ER131_project_group1

October 6, 2021

1 Predicting Extent of Power Overload in Northern California

Fall 2020

1. Yash Chainani
   (Sorted and merged PG&E dataset with census dataset, did the EDA section, edited interpretations)
2. Nicole Chang
   (Sorted and extracted forecasted climate data, wrote background, abstract and parts of interpretations, did little work on predictions)
3. Sancialita Sathiyamoorthy
   (sorted and found census dataset, performed majority of regression predictions and visualization, cleaned and annotated notebook)
4. Amy Tong
   (sorted and merged economic dataset, performed KNN regression for the prediction part)

1.1 Abstract (5 points)

Rolling blackouts have occurred in California at an increasing rate due to factors such as drought, thunderstorms and other impacts of climate change. This has created disparity and inequality in people's ability to do work or school due to the unreliable nature of the power available. In this project we investigated the feasibility of using modeled climate data, economic factors, and census data to predict to what extent specific substations are burdened in various counties in northern California by implementing regression techniques including linear regression with Lasso and Ridge regularization and KNN nonparametric regression. Test mean squared errors (MSEs) show that KNN outperforms OLS, LASSO, and Ridge regressions by two to three fold. For instance, temporal predictions show that KNN yields a MSE of 1200 while OLS, Ridge, and LASSO all yield equally large MSEs of more than 8200. Thus we determined that using KNN, the only non-parametric method we used, was the best model for all of our overload predictions and that predicting forward 6 months can be reasonably done for each substation. For our linear regression methods, our temporal and time-lag predictions were reasonable but had high bias whereas the linear regression spatial predictions performed extremely poorly. KNN as a method performed much better overall for all predictions, but the spatial predictions were still much worse than the temporal and time-lag predictions. Further research and refining of alternate models needs to be done before they can be properly used to advise state policy and decisions of where to reinforce and invest in providing more power.

1.2 Project Background (5 points)

Rolling blackouts have occurred in California with increasing frequency over the past few years. It has had a widespread impact for the state, creating difficulties for work and school, and has left Californians subject to the mercy of extenuating factors for energy availability. Climate change has had a large impact and has resulted in scheduled blackouts associated with both fire and overload risk. Not only has temperature been increasing, but a worsening drought over the years has led to more dry underbrush. Fires can be started when trees and dry branches come into contact with active power lines. A combination of winds, increasing temperatures, and drought led to the disastrous fires that occurred in 2017, when it was proclaimed that power lines owned by the Pacific Gas and Electric Company (PG&E) were the cause of over a dozen fires in Northern California and over the summer of 2018, the cause of an additional 17 more. This led to the public safety power shut offs we now have which help mitigate risk. There have also been the kinds of shut offs associated with overload on the power grid. This kind of shut off is one California should be able to reduce in the future with better planning. During hot weather when people are using air conditioning and trying to keep cool, electrical equipment is at high risk of being overloaded which triggers power outages as circuit breakers shut off power to prevent damage to electrical equipment. The California Independent System Operator (CAISO) manages the grid and the distribution of power and reserve power. They instruct the utility companies like PG&E to conduct rolling blackouts. Now in 2020, not only has California experienced unusual thunderstorms, but also high heat associated with climate change and an unprecedented pandemic that has forced people to stay at home. It is thus more important now than ever to better predict how the power grid is used and analyze the actual amount of power that communities are using. For more details on the power grid, which is also known as the bulk power system, there are transmission lines that connect generating stations to different substations. These substations which are located throughout a county lead to specific feeder stations that supply a specific amount of power to streets or regions. Doing an analysis of overload on specific substations will allow the state to make more informed decisions about California's needed power. Currently, California depends heavily on imported energy. There are also more vulnerabilities due to our shifting dependency on renewables such as solar and wind as more coal power plants have been shut down. These renewables are also intermittent and there is not enough battery storage for these types of energy. It is thus important to be able to predict how power demand will change with different regions and different time periods so that proper measures can be taken by CAISO, utility companies, and the state to improve our current predicament. These prediction problems can guide decisions for where it is important to make more contracts to get additional power, guide power rerouting decisions, and do more research on energy storage so that Californias will have a more resilient power infrastructure in place. In this project, we focus on the overload on substations that belong to the Pacific Gas and Electric Company (PG&E), which covers some counties in Northern California. This is a starting point for predicting overload more broadly in all counties in California in the future. Sources: 1. https://www.pge.com/en_US/residential/outages/public-safety-power-shuttoff/why-psps-events-occur.page 2. https://electrical-engineering-portal.com/substation-basics 3. https://www.pge.com/includes/docs/pdfs/shared/edusafety/systemworks/electric/howhotweatheroutages/how_c
4. https://www.kqed.org/news/11833911/californias-overloaded-power-grid

1.3 Project Objective (5 points)

The purpose of this project is to train and test various models to predict overload of substations in different counties in Northern California that have power provided by PG&E both spatially and

temporally using modeled climate change and weather data from Cal-Adapt, demographic data from the American Census Survey, economic data from California's Employment Development Department and Labor Force, and PG&E customer and power usage data. The spatial prediction will enable us to predict which locations are likely to be overloaded or underloaded and by how much, and this information is very useful for CAISO and PG&E to know where to reroute power and how much based on where there is overload or close to overload. The temporal prediction will enable prediction of overload or underload for various months and allow the state to evaluate if power usage changes in some consistent or measurable way over a period of time. Our additional objective is to train and test various models to predict overload of substations in different counties in California 6 months in the future also using the same datasets as for the spatial and temporal prediction problems above. Ideally, forecasted weather data would be used for this prediction problem because that is the data we would have if we were to use the model in practice, but we were unable to find archived weather forecast data. Hence we trained this model with a time lag of 6 months, using actual data from 6 months before a certain time to predict the overload at that time. This prediction might be more accurate if we trained the model on data from a year in advance (so that we match the same months) since this would reflect the seasonal changes of power usage and have similar weather patterns, but we were not able to obtain PG&E data for this. This prediction is also very useful for knowing if there will be an overload and how much at some point in the future, which means that measures can be taken ahead of time for prevention by PG&E and CAISO. This includes planning for securing more energy contracts or expanding production as well as rerouting power. For all three prediction problems, the response variable is power overload in kW, and the features are weather data, census data, economic data and either location (for spatial prediction) or time (for temporal with and without time lag) predictions.

1.4 Input Data Description (5 points)

Data Source 1: Integrated Circuit Analysis (ICA) load data from Pacific Gas & Electric (PG&E) company

The first dataset in our project comes from the Pacific Gas and Electric Company (PG&E) Integrated Capacity Analysis (ICA) map. After the initial step of having to create an account, PG&E allows users to view a geographical ICA map wherein users can navigate around California and pick various feeders and substations to view their load profiles. All of this geospatial data can also directly be downloaded from the PG&E website. The entire zip file comprises over 20 files and is slightly over 400 megabytes but after some initial data-cleaning described in the data-cleaning section, we were able to import all of the key features from the PG&E data into Pandas dataframes in Jupyter.

PG&E provides load profile data for each feeder in all the counties in California. Each county has a specific number of substations and each substation is connected to a specific number of feeders. Not all of the substations and feeders have included their load profile data in the PG&E dataset, however. Some of the substations have redacted the load profile data of their associated feeders due to confidentiality reasons. The load profiles of the feeders are recorded in terms of light load and high load for twenty four hours each month. These values are obtained by averaging light load and high load values for every hour each month into the twenty-four hours that make up a day. For instance, for the month of January, the light and high load readings are taken at month_hour values of 01_01, 01_02, to 01_24, where 01_01 is the average of all light and high loads taken at 01:00 AM for every day in January while 01_02 is the average of all light and high loads taken at 02:00 AM for every day in January and so on. These constitute the **granularity** of our dataset.

The **scope** of our data are all values from the year 2019.

In terms of data **faithfulness**, these values were digitally recorded and on cleaning the data, we saw no null values or no erroneous values.

The structure of the files downloaded from PG&E has various extensions such as .atx, .gdbindexes, .freelist, .gdbtablx but a lot of these extensions cannot be opened using the modules in Jupyter and the file extension could be opened using Geopandas was .gdbtable. So the **structure** of our data from PG&E is mainly files with .gdbtable extensions.

One important feature in the final PG&E dataframe that we create from merging several PG&E datasets (substations, feeders, load profiles, etc.) is the load capacity and this has units of kilowatts. This represents the threshold above which a particular feeder would be overloaded. **Our response variable is the overload value, which is obtained by subtracting by the high load from load capacity.** This means that negative values would indicate that a feeder is overloaded while positive values would indicate that a feeder is underloaded or not at risk of power failure. One issue that we ran into in creating our response variable from this feature is that each feeder also comprised several linesections and each linesection had its own load capacity value associated with it so in order to decide on a single threshold value for each feeder, we took the maximum value across all the feeders. Officially, the load capacity is defined as the number of amperes of electric current a wire will carry without becoming unduly heated; the capacity of a machine, apparatus, or devices is the maximum of which it is capable under existing service conditions; the load for which a generator, turbine, transformer, transmission circuit, apparatus, station, or system is rated. Capacity is also used synonymously with capability.

Other key features that make up our data are distributed generation, residential customers, industrial customers, latitude, and longitude.

Data Source 2: Geographic dataset

This data defines the California County boundaries in shapefile format from the US Census Bureau's 2016 MAF/TIGER database.

- Structure: .shp files
- Granularity: for each county, it defined a spatial object of the boundary of a specific county
- Scope: is the current boundaries of counties as of September 10, 2019
- Temporality: no time data associated with this, but it was last updated on the above date
- Faithfulness: This data was from the US Census Bureau's 2016 MAF/TIGER database. Methods of aggregation were not stated but the Census Bureau gets data from census data as well as states and government agencies.
- Features: County locations and boundaries

Data Source 3: Climate Model prediction data

Cal-Adapt has been developed by the Geospatial Innovation Facility at University of California, Berkeley with funding and advisory oversight by the California Energy Commission and California Strategic Growth Council. Daily climate projections for California at a resolution of 1/16° (about 6 km, or 3.7 miles) were generated to support climate change impact studies for California's Fourth Climate Change Assessment. The data, derived from 32 coarse-resolution (~100 km) global climate models from the CMIP5 archive were bias corrected and downscaled using the Localized Constructed Analogues (LOCA) statistical method. From the various Global Climate Models (GCMs), we chose to use CanESM2 model (Canadian Centre for Climate Modelling and Analysis, Victo-

ria, BC, Canada), or the "average" model and selected data for the Representative Concentration Pathway (RCP) 8.5 which refers to the concentration of carbon that delivers global warming at an average of 8.5 watts per square meter across the planet and is the "business-as-normal" climate model.

We used a Climate Change Assessment from the California Energy Commission that detailed the climate models and guided data selection.

- Structure: tabular data(excel files), first column was time data (format: 2010-10-01)
- Granularity: each record or row represents the variable data (temperature, windspeed, etc) for each of the 58 counties in California
- Scope:The modeled data was available between 2006 to 2100, but we chose a subset from 2015-2025 to use which was most relevant to making current predictions.
- Temporality:(format: 2010-10-01) daily data, no strange timestamps found (checked 1990-01-01 excel format and this was not found)
- Faithfulness:Uses a statistical model to predict data based on a global climate model which was described to be a good approximation for California. May be unrealistic since it predicts data into the future. No hand-entered data though.
- Features: Temperature, max [C], Temperature, min[C], Relative Humidity, min [RH, %], relative humidity [RH, %] (a measure of the amount of moisture held in the atmosphere), Relative Humidity, max [RH, %], Precipitation [cm/time], Wind speed [m/s], Solar radiation [W/m^2]

Data Source 4: Census Data from the US Census Bureau's ACS Demographic and Housing Estimates.

ACS stands for American Census Survey. We are using the 2018 5-year estimates data because it is the most recent available data that has all areas, even with smaller populations. Guidance about which dataset to use when tells us that 1 year estimates are the most current but least reliable. It is merged with the PG&E dataset on county. Estimates are made by The Census Bureau, which selects a random sample of addresses to be included in the ACS each month, and the estimate is made by averaging over 5 years.

- Structure: tabular data(Excel .xlsx files). By using the .head method, we see that the column titles are the first row, so it is reformatted and columns are renamed to be more descriptive.
- Granularity: Each record or row represents data (total population, population by race, population by sex and age, and total housing units) for a particular county in California.
- Scope: The spatial scope of the data is all of California and temporal scope is an average between 2014-2018. This is a little wider than necessary since we only have response variable data for a fraction of counties but will be addressed when the datasets are merged. The data includes estimates, percent estimates, margin of error and percent margin of error. Only the estimates are kept.
- Temporality: The data are from 2014-2018 (averaged). There are no time entries in the dataset.
- Faithfulness: There does not appear to be any outliers. There are some missing entries denoted with (X) in some of the percent estimate or percent margin of error columns, but these columns aren't necessary for our analysis and were dropped. The data appears faithful.
- Features: Total population, population by race (White, Black or African American, American Indian and Alaska Native, Asian, Native Hawaiian and Other Pacific Islander, Some other race, Two or more races), population by sex (Male, Female), and total housing units

Data Source 5: Labor Force Data Labor force data is taken from California's Employment Development Department (EDD), Labor Market Information Division.

- Structure: .pdf files, manually converted to .xlsx
- Granularity: each row contains data on the amount of the population employed, unemployed, and in the labor force for one of the 58 counties in CA
- Scope: data from Jan 2019 to Oct 2020, but for now our subset of data is from Jan-Dec 2019
- Temporality: monthly data
- Faithfulness: uses time-series models to produce data for Los Angeles County, and indirect estimation techniques for other counties
- Features: Labor force (includes persons aged 16 and older who were not institutionalized or on active military duty and were either employed or unemployed), unemployment rate

Data Source 6: Wage Data

"Quarterly Census of Employment & Wages (QCEW)" is taken from California's Employment Development Department (EDD), Labor Market Information Division.

- Structure: .csv file
- Granularity: for each of the 58 counties in CA, includes data of average weekly wages
- Scope: data from 1990 to 2020, but our subset of data is of 2019
- Temporality: quarterly and yearly data
- Faithfulness: all employers covered under CA UI laws must report their quarterly wages, data that is then used to populate this database; thus highly reliable
- Features: Average weekly wages

1.5 Data Cleaning (10 points)

We went through a significant amount of data cleaning for the PG&E ICA dataset. The first task we attempted to complete was to read in the .gdbtable files from PG&E into Jupyterhub but we repeatedly ran into issues with the kernel dying. This was a strange issue because even though the files themselves were only around 100 megabytes, reading them into a notebook on Jupyterhub suddenly caused our notebook memory to swell and exceed a gigabyte, which is close to the maximum amount that the jupyterhub notebooks can handle. We suspect that the geometry and other geospatial features of these datasets was causing jupyterhub to crash so we initially tried to remove the geometry column, export the data as a csv and then re-import it into our notebooks after restarting the kernel but ultimately, we circumvented the memory overload issue by installing geopandas locally and then using the geopandas.read() command to read the .gdbtable files into a local notebook. There are at least ten .gdbtable files given to us by PG&E but a number of them contain metadata so after reading in each file, we realized that there are about 3-4 .gdbtable files that were most essential and contained useful data.

The first of these files is "a00000016.gdbtable", which contains information about the maximum load capacities of various feeders. While this file was the largest to read in and would almost certainly cause our jupyter notebooks on jupyterhub to crash, it was the most important file because it contained information about the load capacity thresholds that would later be used to calculate our response variable, which was the overload value (negative values indicate risk of power failure. The next important file was "a00000020.gdbtable", which contained temporal information about the load profiles of each of the feeders in month_hour format. Another important file was "a0000001f.gdbtable", which contained information on each of the substations. For this

dataset, the geometry column was actually useful for us to do some exploratory data analysis (EDA) on but we had to change the Coordinate Reference System (CRS) to epsg 4326 using the geopands.to_crs() command. All other information about the feeders associated with each substation, such as the number of residential, industrial, and agricultural customers, etc are found in the file titled "a0000001e.gdbtable". After reading in data about the feeders, substations, and feeder load profiles, we merge all dataframes on the unique feeder identification numbers (FeederID) to create a master dataframe that contains all of the PG&E data, i.e load profile, feeder information, and substation information along with our response variable, which is the overload, where a positive value indicates that the feeder is not at risk of power failure while a negative value indicates that the feeder is. **Our entire data cleaning and merging work for PG&E can be found in a separate notebook in our zip file that we have titled: "Merging PG&E and census".**

For each dataset, we chose to do a couple checks individually on our datasets in separate notebooks. We converted the string number values into integers, checked if there were any NaN values, and checked for redundant column names. We each did research individually on how the data was generated (for example, all the modeled temperature, humidity, and wind speed data was generated from a statistical model and nothing was hand-entered), and checked the column titles after each merge. We also checked what types of variables existed in each column. Lastly, we checked the datasets column names for leading and trailing whitespaces.

We also realized that substation load capacities were either 0 or at their actual capacity. We decided to drop all the values at which the variable 'Load Capacity kW' was equal to 0 as this indicates times in which substations were most likely not operating or their sensors for load capacity were not operational and was thus not useful for evaluating the degree to which a system is overloaded or underloaded. For the census data, the data was cleaned by dropping columns that were not going to be used as features, namely the columns that provided percent estimates, margins of error and percent margins of error. Any columns containing (X)'s were dropped. No other columns contained null values. The census data had a granularity of the county level, so this dataset was merged with the master dataset containing the response variable (from the PG&E dataset) on county. The county names were cleaned by dropping the ' County, California' in the names (eg. Alameda County, California $\rightarrow$ Alameda) so that they matched the PG&E dataset. The titles were cleaned to be more descriptive.

From the full dataset, the strings in the `Average weekly wages` column were converted from strings to numbers so that they could be used in regression models. Additionally, census data about populations (racial and gender data) were converted to fractional values rather than absolute values so that we could use fractional racial and gender composition as features. Columns that would cause linear independence are dropped in the modeling/prediction section below.

The forecasted weather data was cleaned by first splitting the time column which had data for each day and month into just a monthly data entry. This monthly data was then averaged over all the days in the month for all features except for precipitation; precipitation was aggregated by taking the sum of all the days in the month as this was recorded as precipitation per day. The year column was dropped and the data frame was reshaped so that columns became rows.

