



W205 Project Final Documentation

Solar Growth Drivers and Generation Estimation

Section 5: Vincent Chu, Elizabeth Shulok



Scope, Goals & Methodology

Research question:

How to help utilities focus grid modernization efforts on high growth areas for solar PV and accurately size the impact from reverse power flows?

We focused on the state of California for the above research question. This is because of its number 1 ranking of installed capacity in the US (3,266 MW at the end of in 2015, with about 60% of the entire nation's installed solar PV capacity) and the abundance of public data sources due to state-sponsored solar initiatives. The geographic granularity we investigated on was county level.

Our team built a tool to answer the research question by focusing on the following two aspects:

- **Potential Driver:** Identify high growth counties for solar adoption by exploring the following factors:
 - Average Global Horizontal Irradiance (GHI)¹ by day (kWh/m²/day)
 - Average residential electricity usage per capita
 - Household income
 - Income per capita
 - Number of EVs (Electrical Vehicles)

In the past, financial factors (such as income) that serve as indicator of wealth were main drivers used in understanding and projecting growth of solar adoption. Now, with the insights from our tool, utilities (and solar providers such as SolarCity) can start to rank markets also by other non-financial factors.

- **Solar Generation:** Estimate actual solar generation (in kilowatt - kW) based on solar panel capacity and amount of irradiance.

By aggregating individual interconnection data from CSI (California Solar Initiative) to the county level, the team was able to join installed capacity data with irradiance data to estimate solar generation at an annual average level and also by month.

The raw irradiance data we collect is the amount of Global Horizontal

¹ Global Horizontal Irradiance (GHI) is the total amount of radiation received by the ground. It includes both Direct Normal Irradiance (DNI) and Diffuse Horizontal Irradiance (DHI). Please see the following link for more information:
<https://firstgreenconsulting.wordpress.com/2012/04/26/differentiate-between-the-dni-dhi-and-ghi/>

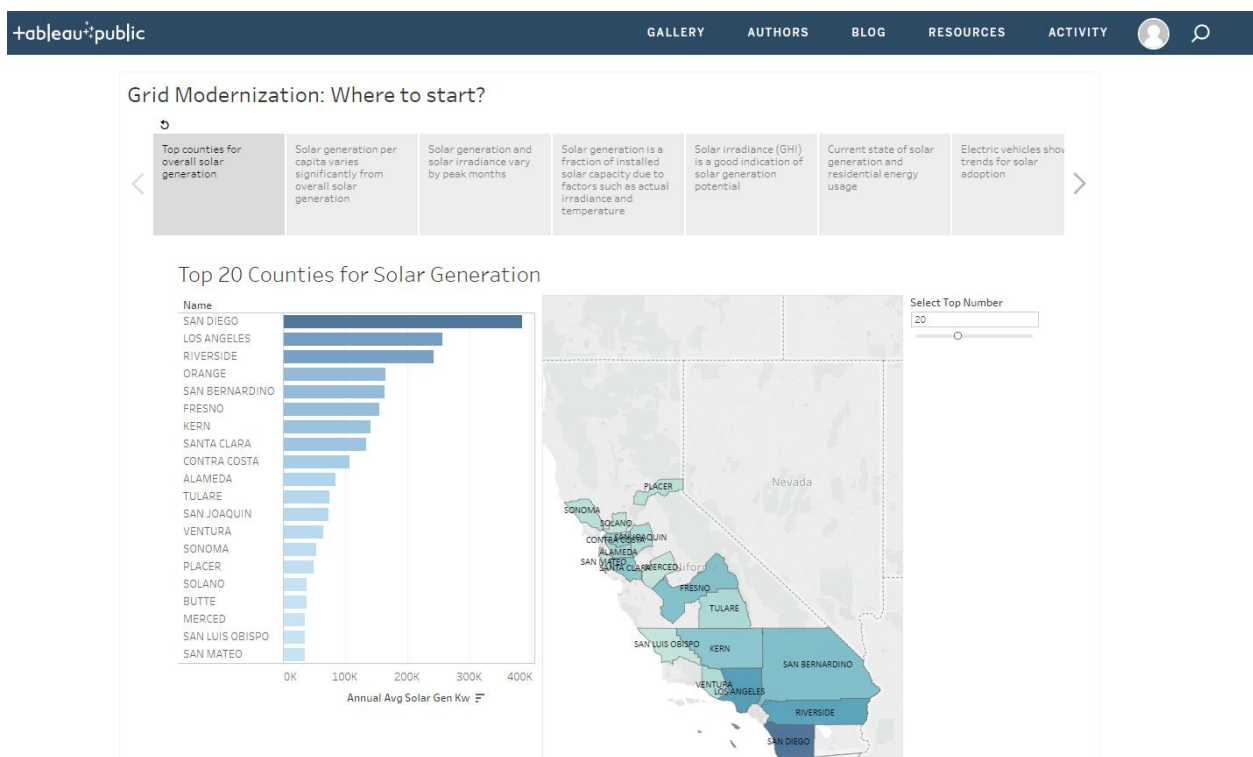
Section 5: Vincent Chu, Elizabeth Shulok

Irradiance (GHI) per day in the unit of kilowatt-hours per area of meter square (kWh/M^2). In order to be able to estimate the actual solar generation, the team calculated an index named SPI (Solar Potential Index), which ranges from 0 to 1.

First, an overall maximum potential GHI value is found with an aggregate query that unions GHI data across all the individual monthly columns. Second, the annual and monthly average GHI is divided by this maximum potential GHI. The idea is that the solar systems will be close to generating at their installed capacity when irradiance is at its maximum. Finally, after a set of annual and monthly SPI numbers have been calculated for each counties in the state of California, the amount of solar generation in kilowatt is calculated by multiplying the SPI (which is a decimal number between 0 and 1) to the total installed capacity (which in kW).

Our tool includes a front-end Tableau interface as well as a set of Spark-SQL queries that can be run from our EC2 instance.

Screenshot from our Tableau front-end:



<https://public.tableau.com/profile/elizabeth.shulok#!/vizhome/SolarProject/SolarAdoptionPrediction>

