EV vs Air Quality Project

By Vivaan Kuruganti

Table of contents

The Problem

With significant tax incentives and the being given to EV purchases from brands like Tesla, Hyundai, GM, Cadillac, governments would need a way of predicting the effect of increased purchases of such vehicles on state and county air quality and how those changes might influence respiratory condition or diseases.

The Goal

Predict the air quality for each quarter in each county in California as close as possible based on all types of EV sales data, gas price data, natural gas consumption data, and electricity consumption data.

Why -

Predicting air quality based on changes in number of EVs/ HEVs on the road

- Why this is useful:
    - Federal or state governments could use this kind of a tool to determine how many tax incentives to provide for EVs based on their desired outcome for air quality.
    - Health agencies could link predicted air quality to expected rates of asthma attacks, COPD flare-ups, or other respiratory issues. This would help estimate hospital admissions and healthcare costs, and guide resource allocation.
    - Cities/ municipalities could decide where to invest in infrastructure such as EV charging stations, bus electrification, or low-emission zones.
    - Insurance companies could use the predictions to model long-term health risks for populations and adjust health or life insurance premiums accordingly.

Data Source 1 -

<u>ZEV + LDV Sales</u>

- Data for ZEV Sales and LDV Sales and ZEV Sales share for each county and each quarter in California going back to 2008
- The data is from the California Energy Commision
- The data is from 2008 - 2025 and is split up by counties for each quarter

- Where is the data from?
- What information does the data provide?
- What is the timeframe of the data? (yearly, quarterly, range of years provided)

Data Source 2 -

Air Quality Data

- Daily AQI value for each county and quarter in California from 2008 - 2024
- The data is from the California Air Resources Board
- The data is split up into data sets by year so each data set represents one year
- Each data shows each days air quality for the whole year by showing the PM 2.5 and the daily AQI average

Data Source 3-

Gas Price Data-

- Average monthly gas price data in California from 2000 - 2025
- The data is from the US Energy Information Administration
- It is split up by average monthly gas price data for all of california
- It goes from May 2000 - June 2025

Data Source 4-

Natural Gas Consumption -

- Natural Gas Consumption Data for each county in California from 1990 - 2023
- The data is from the California Energy Commision
- It is split up by year for each county in California
- The data goes from 1990 - 2023
- For each year and county the data is split up into sectors and if it is residential or non residential
- The data is in MMtherms

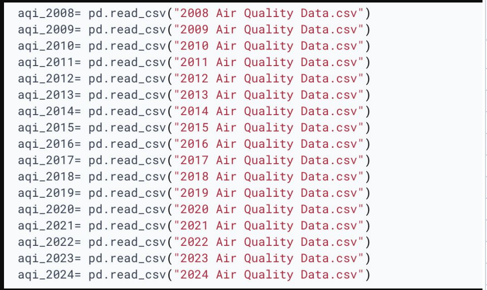
Data Source 5 -

Electricity Consumption -

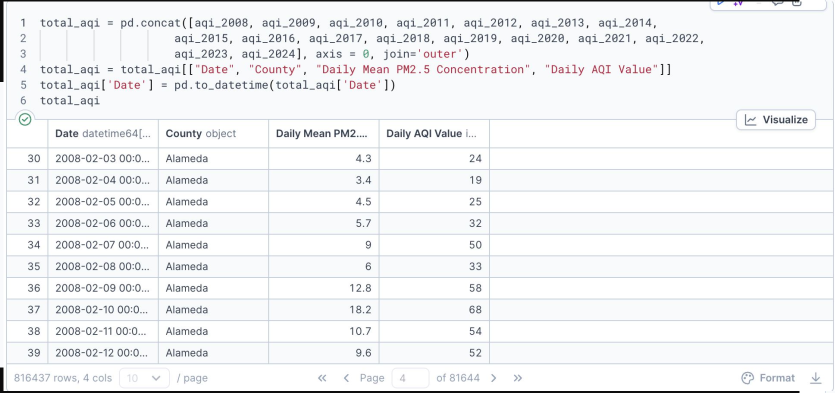
- Electricity Consumption Data for each county in California from 1990 - 2023
- The data is from the California Energy Commision
- It is split up by year for each county in California
- The data is in GWh
- The data goes from 1990 - 2023
- For each year and county the data is split up into sectors and if it is residential or non residential

Data Cleaning - Air Quality Data

- Merged all air quality data sets from 2008 - 2024 into one table
- Averaged daily data for every county
- Merged the County, Quarters and year into one data set called County_Quarter
- Made the County_Quarter column uppercase



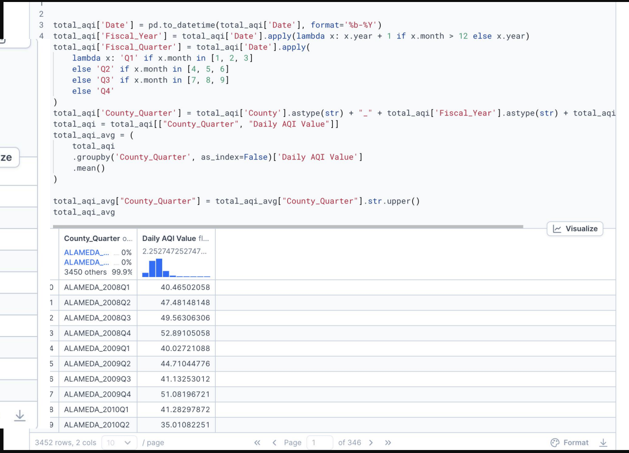
```
aqi_2008= pd.read_csv("2008 Air Quality Data.csv")
aqi_2009= pd.read_csv("2009 Air Quality Data.csv")
aqi_2010= pd.read_csv("2010 Air Quality Data.csv")
aqi_2011= pd.read_csv("2011 Air Quality Data.csv")
aqi_2012= pd.read_csv("2012 Air Quality Data.csv")
aqi_2013= pd.read_csv("2013 Air Quality Data.csv")
aqi_2014= pd.read_csv("2014 Air Quality Data.csv")
aqi_2015= pd.read_csv("2015 Air Quality Data.csv")
aqi_2016= pd.read_csv("2016 Air Quality Data.csv")
aqi_2017= pd.read_csv("2017 Air Quality Data.csv")
aqi_2018= pd.read_csv("2018 Air Quality Data.csv")
aqi_2019= pd.read_csv("2019 Air Quality Data.csv")
aqi_2020= pd.read_csv("2020 Air Quality Data.csv")
aqi_2021= pd.read_csv("2021 Air Quality Data.csv")
aqi_2022= pd.read_csv("2022 Air Quality Data.csv")
aqi_2023= pd.read_csv("2023 Air Quality Data.csv")
aqi_2024= pd.read_csv("2024 Air Quality Data.csv")
```



```
total_aqi = pd.concat([aqi_2008, aqi_2009, aqi_2010, aqi_2011, aqi_2012, aqi_2013, aqi_2014,
                       aqi_2015, aqi_2016, aqi_2017, aqi_2018, aqi_2019, aqi_2020, aqi_2021, aqi_2022,
                       aqi_2023, aqi_2024], axis = 0, join='outer')
total_aqi = total_aqi[["Date", "County", "Daily Mean PM2.5 Concentration", "Daily AQI Value"]]
total_aqi['Date'] = pd.to_datetime(total_aqi['Date'])
total_aqi
```

	Date datetime64[...	County object	Daily Mean PM2....	Daily AQI Value i...
30	2008-02-03 00:0...	Alameda	4.3	24
31	2008-02-04 00:0...	Alameda	3.4	19
32	2008-02-05 00:0...	Alameda	4.5	25
33	2008-02-06 00:0...	Alameda	5.7	32
34	2008-02-07 00:0...	Alameda	9	50
35	2008-02-08 00:0...	Alameda	6	33
36	2008-02-09 00:0...	Alameda	12.8	58
37	2008-02-10 00:0...	Alameda	18.2	68
38	2008-02-11 00:0...	Alameda	10.7	54
39	2008-02-12 00:0...	Alameda	9.6	52

816437 rows, 4 cols 10 / page Page 4 of 81644



```
total_aqi['Date'] = pd.to_datetime(total_aqi['Date'], format='%b-%Y')
total_aqi['Fiscal_Year'] = total_aqi['Date'].apply(lambda x: x.year + 1 if x.month > 12 else x.year)
total_aqi['Fiscal_Quarter'] = total_aqi['Date'].apply(
    lambda x: 'Q1' if x.month in [1, 2, 3]
    else 'Q2' if x.month in [4, 5, 6]
    else 'Q3' if x.month in [7, 8, 9]
    else 'Q4'
)
total_aqi['County_Quarter'] = total_aqi['County'].astype(str) + "_" + total_aqi['Fiscal_Year'].astype(str) + total_aqi
total_aqi = total_aqi[["County_Quarter", "Daily AQI Value"]]
total_aqi_avg = (
    total_aqi
    .groupby('County_Quarter', as_index=False)['Daily AQI Value']
    .mean()
)

total_aqi_avg["County_Quarter"] = total_aqi_avg["County_Quarter"].str.upper()
total_aqi_avg
```

	County_Quarter o...	Daily AQI Value fl...
	ALAMEDA_... 0% ALAMEDA_... 0% 3450 others 99.9%	2.252747252747...
0	ALAMEDA_2008Q1	40.46502058
1	ALAMEDA_2008Q2	47.48148148
2	ALAMEDA_2008Q3	49.56306306
3	ALAMEDA_2008Q4	52.89105058
4	ALAMEDA_2009Q1	40.02721088
5	ALAMEDA_2009Q2	44.71044776
6	ALAMEDA_2009Q3	41.13253012
7	ALAMEDA_2009Q4	51.08196721
8	ALAMEDA_2010Q1	41.28297872
9	ALAMEDA_2010Q2	35.01082251

3452 rows, 2 cols 10 / page Page 1 of 346

Data Cleaning ZEV+LDV Sales

- Got rid of all data after 2024
- Got rid of out of state data
- Merged the county, quarters and year into one data set called County_Quarter
- Made the County_Quarter column uppercase

Data Cleaning - Natural Gas Consumption

- Averaged MMtherms data for each quarter and type of sector
- Created Quarter column by multiplying the data by 4 for each county
- Got rid of all data before 2008
- Created County_Quarter Column

Data Cleaning - Electricity Consumption

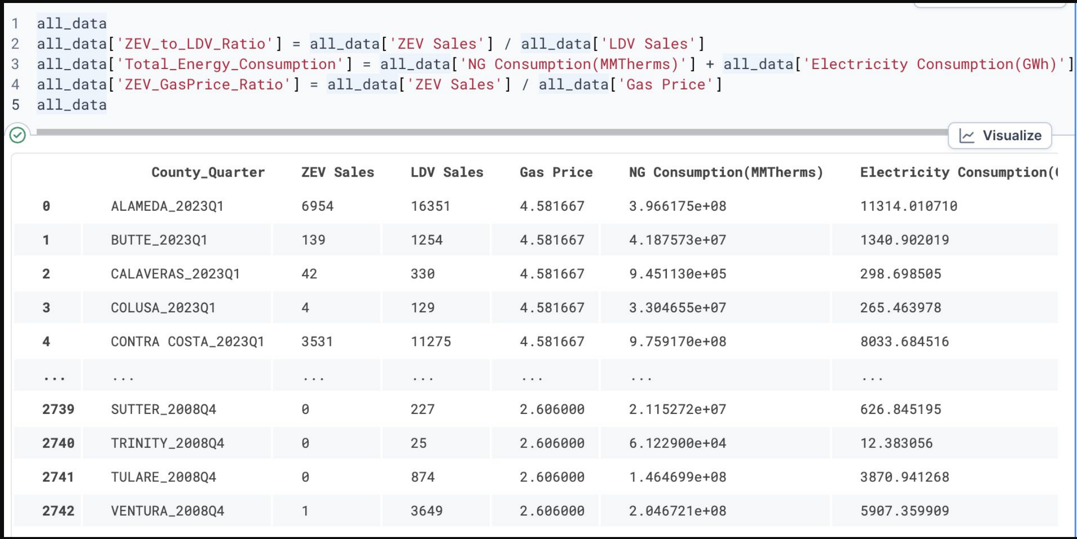
- Averaged GWh data for each quarter and type of sector
- Created Quarter column by multiplying the data by 4 for each county
- Got rid of all data before 2008
- Created County_Quarter Column

Data Cleaning - Gas Price

- Created quarters by averaging monthly data
- Got rid of all data before 2008
- Manufactured county data by adding a column for counties and having each county have the same data
- Renamed the columns to work better
- Created County_Quarter column

Feature Engineering

- Created 3 new columns
  - ZEV_to_LDV Ratio = ZEV sales / LDV sales
  - Total_Energy Consumption = NG Consumption + Electricity Consumption
  - ZEV_GasPrice_Ratio = ZEV Sales/ Gas Price



```
all_data
all_data['ZEV_to_LDV_Ratio'] = all_data['ZEV Sales'] / all_data['LDV Sales']
all_data['Total_Energy_Consumption'] = all_data['NG Consumption(MMTherms)'] + all_data['Electricity Consumption(GWh)']
all_data['ZEV_GasPrice_Ratio'] = all_data['ZEV Sales'] / all_data['Gas Price']
all_data
```

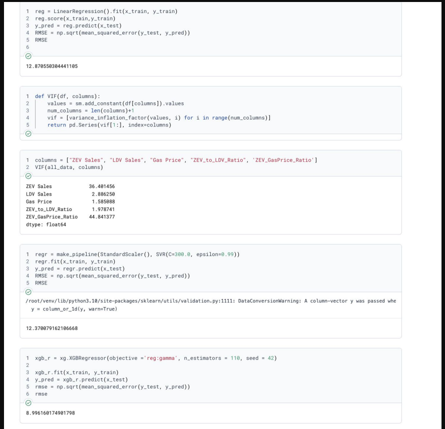
	County_Quarter	ZEV Sales	LDV Sales	Gas Price	NG Consumption(MMTherms)	Electricity Consumption(
0	ALAMEDA_2023Q1	6954	16351	4.581667	3.966175e+08	11314.010710
1	BUTTE_2023Q1	139	1254	4.581667	4.187573e+07	1340.902019
2	CALAVERAS_2023Q1	42	330	4.581667	9.451130e+05	298.698505
3	COLUSA_2023Q1	4	129	4.581667	3.304655e+07	265.463978
4	CONTRA COSTA_2023Q1	3531	11275	4.581667	9.759170e+08	8033.684516
...	...	...	...	...	...	...
2739	SUTTER_2008Q4	0	227	2.606000	2.115272e+07	626.845195
2740	TRINITY_2008Q4	0	25	2.606000	6.122900e+04	12.383056
2741	TULARE_2008Q4	0	874	2.606000	1.464699e+08	3870.941268
2742	VENTURA_2008Q4	1	3649	2.606000	2.046721e+08	5907.359909

Model Selection

Tested many different models

- Linear Regression
- XGB Regression
- Random Forest Regressor
- Ridge Regression
- Lasso regression
- Elastic Regression

Since the XGB regression model provided the lowest RMSE it was the best option for hyperparameter tuning



```
reg = LinearRegression().fit(x_train, y_train)
reg.score(x_train,y_train)
y_pred = reg.predict(x_test)
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
RMSE

```

```
12.870550304441105
```

```
def VIF(df, columns):
    values = sm.add_constant(df[columns]).values
    num_columns = len(columns)+1
    vif = [variance_inflation_factor(values, i) for i in range(num_columns)]
    return pd.Series(vif[1:], index=columns)
```

```
columns = ["ZEV Sales", "LDV Sales", "Gas Price", "ZEV_to_LDV_Ratio", 'ZEV_GasPrice_Ratio']
VIF(all_data, columns)
```

```
ZEV Sales            36.401456
LDV Sales             2.886250
Gas Price             1.585088
ZEV_to_LDV_Ratio      1.978741
ZEV_GasPrice_Ratio   44.841377
dtype: float64
```

```
regr = make_pipeline(StandardScaler(), SVR(C=300.0, epsilon=0.99))
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
RMSE
```

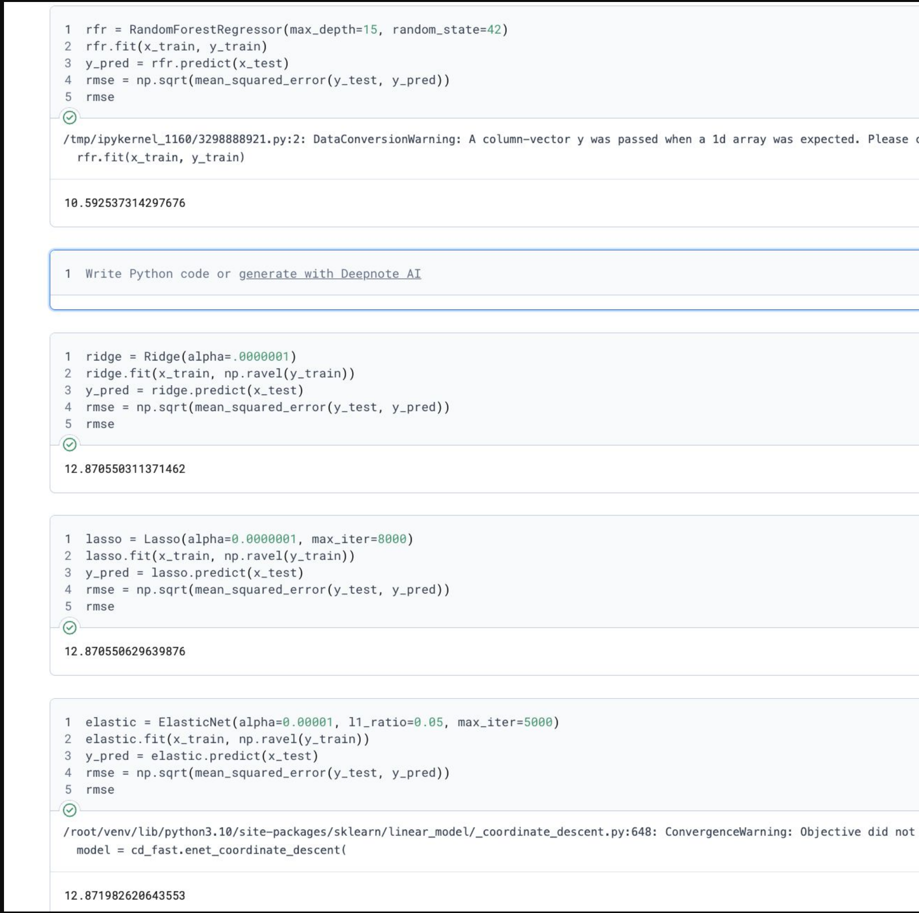
```
/root/venv/lib/python3.10/site-packages/sklearn/utils/validation.py:1111: DataConversionWarning: A column-vector y was passed whe
  y = column_or_1d(y, warn=True)

12.370079162106668
```

```
xgb_r = xg.XGBRegressor(objective ='reg:gamma', n_estimators = 110, seed = 42)

xgb_r.fit(x_train, y_train)
y_pred = xgb_r.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
8.996160174901798
```



```
rfr = RandomForestRegressor(max_depth=15, random_state=42)
rfr.fit(x_train, y_train)
y_pred = rfr.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
/tmp/ipykernel_1160/3298888921.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
  rfr.fit(x_train, y_train)

10.592537314297676
```

```
Write Python code or generate with Deepnote AI
```

```
ridge = Ridge(alpha=.0000001)
ridge.fit(x_train, np.ravel(y_train))
y_pred = ridge.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
12.870550311371462
```

```
lasso = Lasso(alpha=0.0000001, max_iter=8000)
lasso.fit(x_train, np.ravel(y_train))
y_pred = lasso.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
12.870550629639876
```

```
elastic = ElasticNet(alpha=0.00001, l1_ratio=0.05, max_iter=5000)
elastic.fit(x_train, np.ravel(y_train))
y_pred = elastic.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

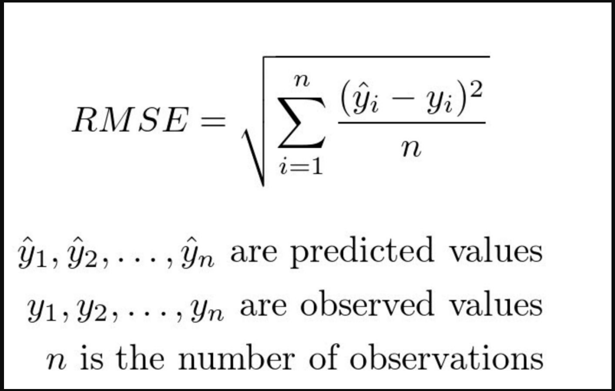
```
/root/venv/lib/python3.10/site-packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning: Objective did not
  model = cd_fast.enet_coordinate_descent(

12.871982620643553
```

RMSE

Our metric for evaluating each model’s performance was it’s RMSE.
RMSE, or Root Mean Squared Error, measures how far a model’s predictions are from the actual values on average.
The RMSE is calculated by finding the mean error between our predicted values and the observed, squaring them to eliminate negative errors, and the square rooting it so that the units are the same as our observed values.

RMSE Formula →



$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values

$y_1, y_2, \ldots, y_n$ are observed values

n is the number of observations

Linear Regression Model

The first model tested for this project was a Linear Regression Model. A linear regression model is a method used to show the relationship between one or more input variables and an output variable. Our RMSE for the linear regression model was 12.87.

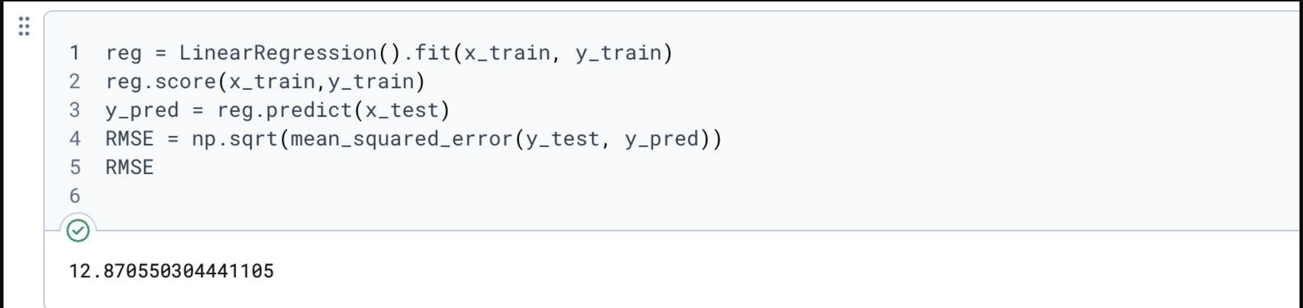
In multiple linear regression, there is one dependent variable and several independent variables. The equation becomes:

$Y = \beta 0 * X0 + \beta 1X1 + \beta 2X2 + \ldots + \beta nXn + \varepsilon$

When you add more predictors, the number of β coefficients increases to match.

This method looks at how several factors together affect an outcome. In our case we are using:

- ZEV Sales, Gas Prices, Electricity Consumption, and NG Consumption on air quality data



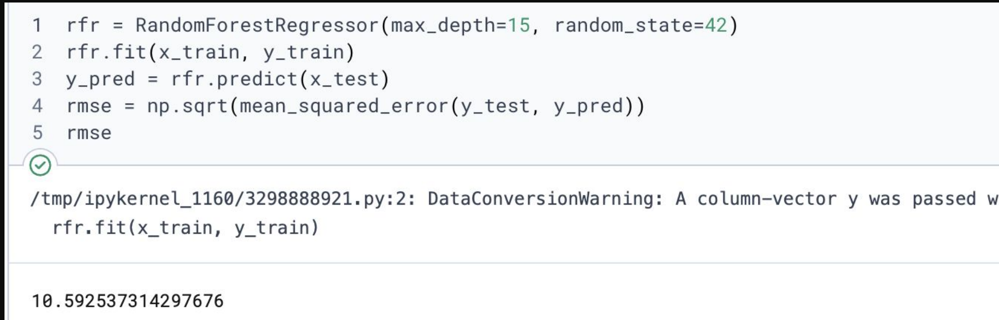
```
reg = LinearRegression().fit(x_train, y_train)
reg.score(x_train,y_train)
y_pred = reg.predict(x_test)
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
RMSE

```

```
12.870550304441105
```

Random Forest Regression

The third model tested was a Random Forest regression model. This model is based off of a linear regression model. A Random Forest regression model, unlike linear regression, uses many decision trees to capture complex non-linear patterns. Random Forest regression is a machine learning method that uses many decision trees to make predictions. Each tree is built from a random sample of the data, and the final prediction is the average of all the trees' outputs. This approach reduces overfitting and improves accuracy compared to using a single decision tree. The ending RMSE was 10.59



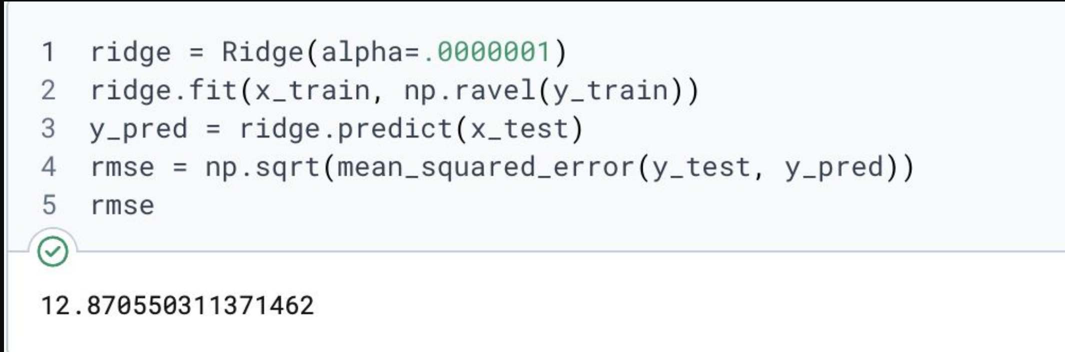
```
rfr = RandomForestRegressor(max_depth=15, random_state=42)
rfr.fit(x_train, y_train)
y_pred = rfr.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
/tmp/ipykernel_1160/3298888921.py:2: DataConversionWarning: A column-vector y was passed w
  rfr.fit(x_train, y_train)

10.592537314297676
```

Ridge Regression

The fourth model tested was a Ridge regression model. This model is based off of a linear regression model. Ridge regression, unlike standard linear regression, adds a penalty to large coefficients to prevent overfitting Random Forest regression is a machine learning method that uses many decision trees to make predictions.
Each tree is built from a random sample of the data, and the final prediction is the average of all the trees' outputs. This approach reduces overfitting and improves accuracy compared to using a single decision tree. The model's ending RMSE was a 12.87.

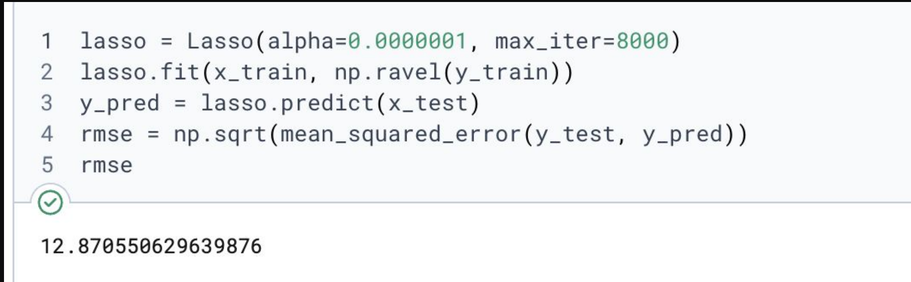


```
ridge = Ridge(alpha=.0000001)
ridge.fit(x_train, np.ravel(y_train))
y_pred = ridge.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
12.870550311371462
```

Lasso Regression

The fifth model tested was a Lasso regression model. This model is based off of a linear regression model. Lasso regression, unlike standard linear regression, adds a penalty that can shrink some coefficients to zero for simpler models Lasso regression is a type of linear regression that adds a penalty to the model's coefficients.
This penalty can reduce some coefficients to exactly zero, which removes less important variables from the model.
It is useful for both improving accuracy and selecting the most relevant predictors. The ending RMSE for the model was 12.87



```python
lasso = Lasso(alpha=0.0000001, max_iter=8000)
lasso.fit(x_train, np.ravel(y_train))
y_pred = lasso.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
12.870550629639876
```

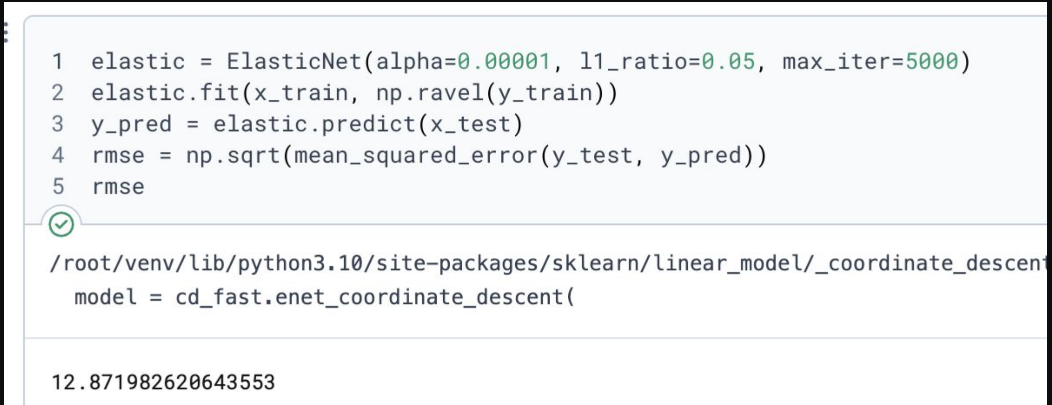
Elastic Regression

The elastic model tested was a Lasso regression model. Elastic Net regression, unlike standard linear regression, applies penalties that can shrink coefficients and remove less important variables.

Elastic Net regression is a type of linear regression that combines the penalties used in Ridge and Lasso regression.

It can shrink coefficients to reduce overfitting and set some to zero to remove less important variables.

This makes it useful when predictors are correlated or when variable selection is needed.



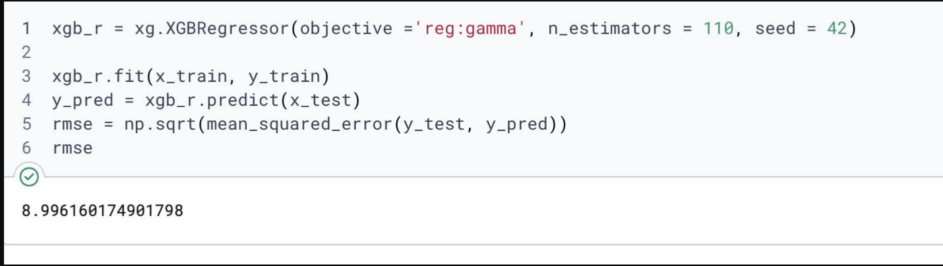
```python
elastic = ElasticNet(alpha=0.00001, l1_ratio=0.05, max_iter=5000)
elastic.fit(x_train, np.ravel(y_train))
y_pred = elastic.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
/root/venv/lib/python3.10/site-packages/sklearn/linear_model/_coordinate_descen
  model = cd_fast.enet_coordinate_descent(
```

```
12.871982620643553
```

XGB Regression

The second model tested was a XGB regression model. This model is based off of a linear regression model. An XGB regression model, unlike linear regression, can capture complex non-linear patterns, but both models use input variables to predict an output and measure accuracy against actual values. For this project the model's RMSE was 8.99.
An XGB regression model is a machine learning method based on decision trees.
It builds many small trees one after another, with each new tree fixing the errors made by the previous ones.
This process, called boosting, helps the model learn patterns in the data more accurately.
XGB is known for being fast and effective, especially with large or complex datasets.



```
xgb_r = xg.XGBRegressor(objective ='reg:gamma', n_estimators = 110, seed = 42)

xgb_r.fit(x_train, y_train)
y_pred = xgb_r.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
rmse
```

```
8.996160174901798
```

Hyperparameter Tuning

Hyperparameter tuning is the process of finding the best settings for a machine learning model before training it. These settings, called hyperparameters, control how the model learns, such as the learning rate or the number of trees. Choosing the right values helps the model perform better and make more accurate predictions.

These are the hyperparameters we tested for the XGB Model.

```
 "n_estimators": [300, 600, 900],
  "max_depth": [4, 6, 8],
  "learning_rate": [0.02, 0.04, 0.08],
  "subsample": [0.7, 0.85, 1.0],
  "colsample_bytree": [0.7, 0.85, 1.0],
  "reg_lambda": [0.5, 1, 5],
  "min_child_weight": [1, 3, 5],
```

- Explain what each of the hyperparameters do (for the XGB model)
- Explain why we used these specific ranges

What Each Hyperparameters do

n_estimators – Number of boosting rounds (trees) built by the model. More trees can improve accuracy but increase training time and risk of overfitting.

max_depth – Maximum depth of each tree. Deeper trees capture more complex patterns but can overfit.

learning_rate – How much each tree's contribution is scaled down. Lower values improve stability but require more trees.

subsample – Fraction of training data sampled for each tree. Lower values add randomness and reduce overfitting.

colsample_bytree – Fraction of features used for each tree. Lower values help prevent overfitting by reducing reliance on any one feature.

reg_lambda – L2 regularization strength. Higher values make the model simpler and reduce overfitting.

min_child_weight – Minimum sum of instance weights needed in a leaf. Higher values prevent very specific, small splits.

What is L2 Regularization

L2 regularization is a way to stop a model from overfitting. It adds a penalty when the model’s weights get too large.This penalty makes the weights smaller, but not zero, which keeps all features in the model. The strength of this effect is set by a value called lambda A bigger lambda makes the model simpler, a smaller lambda makes it more flexible.

Conclusion - What could have been done better

- I would spend time finding a model that could achieve a better RMSE. With the model used and the data used in this project the RMSE was the lowest it could have been but spending more time choosing a better model that would fit the project and data better could have helped slightly lower the RMSE.

- I would find better data to help lower the RMSE. Most of the data used was duplicated and slightly altered data as most of the data sets did not fit the quarter and county system from 2008 - 2024

- Most of the data we used did not have specific quarterly or county info. For example, the electricity consumption data provided electricity consumption amounts for counties in California on a yearly basis. This means that when I had to turn it into quarterly data, each year's data would have to be duplicated 3 more times to fit into each quarter of each year. If I had more time/ resources for this project, I would find more granular electricity consumption data that provided electricity consumption on a quarterly basis.

- Figure out a way to web-scrape data for sites that didn't allow to download data

Conclusion - Limitations

The main limitation of the project was:

- There was very limited data that was available for use as a data set on the internet
    - If each of the data sets had data for each quarter in every county in California from 2008 - 2024 then the ending RMSE could have possibly been much lower.
    - There was also some data that was on the internet but not accessible as a dataset like the California gas price data per county was not a data set but there where chunks of the data on the internet. But at the end because it wasn't a data set we were forced to use data that only contained all of California's average gas price data instead of each County