Economic data was collected and parsed in a similar matter to the other datasets to aggregate per month and was joined on the county from the PG&E dataset.

Importing relevant libraries and packages

[4]:
```
# import geopandas as gpd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import math
import os, csv

pd.set_option('display.max_columns', None)
```

[2]:
```
!pip install xlrd
```

```
Requirement already satisfied: xlrd in
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-packages (1.2.0)
```

Load in csv file of already merged PG&E data and census data see additional appendix materials for census and PG&E data merging

[3]:
```
# Load in datasets for predicted temperature data from datasource 3
master_df = pd.read_csv("Master_dataset_PG&E_census")
```

[4]:
```
# Load in datasets for predicted temperature data from datasource 3

master_df = pd.read_csv("Master_dataset_PG&E_census")
min_temperatures = pd.read_excel("min_temp_data.xlsx")
max_temperatures = pd.read_excel("max_temp_data.xlsx")
min_humidities = pd.read_excel("min_humidities_data.xlsx")
max_humidities = pd.read_excel("max_humidities_data.xlsx")
wind_speeds = pd.read_excel("wind_speeds.xlsx")
solar_radiation = pd.read_excel("solar_rad.xlsx")
precipitation = pd.read_excel("precip.xlsx")
```

[5]:
```
master_df.head()
```

[5]:

	County	FeederID	Month	Hour	High	LoadCapacity_kW	overload	\
0	Alameda	14422111	1	1	5935	10000	4065	
1	Alameda	13751102	1	1	1828	0	-1828	
2	Alameda	13751101	1	1	1180	4480	3300	
3	Alameda	14422109	1	1	452	10000	9548	
4	Alameda	14402105	1	1	5910	0	-5910	

	Substation	Feeder_Name	ResCust	ComCust	IndCust	AgrCust	OthCust	\
0	CAYETANO	CAYETANO 2111	1748	171	121	4	10	
1	VASCO	VASCO 1102	1407	183	27	36	8	
2	VASCO	VASCO 1101	1626	50	28	0	4	
3	CAYETANO	CAYETANO 2109	2849	205	58	23	9	
4	LAS POSITAS	LAS POSITAS 2105	1309	427	297	0	11	

```
   Existing_DG  Queued_DG  Total_DG  SUBSTATIONID  \
0         3550         20      3570          1442
1         1650        430      2080          1375
2          530         10       540          1375
3         7580       3130     10710          1442
4         3380         10      3390          1440

                                      geometry   Longitude  Latitudes  \
0  POINT (-121.7706964837614 37.73836825980029) -121.770696  37.738368
1  POINT (-121.7156770257398 37.71097214991492) -121.715677  37.710972
2  POINT (-121.7156770257398 37.71097214991492) -121.715677  37.710972
3  POINT (-121.7706964837614 37.73836825980029) -121.770696  37.738368
4  POINT (-121.7335062173786 37.70094251246452) -121.733506  37.700943

   Total population   White  Black or African American  \
0           1643700  681725                      177135
1           1643700  681725                      177135
2           1643700  681725                      177135
3           1643700  681725                      177135
4           1643700  681725                      177135

   American Indian and Alaska Native   Asian  \
0                              10712  486434
1                              10712  486434
2                              10712  486434
3                              10712  486434
4                              10712  486434

   Native Hawaiian and Other Pacific Islander  Some other race  \
0                                       13768           169771
1                                       13768           169771
2                                       13768           169771
3                                       13768           169771
4                                       13768           169771

   Two or more races  Total housing units    Male  Female
0             104155               601942  807171  836529
1             104155               601942  807171  836529
2             104155               601942  807171  836529
3             104155               601942  807171  836529
4             104155               601942  807171  836529
```

Data cleaning for the weather datasets

[6]:
```
# Function for averaging data for each county by month,
# extracting the specific desired year, and reshaping the dataframe for merging.
```

```
def weather_data_cleaning(df, year, data_name):
    # Convert the "time" column to "year" and "month"
    year_col = df["time"].apply(lambda x: int(x.split('-')[0]))
    month_col = df["time"].apply(lambda x: int(x.split('-')[1]))

    # Insert the new "year" and "month" columns and remove the "time" column
    df.insert(0, 'Year', year_col)
    df.insert(1, 'Month', month_col)
    df.drop(columns=["time"], inplace=True)

    # Group dataset by year+month, then average the data in each group
    df = df.groupby(["Year", "Month"]).mean().reset_index()
    if data_name == "Precipitation":
        df = df.groupby(["Year", "Month"]).sum().reset_index()

    # Get data from the specific year desired, then drop the "year" column
    new_df = df[df["Year"] == year].reset_index(drop=True)
    new_df = new_df.drop(columns=["Year"])

    # "Unpivot the dataframe", aka reshape dataframe from (12,59) to (696,3) by␣
 ↪turning county columns to rows
    new_df = new_df.melt(id_vars="Month", var_name="County",␣
 ↪value_name=data_name)

    return new_df
```

[7]:
```
# The year we want to use for data analysis.
# REPLACE THIS NUMBER TO CHANGE WHICH YEAR WE USE.
year = 2019

# Data cleaning for the seven weather datasets, using the above function
new_mintemp = weather_data_cleaning(min_temperatures, year, "Minimum␣
 ↪temperature")
new_maxtemp = weather_data_cleaning(max_temperatures, year, "Maximum␣
 ↪temperature")
new_minhum  = weather_data_cleaning(min_humidities,   year, "Minimum humidity")
new_maxhum  = weather_data_cleaning(max_humidities,   year, "Maximum humidity")
new_wind    = weather_data_cleaning(wind_speeds,      year, "Wind speed")
new_solar   = weather_data_cleaning(solar_radiation,  year, "Solar radiation")
new_rain    = weather_data_cleaning(precipitation,    year, "Precipitation")
```

Data cleaning for economic datasets dataset #1: labor force data

[8]:
```
laborforce = pd.DataFrame(columns=["COUNTY", "LABOR FORCE", "RATE", "MONTH"])
filenames = ["laborforce_01_19.xlsx", "laborforce_02_19.xlsx",
```

```
            "laborforce_03_19.xlsx", "laborforce_04_19.xlsx",
            "laborforce_05_19.xlsx", "laborforce_06_19.xlsx",
            "laborforce_07_19.xlsx", "laborforce_08_19.xlsx",
            "laborforce_09_19.xlsx", "laborforce_10_19.xlsx",
            "laborforce_11_19.xlsx", "laborforce_12_19.xlsx"]

# For each file, read it in, data clean, and then merge to the laborforce␣
 ↪master dataset.
for i in np.arange(0,12):
    curr_month = pd.read_excel(filenames[i])
    curr_month = curr_month.drop(columns=['RANK BY\nRATE', 'EMPLOYMENT',␣
 ↪'UNEMPLOYMENT'])
    curr_month = curr_month.drop(index=[0])
    curr_month["MONTH"] = i+1
    laborforce = laborforce.merge(curr_month, how="outer")

# Rename and reorder the columns
laborforce = laborforce[["COUNTY", "MONTH", "LABOR FORCE", "RATE"]]
laborforce.columns = ["County", "Month", "Labor force", "Rate unemployed"]
# Format county names correctly.
laborforce["County"] = laborforce["County"].str.title()
```

dataset #2: wage data

[9]:
```
# Read in the csv file and drop unnecessary columns
wages = pd.read_csv("wagecensus_2019.csv")
wages = wages.drop(columns=["Year", "Ownership", "Division", "Level", "Ind␣
 ↪Code", "Industry"])

# Remove "County" from all county name strings
wages["Area"] = wages["Area"].apply(lambda x: x.rsplit(' ', 1)[0])

# Drop all rows w/ yearly temporal granularity
wages = wages[wages["Period"] != "Annual"].reset_index(drop=True)

# Replicate each row twice, and then append a "Month" column
wages = pd.DataFrame(np.repeat(wages.values, 3, axis=0))
wages.columns = ["Period", "County", "Average weekly wages"]
wages["Month"] = np.tile(np.arange(1,13), wages["County"].unique().shape[0])
# Checking that month values are assigned to the right quarter
# wages[["Period", "Month"]].value_counts()
wages = wages.drop(columns="Period")
wages = wages[["County", "Month", "Average weekly wages"]]
```

Merge weather and economic data with the master dataset

[10]:
```
master_df.head(3)
```

[10]:
```
   County  FeederID  Month  Hour  High  LoadCapacity_kW  overload Substation  \
0  Alameda  14422111      1     1  5935            10000      4065   CAYETANO
1  Alameda  13751102      1     1  1828                0     -1828      VASCO
2  Alameda  13751101      1     1  1180             4480      3300      VASCO

     Feeder_Name  ResCust  ComCust  IndCust  AgrCust  OthCust  Existing_DG  \
0  CAYETANO 2111     1748      171      121        4       10         3550
1     VASCO 1102     1407      183       27       36        8         1650
2     VASCO 1101     1626       50       28        0        4          530

   Queued_DG  Total_DG  SUBSTATIONID  \
0         20      3570          1442
1        430      2080          1375
2         10       540          1375

                                       geometry   Longitude  Latitudes  \
0  POINT (-121.7706964837614 37.73836825980029) -121.770696  37.738368
1  POINT (-121.7156770257398 37.71097214991492) -121.715677  37.710972
2  POINT (-121.7156770257398 37.71097214991492) -121.715677  37.710972

   Total population   White  Black or African American  \
0           1643700  681725                      177135
1           1643700  681725                      177135
2           1643700  681725                      177135

   American Indian and Alaska Native   Asian  \
0                              10712  486434
1                              10712  486434
2                              10712  486434

   Native Hawaiian and Other Pacific Islander  Some other race  \
0                                       13768           169771
1                                       13768           169771
2                                       13768           169771

   Two or more races  Total housing units    Male  Female
0             104155               601942  807171  836529
1             104155               601942  807171  836529
2             104155               601942  807171  836529
```

[11]:
```
# Merge weather data w/ the master dataset on the "County" and "Month" columns
master_df = master_df.merge(new_mintemp, how="left", on=["County","Month"])
master_df = master_df.merge(new_maxtemp, how="left", on=["County","Month"])
master_df = master_df.merge(new_minhum,  how="left", on=["County","Month"])
master_df = master_df.merge(new_maxhum,  how="left", on=["County","Month"])
master_df = master_df.merge(new_wind,    how="left", on=["County","Month"])
master_df = master_df.merge(new_solar,   how="left", on=["County","Month"])
```

```
master_df = master_df.merge(new_rain,    how="left", on=["County","Month"])
```

[12]:
```
# Merge economic data w/ master dataset
master_df = master_df.merge(laborforce,  how="left", on=["County","Month"])
master_df = master_df.merge(wages,       how="left", on=["County","Month"])
```

[13]:
```
#writes out to a new csv file called Master_dataset and has all merged data
master_df.to_csv("Master_dataset",index=False)
```

1.5.1 *NOTE: You can begin here if you'd like to skip the merging steps above*

[5]:
```
#reads in full Master_dataset
master_df = pd.read_csv('Master_dataset')
```

```
/opt/conda/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3146:
DtypeWarning: Columns (41) have mixed types.Specify dtype option on import or
set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[6]:
```
master_df = master_df[master_df["LoadCapacity_kW"]!=0]
# We suspect that load capacities may be equal to zero when the substations␣
 ↪were not operating so we removed those values
```

[7]:
```
master_df.isnull().values.any()
# Check for null values
```

[7]: False

[8]:
```
# cleaning data further and renaming to be used in the prediction problems
full_dataset = master_df

# converting the strings in the average weekly wages to a numeric value
full_dataset['Average weekly wages'] = full_dataset['Average weekly wages'].
 ↪apply(lambda x: int(str(x).replace(',','')))

# converting census data to fractional rather than absolute values (data␣
 ↪transformation)
columns_to_convert = ['White', 'Black or African American',
       'American Indian and Alaska Native', 'Asian',
       'Native Hawaiian and Other Pacific Islander', 'Some other race',
       'Two or more races','Male', 'Female', 'Labor force']

for column in columns_to_convert:
    full_dataset[column] = full_dataset[column].divide(full_dataset['Total␣
 ↪population'])
```

[9]:
```
full_dataset.head()
```

[9]:

	County	FeederID	Month	Hour	High	LoadCapacity_kW	overload
0	Alameda	14422111	1	1	5935	10000	4065
2	Alameda	13751101	1	1	1180	4480	3300
3	Alameda	14422109	1	1	452	10000	9548
5	Alameda	14402110	1	1	6767	9260	2493
6	Alameda	14402106	1	1	3431	9180	5749

	Substation	Feeder_Name	ResCust	ComCust	IndCust	AgrCust	OthCust
0	CAYETANO	CAYETANO 2111	1748	171	121	4	10
2	VASCO	VASCO 1101	1626	50	28	0	4
3	CAYETANO	CAYETANO 2109	2849	205	58	23	9
5	LAS POSITAS	LAS POSITAS 2110	995	280	136	2	11
6	LAS POSITAS	LAS POSITAS 2106	4550	143	13	0	8

	Existing_DG	Queued_DG	Total_DG	SUBSTATIONID
0	3550	20	3570	1442
2	530	10	540	1375
3	7580	3130	10710	1442
5	3340	350	3690	1440
6	4030	250	4280	1440

	geometry	Longitude	Latitudes
0	POINT (-121.7706964837614 37.73836825980029)	-121.770696	37.738368
2	POINT (-121.7156770257398 37.71097214991492)	-121.715677	37.710972
3	POINT (-121.7706964837614 37.73836825980029)	-121.770696	37.738368
5	POINT (-121.7335062173786 37.70094251246452)	-121.733506	37.700943
6	POINT (-121.7335062173786 37.70094251246452)	-121.733506	37.700943

	Total population	White	Black or African American
0	1643700	0.41475	0.107766
2	1643700	0.41475	0.107766
3	1643700	0.41475	0.107766
5	1643700	0.41475	0.107766
6	1643700	0.41475	0.107766

	American Indian and Alaska Native	Asian
0	0.006517	0.295938
2	0.006517	0.295938
3	0.006517	0.295938
5	0.006517	0.295938
6	0.006517	0.295938

	Native Hawaiian and Other Pacific Islander	Some other race
0	0.008376	0.103286
2	0.008376	0.103286

```
3                                    0.008376         0.103286
5                                    0.008376         0.103286
6                                    0.008376         0.103286

   Two or more races  Total housing units     Male   Female  \
0           0.063366               601942  0.49107  0.50893
2           0.063366               601942  0.49107  0.50893
3           0.063366               601942  0.49107  0.50893
5           0.063366               601942  0.49107  0.50893
6           0.063366               601942  0.49107  0.50893

   Minimum temperature  Maximum temperature  Minimum humidity  \
0           277.434532           287.580572          56.68057
2           277.434532           287.580572          56.68057
3           277.434532           287.580572          56.68057
5           277.434532           287.580572          56.68057
6           277.434532           287.580572          56.68057

   Maximum humidity  Wind speed  Solar radiation  Precipitation  Labor force  \
0         91.951602    3.883869       146.940284       0.000027     0.521446
2         91.951602    3.883869       146.940284       0.000027     0.521446
3         91.951602    3.883869       146.940284       0.000027     0.521446
5         91.951602    3.883869       146.940284       0.000027     0.521446
6         91.951602    3.883869       146.940284       0.000027     0.521446

   Rate unemployed  Average weekly wages
0            0.034                  1555
2            0.034                  1555
3            0.034                  1555
5            0.034                  1555
6            0.034                  1555
```

1.6 Data Summary and Exploratory Data Analysis (10 points)

EDA 1: Map of all PG&E substations in California We are going to create a map showing the locations of all the PG&E substations in California. We do this by reading in a dataset containing information about the substations in the form of a geopandas dataframe. Next, because we want to use the maps feature from the datascience module to create our map of all the substations in california, we must change the coordinate system of the geometry column in the geopandas dataframe to the EPSG 4326 Coordinate Reference System (CRS) that is recognized by the maps feature of the datascience module. Finally, we wrote a function to assign a red color to locations of substations that had their load profile informations redacted due to confidentiality reasons while substations that did not have their load profile informations redacted were assigned a green color.

NOTE: May need to check if the notebook is "trusted" for the map to display. (File>Trust Notebook)

[10]:
```
from datascience import *
import geopandas as gpd
# The first dataset we will import is really important
# It is a list of all substations (granularity = substation name or substation␣
 ↪ID)
# It also has the minimum voltage, NUMBANKS, REDACTED, OBJECTID, UNGROUNDEDBANKS
substation_df = gpd.read_file("a0000001f.gdbtable")

# the Coordinate Reference System (CRS) of this dataset is EPSG 26910
# but the most common CRS we use is EPSG 4326 and switching coordinate systems␣
 ↪will also help us map the substations
# So we are going to change this CRS using geopandas.to_crs
substation_df = substation_df.to_crs(epsg=4326)

# After we have converted the coorinate system, we want to collect and store␣
 ↪the latitudes & Longitudes of each susbation
substation_longitudes = substation_df["geometry"].x
substation_latitudes = substation_df["geometry"].y

# Realize now that there is a redacted column for which some substations have␣
 ↪entry "yes" and others have entry "no"
# Substations with entry "no" will not have load profile data in the other␣
 ↪dataframes so we want to remove them

# Redacted entries
substation_df_redacted = substation_df[substation_df["REDACTED"]=="Yes"]
substation_longitudes_redacted = substation_df_redacted["geometry"].x
substation_latitudes_redacted = substation_df_redacted["geometry"].y

# Non-Redacted entries
substation_df_not_redacted = substation_df[substation_df["REDACTED"]=="No"]
substation_longitudes_not_redacted = substation_df_not_redacted["geometry"].x
substation_latitudes_not_redacted = substation_df_not_redacted["geometry"].y

substation_df.head(2)
substations_list = substation_df["SUBNAME"]
Redacted_Information = substation_df["REDACTED"]
minimum_voltage = substation_df["MIN_KV"]

def color_fn(redacted_or_not):
    if redacted_or_not == 'No':
        return 'green'
    else:
        return 'red'