Screenshot of the results from running our Spark-SQL query top 20 solar gen counties.sql:

```

SET spark.sql.hive.version=1.2.1
SET spark.sql.hive.version=1.2.1
SET hive.cli.print.header=true
SET hive.cli.print.header=true
key      value
hive.cli.print.header  true
Time taken: 3.502 seconds, Fetched 1 row(s)
country annual_avg_solar_gen_kw annual_spi annual_avg_ghi_per_day installed_capacity ev_count 2014_res_usage_percapita
014_household_income
SAN DIEGO 384821.477117438 0.6583629893238434 5.55 584512.6220000004 489 0.002103369114284935 31043.0 63996.0
LOS ANGELES 256797.16547872828 0.6323699421965318 5.47 406086.92529999994 756 0.0020516084713352815 27987.0 55870.0
RIVERSIDE 242558.5080909094 0.6701631701631702 5.75 361939.47816000046 280 0.002905918711476681 23660.0 56592.0
ORANGE 165289.77015766798 0.6590621039290241 5.2 250795.43972000008 750 0.0022374756604244457 34416.0 75998.0
SAN BERNARDINO 163918.8810866666 0.6666666666666667 5.8 245878.32162999987 147 0.0022556431064948298 21384.0 54100.0
FRESNO 154730.24076843736 0.625 5.25 247568.38522949978 55 0.0027883107423181163 20231.0 45201.0
KERN 140579.00971796224 0.6398158803222094 5.56 219717.9126707 38 0.0026287234117968556 20467.0 48574.0
SANTA CLARA 133633.3467633452 0.623329283110571 5.13 214386.44129870003 865 0.002026501173067737 42666.0 93854.0
CONTRA COSTA 107588.65232724053 0.6162361623616235 5.01 174589.96874659995 362 0.0024543713970264698 38770.0 79799.0
ALAMEDA 84320.34492697785 0.6138855034811205 5.04 137355.16504970004 553 0.001788199629901156 36439.0 73775.0
TULARE 75190.82361458434 0.622093023255814 5.35 120867.49216550005 21 0.0026705176604873877 17888.0 42863.0
SAN JOAQUIN 73002.32969105305 0.623931623931624 5.11 117003.73388840006 49 0.0024550115442071446 22642.0 53253.0
VENTURA 65238.80201652892 0.639905548996458 5.42 101950.674 196 0.002255714623873464 33308.0 77335.0
SONOMA 53660.62250366707 0.5926829268292684 4.86 90538.49887449999 148 0.0025648675213675214 33361.0 63799.0
PLACER 49600.64272857907 0.608904933814681 5.06 81438.76305820003 70 0.0036847283276028133 35711.0 73747.0
SOLANO 38442.15228801726 0.6206896551724138 5.04 61934.57868770003 54 0.0023332359097350923 29132.0 67341.0
BUTTE 37961.259923116566 0.597357692307692 4.97 63548.829488999974 11 0.0032033497085724736 24430.0 43165.0
MERCED 35190.18433336104 0.6322815533980582 5.21 55655.87694949994 2 0.0026880159938127224 18464.0 43066.0
SAN LUIS OBISPO 34793.79789716969 0.6498194945848376 5.4 53543.788986200016 55 0.002257805796125167 30392.0 59454.0
SAN MATEO 34750.087455883324 0.5833333333333334 4.62 59571.578495799986 201 0.0019774341342585697 47198.0 91421.0
Time taken: 27.642 seconds, Fetched 20 row(s)

```

Screenshot of the results from running our Spark-SQL query solar_correlations.sql:

```

[w205@ip-172-31-10-8 solar]$ spark-sql --driver-class-path /usr/share/java/postgresql-jdbc.jar -f investigate/solar_correlations.sql
16/12/11 04:34:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/12/11 04:34:20 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/12/11 04:34:21 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/12/11 04:34:34 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
16/12/11 04:34:34 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
16/12/11 04:34:37 WARN DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
16/12/11 04:34:42 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.
SET hive.support.sql11.reserved.keywords=false
16/12/11 04:34:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/12/11 04:34:49 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/12/11 04:34:49 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/12/11 04:34:58 WARN DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
SET spark.sql.hive.version=1.2.1
SET spark.sql.hive.version=1.2.1
SET hive.cli.print.header=true
SET hive.cli.print.header=true
key      value
hive.cli.print.header  true
Time taken: 3.508 seconds, Fetched 1 row(s)
solar_gen_inn_corr solar_gen_solar_cap_corr solar_gen_ev_corr solar_gen_res_usage_corr solar_gen_pc_income_corr solar_gen_hh_income_corr
0.4561174012270995 0.9994593257052496 0.6999077532868224 -0.33732933581911617 0.059560572738075496 0.22598226839331223
Time taken: 31.333 seconds, Fetched 1 row(s)

```

Data Architecture

Backend Architecture

In order to build our analytical tools on multiple data sets from different sources, we deployed a HDFS Data Lake to allow flexibility in the data ingestion and exploration processes.

Because all of our data sets are highly structured, we thought that having a SQL-based relational database might be the most appropriate design, especially since we do not foresee incorporation of any unstructured data such as tweets or facebook updates.

We started with a Hive setup in order to make Tableau work using a Cloudera ODBC Driver for Apache Hive. While building the data transformation layer, we noticed that hive takes a relatively long time to run queries that involve a few joins and/or subqueries, which were required since we were “fusing” data sets from a variety of sources (just the reality of solar related data currently).

The team then considered using Spark-SQL, and did some benchmarking of Hive vs. Spark-SQL. As expected we saw a big improvement on many queries, even the rather basic ones such as the following:

```
SELECT * FROM spi ORDER BY annual_spi DESC;
```

The following are logs from running the exact same query on the same data set. Hive took more than 2x time to run the above query compared to Spark-SQL:

```
HIVE: Time taken: 88.447 seconds, Fetched: 58 row(s)  
SPARK-SQL: Time taken: 40.75 seconds, Fetched 58 row(s)
```

However, we did some research and it doesn't seem like Tableau would easily work with Spark-SQL with a default derby metastore. Tableau seems to work more flawlessly if the backend metastore was a Postgres database. Knowing that Hive was setup to point to a Postgres database (which was something set up by the `setup_ucb_complete_plus_postgres.sh` script run to setup the EC2 instance initially), the team decided to redirect Spark-SQL to point to the same backend metastore (i.e., the Postgres database) Hive also points to. However, what seems to be an easy copy of `hive-site.xml` from Hive's configuration directory to Spark's configuration directory took the team about 4 hours to trace the root cause of the errors thrown by the underlying Java code. Finally, the team found the solution, which includes specification of a jar library for Postgres and a change in the `hive-site.xml` file.

