Supervised Learning - Predicting Monthly Energy Prices By State

November 9, 2020

1 Predicting Retail Electricity Prices Using Fossil Fuel Prices

1.1 Supervised Machine Learning Approaches

1.1.1 Monthly Data by State, Energy Type, and Sector from January 2008 to July 2020

1.2 1. Introduction

I will be predicting average retail electricity prices throughout the United States for nearly twelve (12) years by month and state based upon fossil fuel prices and generation volumes by energy type. The power generation industry is a very long-term industry with most company revenue contracts and fixed asset investment horizons ranging from ten (10) to thirty (30) years long. When making such large, long-term investments and commitments, power generators expose themselves to significant known and unknown business risks. Today, generators are exposed more and more to fluctuating energy market prices with the continued establishment of idependent system operators (ISOs). Being able to forecast energy prices accurately will assist with strategic positioning (i.e. if this nuclear plant shuts down, what will power prices do for 1 month, 3 months, or 6 months) and risk management by calibrating hedging and swap portfolios.

1. Assuming trends in fossil fuel production and pricing markets, can we reasonably predict retail electricity prices by month and state? Being able to predict changes in electricity prices relative to changes in fossil fuel markets improves strategic visibility for regional acquisitions and could improve risk management by quantifying incoming market impacts from upstream events and improving calibration of hedging portfolios.

2. Can retail power prices be reasonably determined based only on the cost inputs of fossil fuels and not any renewable power or nuclear generation costs? If the modeled predictions are reasonably accurate, we can observe market spark margin efficiencies and conclude renewable power companies need to guide their merchant power price strategy based on expected long-term fossil-fuel markets.

1.3 2. Data

The dataset has been downloaded from the Energy Information Administration (EIA) website at https://www.eia.gov/electricity/data/browser/, or https://www.eia.gov/opendata/qb.php?category=40. The data is somewhat comprehensive including generation statistics, average fuel costs, fuel deliveries, fuel consumption, and fuel inventories. For this project I used these feature variable to predict the average retail electricity prices in the United States on a monthly basis from 2001 to July 2020. The electricity price data is broken down by region, state, and sector, and has been downloaded into a usable .csv file in the local folder.

Within the electricity prices report, there are 7,701 observations and twenty-two (22) variables including location, sequential year, month number, sector, the target electricity price variable, the average cost of fossil fuels in electricity generation, net generation, fossil fuel stocks, and fuel consumption volumes by electricity generation. There are toggles so that the project can be run with state-level data or region-level data by location, but not both as to avoid the effects of overlapping and cocorrelation from variables which may overlap.

- **Energy Types:** Coal, Natural Gas, Petroleum Coke, Petroleum Liquids, Renewable and Other Forms, and General Electricity (the electricity prices are not broken down by energy type, but only location and sector really).
- **Sectors:** Commercial, Industrial, Residential, Electric Utility, Independent Power Producers

[1]:
```
# # Import Python libraries.
from custom_functions import (
    short_to_long_form_and_wrangle,
    correct_section_classification,
    correct_energy_units,
    show_category_breakdown,
    show_agg_stats_by_category,
    long_form_to_xdate,
    null_table_graph,
    back_forward_fill,
    drop_empty_multi_idx_cols,
    plot_discrete_features_by_col_idx,
    times_series_to_long_form,
    get_model_df,
    adjust_MMBtu_units,
    add_chrono_features,
    show_distributions,
    transform_check_distributions,
    discrete_feature_effects,
    split_training_test_data,
    show_correlations,
    generate_linear_regression,
    # ----------------
    pd,
    np,
    plt,
    sns,
    df_img_out,
    RandomForestRegressor,
    RandomizedSearchCV,
    GridSearchCV,
    SVR,
```

```
    GradientBoostingRegressor,
    RidgeCV,
    LassoCV,
    ElasticNetCV,
    cross_val_score,
    mean_absolute_error,
    mse,
    rmse
)
%matplotlib inline
```

2 Import Raw Data and View Row/Column Schema

[2]:
```
# Data obtained from EIA at https://www.eia.gov/electricity/data/browser/,␣
↪'View a pre-generated report'.
eia_raw_data_df = pd.read_csv('eia_data.csv').drop(columns=['index'])
df_img_out(eia_raw_data_df.head(3).iloc[:, :7].describe(), 'eia_raw_data_df')
target_var = 'Average_retail_price_of_electricity_cents_per_kilowatthour'
```

	Jan 2008	Feb 2008	Mar 2008
count	3.000	3.000	3.000
mean	1.867	1.877	1.920
std	0.006	0.012	0.026
min	1.860	1.870	1.900
25%	1.865	1.870	1.905
50%	1.870	1.870	1.910
75%	1.870	1.880	1.930
max	1.870	1.890	1.950

3 Convert and Wrangle Data to Long-long Format

[3]:
```
folder_df_nz_long = short_to_long_form_and_wrangle(eia_raw_data_df,␣
↪loc_method='state')
df_img_out(folder_df_nz_long.head(3).iloc[:, :7], 'folder_df_nz_long')
```

index	variable	energy_type	location	sector	source key	year_month	values
12432	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-01-01	nan
12580	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-02-01	nan
12728	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-03-01	nan

4 Correct 'sector' Bad Data in Receipts of Fossil Fuels Variables

[4]:
```
folder_df_nz_long_sct = correct_section_classification(folder_df_nz_long)
df_img_out(folder_df_nz_long.head(3).iloc[:, :7], 'folder_df_nz_long_sct')
```

index	variable	energy_type	location	sector	source key	year_month	values
12432	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-01-01	nan
12580	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-02-01	nan
12728	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	Alabama	all sectors	ELEC.COST_BTU.COW-AL-98.M	2008-03-01	nan

[5]:
```
folder_df_nz_long_sct_conv = correct_energy_units(folder_df_nz_long_sct)
```

5 Data Structure: Categroical Variable Values (Location, Energy Type, and Industry Sector)

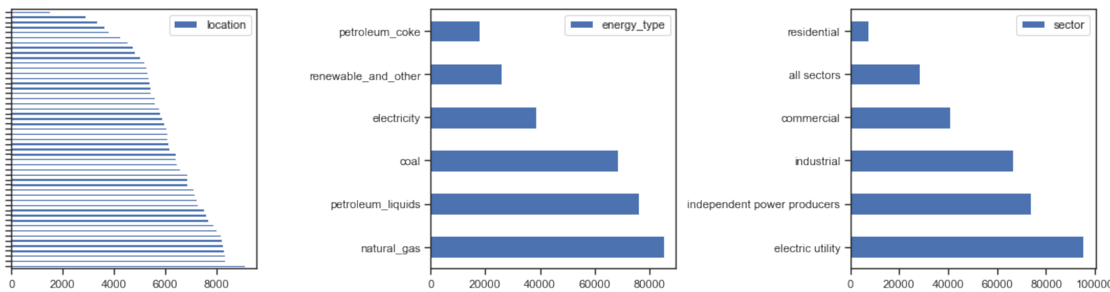
We can see that petroleum coke 'energy__type' and residential 'sector' have very limited data points, however, petroleum coke was not recorded as much throughout various states nor was it recorded over a similarly long period of time. Petroleum coke comprises a significant portion of the energy markets and is a key source of cost-effective energy value for electricity generation because it is a bi-product (not a primary output) of the petroleum refining process.

[6]:
```
show_category_breakdown(folder_df_nz_long_sct_conv, 'variable', ['location',␣
↪'energy_type', 'sector'])
```

	0	1	2	3	4	5	45	6
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	Alabama	Alaska	Arizona	Arkansas	California	Colorado	Connecticut	Delaware
Average_retail_price_of_electricity_cents_per_kilowatthour	Alabama	Alaska	Arizona	Arkansas	California	Colorado	Vermont	Connecticut
Consumption_by_electricity_generation_MMBtu	Alabama	Alaska	Arizona	Arkansas	California	Colorado	West Virginia	Connecticut
Fossil_fuel_stocks_in_electricity_generation_MMBtu	Alabama	Alaska	Arizona	Arkansas	California	Colorado	Wisconsin	Connecticut
Net_generation_thousand_megawatthours	Alabama	Alaska	Arizona	Arkansas	California	Colorado	West Virginia	Connecticut

	0	1	2	3	4
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	coal	natural_gas	petroleum_coke	petroleum_liquids	NaN
Consumption_by_electricity_generation_MMBtu	coal	natural_gas	petroleum_coke	petroleum_liquids	NaN
Net_generation_thousand_megawatthours	coal	natural_gas	petroleum_coke	petroleum_liquids	renewable_and_other
Receipts_of_fossil_fuels_electricity_generation_MMBtu	coal	natural_gas	petroleum_coke	petroleum_liquids	NaN
Fossil_fuel_stocks_in_electricity_generation_MMBtu	coal	petroleum_coke	petroleum_liquids	NaN	NaN

	0	1	3	2
Average_retail_price_of_electricity_cents_per_kilowatthour	all sectors	NaN	NaN	NaN
Fossil_fuel_stocks_in_electricity_generation_MMBtu	all sectors	NaN	NaN	NaN
Retail_sales_of_electricity_million_kilowatthours	all sectors	NaN	NaN	NaN
Revenue_from_retail_sales_of_electricity_million_dollars	commercial	industrial	NaN	residential
Consumption_by_electricity_generation_MMBtu	electric utility	independent power producers	commercial	industrial



6 Data Structure: Categorical Group Statistics and Value Ranges

[7]:
```
show_agg_stats_by_category(folder_df_nz_long_sct_conv, ['energy_type',␣
↪'sector'],
                                  ['min', 'mean', 'max', 'count', 'sum'])
```

variable	energy_type	values min	values mean	values max	values count	values sum
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	**coal**	0.440	2.368	18.230	6438	15,248.170
	natural_gas	0.460	4.788	59.320	8920	42,713.130
	petroleum_coke	0.300	1.933	15.300	813	1,571.390
	petroleum_liquids	3.060	17.129	53.310	5813	99,572.800
Average_retail_price_of_electricity_cents_per_kilowatthour	**electricity**	5.270	10.645	36.370	7701	81,977.760
Consumption_by_electricity_generation_MMBtu	**coal**	18.875	12,718.070	128,746.375	14724	187,260,856.875
	natural_gas	0.001	4.952	149.136	24106	119,375.046
	petroleum_coke	24.800	561.500	4,786.400	2349	1,318,963.200
	petroleum_liquids	5.698	175.982	12,581.184	12048	2,120,225.800
Fossil_fuel_stocks_in_electricity_generation_MMBtu	**coal**	18.875	83,253.437	376,273.125	4940	411,271,980.500
	petroleum_coke	24.800	2,035.324	12,028.000	187	380,605.600
	petroleum_liquids	5.698	3,825.504	52,940.118	7043	26,943,021.490
Net_generation_thousand_megawatthours	**coal**	-5.000	1,196.537	11,075.000	15315	18,324,970.000
	natural_gas	-93.000	666.761	19,682.000	22910	15,275,501.000
	petroleum_coke	-5.000	55.975	509.000	2529	141,560.000
	petroleum_liquids	-18.000	20.472	1,267.000	10620	217,413.000
	renewable_and_other	-31.000	859.097	23,440.000	24260	20,841,695.000
Receipts_of_fossil_fuels_electricity_generation_MMBtu	**coal**	18.875	14,534.364	117,987.625	12951	188,234,542.625
	natural_gas	0.001	7.251	158.772	16827	122,006.639
	petroleum_coke	24.800	921.032	6,026.400	1539	1,417,468.800
	petroleum_liquids	5.698	224.917	15,162.378	8861	1,992,989.460
Retail_sales_of_electricity_million_kilowatthours	**electricity**	389.000	6,100.060	43,866.000	7701	46,976,564.000
Revenue_from_retail_sales_of_electricity_million_dollars	**electricity**	1.000	206.722	2,427.000	23089	4,772,999.000

variable	sector	values min	mean	max	count	sum
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	electric utility	0.460	7.484	59.320	17463	130,696.540
	independent power producers	0.300	6.284	35.880	4521	28,408.950
Average_retail_price_of_electricity_cents_per_kilowatthour	all sectors	5.270	10.645	36.370	7701	81,977.760
Consumption_by_electricity_generation_MMBtu	commercial	0.001	10.514	679.500	6637	69,780.118
	electric utility	0.001	7,430.266	101,962.750	18907	140,484,035.063
	independent power producers	0.001	3,466.603	128,746.375	14162	49,094,031.814
	industrial	0.001	86.648	6,511.875	13521	1,171,573.927
Fossil_fuel_stocks_in_electricity_generation_MMBtu	all sectors	5.698	36,039.080	376,273.125	12170	438,595,607.590
Net_generation_thousand_megawatthours	commercial	-6.000	17.175	207.000	10225	175,618.000
	electric utility	-31.000	1,196.175	18,238.000	25111	30,037,157.000
	independent power producers	-5.000	1,063.926	23,440.000	21080	22,427,552.000
	industrial	-93.000	112.437	4,023.000	19218	2,160,812.000
Receipts_of_fossil_fuels_electricity_generation_MMBtu	commercial	0.001	96.661	1,736.500	1370	132,425.826
	electric utility	0.001	8,104.982	101,415.375	17197	139,381,369.121
	independent power producers	0.001	3,750.793	117,987.625	12994	48,737,810.165
	industrial	0.001	407.961	5,907.875	8617	3,515,402.412
Retail_sales_of_electricity_million_kilowatthours	all sectors	389.000	6,100.060	43,866.000	7701	46,976,564.000
Revenue_from_retail_sales_of_electricity_million_dollars	commercial	20.000	228.322	2,323.000	7701	1,758,311.000
	industrial	1.000	109.343	967.000	7699	841,834.000
	residential	12.000	282.593	2,427.000	7689	2,172,854.000

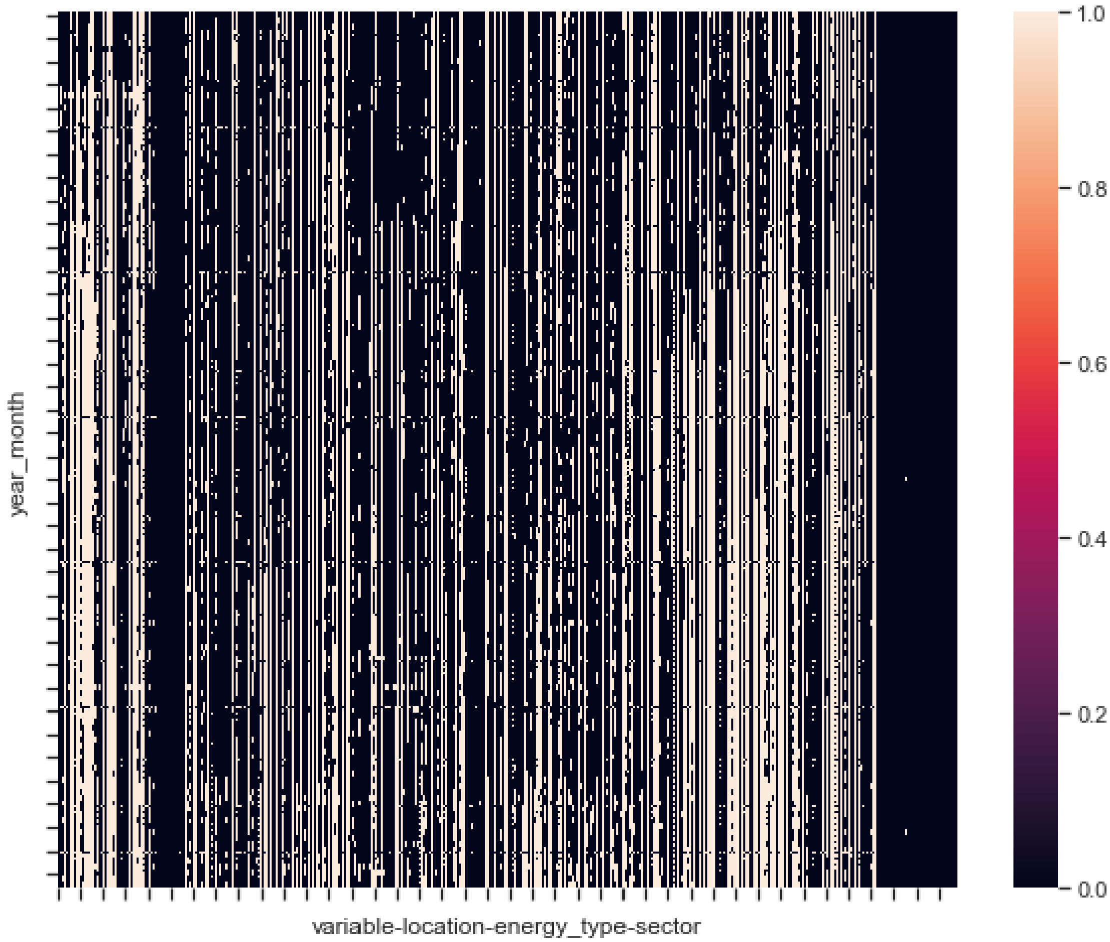
7 Wrangle Data to Chronological Time-Series Format with 'year_month' as the X-axis.

[8]:
```
folder_df_nz_xdate = long_form_to_xdate(folder_df_nz_long_sct_conv)
df_img_out(folder_df_nz_xdate.head(3).iloc[:, :7], 'folder_df_nz_xdate')
```

variable	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu						
location	Alabama					Alaska	
energy_type	coal		natural_gas		petroleum_liquids	coal	natural_gas
sector	electric utility	independent power producers	electric utility	independent power producers	electric utility	electric utility	electric utility
year_month							
2008-01-01	2.150	nan	8.240	8.810	18.210	1.340	4.050
2008-02-01	2.120	nan	8.810	10.730	20.020	1.310	4.070
2008-03-01	2.190	nan	9.770	nan	21.100	1.340	4.130

8 Data Cleaning: Chronologically Forward-fill and Back-fill variable data after breaking it down by variable, location, and energy type.

[9]:
```
null_table_graph(folder_df_nz_xdate.replace([0, np.inf, -np.inf], np.nan))
df_img_out(folder_df_nz_xdate.head(3).iloc[:, :7], 'folder_df_nz_xdate2')
```

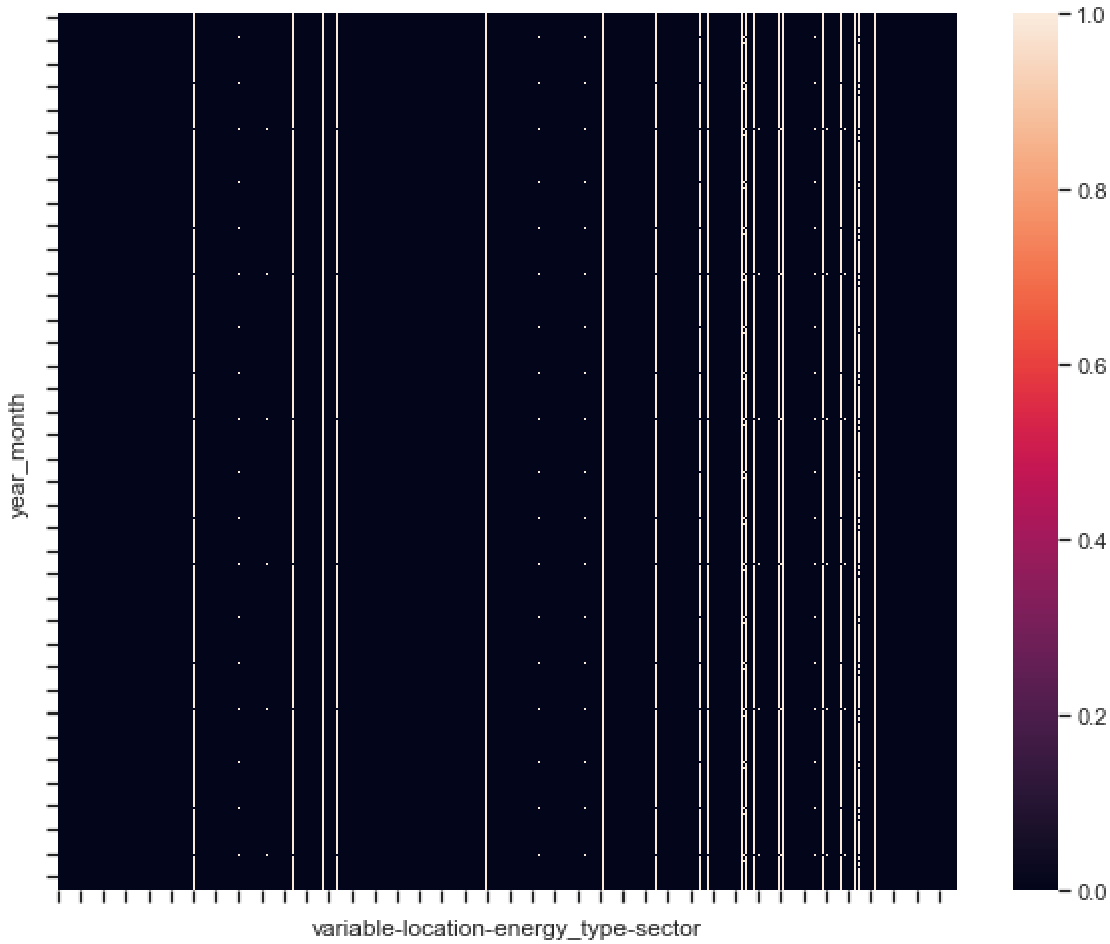


variable	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu						
location	Alabama					Alaska	
energy_type	coal		natural_gas		petroleum_liquids	coal	natural_gas
sector	electric utility	independent power producers	electric utility	independent power producers	electric utility	electric utility	electric utility
year_month							
2008-01-01	2.150	nan	8.240	8.810	18.210	1.340	4.050
2008-02-01	2.120	nan	8.810	10.730	20.020	1.310	4.070
2008-03-01	2.190	nan	9.770	nan	21.100	1.340	4.130

[10]:
```
folder_df_nz_xdate_fill = back_forward_fill(folder_df_nz_xdate)
print("\nShowing the difference in .describe() statistics after applying the␣
↪fill assumptions.\n")
df_img_out(folder_df_nz_xdate_fill.head(3).iloc[:, :7],␣
↪'folder_df_nz_xdate_fill')
null_table_graph(folder_df_nz_xdate_fill)
```

```
Showing the difference in .describe() statistics after applying the fill
assumptions.
```

variable	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu						
location	Alabama					Alaska	
energy_type	coal		natural_gas		petroleum_liquids	coal	natural_gas
sector	electric utility	independent power producers	electric utility	independent power producers	electric utility	electric utility	electric utility
year_month							
2008-01-01	2.150	3.000	8.240	8.810	18.210	1.340	4.050
2008-02-01	2.120	3.000	8.810	10.730	20.020	1.310	4.070
2008-03-01	2.190	3.000	9.770	13.150	21.100	1.340	4.130



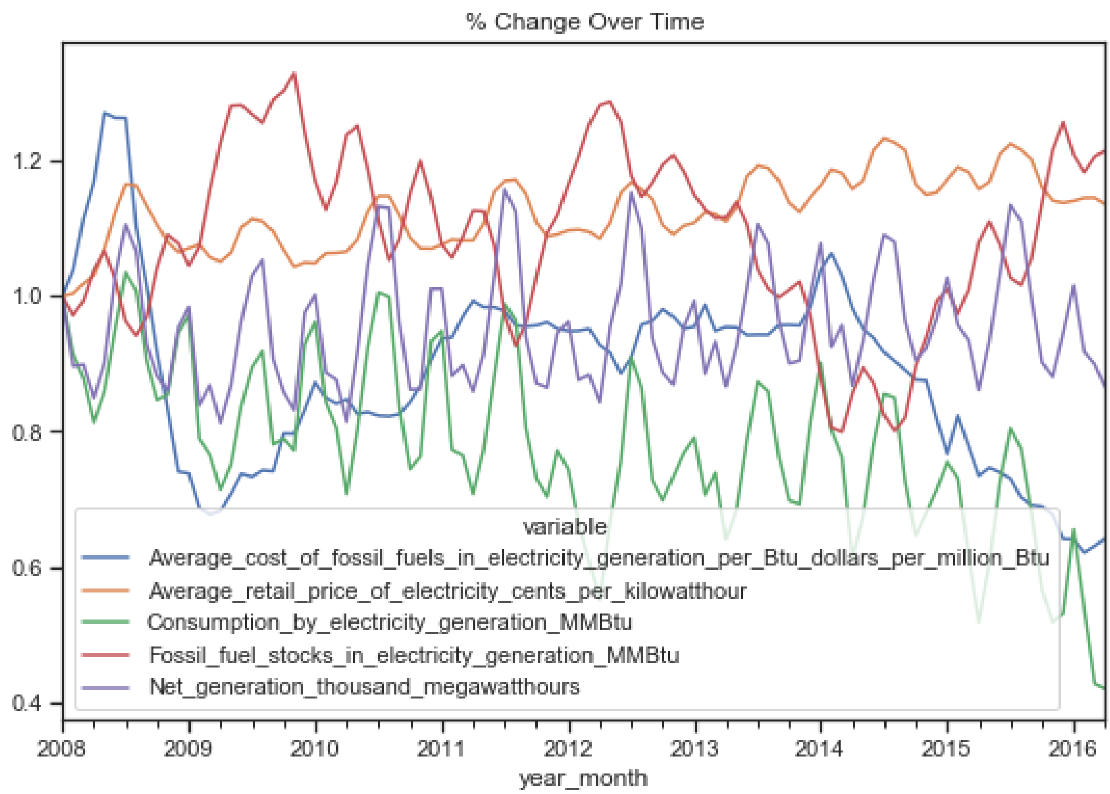
[11]:
```python
folder_df_nz_xdate_fill_drop =␣
 ↪drop_empty_multi_idx_cols(folder_df_nz_xdate_fill)
col_count_chg = folder_df_nz_xdate_fill_drop.columns.size -␣
 ↪folder_df_nz_xdate_fill.columns.size
print("Change in count of columns after dropping: {:,.0f} ({:,.0f} remaining␣
 ↪columns)".format(
    col_count_chg, folder_df_nz_xdate_fill_drop.columns.size))
df_img_out(folder_df_nz_xdate_fill_drop.head(3).iloc[:, :10],␣
 ↪'folder_df_nz_xdate_fill_drop')
```

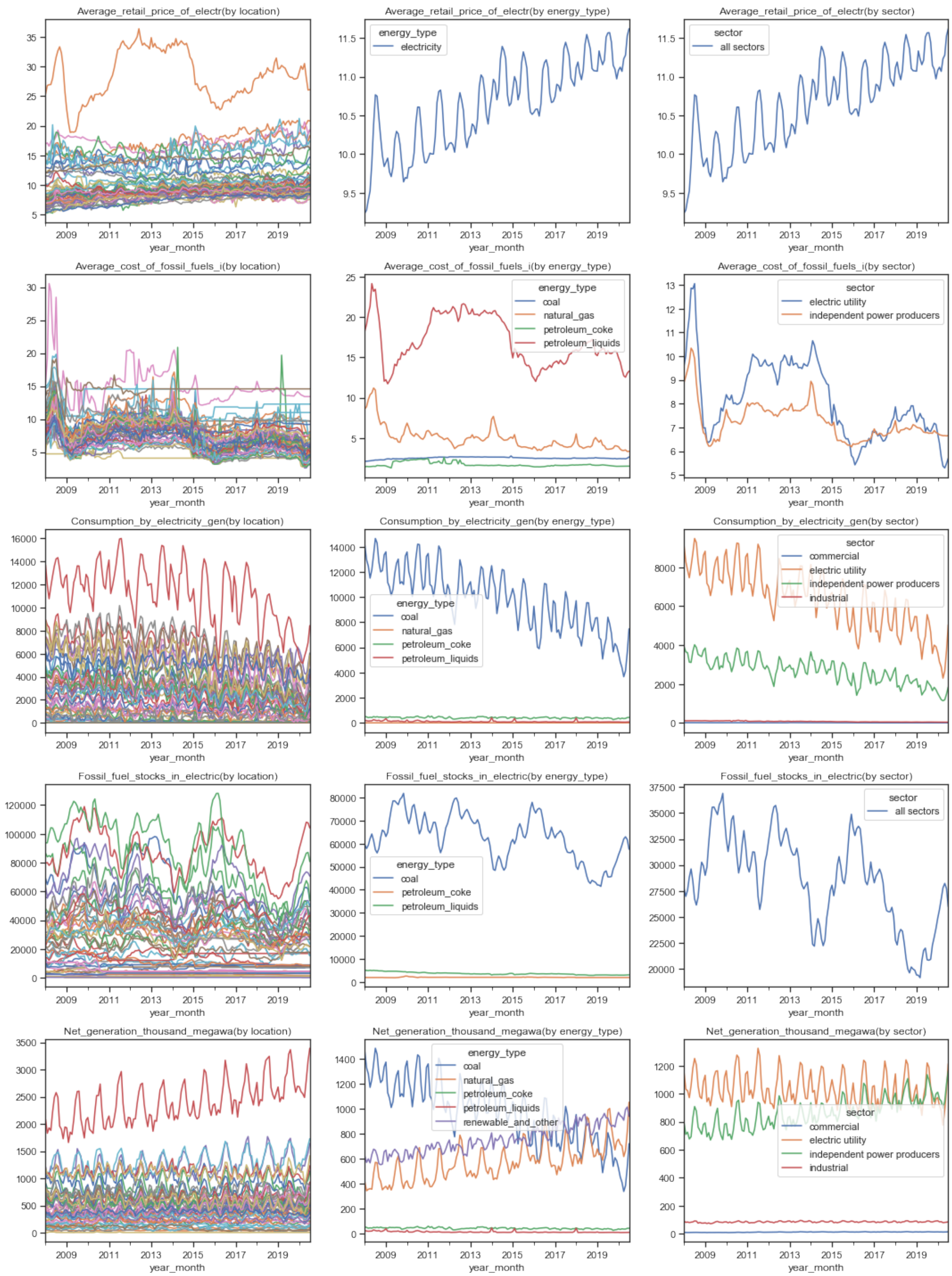
```
Change in count of columns after dropping: -99 (2,172 remaining columns)
```

variable	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu									
location	Alabama					Alaska			Arizona	
energy_type	coal		natural_gas		petroleum_liquids	coal	natural_gas	petroleum_liquids	coal	natural_gas
sector	electric utility	independent power producers	electric utility	independent power producers	electric utility	electric utility	electric utility	electric utility	electric utility	electric utility
year_month										
2008-01-01	2.150	3.000	8.240	8.810	18.210	1.340	4.050	19.700	1.620	8.010
2008-02-01	2.120	3.000	8.810	10.730	20.020	1.310	4.070	20.550	1.690	8.620
2008-03-01	2.190	3.000	9.770	13.150	21.100	1.340	4.130	22.680	1.720	9.520

9 Chronological Trends in Raw Variable Values: Fossil Fuel Costs, Consumption, Inventory Stocks

[12]:
```
plot_discrete_features_by_col_idx(folder_df_nz_xdate_fill_drop, target_var.
 ↪split('__')[0])
```





- We can see that the average retail price of electricity tends to move seasonally throughout the year by the monthly data. If we had daily data, we would see the variation between on-peak and off-peak hours, but that is beyond the scope of this project.

- The gradual increase in electricity prices generally tend to appear inflationary and track coal prices, as coal is such a significant portion of the overall power grid. Natural gas, petroleum coke, and petroleum liquids fuel prices have trended downward over the past five years as the natural gas glut from fracking continues to maintain negative pricing pressure across the markets.
- It is interesting that the average cost of fuel between utilities and independent power generators generally shows that private companies are more efficient at maintaining low costs in their operational processes and procurement by a noticeable margin when compared to quasi-governmental utility generators which are heavily regulated.
- Looking at consumption of fuels for electricity, utilities are a much larger portion of the market than independent power producers but still cannot garner pricing economies of scale in fuel costs comparatively.
- Independent power generators have been steadily increasing their portion of total generation over the past decade and are now at a similar production level than utility companies.
- Coal generation has been decreasing to about half of its level a decade ago, while being replaced with natural gas and renewable energy sources.
- Receipts of all types of fossil fuels have been declining consistently, except for natural gas.
- Fossil fuel stocks of all types reaches a very low level in 2019.

[13]:
```
df_fill_long = times_series_to_long_form(folder_df_nz_xdate_fill_drop)
df_img_out(df_fill_long.head(3).iloc[:, :10], 'df_fill_long')
```

	variable	location	energy_type	sector	year_month	values
0	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	Alabama	coal	electric utility	2008-01-01	2.150
1	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	Alabama	coal	electric utility	2008-02-01	2.120
2	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu	Alabama	coal	electric utility	2008-03-01	2.190

10 Data Wrangling: Make Data Ready for Machine Learning Models

In order to homogenize the data columns more, I converted each of the fossil-fuel-related variables from either tons, metric cubic feet, or gallons of fossil fuels into British Thermal Units (BTUs) so the figures the machine learning model is comparing are of the same units across various energy types of generation. We can also see a more accurate, relevant, and sometimes very different pricing perspective when comparing fossil fuel costs by energy content rather than weight or volume.

[30]:
```
# Get DataFrame format ready for data modeling.
model_df = get_model_df(df_fill_long)
model_df = model_df.reset_index().drop(columns='location')
df_img_out(model_df.iloc[:3, :2], 'model_df')
df_img_out(model_df.iloc[:5, :20].transpose(), 'model_df')
```

	year_month	Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__coal
0	2008-01-01	2.575
1	2008-02-01	2.560
2	2008-03-01	2.595

	0	1	2	3	4
year_month	2008-01-01 00:00:00	2008-02-01 00:00:00	2008-03-01 00:00:00	2008-04-01 00:00:00	2008-05-01 00:00:00
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__coal	2.575	2.560	2.595	2.595	2.705
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__natural_gas	8.525	9.770	11.460	11.645	12.895
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__petroleum_liquids	18.210	20.020	21.100	25.560	26.830
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__petroleum_coke	1.640	1.640	1.640	1.640	1.640
Average_retail_price_of_electricity_cents_per_kilowatthour__electricity	7.680	7.490	7.220	7.490	7.880
Consumption_by_electricity_generation_MMBtu__coal	18,510.083	16,641.458	18,503.792	18,428.292	20,347.250
Consumption_by_electricity_generation_MMBtu__natural_gas	5.789	3.612	3.212	2.247	1.914
Consumption_by_electricity_generation_MMBtu__petroleum_liquids	123.457	34.188	32.289	37.987	43.685
Consumption_by_electricity_generation_MMBtu__petroleum_coke	756.400	756.400	756.400	756.400	756.400
Fossil_fuel_stocks_in_electricity_generation_MMBtu__coal	85,937.875	90,165.875	89,580.750	87,900.875	82,615.875
Fossil_fuel_stocks_in_electricity_generation_MMBtu__petroleum_liquids	1,908.830	1,886.038	1,863.246	1,829.058	1,720.796
Fossil_fuel_stocks_in_electricity_generation_MMBtu__petroleum_coke	967.200	967.200	967.200	967.200	967.200
Net_generation_thousand_megawatthours__coal	2,101.667	1,889.000	2,088.333	2,040.333	2,222.667
Net_generation_thousand_megawatthours__natural_gas	703.333	475.667	392.000	285.000	243.333
Net_generation_thousand_megawatthours__petroleum_liquids	11.667	4.333	6.333	8.000	6.667
Net_generation_thousand_megawatthours__renewable_and_other	1,466.667	1,613.667	1,614.667	1,286.667	1,411.667
Net_generation_thousand_megawatthours__petroleum_coke	71.500	71.500	71.500	71.500	71.500
Receipts_of_fossil_fuels_electricity_generation_MMBtu__coal	19,290.250	17,811.708	18,598.167	20,271.750	18,138.875
Receipts_of_fossil_fuels_electricity_generation_MMBtu__natural_gas	6.305	4.011	3.616	2.602	2.257

11 Adjust MMBtu Variable Units to Decrease Relative Magnitude Versus Other Variable Values

[16]:
```
model_df2 = adjust_MMBtu_units(model_df)
```

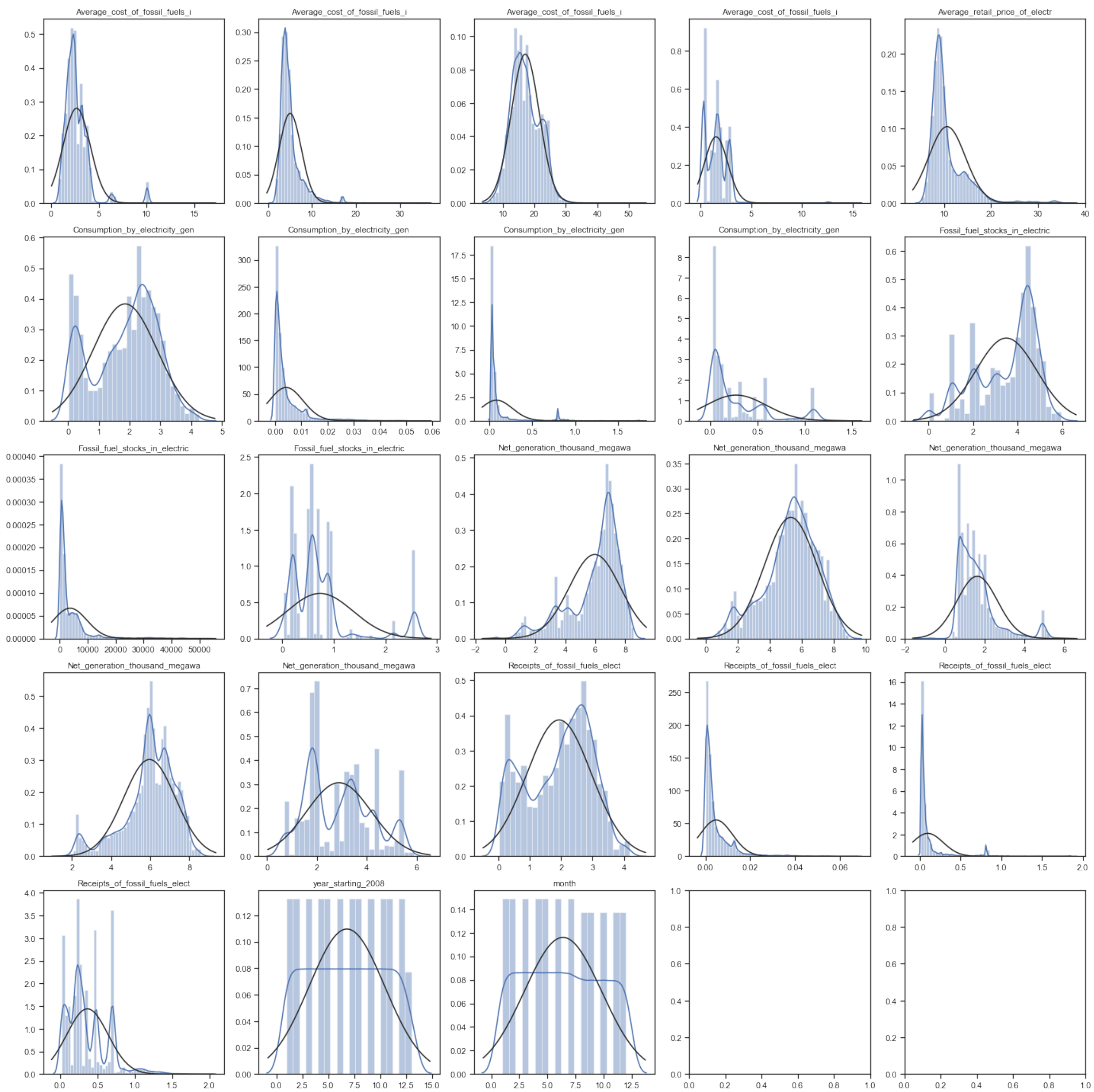
12 Create Chronological Features: Capturing Seasonality by Month and Sequential Time Increments of Years and Months

[17]:
```
model_df3 = add_chrono_features(model_df2)
```

13 Review Distributions of Variables and Perform Log Transformations

- The average retail price of electricity does not have a normal distribution, and appears to be right-skewed.
- The remaining feature variables were log transformed (excluding chronological, etc.) to work with more normalized distributions of figures.

[18]:
```
model_df4 = transform_check_distributions(model_df3)
model_df4 = model_df4.reset_index(drop=True)
target_var =␣
 ↪'Average_retail_price_of_electricity_cents_per_kilowatthour__electricity'
```

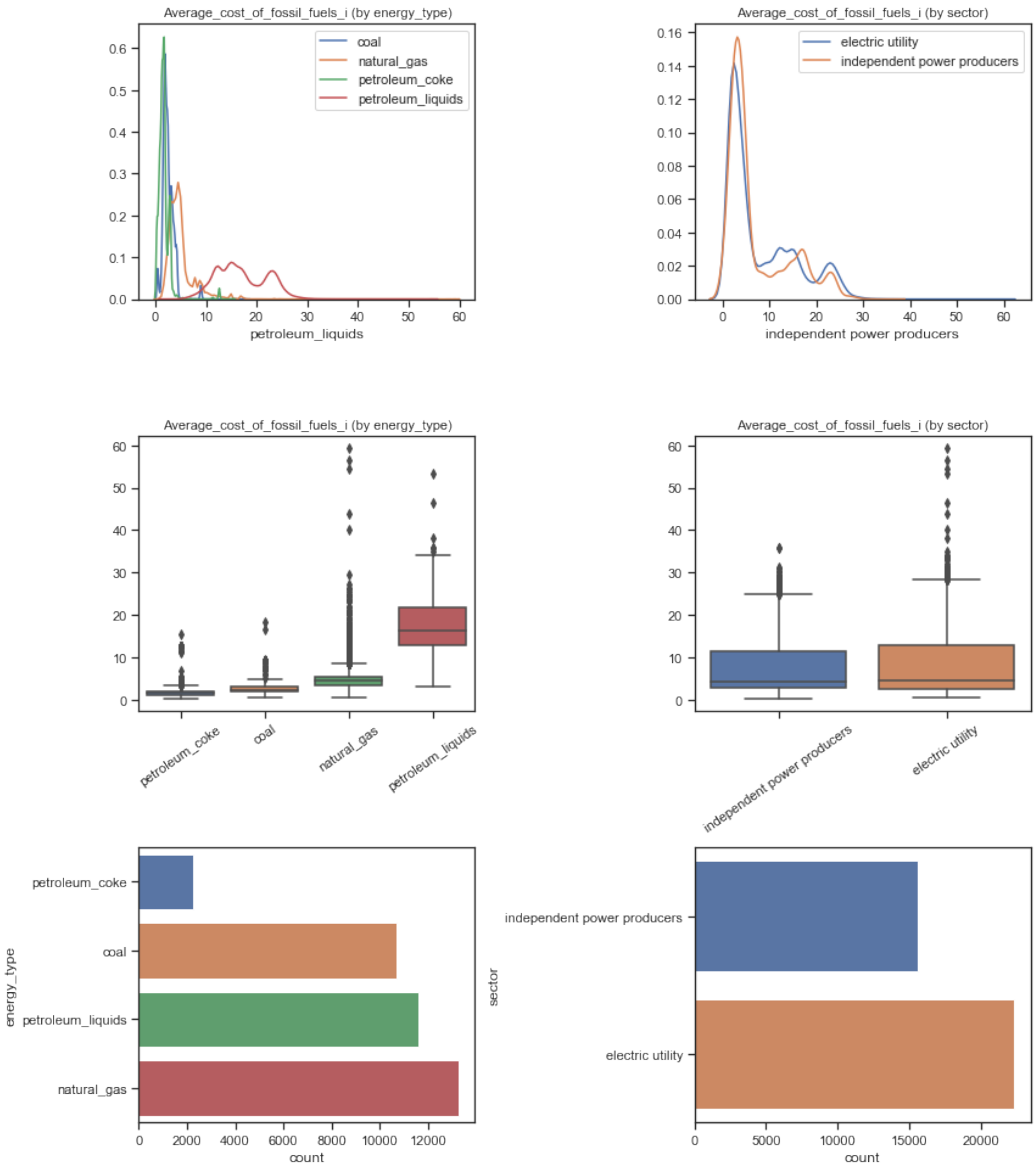


14 Review Impacts of Key Categorical Variables on Target Variable

- Electricity prices tend to be quite volatile and cyclical, with annual and hourly seasonality behaviors in between. Needless to say, there can be quite volatile and will generally show significant outliers from a variety of unique events impacting the market. For example, when a nuclear plant goes down on the east coast it can cause power prices in the area to rocket up over $1,000 per MWh for a brief period of time.

- Petroleum liquids have the largest variation and range in prices per Btu compared to natural gas, petroleum coke, and coal. Petroleum liquids are also the most expensive per Btu, however, petroleum coke is the most inexpensive, high-energy-value fuel in the market and comes off as a bi-product of the petroleum refining process.
- The total data points on petroleum liquids are lower than other energy types, so reliability could be slightly compromised with a smaller data set and a comparatively smaller grouping in the dataset.

[19]:
```
graph_df = df_fill_long.set_index(
    ['variable', 'location', 'energy_type', 'sector', 'year_month']
    ).unstack('variable')
graph_df.columns = graph_df.columns.droplevel(0)

graph_var_list = [
   ␣
 ↪'Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu'
]

for var in graph_var_list:
    target_var_graph_df = graph_df[var].reset_index().dropna()
    discrete_feature_effects(target_var_graph_df.reset_index(),
                             var, ['energy_type', 'sector'])
```



15 Separate Data Into Training and Test Sets for Supervised Machine Learning Models

[20]:
```
# Split data into training and test sets.
test_size = 0.20
random_state = 432
X_train, X_test, y_train, y_test = \
```

```
    split_training_test_data(model_df4.copy(deep=True), target_var, test_size,␣
↪random_state)
```

```
The number of observations in training set is 6160
The number of observations in test set is 1541
```

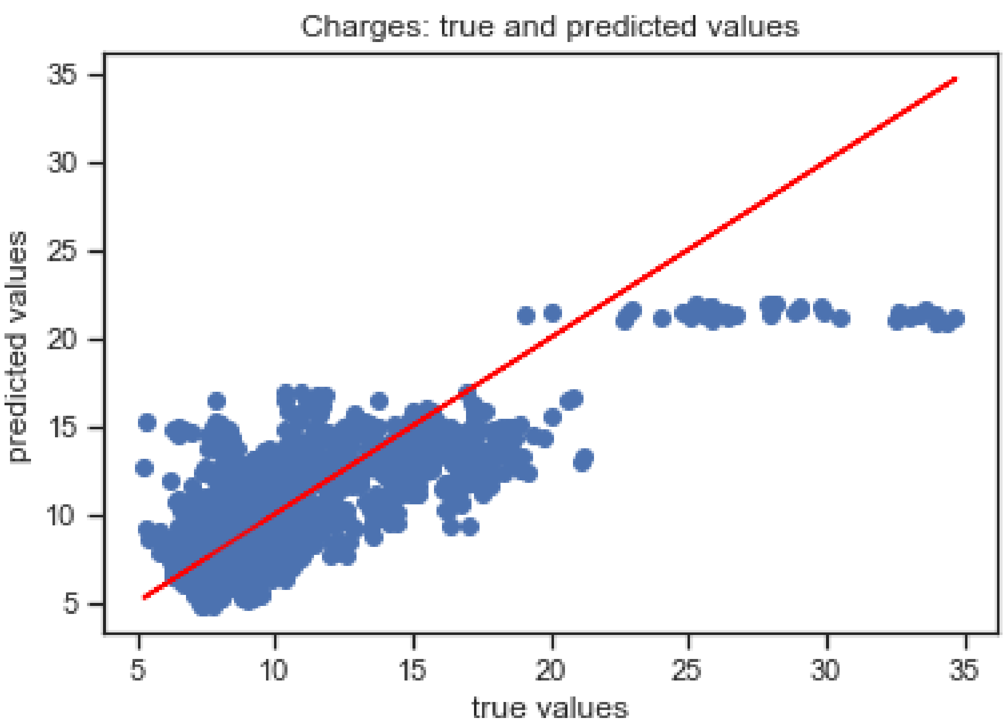
16 Run a Linear Regression on the Model Data

[21]:
```
# Generate linear regression using OLS and check for Markov assumptions.
lr_rval, lr_rmse = generate_linear_regression(model_df4, target_var, X_train,␣
↪y_train, X_test, y_test)
print("SK Learn Linear Regression - Adjusted R-squared value: {:,.2f} with RMSE␣
↪of {:,.1f}.".format(
    lr_rval, lr_rmse))
```

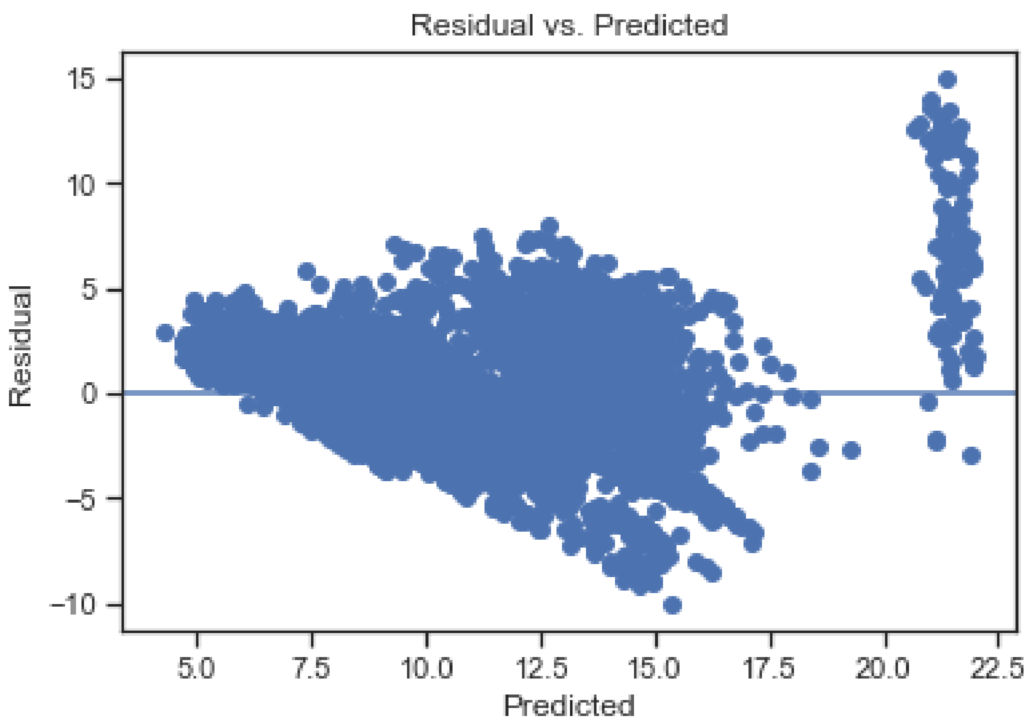
```
Stats Models Results:

<class 'statsmodels.iolib.summary.Summary'>
"""
                                        OLS Regression Results
==========================================================================================
Dep. Variable:     Average_retail_price_of_electricity_cents_per_kilowatthour__electricity   R-
Model:                                                                                 OLS   Ad
Method:                                                                      Least Squares   F-
Date:                                                                     Mon, 09 Nov 2020   Pr
Time:                                                                             15:30:43   Lo
No. Observations:                                                                     6160   AI
Df Residuals:                                                                         6137   BI
Df Model:                                                                               22
Covariance Type:                                                                 nonrobust
==========================================================================================

------------------------------------------------------------------------------------------
const
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__coal
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__natural
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__petrole
Average_cost_of_fossil_fuels_in_electricity_generation_per_Btu_dollars_per_million_Btu__petrole
Consumption_by_electricity_generation_million_MMBtu__coal
Consumption_by_electricity_generation_million_MMBtu__natural_gas
Consumption_by_electricity_generation_million_MMBtu__petroleum_liquids
Consumption_by_electricity_generation_million_MMBtu__petroleum_coke
Fossil_fuel_stocks_in_electricity_generation_million_MMBtu__coal
Fossil_fuel_stocks_in_electricity_generation_MMBtu__petroleum_liquids
Fossil_fuel_stocks_in_electricity_generation_million_MMBtu__petroleum_coke
Net_generation_thousand_megawatthours__coal
Net_generation_thousand_megawatthours__natural_gas
```

```
Net_generation_thousand_megawatthours__petroleum_liquids
Net_generation_thousand_megawatthours__renewable_and_other
Net_generation_thousand_megawatthours__petroleum_coke
Receipts_of_fossil_fuels_electricity_generation_million_MMBtu__coal
Receipts_of_fossil_fuels_electricity_generation_million_MMBtu__natural_gas
Receipts_of_fossil_fuels_electricity_generation_million_MMBtu__petroleum_liquids
Receipts_of_fossil_fuels_electricity_generation_million_MMBtu__petroleum_coke
year
year_starting_2008
month
time_period
==============================================================================
Omnibus:                      792.311   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3006.763
Skew:                           0.608   Prob(JB):                         0.00
Kurtosis:                       6.199   Cond. No.                     4.33e+19
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.66e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

The p-values are less than zero for all coefficients, so they are statistically significant.

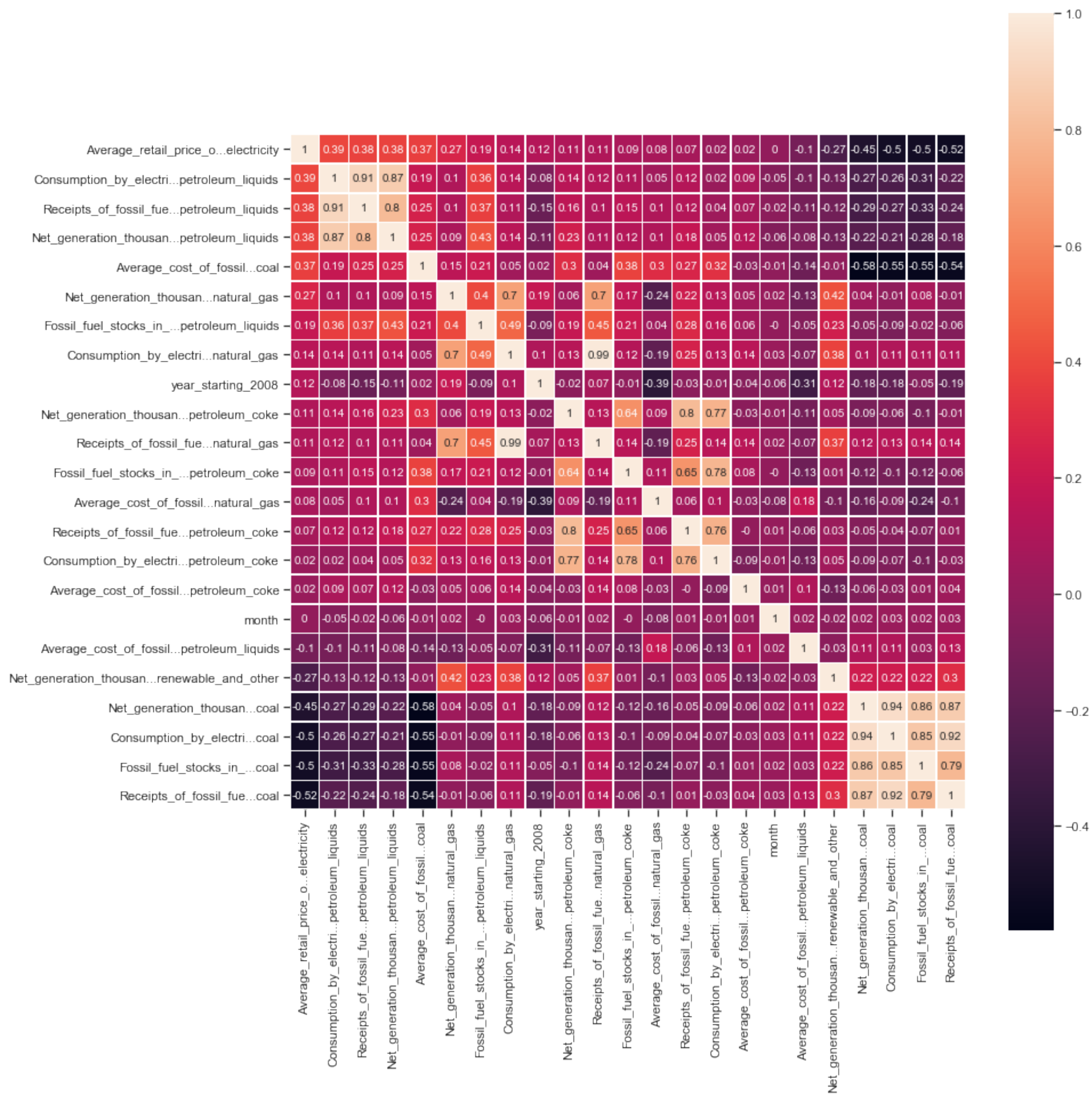


```
Mean absolute error of the prediction is: 1.9
Mean squared error of the prediction is: 7.0
Root mean squared error of the prediction is: 2.6
Mean absolute percentage error of the prediction is: 18.2

Coefficients:
 [ 1.15547373e-01  1.76815065e-01  1.22790864e-03 -2.43443167e-01
 -1.31408713e+00  4.25659562e+02  2.03446639e+00 -2.47260670e+00
 -9.26245826e-01 -1.69389703e-06  3.84222108e-01  8.05442952e-01
  1.03262874e+00  1.76868209e-01 -1.13847279e+00  9.01165265e-01
 -8.45427141e-02 -3.55878062e+02  3.64379054e-01 -3.18793059e+00
 -3.12401893e-04 -3.12401965e-04  1.24072275e-02  8.65840814e-03]

Intercept:
 10.55566095136773
The error term should be zero on average.
Mean of the errors in the model is: -0.000
```

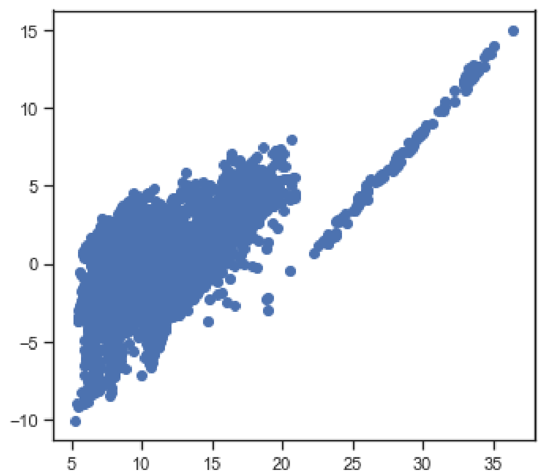


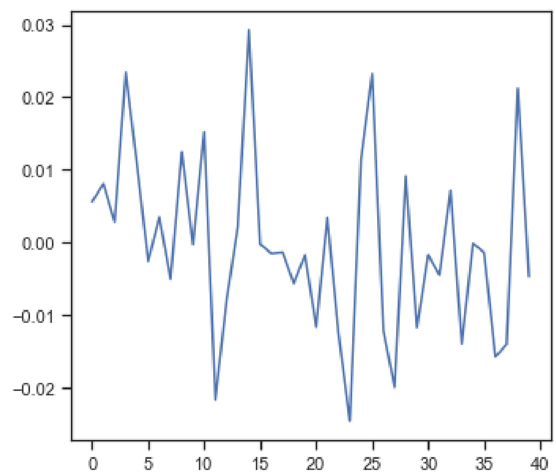

```
Bartlett test statistic value is 83.0 and p value is 0.000
Levene test statistic value is 86.7 and p value is 0.000
Bartlett and Levene tests both share a null hypothesis that the errors are
homoscedastic. If the p-values are less than 0.05, then the results reject the
```

null hypothesis and the errors are heteroscedastic.
Causes of heteroscedasticity include outliers in the data and omitted variables important in explaining the target variance. Include relevant features that target the poorly-estimated areas or transform the dependent variable. Models which suffer from heteroscedasticity still have estimated coefficients which are consistent (still valid). The reliability of some statistical tests, like the t-test, are affected and may make some estimated coefficients falsely appear to be statistically insignificant.

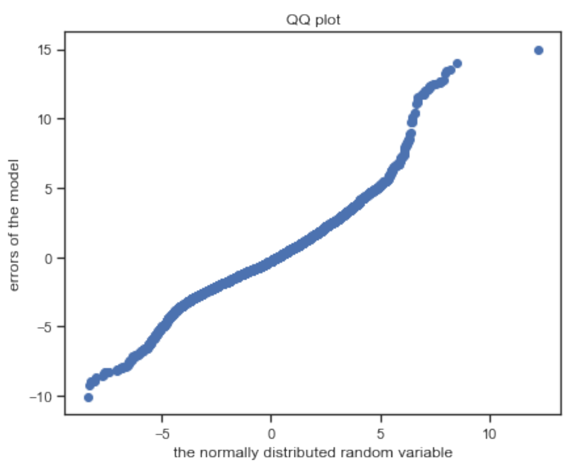


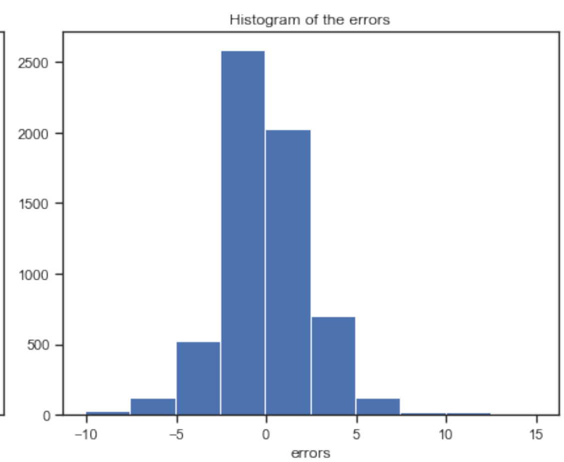
Individual features are only weakly correlated with one another, therefore we have low multicolinearity.



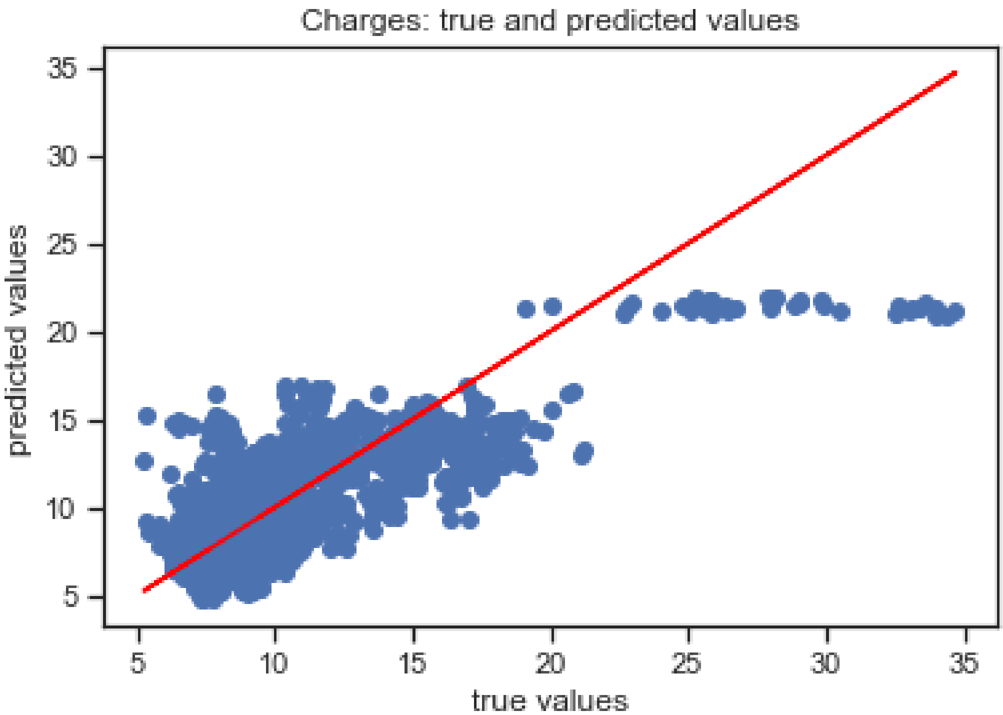


The error terms should be uncorrelated with one another (low R-values).





Jarque-Bera test statistics is 3,006.8 and p value is 0.000
Normality test statistics is 792.3 and p value is 0.000
The errors appear to be normally distributed from a visual inspection.
The p-values of both tests (<0.05) indicate that our errors are not normally distributed.




```
Mean absolute error of the prediction is: 1.9
Mean squared error of the prediction is: 7.0
Root mean squared error of the prediction is: 2.6
Mean absolute percentage error of the prediction is: 18.2
SK Learn Linear Regression - Adjusted R-squared value: 0.57 with RMSE of 2.6.
```

17 Run a Random Forest Regression, Plain Vanilla

[22]:
```
rf_regr = RandomForestRegressor(max_depth=None, random_state=432)
rf_regr.fit(X_train, y_train)
rfr_pred = pd.Series(rf_regr.predict(X_test))
display(rfr_pred.describe())
rf_regr_score = rf_regr.score(X_test, y_test)
print("Adjusted R-value: {:,.2f}".format(rf_regr_score), "\n\nCross Validation␣
↪Scores:")
display(cross_val_score(rf_regr, pd.concat([X_train, X_test]), pd.
↪concat([y_train, y_test]), cv=10).round(2))

print("Mean absolute error of the prediction is: ${:,.2f}".
↪format(mean_absolute_error(y_test, rfr_pred)))
print("Mean squared error of the prediction is: ${:,.2f}".format(mse(y_test,␣
↪rfr_pred)))
```

```
print("Root mean squared error of the prediction is: ${:,.2f}".
↪format(rmse(y_test, rfr_pred)))
print("Mean absolute percentage error of the prediction is: {:,.1f}%".format(np.
↪mean(np.abs((y_test - rfr_pred) / y_test)) * 100))

plt.scatter(y_test, rfr_pred)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Random Forest Regressor - Plain Vanilla")
plt.show()
```

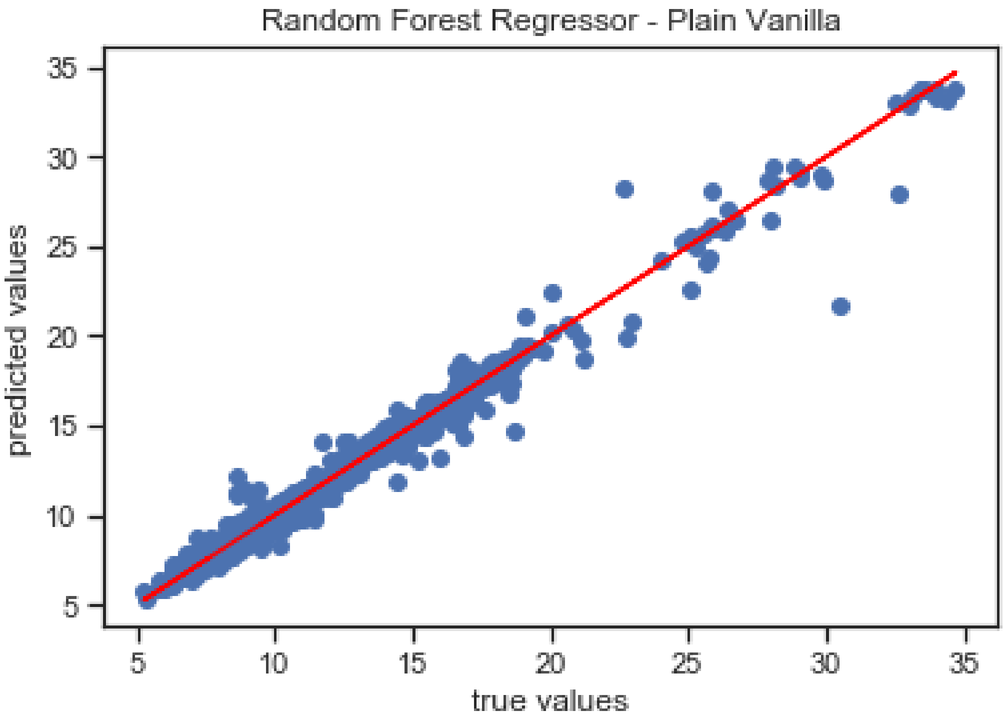
```
count   1,541.000
mean       10.693
std         3.970
min         5.368
25%         8.435
50%         9.404
75%        11.412
max        33.776
dtype: float64
```

```
Adjusted R-value: 0.98
```

```
Cross Validation Scores:
```

```
array([0.98, 0.97, 0.94, 0.98, 0.99, 0.98, 0.98, 0.97, 0.98, 0.98])
```

```
Mean absolute error of the prediction is: $0.32
Mean squared error of the prediction is: $0.32
Root mean squared error of the prediction is: $0.56
Mean absolute percentage error of the prediction is: 39.3%
```



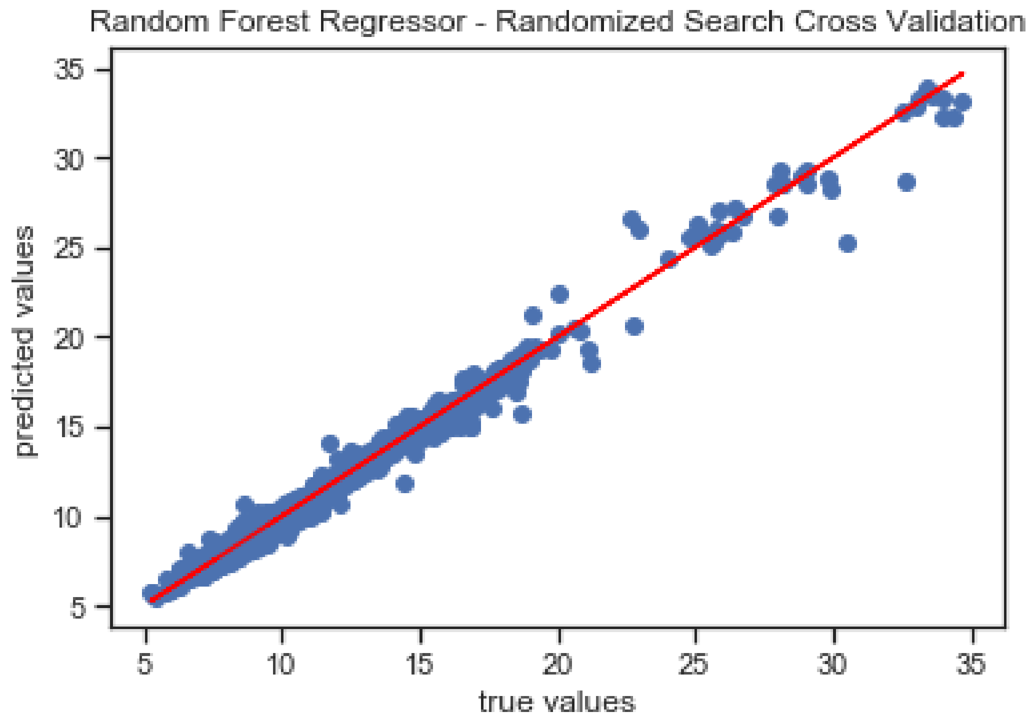
18 Random Forest Regression with Randomized Search Cross Validation for Hyperparameter Tuning

[23]:
```
rf_regr_random = RandomForestRegressor(max_depth=None, random_state=432)

random_grid = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 30],
    'max_features': [2, 3, 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [50, 75, 100]
}

rf_random = RandomizedSearchCV(estimator=rf_regr_random,␣
 ↪param_distributions=random_grid,
                               n_iter=100, cv=3, verbose=2, random_state=42,␣
 ↪n_jobs=-1)
                               # scoring="neg_mean_squared_error")
# Fit the random search model
rf_random.fit(X_train, y_train)
print("RF Random best parameters:", rf_random.best_params_)
```

```
rf_best_random = rf_random.best_estimator_
rfr_random_pred = pd.Series(rf_best_random.predict(X_test))
display(rfr_random_pred.describe())
print("Adjusted R-value: {:,.2f}".format(rf_best_random.score(X_test, y_test)),
 "\n\nCross Validation Scores:")
# display(cross_val_score(rf_random, pd.concat([X_train, X_test]), pd.
 concat([y_train, y_test]), cv=10).round(2))

print("Mean absolute error of the prediction is: ${:,.2f}".
 format(mean_absolute_error(y_test, rfr_random_pred)))
print("Mean squared error of the prediction is: ${:,.2f}".format(mse(y_test,
 rfr_random_pred)))
print("Root mean squared error of the prediction is: ${:,.2f}".
 format(rmse(y_test, rfr_random_pred)))
print("Mean absolute percentage error of the prediction is: {:,.1f}%".format(np.
 mean(np.abs((y_test - rfr_random_pred) / y_test)) * 100))

plt.scatter(y_test, rfr_random_pred)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Random Forest Regressor - Randomized Search Cross Validation")
plt.show()
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   23.2s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   45.0s finished

RF Random best parameters: {'n_estimators': 100, 'min_samples_split': 5,
'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 20, 'bootstrap':
False}

count   1,541.000
mean       10.701
std         3.970
min         5.507
25%         8.472
50%         9.388
75%        11.451
max        33.894
dtype: float64


Adjusted R-value: 0.99

Cross Validation Scores:
```

```
Mean absolute error of the prediction is: $0.29
Mean squared error of the prediction is: $0.22
Root mean squared error of the prediction is: $0.47
Mean absolute percentage error of the prediction is: 39.4%
```



19 Random Forest Regression with Grid Search Cross Validation for Hyperparameter Tuning

[24]:

```
param_grid = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 30],
    'max_features': [2, 3, 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [50, 75, 100]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
rf_grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                              cv=3, n_jobs=-1, verbose = 2)
rf_grid_search.fit(X_train, y_train)
```

```
print(rf_grid_search.best_params_)
rf_best_grid = rf_grid_search.best_estimator_
rfr_grid_pred = pd.Series(rf_best_grid.predict(X_test))
display(rfr_grid_pred.describe())
print("Adjusted R-value: {:,.2f}".format(rf_best_grid.score(X_test, y_test)),␣
 ↪"\n\nCross Validation Scores:")
# display(cross_val_score(rf_random, pd.concat([X_train, X_test]), pd.
 ↪concat([y_train, y_test]), cv=10).round(2))

print("Mean absolute error of the prediction is: ${:,.2f}".
 ↪format(mean_absolute_error(y_test, rfr_grid_pred)))
print("Mean squared error of the prediction is: ${:,.2f}".format(mse(y_test,␣
 ↪rfr_grid_pred)))
print("Root mean squared error of the prediction is: ${:,.2f}".
 ↪format(rmse(y_test, rfr_grid_pred)))
print("Mean absolute percentage error of the prediction is: {:,.1f}%".format(np.
 ↪mean(np.abs((y_test - rfr_grid_pred) / y_test)) * 100))

plt.scatter(y_test, rfr_grid_pred)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Random Forest Regressor - Grid Search Cross Validation")
plt.show()
```

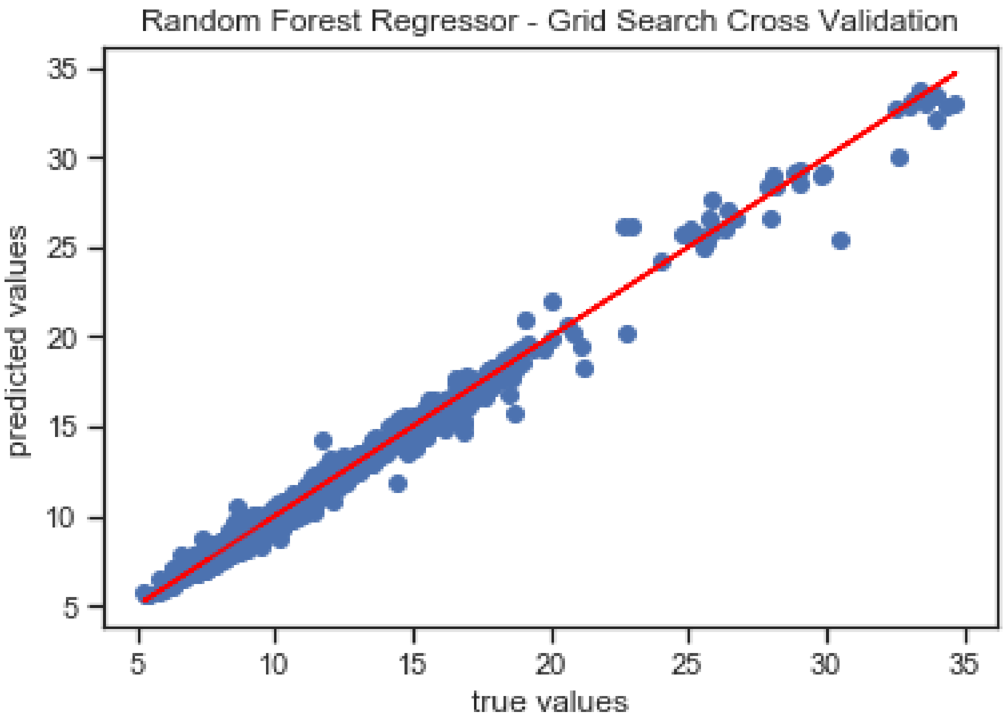
```
Fitting 3 folds for each of 486 candidates, totalling 1458 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.2s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   13.3s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   37.5s
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 997 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 1442 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 1458 out of 1458 | elapsed:  3.4min finished
```

```
{'bootstrap': False, 'max_depth': 30, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

```
count   1,541.000
mean       10.702
std         3.975
min         5.559
25%         8.465
50%         9.391
75%        11.401
max        33.754
dtype: float64
```

```
Adjusted R-value: 0.99

Cross Validation Scores:
Mean absolute error of the prediction is: $0.29
Mean squared error of the prediction is: $0.20
Root mean squared error of the prediction is: $0.45
Mean absolute percentage error of the prediction is: 39.5%
```

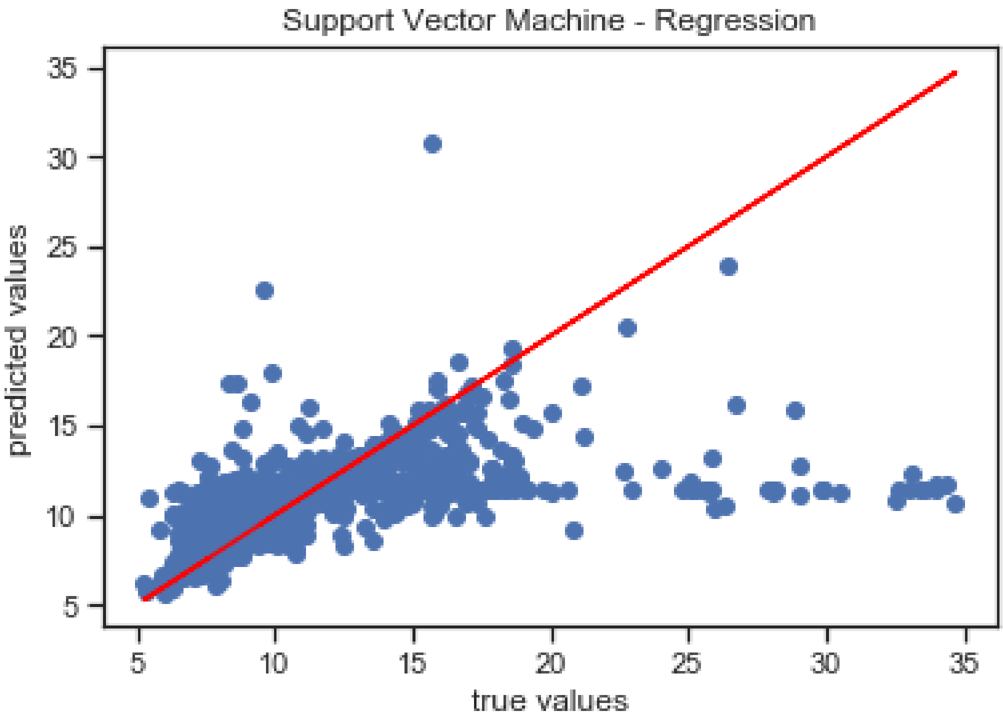


20 Support Vector Machine (SVM), Plain Vanilla

[25]:
```
svr = SVR(kernel='rbf', C=1e5, gamma=0.01)
svr.fit(X_train, y_train)
svr_pred = pd.Series(svr.predict(X_test))
display(svr_pred.describe())
print("Adjusted R-value: {:,.2f}".format(svr.score(X_test, y_test)))

print("Mean absolute error of the prediction is: {:,.1f}".
 ↪format(mean_absolute_error(y_test, svr_pred)))
print("Mean squared error of the prediction is: {:,.1f}".format(mse(y_test,␣
 ↪svr_pred)))
```

```
print("Root mean squared error of the prediction is: {:,.1f}".
 ↪format(rmse(y_test, svr_pred)))
print("Mean absolute percentage error of the prediction is: {:,.1f}".format(np.
 ↪mean(np.abs((y_test - svr_pred) / y_test)) * 100))

plt.scatter(y_test, svr_pred)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Support Vector Machine - Regression")
plt.show()
```

```
count   1,541.000
mean       10.466
std         2.056
min         5.664
25%         9.091
50%        10.480
75%        11.416
max        30.823
dtype: float64


Adjusted R-value: 0.30
Mean absolute error of the prediction is: 1.8
Mean squared error of the prediction is: 11.4
Root mean squared error of the prediction is: 3.4
Mean absolute percentage error of the prediction is: 32.0
```



21 Gradient Boosting Regression, Plain Vanilla Hyperparameters

[26]:
```
reg = GradientBoostingRegressor(random_state=0)
reg.fit(X_train, y_train)
gb_pred = pd.Series(reg.predict(X_test))  # [1:2]
display(gb_pred.describe())
print("{:,.2f}".format(reg.score(X_test, y_test)))

print("Mean absolute error of the prediction is: {:,.1f}".
 ↪format(mean_absolute_error(y_test, gb_pred)))
print("Mean squared error of the prediction is: {:,.1f}".format(mse(y_test,␣
 ↪gb_pred)))
print("Root mean squared error of the prediction is: {:,.1f}".
 ↪format(rmse(y_test, gb_pred)))
print("Mean absolute percentage error of the prediction is: {:,.1f}".format(np.
 ↪mean(np.abs((y_test - gb_pred) / y_test)) * 100))

plt.scatter(y_test, gb_pred)
plt.plot(y_test, y_test, color="red")
plt.xlabel("true values")
plt.ylabel("predicted values")
plt.title("Gradient Boosting - Regression")
```

```
plt.show()
```

```
count   1,541.000
mean       10.684
std         3.817
min         6.075
25%         8.588
50%         9.440
75%        11.354
max        33.635
dtype: float64


0.95
Mean absolute error of the prediction is: 0.7
Mean squared error of the prediction is: 0.9
Root mean squared error of the prediction is: 0.9
Mean absolute percentage error of the prediction is: 38.4
```

22 Linear Regressions: Ridge Regression, Lasso Regression, and Elastic Regression

[32]:
```
data_dict = {
    'index': [],
    'r2_training': [],
    'r2_test': [],
    'avg_abs_err': [],
    'avg_sqrd_err': [],
    'root_mean_sqrd_err': [],
    'avg_abs_pct_err': [],
    'est_regularization_alpha': [],
    'intercept': [],
    'coef_vectors': [],
    'l1_l2_ratio': []
}

# Ridge Regression with Cross Validation.
ridge_regr_cv = RidgeCV(
    alphas=(0.10, 0.25, 0.50, 0.75, 1.00, 1.50, 2.00, 3.00, 5.00, 10.00),
    fit_intercept=True,
    normalize=False,
    scoring=None,
    cv=5,
    gcv_mode=None,
    store_cv_values=False # ridge_regr_cv.coef_
)
ridge_regr_cv.fit(X_train, y_train)
y_preds_train_ridgecv = ridge_regr_cv.predict(X_train)
y_preds_test_ridgecv = ridge_regr_cv.predict(X_test)
data_dict['index'].append('Ridge Regression with CV')
data_dict['r2_training'].append(ridge_regr_cv.score(X_train, y_train))
data_dict['r2_test'].append(ridge_regr_cv.score(X_test, y_test))
data_dict['avg_abs_err'].append(mean_absolute_error(y_test,␣
↪y_preds_test_ridgecv))
data_dict['avg_sqrd_err'].append(mse(y_test, y_preds_test_ridgecv))
data_dict['root_mean_sqrd_err'].append(rmse(y_test, y_preds_test_ridgecv))
data_dict['avg_abs_pct_err'].append( np.mean( np.abs( (y_test -␣
↪y_preds_test_ridgecv) / y_test ) ) * 100 )
data_dict['est_regularization_alpha'].append(ridge_regr_cv.alpha_)
data_dict['intercept'].append(ridge_regr_cv.intercept_)
data_dict['coef_vectors'].append(ridge_regr_cv.coef_)
data_dict['l1_l2_ratio'].append(np.nan)


# Lasso Regression with Cross Validation.
lasso_regr_cv = LassoCV(
```

```
    alphas=(0.10, 0.25, 0.50, 0.75, 1.00, 1.50, 2.00, 3.00, 5.00, 10.00),
    fit_intercept=True,
    normalize=False,
    cv=5,
    n_alphas=None
)
lasso_regr_cv.fit(X_train, y_train)
y_preds_train_lassocv = lasso_regr_cv.predict(X_train)
y_preds_test_lassocv = lasso_regr_cv.predict(X_test)
data_dict['index'].append('Lasso Regression with CV')
data_dict['r2_training'].append(lasso_regr_cv.score(X_train, y_train))
data_dict['r2_test'].append(lasso_regr_cv.score(X_test, y_test))
data_dict['avg_abs_err'].append(mean_absolute_error(y_test,␣
 ↪y_preds_test_lassocv))
data_dict['avg_sqrd_err'].append(mse(y_test, y_preds_test_lassocv))
data_dict['root_mean_sqrd_err'].append(rmse(y_test, y_preds_test_lassocv))
data_dict['avg_abs_pct_err'].append( np.mean( np.abs( (y_test -␣
 ↪y_preds_test_lassocv) / y_test ) ) * 100 )
data_dict['est_regularization_alpha'].append(lasso_regr_cv.alpha_)
data_dict['intercept'].append(lasso_regr_cv.intercept_)
data_dict['coef_vectors'].append(lasso_regr_cv.coef_)
data_dict['l1_l2_ratio'].append(np.nan)


# ElasticNet Regression with Cross Validation.
elastic_regr_cv = ElasticNetCV(
    l1_ratio=[0.10, 0.50, 0.70, 0.90, 0.95, 0.99, 1.00],
    alphas=(0.10, 0.25, 0.50, 0.75, 1.00, 1.50, 2.00, 3.00, 5.00, 10.00),
    fit_intercept=True,
    normalize=False,
    cv=5,
    n_alphas=None
)
elastic_regr_cv.fit(X_train, y_train)
y_preds_train_elasticcv = elastic_regr_cv.predict(X_train)
y_preds_test_elasticcv = elastic_regr_cv.predict(X_test)
data_dict['index'].append('ElasticNet Regression with CV')
data_dict['r2_training'].append(elastic_regr_cv.score(X_train, y_train))
data_dict['r2_test'].append(elastic_regr_cv.score(X_test, y_test))
data_dict['avg_abs_err'].append(mean_absolute_error(y_test,␣
 ↪y_preds_test_elasticcv))
data_dict['avg_sqrd_err'].append(mse(y_test, y_preds_test_elasticcv))
data_dict['root_mean_sqrd_err'].append(rmse(y_test, y_preds_test_elasticcv))
data_dict['avg_abs_pct_err'].append( np.mean( np.abs( (y_test -␣
 ↪y_preds_test_elasticcv) / y_test ) ) * 100 )
data_dict['est_regularization_alpha'].append(elastic_regr_cv.alpha_)
data_dict['intercept'].append(elastic_regr_cv.intercept_)
```

```
data_dict['coef_vectors'].append(elastic_regr_cv.coef_)
data_dict['l1_l2_ratio'].append(elastic_regr_cv.l1_ratio_)

data_dict.pop('coef_vectors')
df = pd.DataFrame.from_dict(data_dict).set_index('index')
df_img_out(df.head(5).iloc[:, :7], 'ridge_lasso_elastic')
```

index	r2_training	r2_test	avg_abs_err	avg_sqrd_err	root_mean_sqrd_err	avg_abs_pct_err	est_regularization_alpha
Ridge Regression with CV	0.548	0.562	1.957	7.142	2.672	18.486	0.100
Lasso Regression with CV	0.497	0.503	2.018	8.112	2.848	18.823	0.100
ElasticNet Regression with CV	0.522	0.528	1.985	7.706	2.776	18.471	0.100

23 Analyze and Compare Model Approaches and Final Results

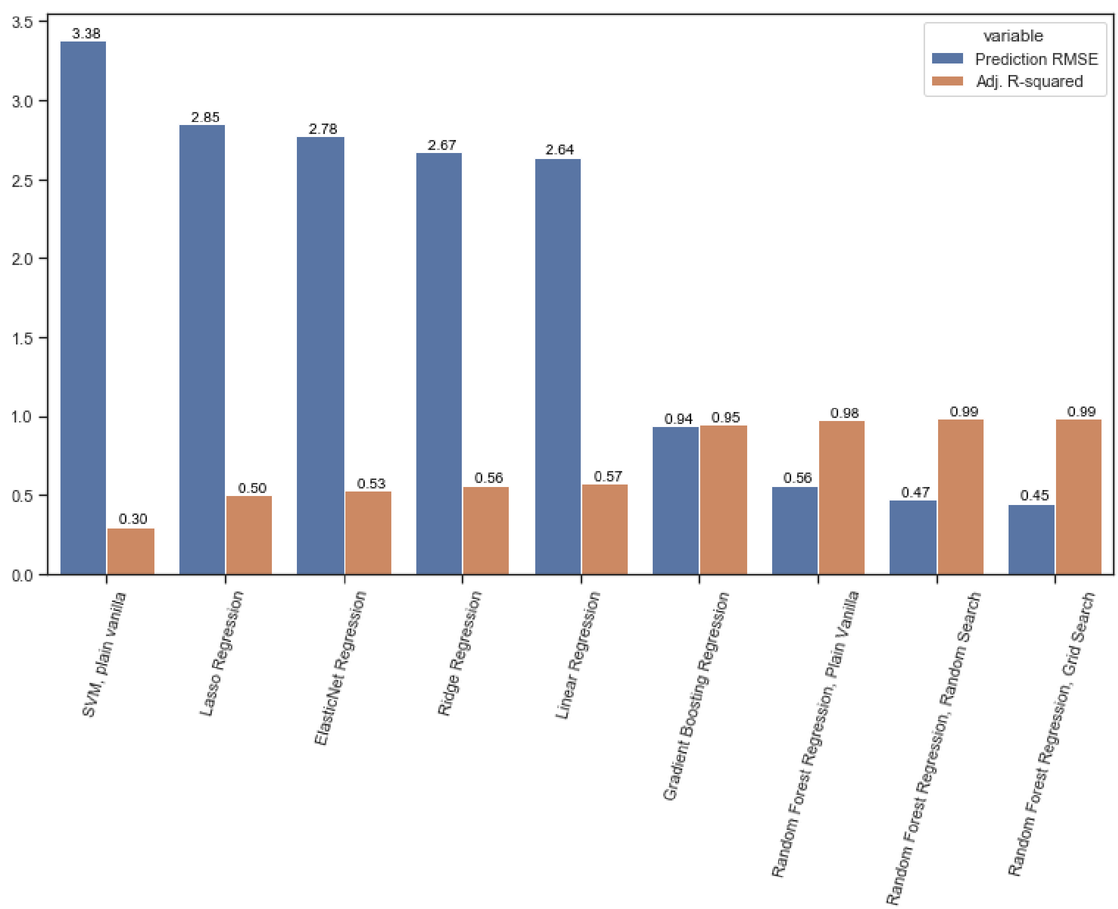
The Random Forest Regression with a Grid Search Cross Validation was the best model based upon the highest adjusted R-value of 0.987, indicating a goodness of fit to the test data and the lowest root mean squared error (RMSE) of $0.45 indicating superior results with the test data or predictive modeling power.

- A Linear Regression resulted in an RMSE of the prediction of `$2.64` and adjusted R-squared value of `0.57`. Linear regression kernel variations of Ridge Regression, Lasso Regression, and ElasticNet Regression came to similar figures but were slightly less optimal.
- A Random Forest Regression, Plain Vanilla resulted in an RMSE of `$0.56` and an adjusted R-squared value of `0.981`. When Random Search or Grid Search Cross Validation were applied to hyperparameter tuning, the random forest regression improved to an RMSE of `$0.45 - $0.46` and an adjusted R-squared value of `0.987`.
- A Support Vector Machine (SVM) Regression, plain vanilla, resulted in an RMSE of `$3.38` and an adjusted R-squared value of `0.30`. These results were not comparable to the higher performances of other model approaches.
- A Gradient Boosting Regression resulted in an RMSE of `$0.94` and an adjusted R-squared value of `0.95`.

[33]:
```
model_results_df = pd.DataFrame.from_dict( {
    'Linear Regression': {
        'Prediction RMSE': lr_rmse,
         'Adj. R-squared': lr_rval
    },
    'Ridge Regression': {
        'Prediction RMSE': rmse(y_test, y_preds_test_ridgecv),
         'Adj. R-squared': ridge_regr_cv.score(X_test, y_test)
    },
    'Lasso Regression': {
        'Prediction RMSE': rmse(y_test, y_preds_test_lassocv),
```

```
        'Adj. R-squared': lasso_regr_cv.score(X_test, y_test)
    },
    'ElasticNet Regression': {
        'Prediction RMSE': rmse(y_test, y_preds_test_elasticcv),
         'Adj. R-squared': elastic_regr_cv.score(X_test, y_test)
    },
    'Random Forest Regression, Plain Vanilla': {
        'Prediction RMSE': rmse(y_test, rfr_pred),
         'Adj. R-squared': rf_regr_score # rf_regr.score(X_test, y_test)
    },
    'Random Forest Regression, Random Search': {
        'Prediction RMSE': rmse(y_test, rfr_random_pred),
         'Adj. R-squared': rf_best_random.score(X_test, y_test)
    },
    'Random Forest Regression, Grid Search': {
        'Prediction RMSE': rmse(y_test, rfr_grid_pred),
         'Adj. R-squared': rf_best_grid.score(X_test, y_test)
    },
    'SVM, plain vanilla': {
        'Prediction RMSE': rmse(y_test, svr_pred),
         'Adj. R-squared': svr.score(X_test, y_test)
    },
    'Gradient Boosting Regression': {
        'Prediction RMSE': rmse(y_test, gb_pred),
         'Adj. R-squared': reg.score(X_test, y_test)},
} ).transpose().sort_values(by=['Adj. R-squared'])
df_img_out(model_results_df.head(20).iloc[:, :7], 'model_results_df')
sns_df = model_results_df.reset_index().melt(id_vars=['index'])
plt.figure(figsize=(13,7))
ax = sns.barplot(x='index', y='value', hue='variable', data=sns_df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=75)
for p in ax.patches:
        ax.annotate('{:<8,.2f}'.format(p.get_height()), ( p.get_x() + 0.3, p.
 ↪get_height() ),
                    ha='center', va='bottom', color='black', fontsize='medium')
ax.set_ylabel('')
ax.set_xlabel('')
plt.show()
```

	Prediction RMSE	Adj. R-squared
SVM, plain vanilla	3.378	0.301
Lasso Regression	2.848	0.503
ElasticNet Regression	2.776	0.528
Ridge Regression	2.672	0.562
Linear Regression	2.641	0.572
Gradient Boosting Regression	0.940	0.946
Random Forest Regression, Plain Vanilla	0.562	0.981
Random Forest Regression, Random Search	0.470	0.986
Random Forest Regression, Grid Search	0.450	0.988



24 Appendix: Additional Data Wrangling Code to Create 'eia_data.csv'.

[29]:

```
"""
    Take a folder path and generate a 2D dictionary of file names and paths of␣
↪all .csv files
    within the selected folder.  The keys are the file names with '.csv'␣
↪extensions in the folder.
    Each item value is a 2-key dictionary holding 'path' and 'name.'  This csv␣
↪dictionary is later
    used as an input parameter for csv_list_to_dataframes().
        Function call example:
        location = "C:\\dev\\Thinkful\\25. Supervised Learning Capstone␣
↪Project\\EIA Data Sources\\"
        csv_data_set1 = make_csv_dict(location)
"""
def make_csv_dict(location):
    file_list = []
    # r=>root, d=>directories, f=>files
    for r, d, f in os.walk(location):
       for item in f:
          file_list.append(item) if '.csv' in item else next
    csv_data_set = {}
    for item in file_list:
        csv_data_set[item.replace('.csv', '')] = {'path': location, 'name':␣
↪item}
    return csv_data_set




"""
    Read csv file to determine row number of data headers.
        Helper function for csv_list_to_df_dict().
        csv_header_row = get_header_row(full_csv_path)
"""
def get_header_row(full_csv_path):
    file_obj = open(full_csv_path)
    csv_reader_obj = csv.reader(file_obj)
    header_row = 0
    for row in csv_reader_obj:
        if len(row) > 1:
            break
        header_row += 1
    return header_row
```

```
"""
    Take a structured dictionary of CSV file locations or web addresses and␣
↪download
    the data into a dictionary of dataframes.  Also checks to see if all the␣
↪columns
    are matching on the tables or not.  Function call example:
        columns_df = csv_list_to_dataframes(csv_data_set, df_dict)
"""
def csv_list_to_df_dict(csv_data_set, df_dict, na_values):
    # Setup data buckets for loop through csv files.
    summary_df = pd.DataFrame()
    dfs_summary_dict = {}
    summary_df_col_index = []
    col_rename_dict = {}
    column_max_count, max_key = 0, ''

    # Loop keys/csv filenames to fill df dictionary and matching column check␣
↪table.
    for table_name in csv_data_set.keys():

        # Download the csv file into a DataFrame and add it to the df_dict.
        full_csv_path = csv_data_set[table_name]['path'] \
                        + csv_data_set[table_name]['name']
        csv_header_row = get_header_row(full_csv_path)
        data = pd.read_csv(full_csv_path, header = csv_header_row,␣
↪na_values=na_values)#.fillna(0)
        df_dict[table_name] = data.copy()

        # Collect summary information on the DataFrames.
        col_data = list(df_dict[table_name].columns)
        dfs_summary_dict.update({ table_name: col_data })
        if len(df_dict[table_name].columns) > column_max_count:
            column_max_count = len(df_dict[table_name].columns)
            max_key = table_name

    # Select one of the largest DataFrames to set the column order.
    column_index_key = df_dict[max_key].columns

    # Modify list of column values in dfs_summary_dict > summary_df.
    matches, insertions = 0, 0
    for report, col_data in dfs_summary_dict.items():
        col_idx = 0
        for key_idx in range(0, len(column_index_key)):
            no_match = column_index_key[key_idx] != col_data[col_idx]
```

```
            if no_match:
                col_data.insert(col_idx, np.nan)
                insertions += 1
                col_idx += 1
            else:
                matches += 1
                col_idx += 1
    equalized_columns = { key: pd.Series(value) for key, value in␣
↪dfs_summary_dict.items() }
    equalized_columns_df = pd.DataFrame.from_dict(equalized_columns)
    summary_df_col_index += list(equalized_columns_df.columns)
    for i in range(len(summary_df_col_index)):
        col_rename_dict.update({i: summary_df_col_index[i]})
    concat_list = [summary_df, equalized_columns_df]
    summary_df = pd.concat(concat_list, copy=False,␣
↪axis='columns')[equalized_columns_df.columns]
    summary_df.rename(columns=col_rename_dict, inplace=True)
    equal_columns = summary_df.apply(lambda row: row[0] == row.all(), axis=1)
    summary_df.insert(loc=0, column='_equal_table_columns', value=equal_columns)

    # Copy into a column-only dataframe, match_columns_df, excluding dimension␣
↪rows.
    match_columns_df = summary_df.reset_index(drop=True).copy(deep=True)
    val_cnts = match_columns_df['_equal_table_columns'].value_counts()
    summary_df.index.name = 'col_pos'
    summary_df = summary_df.transpose()
    summary_df.index.name = 'table_names'
    summary_df.reset_index(inplace=True)
    print("\n{:,} matching columns and {:,} mis-matching columns (element-wise␣
↪including NaNs) for all csv tables > dataframes.".format(
        val_cnts.at[True] if True in val_cnts.index else 0,
        val_cnts.at[False] if False in val_cnts.index else 0))
    print("{:,} max columns across all dataframes.".format(column_max_count))
    return summary_df, column_index_key




"""
    Take a dictionary of dataframes and convert them into a single large␣
↪dataframe.
    Also checks to see if the columns names and total row numbers are matching␣
↪on the tables or not.
    Function call example:
        folder_df1 = df_dict_to_single_df(df_dict1)
"""
```

```
def df_dict_to_folder_df(df_dict, column_index_key):
    # Concatenate vertically along 0/columns and multi-index on csv names.
    new_row_label = 'csv_table_names'
    df_list = []
    table_row_counts = []
    for key, df in df_dict.items():
        table_row_counts.append(df.shape[0])
        csv_name_col = df.apply(lambda row: key, axis=1)
        df.insert(loc=0, column=new_row_label, value=csv_name_col)
        df_list.append(df)
    column_index_key = column_index_key.insert(0, new_row_label)
    folder_df = pd.concat(df_list)
    folder_df.reset_index(drop=True, inplace=True)

    # Check if concatenation may have date-based alignment errors.
    folder_df_concat_good = folder_df.shape[0] == sum(table_row_counts)
    if not folder_df_concat_good:
        print("ERROR: Concatenation of DataFrames did not result in the same␣
↪number of rows.".format(bad_stat))
        print("{:,} total rows in the dictionary of DataFrames.".
↪format(sum(table_row_counts)))
        print("{:,} rows in the vertically concatenated DataFrame result.".
↪format(folder_df.shape[0]))
    return folder_df, column_index_key




"""
    Drop zero-sum numeric rows from dataframe folder_df.  If non_numeric_cols␣
↪are not provided
    in a manual list, then Numeric columns are selected by their data type␣
↪being either
    int64 or float64.  Function call example:
        non_numeric_cols = ['description', 'units', 'source key']
        drop_zero_sum_numeric_rows(folder_df, non_numeric_cols)
"""
def drop_zero_sum_numeric_rows(folder_df, non_numeric_cols=None):
    if non_numeric_cols == None:
        non_numeric_cols = []
        dtype_dict = dict(folder_df.dtypes)
        for col in dtype_dict:
            if dtype_dict[col] != 'float64' and dtype_dict[col] != 'int64':
                non_numeric_cols.append(col)
    numeric_df = folder_df.drop(columns=non_numeric_cols)
    zero_rows_df = pd.DataFrame(numeric_df.apply(lambda row: True if np.
↪sum(row) == 0 else False, axis=1),
```

```
                                columns=['zero_sum_rows'],
                                index=numeric_df.index)
    zero_rows_only_df = zero_rows_df[zero_rows_df['zero_sum_rows'] == True]

    # Drop rows with zero sum numeric amounts.
    folder_df.drop(index=zero_rows_only_df.index, inplace=True)

    # Count rows and check for potential errors.
    all_row_ct = zero_rows_df.shape[0]
    zero_sum_row_ct = zero_rows_only_df.shape[0]
    remaining_row_ct = folder_df.shape[0]
    if all_row_ct != (zero_sum_row_ct + remaining_row_ct):
        print("Total rows {} less {} non-numeric rows equals {}, but folder_df␣
↪now has {}".format(
              all_row_ct, zero_sum_row_ct, all_row_ct - zero_sum_row_ct,␣
↪remaining_row_ct))
    folder_df.reset_index(drop=True, inplace=True)
    folder_df.index.rename('index', inplace=True)
    return folder_df



"""
    Aggregate functions to combine data from multiple CSV files within a␣
↪specified folder.
"""
def csv_folder_to_df(folder_location, na_values):
    df_dict = {}
    column_index_key = pd.DataFrame()
    csv_data_set = make_csv_dict(folder_location)
    summary_df, column_index_key = csv_list_to_df_dict(csv_data_set, df_dict,␣
↪na_values)
    folder_df, column_index_key = df_dict_to_folder_df(df_dict,␣
↪column_index_key)
    folder_df_non_zero = drop_zero_sum_numeric_rows(folder_df)
    return folder_df_non_zero

# folder_location = "C:\\dev\\Thinkful\\25. Supervised Learning Capstone␣
↪Project\\EIA Data Sources\\"
# eia_raw_data_df = csv_folder_to_df(folder_location, na_values)
# eia_raw_data_df.to_csv('eia_data.csv')
# na_values = ['', '--', 'NM', 'W']
```