Substation_table = Table().
 ↪with_columns("Substation",substations_list,"Latitude",substation_latitudes,"Longitude",subs
```

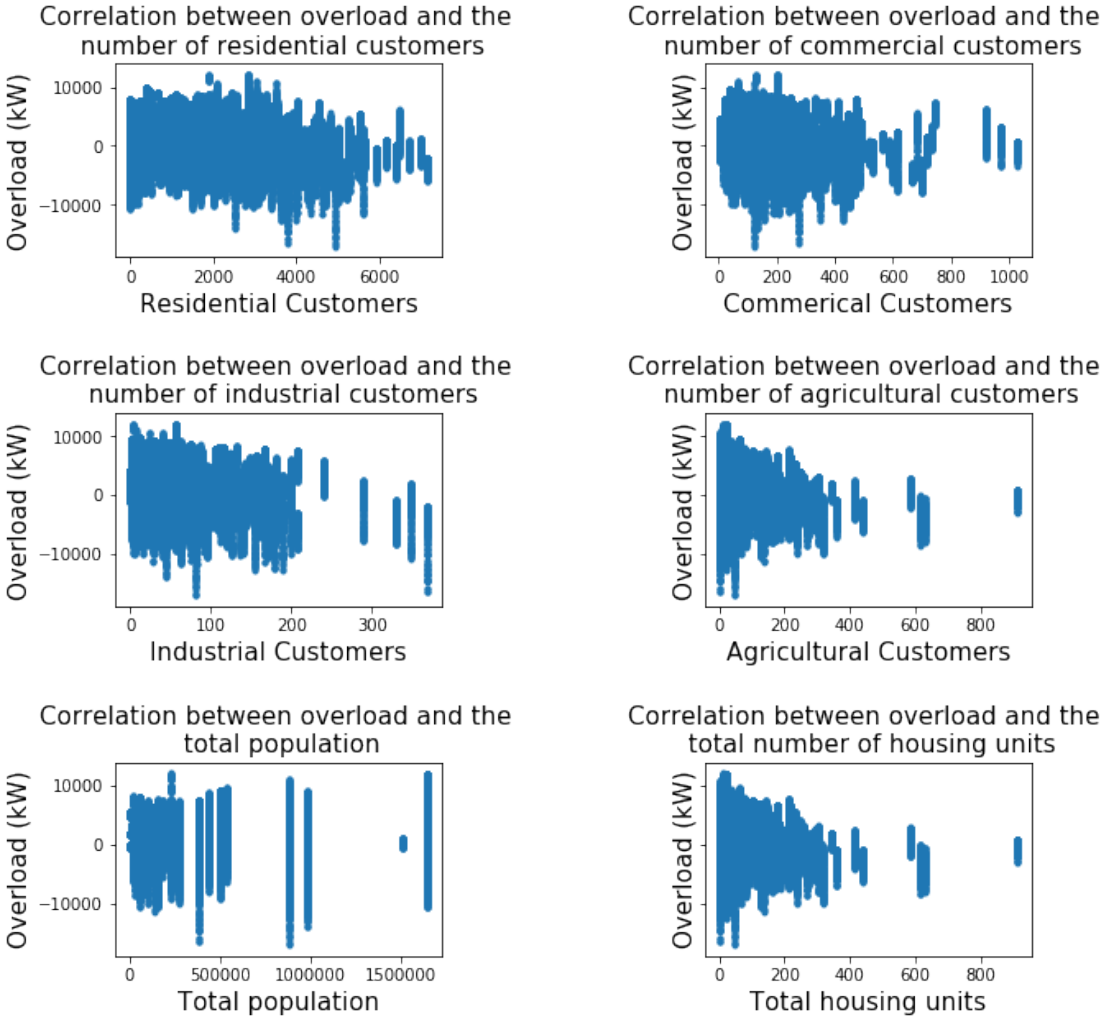
```
t = Table().with_columns([
    'Substation Latitudes',substation_latitudes,
    'Substation Longitudes',substation_longitudes,
    'label',minimum_voltage,
    'color',Substation_table.apply(color_fn,"Redacted_Information"),
    'area',10*minimum_voltage
])
Circle.map_table(t)
```

[10]: <datascience.maps.Map at 0x7fd71c0eefd0>

EDA 2: Correlation plots between overload and some features Next, we tried to visualize the correlation between our response variable, which is the overload, and several other features. The features that we tried were the number of residential customers, commercial customers, industrial customers, agricultural customers, total population and the total number of housing units. We can see that for all of these correlation plots, the overload values are somewhat equally distributed about the zero value indicating a weak correlation between the overload value and each of these features

[11]:
```
%matplotlib inline
fig,ax = plt.subplots(nrows=3,ncols=2,figsize=(10,10),sharey=True)
plt.subplots_adjust(hspace=0.8,wspace=0.8)
ax[0,0].scatter(master_df["ResCust"],master_df["overload"],marker='.');
ax[0,0].set_ylabel("Overload (kW)",fontsize=15);
ax[0,0].set_xlabel("Residential Customers",fontsize=15);
ax[0,0].set_title("Correlation between overload and the \n number of␣
 ↪residential customers",fontsize=15);

ax[0,1].scatter(master_df["ComCust"],master_df["overload"],marker='.');
ax[0,1].set_ylabel("Overload (kW)",fontsize=15);
ax[0,1].set_xlabel("Commerical Customers",fontsize=15);
ax[0,1].set_title("Correlation between overload and the \n number of commercial␣
 ↪customers",fontsize=15);

ax[1,0].scatter(master_df["IndCust"],master_df["overload"],marker='.');
ax[1,0].set_ylabel("Overload (kW)",fontsize=15);
ax[1,0].set_xlabel("Industrial Customers",fontsize=15);
ax[1,0].set_title("Correlation between overload and the \n number of industrial␣
 ↪customers",fontsize=15);

ax[1,1].scatter(master_df["AgrCust"],master_df["overload"],marker='.');
ax[1,1].set_ylabel("Overload (kW)",fontsize=15);
ax[1,1].set_xlabel("Agricultural Customers",fontsize=15);
ax[1,1].set_title("Correlation between overload and the \n number of␣
 ↪agricultural customers",fontsize=15);
```

```
ax[2,0].scatter(master_df["Total population"],master_df["overload"],marker='.');
ax[2,0].set_ylabel("Overload (kW)",fontsize=15);
ax[2,0].set_xlabel("Total population",fontsize=15);
ax[2,0].set_title("Correlation between overload and the \n total␣
 ↪population",fontsize=15);

ax[2,1].scatter(master_df["AgrCust"],master_df["overload"],marker='.');
ax[2,1].set_ylabel("Overload (kW)",fontsize=15);
ax[2,1].set_xlabel("Total housing units",fontsize=15);
ax[2,1].set_title("Correlation between overload and the \n total number of␣
 ↪housing units",fontsize=15);
```



EDA 3: Tabulating overload values for user's choice of feeder and month from drop-down selection We also wanted to give users the chance to pick a certain feeder and a certain month for that feeder to see if it would be possible to visualize the load profiles for that feeder and

month as a function of time. Since the Matplotlib GUI is not compatible with Ipywidgets, however, we have tabulated the values for each feeder and month. So below, we have created two dropdown boxes in which a user can choose a county and a month for that county and our function prints out tabulated values of the overload with time.

[12]:
```
counties_list_df = pd.DataFrame(master_df["County"].unique())
counties_list_df.to_csv("counties_list.csv",index=False)
counties_list_df = pd.read_csv("counties_list.csv")
counties_list = list(counties_list_df.iloc[:,0])

month_list = list([1,2,3,4,5,6,7,8,9,10,11,12])

Feeder_Name_list_df = pd.DataFrame(master_df["Feeder_Name"].unique())
Feeder_Name_list_df.to_csv("Feeder_Name.csv",index=False)
Feeder_Name_list_df = pd.read_csv("Feeder_Name.csv")
Feeder_Name_list = list(Feeder_Name_list_df.iloc[:,0])


def feeder_profile_plotter(Feeder,month):
    df = master_df[master_df["Feeder_Name"]==Feeder]
    df = df[df["Month"]==month]
    return df
```

[13]:
```
import ipywidgets as widgets
_ = widgets.interact(feeder_profile_plotter,
                     month= list(month_list),
                     Feeder= list(Feeder_Name_list))
```

```
interactive(children=(Dropdown(description='Feeder', options=('CAYETANO 2111', 'VASCO 1101', '
```

EDA 4: Map of overloaded (in red) and underloaded (in green) features per county (user can select county using dropdown menu and wait for map to reload) The next map we created was so that we could visualize the feeders within a certain county and see which feeders were overloaded or underloaded. We used the Ipywidgets module to create a dropdown box that would allow a user to select which county they wanted to take a look at. Next, we created a function that would filter the master dataframe to extract only the entries belonging to the county that the user selected and plotted these feeders onto the map. It might take some time for the map to reload but essentially, one can see the underloaded feeders (in green) or overloaded feeders (in red) for each county using the dropdown box.

[ ]:
```
# Now, we are going to create a map with a dropdown box
# Through this dropdown box, we will be able to select a certain county
# And once we select a county, we will be able to see which feeders in the␣
 ↪county have been overloaded
# Overloaded feeders will show up as red circles
# While underloaded feeders will show up as green circles
# All circles will have the same area
```

```
# When each county is seletected, give some time for the map to reload
counties_list_df = pd.DataFrame(master_df["County"].unique())
counties_list_df.to_csv("counties_list.csv",index=False)
counties_list_df = pd.read_csv("counties_list.csv")
counties_list = list(counties_list_df.iloc[:,0])
counties_list

def country_profile(county): # Create a function to filter out the dataframe␣
↪based on county
    county_df = master_df[master_df["County"]==county]

    feeder_names = county_df["Feeder_Name"]
    feeder_longitudes = county_df["Longitude"]
    feeder_latitudes = county_df["Latitudes"]
    overload = county_df["overload"]


    Feeder_table = Table().with_columns("Feeder",feeder_names,
                                        "Latitude",feeder_latitudes,
                                        "Longitude",feeder_longitudes,
                                        "Overload",overload)

    def color_fn(overload): # Create a function for assigning color based on␣
↪load
        if overload > 0: # If this value is positive, that is, the system is␣
↪underloaded, then display green
            return 'green'
        elif overload == 0 : # If this value is zero, that is system is neither␣
↪under or overloaded, display blue
            return 'blue'
        elif overload < 0: # If this value is less than zero, that is system is␣
↪overloaded, display red
            return 'red'

    t = Table().with_columns([
    'Feeder Latitudes',feeder_latitudes,
    'Feeder Longitudes',feeder_longitudes,
    'label',overload,
    'color',Feeder_table.apply(color_fn,"Overload"),
    'area',100]) # Let all circles have the same area

    return Circle.map_table(t)

import ipywidgets as widgets
_ = widgets.interact(country_profile, county = list(counties_list))
```

```
interactive(children=(Dropdown(description='county', options=('Alameda', 'Alpine', 'Amador', '
```

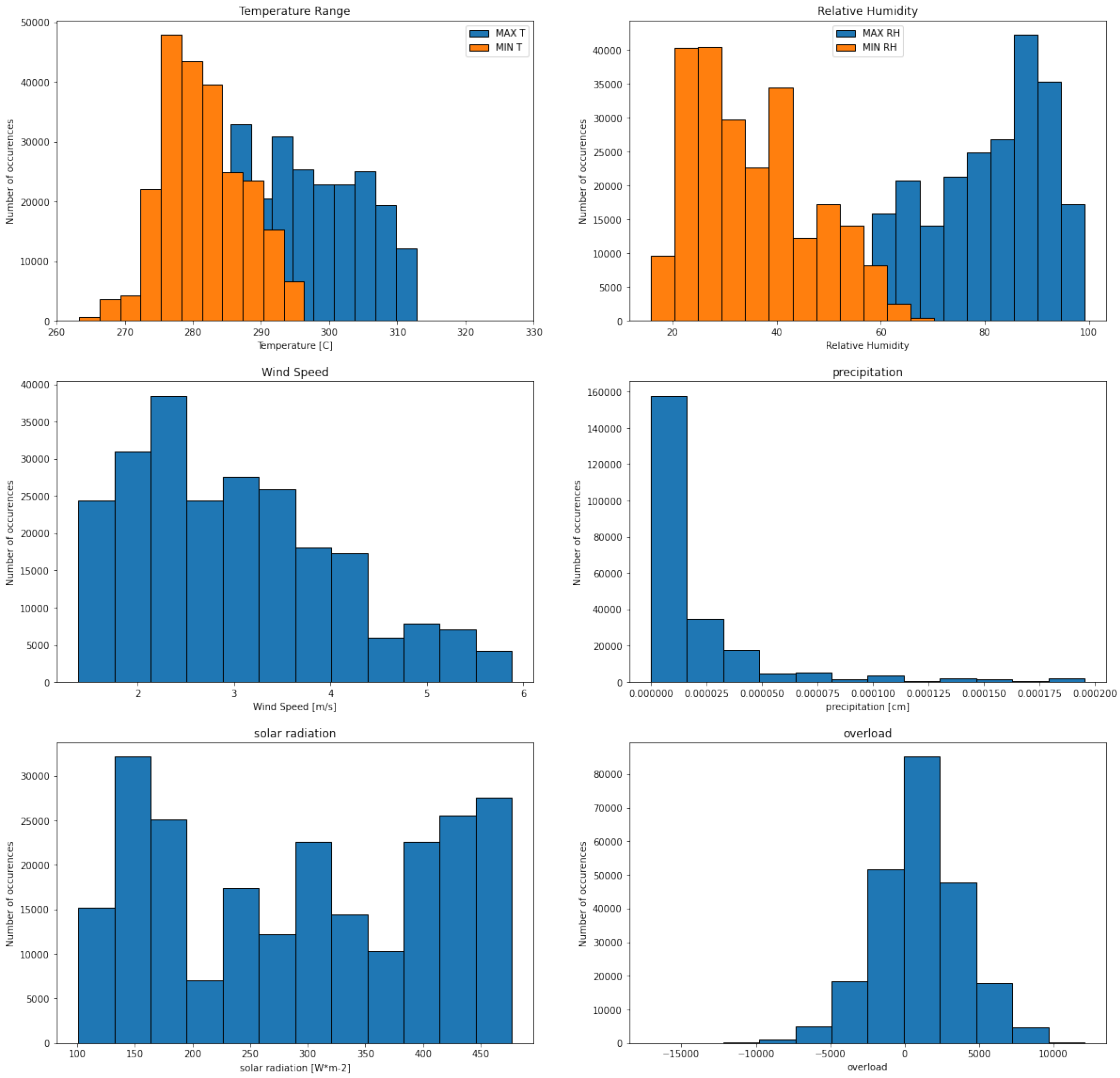
EDA 5: Range and frequencies of features There is a wide range of values for `overload`.

For temperature and relative humidity, the minimum values are slightly skewed towards the left end of their distributions. Maximum RH is right-skewed and maximum temperature seems unskewed, with their mean values higher than the mean values of minimum RH and minimum temperature, as expected. Wind speed is slightly left-skewed and precipitation is strongly left-skewed. We don't see a very clear pattern for solar radiation.

[24]:
```
#doing some EDA on the forecasted weather data and looking at the range of␣
↪overload values

fig=plt.figure(figsize=[20,20])
plt.subplot(3,2,1)
plt.hist(master_df['Maximum temperature'], bins =12,label='MAX T')
plt.hist(master_df['Minimum temperature'], bins =12,label='MIN T')
plt.title("Temperature Range")
plt.xlabel("Temperature [C]")
plt.xlim(260,330)
plt.ylabel('Number of occurences')
plt.legend()

plt.subplot(3,2,2)
plt.hist(master_df['Maximum humidity'], bins =12,label='MAX RH')
plt.hist(master_df['Minimum humidity'], bins =12,label='MIN RH')
plt.title("Relative Humidity")
plt.xlabel("Relative Humidity")
#plt.xlim(260,330)
plt.ylabel('Number of occurences')
plt.legend()

plt.subplot(3,2,3)
plt.hist(master_df['Wind speed'], bins =12)
plt.title("Wind Speed")
plt.xlabel("Wind Speed [m/s]")
plt.ylabel('Number of occurences')

plt.subplot(3,2,4)
plt.hist(master_df['Precipitation'], bins =12)
plt.title("precipitation")
plt.xlabel("precipitation [cm]")
plt.ylabel('Number of occurences')

plt.subplot(3,2,5)
plt.hist(master_df['Solar radiation'], bins =12)
plt.title('solar radiation')
plt.xlabel('solar radiation [W*m-2]')
```

```
plt.ylabel('Number of occurences')

plt.subplot(3,2,6)
plt.hist(master_df['overload'], bins =12)
plt.title('overload')
plt.xlabel('overload')
plt.ylabel('Number of occurences')
```

[24]: Text(0, 0.5, 'Number of occurences')



1.7 Forecasting and Prediction Modeling (25 points)

There are 3 sections below, each for one of the 3 prediction problems (prediction in space, time, and with a time lag). We investigated 4 prediction modeling approaches for each (OLS, Ridge, Lasso

and KNN regression). Since all the linear regression models resulted in very high MSE, this led us to believe that classification or nonlinear modeling methods may be a better fit for the data, and we thus used KNN to test out a non-parametric model. Although classification would remove the nuance a regression problem would give us of how far from overload a substation is, it may return better results since there is less variability.

Import necessary packages for processing dataframes and for prediction pipeline

[25]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
```

We created some key functions to do 3 things:

1. Create a train test split
2. Create a a standardized train-test split
3. a Function to fit different regression models

For our various models, we chose to do a 25/75 split for our training testing data as this is deemed a reasonable split for datasets with lots of values. We chose to perform a 5-fold cross validation on all the models except for the simple linear regression to prevent overfitting and to ensure our models more reasonably represent what is happening. We also chose to standardize the features so that our Lasso model pushes more coefficients to zero and so our Ridge models push the coefficients closer to zero. When we don't normalize, variables with a large range are relatively less subject to the shrinkage methods than are variables with a small range.