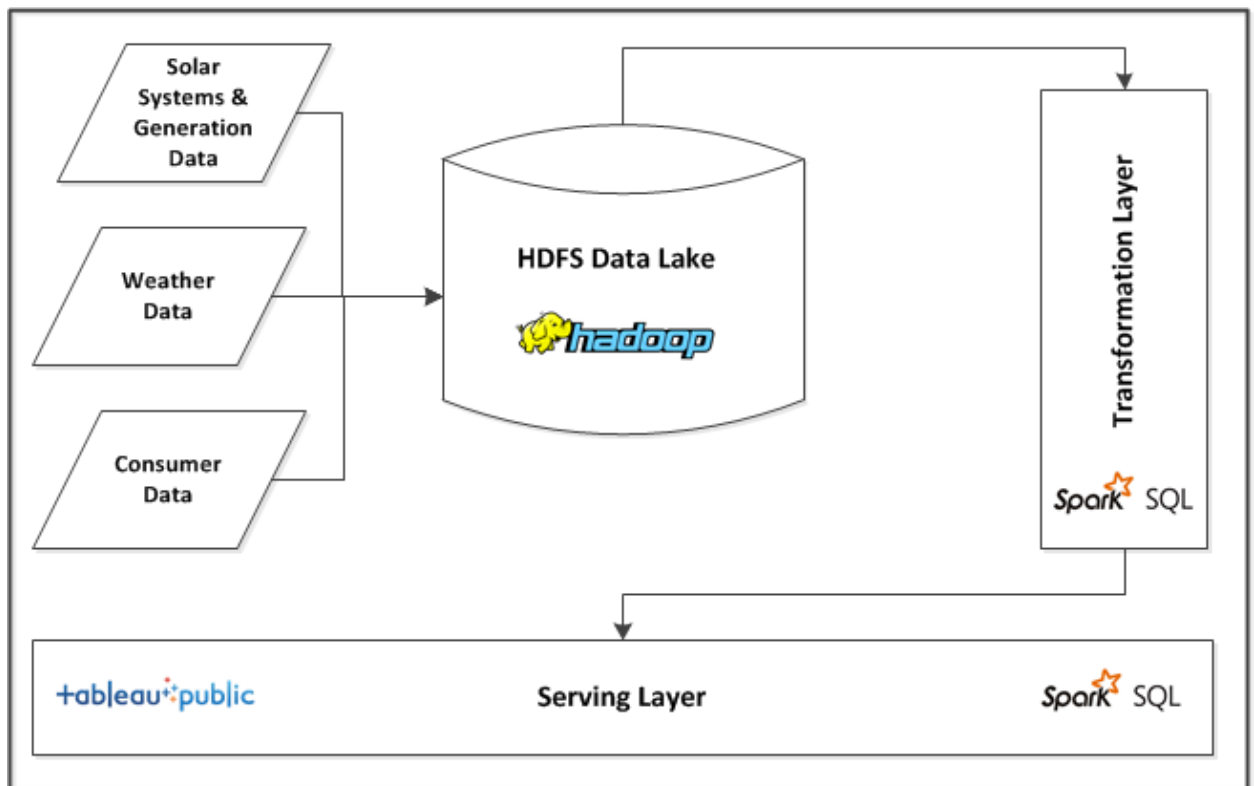
To sum up, we were able to obtain the architecture we wanted to explore: having Tableau (via Hive) and Spark-SQL points to the same backend data metastore.

Data Flow / Layers

A data transformation layer, aimed at cleansing, normalizing and conditioning the raw data into a practical, usable form, was built using Spark-SQL. This layer also contains a set of relatively complex intermediate queries that might need to be referenced over and over again. A set of such intermediate tables were thus created based on these transform queries.

Finally, we built a serving layer as an interface for the end users to obtain results from our final product. This layer consists of a Tableau front-end that presents data that shows solar generation in relationship to the proposed drivers that might be used to understand current solar adoption level and potentially project future growth. The Tableau front-end was designed and implemented as a “storyboard” aimed at enhancing the overall user experience of our tool.

A set of Spark-SQL queries that serve to ultimately answer the two aspects we focused on in regards to the research question was also made available as part of our tool.



Data Sources

The following table lists the finalized data sources we identified for our project. Because solar is still a rather segmented and relatively new industry with data coming from an array of government agencies and organizations, we definitely had a variety of data from various sources to deal with. This made our project very interesting and at the same time posed challenges in aligning data at different granularity and resolution.

Even though we performed our analysis at the county level, many data sets, such as interconnection data (i.e., information about each solar system installed in California), is at much finer granularity and thus have brought a huge volume of data initially. Of course, scripts in our transformation layer eventually take care of aggregating these data sets to the county level we ultimately need for building the final product.

Data Type	Data Name	Source	URL	Update Frequency
Setup	City	Wikipedia	https://en.wikipedia.org/wiki/List_of_cities_and_towns_in_California	Static
Setup	County	indexmundi	http://www.indexmundi.com/facts/united-states/quick-facts/california/population#table	Static
Weather	Irradiance	NREL	http://www.nrel.gov/gis/docs/SolarSummaries.xlsx	Static (10 to 11 years average updated every few years)
Solar System and Generation	Interconnections	CSI	http://www.californiadgstats.ca.gov/download/interconnection_nem_pv_projects/NEM_CurrentlyInterconnectedDataset_2016-08-30.zip	Quarterly – Half Yearly
Consumer	Household Income	Census Bureau	http://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?pid=ACS_14_5YR_B19301&prodType=table	Yearly
Consumer	Per Capita Income	American Fact Finder by Census Bureau	http://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?pid=ACS_14_5YR_B19301&prodType=table	Yearly
Consumer	Energy Usage (GWh)	CA Energy Commission	http://ecdms.energy.ca.gov/elecbycounty.aspx	Yearly
Consumer	Party Affiliation	Wikipedia	https://en.wikipedia.org/wiki/California_locations_by_voter_registration	Static

Error Detection & Data Cleansing

Since all of our data is based on county, we needed to be sure the county names would match up when we joined tables or ran queries. We knew from looking at the csv files that some counties were all uppercase and some were not. And the income tables listed the county name along with the text " County, California", for example "San Diego County, California" rather than simply "San Diego". These issues were address in our transformation scripts.

Our team used OpenRefine for further data validation. By opening each .csv file in OpenRefine, we verified that the county data was accurate. We first noted by doing a Text Facet on ca_counties.csv that there were 58 unique county names. A Google search confirmed that California has 58 counties.

A Text Facet on ca_cities.csv found only 55 county names. Using a list of all counties in California and comparing to the Text Facet list, we found that the 3 missing counties - Alpine, Mariposa and Trinity - contain no cities.

Since the ghi_by_county.csv file contain records for all states, we first did a text filter to only select records from CA, then a Text Facet revealed that the resulting 58 records corresponded to 58 unique county names.

Both electricity_usage.csv and party_affiliation.csv contained 58 records and a Text Facet confirmed 58 unique county names.

Percapita_income.csv and household_income.csv contained 59 rows. A Text Facet revealed one line with the word Geography in place of the county name. Clicking on it revealed it was from the header line.

The interconnection data in NEM_CurrentlyInterconnectedDataset_2016-08-30.csv was the most difficult to check. The file is nearly 200 MB and we were initially unable to open it in OpenRefine, which crashed the computer when trying to open it. We were able to figure out how to increase the memory size for OpenRefine and then the file opened without issue. A Text Facet showed 56 unique Service County names. However that included duplicates due to Orange and Riverside having uppercase equivalents listed separately. Doing a cluster and then merging brought the number of unique names to 54 and we confirmed 4 missing counties from the data: Alpine, Del Norte, Modoc, Siskiyou.

Section 5: Vincent Chu, Elizabeth Shulok

Refine NEM_CurrentlyInterconnectedDataset_2016 08 30 csv Permalink

Facet / Filter Undo / Redo 1

Refresh Reset All Remove All

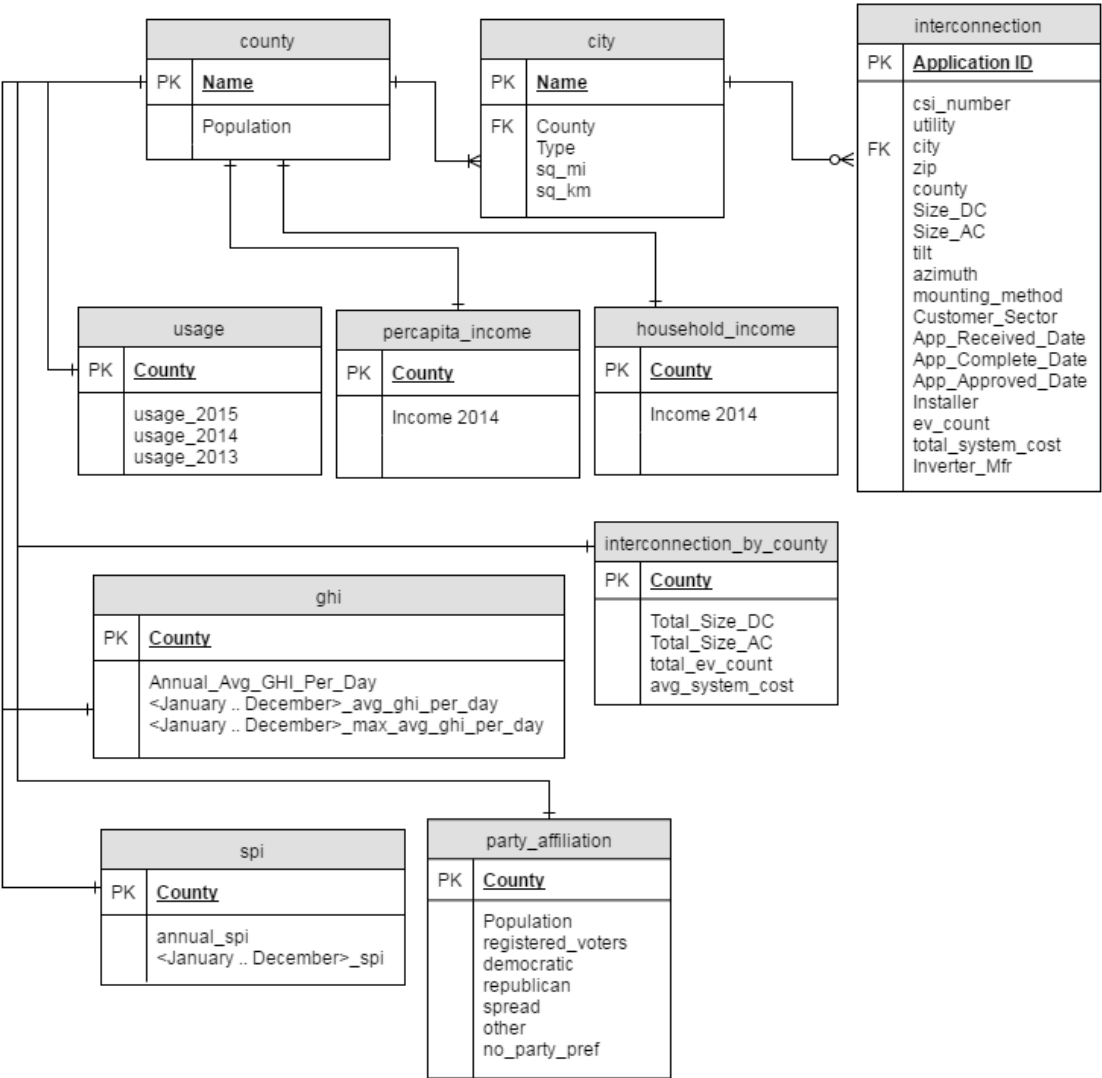
551505 rows Extensions:

Show as: rows records Show: 5 10 25 50 rows « first < previous 1 - 10 next > last »

Service County	Application Id	Matched CSI Ap	Utility	Service City	Service Zip	Service County	Technology Type	System S
Alameda 19925	1. PGE-INT-10		PGE	Fairfax	94930	Marin	Solar PV	2.544
Amador 809	2. PGE-INT-100		PGE	Berkeley	94707	Alameda	Solar PV	4.032
Butte 6423	3. PGE-INT-1000		PGE	Corte Madera	94925	Marin	Solar PV	2.64
Calaveras 1324	4. PGE-INT-10000		PGE	Campbell	95008	Santa Clara	Solar PV	
Colusa 750	5. PGE-INT-100001	PGE-CSI-56467	PGE	El Dorado Hills	95762	El Dorado	Solar PV	17.7826
Contra Costa 24824	6. PGE-INT-100002	PGE-CSI-56484	PGE	Marysville	95901	Yuba	Solar PV	18.174
El Dorado 6742	7. PGE-INT-100004	PGE-CSI-56615	PGE	Kerman	93630	Fresno	Solar PV	511.872
Fresno 25059	8. PGE-INT-100005	PGE-CSI-56620	PGE	Sanger	93657	Fresno	Solar PV	511.872
Glenn 1222	9. PGE-INT-100006	PGE-CSI-56666	PGE	Manteca	95336	San Joaquin	Solar PV	5.5488
Humboldt 1051	10. PGE-INT-100007	PGE-CSI-56738	PGE	Shingle Springs	95682	El Dorado	Solar PV	12.5216
Imperial 13								
Inyo 339								

After verifying that the county data looked correct and would match up for our queries, we created sql scripts to import the csv tables into raw tables with string data types. Then we wrote a set of transformation sql queries to condition the data and create new tables with proper data types for each field. For all tables, the county name was converted to upper case. For the income tables (per capita and household), the county name was shortened to match the format of the other tables, removing the " County, California" text.

Entity-Relationship Diagram



Environment Setup

Data Lake Set-up and Load Script

We created a bash script to set up local directories for our project, grab raw data files from various sources, prepare directory structure in the HDFS data lake for the data load scripts and place raw files in the HDFS directories created. This enables our team to set up the appropriate environment in our own AWS EC2 instances for unit testing. It will also enable the instructor (and any other clients) to be able to execute our project code.

Tableau

To view our data in Tableau, the Cloudera ODBC driver needs to be installed and configured. Install the Cloudera ODBC Driver for Apache Hive if not already installed. Then open the ODBC data source, go to the System DSN tab and configure the Cloudera Hive DSN as shown below, using the Host address of the currently running AWS with the solar project loaded:

Cloudera ODBC Driver for Apache Hive DSN Setup

Data Source Name: Sample Cloudera Hive DSN

Description: Sample Cloudera Hive DSN

Hive Server Type: Hive Server 2

Service Discovery Mode: No Service Discovery

Host(s): ec2-54-159-87-107.compute-1.amazonaws.com

Port: 10000

Database: default

ZooKeeper Namespace:

Authentication Mechanism: User Name

Realm:

Host FQDN: _HOST

Service Name: hive

☐ Delegate Kerberos Credentials

User Name:

Password:

☐ Save Password (Encrypted)

Delegation UID:

Thrift Transport: SASL

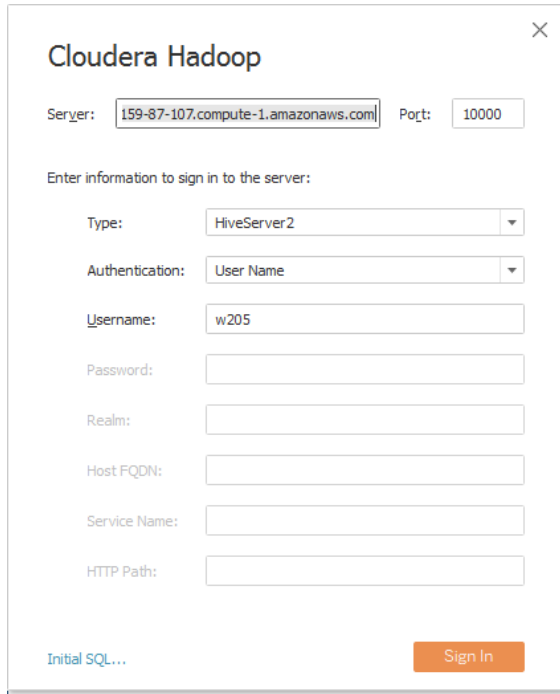
HTTP Options SSL Options

Advanced Options... Logging Options...

v2.5.20.1006 (64 bit) Test OK Cancel

Note that Tableau uses port 10000 and this port must be opened on the EC2 instance.

In Tableau, connect to a Cloudera Hadoop server. Configure it as shown below, using the Server address of the currently running AWS with the solar project loaded.

The image shows a 'Cloudera Hadoop' connection window. At the top, the 'Server' field contains '159-87-107.compute-1.amazonaws.com' and the 'Port' field contains '10000'. Below this, a section titled 'Enter information to sign in to the server:' contains several fields: 'Type' is set to 'HiveServer2', 'Authentication' is set to 'User Name', 'Username' is 'w205', and 'Password', 'Realm', 'Host FQDN', 'Service Name', and 'HTTP Path' are all empty. At the bottom left is a link 'Initial SQL...' and at the bottom right is an orange 'Sign In' button.

To publish our data to Tableau Public, go to the Server menu and select Tableau Public – Save to Tableau Public. If an account is not already set up with Tableau, one must be created.

Data Load

We created multiple Spark SQL script files to load the data into raw tables from the .csv data files. For this step, all fields are loaded as strings.

- **load_setup_data.sql** - Loads the city and county data into raw tables.
- **load_consumer_data.sql** - Loads the energy usage, per capita income, household income, and political party affiliation data into raw tables. We were undecided about using per capita or household income and decided to explore both.
- **load_solar_data.sql** - Loads the interconnections data into a raw table.
- **load_weather_data.sql** - Loads Global Horizontal Irradiance data by county into a raw table. Loads data for all states.

Data Transform

We created multiple Spark SQL script files to transform the raw tables that were created from the .csv data files. The following .sql files perform simple transformations on the fields to normalize the data and create fields of the appropriate data types. As noted previously, these scripts also do basic data cleaning on the county names as needed.

- **create_city.sql** - Text fields including county name are cast to all uppercase and numeric values such as sq_mi are cast as doubles.
- **create_county.sql** - County name cast to all uppercase and population as a BIGINT.
- **create_county_info.sql** - Creates a county_info table that includes a calculation of population per area by using data from both the city and county tables. Data is ordered by population per square mile.
- **create_percapita_income.sql** - Creates a percapita_income table of income per capita for 2014, by county.
- **create_household_income.sql** - Creates a household_income table for 2014, by county.
- **create_usage.sql** - Creates a usage table with data for 3 years, with usage values cast as doubles.
- **create_party_affiliation.sql** - Creates a party_affiliation table, with DOUBLE values for percentage of voters that are Republican, Democratic, Other, or No Party Preference.
- **create_ghi_data.sql** - Creates the ghi table by selecting only the data relevant to California. ghi values are casted as double.
- **create_spi_data.sql** - Creates the spi table by calculating the Solar Potential Index by dividing annual and monthly GHI average by the maximum potential GHI across all the months.
- **create_interconnection.sql** - This was one of the more complex data sources and the fields from the corresponding raw table are cast to various data types. Additionally, only those records in the raw table where technology_type = 'Solar PV' are selected.
- **create_interconnection_by_county.sql** - Creates an aggregate table for interconnection data that calculates, at the county level, sum of installed solar capacity, total number of EVs, etc.
- **create_usage_percapita_info.sql** - Creates a usage_info table that calculates the energy usage per capita using data from the usage and county tables. Data is ordered by usage per capita. This identifies the counties with the highest energy use per person.
- **create_solar_summary_by_year.sql** - Sum of installed solar capacity, number of EVs, average solar irradiance, electricity usage and household income by county, ordered by installed solar capacity grouped by year
- **create_solar_summary_by_month.sql** - Sum of installed solar capacity, number of EVs, average solar irradiance, electricity usage and household

income by county, ordered by installed solar capacity grouped by month

- **create_solar_generation_by_county.sql** - Calculates the total solar generation based on total installed solar capacity (AC) x SPI.

Serving Layer

1. Tableau Front-end Dashboards:

We created multiple dashboards in Tableau to explore the data and combined them into a single story board to communicate the information.

- **Top Counties** – Displays a bar graph of the top N counties for solar generation, with a control to let the user select N. Default is 10. A map alongside the graph displays the top N counties, shaded based on solar generation.
- **Top Counties Per Capita** – Same as above, but based on solar generation per capita rather than total solar generation.
- **Monthly** – Displays maps of solar generation and average daily GHI based on the month selected by the user.
- **Compare Installed Capacity** – Map of installed capacity alongside map of annual average solar generation.
- **Compare GHI** – Map of annual average solar generation alongside map of annual average GHI per day.
- **Compare Usage** – Map of annual average solar generation alongside map of residential energy usage per capita.
- **Compare EV** – Map of annual average solar generation alongside map of number of purchased electric vehicles.
- **Compare Income** – Map of annual average solar generation alongside map of income per capita.
- **Compare Household Income** – Map of annual average solar generation alongside map of household income
- **Compare Party Affiliation** – Map of annual average solar generation alongside map of political party affiliation.

2. Spark-SQL Scripts:

In addition to the visualization provided by our Tableau front-end, we also built a set of backend queries that can be run using Spark-SQL².

- **solar_correlations.sql** – Calculates the Pearson's Correlation number between solar generation and each of the potential drivers for growth in solar adoption, e.g., irradiance, number of electrical vehicles,

² Please be aware that the jar file for postgres must be specified when invoking spark-sql (see below):

```
spark-sql --driver-class-path /usr/share/java/postgresql-jdbc.jar -f <filename>
```

household income, income per capita, etc.

- **top_20_ev_counties.sql** – Displays the top 20 counties with the highest number of electrical vehicles, with the corresponding estimated solar generation in kW.
- **top_20_household_income_counties.sql** - Displays the top 20 counties with the highest household income, with the corresponding estimated solar generation in kW.
- **top_20_percapita_income_counties.sql** - Displays the top 20 counties with the highest income per capita, with the corresponding estimated solar generation in kW.
- **top_20_irradiance_counties.sql** - Displays the top 20 counties with the highest average daily GHI (kWh/m²), with the corresponding estimated solar generation in kW.
- **top_20_solar_capacity_counties.sql** - Displays the top 20 counties with the highest total installed capacity (kW), with the corresponding estimated solar generation in kW.
- **top_20_solar_gen_counties.sql** – Displays the top 20 counties with the highest estimates solar generation in kW, along with all the potential drivers being studied.
- **top_20_usage_counties.sql** – Displays the top 20 counties with the highest electricity usage (in other words, energy consumption) per capita in the residential sector, with the corresponding estimated solar generation in kW.

Build/Run Instructions

To run this project, complete the following steps.

EC2 start up:

1. Start an AWS EC2 instance using the "UCB W205 Spring 2016" AMI.
2. Check security rules to ensure following TCP ports on 0.0.0.0/0 are open.
 - a. 22
 - b. 4040
 - c. 7180
 - d. 8080
 - e. 8088
 - f. 50070
 - g. 10000
3. Create an EBS volume of 200 GB and attach it to the instance created above.
4. Connect to the EC2 instance using ssh: `ssh -i "<key file name>" root@<EC2 host path>`.
5. Use `fdisk -l` to find the disk path of the EBS volume created in step 3 above. Note down this path for step 7.
6. Set the /data directory as readable, writable and executable by all users, since /data is where the disk will be mounted.

```
$ chmod a+rwX /data
```

7. Get and run the set up script to set up hadoop, postgres, hive, spark, etc.

```
$ wget https://s3.amazonaws.com/ucbdatasciencew205/setup_ucb_complete_plus_postgres.sh
$ chmod +x ./setup_ucb_complete_plus_postgres.sh
$ ./setup_ucb_complete_plus_postgres.sh <the disk path from step 5>
```

Redirect spark-sql to point to the same metastore as hive:

1. While logged on as the root user, change the value of the `hive.metastore.schema.validation` to false in the `/data/hadoop/hive/conf/hive-site.xml` file. Below is the part of the `hive-site.xml` file where the value of the `<value>` tag needs to be set to false.

```
<property>
<name>hive.metastore.schema.validation</name>
<value>>false</value>
</property>
```

Alternatively, you can overwrite the `hive-site.xml` file in the `/data/hadoop/hive/conf` directory in your EC2 instance with the file of the same name in the Setup folder of the Solar repository.

2. Change to the w205 user.

```
$ su - w205
```

3. Copy the hive-site.xml to the \$SPARK_HOME/conf directory:

```
$ cp /data/hadoop/hive/conf/hive-site.xml $SPARK_HOME/conf/
```

You can alternatively copy the "hive-site.xml" file from the Setup/ folder of the Solar repository to \$SPARK_HOME/conf

After these steps, hive and spark-sql will be pointing to and working on the same backend data metastore. Please note that after the steps above, spark-sql must be invoked with the --driver-class-path flag to specify the postgresql-jdbc.jar file to be included since the backend data metastore was set to be a postgres database by the setup_ucb_complete_plus_postgres.sh script run in step 7 of the EC2 start-up section above. Otherwise, an error will be thrown since spark-sql cannot connect to the postgres database behind hive.

```
spark-sql --driver-class-path /usr/share/java/postgresql-jdbc.jar
```

Steps to run the Solar Course Project using git clone

1. Change Directory to home directory for the w205 user:

```
$ cd /home/w205/
```

2. Get all the files for this project by git clone:

```
$ git clone https://github.com/w205-5-Solar-Project/Solar.git
```

3. Execute the run_project.sh script to kick off the load_data_lake.sh and create_sparksql_tables.sh scripts:

```
$ cd Solar
$ bash Setup/run_project.sh
```

Steps to set up and run our Tableau workbook

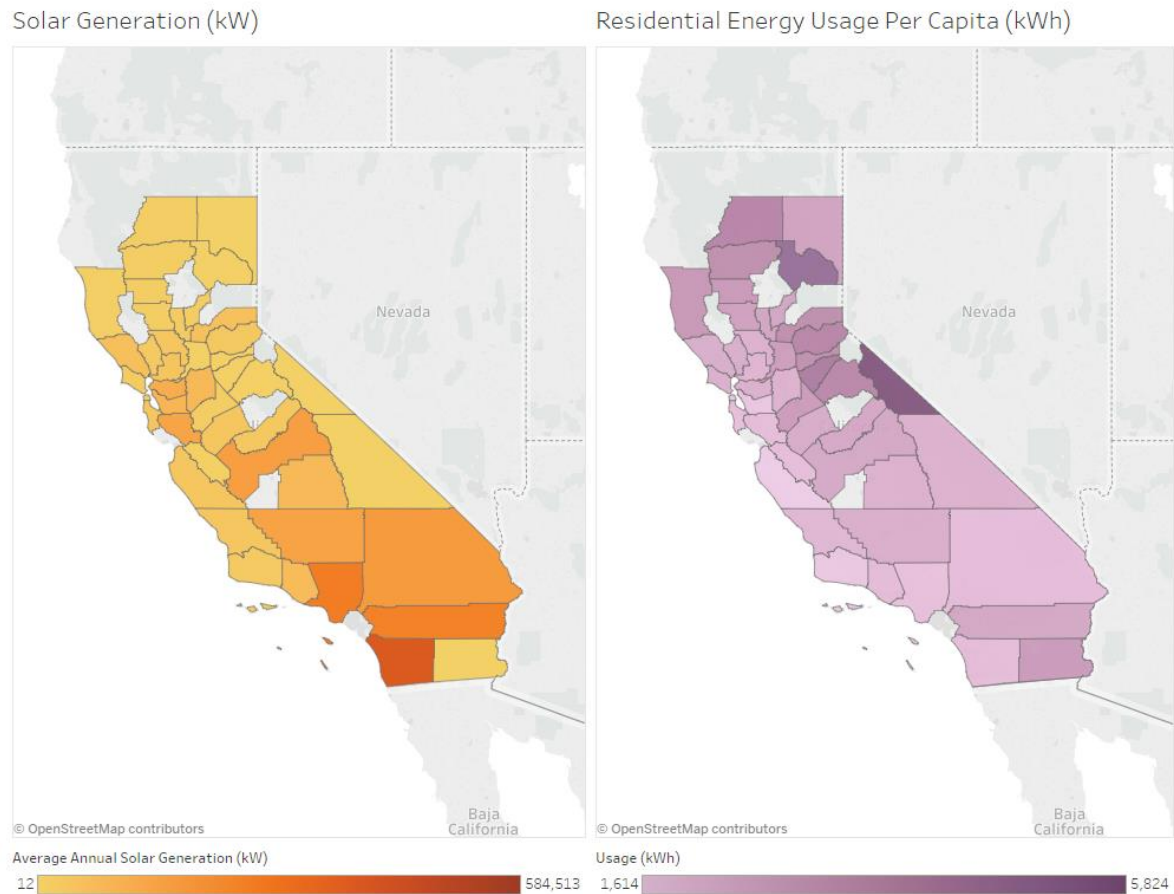
1. Start hiveserver2 from the EC2 instance:
2. Make sure port 10000 is opened on the EC2 instance (should have been during EC2 star-up).
3. Install the Cloudera ODBC Driver for Apache Hive.
4. Configure the Cloudera ODBC Driver by going to the "ODBC Data Source Administrator" window via Control Panel. Use the following settings:

```
Host: <EC2 host path>
Port: 10000
Database: default
HiveServer Type: HiveServer2
```

When opening the "SolarProject.twb" Tableau workbook from the "Investigate" folder of the Solar repository, please change the Host to the full path of your EC2 instance.

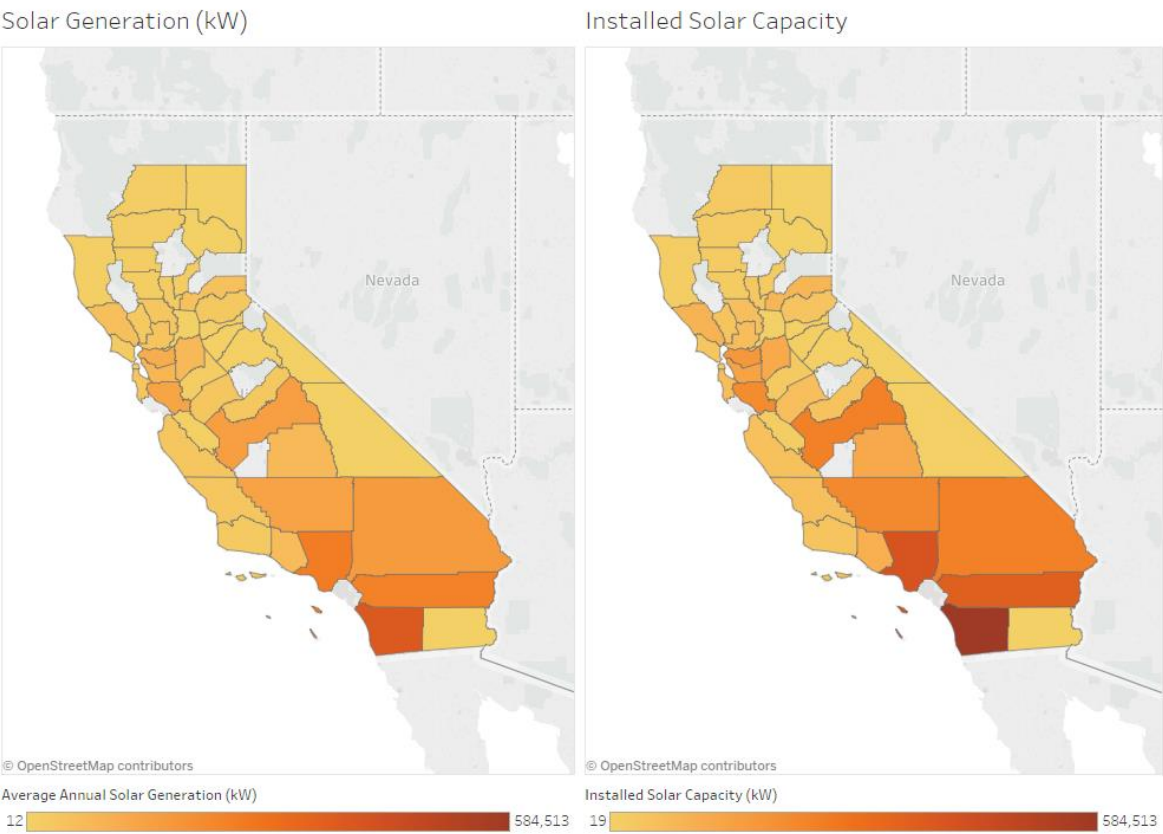
Results & Use Cases

When looking at the data in Tableau, we noticed a few interesting things. When comparing solar generation with energy usage, areas with the highest solar generation were not the areas with the highest usage per capita. And the areas with the highest usage had relatively low solar generation levels.



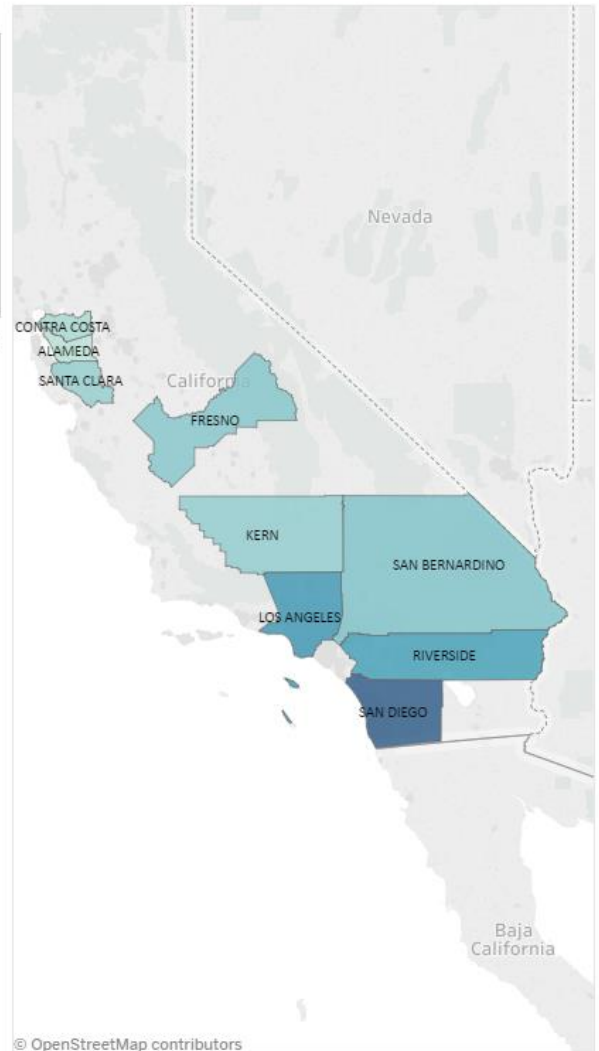
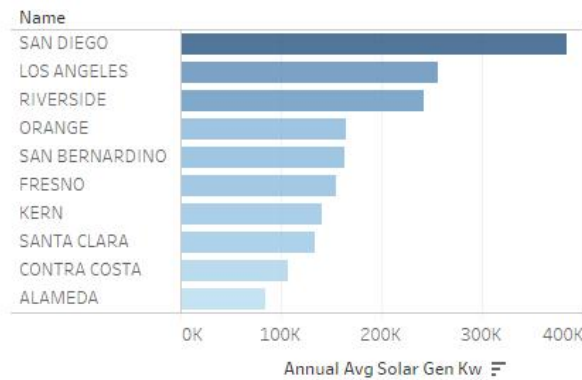
Less surprising is the correlation between solar generation and solar capacity. Solar generation will never be as high as the capacity, and areas with high solar capacity had the greatest differential between capacity and solar generation.

Section 5: Vincent Chu, Elizabeth Shulok



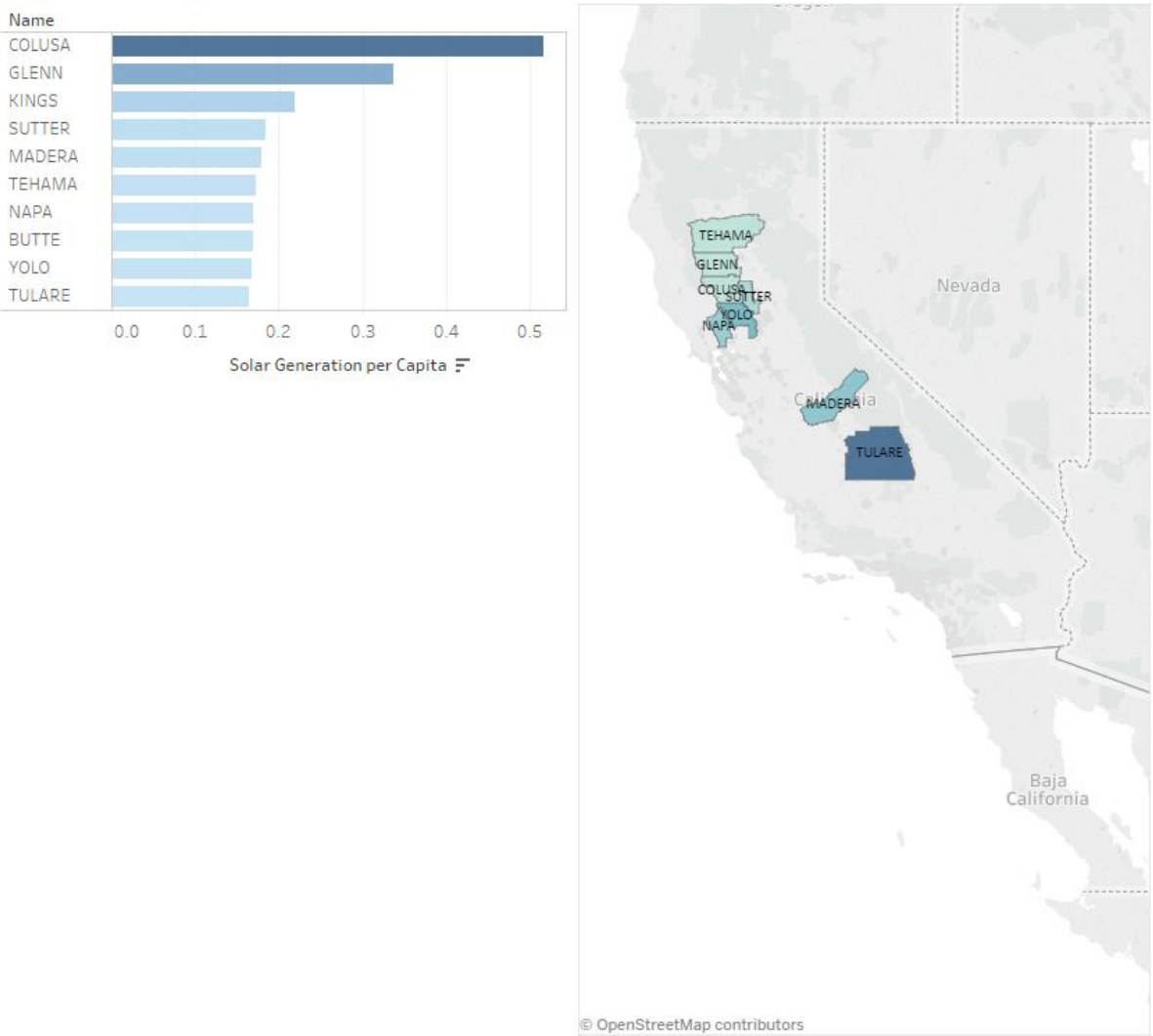
Another interesting find was the stark difference between the top 10 counties based on total solar generation and the top 10 counties based on solar generation per capita. In fact, there is no overlap in these two top 10 lists as seen below.

Top 10 Counties for Solar Generation



We built a dashboard that shows the top 10 counties with the highest solar generation. Not surprisingly, top 5 are all counties in Southern California where there is a lot of sunlight all year round. Fresno in the inland empire is next on the list. It is a county with a significant solar penetration and thus a pilot ground for a lot of solar related technologies for the Pacific Gas & Electric company that operates in that county.

Top 10 Counties for Solar Generation Per Capita



We also built a dashboard that shows the top 10 counties with the highest solar generation per capita. The list top 10 list is totally different since this ranking favors the counties with much less population than the ones seen in the previous top 10 list for total solar generation. Thus, while this might be an interesting FYI, it might not be the most applicable tool for our clients.

Section 5: Vincent Chu, Elizabeth Shulok

Among the backend Spark-SQL scripts, the most interesting results came from the solar_correlations.sql script, which calculates the Pearson's Correlation number between solar generation and each of the potential drivers for growth in solar adoption.

	Correlation with Estimated Solar Generation (kW)
Irradiance	0.456117401
Installed Solar PV Capacity	0.999459326
Number of Electrical Vehicles	0.699907753
Residential Energy Usage Per Capita	-0.337329336
Income Per Capita	0.059560573
Household Income	0.225982268

As shown in the table above, the drivers with the most correlation with estimated solar generation (besides installed solar PV capacity, which as expected, is almost perfectly correlated) are number of electrical vehicles (EVs) and amount of irradiance.

With regards to financial factors, income per capita actually has almost no correlation with solar generation. On the other hand, household income does have a 0.23 correlation with solar generation. This makes high-level sense since solar system installation is a household decision. But in general, with costs of solar panels, DC/AC inverters and other solar equipment continue to drop year over year, financial factors might have decreasing influence.

Challenges

Type	Challenge	Resolved?	Mitigation
Data	Initially, we wanted to use a finer granularity than county for income and usage data	N	Settled for county level data
Data	The accuracy of our solar generation estimation tool relies on the two main factors: irradiance and installed capacity (from interconnection data). The update frequency for both data sets might pose accuracy issues, especially if newly installed solar systems are not updated timely in the interconnection data sets.	N	There might not be anything we can do to influence the update frequency (or velocity) of public data sets. From our side, maybe some kind of a confidence intervals can be calculated.
Visualization	Wanted to color counties of a map based on whether it was more democratic or republican. The data included No Party Pref and Other. I wanted to color based on % Democrat using a red-blue color gradient, but couldn't find any settings that worked for counties when both R and D were lower than 40%.	Y	I created a ratio of D/R and then colored the counties by the ratio value, setting the center point for the color spectrum to .89 after testing different values to see what yielded the most accurate map coloring.
Backend	Could not get sparkpy work with HiveContext, kept getting "javax.jdo.JDOFatalInternalException: Error creating transactional connection factory", which seems to be caused by incompatible versioning of Hadoop and SPARK.	N	Included Spark-sql queries only (no pyspark scripts)
Human Resources	Since our group got reduced in number of data scientists in the middle of the project, availability of time and brain power became one of the main challenges in bringing this project to the finish line.	Y	Extended scope cut. Lost sleep for the remaining team.

Future Frontier (*Potential Next Steps*)

1. We limited the scope of our project to a single state to simplify data acquisition. We chose California due to its high solar adoption rates and public data sets. One obvious extension of this tool would be to extend it to include all 50 states. Data acquisition would be burdensome since of the data used for this project, only income (per capita and household) and solar irradiance are available from national sources. Energy usage and Interconnection Data come from state sources.
2. In order to accommodate more states and more data sets, we need to consider scaling issues. We can potentially deploy S3 storage on AWS. We can potentially segment data and have logic to intelligently keep more recent data on EBS and swap more stale data to S3.
3. Another enhancement to this project could be the use of machine learning to more currently predict solar adoption based on relationship between the potential drivers and actual solar generation.
4. Similar to #3. Above, machine learning methodology can be used to more accurately estimate actual solar generation by modeling effect of local and regional shading, e.g., irradiance blocked by vegetation, buildings, mountain ranges, etc.