Machine Learning for prediction

Purpose

This project aims to use Machine learning in order to cast a prediction of the Electrical Expenditures in the residential sector for the year 2021

Background

EIA's State Energy Data System (SEDS) is a comprehensive data set that consists of annual time series estimates of state-level energy use by major economic sectors, energy production and and State-level energy price and expenditure data. The system provides data back from 1960. Data are presented in physical units, BTUs, and dollars. While some SEDS data series come directly from surveys conducted by EIA, many are estimated using other available information. These estimations are necessary for the compilation of "total energy" estimates.

Useful Links:

The main website: https://catalog.data.gov/dataset/state-energy-data-system-seds

Additional information: https://www.eia.gov/state/seds/

Codes and descriptions: https://www.eia.gov/state/seds/CDF/Codes_and_Descriptions.xlsx

From the State Data energy System description sheets:
The MSNs are five-character codes, most of which are structured as follows: First and second characters - describes an energy source (for example, NG for natural gas, MG for motor gasoline) Third and fourth characters - describes an energy sector or an energy activity (for example, RC for residential consumption, PR for production) Fifth character - describes a type of data (for example, P for data in physical unit, B for data in billion Btu and D is for dollars per million BTU)

The aim of this project is to predict the Primary average price in the residential sector for the year 2021 based on residential sector features.

Data Loading, cleaning, visualization

In [1]:
```
import pandas as pd
import pyarrow as pa
import os
import zipfile
import matplotlib.pyplot as plt
```

```
from collections import Counter
import numpy as np
import statsmodels.api as sm
import pylab
import scipy.stats as stats
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
import statsmodels.api as sm
import pylab
import scipy.stats as stats

pd.options.display.float_format = '{:.2f}'.format
```

Importing Files Here

In [2]:
```
#pip freeze > requirements.txt
#aaaagggg
os.getcwd()
os.chdir('c:\\Users\\amcfa\\gitfiles\\Projects\\MastersWork\\shorter_ML_Projects\\M
file = pd.read_csv(os.getcwd()+ '\Complete_SEDS.csv')
file1=file
codes = pd.read_csv(os.getcwd()+ '/residential_sector_codes.csv')
codes.head(1)
```

Out[2]:

	CLHCK
0	CLRCB

When loaded the file contains several columns:
`Data_Status` - A Code
`MSN` - A 5 letter combination to signify energy source, sector and unit of data.
Units for each category are given below.
Please see :https://www.eia.gov/state/seds/CDF/Codes_and_Descriptions.xlsx for a full description of each category.

- Million Btu per short ton
- Billion Btu
- Dollars per million Btu
- Thousand short tons
- Million dollars
- Thousand barrels

`StateCode` Two letter state abbreviation
`Year` YYYY-MM-DD
`Data` Numerical - associated with MSN

In [75]:
```
file1.head(5)
```

Out[75]:

	Data_Status	MSN	StateCode	Year	Data
0	2021F	ABICB	AK	1960-01-01	0.00
1	2021F	ABICB	AK	1961-01-01	0.00
2	2021F	ABICB	AK	1962-01-01	0.00
3	2021F	ABICB	AK	1963-01-01	0.00
4	2021F	ABICB	AK	1964-01-01	0.00

Data inspection, cleaning

After loading the data, I will inspect it. A few possible ways are below.

1. Using the .describe() function. Which shows the numerical distribution of the data.

In [4]:
```
file1.describe()
```

Out[4]:

	Year	Data
count	2007511.00	2007511.00
mean	1993.07	102832.83
std	17.15	1351844.51
min	1960.00	-770830.00
25%	1979.00	0.57
50%	1993.00	76.00
75%	2008.00	6503.95
max	2021.00	101641185.00

2. As well as using the dtype function. Which shows that Data is a float column, so there are no Nan variables or strings present.

In [5]:
```
file1.dtypes
```

Out[5]:
```
Data_Status     object
MSN             object
StateCode       object
Year             int64
Data           float64
dtype: object
```

3. Using the isna function to see if there are any na values anywhere.

In [6]:
```
empty_rows = file1[file1.isna().any(axis=1)]
print('There are',len(empty_rows),"empty rows")
```

```
There are 0 empty rows
```

Updating the year to be a datetime object will make working with the dataframe much simpler. Additionally since I am not estimating the expenditures for the United States as a whole. I will exclude this column.

In [7]:
```
file1['Year'] = pd.to_datetime(file1['Year'], format='%Y')
file1=file1[~(file1['StateCode']=='US')]
```

Creating a list of relevant features

In [8]:
```
pqr = codes['CLHCK'].unique() #Taking the unique list of codes
abc = len(pqr) #The length of codes
new_list_of_titles = []
dfs = []
for i in pqr:
        if i not in new_list_of_titles:
                new_list_of_titles.append(i)
print(new_list_of_titles)
```

```
['CLRCB', 'CLRCD', 'CLRCP', 'CLRCV', 'DFRCB', 'DFRCD', 'DFRCP', 'DFRCV', 'GERCB', 'H
LRCB', 'HLRCD', 'HLRCP', 'HLRCV', 'KSRCB', 'KSRCD', 'KSRCP', 'KSRCV', 'LORCB', 'NGRC
B', 'NGRCD', 'NGRCP', 'NGRCV', 'PARCB', 'PARCD', 'PARCK', 'PARCP', 'PARCV', 'PERCD',
'PERCV', 'PERSB', 'PQRCB', 'PQRCD', 'PQRCP', 'PQRCV', 'SFRCB', 'SOR7P', 'SORCB', 'SO
TCB', 'SOTGP', 'SOTXB', 'TERCB', 'TERCD', 'TERCV', 'TERPB', 'TNRCB', 'TNRSB', 'WDRC
B', 'WDRCD', 'WDRCV', 'WDRSB', 'WDRXB']
```

Transposing the data so it will be easier to work with

In [9]:
```
general_consumption= file1[file1['MSN'].str.contains('ESRCV')]
# Convert new_list_of_titles to a single regex pattern
pattern1 = '|'.join(new_list_of_titles)

# Use str.contains() with the regex pattern
filtered_df11 = file1[file1['MSN'].str.contains(pattern1)]
merged_data = pd.merge(general_consumption, filtered_df11, on=["StateCode", "Year",
import pandas as pd
pivoted_data = merged_data.pivot_table(index=["StateCode", "Year", "Data_Status","M
# Reset index to flatten the dataframe
pivoted_data.reset_index(inplace=True)
```