Models we chose to use: 1. Linear regression 2. Linear regression with Lasso 3. Linear regression with Ridge 4. KNN

We also engineered a couple of our own features by squaring specific features that had some of the larger coefficients when we performed linear regression and added them to our dataset. We used these features to see if we could identify the nonlinear correlations between our features and overload such as population and temperature. We made this decision to investigate if this would help improve our model and decrease the bias present in our regression models. These engineered features didn't improve the MSE of our models significantly, so we chose not to use them. Below is the code we used to try to add them.

[26]:
```
#picked a couple features to try squaring to make artifical features since our␣
↪initial results yielded bad regression models
```

```
# columns_to_square = ['Total population', 'Maximum temperature','Minimum␣
↪temperature', 'Maximum temperature', 'Minimum humidity',
#        'Maximum humidity', 'Wind speed', 'Solar radiation', 'Precipitation',
#        'Labor force', 'Rate unemployed', 'Average weekly wages', 'ResCust',␣
↪'ComCust', 'Total_DG']

# for column in columns_to_square:
#     full_dataset[column+' squared'] = full_dataset[column]**2
```

[27]:
```
# define some key functions

# creates train-test split without standardizing (we used this to test the␣
↪effect of standardizing and
# did not observe a notable difference, but stuck to using standardized␣
↪variables)
def create_train_test(X,y,test_size, random_state):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test

# standardizes and creates train-test split
def get_X_y_stnd(X,y,test_prop, rand_seed):
    """
    This function returns four dataframes containing the testing and training X␣
↪and y values used in land-use regression.
    It standardizes the X variables prior to splitting the data.
    Input: df, a Pandas dataframe;
        test_prop, a float between 0 and 1 indicating the fraction of the data␣
↪to include in the test split
        rand_seed, an integer, used to define the random state
    Returns: X_train, X_test, y_train, y_test, four dataframes containing the␣
↪training and testing subsets of the
    feature matrix X and response matrix y
    """

    # Standardize the data
    scaler = StandardScaler()
    scaler.fit(X)
    X_stnd = scaler.transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_stnd, y,␣
↪test_size=test_prop, random_state = rand_seed)

    return X_train, X_test, y_train, y_test
```

```
# (Only for KNN) Returns the optimal K value and all the scores for the K␣
 ↪values, the latter for visualizations
def choose_optimal_K(X_train, y_train, K_values):
    cv_scores = []
    # 5-fold cross-validation
    for K in K_values:
        model = KNeighborsRegressor(n_neighbors = K)
        scores = cross_val_score(model, X_train, y_train, scoring =␣
 ↪"neg_mean_squared_error")
        cv_scores.append(scores.mean())

    return np.argmax(cv_scores)+1, cv_scores


# fits data to the chosen model and does cross-validation if applicable. Tests␣
 ↪the fit with test data
# at the very end. Outputs training mse (mse_train), coefficients, r^2,␣
 ↪residuals and testing mse (mse)
def fit_model(Model, X_train, X_test, y_train, y_test, n_splits, random_state,␣
 ↪alphas_array, K=5):
    """
    This function fits a model of type Model to the data in the training set of␣
 ↪X and y, and finds the MSE on the test set
    Inputs:
        Model (sklearn model): the type of sklearn model with which to fit the␣
 ↪data - LinearRegression, RidgeCV, LassoCV, or KNeighborsRegressor
        X_train: the set of features used to train the model
        y_train: the set of response variable observations used to train the␣
 ↪model
        X_test: the set of features used to test the model
        y_test: the set of response variable observations used to test the model
        n_splits: number of splits, to be used with RidgeCV and LassoCV models␣
 ↪only
        random_state: random state for cross validation, to be used with␣
 ↪RidgeCV and LassoCV models only
        alphas_array: array of alpha values to be tested, to be used with␣
 ↪RidgeCV and LassoCV models only
        K: number of neighbors for KNeighorsRegressor models, w/ default value 5
    """

    if Model == LinearRegression:
        model = Model()
        model.fit(X_train, y_train)
        alpha = 'N/A'
    elif Model == LassoCV:
```

```
        kf = KFold(n_splits = n_splits, shuffle = True, random_state =␣
↪random_state)
        model = Model(cv = kf, alphas=alphas_array, max_iter = 10000)
        model.fit(X_train,y_train)
        alpha = model.alpha_
        print("optimal alpha (Lasso):", alpha)
    elif Model == RidgeCV:
        kf = KFold(n_splits = n_splits, shuffle = True, random_state =␣
↪random_state)
        model = Model(cv = kf, alphas=alphas_array)
        model.fit(X_train,y_train)
        alpha = model.alpha_
        print("optimal alpha (Ridge):", alpha)
    else:  # KNeighborsRegressor
        model = Model(n_neighbors = 2)
        model.fit(X_train, y_train)
        alpha = 'N/A'

    mse = mean_squared_error(y_test, model.predict(X_test))
    coef = (model.coef_.flatten() if Model != KNeighborsRegressor else 'N/A')
    r2 = r2_score(y_test, model.predict(X_test))

    residual = model.predict(X_test) - y_test
    mse_train = mean_squared_error(y_train, model.predict(X_train))

    return mse, coef, r2, residual, mse_train
```

[28]:
```
# Visualizing the magnitudes of coefficients for the different methods
def visualize_coef(coef_):
    ind = np.arange(coef_.shape[1])
    width = 0.25
    pos = np.array([ind - width, ind, ind + width])
    modelNames = ["Linear regression", "Lasso", "Ridge"]

    fig = plt.figure(figsize = (15,6))

    # plt.subplot(211)
    for i in np.arange(coef_.shape[0]):
        plt.bar(pos[i], height = coef_[i,:], width = width, label =␣
↪modelNames[i])
    plt.legend()
    plt.xlabel("Feature number")
    plt.ylabel("Feature coefficients")
    plt.title("Model comparisons")
    plt.yscale('symlog')

    plt.tight_layout()
```

```
    plt.show()
```

[29]:
```
#plots the residuals of the regression models to evaluate bias

#pred_num is the number of the prediction (0=spatial, 1=temporal, 2=timelag)
def plotresiduals(resid_LR, resid_Lasso, resid_Ridge, y_test, pred_num):
    pred_type=['Spatial','Temporal','Time Lag']
    ypred_LR = resid_LR + y_test
    ypred_Lasso = resid_Lasso + y_test
    ypred_Ridge = resid_Ridge + y_test

    fig = plt.figure(figsize = (20,16))
    fig.suptitle(pred_type[pred_num] + ' Regression Results', fontsize=18)

    # consider maybe only visualizing one of OLS/Ridge/Lasso since they're so␣
 ↪similar

    plt.subplot(321)
    plt.plot(y_test, y_test, color = 'red')
    plt.scatter(ypred_LR, y_test)
    plt.title('Predicted vs actual overload using OLS regression')
    plt.ylabel('Predicted overload (kW)')
    plt.xlabel('Actual overload (kW)');

    plt.subplot(322)
    plt.axhline(0, color = 'r')
    plt.scatter(resid_LR, y_test)
    plt.title('Residual vs actual overload using OLS regression')
    plt.ylabel('Residual (kW)')
    plt.xlabel('Overload (kW)');

    plt.subplot(323)
    plt.plot(y_test, y_test, color = 'red')
    plt.scatter(ypred_Lasso, y_test)
    plt.title('Predicted vs actual overload using Lasso regression')
    plt.ylabel('Predicted overload (kW)')
    plt.xlabel('Actual overload (kW)');

    plt.subplot(324)
    plt.scatter(resid_Lasso, y_test)
    plt.axhline(0, color = 'r')
    plt.title('Residual vs actual overload using Lasso regression')
    plt.ylabel('Residual (kW)')
    plt.xlabel('Overload (kW)');

    plt.subplot(325)
    plt.plot(y_test, y_test, color = 'red')
```

```
    plt.scatter(ypred_Ridge, y_test)
    plt.title('Predicted vs actual overload using Ridge regression')
    plt.ylabel('Predicted overload (kW)')
    plt.xlabel('Actual overload (kW)');

    plt.subplot(326)
    plt.scatter(resid_Ridge, y_test)
    plt.axhline(0, color = 'r')
    plt.title('Residual vs actual overload using Ridge regression')
    plt.ylabel('Residual (kW)')
    plt.xlabel('Overload (kW)');

    plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top=0.9, wspace=0.2,␣
 ↪hspace=0.3)
```

[30]:
```
# replotting residuals to compare the biases of regression models and KNN
# we decided to visualize only one of OLS/Ridge/Lasso for comparison's sake␣
 ↪since they're so similar

#pred_num is the number of the prediction (0=spatial, 1=temporal, 2=timelag)
def plotresiduals_re(resid_LR, resid_KNN, y_test, pred_num):
    pred_type=['Spatial','Temporal','Time Lag']
    ypred_LR = resid_LR + y_test
    ypred_KNN = resid_KNN + y_test

    fig = plt.figure(figsize = (14,10))
    fig.suptitle(pred_type[pred_num] + ' Model Results', fontsize=18)

    plt.subplot(221)
    plt.plot(y_test, y_test, color = 'red')
    plt.scatter(ypred_LR, y_test)
    plt.title('Predicted vs actual overload using OLS regression')
    plt.ylabel('Predicted overload (kW)')
    plt.xlabel('Actual overload (kW)');

    plt.subplot(222)
    plt.axhline(0, color = 'r')
    plt.scatter(resid_LR, y_test)
    plt.title('Residual vs actual overload using OLS regression')
    plt.ylabel('Residual (kW)')
    plt.xlabel('Overload (kW)');

    plt.subplot(223)
    plt.plot(y_test, y_test, color = 'red')
    plt.scatter(ypred_KNN, y_test)
    plt.title('Predicted vs actual overload using KNN regression')
    plt.ylabel('Predicted overload (kW)')
```

```
    plt.xlabel('Actual overload (kW)');

    plt.subplot(224)
    plt.scatter(resid_KNN, y_test)
    plt.axhline(0, color = 'r')
    plt.title('Residual vs actual overload using KNN regression')
    plt.ylabel('Residual (kW)')
    plt.xlabel('Overload (kW)');


    plt.subplots_adjust(left=0.125, bottom=0.3, right=0.9, top=0.9, wspace=0.3,␣
 ↪hspace=0.3)
```

[31]:
```
# dropping columns that we don't want to keep for some of the prediction␣
 ↪problems
# dropping feeder info because we don't use this anywhere
# this dataset already has MonthHour split into two separate columns
cleaned_dataset = full_dataset.drop(columns = ['FeederID','SUBSTATIONID',␣
 ↪'geometry', 'Feeder_Name', 'High'])
cleaned_dataset.head()
```

[31]:

	County	Month	Hour	LoadCapacity_kW	overload	Substation	ResCust
0	Alameda	1	1	10000	4065	CAYETANO	1748
2	Alameda	1	1	4480	3300	VASCO	1626
3	Alameda	1	1	10000	9548	CAYETANO	2849
5	Alameda	1	1	9260	2493	LAS POSITAS	995
6	Alameda	1	1	9180	5749	LAS POSITAS	4550

	ComCust	IndCust	AgrCust	OthCust	Existing_DG	Queued_DG	Total_DG
0	171	121	4	10	3550	20	3570
2	50	28	0	4	530	10	540
3	205	58	23	9	7580	3130	10710
5	280	136	2	11	3340	350	3690
6	143	13	0	8	4030	250	4280

	Longitude	Latitudes	Total population	White
0	-121.770696	37.738368	1643700	0.41475
2	-121.715677	37.710972	1643700	0.41475
3	-121.770696	37.738368	1643700	0.41475
5	-121.733506	37.700943	1643700	0.41475
6	-121.733506	37.700943	1643700	0.41475

	Black or African American	American Indian and Alaska Native	Asian
0	0.107766	0.006517	0.295938
2	0.107766	0.006517	0.295938
3	0.107766	0.006517	0.295938
5	0.107766	0.006517	0.295938

```
6                   0.107766                              0.006517  0.295938

   Native Hawaiian and Other Pacific Islander  Some other race  \
0                                    0.008376         0.103286
2                                    0.008376         0.103286
3                                    0.008376         0.103286
5                                    0.008376         0.103286
6                                    0.008376         0.103286

   Two or more races  Total housing units     Male   Female  \
0           0.063366               601942  0.49107  0.50893
2           0.063366               601942  0.49107  0.50893
3           0.063366               601942  0.49107  0.50893
5           0.063366               601942  0.49107  0.50893
6           0.063366               601942  0.49107  0.50893

   Minimum temperature  Maximum temperature  Minimum humidity  \
0           277.434532           287.580572          56.68057
2           277.434532           287.580572          56.68057
3           277.434532           287.580572          56.68057
5           277.434532           287.580572          56.68057
6           277.434532           287.580572          56.68057

   Maximum humidity  Wind speed  Solar radiation  Precipitation  Labor force  \
0         91.951602    3.883869       146.940284       0.000027     0.521446
2         91.951602    3.883869       146.940284       0.000027     0.521446
3         91.951602    3.883869       146.940284       0.000027     0.521446
5         91.951602    3.883869       146.940284       0.000027     0.521446
6         91.951602    3.883869       146.940284       0.000027     0.521446

   Rate unemployed  Average weekly wages
0            0.034                  1555
2            0.034                  1555
3            0.034                  1555
5            0.034                  1555
6            0.034                  1555
```

1.7.1 Spatial Prediction (for June 2019)

This prediction problem aims to predict overload spatially. To that end, at a particular time (June 2019) was chosen and only the data that corresponded to this month was retained from the full dataset (`full_dataset`). We also tried predictions for a particular hour of a particular month, but predictions were worse, probably because some of the features such as weather-related data were month-averaged data, and because fewer observations were used to train the model. All temporal columns were dropped from the feature matrix. The spatial granularity is substation level. The features that provide spatial information are latitudes and longitude, which are numeric and thus suitable for regression techniques. The response variable is `overload`, which is overload of the

substation in units kW. Negative values indicate an overload, meaning there is more demand than capacity, and positive values indicate an underload. A 0 value would indicate perfectly matched demand and capacity.

Four models were used in this section, namely ordinary least squares regression (OLS), lasso regression (Lasso), ridge regression (Ridge) and K-Nearest Neighbors regression (KNN). Our choice of regression methods is based on the fact that we would like to predict a value for overload or underload. All our features are numeric and easily lend themselves to regression without the need for data transformation or one-hot-encoding. While we could also use classification techniques to predict if a particular location might have overload or underload, this would remove the nuance of how much overload or underload there is, which is crucial to the resource allocation problem of where to reroute power and how much. All our features were standardized in this analysis because this is necessary for a valid result for Lasso and Ridge.

We used OLS as the simplest model because we believed that there would be a high correlation between energy usage and our feature choices such as temperature and humidity - for example, if the weather is hotter, we expected more energy use for applications such as fans and air conditioning.

As an extension to OLS, we tried Ridge because this would introduce a regularization penalty for coefficients that were too big, which would reduce the possibility of overfitting by penalizing a more complex model. The hyperparameter alpha was selected by feeding a wide range of possible alpha values into RidgeCV, which does cross-validation to select the best alpha by minimizing MSE. 5 folds were chosen for cross-validation since this is generally a reasonable number of folds, and though more folds could give us more accurate results, it would be computationally too expensive since we were limited by our use of Datahub. The optimal alpha was 0.001, which was the minimum value we inputted and very close to 0 - indicating that little penalty was applied and perhaps OLS was indeed good enough. The results from this model were very similar to OLS.

Since we have many features and they were all included because we thought they would be useful but we don't know if that is true, Lasso regression was a natural choice to add a regularization penalty and perform a more rigorous feature selection, since it would set irrelevant features to have a coefficient of 0. Similar to Ridge, we used 5 folds for cross-validation in LassoCV. Since LassoCV allows specifying the max number of iterations, we increased it from the default 1000 to 10000 but did not raise it further because it is computationally too expensive. Surprisingly, Lasso removed no features and also returned an alpha value of 0.001, and very similar results to OLS.

Since the above three methods all resulted in similar, poor results (R^2 values close to 0.2), we hypothesized that a linear model may not be suitable to explain our data. We thus tried KNN to investigate if perhaps a non-parametric method would provide a better fit. The k value was chosen by feeding in possible k values from 2 to 25 into in-built methods of cross-validation in sklearn to select the value that minimized MSE. Similar to Lasso and Ridge, 5 folds were used for cross-validation.

[32]:
```
# note that this dataset is NOT linearly independent - there's collinearity␣
↪(total population vs all the races, genders)
# spatial_dataset = cleaned_dataset[(cleaned_dataset['Month'] == 6) &␣
↪(cleaned_dataset['Hour'] == 12) &
#                                   (cleaned_dataset['LoadCapacity_kW'] != 0)].
↪drop(columns =['Month', 'Hour', 'County']).set_index('Substation')
```

```
spatial_dataset = cleaned_dataset[cleaned_dataset['Month'] == 6].drop(columns=['Month', 'Hour', 'County']).set_index('Substation')
spatial_dataset.head()
```

[32]:

Substation	LoadCapacity_kW	overload	ResCust	ComCust	IndCust	AgrCust
CAYETANO	10000	7326	2849	205	58	23
CAYETANO	10000	3108	1748	171	121	4
VASCO	4480	3120	1626	50	28	0
LAS POSITAS	6270	583	5022	442	92	1
LAS POSITAS	6730	960	4271	335	157	0

Substation	OthCust	Existing_DG	Queued_DG	Total_DG	Longitude	Latitudes
CAYETANO	9	7580	3130	10710	-121.770696	37.738368
CAYETANO	10	3550	20	3570	-121.770696	37.738368
VASCO	4	530	10	540	-121.715677	37.710972
LAS POSITAS	14	4410	340	4750	-121.733506	37.700943
LAS POSITAS	13	3350	110	3460	-121.733506	37.700943

Substation	Total population	White	Black or African American
CAYETANO	1643700	0.41475	0.107766
CAYETANO	1643700	0.41475	0.107766
VASCO	1643700	0.41475	0.107766
LAS POSITAS	1643700	0.41475	0.107766
LAS POSITAS	1643700	0.41475	0.107766

Substation	American Indian and Alaska Native	Asian
CAYETANO	0.006517	0.295938
CAYETANO	0.006517	0.295938
VASCO	0.006517	0.295938
LAS POSITAS	0.006517	0.295938
LAS POSITAS	0.006517	0.295938

Substation	Native Hawaiian and Other Pacific Islander	Some other race
CAYETANO	0.008376	0.103286
CAYETANO	0.008376	0.103286
VASCO	0.008376	0.103286
LAS POSITAS	0.008376	0.103286
LAS POSITAS	0.008376	0.103286

Substation	Two or more races	Total housing units	Male	Female
CAYETANO	0.063366	601942	0.49107	0.50893

```
CAYETANO              0.063366              601942  0.49107  0.50893
VASCO                 0.063366              601942  0.49107  0.50893
LAS POSITAS           0.063366              601942  0.49107  0.50893
LAS POSITAS           0.063366              601942  0.49107  0.50893

            Minimum temperature  Maximum temperature  Minimum humidity  \
Substation
CAYETANO              285.453815           303.859305          28.14509
CAYETANO              285.453815           303.859305          28.14509
VASCO                 285.453815           303.859305          28.14509
LAS POSITAS           285.453815           303.859305          28.14509
LAS POSITAS           285.453815           303.859305          28.14509

            Maximum humidity  Wind speed  Solar radiation  Precipitation  \
Substation
CAYETANO           78.932004    5.162031       465.398074            0.0
CAYETANO           78.932004    5.162031       465.398074            0.0
VASCO              78.932004    5.162031       465.398074            0.0
LAS POSITAS        78.932004    5.162031       465.398074            0.0
LAS POSITAS        78.932004    5.162031       465.398074            0.0

            Labor force  Rate unemployed  Average weekly wages
Substation
CAYETANO       0.514267            0.031                  1498
CAYETANO       0.514267            0.031                  1498
VASCO          0.514267            0.031                  1498
LAS POSITAS    0.514267            0.031                  1498
LAS POSITAS    0.514267            0.031                  1498
```

[33]:
```
# split into feature and response variables and dropped some of the features␣
↪which were linearly dependent
Y_space = spatial_dataset['overload']
X_space = spatial_dataset.drop(columns = ['LoadCapacity_kW', 'overload'])
X_space = X_space.drop(columns = ['Female', 'Some other race', 'Queued_DG'])
```

[34]:
```
X_space_train, X_space_test, y_space_train, y_space_test =␣
↪get_X_y_stnd(X_space,Y_space,0.25, 2020)
```

[35]:
```
mse_space_LR, coef_space_LR, r2_space_LR, resid_space_LR, mse_train_space_LR =␣
↪fit_model(LinearRegression, X_space_train, X_space_test, y_space_train,␣
↪y_space_test, 5, 2020, np.linspace(0.001,100, 100))
mse_space_Lasso, coef_space_Lasso, r2_space_Lasso, resid_space_Lasso,␣
↪mse_train_space_Lasso = fit_model(LassoCV, X_space_train, X_space_test,␣
↪y_space_train, y_space_test, 5, 2020, np.linspace(0.001,100, 100))
mse_space_Ridge, coef_space_Ridge, r2_space_Ridge, resid_space_Ridge,␣
↪mse_train_space_Ridge = fit_model(RidgeCV, X_space_train, X_space_test,␣
↪y_space_train, y_space_test, 5, 2020, np.linspace(0.001,100, 100))
```

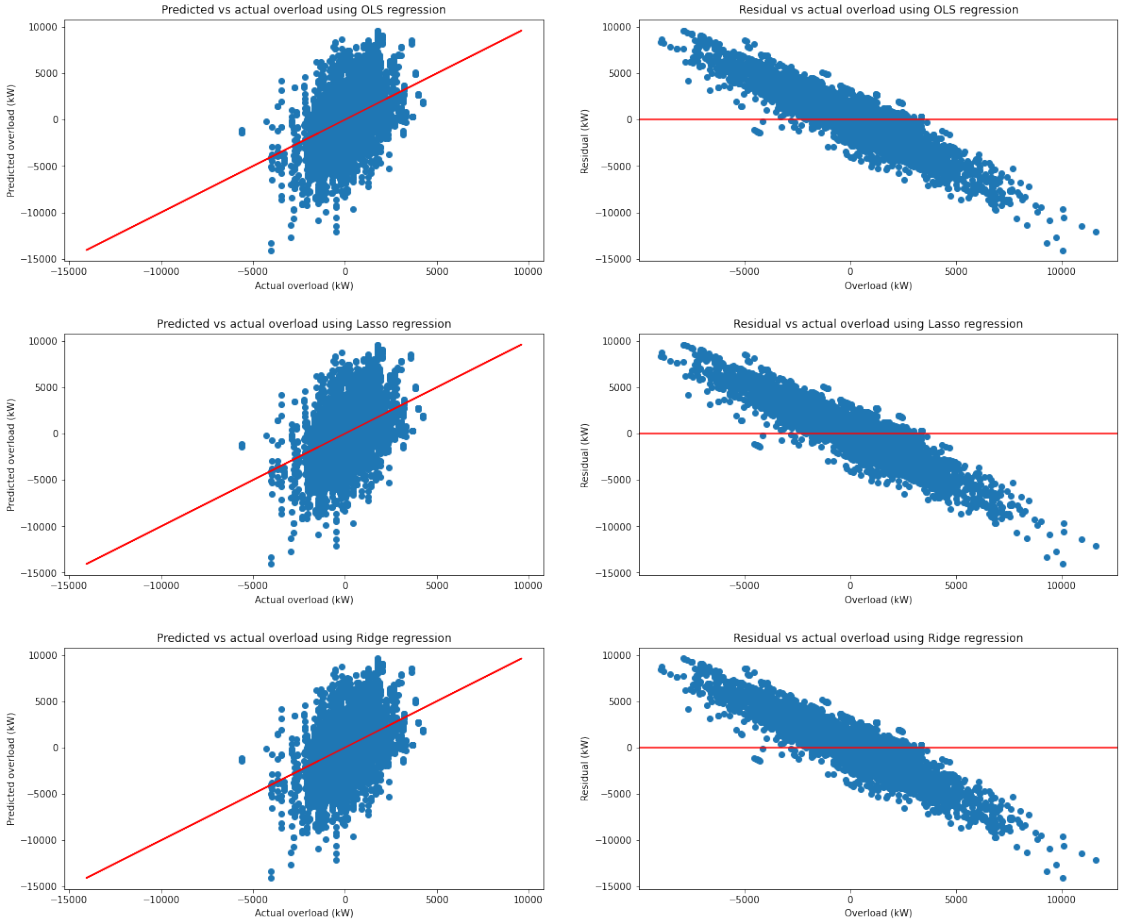
```
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 53879278.45198059, tolerance: 11354480.450348781
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 44546753513.23242, tolerance: 11354480.450348781
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 45592625.0501709, tolerance: 11287290.81822244
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 44227091271.24592, tolerance: 11287290.81822244
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 44230317.73371887, tolerance: 11358008.085096914
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 44705304051.51253, tolerance: 11358008.085096914
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 46723336.63449097, tolerance: 11412584.060403971
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 44905412168.021996, tolerance: 11412584.060403971
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 34580858.07426453, tolerance: 11497928.226730967
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
```

```
Duality gap: 44731945587.11847, tolerance: 11497928.226730967
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 56979851144.74487, tolerance: 14228121.438402152
  model = cd_fast.enet_coordinate_descent(

optimal alpha (Lasso): 0.001
optimal alpha (Ridge): 0.001
```

[36]:
```
mse_space = np.array([mse_space_LR, mse_space_Lasso, mse_space_Ridge])
coef_space = np.array([coef_space_LR, coef_space_Lasso, coef_space_Ridge])
r2_space = np.array([r2_space_LR, r2_space_Lasso, r2_space_Ridge])
resid_space = np.array([resid_space_LR, resid_space_Lasso, resid_space_Ridge])
mse_train_space = np.array([mse_train_space_LR, mse_train_space_Lasso,␣
 ↪mse_train_space_Ridge])

print('Test MSE:', mse_space)
print('Training MSE:', mse_train_space)
print('Residuals:', resid_space)
```

```
Test MSE: [7815549.42269661 7815340.11655848 7815545.56564946]
Training MSE: [8082526.98913197 8082540.37167735 8082526.99377489]
Residuals: [[ -470.05985598 -3814.61473982 -7204.05372069 …  -417.16726437
   -773.86120049   337.6861347 ]
 [ -468.32890886 -3821.29650008 -7204.43148605 …  -419.61879801
   -772.01674548   338.67620794]
 [ -470.0199601  -3814.74432294 -7204.06013949 …  -417.21279659
   -773.82579411   337.70561669]]
```

[37]:
```
# plots residual graphs
plotresiduals(resid_space_LR, resid_space_Lasso, resid_space_Ridge,␣
 ↪y_space_test, 0)

# Residual plots show that we have a lot of bias
# These plots also show very similar results for OLS, Lasso and Ridge - aligns␣
 ↪with the optimal alphas returned
# from cross-validation, 0.001 for both ridge and lasso. This was the smallest␣
 ↪value fed into the alpha choice array.
```



[38]:
```
features_space = np.array(X_space.columns)
```

[39]:
```
# this array shows the most highly correlated to least highly correlated␣
↪features for OLS
sorted_features_space_LR = features_space[(-abs(coef_space_LR)).argsort()]
sorted_features_space_LR
```

[39]:
```
array(['Total housing units', 'Total population', 'Precipitation',
       'Rate unemployed', 'Native Hawaiian and Other Pacific Islander',
       'Black or African American', 'ResCust', 'Longitude', 'IndCust',
       'AgrCust', 'Male', 'Latitudes', 'Asian', 'White', 'Wind speed',
       'Maximum humidity', 'Solar radiation', 'Maximum temperature',
       'Total_DG', 'Minimum humidity', 'Minimum temperature',
       'Two or more races', 'American Indian and Alaska Native',
       'Labor force', 'ComCust', 'OthCust', 'Existing_DG',
       'Average weekly wages'], dtype=object)
```

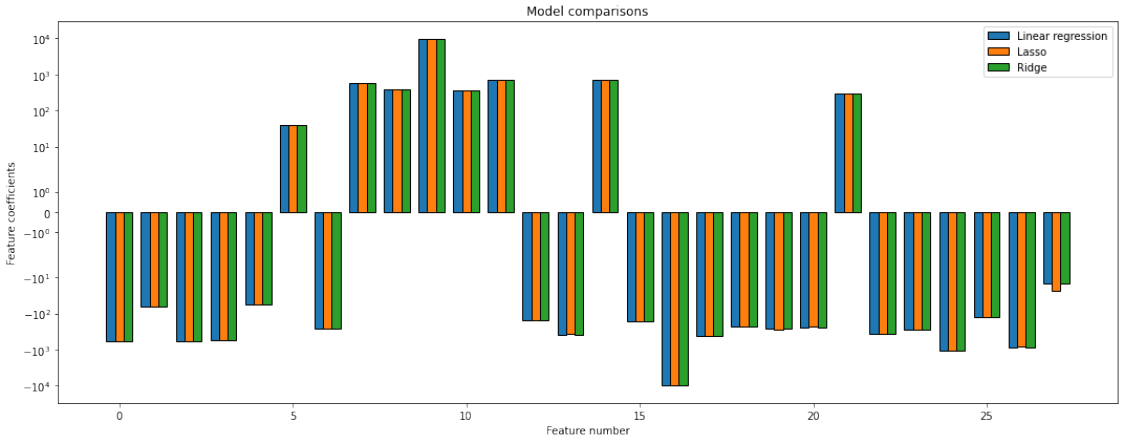
[40]:
```
# this array shows the most highly correlated to least highly correlated␣
↪features for Lasso
sorted_features_space_Lasso = features_space[(-abs(coef_space_Lasso)).argsort()]
sorted_features_space_Lasso
# similar to OLS
```

[40]:
```
array(['Total housing units', 'Total population', 'Precipitation',
       'Rate unemployed', 'Native Hawaiian and Other Pacific Islander',
       'Black or African American', 'ResCust', 'Longitude', 'IndCust',
       'AgrCust', 'Male', 'Latitudes', 'Asian', 'White', 'Wind speed',
       'Maximum humidity', 'Solar radiation', 'Maximum temperature',
       'Total_DG', 'Minimum humidity', 'Minimum temperature',
       'Two or more races', 'American Indian and Alaska Native',
       'Labor force', 'ComCust', 'OthCust', 'Existing_DG',
       'Average weekly wages'], dtype=object)
```

[41]:
```
# this array shows the most highly correlated to least highly correlated␣
↪features for Ridge
sorted_features_space_Ridge = features_space[(-abs(coef_space_Ridge)).argsort()]
sorted_features_space_Ridge
# similar to Lasso
```

[41]:
```
array(['Total housing units', 'Total population', 'Precipitation',
       'Rate unemployed', 'Native Hawaiian and Other Pacific Islander',
       'Black or African American', 'ResCust', 'Longitude', 'IndCust',
       'AgrCust', 'Male', 'Latitudes', 'Asian', 'White', 'Wind speed',
       'Maximum humidity', 'Solar radiation', 'Maximum temperature',
       'Total_DG', 'Minimum humidity', 'Minimum temperature',
       'Two or more races', 'American Indian and Alaska Native',
       'Labor force', 'ComCust', 'OthCust', 'Existing_DG',
       'Average weekly wages'], dtype=object)
```

[43]:
```
visualize_coef(coef_space)
# the feature magnitudes are similar across all the linear regression methods.
# Regularization doesn't seem to provide a benefit.
```



1.7.2 Temporal Prediction (all times in 2019 for Berkeley T Substation)

This prediction problem aims to predict overload temporally for any hour of a month. To that end, a particular substation (Berkeley T) was chosen and only the data that corresponded to this substation was retained from the full dataset (`full_dataset`). We also tried predictions for a whole county, but predictions were much worse. All spatial columns were dropped from the feature matrix. The temporal granularity is hour of a month.

Similar to the spatial prediction problem, we used OLS, Ridge, Lasso and KNN. The same number of cross-validation folds and methods were used. We also tried feature engineering the same way on this dataset. Again, we observe similar results between the linear regression models, though these results are much better than the spatial prediction, with R^2 values of about 0.94.

[65]:
```
temporal_dataset = cleaned_dataset[(cleaned_dataset['Substation'] == 'BERKELEY␣
↪T') &
                                   (cleaned_dataset['LoadCapacity_kW'] != 0)].
↪drop(
                                   columns =['Substation', 'County',␣
↪'Latitudes', 'Longitude'])
temporal_dataset
```

[65]:

	Month	Hour	LoadCapacity_kW	overload	ResCust	ComCust	IndCust	\
9	1	1	1580	1390	415	38	8	
28	1	1	1200	30	2085	133	36	
29	1	1	1390	1070	568	17	2	
163	1	2	1580	1410	415	38	8	
183	1	2	1200	140	2085	133	36	
…	…	…	…	…	…	…	…	
42499	12	22	1580	1330	415	38	8	
42500	12	22	1200	-310	2085	133	36	
42635	12	23	1390	990	568	17	2	
42653	12	23	1580	1340	415	38	8	

```
42654     12    23             1200      -240     2085      133       36

       AgrCust  OthCust  Existing_DG  Queued_DG  Total_DG  Total population  \
9            0        4           10          0        10           1643700
28           0        7          230         10       240           1643700
29           0        2          130         20       150           1643700
163          0        4           10          0        10           1643700
183          0        7          230         10       240           1643700
…          …        …            …          …        …                 …
42499        0        4           10          0        10           1643700
42500        0        7          230         10       240           1643700
42635        0        2          130         20       150           1643700
42653        0        4           10          0        10           1643700
42654        0        7          230         10       240           1643700

         White  Black or African American  American Indian and Alaska Native  \
9      0.41475                   0.107766                            0.006517
28     0.41475                   0.107766                            0.006517
29     0.41475                   0.107766                            0.006517
163    0.41475                   0.107766                            0.006517
183    0.41475                   0.107766                            0.006517
…        …                        …                                  …
42499  0.41475                   0.107766                            0.006517
42500  0.41475                   0.107766                            0.006517
42635  0.41475                   0.107766                            0.006517
42653  0.41475                   0.107766                            0.006517
42654  0.41475                   0.107766                            0.006517

          Asian  Native Hawaiian and Other Pacific Islander  Some other race  \
9      0.295938                                    0.008376         0.103286
28     0.295938                                    0.008376         0.103286
29     0.295938                                    0.008376         0.103286
163    0.295938                                    0.008376         0.103286
183    0.295938                                    0.008376         0.103286
…        …                                          …               …
42499  0.295938                                    0.008376         0.103286
42500  0.295938                                    0.008376         0.103286
42635  0.295938                                    0.008376         0.103286
42653  0.295938                                    0.008376         0.103286
42654  0.295938                                    0.008376         0.103286

       Two or more races  Total housing units     Male   Female  \
9               0.063366               601942  0.49107  0.50893
28              0.063366               601942  0.49107  0.50893
29              0.063366               601942  0.49107  0.50893
163             0.063366               601942  0.49107  0.50893
183             0.063366               601942  0.49107  0.50893
```

```
…                  …                    …       …       …
42499           0.063366                601942  0.49107  0.50893
42500           0.063366                601942  0.49107  0.50893
42635           0.063366                601942  0.49107  0.50893
42653           0.063366                601942  0.49107  0.50893
42654           0.063366                601942  0.49107  0.50893

       Minimum temperature  Maximum temperature  Minimum humidity  \
9               277.434532           287.580572         56.680570
28              277.434532           287.580572         56.680570
29              277.434532           287.580572         56.680570
163             277.434532           287.580572         56.680570
183             277.434532           287.580572         56.680570
…                      …                    …                 …
42499           273.981774           285.492741         42.818402
42500           273.981774           285.492741         42.818402
42635           273.981774           285.492741         42.818402
42653           273.981774           285.492741         42.818402
42654           273.981774           285.492741         42.818402

       Maximum humidity  Wind speed  Solar radiation  Precipitation  \
9              91.951602    3.883869       146.940284       0.000027
28             91.951602    3.883869       146.940284       0.000027
29             91.951602    3.883869       146.940284       0.000027
163            91.951602    3.883869       146.940284       0.000027
183            91.951602    3.883869       146.940284       0.000027
…                    …           …                …              …
42499          86.456729    4.230522       142.863134       0.000024
42500          86.456729    4.230522       142.863134       0.000024
42635          86.456729    4.230522       142.863134       0.000024
42653          86.456729    4.230522       142.863134       0.000024
42654          86.456729    4.230522       142.863134       0.000024

       Labor force  Rate unemployed  Average weekly wages
9         0.521446            0.034                  1555
28        0.521446            0.034                  1555
29        0.521446            0.034                  1555
163       0.521446            0.034                  1555
183       0.521446            0.034                  1555
…              …                …                     …
42499     0.514206            0.025                  1575
42500     0.514206            0.025                  1575
42635     0.514206            0.025                  1575
42653     0.514206            0.025                  1575
42654     0.514206            0.025                  1575

[828 rows x 33 columns]
```

[66]:
```
# Y_time = temporal_dataset['LoadCapacity_kW']
Y_time = temporal_dataset['overload']
X_time = temporal_dataset.drop(columns = ['LoadCapacity_kW', 'overload'])
X_time = X_time.drop(columns = ['Female', 'Some other race', 'Queued_DG'])
```

[67]:
```
X_time_train, X_time_test, y_time_train, y_time_test =␣
↪get_X_y_stnd(X_time,Y_time,0.25, 2020)
```

[68]:
```
mse_time_LR, coef_time_LR, r2_time_LR, resid_time_LR, mse_train_time_LR =␣
↪fit_model(LinearRegression, X_time_train, X_time_test, y_time_train,␣
↪y_time_test, 5, 2020, np.linspace(0.001,100, 100))
mse_time_Lasso, coef_time_Lasso, r2_time_Lasso, resid_time_Lasso,␣
↪mse_train_time_Lasso = fit_model(LassoCV, X_time_train, X_time_test,␣
↪y_time_train, y_time_test, 5, 2020, np.linspace(0.001,100, 100))
mse_time_Ridge, coef_time_Ridge, r2_time_Ridge, resid_time_Ridge,␣
↪mse_train_time_Ridge = fit_model(RidgeCV, X_time_train, X_time_test,␣
↪y_time_train, y_time_test, 5, 2020, np.linspace(0.001,100, 100))
```

```
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2397095.4254381824, tolerance: 16709.833548387098
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2359200.569684295, tolerance: 16976.847283702213
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2229978.750564459, tolerance: 16498.259839034206
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2418980.0139041515, tolerance: 16726.98245472837
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:525: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2479951.3934356035, tolerance: 16554.45605633803
  model = cd_fast.enet_coordinate_descent_gram(
/Users/yashchainani96/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2995810.87868464, tolerance: 20873.134621578098
```

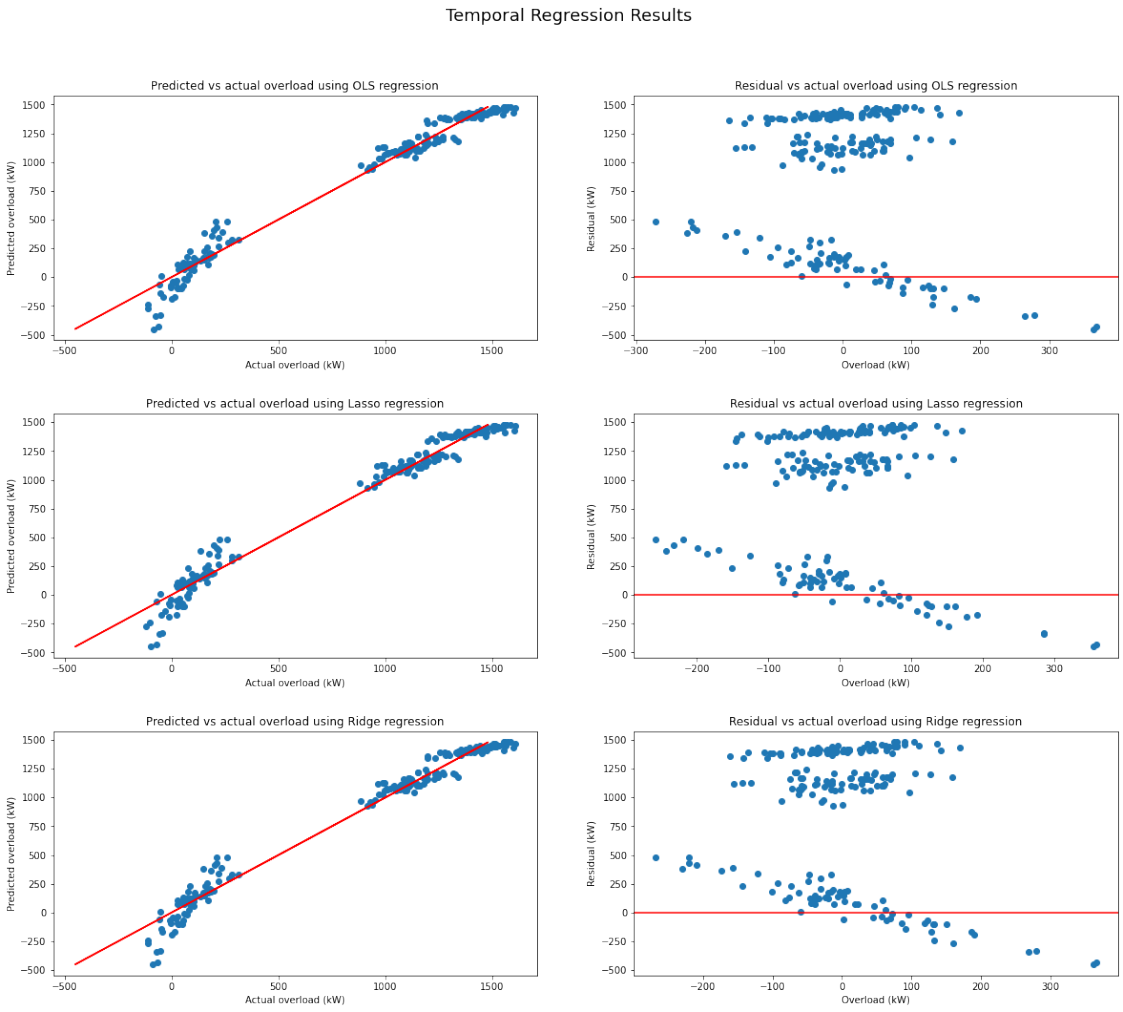
```
  model = cd_fast.enet_coordinate_descent(
```

```
optimal alpha (Lasso): 0.001
optimal alpha (Ridge): 0.001
```

[69]:
```
mse_time = np.array([mse_time_LR, mse_time_Lasso, mse_time_Ridge])
coef_time = np.array([coef_time_LR, coef_time_Lasso, coef_time_Ridge])
r2_time = np.array([r2_time_LR, r2_time_Lasso, r2_time_Ridge])
resid_time = np.array([resid_time_LR, resid_time_Lasso, resid_time_Ridge])
mse_train_time = np.array([mse_train_time_LR, mse_train_time_Lasso,␣
 ↪mse_train_time_Ridge])

r2_time # this is really good for Berkeley T
```

[69]:
```
array([0.97646696, 0.97607555, 0.97644024])
```

[70]:
```
#plots the residuals for the temporal data
plotresiduals(resid_time_LR, resid_time_Lasso, resid_time_Ridge, y_time_test, 1)

# Residual plots show almost all positive residuals, which is indicative of␣
 ↪bias (though probably less than the spatial prediction)
# We can't tell a particular pattern (like spatial which was strongly linear)
```



[71]:
```
features_time = np.array(X_time.columns)
sorted_features_time_LR = features_time[(-abs(coef_time_LR)).argsort()]
sorted_features_time_LR
# this array shows the most important to least important features.
```

[71]:
```
array(['Maximum temperature', 'Minimum temperature', 'Minimum humidity',
       'Maximum humidity', 'Solar radiation', 'Rate unemployed',
       'Wind speed', 'Total_DG', 'Existing_DG', 'ResCust', 'Labor force',
       'IndCust', 'ComCust', 'Precipitation', 'Hour', 'OthCust',
       'Average weekly wages', 'Month', 'Total population', 'Asian',
       'White', 'Black or African American', 'AgrCust',
       'Total housing units', 'Two or more races',
       'Native Hawaiian and Other Pacific Islander',
       'American Indian and Alaska Native', 'Male'], dtype=object)
```

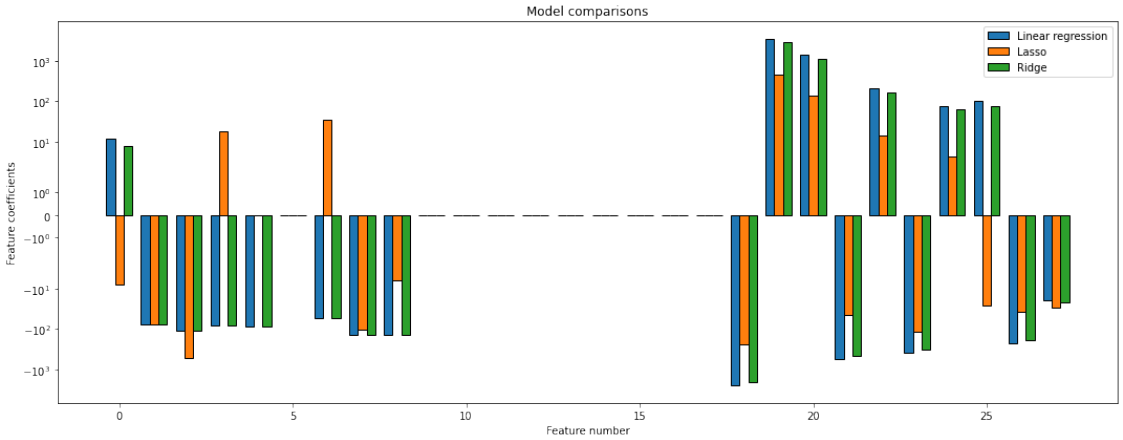
[72]:
```
sorted_features_time_Lasso = features_time[(-abs(coef_time_Lasso)).argsort()]
sorted_features_time_Lasso
# a little different from OLS
```

[72]:
```
array(['ResCust', 'Maximum temperature', 'Minimum temperature',
       'Minimum humidity', 'Solar radiation', 'Existing_DG', 'Hour',
       'Maximum humidity', 'Rate unemployed', 'OthCust',
       'Average weekly wages', 'Labor force', 'ComCust', 'Wind speed',
       'Month', 'Total_DG', 'Precipitation', 'Two or more races',
       'Total housing units', 'Male', 'American Indian and Alaska Native',
       'Black or African American', 'White', 'Total population',
       'AgrCust', 'IndCust', 'Native Hawaiian and Other Pacific Islander',
       'Asian'], dtype=object)
```

[73]:
```
sorted_features_time_Ridge = features_time[(-abs(coef_time_Ridge)).argsort()]
sorted_features_time_Ridge
# mostly similar to OLS
```

[73]:
```
array(['Maximum temperature', 'Minimum temperature', 'Minimum humidity',
       'Maximum humidity', 'Solar radiation', 'Rate unemployed',
       'Wind speed', 'Total_DG', 'Existing_DG', 'ResCust', 'IndCust',
       'ComCust', 'Labor force', 'Hour', 'Precipitation', 'OthCust',
       'Average weekly wages', 'Month', 'Total housing units',
       'Two or more races', 'Native Hawaiian and Other Pacific Islander',
       'American Indian and Alaska Native', 'Black or African American',
       'White', 'Total population', 'AgrCust', 'Male', 'Asian'],
      dtype=object)
```

[75]:
```
visualize_coef(coef_time)
# OLS provides slightly bigger feature coefficients than Ridge in general and␣
 ↪Lasso is smaller still, sometimes in the opposite direction.
# This makes sense since regularization penalizes large coefficients to prevent␣
 ↪overfitting.
```



1.7.3 Predicting overload in the second half of 2019 using features from first half of 2019 (time lag) for Berkeley T substation

The weakness of the previous prediction model is that it is trained only on past data, and so we can produce reliable predictions for the time frame the data is trained on, but not for the future. To address this, we train a model with a time lag. We wanted to assess how well we can use data from the first half of the year to predict the latter half of the year. Features from the first half of the month were matched with response variables from the second half of the year. For example noon in February was matched with noon in August (6 months later). It might produce better predictions if we did the time lag data by seasons (aka match Jan-Mar of 2018 features to Jan-Mar 2019 response variables) or month to month (Jan of 2018 to Jan of 2019), but we did not have access to the data necessary to test this out. This would allow us to predict the overload response by season as the climate and temperature changes vary in similar ways with respect to this.

Again, we used OLS, Ridge, Lasso and KNN. The same number of cross-validation folds and methods were used. We also tried feature engineering the same way on this dataset. Again, we observe similar results between the linear regression models. These results were much more similar to the temporal prediction above (without a time lag) than the spatial prediction.

[76]:
```
lag_dataset = full_dataset[(full_dataset['Substation'] == 'BERKELEY T') &
                           (full_dataset['LoadCapacity_kW'] != 0)].drop(
                           columns =['Substation', 'County', 'Latitudes',␣
↪'Longitude',
                                     'SUBSTATIONID', 'geometry',␣
↪'Feeder_Name', 'High'])
lag_half_one = lag_dataset[lag_dataset['Month']<7]
lag_half_two = lag_dataset[lag_dataset['Month']>6]

# map the second half to the first half of the year
lag_half_two['Month'] = lag_half_two['Month'] - 6
```

```
<ipython-input-76-c55e52a0fbf9>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  lag_half_two['Month'] = lag_half_two['Month'] - 6
```

[77]:
```
lag_half_two
```

[77]:

	FeederID	Month	Hour	LoadCapacity_kW	overload	ResCust	ComCust	\
21416	12660401	1	1	1580	1450	415	38	
21423	12660404	1	1	1390	1170	568	17	
21424	12660402	1	1	1200	320	2085	133	
21572	12660401	1	2	1580	1460	415	38	

```
21579  12660404      1     2             1390      1190      568       17
…         …    …   …              …        …       …       …
42499  12660401      6    22             1580      1330      415       38
42500  12660402      6    22             1200      -310     2085      133
42635  12660404      6    23             1390       990      568       17
42653  12660401      6    23             1580      1340      415       38
42654  12660402      6    23             1200      -240     2085      133

       IndCust  AgrCust  OthCust  Existing_DG  Queued_DG  Total_DG  \
21416        8        0        4           10          0        10
21423        2        0        2          130         20       150
21424       36        0        7          230         10       240
21572        8        0        4           10          0        10
21579        2        0        2          130         20       150
…         …       …       …           …         …       …
42499        8        0        4           10          0        10
42500       36        0        7          230         10       240
42635        2        0        2          130         20       150
42653        8        0        4           10          0        10
42654       36        0        7          230         10       240

       Total population    White  Black or African American  \
21416           1643700  0.41475                    0.107766
21423           1643700  0.41475                    0.107766
21424           1643700  0.41475                    0.107766
21572           1643700  0.41475                    0.107766
21579           1643700  0.41475                    0.107766
…                 …      …                         …
42499           1643700  0.41475                    0.107766
42500           1643700  0.41475                    0.107766
42635           1643700  0.41475                    0.107766
42653           1643700  0.41475                    0.107766
42654           1643700  0.41475                    0.107766

       American Indian and Alaska Native     Asian  \
21416                           0.006517  0.295938
21423                           0.006517  0.295938
21424                           0.006517  0.295938
21572                           0.006517  0.295938
21579                           0.006517  0.295938
…                                  …       …
42499                           0.006517  0.295938
42500                           0.006517  0.295938
42635                           0.006517  0.295938
42653                           0.006517  0.295938
42654                           0.006517  0.295938
```

	Native Hawaiian and Other Pacific Islander	Some other race	\
21416	0.008376	0.103286	
21423	0.008376	0.103286	
21424	0.008376	0.103286	
21572	0.008376	0.103286	
21579	0.008376	0.103286	
…	…	…	
42499	0.008376	0.103286	
42500	0.008376	0.103286	
42635	0.008376	0.103286	
42653	0.008376	0.103286	
42654	0.008376	0.103286	

	Two or more races	Total housing units	Male	Female	\
21416	0.063366	601942	0.49107	0.50893	
21423	0.063366	601942	0.49107	0.50893	
21424	0.063366	601942	0.49107	0.50893	
21572	0.063366	601942	0.49107	0.50893	
21579	0.063366	601942	0.49107	0.50893	
…	…	…	…	…	
42499	0.063366	601942	0.49107	0.50893	
42500	0.063366	601942	0.49107	0.50893	
42635	0.063366	601942	0.49107	0.50893	
42653	0.063366	601942	0.49107	0.50893	
42654	0.063366	601942	0.49107	0.50893	

	Minimum temperature	Maximum temperature	Minimum humidity	\
21416	288.296456	306.482643	24.983206	
21423	288.296456	306.482643	24.983206	
21424	288.296456	306.482643	24.983206	
21572	288.296456	306.482643	24.983206	
21579	288.296456	306.482643	24.983206	
…	…	…	…	
42499	273.981774	285.492741	42.818402	
42500	273.981774	285.492741	42.818402	
42635	273.981774	285.492741	42.818402	
42653	273.981774	285.492741	42.818402	
42654	273.981774	285.492741	42.818402	

	Maximum humidity	Wind speed	Solar radiation	Precipitation	\
21416	66.162095	4.331382	447.071037	6.911915e-08	
21423	66.162095	4.331382	447.071037	6.911915e-08	
21424	66.162095	4.331382	447.071037	6.911915e-08	
21572	66.162095	4.331382	447.071037	6.911915e-08	
21579	66.162095	4.331382	447.071037	6.911915e-08	
…	…	…	…	…	
42499	86.456729	4.230522	142.863134	2.394550e-05	

```
42500          86.456729    4.230522       142.863134   2.394550e-05
42635          86.456729    4.230522       142.863134   2.394550e-05
42653          86.456729    4.230522       142.863134   2.394550e-05
42654          86.456729    4.230522       142.863134   2.394550e-05

       Labor force  Rate unemployed  Average weekly wages
21416     0.520655            0.033                  1493
21423     0.520655            0.033                  1493
21424     0.520655            0.033                  1493
21572     0.520655            0.033                  1493
21579     0.520655            0.033                  1493
…              …                …                     …
42499     0.514206            0.025                  1575
42500     0.514206            0.025                  1575
42635     0.514206            0.025                  1575
42653     0.514206            0.025                  1575
42654     0.514206            0.025                  1575

[414 rows x 34 columns]
```

[78]:
```
lag_dataset = full_dataset[(full_dataset['Substation'] == 'BERKELEY T') &
                           (full_dataset['LoadCapacity_kW'] != 0)].drop(
                           columns =['Substation', 'County', 'Latitudes',␣
 ↪'Longitude',
                                     'SUBSTATIONID', 'geometry',␣
 ↪'Feeder_Name', 'High'])
lag_half_one = lag_dataset[lag_dataset['Month']<7]
lag_half_two = lag_dataset[lag_dataset['Month']>6]

# map the second half to the first half of the year
lag_half_two['Month'] = lag_half_two['Month'] - 6

# we have to match based on substation
lag_dataset.head()
lag_merge = lag_half_one.merge(lag_half_two, on = ['FeederID', 'Hour',␣
 ↪'Month']).drop(
            columns = ['LoadCapacity_kW_x','overload_x', 'ResCust_y',␣
 ↪'ComCust_y', 'IndCust_y', 'AgrCust_y',
                       'OthCust_y','Existing_DG_y', 'Queued_DG_y', 'Total_DG_y',␣
 ↪'Total population_y',
                       'White_y', 'Black or African American_y', 'American␣
 ↪Indian and Alaska Native_y',
                       'Asian_y','Native Hawaiian and Other Pacific Islander_y',␣
 ↪'Some other race_y',
                       'Two or more races_y', 'Total housing units_y', 'Male_y',␣
 ↪'Female_y','Minimum temperature_y',
```

```
                      'Maximum temperature_y', 'Minimum humidity_y','Maximum␣
 ↪humidity_y', 'Wind speed_y',
                      'Solar radiation_y','Precipitation_y', 'Labor force_y',␣
 ↪'Rate unemployed_y',
                      'Average weekly wages_y'])

Y_lag = lag_merge['overload_y']
X_lag = lag_merge.drop(columns = ['FeederID', 'LoadCapacity_kW_y','overload_y'])
X_lag = X_lag.drop(columns = ['Female_x', 'Some other race_x', 'Queued_DG_x'])

X_lag.columns = X_lag.columns.str.replace('_x','')
```

```
<ipython-input-78-9732d89ec717>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  lag_half_two['Month'] = lag_half_two['Month'] - 6
```

[79]:
```
X_lag_train, X_lag_test, y_lag_train, y_lag_test = get_X_y_stnd(X_lag,Y_lag,0.
 ↪25, 2020)
```

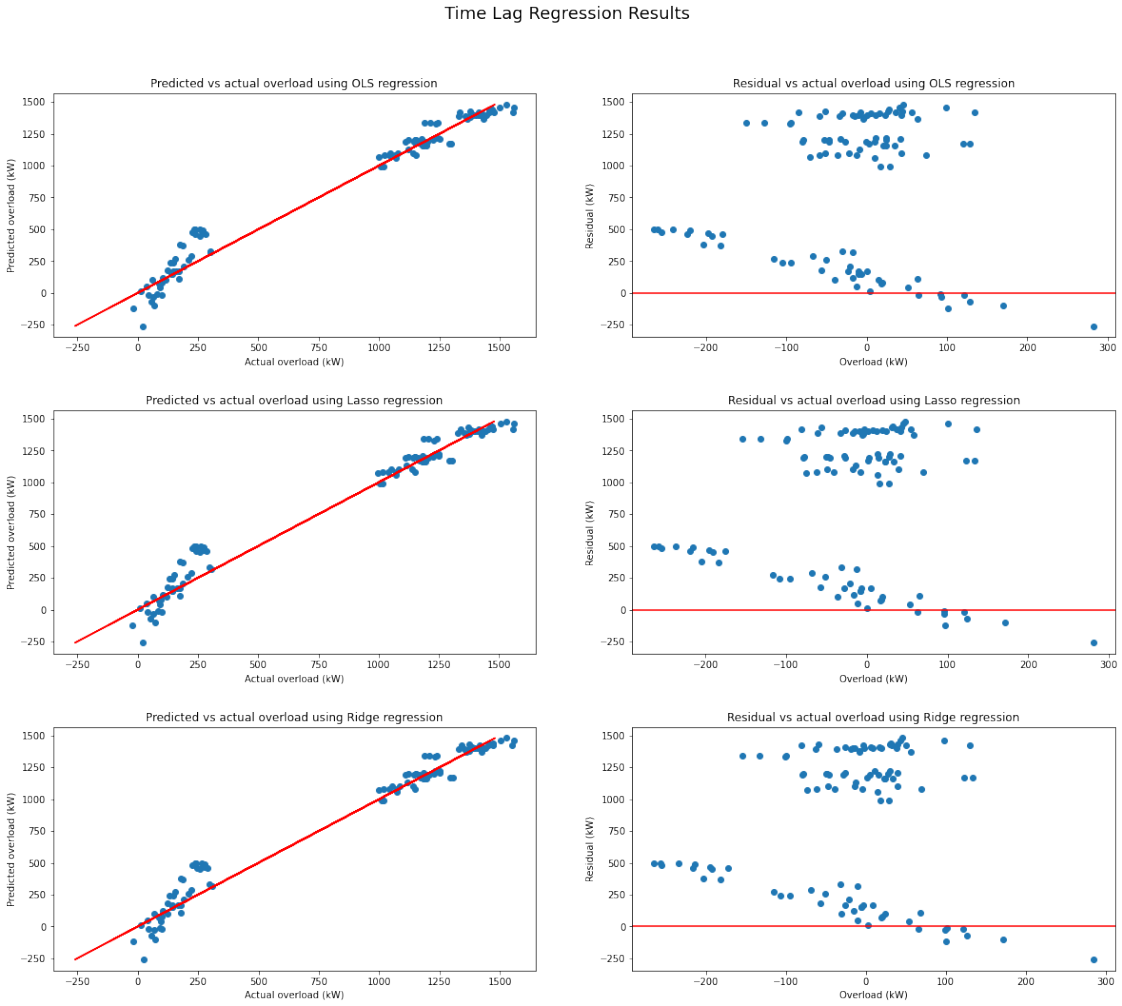
[80]:
```
mse_lag_LR, coef_lag_LR, r2_lag_LR, resid_lag_LR, mse_train_lag_LR =␣
 ↪fit_model(LinearRegression, X_lag_train, X_lag_test, y_lag_train,␣
 ↪y_lag_test, 5, 2020, np.linspace(0.01,100, 100))
mse_lag_Lasso, coef_lag_Lasso, r2_lag_Lasso, resid_lag_Lasso,␣
 ↪mse_train_lag_Lasso = fit_model(LassoCV, X_lag_train, X_lag_test,␣
 ↪y_lag_train, y_lag_test, 5, 2020, np.linspace(0.01,100, 100))
mse_lag_Ridge, coef_lag_Ridge, r2_lag_Ridge, resid_lag_Ridge,␣
 ↪mse_train_lag_Ridge = fit_model(RidgeCV, X_lag_train, X_lag_test,␣
 ↪y_lag_train, y_lag_test, 5, 2020, np.linspace(0.01,100, 100))
```

```
optimal alpha (Lasso): 0.01
optimal alpha (Ridge): 4.05
```

[81]:
```
mse_lag = np.array([mse_lag_LR, mse_lag_Lasso, mse_lag_Ridge])
coef_lag = np.array([coef_lag_LR, coef_lag_Lasso, coef_lag_Ridge])
r2_lag = np.array([r2_lag_LR, r2_lag_Lasso, r2_lag_Ridge])
resid_lag =  np.array([resid_lag_LR, resid_lag_Lasso, resid_lag_Ridge])
mse_train_lag = np.array([mse_train_lag_LR, mse_train_lag_Lasso,␣
 ↪mse_train_lag_Ridge])

#mse_lag
r2_lag # this is really good for Berkeley T
```

[81]:
```
array([0.96960167, 0.96971739, 0.96986533])
```

[82]:
```
#plots residuals for time-lag model
plotresiduals(resid_lag_LR,resid_lag_Lasso, resid_lag_Ridge,y_lag_test, 2)

# Residual plots show almost all positive residuals, which is indicative of␣
 ↪bias (though probably less than the spatial prediction).
# These results are similar to the temporal results (they were both done on the␣
 ↪same location)
```



[83]:
```
features_lag = np.array(X_lag.columns)
sorted_features_lag_LR = features_lag[(-abs(coef_lag_LR)).argsort()]
sorted_features_lag_LR
# this array shows the most important to least important features.
```

[83]:
```
array(['Labor force', 'Minimum humidity', 'Rate unemployed',
       'Maximum temperature', 'Solar radiation', 'Precipitation',
       'Wind speed', 'Maximum humidity', 'Average weekly wages', 'Month',
```

```
       'Minimum temperature', 'Existing_DG', 'IndCust', 'Total_DG',
       'OthCust', 'AgrCust', 'ResCust', 'ComCust', 'Hour', 'White',
       'Male', 'Total housing units', 'Two or more races',
       'Native Hawaiian and Other Pacific Islander',
       'American Indian and Alaska Native', 'Black or African American',
       'Total population', 'Asian'], dtype=object)
```

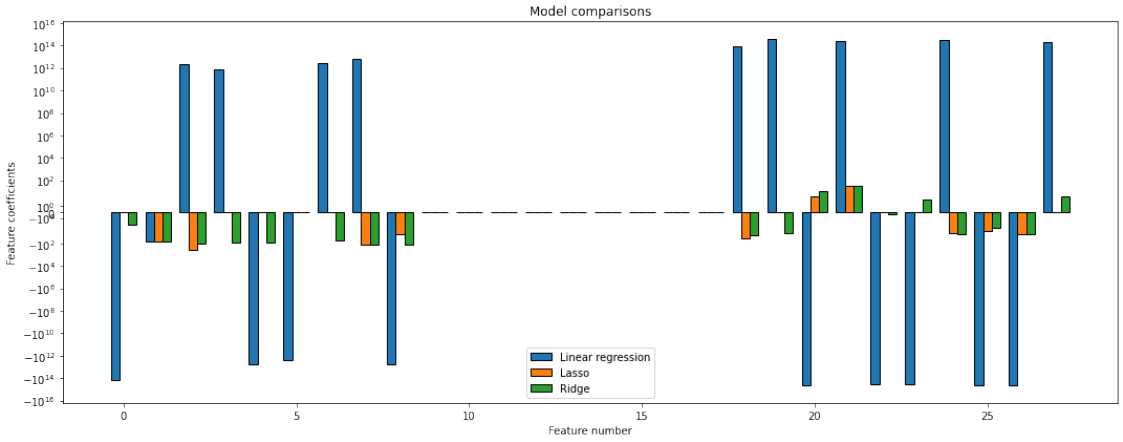
[84]:
```
sorted_features_lag_Lasso = features_lag[(-abs(coef_lag_Lasso)).argsort()]
sorted_features_lag_Lasso
# this array shows the most important to least important features.
```

[84]:
```
array(['ResCust', 'Existing_DG', 'Hour', 'Minimum temperature',
       'Maximum humidity', 'Total_DG', 'Rate unemployed', 'Precipitation',
       'Labor force', 'Minimum humidity', 'Month', 'Solar radiation',
       'Wind speed', 'Maximum temperature', 'Male', 'Total housing units',
       'Asian', 'Native Hawaiian and Other Pacific Islander',
       'American Indian and Alaska Native', 'Black or African American',
       'White', 'Total population', 'OthCust', 'AgrCust', 'IndCust',
       'ComCust', 'Two or more races', 'Average weekly wages'],
      dtype=object)
```

[85]:
```
sorted_features_lag_Ridge = features_lag[(-abs(coef_lag_Ridge)).argsort()]
sorted_features_lag_Ridge
# this array shows the most important to least important features.
```

[85]:
```
array(['Total_DG', 'Existing_DG', 'ResCust', 'IndCust', 'ComCust', 'Hour',
       'OthCust', 'Maximum humidity', 'Minimum temperature',
       'Precipitation', 'Rate unemployed', 'Maximum temperature',
       'Minimum humidity', 'Average weekly wages', 'Labor force', 'Month',
       'Solar radiation', 'Wind speed', 'Total housing units', 'Male',
       'Native Hawaiian and Other Pacific Islander',
       'American Indian and Alaska Native', 'Black or African American',
       'White', 'Total population', 'AgrCust', 'Two or more races',
       'Asian'], dtype=object)
```

[86]:
```
visualize_coef(coef_lag)
# OLS provides bigger feature coefficients than Lasso and Ridge. This makes␣
 ↪sense since regularization penalizes large
# coefficients to prevent overfitting. Some features have 0 coefficients for␣
 ↪Lasso - this is due to its feature selection.
# This effect is more pronounced than in the previous prediction problem.
```



Additional comments or suggestions: Note: Just by looking at coefficients, we can see whether or not some features are positively or negatively correlated with specific features, but we cannot determine the causal impacts on our actual prediction directly. For example, if the total population is very highly negatively correlated, this may result because of some collinearity of total population with some of the features or could be because some substations were made to be even more robust since they need to provide for more people.
Additionally, for the spatial predictions, total housing units was the most highly correlated to overload. For the temporal prediction, max and min humidity seemed to have a greater correlation to the overload than other variables. For the time-lag prediction, the most highly correlated feature changed with method used.

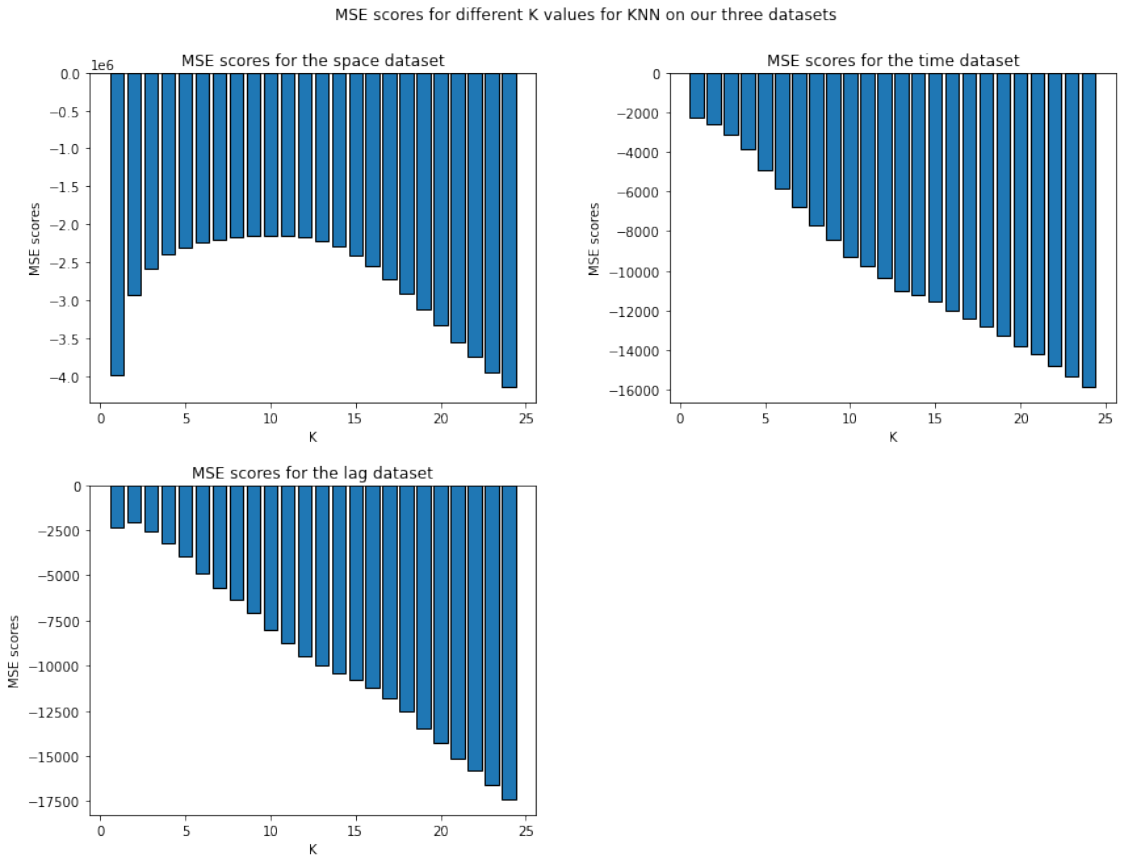
1.7.4 Non-Parametric Method

In addressing the prediction problems above, we realized that a linear model may not be the best model to fit the data, especially for the spatial dataset. Here we explore a non-parametric KNN model instead for the spatial data.

[87]:
```
# X_space_KNN = X_space.drop(columns = [col for col in X_space.columns if␣
↪"squared" in col])
# X_space_train_KNN, X_space_test_KNN, y_space_train_KNN, y_space_test_KNN =␣
↪get_X_y_stnd(X_space_KNN,Y_space,0.25,2020)

# X_time_KNN = X_time.drop(columns = [col for col in X_time.columns if␣
↪"squared" in col])
# X_time_train_KNN, X_time_test_KNN, y_time_train_KNN, y_time_test_KNN =␣
↪get_X_y_stnd(X_time_KNN,Y_time,0.25,2020)

# X_lag_KNN = X_lag.drop(columns = [col for col in X_lag.columns if "squared"␣
↪in col])
# X_lag_train_KNN, X_lag_test_KNN, y_lag_train_KNN, y_lag_test_KNN =␣
↪get_X_y_stnd(X_lag_KNN,Y_lag,0.25,2020)
```

[88]:
```
K_values = list(range(1,25,1))
# K_space, K_scores_space = choose_optimal_K(X_space_train_KNN,␣
 ↪y_space_train_KNN, K_values)
# K_time, K_scores_time = choose_optimal_K(X_time_train_KNN, y_time_train_KNN,␣
 ↪K_values)
# K_lag, K_scores_lag = choose_optimal_K(X_lag_train_KNN, y_lag_train_KNN,␣
 ↪K_values)

K_space, K_scores_space = choose_optimal_K(X_space_train, y_space_train,␣
 ↪K_values)
K_time, K_scores_time = choose_optimal_K(X_time_train, y_time_train, K_values)
K_lag, K_scores_lag = choose_optimal_K(X_lag_train, y_lag_train, K_values)
```

[89]:
```
K_scores_df = pd.DataFrame({"K":K_values,
                            "MSE_space":K_scores_space,
                            "MSE_time":K_scores_time,
                            "MSE_lag":K_scores_lag})

fig = plt.figure(figsize = (14,10))
fig.suptitle("MSE scores for different K values for KNN on our three datasets")

plt.subplot(221)
plt.bar(K_scores_df["K"], K_scores_df["MSE_space"])
plt.title("MSE scores for the space dataset")
plt.ylabel("MSE scores")
plt.xlabel("K")

plt.subplot(222)
plt.bar(K_scores_df["K"], K_scores_df["MSE_time"])
plt.title("MSE scores for the time dataset")
plt.ylabel("MSE scores")
plt.xlabel("K")

plt.subplot(223)
plt.bar(K_scores_df["K"], K_scores_df["MSE_lag"])
plt.title("MSE scores for the lag dataset")
plt.ylabel("MSE scores")
plt.xlabel("K")

plt.subplots_adjust(top=0.91, wspace=0.3, hspace=0.25)
```



[92]:
```
mse_space_KNN, coef_space_KNN, r2_space_KNN, resid_space_KNN,␣
 ↪mse_train_space_KNN = fit_model(KNeighborsRegressor, X_space_train,␣
 ↪X_space_test, y_space_train, y_space_test,␣
 ↪n_splits=5,random_state=2020,alphas_array=[], K=K_space)
mse_time_KNN, coef_time_KNN, r2_time_KNN, resid_time_KNN, mse_train_time_KNN =␣
 ↪fit_model(KNeighborsRegressor, X_time_train, X_time_test, y_time_train,␣
 ↪y_time_test,  n_splits=5,random_state=2020,alphas_array=[], K=K_time)
mse_lag_KNN, coef_lag_KNN, r2_lag_KNN, resid_lag_KNN, mse_train_lag_KNN =␣
 ↪fit_model(KNeighborsRegressor, X_lag_train, X_lag_test, y_lag_train,␣
 ↪y_lag_test, n_splits=5,random_state=2020, alphas_array=[], K=K_lag)
```

[93]:
```
print("For the following MSE arrays, the four printed values result from the␣
 ↪OLS, Lasso, Ridge, and KNN models respectively.")
print()

mse_space_w_KNN = np.around(np.array([mse_space_LR, mse_space_Lasso,␣
 ↪mse_space_Ridge, mse_space_KNN]), decimals=2)
r2_space_w_KNN = np.around(np.array([r2_space_LR, r2_space_Lasso,␣
 ↪r2_space_Ridge, r2_space_KNN]), decimals=2)
```

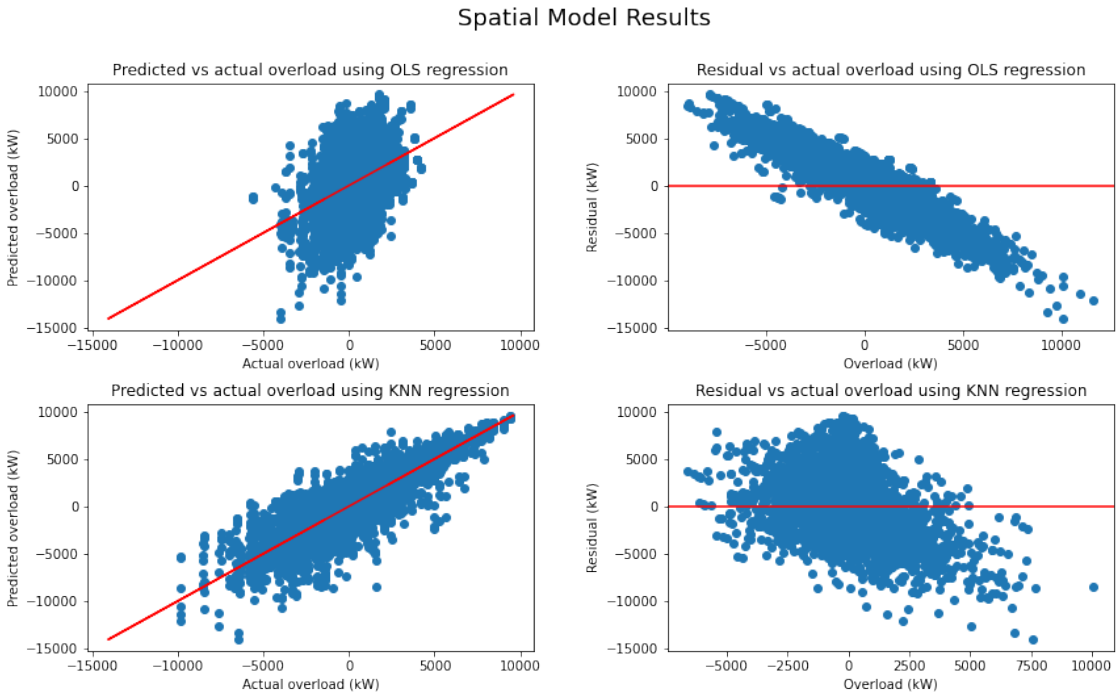
```
resid_space_w_KNN = np.around(np.array([resid_space_LR, resid_space_Lasso,␣
 ↪resid_space_Ridge, resid_space_KNN]), decimals=2)
mse_train_space_w_KNN = np.around(np.array([mse_train_space_LR,␣
 ↪mse_train_space_Lasso, mse_train_space_Ridge, mse_train_space_KNN]),␣
 ↪decimals=2)

mse_time_w_KNN = np.around(np.array([mse_time_LR, mse_time_Lasso,␣
 ↪mse_time_Ridge, mse_time_KNN]), decimals=2)
r2_time_w_KNN = np.around(np.array([r2_time_LR, r2_time_Lasso, r2_time_Ridge,␣
 ↪r2_time_KNN]), decimals=2)
resid_time_w_KNN = np.around(np.array([resid_time_LR, resid_time_Lasso,␣
 ↪resid_time_Ridge, resid_time_KNN]), decimals=2)
mse_train_time_w_KNN = np.around(np.array([mse_train_time_LR,␣
 ↪mse_train_time_Lasso, mse_train_time_Ridge, mse_train_time_KNN]), decimals=2)

mse_lag_w_KNN = np.around(np.array([mse_lag_LR, mse_lag_Lasso, mse_lag_Ridge,␣
 ↪mse_lag_KNN]), decimals=2)
r2_lag_w_KNN = np.around(np.array([r2_lag_LR, r2_lag_Lasso, r2_lag_Ridge,␣
 ↪r2_lag_KNN]), decimals=2)
resid_lag_w_KNN =  np.around(np.array([resid_lag_LR, resid_lag_Lasso,␣
 ↪resid_lag_Ridge, resid_lag_KNN]), decimals=2)
mse_train_lag_w_KNN = np.around(np.array([mse_train_lag_LR,␣
 ↪mse_train_lag_Lasso, mse_train_lag_Ridge, mse_train_lag_KNN]), decimals=2)

print('Test MSE (spatial):', mse_space_w_KNN)
print('Training MSE (spatial):', mse_train_space_w_KNN)

print('Test MSE (temporal):', mse_time_w_KNN)
print('Training MSE (temporal):', mse_train_time_w_KNN)

print('Test MSE (time lag):', mse_lag_w_KNN)
print('Training MSE (time lag):', mse_train_lag_w_KNN)

print()
print('R2 (spatial):', r2_space_w_KNN)
print('R2 (temporal):', r2_time_w_KNN)
print('R2 (time lag):', r2_lag_w_KNN)
```

For the following MSE arrays, the four printed values result from the OLS, Lasso, Ridge, and KNN models respectively.

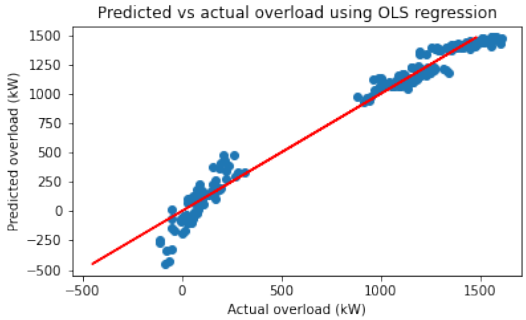
```
Test MSE (spatial): [7815549.42 7815340.12 7815545.57 2790304.65]
Training MSE (spatial): [8082526.99 8082540.37 8082526.99 2536817.76]
Test MSE (temporal): [8239.35 8376.39 8248.71 1202.05]
Training MSE (temporal): [10175.12 10291.76 10180.1    497.26]
Test MSE (time lag): [9321.07 9285.58 9240.22 1301.44]
Training MSE (time lag): [8022.23 8014.77 8020.31  580.16]
```

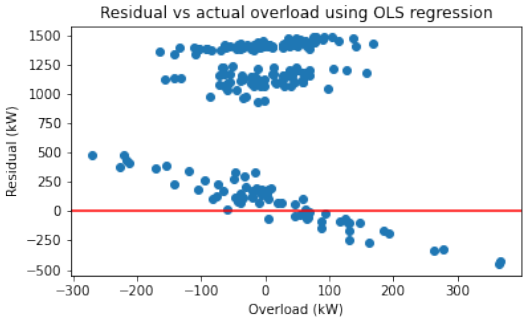
```
R2 (spatial): [0.19 0.19 0.19 0.71]
R2 (temporal): [0.98 0.98 0.98 1.  ]
R2 (time lag): [0.97 0.97 0.97 1.  ]
```

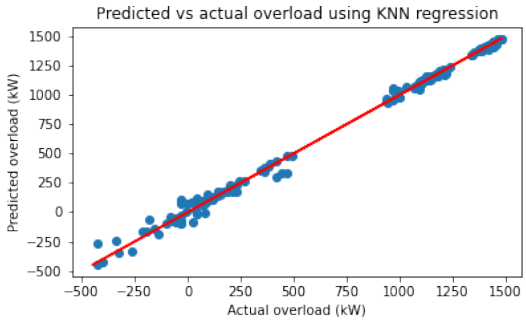
[94]:
```
#plots residual graphs to compare OLS/Ridge/Lasso to KNN
plotresiduals_re(resid_space_LR, resid_space_KNN, y_space_test, 0)
plotresiduals_re(resid_time_LR, resid_time_KNN, y_time_test, 1)
plotresiduals_re(resid_lag_LR, resid_lag_KNN, y_lag_test, 2)
```

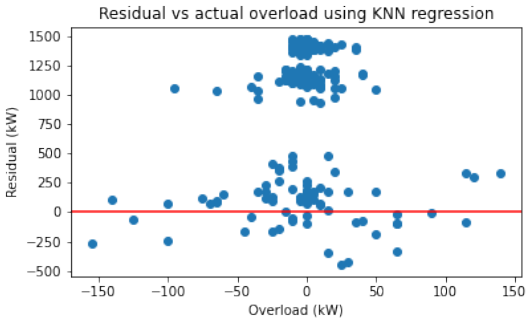


Temporal Model Results

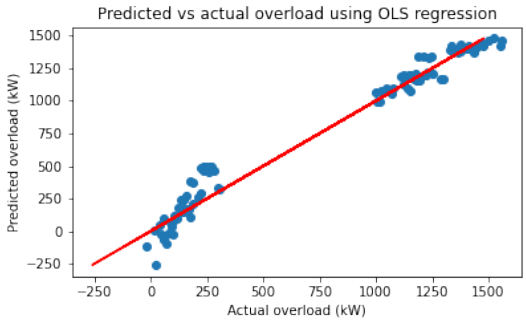


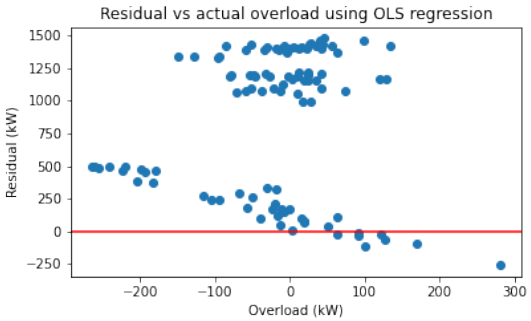


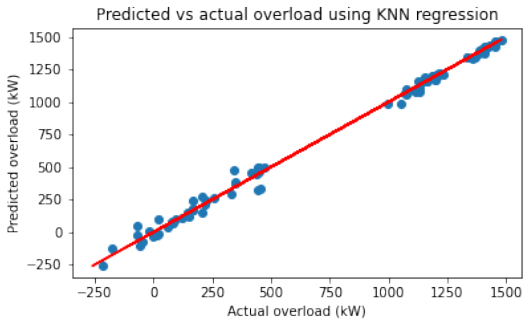


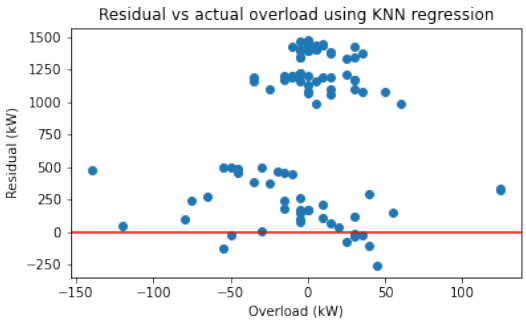


Time Lag Model Results









1.8 Interpretation and Conclusions (20 points)

The overarching goal we had hoped to address with this project was to provide a better idea of where and when electrical substations in California might be overloaded, and by how much. We limited our scope to the counties of Northern California that have their power supplied by PG&E because of data availability.

Even with feature engineering and standardization for our regression techniques, there was still a lot of bias with the parametric regression models. We found that our spatial analysis for June 2019 resulted in a lot of bias for our model as compared to temporal ad time-delay analysis of Berkeley substation T. From plotting residuals graphs, there was a large decrease in the residual as the predicted value increased for spatial predictions. The results were similar for all three of OLS, Lasso and Ridge. The residual plot was a straight line with a negative slope. Below an actual overload of about 0, the predictions are consistently higher than the observations and above 0, the predictions are consistently lower. We also found that the MSEs of all three models were comparable, with OLS actually having a smaller MSE than Lasso and Ridge. As the regularization techniques are meant to correct variance, we suspected the bias was the reason for this as well. We investigated further by inspecting the residuals and engineering features by squaring them to see if we could improve our models but this yielded little improvement in MSE and R^2. We found that between the different counties, the MSEs were extremely large, indicating our models did not perform well spatially most likely due to the unique nature of each region. The MSEs were relatively constant for different months.

For the temporal and time-lag predictions on Berkeley T, the regression models performed much better than the spatial prediction. Unlike the spatial analyses, where the coefficients were unchanged by regularization, Lasso and Ridge resulted in slightly smaller coefficients than OLS for both these predictions in time. This effect is more pronounced for the time-lag prediction, where Lasso successfully performed feature selection - setting some coefficients to 0. It's likely that the model was not overfit but might still be biased, based on the residual plots which seem to have some sort of pattern. The residual graphs were biased to a different extent than in spatial prediction. It is also interesting to note that though the MSE's were smaller than spatial, there was a very wide variation when the temporal and time-lag predictions were performed on different counties.

For all three prediction problems various features were highly correlated. However, a linear model does not appear adequate to fully explain the response variable, overload. These results led us to implement a non-parametric modeling technique with KNN regression which resulted in better models that were much more accurate. The test error for the spatial OLS, LASSO, Ridge, and KNN regression models were all on the order of magnitude of 10^6 with OLS, LASSO, and Ridge models having very similar errors of around 7.8 x 10^6 but the KNN having a more than two fold lower test error of 2 around 2.8 x 10^6. Similar trends can be observed with the temporal and time-lag models wherein the test errors are lower for KNN by a couple fold than for OLS, LASSO, and Ridge, which have similar test errors. The correlation coefficients for KNN are also far higher for spatial, temporal, and time lag predictions than for OLS, LASSO, and Ridge. For spatial predictions for instance, the KNN has a very high correlation coefficient of 0.71 while OLS, LASSO, and Ridge have equally low correlations coefficients of 0.19. From our results, we determined that the best model for making predictions for overload was the KNN method and that this model worked best for the temporal and time-lag predictions. While the KNN model performed better when compared to the regression techniques based on MSE values, using an AIC score in the future will allow us to better compare our model results.

Therefore, we do not advise the state of California to make decisions based on these results, especially the linear models for spatial prediction, in determining how to invest in more renewable power sources in specific regions and institute policies for the amount of power utility companies should plan to purchase in the future.

For our linear regression methods, our temporal and time-lag predictions were reasonable but had high bias whereas the linear regression spatial predictions performed extremely poorly. KNN as a method performed much better overall for all predictions, but the spatial predictions were still much worse than the temporal and time-lag predictions. We recommend further research before employing the model such as using neural nets and decision trees. Decision trees are beneficial as they are more interpretable than KNN and can be used to identify feature importance which will be highly useful for deciding what factors and features impact overload. Neural nets are adaptive models and thus will be able to also better model overload. We believe that more research is necessary to determine additional features that may impact potential overload including extracting details about rooftop solar as well as finding a more granular portfolio of the energy sources that feed to specific substations and feeders. All these steps will help the state better prepare for power shortages and prevent future power outages due to lack of preparation. Assuming a proper model can be developed, the state will be able to then properly reserve the correct amount of power based on use by accounting for the additional power needed for substations that consistently use a specific threshold percentage of the capacity of a substation. Understanding what regions in northern California put a lot of stress on the energy grid and predicting how these change in time are important concepts to understand so that Californians will not be inconvenienced by power shut offs and allow individuals to lead their lives without continuous disruptions to their productivity.

[ ]: