_totUnits           0.865416
prev_month_totUnits            0.867671
totUnits                       0.874069
rolling_avg_totSales           0.990862
prev_month_totSales            0.993192
totSales                       1.000000
Name: totSales, dtype: float64
rolling_avg_ARP   -0.080241
ARP               -0.078492
prev_month_ARP    -0.078354
other_cat2        -0.057762
other_cat3        -0.053354
Name: totUnits, dtype: float64
rolling_avg_totSales           0.875550
generic_vendor_cross_flower    0.912745
rolling_avg_totUnits           0.991780
prev_month_totUnits            0.993070
totUnits                       1.000000
Name: totUnits, dtype: float64
```

1.4 Run standard models

[ ]:
```
def regression_results(y_true, y_pred):
    '''function to get model performance'''
    # Regression metrics
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    return {'explained_variance': round(explained_variance, 4),
            'r2': round(r2, 4),
            'MAE': round(mean_absolute_error, 4),
            'MSE': round(mse, 4),
            'RMSE': round(np.sqrt(mse), 4)}
```

[ ]:
```
# generate the train/test split we'll use for all of these models

# since we want to predict the future,
# use the most recent time points as the test set
cutoffdate = '2021-08-01'
```

```
train = cookies_x.loc[cookies['Months'] < cutoffdate]
test = cookies_x.loc[cookies['Months'] >= cutoffdate]
train_lab = cookies_y.loc[cookies['Months'] < cutoffdate]
test_lab = cookies_y.loc[cookies['Months'] >= cutoffdate]

# scale and center features
train_pipe = pipe.fit_transform(train)
test_pipe = pipe.transform(test)

print(train_pipe.shape, train_lab.shape)
print(test_pipe.shape, test_lab.shape)
```

```
(18760, 29) (18760,)
(1553, 29) (1553,)
```

1.4.1 Linear regression

[ ]:
```
# get a linear regression model
linreg = LinearRegression()

# fit to training data
linreg.fit(train_pipe, train_lab)

# predict test labels
test_pred_linreg = linreg.predict(test_pipe)

# check performance
regression_results(test_lab, test_pred_linreg)
```

[ ]:
```
{'explained_variance': 0.9881,
 'r2': 0.9872,
 'MAE': 75054.3482,
 'MSE': 30399308802.5012,
 'RMSE': 174353.9756}
```

[ ]:
```
# find individual stats for each feature
olsmod = sm.OLS(train_lab, train_pipe) # define OLS model
olsfit = olsmod.fit() # fit model
print(olsfit.summary()) # get statistics
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                totSales   R-squared (uncentered):
0.910
Model:                            OLS   Adj. R-squared (uncentered):
0.910
```

```
Method:                 Least Squares   F-statistic:
8202.
Date:                Wed, 01 Dec 2021   Prob (F-statistic):
0.00
Time:                        15:06:53   Log-Likelihood:
-2.7450e+05
No. Observations:               18760   AIC:
5.491e+05
Df Residuals:                   18737   BIC:
5.492e+05
Df Model:                          23
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          1.748e+06   6654.229    262.680      0.000    1.73e+06    1.76e+06
x2         -4451.1403   4680.706     -0.951      0.342   -1.36e+04    4723.467
x3         -2548.1037   4991.720     -0.510      0.610   -1.23e+04    7236.121
x4          1181.9108   6030.935      0.196      0.845   -1.06e+04     1.3e+04
x5          -418.9173   5089.999     -0.082      0.934   -1.04e+04    9557.942
x6          2592.2147   4700.220      0.552      0.581   -6620.642    1.18e+04
x7         -1482.6882   7466.943     -0.199      0.843   -1.61e+04    1.32e+04
x8         -7340.4637   5795.184     -1.267      0.205   -1.87e+04    4018.623
x9         -2777.1339   5484.621     -0.506      0.613   -1.35e+04    7973.220
x10        -3069.6368   3.51e+04     -0.087      0.930   -7.19e+04    6.58e+04
x11         2355.4764   2.41e+04      0.098      0.922   -4.48e+04    4.95e+04
x12         2746.7412   8155.861      0.337      0.736   -1.32e+04    1.87e+04
x13         1226.5153   1.24e+04      0.099      0.921   -2.31e+04    2.55e+04
x14         -237.9290   1.54e+04     -0.015      0.988   -3.05e+04       3e+04
x15         1.561e+04   4311.489      3.620      0.000    7157.963    2.41e+04
x16        -6542.2609   1.01e+05     -0.065      0.948   -2.04e+05    1.91e+05
x17         1636.5852   3334.017      0.491      0.624   -4898.390    8171.560
x18         2071.7665   6.76e+04      0.031      0.976    -1.3e+05    1.35e+05
x19         2869.4218   2.18e+04      0.132      0.895   -3.98e+04    4.55e+04
x20          -673.2299   1.09e+04     -0.062      0.951   -2.19e+04    2.06e+04
x21         1554.4599   2.13e+04      0.073      0.942   -4.01e+04    4.32e+04
x22         6929.5697    7.2e+04      0.096      0.923   -1.34e+05    1.48e+05
x23         6900.7910   8.39e+04      0.082      0.934   -1.58e+05    1.71e+05
x24         1782.8385   3680.093      0.484      0.628   -5430.478    8996.155
x25        -2074.2871   4.64e+04     -0.045      0.964    -9.3e+04    8.88e+04
x26          399.0738   3762.053      0.106      0.916   -6974.891    7773.038
x27        -8880.2305   7.76e+04     -0.114      0.909   -1.61e+05    1.43e+05
x28         6117.2195   6155.170      0.994      0.320   -5947.471    1.82e+04
x29        -1.913e+04   6501.841     -2.942      0.003   -3.19e+04   -6382.551
==============================================================================
Omnibus:                    15228.322   Durbin-Watson:                   0.371
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          9757923.229
Skew:                           2.740   Prob(JB):                         0.00
```

```
Kurtosis:                       114.595   Cond. No.                     1.23e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 7.78e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[ ]:
```
# single fit
lassoreg = Lasso()
lassoreg.fit(train_pipe, train_lab)
test_pred_lasso = lassoreg.predict(test_pipe)
regression_results(test_lab, test_pred_lasso)
```

```
explained_variance:  0.9881
r2:  0.9872
MAE:  72255.3496
MSE:  30266459190.1446
RMSE:  173972.5817
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 327787615307092.25, tolerance: 5772225505213.152
  positive)
```

[ ]:
```
# check which features were most important, or were zeroed out
betadf = pd.DataFrame({'feature': train.columns,
                       'beta': lassoreg.coef_})
betadf = betadf.sort_values(by = 'beta')
print(betadf.head())
print(betadf.tail())
print(betadf[betadf['beta'] == 0])
```

1.4.2 PCA

Figure 2

[ ]:
```
NCOMP = 10

# get PCA object
pcaobj = PCA(n_components = NCOMP)

# calculate PCs
PCs = pcaobj.fit_transform(pipe.transform(cookies_x))
```

[ ]:
```
Text(0, 0.5, 'Proportion of variance explained')
```

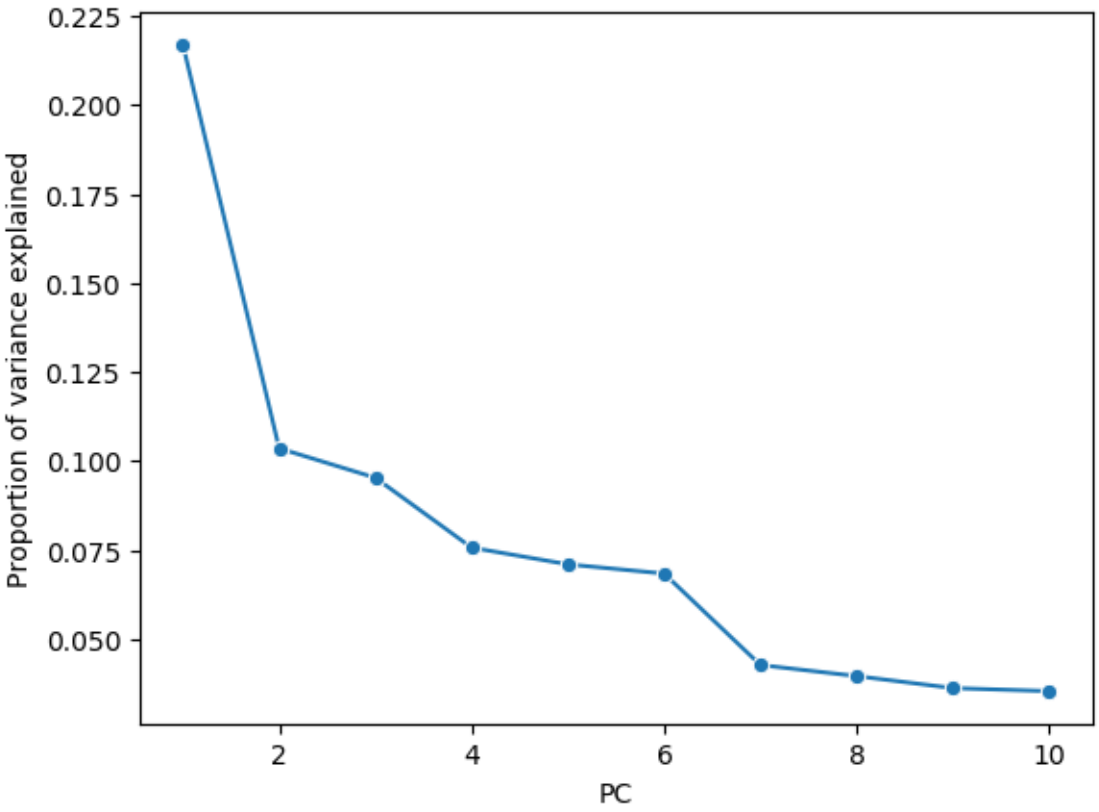
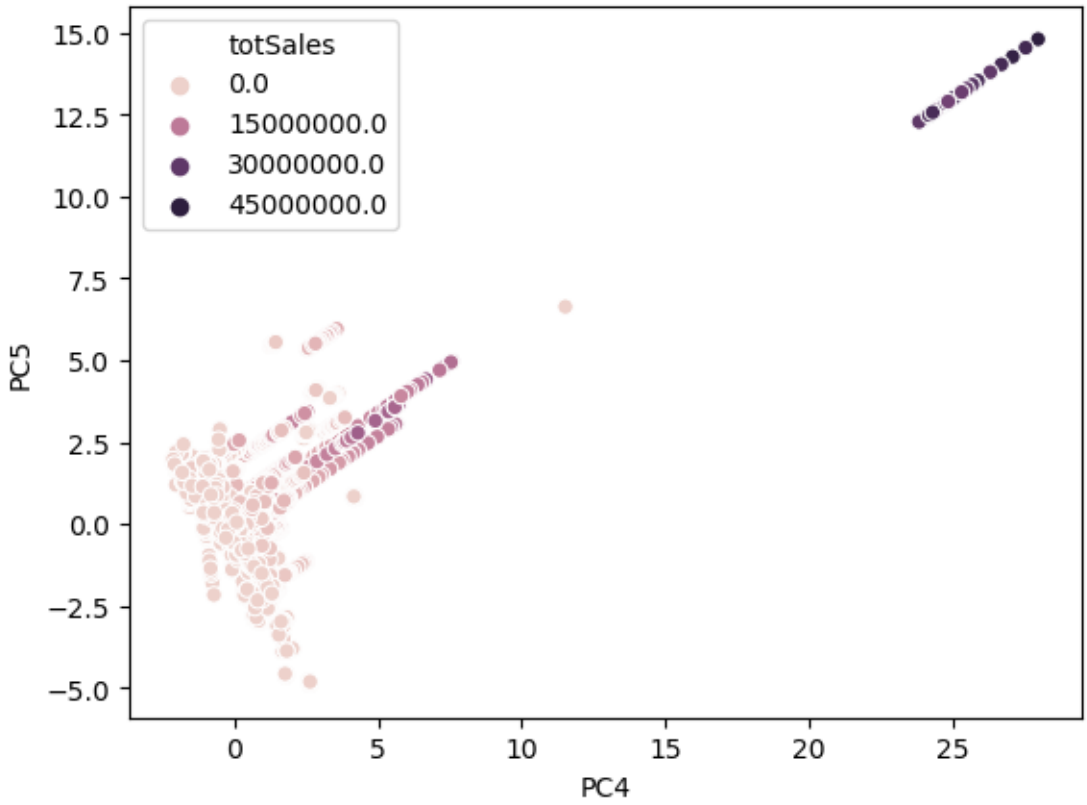


Figure 3

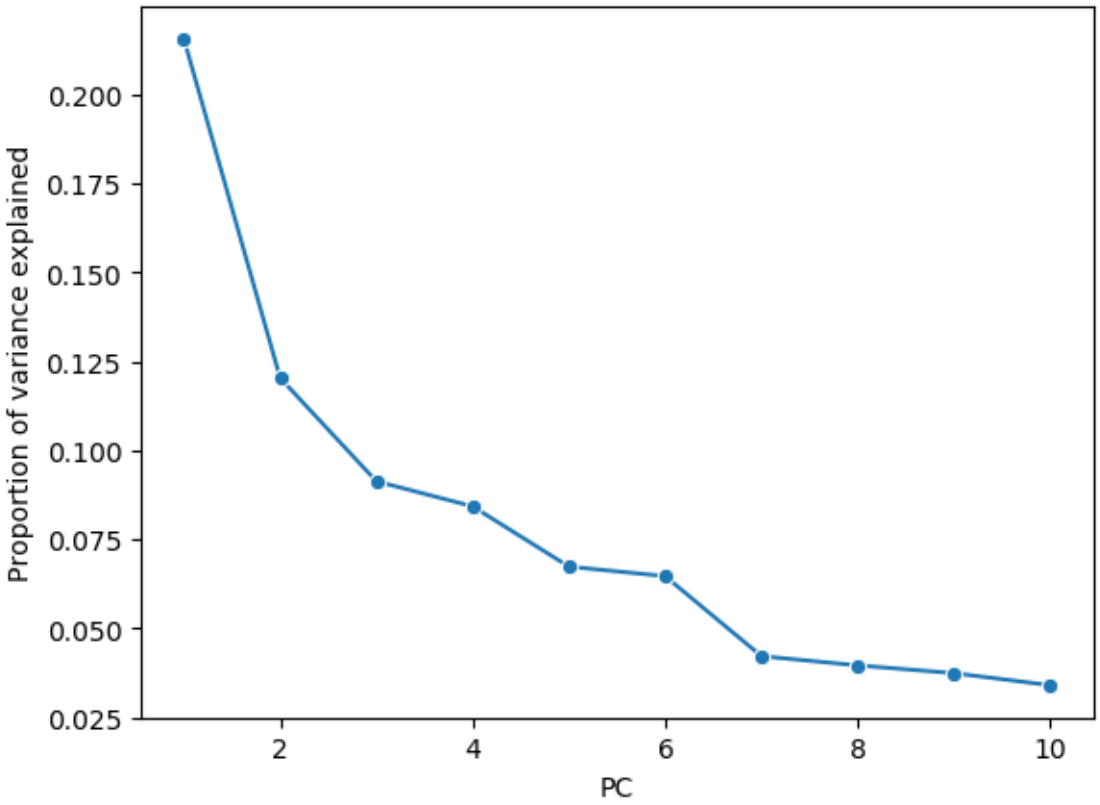
[ ]:
```
PCs.shape
```

[ ]:
```
Text(0, 0.5, 'PC5')
```



[ ]:
```
sns.reset_defaults()
# plot variance explained per PC
varplot = sns.lineplot(list(range(1, NCOMP+1)),
            pcaobj.explained_variance_ratio_,
            marker = 'o')
varplot.set_xlabel("PC")
varplot.set_ylabel("Proportion of variance explained")
```

[ ]:
```
Text(0, 0.5, 'Proportion of variance explained')
```



[ ]:
```
# plot some PCs
pcaplot = sns.scatterplot(PCs[:,4], PCs[:,5], hue = cookies_y)
pcaplot.set_xlabel("PC4")
pcaplot.set_ylabel("PC5")
```

[ ]:
```
{'explained_variance': 0.9848,
 'r2': 0.9839,
 'MAE': 77521.648,
 'MSE': 38254012226.3888,
 'RMSE': 195586.3293}
```

1.4.3 Ensemble method

[ ]:
```
# get a RF regressor
rfreg = RandomForestRegressor(n_estimators = 50, max_depth = 5)

# fit to train data
rfreg.fit(train_pipe, train_lab)

# predict test labels
test_pred_rfreg = rfreg.predict(test_pipe)
```

```
regression_results(test_lab, test_pred_rfreg)
```

```
TRAIN: [   0    1    2 … 3385 3386 3387] TEST: [3388 3389 3390 … 6770 6771
6772]
TRAIN: [   0    1    2 … 6770 6771 6772] TEST: [ 6773  6774  6775 … 10155
10156 10157]
TRAIN: [    0     1     2 … 10155 10156 10157] TEST: [10158 10159 10160 …
13540 13541 13542]
TRAIN: [    0     1     2 … 13540 13541 13542] TEST: [13543 13544 13545 …
16925 16926 16927]
TRAIN: [    0     1     2 … 16925 16926 16927] TEST: [16928 16929 16930 …
20310 20311 20312]
```

1.5 K-fold cross validation

[ ]:
```
# since this is time series data, I will use K different cutoff points
# defining what is 'past' and what is 'future'
K_ALL = 5

tscv = TimeSeriesSplit(n_splits = K_ALL)

# verify the time series split worked correctly
for train_index, test_index in tscv.split(cookies_x):
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
TRAIN: [   0    1    2 … 3385 3386 3387] TEST: [3388 3389 3390 … 6770 6771
6772]
TRAIN: [   0    1    2 … 6770 6771 6772] TEST: [ 6773  6774  6775 … 10155
10156 10157]
TRAIN: [    0     1     2 … 10155 10156 10157] TEST: [10158 10159 10160 …
13540 13541 13542]
TRAIN: [    0     1     2 … 13540 13541 13542] TEST: [13543 13544 13545 …
16925 16926 16927]
TRAIN: [    0     1     2 … 16925 16926 16927] TEST: [16928 16929 16930 …
20310 20311 20312]
```

[ ]:
```
# k-fold CV on the linear regression
linreg_kfold_pipe = make_pipeline(pipe, linreg)
linreg_result_kfold = cross_val_score(linreg_kfold_pipe,
                                      cookies_x, cookies_y, cv = tscv,
                                      scoring = 'neg_mean_squared_error')

nmse_linreg = linreg_result_kfold.mean()
print("Mean negative MSE from linear regression CV: " + str(linreg_result_kfold.
 ↪mean()))
```

[ ]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=5),
             error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'n_estimators': [1, 5, 10, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

1.6 Grid search

1.7 Experiment with more models

1.7.1 Lasso regression

[ ]:
```
# grid search on RF with time series cross validation
rfgrid = {'n_estimators': [1, 5, 10, 50, 100],
            'max_depth': [1, 2, 3, 4, 5]}

gridsearch_rf = GridSearchCV(RandomForestRegressor(), rfgrid, cv = tscv,␣
 ↪scoring = 'neg_mean_squared_error')

gridsearch_rf.fit(cookies_x, cookies_y)
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 334516881331220.3, tolerance: 5772225505213.152
  positive)
```

[ ]:
```
{'explained_variance': 0.9881,
 'r2': 0.9872,
 'MAE': 72255.3887,
 'MSE': 30266469971.456,
 'RMSE': 173972.6127}
```

[ ]:
```
# x-val
lasso_kfold_pipe = make_pipeline(pipe, lassoreg)
lasso_result_kfold = cross_val_score(lasso_kfold_pipe,
                                     cookies_x, cookies_y, cv = tscv,
                                     scoring = 'neg_mean_squared_error')

nmse_lasso = lasso_result_kfold.mean()
print("Mean negative MSE from LASSO CV: " + str(lasso_result_kfold.mean()))
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 5687585503273.344, tolerance: 1608331007322.793
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 144824700210131.84, tolerance: 2486474291108.579
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 28146278901258.312, tolerance: 3429120602395.4688
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 132907821352820.75, tolerance: 4341481215237.294
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 329781811795067.25, tolerance: 5336511162825.121
  positive)
```

```
Mean negative MSE from LASSO CV: -56349565558.83195
```

[ ]:
```
nmse_rf = gridsearch_rf.best_score_

print("Best parameters from grid search: " + str(gridsearch_rf.best_params_))
print("Best negative MSE from grid search: " + str(gridsearch_rf.best_score_))
```

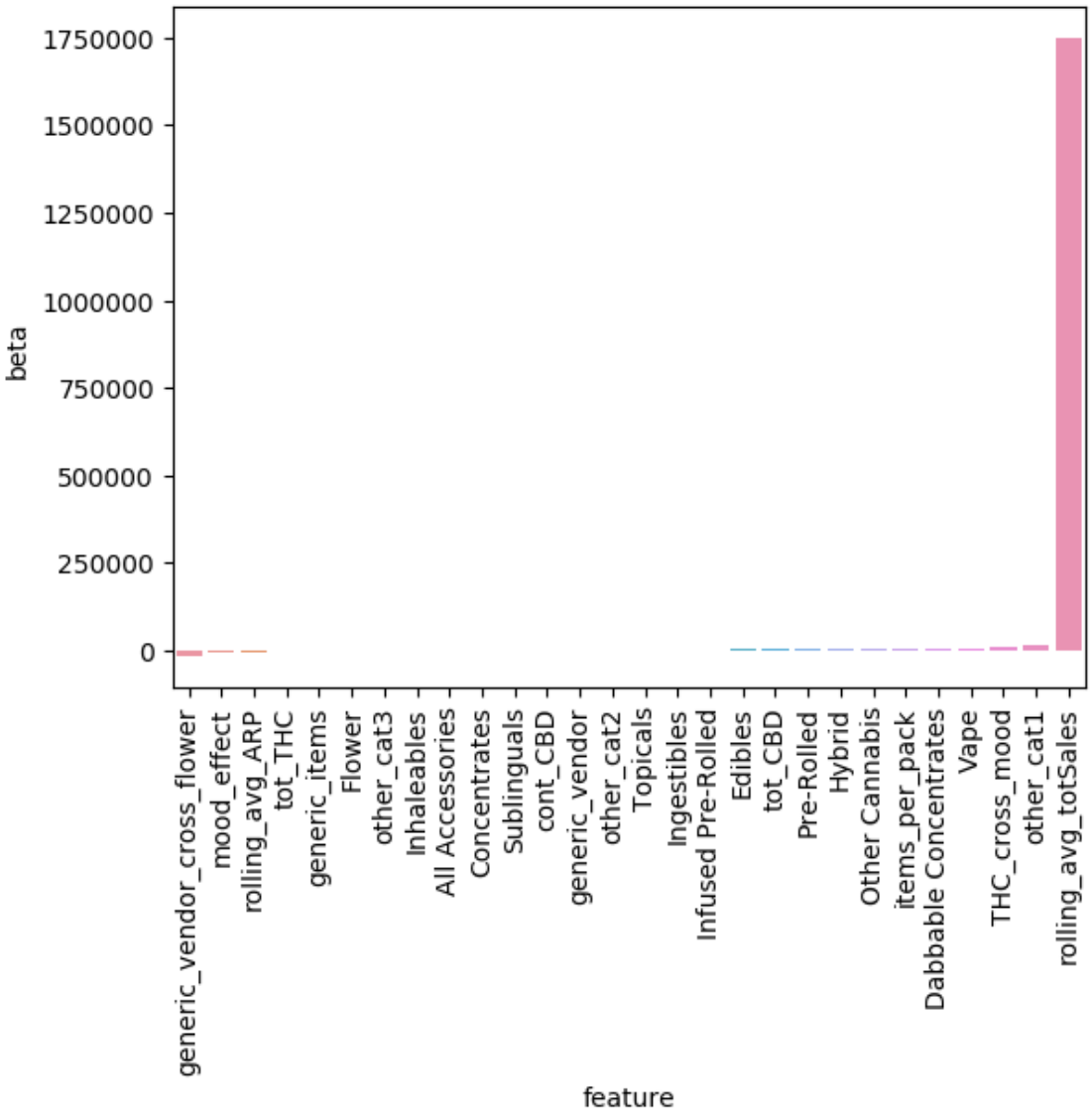
```
                      feature          beta
28  generic_vendor_cross_flower -19125.860146
7                   mood_effect  -7331.386936
1               rolling_avg_ARP  -4445.786807
8                       tot_THC  -2765.043037
2                 generic_items  -2548.024578
                  feature          beta
```

```
22  Dabbable Concentrates  3.635595e+03
21                   Vape  4.129247e+03
27         THC_cross_mood  6.114038e+03
14             other_cat1  1.560850e+04
0    rolling_avg_totSales  1.747930e+06
       feature  beta
20  other_cat2  -0.0
```

Figure 5

[ ]:
```
betaplot = sns.barplot('feature', 'beta', data = betadf)
betaplot.set_xticklabels(betaplot.get_xticklabels(),rotation = 90)
```

[ ]:
```
[Text(0, 0, 'generic_vendor_cross_flower'),
 Text(0, 0, 'mood_effect'),
 Text(0, 0, 'rolling_avg_ARP'),
 Text(0, 0, 'tot_THC'),
 Text(0, 0, 'generic_items'),
 Text(0, 0, 'Flower'),
 Text(0, 0, 'other_cat3'),
 Text(0, 0, 'Inhaleables'),
 Text(0, 0, 'All Accessories'),
 Text(0, 0, 'Concentrates'),
 Text(0, 0, 'Sublinguals'),
 Text(0, 0, 'cont_CBD'),
 Text(0, 0, 'generic_vendor'),
 Text(0, 0, 'other_cat2'),
 Text(0, 0, 'Topicals'),
 Text(0, 0, 'Ingestibles'),
 Text(0, 0, 'Infused Pre-Rolled'),
 Text(0, 0, 'Edibles'),
 Text(0, 0, 'tot_CBD'),
 Text(0, 0, 'Pre-Rolled'),
 Text(0, 0, 'Hybrid'),
 Text(0, 0, 'Other Cannabis'),
 Text(0, 0, 'items_per_pack'),
 Text(0, 0, 'Dabbable Concentrates'),
 Text(0, 0, 'Vape'),
 Text(0, 0, 'THC_cross_mood'),
 Text(0, 0, 'other_cat1'),
 Text(0, 0, 'rolling_avg_totSales')]
```



1.7.2 Regression on PCs

[ ]:
```
# get a dataframe together with the PCs and target variable
# make sure it's sorted by month for train/test split
pcdf = pd.concat([cookies[['Months', 'Brand', 'totSales']].reset_index(drop =␣
↪True), pd.DataFrame(PCs).reset_index(drop = True)], axis = 1).sort_values(by␣
↪= 'Months')
pcdf # columns 0-9 are PC1-PC10
```

[ ]:

	Months	Brand	totSales	0	\
0	2018-10-01	Vet CBD	329857.741054	2.797215	
198	2018-10-01	ProCana (CA)	219261.440893	4.174209	

```
197   2018-10-01             Nasha Extracts  615978.497305 -1.815186
196   2018-10-01                Jade Nectar  399811.383476  5.035759
195   2018-10-01  Dr. Kerklaan Therapeutics  255384.457981  4.255862
…            …                          …            …       …
19809 2021-09-01         Clique Farming Co.    1624.752028 -1.991850
19810 2021-09-01                Fruit Slabs   11671.601292  2.795208
19811 2021-09-01                       Clix    6680.242886 -0.240090
19802 2021-09-01       Coastal Sun Cannabis  976385.483142 -1.950858
20312 2021-09-01          Santa Cruz Roots   156660.034841 -1.935964

              1         2         3         4         5         6         7  \
0      0.331513  3.749931  1.057091  0.112494 -1.116979 -0.663338  4.353606
198   -0.358779 -1.821920 -1.552975 -0.554267  1.235184 -0.674077  1.067862
197    2.673122  0.007791 -0.629707 -0.235753 -0.105993 -1.433847 -0.649676
196    0.379636 -1.421118  0.804212  0.874121 -1.203935 -0.815573 -0.311491
195    0.705546  2.449942  2.661091  0.961465 -2.963885 -0.371805  1.919132
…           …         …         …         …         …         …         …
19809  0.920543 -0.183531 -0.763589  0.011750 -0.542507 -0.996908 -0.598170
19810 -0.621223 -1.148614 -2.128314 -1.029215  1.755913 -0.110186  0.548376
19811 -1.163514 -1.667158  2.605210 -1.436077  1.036598  1.060732 -0.585578
19802 -2.153775 -0.452866 -0.944758  1.205169 -1.449520  0.150098 -0.105346
20312  0.459095 -0.169441 -0.990166  0.479013 -0.753419 -0.172063 -0.276452

              8         9
0      3.119711  2.756632
198   -0.358339 -0.591005
197    0.863546 -0.430336
196   -0.294936  0.897722
195   -0.349626 -2.411505
…           …         …
19809  0.429390 -0.249615
19810 -0.011884 -0.188776
19811  1.224595 -0.639434
19802 -0.198297  0.354683
20312 -0.204660  0.116501

[20313 rows x 13 columns]
```

[ ]:
```
# split PCs into one train and test set
pcdf_y = pcdf.loc[:,'totSales']
pcdf_x = pcdf.drop(['Months', 'Brand', 'totSales'], axis = 1)

train_pc = pcdf_x.loc[pcdf['Months'] < cutoffdate]
test_pc = pcdf_x.loc[pcdf['Months'] >= cutoffdate]
train_lab_pc = pcdf_y.loc[pcdf['Months'] < cutoffdate]
test_lab_pc = pcdf_y.loc[pcdf['Months'] >= cutoffdate]
```

```
print(train_pc.shape, train_lab_pc.shape)
print(test_pc.shape, test_lab_pc.shape)
```

```
(18760, 10) (18760,)
(1553, 10) (1553,)
```

[ ]:
```
# fit an elasticnet model on the PCs
pcreg = ElasticNet()
pcreg.fit(train_pc, train_lab_pc)
test_pred_pcreg = pcreg.predict(test_pc)
regression_results(test_lab_pc, test_pred_pcreg)
```

[ ]:
```
{'explained_variance': 0.7558,
 'r2': 0.7536,
 'MAE': 339804.184,
 'MSE': 584640196734.8615,
 'RMSE': 764617.6801}
```

[ ]:
```
# do a cross-validation and grid search on lasso/ridge balance
grid_pc = {'l1_ratio': np.arange(0., 1., 0.1),
           'alpha': [1, 5, 10, 100]}

gridsearch_pc = GridSearchCV(ElasticNet(), grid_pc, cv = tscv, scoring =␣
↪'neg_mean_squared_error')

gridsearch_pc.fit(pcdf_x, pcdf_y)
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2401357849054968.0, tolerance: 1473487348148.05
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 4576568464310678.0, tolerance: 2572749070017.8306
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 6980641668985342.0, tolerance: 3596978872552.5327
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 9086496051255886.0, tolerance: 4466191010199.623
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
```

```
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.1377295757439928e+16, tolerance: 5230947835465.862
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 4872998540530142.0, tolerance: 1473487348148.05
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 8860503226783745.0, tolerance: 2572749070017.8306
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.2902663023205916e+16, tolerance: 3596978872552.5327
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.6398803044557832e+16, tolerance: 4466191010199.623
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.9802579998111812e+16, tolerance: 5230947835465.862
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 5824759163773190.0, tolerance: 1473487348148.05
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.0423948495850464e+16, tolerance: 2572749070017.8306
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.4936856249625772e+16, tolerance: 3596978872552.5327
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.881040625542226e+16, tolerance: 4466191010199.623
```

```
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2.243507639263738e+16, tolerance: 5230947835465.862
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 7170770017467288.0, tolerance: 1473487348148.05
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.255894706209894e+16, tolerance: 2572749070017.8306
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.7612214466077064e+16, tolerance: 3596978872552.5327
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2.1906379068864776e+16, tolerance: 4466191010199.623
  positive)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/coordinate_descent.py:475: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2.5713744073867056e+16, tolerance: 5230947835465.862
  positive)
```

[ ]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=5),
             error_score='raise-deprecating',
             estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
                                  l1_ratio=0.5, max_iter=1000, normalize=False,
                                  positive=False, precompute=False,
                                  random_state=None, selection='cyclic',
                                  tol=0.0001, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [1, 5, 10, 100],
                         'l1_ratio': array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
    0.7, 0.8, 0.9])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

[ ]:
```
nmse_pc = gridsearch_pc.best_score_

print(gridsearch_pc.best_score_)
print(gridsearch_pc.best_params_)
```

```
-467910236015.063
{'alpha': 1, 'l1_ratio': 0.9}
```

1.7.3 Neural net

[ ]:
```
# single fit
nn = MLPRegressor()
nn.fit(train_pipe, train_lab)
test_pred_nn = nn.predict(test_pipe)
regression_results(test_lab, test_pred_nn)
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

[ ]:
```
{'explained_variance': 0.0448,
 'r2': 0.0179,
 'MAE': 386245.895,
 'MSE': 2330715775076.726,
 'RMSE': 1526668.1942}
```

[ ]:
```
# x-val grid search
grid_nn = {'hidden_layer_sizes': [10, 50, 100, 200],
           'max_iter': [5, 10, 50, 100, 200]}

gridsearch_nn = GridSearchCV(MLPRegressor(), grid_nn, cv = tscv, scoring =␣
↪'neg_mean_squared_error')
gridsearch_nn.fit(cookies_x, cookies_y)

nmse_nn = gridsearch_nn.best_score_

print(gridsearch_nn.best_score_)
print(gridsearch_nn.best_params_)
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
```

```
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
```

```
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
```

```
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (5) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
/Users/niko/opt/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

-53849868167.59598
{'hidden_layer_sizes': 100, 'max_iter': 100}
```

1.8 Compare all the models

Table 1

[ ]:
```
modelnames = ['linear regression', 'random forest', 'lasso', 'PC lasso',␣
↪'neural net']
allnmse = [nmse_linreg, nmse_rf, nmse_lasso, nmse_pc, nmse_nn]

modelcomp = pd.DataFrame({'model': modelnames, 'NMSE': allnmse}).sort_values(by␣
↪= 'NMSE').reset_index(drop = True)
modelcomp
```

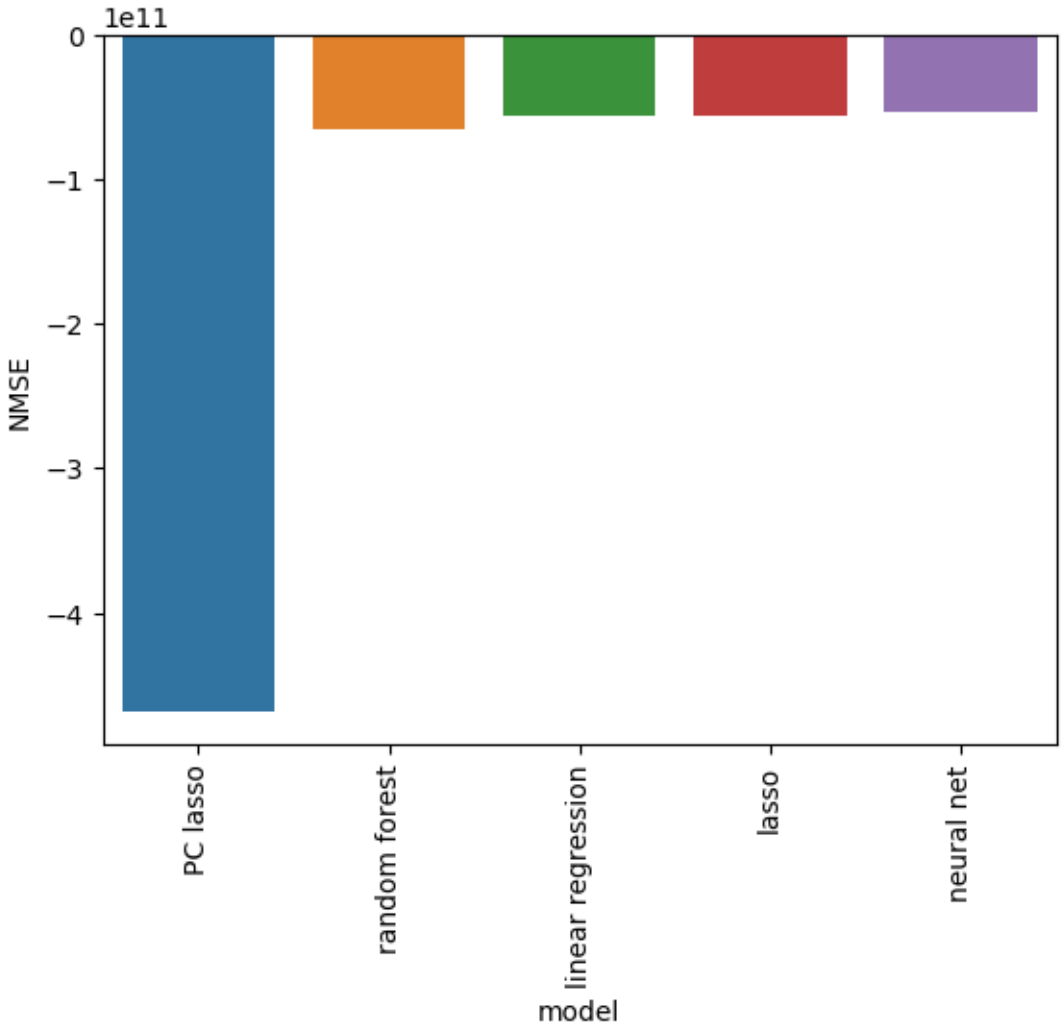
[ ]:

	model	NMSE
0	PC lasso	-4.679102e+11
1	random forest	-6.521408e+10
2	linear regression	-5.634972e+10
3	lasso	-5.634957e+10
4	neural net	-5.384987e+10

Figure 4

[ ]:
```
compplot = sns.barplot('model', 'NMSE', data = modelcomp)
compplot.set_xticklabels(compplot.get_xticklabels(),rotation = 90)
```

[ ]:
```
[Text(0, 0, 'PC lasso'),
 Text(0, 0, 'random forest'),
 Text(0, 0, 'linear regression'),
 Text(0, 0, 'lasso'),
 Text(0, 0, 'neural net')]
```


1e11
0
−1
−2
−3
−4
NMSE
PC lasso
random forest
linear regression
lasso
neural net
model