In [10]:
```
pivoted_data.head(10)
```

Out[10]:

	MSN_y	StateCode	Year	Data_Status	MSN_x	Data_x	CLRCB	CLRCD	CLRCP	CLRCV	
0		AK	1970-01-01	2021F	ESRCV	16.70	233.00	2.47	13.00	0.60	1
1		AK	1971-01-01	2021F	ESRCV	20.10	177.00	2.40	10.00	0.40	1(
2		AK	1972-01-01	2021F	ESRCV	22.00	199.00	2.28	11.00	0.50	8
3		AK	1973-01-01	2021F	ESRCV	22.30	92.00	2.44	5.00	0.20	1(
4		AK	1974-01-01	2021F	ESRCV	25.30	92.00	2.54	5.00	0.20	1(
5		AK	1975-01-01	2021F	ESRCV	31.10	88.00	2.87	5.00	0.30	9
6		AK	1976-01-01	2021F	ESRCV	36.60	71.00	2.78	4.00	0.20	1
7		AK	1977-01-01	2021F	ESRCV	44.80	69.00	2.86	4.00	0.20	1:
8		AK	1978-01-01	2021F	ESRCV	49.70	0.00	0.00	0.00	0.00	1!
9		AK	1979-01-01	2021F	ESRCV	49.20	0.00	0.00	0.00	0.00	5

10 rows × 55 columns

In [11]:
```
needed_columns=(pivoted_data.columns[5:])
```

In [12]:
```
needed_columns1 =pivoted_data.iloc[:,4: ]
```

Visualization

Visualizing the data showed that most of the data would not be considered normal.But with so many data points to use, normality can be assumed to some degree. Additionally, values that would be considered outliers should be left in, there were probably times when the budget of a particular sector was indeed 0.

`QQ plots` There were few plots that actually fit the definition of a normal distribution (most had very heavy tails).

`Box plots` Help illustrate the fact that each category is heavily skewed in one direction. All of the outliers are toward the top.

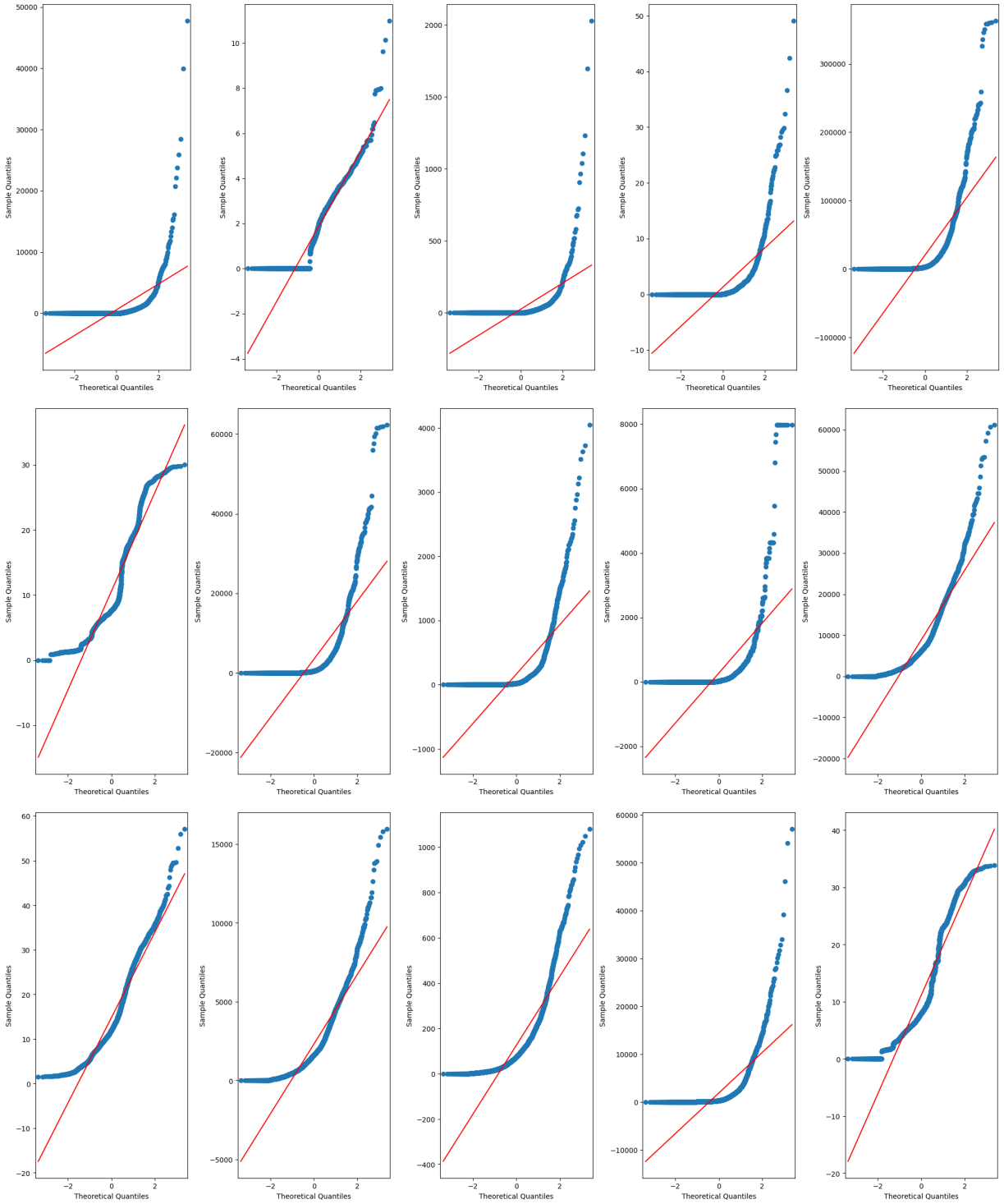
`Corr plot` The correlations were plotted between the target variable and each predictor as well as between the predictors themselves.

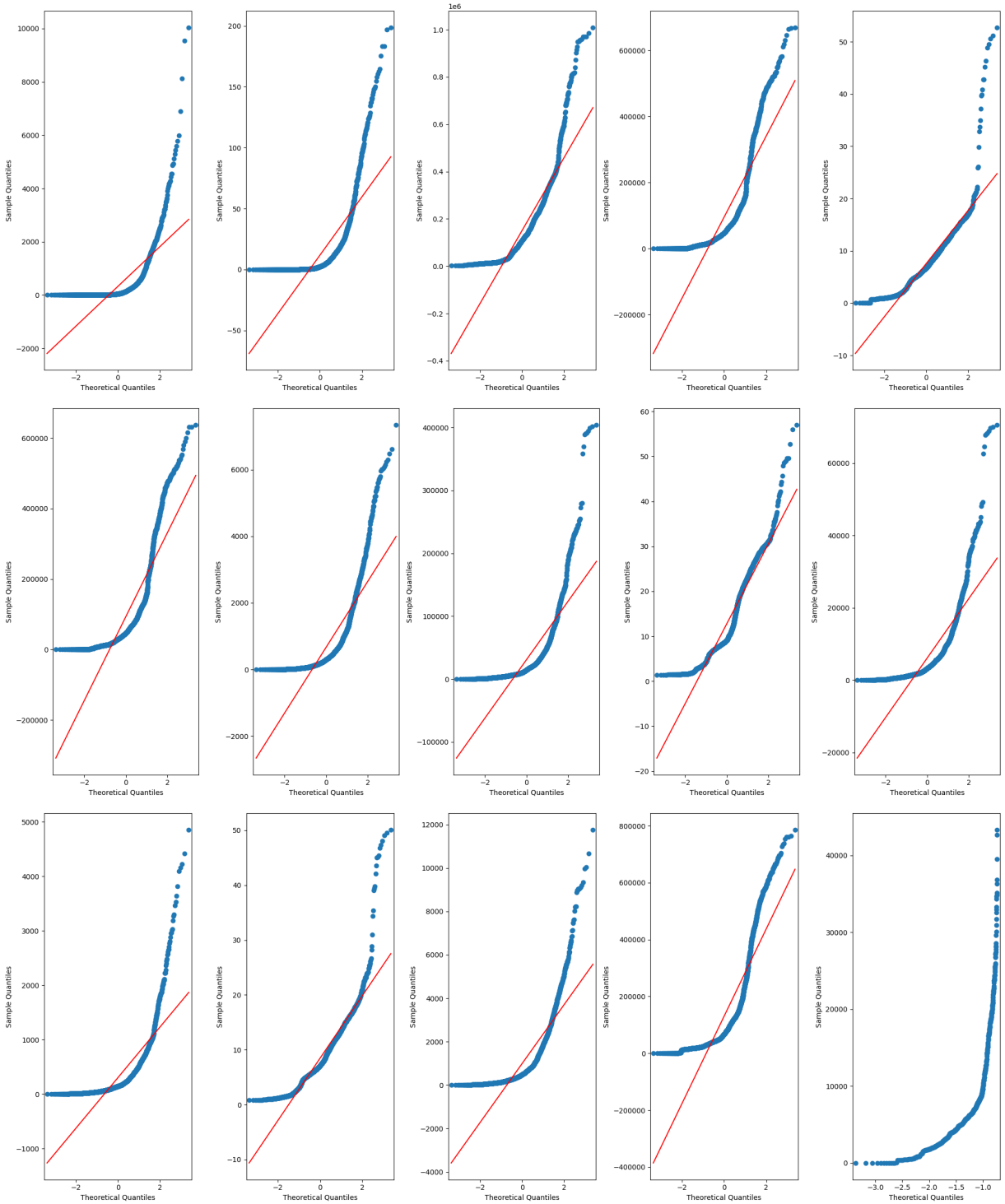
QQ plots

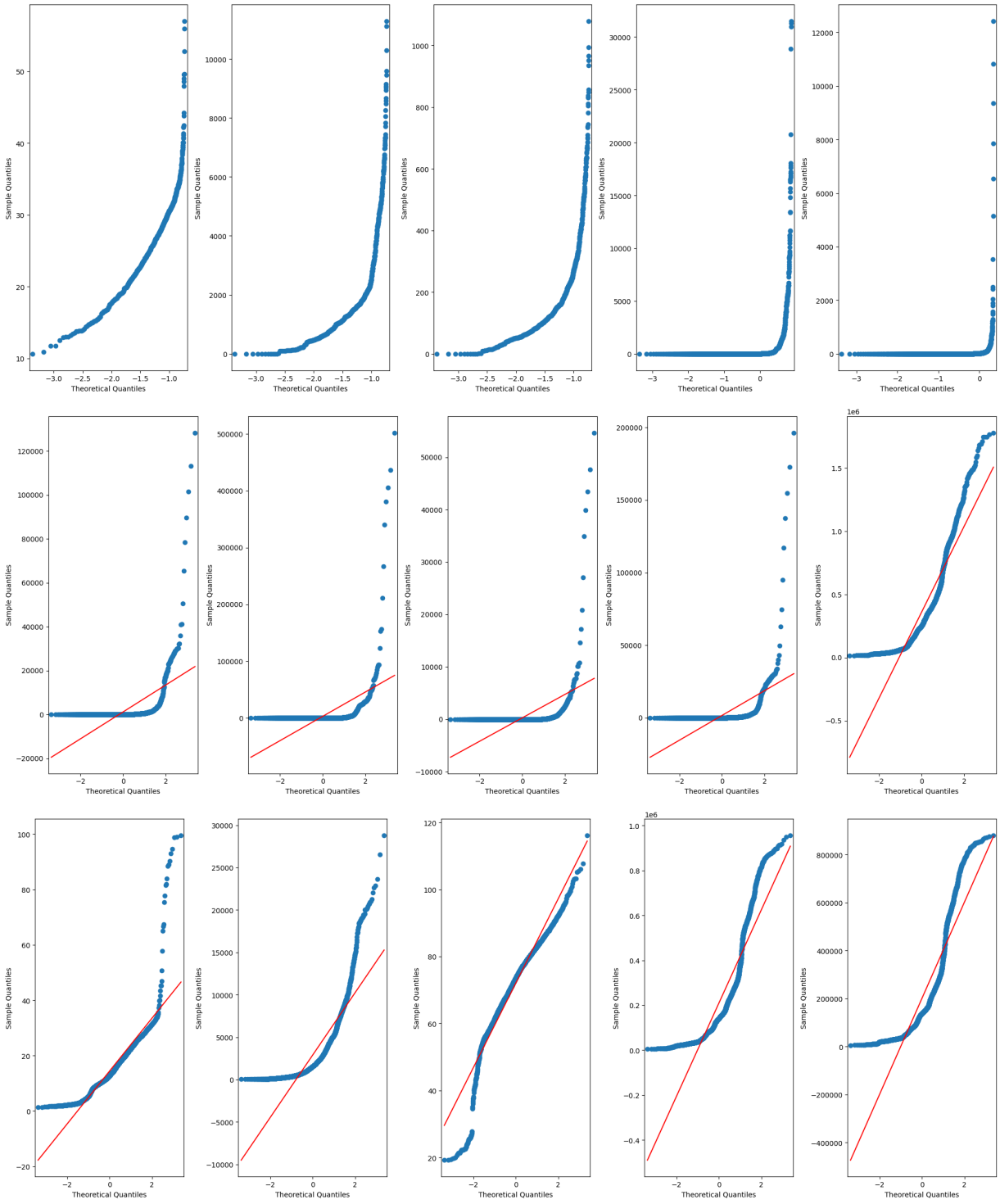
A `QQ plot` (a quantile quantile plot) helps assess if data comes from a normal distribution or not. The theoretical quantile is plotted against the sample quantiles. The theoretical follows a normal distribution and the sample does whatever it is going to do.

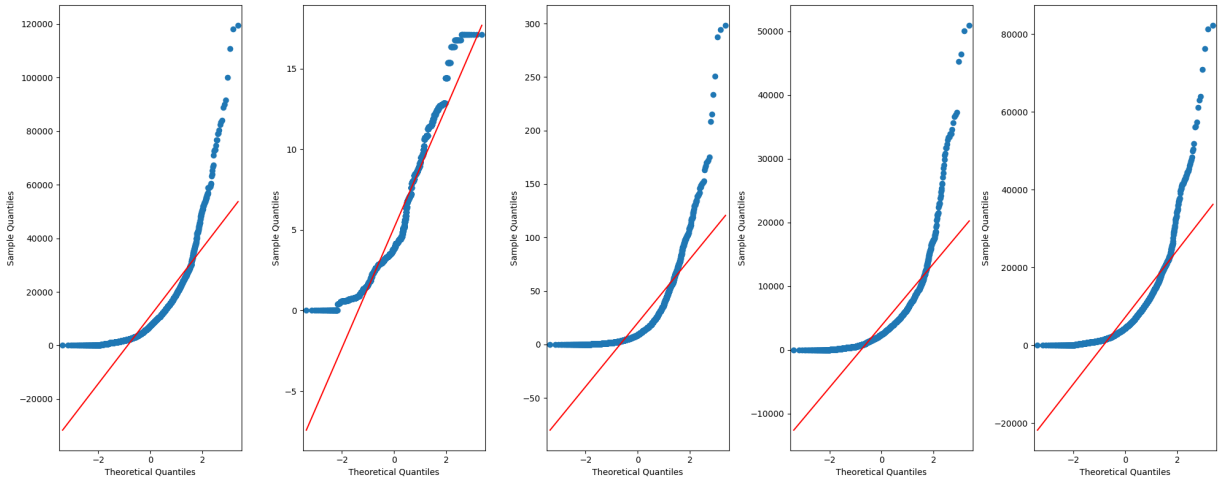
In [13]:
```
import statsmodels.api as sm
import pylab

figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[:5]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()

figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[5:10]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[10:15]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[15:20]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[20:25]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[25:30]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[30:35]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[35:40]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[40:45]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
figure, axes = plt.subplots(1, 5,figsize = (20,8))
for k,b in enumerate(needed_columns[45:50]):
        sm.qqplot(needed_columns1[b], line='s', ax = axes[k])
        plt.tight_layout()
```

```
plt.show()
#pylab.show()
```

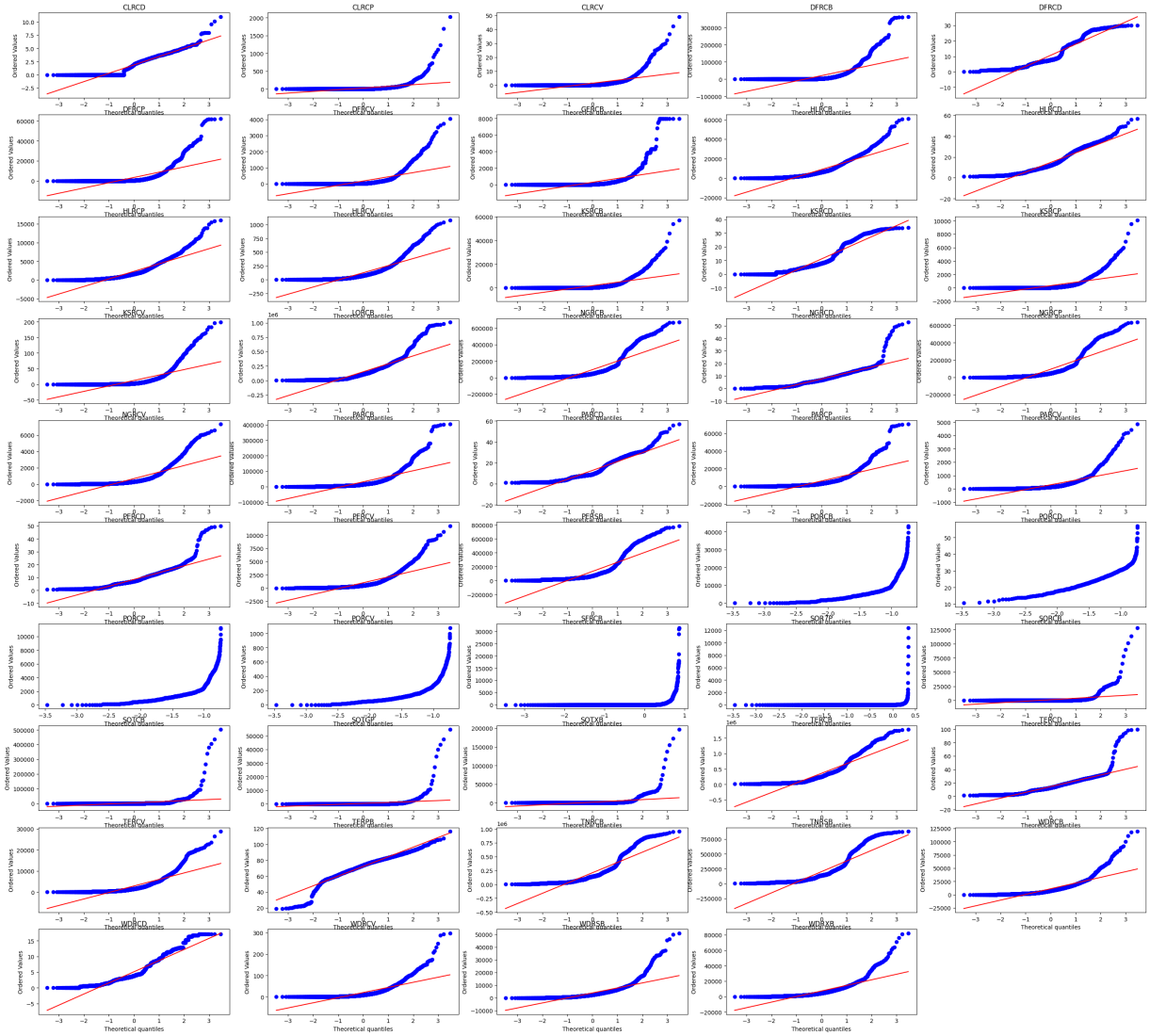



Sample Quantiles
Theoretical Quantiles
1e6


Sample Quantiles
Theoretical Quantiles
1e6



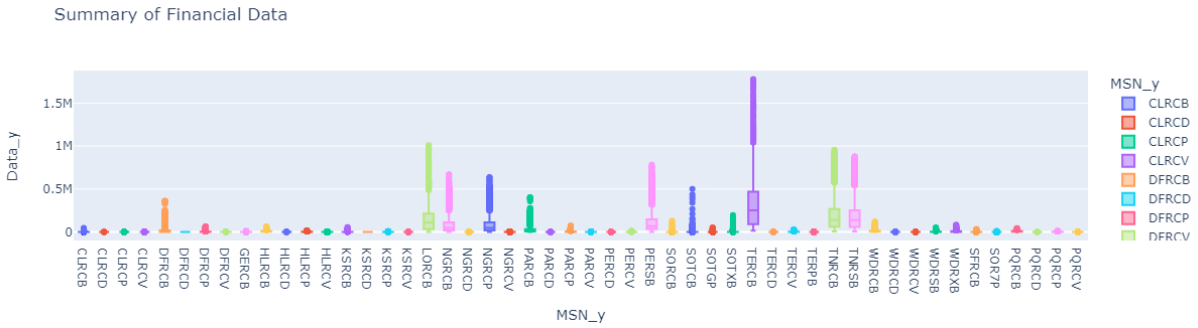
In [14]:
```
#This one works for plotting many plots. :)

fig = plt.figure(figsize=(35,35))
for i,j in enumerate(needed_columns):
    if 0 < i < len(needed_columns):
        ax = fig.add_subplot(11,5 ,i)
        stats.probplot(needed_columns1[j], plot=pylab)
        ax.set_title(j)
```



Box plots

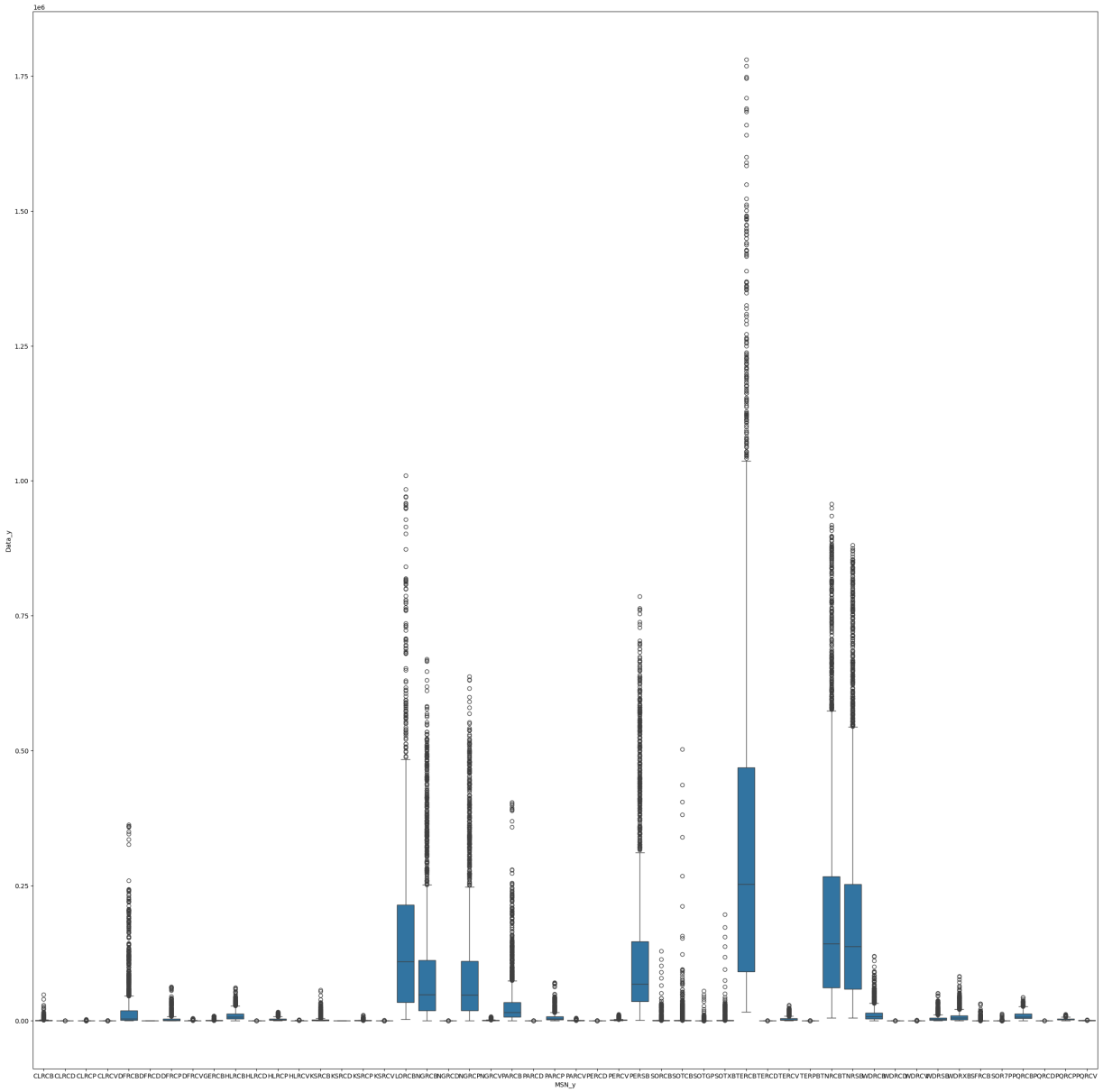
`Box plots` split the data into quartiles. And help illustrate outliers, which are points that lie outside of the 'whiskers' of each plot.

In [15]:
```
import plotly.express as px


fig = px.box(merged_data, x="MSN_y", y="Data_y", title = "Summary of Financial Data
fig.show()
```



In [16]:
```
plt.figure(figsize=(30,30))
sns.boxplot(y='Data_y', x='MSN_y',data=merged_data)
```

Out[16]: <Axes: xlabel='MSN_y', ylabel='Data_y'>



Correlation plots

`correlation plots` illustrate the linear relationship (the correlation) between pairs of variables. The lower the better for our purposes. Values that were not strongly correlated were chosen.

In [17]:
```
## Correlation Coefficients maybe
newlist=[]
for i in needed_columns:
    newlist.append(['Annual Spending',i,(np.corrcoef(pivoted_data['Data_x'], pivote
```

In [18]:
```
corr = needed_columns1.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[18]:

MSN_y	Data_x	CLRCB	CLRCD	CLRCP	CLRCV	DFRCB	DFRCD	DFR
MSN_y								
Data_x	1.000000	-0.072788	-0.170513	-0.078876	-0.061162	0.056089	0.396455	0.056
CLRCB	-0.072788	1.000000	0.053670	0.998730	0.887033	0.371776	-0.236037	0.371
CLRCD	-0.170513	0.053670	1.000000	0.057048	0.197500	0.212304	-0.428412	0.211
CLRCP	-0.078876	0.998730	0.057048	1.000000	0.887982	0.364692	-0.239744	0.364
CLRCV	-0.061162	0.887033	0.197500	0.887982	1.000000	0.451954	-0.259070	0.451
DFRCB	0.056089	0.371776	0.212304	0.364692	0.451954	1.000000	-0.148948	0.999
DFRCD	0.396455	-0.236037	-0.428412	-0.239744	-0.259070	-0.148948	1.000000	-0.147
DFRCP	0.056906	0.371251	0.211605	0.364168	0.451322	0.999995	-0.147804	1.000
DFRCV	0.235589	0.073415	0.104034	0.066387	0.150658	0.699226	0.170487	0.701
GERCB	0.550996	-0.078876	-0.227421	-0.080699	-0.095388	-0.097305	0.359133	-0.097
HLRCB	0.396933	0.132723	-0.065149	0.133677	0.141035	0.059766	0.025692	0.060
HLRCD	0.437919	-0.246352	-0.390784	-0.250435	-0.265746	-0.107317	0.883589	-0.105
HLRCP	0.396933	0.132722	-0.065142	0.133676	0.141036	0.059765	0.025692	0.060
HLRCV	0.702393	-0.097081	-0.238963	-0.101295	-0.098200	0.018750	0.519766	0.020
KSRCB	-0.005296	0.477871	0.162737	0.468458	0.518641	0.565322	-0.251606	0.564
KSRCD	0.387577	-0.212482	-0.421030	-0.216166	-0.235529	-0.143821	0.940827	-0.142
KSRCP	-0.005301	0.477876	0.162738	0.468463	0.518648	0.565319	-0.251610	0.564
KSRCV	0.167883	0.227044	0.235994	0.214413	0.356375	0.526673	-0.038081	0.527
LORCB	0.889537	0.040371	-0.046587	0.032839	0.068553	0.078305	0.198790	0.078
NGRCB	0.490798	0.276132	0.033306	0.271512	0.329543	0.345139	0.013182	0.345
NGRCD	0.350586	-0.231009	-0.210841	-0.237182	-0.248132	-0.086057	0.671601	-0.085
NGRCP	0.489276	0.277845	0.035024	0.273336	0.331390	0.346625	0.011143	0.346
NGRCV	0.709039	-0.001878	-0.044415	-0.008505	0.046016	0.246798	0.317860	0.247
PARCB	0.123462	0.408728	0.197602	0.401548	0.487479	0.979371	-0.154833	0.979
PARCD	0.470090	-0.252143	-0.462508	-0.256216	-0.285364	-0.212587	0.903863	-0.211
PARCP	0.157047	0.410110	0.185881	0.403197	0.487423	0.958812	-0.147730	0.958
PARCV	0.432954	0.040662	0.020073	0.032824	0.110817	0.611340	0.308846	0.613
PERCD	0.349117	-0.238167	-0.288315	-0.243148	-0.256805	-0.108179	0.750222	-0.107
PERCV	0.675721	0.012954	-0.028594	0.005329	0.072445	0.392940	0.347035	0.394

MSN_y	Data_x	CLRCB	CLRCD	CLRCP	CLRCV	DFRCB	DFRCD	DFR
MSN_y								
PERSB	0.442578	0.360994	0.090915	0.354782	0.428627	0.592300	-0.036662	0.592
PQRCB	0.505531	nan	nan	nan	nan	0.208702	0.024590	0.208
PQRCD	0.146870	nan	nan	nan	nan	0.296980	0.280794	0.296
PQRCP	0.505524	nan	nan	nan	nan	0.208698	0.024592	0.208
PQRCV	0.636325	nan	nan	nan	nan	0.378527	0.124343	0.378
SFRCB	-0.030983	0.220907	0.058134	0.228597	0.210588	0.041509	-0.102984	0.041
SOR7P	0.398255	-0.050123	-0.130422	-0.052330	-0.046660	-0.016851	0.110841	-0.016
SORCB	0.587282	-0.051096	-0.095725	-0.052008	-0.064043	-0.057064	0.163562	-0.056
SOTCB	0.493166	-0.041761	-0.132013	-0.042428	-0.053651	-0.048768	0.161866	-0.048
SOTGP	0.425230	-0.035255	-0.137015	-0.035783	-0.045937	-0.039415	0.149410	-0.039
SOTXB	0.557073	-0.049523	-0.118712	-0.050370	-0.062737	-0.044175	0.177259	-0.043
TERCB	0.827487	0.188318	0.001743	0.180315	0.238460	0.321087	0.129358	0.321
TERCD	0.393362	-0.242731	-0.337205	-0.247775	-0.265898	-0.158118	0.691941	-0.157
TERCV	0.962190	-0.047075	-0.132057	-0.054231	-0.016810	0.185156	0.410708	0.186
TERPB	-0.191315	0.070431	0.107444	0.075579	0.093933	-0.025994	-0.009223	-0.026
TNRCB	0.697009	0.279142	0.037505	0.271605	0.340504	0.468903	0.064542	0.469
TNRSB	0.690901	0.281705	0.035634	0.274285	0.341202	0.474086	0.064249	0.474
WDRCB	0.341193	0.186873	0.166931	0.177247	0.300460	0.396629	-0.029186	0.396
WDRCD	0.344160	-0.242023	-0.429108	-0.244224	-0.271323	-0.256105	0.929815	-0.255
WDRCV	0.566683	-0.086933	-0.169176	-0.092774	-0.069928	0.140683	0.500924	0.141
WDRSB	0.431048	0.083998	0.116793	0.073706	0.166616	0.460612	0.091944	0.461
WDRXB	0.257957	0.227315	0.179436	0.218981	0.347631	0.322760	-0.094934	0.322

In [19]:
```
fewer=corr[(corr <0.2)&(corr >-0.2)]
```

In [20]:
```
corr1=corr.reset_index(drop=True)
```

In [21]:
```
fewer=corr1[(corr1 <0.2)&(corr1 >-0.2)]
```

In [40]:
```
corr_result = corr.stack() #Using the corr function is neat
results_now=corr_result[(corr_result != 1.0)&((corr_result > 0.5))]
```

In [41]:
```
not_correlated_features=pd.DataFrame(results_now['Data_x']).reset_index()
list_of_features_to_keep=list(not_correlated_features['MSN_y'])
list_of_features_to_keep.append('Data_x')
list_of_features_to_keep.append('Year')
```

Now that I have a list of features that are going to be useful I need to compare them to each other

In [42]:
```
shorter_now = pivoted_data[list_of_features_to_keep]
```

In [43]:
```
corr1=shorter_now.corr()
corr1.style.background_gradient(cmap='coolwarm')
```

Out[43]:

MSN_y	GERCB	HLRCV	LORCB	NGRCV	PERCV	PQRCB	PQRCP	PQRCV	
MSN_y									
GERCB	1.000000	0.465418	0.475923	0.263634	0.241847	0.344824	0.344813	0.338650	0
HLRCV	0.465418	1.000000	0.587513	0.748496	0.753494	0.927203	0.927201	1.000000	0
LORCB	0.475923	0.587513	1.000000	0.602350	0.553225	0.469164	0.469157	0.571706	0
NGRCV	0.263634	0.748496	0.602350	1.000000	0.964325	0.705208	0.705210	0.794630	0
PERCV	0.241847	0.753494	0.553225	0.964325	1.000000	0.687663	0.687663	0.814705	0
PQRCB	0.344824	0.927203	0.469164	0.705208	0.687663	1.000000	1.000000	0.927203	0
PQRCP	0.344813	0.927201	0.469157	0.705210	0.687663	1.000000	1.000000	0.927201	0
PQRCV	0.338650	1.000000	0.571706	0.794630	0.814705	0.927203	0.927201	1.000000	0
SORCB	0.313937	0.281047	0.410756	0.348877	0.286140	0.184472	0.184468	0.276488	1
SOTXB	0.237203	0.295237	0.353191	0.373644	0.313125	0.196846	0.196844	0.290060	0
TERCB	0.386482	0.620208	0.920772	0.765445	0.730325	0.635219	0.635214	0.738802	0
TERCV	0.481953	0.778879	0.838221	0.861516	0.850960	0.612334	0.612329	0.753389	0
TNRCB	0.280581	0.581310	0.768012	0.808715	0.787588	0.726347	0.726345	0.821830	0
TNRSB	0.276209	0.582232	0.764463	0.809292	0.789310	0.727324	0.727322	0.821503	0
WDRCV	0.223608	0.659033	0.445027	0.658636	0.697224	0.503556	0.503559	0.618173	0
Data_x	0.550996	0.702393	0.889537	0.709039	0.675721	0.505531	0.505524	0.636325	0
Year	0.360293	0.508427	0.227795	0.321533	0.326523	0.021179	0.021181	-0.033843	0

In [44]:
```
list_of_features_to_keep.append('StateCode')
```

For now going to apply model

In [45]:
```
shorter_now = pivoted_data[list_of_features_to_keep]
```

In [46]:
```
shorter_now= shorter_now.fillna(0)
```

In [47]:
```
#selected_column
shorter_now['Year1']=shorter_now['Year']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(shorter_now['Year1'])
shorter_now['Year1'] =le.transform(shorter_now['Year1'])
le = LabelEncoder()
le.fit(shorter_now['StateCode'])
shorter_now['StateCode'] = le.transform(shorter_now['StateCode'])
```

In [48]:
```
# shift column 'Name' to first position
first_column = shorter_now.pop('Year')

# insert column using insert(position,column_name,
# first_column) function
shorter_now.insert(0, 'Year', first_column)
```

In [49]:
```
# shift column 'Name' to first position
first_column = shorter_now.pop('Data_x')

# insert column using insert(position,column_name,
# first_column) function
shorter_now.insert(0, 'Data_x', first_column)
```

In [50]:
```
shorter_now
```

Out[50]:

	MSN_y	Data_x	Year	GERCB	HLRCV	LORCB	NGRCV	PERCV	PQRCB	PQRCP	PQRCV
0	16.70	1970-01-01	0.00	0.60	7074.00	9.40	22.20	0.00	0.00	0.0(
1	20.10	1971-01-01	0.00	0.70	8316.00	10.60	27.70	0.00	0.00	0.0(
2	22.00	1972-01-01	0.00	0.80	8963.00	13.00	27.00	0.00	0.00	0.0(
3	22.30	1973-01-01	0.00	1.80	9488.00	7.90	27.40	0.00	0.00	0.0(
4	25.30	1974-01-01	0.00	1.20	10533.00	6.60	36.20	0.00	0.00	0.0(
...	...	...	...	...	...	...	...	...	...	...	..
2647	315.30	2017-01-01	70.00	82.10	20534.00	113.10	216.30	3505.00	913.00	82.1(
2648	310.20	2018-01-01	70.00	80.00	20438.00	113.00	216.40	3275.00	853.00	80.0(
2649	318.40	2019-01-01	70.00	74.20	21090.00	113.30	210.10	3513.00	915.00	74.2(
2650	319.90	2020-01-01	70.00	54.70	21385.00	114.60	186.30	2994.00	779.00	54.7(
2651	323.70	2021-01-01	70.00	82.10	21307.00	129.00	231.40	3351.00	873.00	82.1(

2652 rows × 19 columns

In [51]:
```
X1 =shorter_now.loc[shorter_now['Year']<'2021-01-01']
X2 = shorter_now.loc[shorter_now['Year']>='2021-01-01']
```

In [52]:
```
y_train=X1['Data_x'].reset_index(drop=True) # Separating all of the data
y_test=X2['Data_x'].reset_index(drop=True)
X_train = X1.iloc[:,2: ]
X_test = X2.iloc[:,2: ]
```

In [53]:
```
X_train
```

Out[53]:

	MSN_y	GERCB	HLRCV	LORCB	NGRCV	PERCV	PQRCB	PQRCP	PQRCV	SORCB	SOTX
0		0.00	0.60	7074.00	9.40	22.20	0.00	0.00	0.00	0.00	0.(
1		0.00	0.70	8316.00	10.60	27.70	0.00	0.00	0.00	0.00	0.(
2		0.00	0.80	8963.00	13.00	27.00	0.00	0.00	0.00	0.00	0.(
3		0.00	1.80	9488.00	7.90	27.40	0.00	0.00	0.00	0.00	0.(
4		0.00	1.20	10533.00	6.60	36.20	0.00	0.00	0.00	0.00	0.(
...		...	...	...	...	...	...	...	...	...	
2646		70.00	51.50	20329.00	102.10	169.70	2597.00	676.00	51.50	20.00	33.(
2647		70.00	82.10	20534.00	113.10	216.30	3505.00	913.00	82.10	28.00	45.(
2648		70.00	80.00	20438.00	113.00	216.40	3275.00	853.00	80.00	41.00	59.(
2649		70.00	74.20	21090.00	113.30	210.10	3513.00	915.00	74.20	61.00	84.(
2650		70.00	54.70	21385.00	114.60	186.30	2994.00	779.00	54.70	91.00	116.(

2601 rows × 17 columns

In [54]:
```
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
scaler = StandardScaler()
scaler.fit(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [55]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
linear_model1=linear_model.LinearRegression()
linear_model=LinearRegression().fit(X_train,y_train)
y_pred = linear_model.predict(X_test)
y_true =y_test
```

In [58]:
```
results_trial1=pd.DataFrame(y_pred,y_true).reset_index()
```

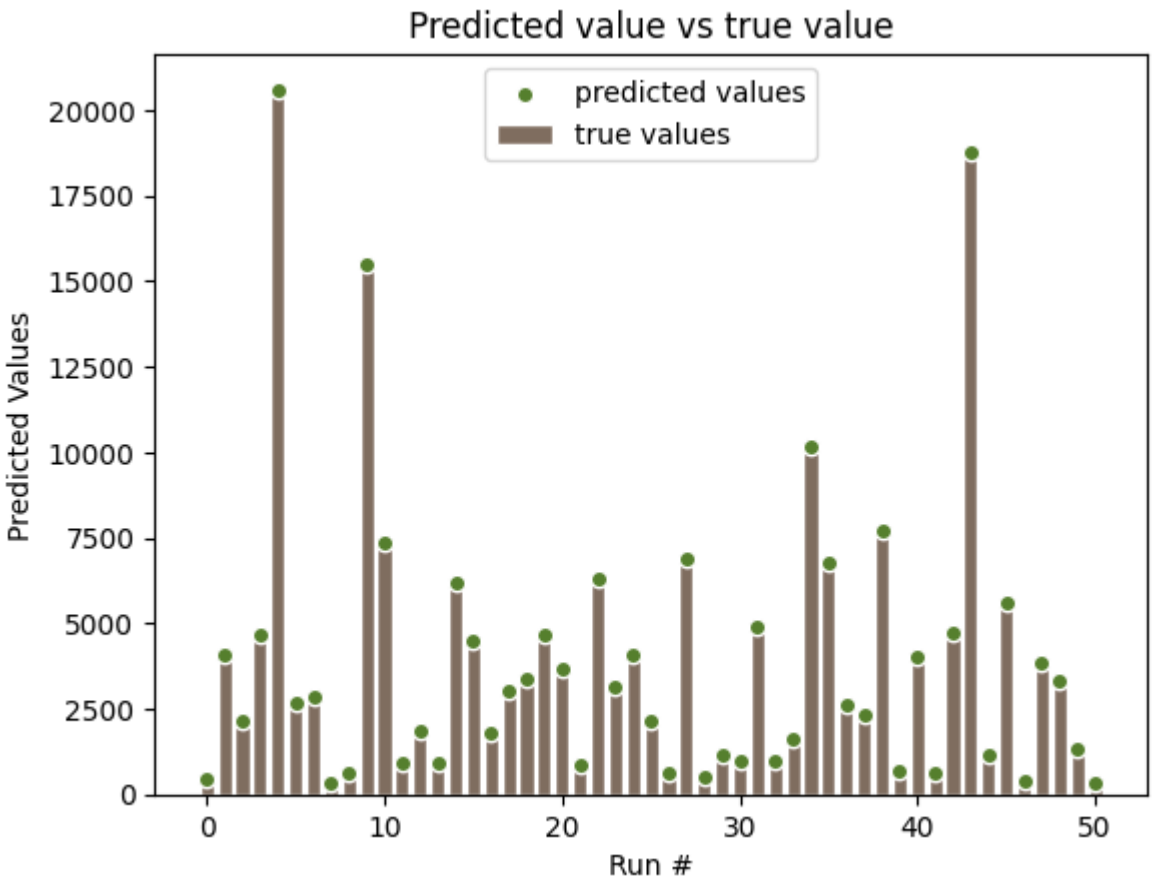
In [74]:
```
results_trial1=pd.DataFrame(y_pred,y_true).reset_index()
results_trial1=results_trial1.rename(columns={'Data_x':'true',0:'predicted'})
results_trial1['how_off']= ((results_trial1['true']-results_trial1['predicted'])/re
results_trial1.head(1)
```

Out[74]:

	true	predicted	how_off
0	469.90	469.90	-0.00

In [73]:
```
r1 = np.arange(len(results_trial1['true']))

plt.bar(r1, results_trial1['true'], color='#7f6d5f', edgecolor='white', label='true
plt.scatter(r1, results_trial1['predicted'], color='#557f2d', edgecolor='white', la
plt.xlabel('Run #')
plt.ylabel('Predicted Values')
plt.title('Predicted value vs true value')

plt.legend()
```

Out[73]: <matplotlib.legend.Legend at 0x183904c3f90>



Results thus far and discussion

The linear model performed very well. With a prediction accuracy of 100%. But this problem was ideal for linear modeling, being a combination of continuous variables.

In [ ]: