

Importing relevant libraries

```
In [1]: # !pip install plotly==5.6.0
```

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
sns.set()
import warnings
warnings.filterwarnings('ignore')
```

Import the data

```
In [3]: '''from google.colab import drive
drive.mount('/content/drive')'''
```

```
Out[3]: "from google.colab import drive\n drive.mount('/content/drive')"
```

```
In [4]: df = pd.read_csv("Energy_data.csv")
df.head()
```

```
Out[4]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
0	1	631.623161	1246.300847	-0.400000	64.000000	7.600000	82.000000
1	2	534.397104	1062.500558	-0.733333	65.333333	7.733333	78.666667
2	3	453.538784	884.586887	-1.066667	66.666667	7.866667	75.333333
3	4	400.699718	786.564121	-1.400000	68.000000	8.000000	72.000000
4	5	378.171092	742.669614	-1.666667	60.333333	8.333333	69.666667

```
In [5]: df.tail()
```

```
Out[5]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
8779	8780	950.369306	0.0	3.333333	64.0	NaN	NaN
8780	8781	880.138770	0.0	2.666667	68.0	NaN	NaN
8781	8782	792.754026	0.0	2.000000	72.0	NaN	NaN
8782	8783	740.446668	0.0	1.333333	76.0	NaN	NaN
8783	8784	706.176769	0.0	0.666667	80.0	NaN	NaN

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Hour                  8784 non-null   int64
1   energy_consumpt_2005  8750 non-null   float64
2   energy_consumpt_2006  8742 non-null   float64
3   full_temp_2005        8784 non-null   float64
4   full_humid_2005       8784 non-null   float64
5   full_temp_2006        8760 non-null   float64
6   full_humid_2006       8760 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 480.5 KB
```

```
In [7]: df.describe()
```

```
Out[7]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
count	8784.000000	8750.000000	8742.000000	8784.000000	8784.000000	8760.000000	8760.000000
mean	4392.500000	488.754321	1146.214842	17.602836	62.17270	18.197500	61.505023
std	2535.866716	167.554212	354.221796	8.031887	18.47477	7.713689	19.454944
min	1.000000	-614.175490	-1989.430430	-3.200000	3.000000	0.000000	4.000000
25%	2196.750000	372.264578	877.224805	11.800000	48.000000	12.066667	46.000000
50%	4392.500000	493.915280	1175.396289	17.400000	65.000000	18.600000	64.000000
75%	6588.250000	571.766044	1350.082898	23.600000	78.000000	24.200000	77.000000
max	8784.000000	6560.013773	14771.529320	41.000000	100.000000	36.400000	100.000000

```
In [8]: df.columns
```

```
Out[8]: Index(['Hour', 'energy_consumpt_2005', 'energy_consumpt_2006',  
              'full_temp_2005', 'full_humid_2005', 'full_temp_2006',  
              'full_humid_2006'],  
             dtype='object')
```

```
In [9]: df.shape
```

```
Out[9]: (8784, 7)
```

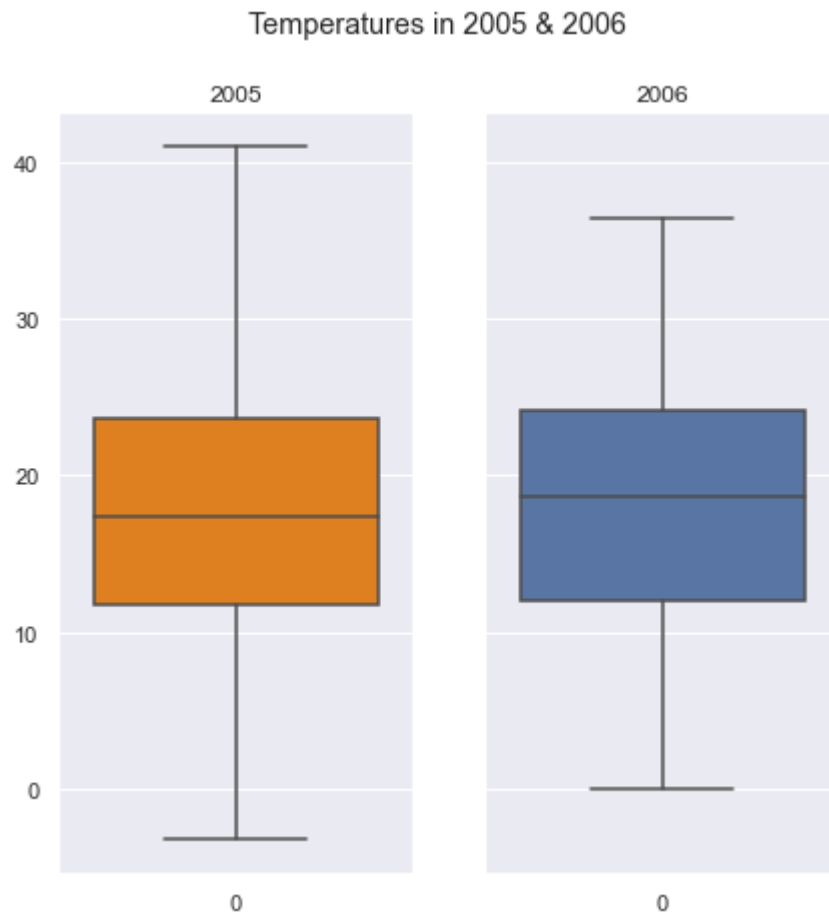
Graphics to visualize the data

Temperature

```
In [10]: fig, (ax1, ax2) = plt.subplots(1, 2, sharey = True, figsize = (7,7))
fig.suptitle("Temperatures in 2005 & 2006")

sns.boxplot(ax = ax1, data = df['full_temp_2005'], palette = "autumn")
ax1.set_title('2005')

sns.boxplot(ax = ax2, data = df['full_temp_2006'])
ax2.set_title('2006')
plt.show()
```



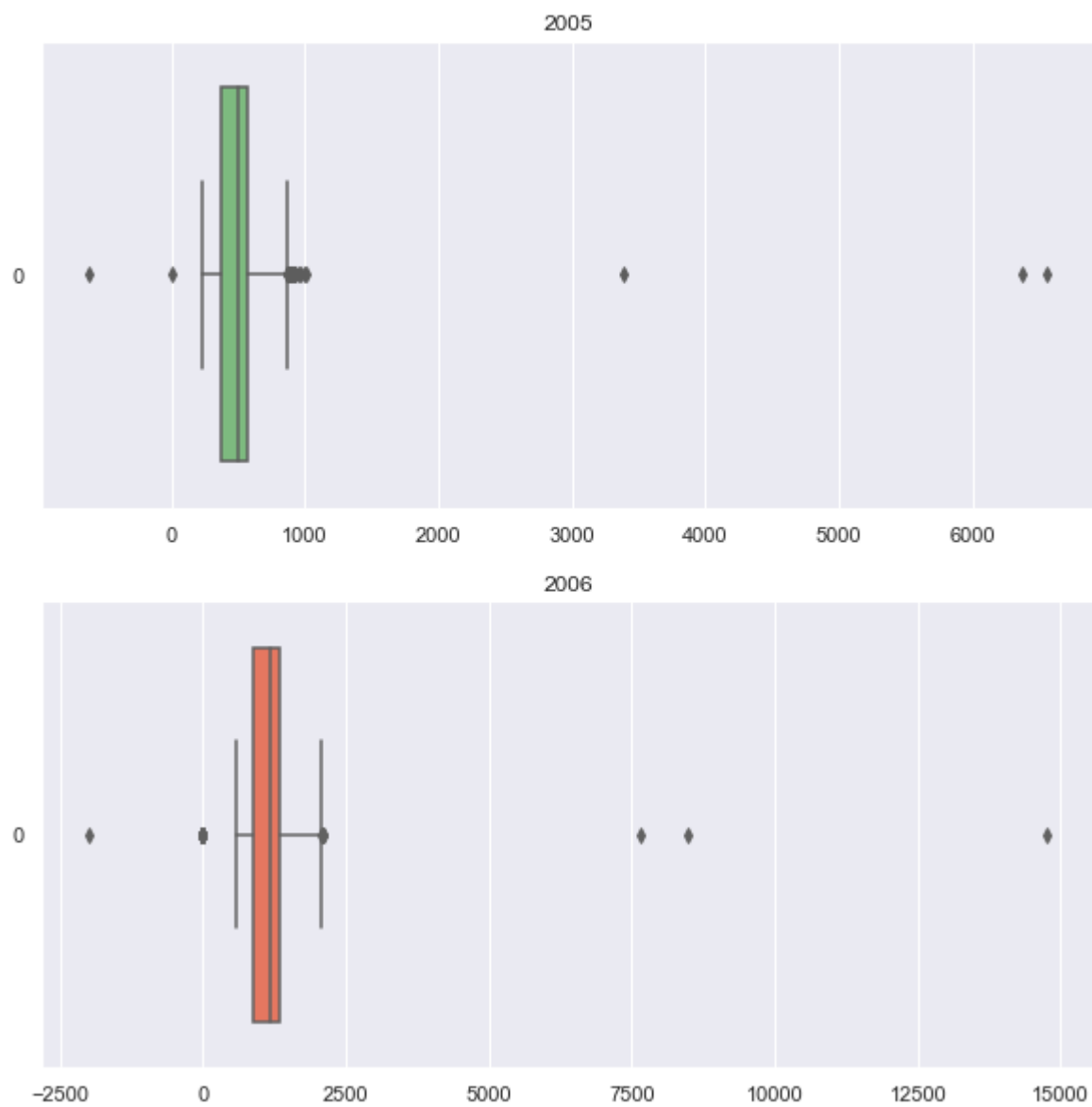
Energy consumpt

```
In [11]: fig, (ax1, ax2) = plt.subplots(2, sharey = True, figsize = (10,10))
fig.suptitle("Energy consumpt in 2005 & 2006")

sns.boxplot(ax = ax1, data = df['energy_consumpt_2005'], orient = "h", palette = "Greens")
ax1.set_title('2005')

sns.boxplot(ax = ax2, data = df['energy_consumpt_2006'], orient = "h", palette = "Reds")
ax2.set_title('2006')
plt.show()
```

Energy consumpt in 2005 & 2006

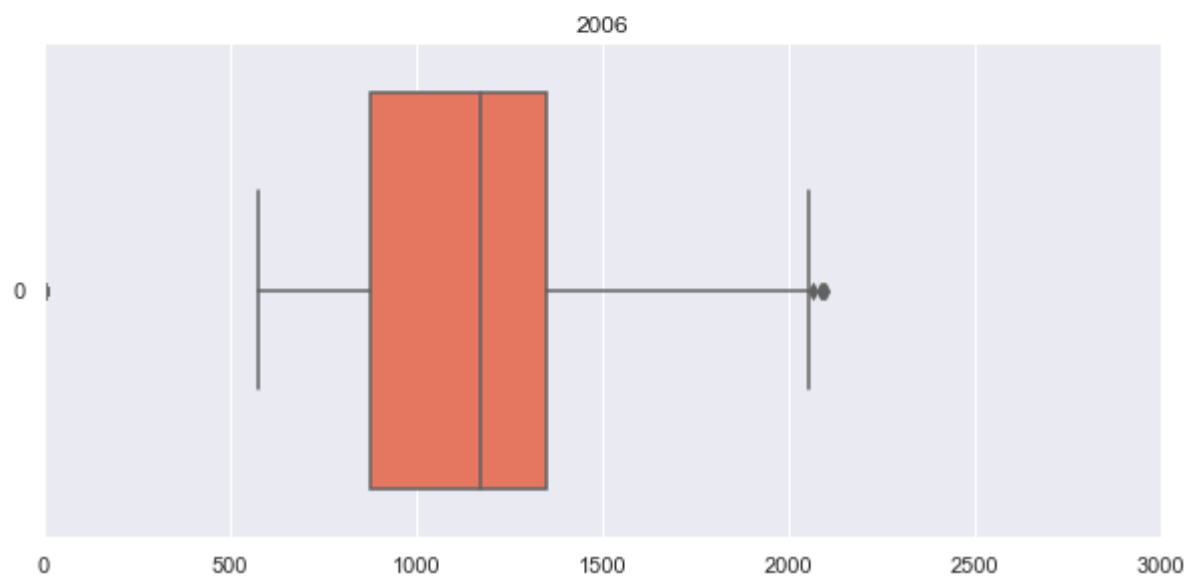
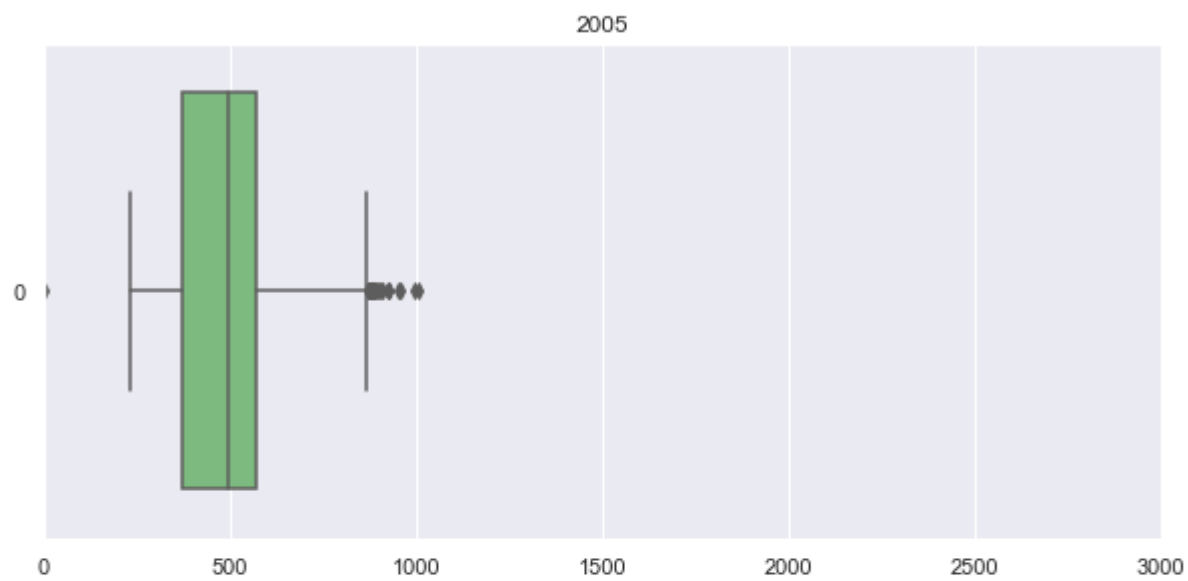


```
In [12]: fig, (ax1, ax2) = plt.subplots(2, sharey = True, figsize = (10,10))
fig.suptitle("Energy consumpt in 2005 & 2006 with a range of (0,3000)")

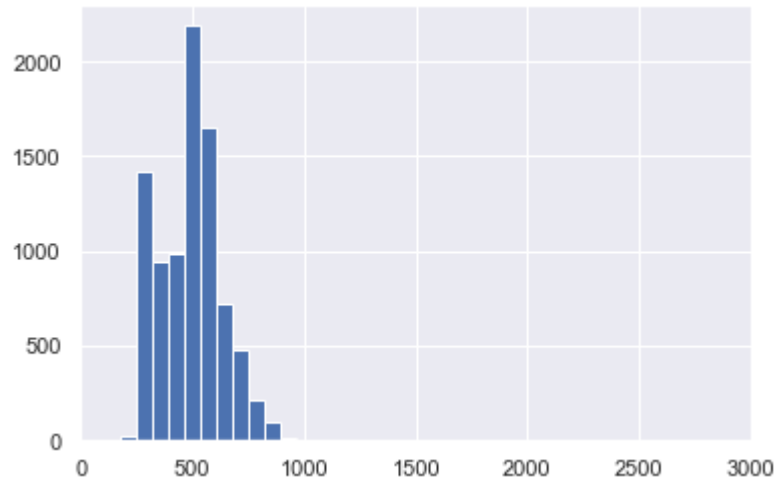
sns.boxplot(ax = ax1, data = df['energy_consumpt_2005'], orient = "h", palette = "Greens")
ax1.set_title('2005')
ax1.set_xlim([0,3000])

sns.boxplot(ax = ax2, data = df['energy_consumpt_2006'], orient = "h", palette = "Reds")
ax2.set_title('2006')
ax2.set_xlim([0,3000])
plt.show()
```

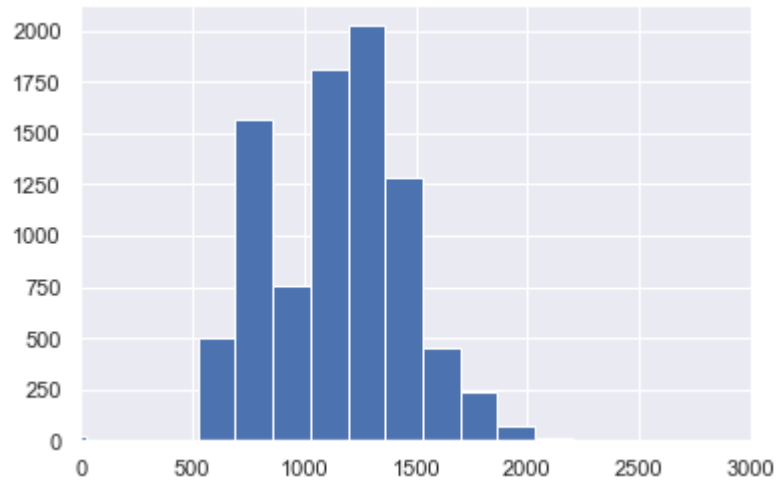

Energy consumpt in 2005 & 2006 with a range of (0,3000)



```
In [13]: plt.hist(df['energy_consumpt_2005'], bins = 100)
plt.xlim(0,3000)
plt.show()
```



```
In [14]: plt.hist(df['energy_consumpt_2006'], bins = 100)
plt.xlim(0,3000)
plt.show()
```

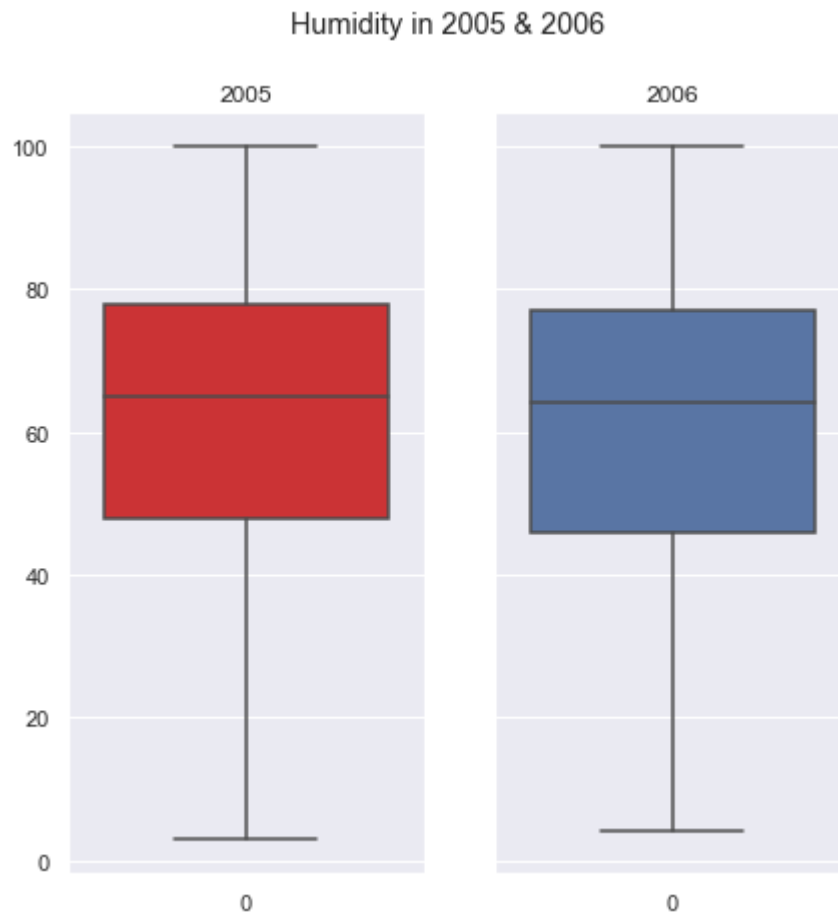


Humidity

```
In [15]: fig, (ax1, ax2) = plt.subplots(1, 2, sharey = True, figsize = (7,7))
fig.suptitle("Humidity in 2005 & 2006")

sns.boxplot(ax = ax1, data = df['full_humid_2005'], palette = "Set1")
ax1.set_title('2005')

sns.boxplot(ax = ax2, data = df['full_humid_2006'])
ax2.set_title('2006')
plt.show()
```



Missing data

We count the missing data

```
In [16]: print("We have this missing data by category")
df.isna().sum()
```

We have this missing data by category

```
Out[16]: Hour      0
energy_consumpt_2005  34
energy_consumpt_2006  42
full_temp_2005      0
full_humid_2005     0
full_temp_2006     24
full_humid_2006     24
dtype: int64
```

```
In [17]: rows, columns = df.shape
cell_count = rows * columns
number_of_nulls = df.isnull().sum().sum()
print("number of nulls: ", number_of_nulls)
percentage_of_missing = (number_of_nulls / cell_count) * 100
print(f'Percentage of missing values: {percentage_of_missing}%')
```

```
number of nulls: 124
Percentage of missing values: 0.20166536559979184%
```

Missing data

```
In [18]: is_NaN = df.isnull()
row_has_NaN = is_NaN.any(axis=1)
rows_with_NaN = df[row_has_NaN]
pd.set_option("display.max_rows", None, "display.max_columns", None)
rows_with_NaN.head()
```

Out[18]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
54	55	410.042675	NaN	10.000000	70.000000	5.600000	82.000000
87	88	NaN	1830.321544	10.000000	88.000000	12.400000	37.000000
118	119	600.571310	NaN	12.933333	85.000000	9.000000	72.000000
144	145	NaN	994.820593	11.400000	64.000000	7.000000	87.000000
172	173	313.304967	NaN	6.866667	75.333333	7.666667	86.666667

```
In [19]: rows_with_NaN.shape
```

Out[19]: (100, 7)

```
In [20]: pd.set_option("display.max_rows", 10, "display.max_columns", None)
rows_with_NaN
```

Out[20]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
54	55	410.042675	NaN	10.000000	70.000000	5.600000	82.000000
87	88	NaN	1830.321544	10.000000	88.000000	12.400000	37.000000
118	119	600.571310	NaN	12.933333	85.000000	9.000000	72.000000
144	145	NaN	994.820593	11.400000	64.000000	7.000000	87.000000
172	173	313.304967	NaN	6.866667	75.333333	7.666667	86.666667
...
8779	8780	950.369306	0.000000	3.333333	64.000000	NaN	NaN
8780	8781	880.138770	0.000000	2.666667	68.000000	NaN	NaN
8781	8782	792.754026	0.000000	2.000000	72.000000	NaN	NaN
8782	8783	740.446668	0.000000	1.333333	76.000000	NaN	NaN
8783	8784	706.176769	0.000000	0.666667	80.000000	NaN	NaN

100 rows × 7 columns

```
In [21]: df.describe()
```

```
Out[21]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
count	8784.000000	8750.000000	8742.000000	8784.000000	8784.000000	8760.000000	8760.000000
mean	4392.500000	488.754321	1146.214842	17.602836	62.17270	18.197500	61.505023
std	2535.866716	167.554212	354.221796	8.031887	18.47477	7.713689	19.454944
min	1.000000	-614.175490	-1989.430430	-3.200000	3.000000	0.000000	4.000000
25%	2196.750000	372.264578	877.224805	11.800000	48.000000	12.066667	46.000000
50%	4392.500000	493.915280	1175.396289	17.400000	65.000000	18.600000	64.000000
75%	6588.250000	571.766044	1350.082898	23.600000	78.000000	24.200000	77.000000
max	8784.000000	6560.013773	14771.529320	41.000000	100.000000	36.400000	100.000000

Filling the missing data

We fill the missing data with the previous value, if we filled it with the mean, it could be biased with outliers

```
In [22]: df['energy_consumpt_2005'].fillna(method = "ffill", inplace = True)
df['energy_consumpt_2006'].fillna(method = "ffill", inplace = True)
df['full_temp_2006'].fillna(method = "ffill", inplace = True)
df['full_humid_2006'].fillna(method = "ffill", inplace = True)
```

```
In [23]: print("There is no left missing data:")
df.isna().sum()
```

There is no left missing data:

```
Out[23]: Hour          0
energy_consumpt_2005  0
energy_consumpt_2006  0
full_temp_2005        0
full_humid_2005       0
full_temp_2006        0
full_humid_2006       0
dtype: int64
```



```
In [24]: df.describe()
```

```
Out[24]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
count	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000
mean	4392.500000	488.994931	1145.741565	17.602836	62.17270	18.15452	61.561020
std	2535.866716	167.598644	354.191947	8.031887	18.47477	7.74679	19.457781
min	1.000000	-614.175490	-1989.430430	-3.200000	3.00000	0.00000	4.000000
25%	2196.750000	372.480527	875.632039	11.800000	48.00000	12.00000	46.000000
50%	4392.500000	494.026229	1175.206198	17.400000	65.00000	18.60000	64.000000
75%	6588.250000	571.826328	1349.877680	23.600000	78.00000	24.20000	77.000000
max	8784.000000	6560.013773	14771.529320	41.000000	100.00000	36.40000	100.000000

Dealing with outliers

Identifying outliers

```
In [25]: Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Hour          4391.500000
energy_consumpt_2005    199.34580
energy_consumpt_2006    474.24564
full_temp_2005         11.80000
full_humid_2005         30.00000
full_temp_2006         12.20000
full_humid_2006         31.00000
dtype: float64
```

```
In [26]: fig, (ax1, ax2) = plt.subplots(2, sharey = True, figsize = (10,10))
fig.suptitle("Energy consumpt in 2005 & 2006")

sns.boxplot(ax = ax1, data = df['energy_consumpt_2005'], orient = "h", palette = "Greens")
ax1.set_title('2005')

sns.boxplot(ax = ax2, data = df['energy_consumpt_2006'], orient = "h", palette = "Reds")
ax2.set_title('2006')
```

```
Out[26]: Text(0.5, 1.0, '2006')
```



```
In [27]: df[df['energy_consumpt_2005'] > 3000]
```

Out[27]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
317	318	3378.521357	850.109412	4.933333	80.666667	6.600000	72.666667
502	503	6364.455548	1325.593826	3.633333	88.000000	6.933333	82.666667
849	850	6560.013773	1369.922506	8.000000	30.000000	9.400000	53.000000

```
In [28]: df[df['energy_consumpt_2006'] > 3000]
```

Out[28]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
412	413	324.136364	7642.733311	0.666667	56.333333	6.466667	59.333333
576	577	780.490339	8480.724131	1.600000	75.000000	3.000000	60.000000
905	906	751.734780	14771.529320	11.466667	61.666667	11.800000	81.666667

We swap the values over 3000 with their previous value

```
In [29]: df["energy_consumpt_2005"] = df["energy_consumpt_2005"].mask(df["energy_consumpt_2005"] > 3000, np.nan)
df["energy_consumpt_2006"] = df["energy_consumpt_2006"].mask(df["energy_consumpt_2006"] > 3000, np.nan)
df['energy_consumpt_2005'].fillna(method = "ffill", inplace = True)
df['energy_consumpt_2006'].fillna(method = "ffill", inplace = True)
```

We swap the values under 500 with their previous value

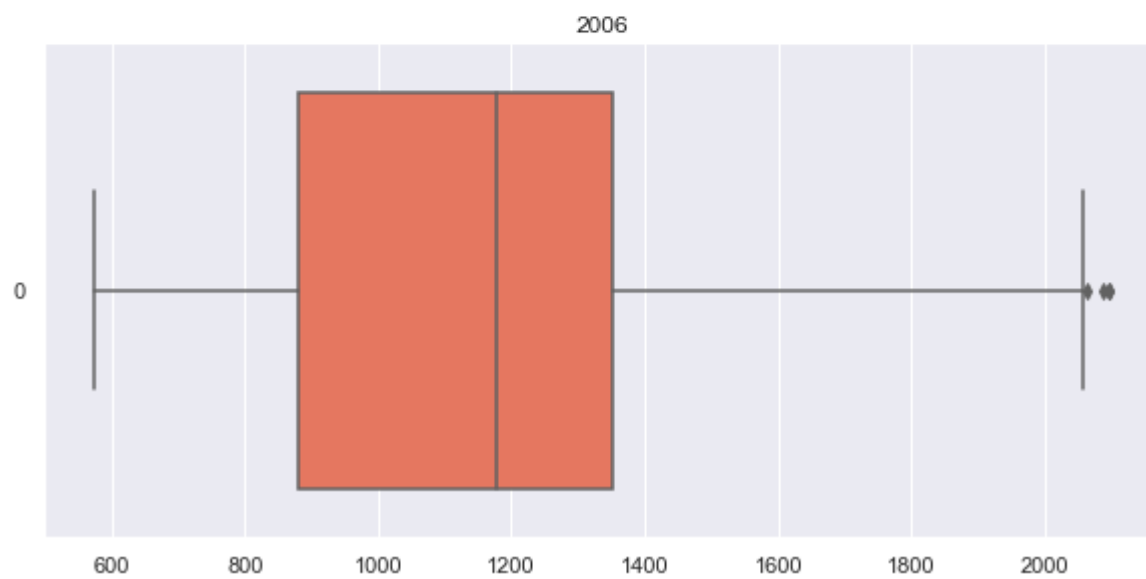
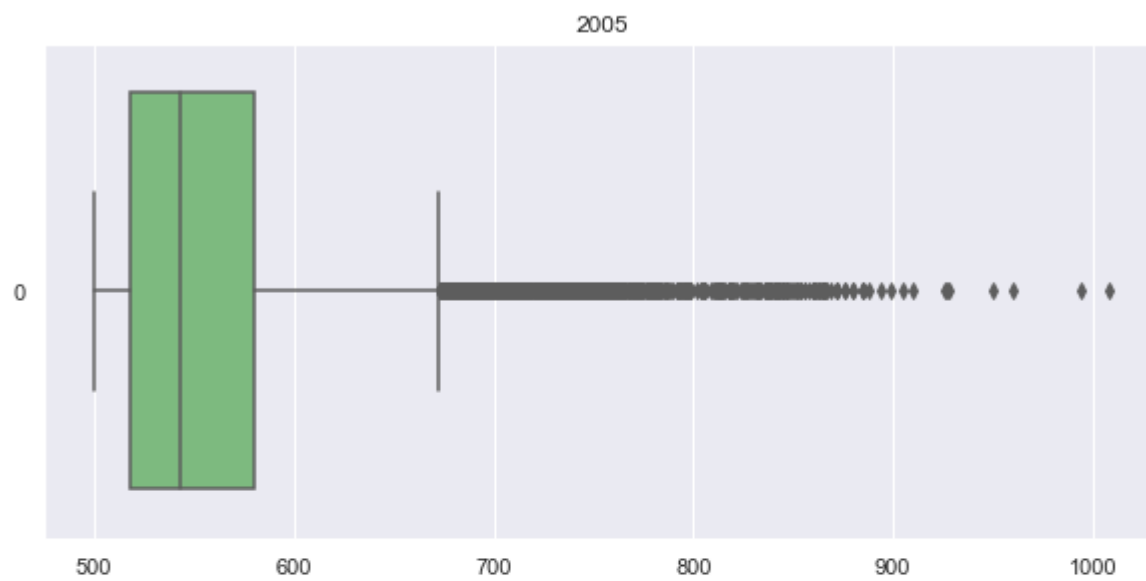
```
In [30]: df["energy_consumpt_2005"] = df["energy_consumpt_2005"].mask(df["energy_consumpt_2005"] < 500, np.nan)
df['energy_consumpt_2005'].fillna(method = "ffill", inplace = True)
df["energy_consumpt_2006"] = df["energy_consumpt_2006"].mask(df["energy_consumpt_2006"] < 500, np.nan)
df['energy_consumpt_2006'].fillna(method = "ffill", inplace = True)
```

```
In [31]: fig, (ax1, ax2) = plt.subplots(2, sharey = True, figsize = (10,10))
fig.suptitle("Energy consumpt in 2005 & 2006")

sns.boxplot(ax = ax1, data = df['energy_consumpt_2005'], orient = "h", palette = "Greens")
ax1.set_title('2005')

sns.boxplot(ax = ax2, data = df['energy_consumpt_2006'], orient = "h", palette = "Reds")
ax2.set_title('2006')
plt.show()
```

Energy consumpt in 2005 & 2006



In [32]: df

Out[32]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
0	1	631.623161	1246.300847	-0.400000	64.000000	7.600000	82.000000
1	2	534.397104	1062.500558	-0.733333	65.333333	7.733333	78.666667
2	3	534.397104	884.586887	-1.066667	66.666667	7.866667	75.333333
3	4	534.397104	786.564121	-1.400000	68.000000	8.000000	72.000000
4	5	534.397104	742.669614	-1.666667	60.333333	8.333333	69.666667
...
8779	8780	950.369306	1384.624630	3.333333	64.000000	2.466667	82.000000
8780	8781	880.138770	1384.624630	2.666667	68.000000	2.466667	82.000000
8781	8782	792.754026	1384.624630	2.000000	72.000000	2.466667	82.000000
8782	8783	740.446668	1384.624630	1.333333	76.000000	2.466667	82.000000
8783	8784	706.176769	1384.624630	0.666667	80.000000	2.466667	82.000000

8784 rows × 7 columns

```
In [33]: df.describe()
```

Out[33]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
count	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000
mean	4392.500000	565.692359	1146.798319	17.602836	62.17270	18.15452	61.561020
std	2535.866716	71.796206	298.178305	8.031887	18.47477	7.74679	19.457781
min	1.000000	500.012673	574.705971	-3.200000	3.000000	0.000000	4.000000
25%	2196.750000	517.865582	880.807078	11.800000	48.000000	12.000000	46.000000
50%	4392.500000	542.736777	1176.682996	17.400000	65.000000	18.600000	64.000000
75%	6588.250000	579.843872	1351.703523	23.600000	78.000000	24.200000	77.000000
max	8784.000000	1008.915444	2097.502137	41.000000	100.000000	36.400000	100.000000

<https://www.pluralsight.com/guides/cleaning-up-data-from-outliers> (<https://www.pluralsight.com/guides/cleaning-up-data-from-outliers>)

<https://towardsdatascience.com/data-cleaning-using-python-pandas-f6fadc433535> (<https://towardsdatascience.com/data-cleaning-using-python-pandas-f6fadc433535>)

<https://statisticsbyjim.com/basics/remove-outliers/> (<https://statisticsbyjim.com/basics/remove-outliers/>)

Data Normalization

Using The min-max feature scaling

```
In [34]: df_original=df.copy()
```



```
In [35]: # apply normalization techniques
for column in df.columns:
    if column != 'Hour':
        df[column] = (df[column] - df[column].min()) / (df[column].max() - df[column].min())
    #view normalized data
df.head()
```

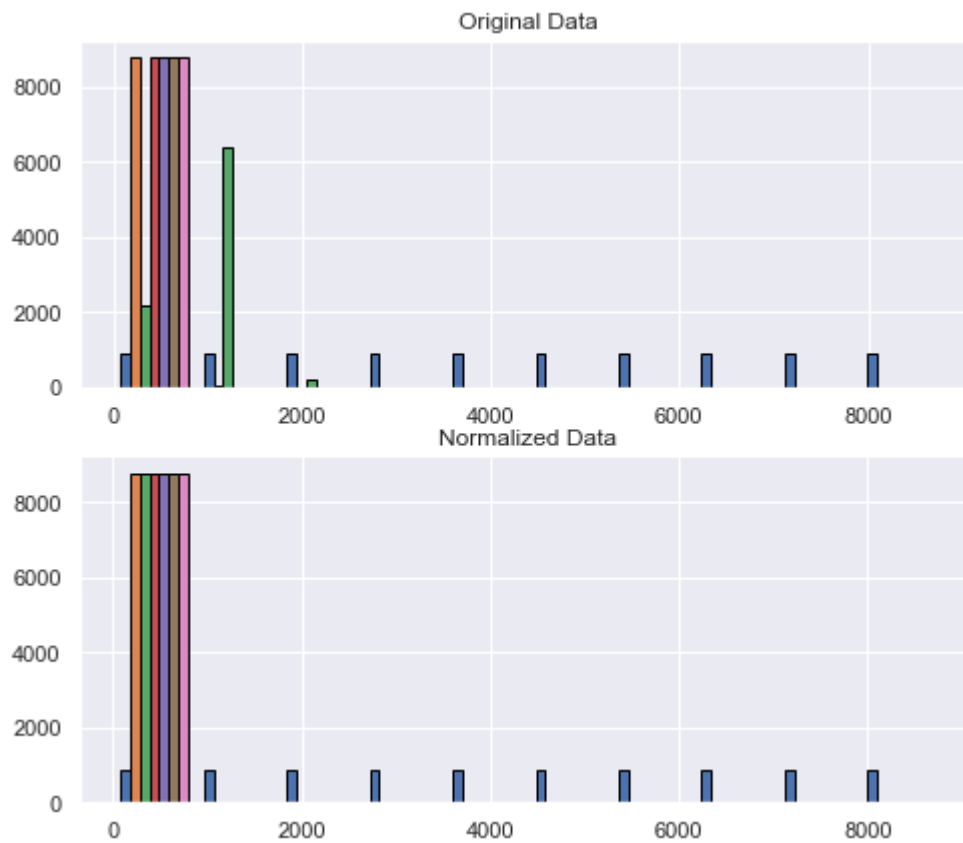
Out[35]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
0	1	0.258616	0.441027	0.063348	0.628866	0.208791	0.812500
1	2	0.067566	0.320328	0.055807	0.642612	0.212454	0.777778
2	3	0.067566	0.203495	0.048265	0.656357	0.216117	0.743056
3	4	0.067566	0.139124	0.040724	0.670103	0.219780	0.708333
4	5	0.067566	0.110299	0.034691	0.591065	0.228938	0.684028

```
In [36]: #define grid of plots
fig, axs = plt.subplots(nrows=2, ncols=1, figsize = (8,7))

#create histograms
axs[0].hist(df_original, edgecolor='black')
axs[1].hist(df, edgecolor='black')

#add title to each histogram
axs[0].set_title('Original Data')
axs[1].set_title('Normalized Data')
plt.show()
```



Dates

```
In [37]: date_dataframe = pd.DataFrame()
date_dataframe['Hour'] = df['Hour']

dias_por_mes = [31,29,31,30,31,31,30,31,30,31,30,31]

dias = []
mes = []
horas_dia = []
semana_dia = []

horas = 1
dia = 1
meses_completos = 1
semana = 0

for i in range(len(date_dataframe['Hour'])):

    if ((i) % 168 == 0):
        semana += 1

    else:
        semana = semana

    if (horas == 24):
        horas_dia.append(horas)
        horas = 1

        if (dia >= dias_por_mes[meses_completos-1]):
            dias.append(dia)
            semana_dia.append(semana)
            mes.append(meses_completos)
            meses_completos += 1
            dia = 1

        else:
            dias.append(dia)
            semana_dia.append(semana)
            mes.append(meses_completos)
            dia += 1

    else:
        horas_dia.append(horas)
```


```
dias.append(dia)
semana_dia.append(semama)
mes.append(meses_completos)
horas += 1
```

```
In [38]: date_dataframe['Hora'] = pd.DataFrame(horas_dia)
date_dataframe['Dia'] = pd.DataFrame(dias)
date_dataframe['Semana'] = pd.DataFrame(semama_dia)
date_dataframe['Mes'] = pd.DataFrame(mes)
```

```
In [39]: fechas_completas_2005 = []
fechas_completas_2006 = []

for i in range(len(date_dataframe['Hour'])):
    fecha = str(date_dataframe.Dia[i]) + '/' + str(date_dataframe.Mes[i]) + '/' + '2005' + ' ' + str(date_dataframe.Hora[i])
    fechas_completas_2005.append(str(fecha))

    fecha = str(date_dataframe.Dia[i]) + '/' + str(date_dataframe.Mes[i]) + '/' + '2006' + ' ' + str(date_dataframe.Hora[i])
    fechas_completas_2006.append(str(fecha))
```



```
In [40]: date_dataframe['Fecha_2005'] = pd.DataFrame(fechas_completas_2005)
date_dataframe['Fecha_2006'] = pd.DataFrame(fechas_completas_2006)
```

```
In [41]: date_dataframe.tail(10)
```

Out[41]:

	Hour	Hora	Dia	Semana	Mes	Fecha_2005	Fecha_2006
8774	8775	15	31	53	12	31/12/2005 15:00	31/12/2006 15:00
8775	8776	16	31	53	12	31/12/2005 16:00	31/12/2006 16:00
8776	8777	17	31	53	12	31/12/2005 17:00	31/12/2006 17:00
8777	8778	18	31	53	12	31/12/2005 18:00	31/12/2006 18:00
8778	8779	19	31	53	12	31/12/2005 19:00	31/12/2006 19:00
8779	8780	20	31	53	12	31/12/2005 20:00	31/12/2006 20:00
8780	8781	21	31	53	12	31/12/2005 21:00	31/12/2006 21:00
8781	8782	22	31	53	12	31/12/2005 22:00	31/12/2006 22:00
8782	8783	23	31	53	12	31/12/2005 23:00	31/12/2006 23:00
8783	8784	24	31	53	12	31/12/2005 24:00	31/12/2006 24:00

```
In [42]: date_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Hour        8784 non-null   int64
1   Hora        8784 non-null   int64
2   Dia         8784 non-null   int64
3   Semana      8784 non-null   int64
4   Mes         8784 non-null   int64
5   Fecha_2005  8784 non-null   object
6   Fecha_2006  8784 non-null   object
dtypes: int64(5), object(2)
memory usage: 480.5+ KB
```

```
In [43]: merged_df = pd.merge(left = df, right = date_dataframe, how = 'inner', on = 'Hour')
merged_df
```

Out[43]:

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006	Hora	Dia
0	1	0.258616	0.441027	0.063348	0.628866	0.208791	0.812500	1	1
1	2	0.067566	0.320328	0.055807	0.642612	0.212454	0.777778	2	1
2	3	0.067566	0.203495	0.048265	0.656357	0.216117	0.743056	3	1
3	4	0.067566	0.139124	0.040724	0.670103	0.219780	0.708333	4	1
4	5	0.067566	0.110299	0.034691	0.591065	0.228938	0.684028	5	1
...
8779	8780	0.884956	0.531863	0.147813	0.628866	0.067766	0.812500	20	31
8780	8781	0.746952	0.531863	0.132730	0.670103	0.067766	0.812500	21	31
8781	8782	0.575240	0.531863	0.117647	0.711340	0.067766	0.812500	22	31
8782	8783	0.472456	0.531863	0.102564	0.752577	0.067766	0.812500	23	31
8783	8784	0.405115	0.531863	0.087481	0.793814	0.067766	0.812500	24	31

8784 rows × 13 columns



Correlation between variables

At this point, data has passed through a cleaning process and as a result, we got the data frame call `data_log` .

```
In [44]: merged_df.head()
```

```
Out[44]:
```

	Hour	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006	Hora	Dia
0	1	0.258616	0.441027	0.063348	0.628866	0.208791	0.812500	1	1
1	2	0.067566	0.320328	0.055807	0.642612	0.212454	0.777778	2	1
2	3	0.067566	0.203495	0.048265	0.656357	0.216117	0.743056	3	1
3	4	0.067566	0.139124	0.040724	0.670103	0.219780	0.708333	4	1
4	5	0.067566	0.110299	0.034691	0.591065	0.228938	0.684028	5	1

```
In [45]: merged_df.shape
```

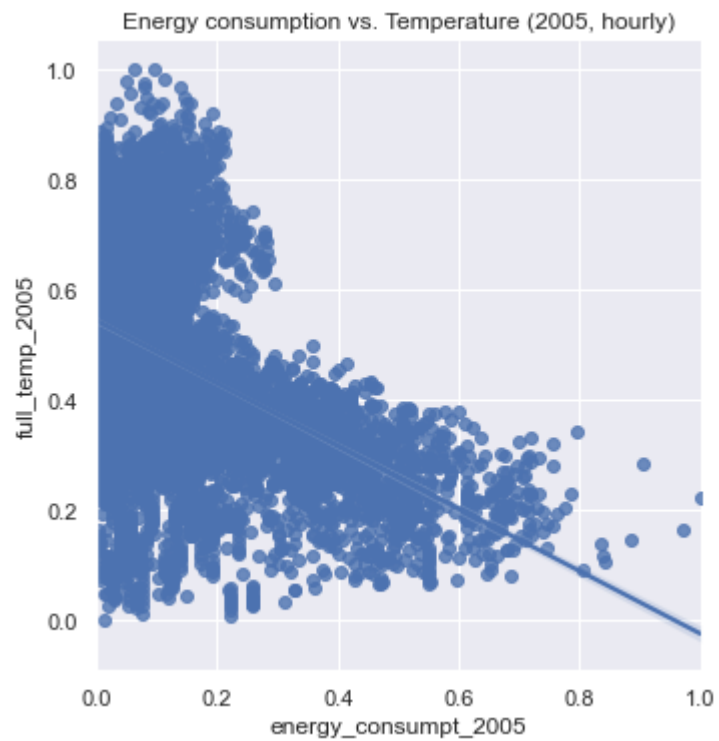
```
Out[45]: (8784, 13)
```

Correlation Analysis

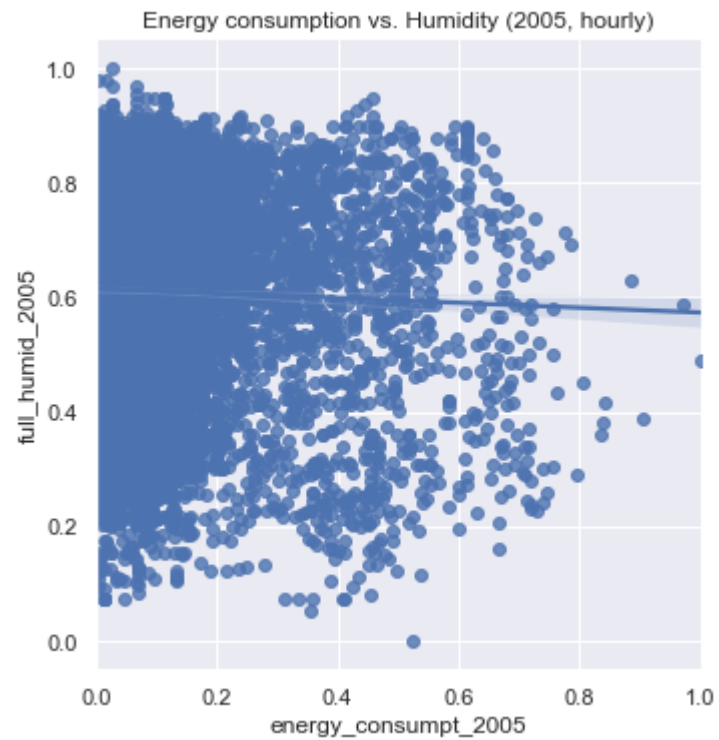
Now, a correlation analysis between the variables of the data frame will be done using three different frequency of data (hourly, daily and weekly).

Hourly Analysis

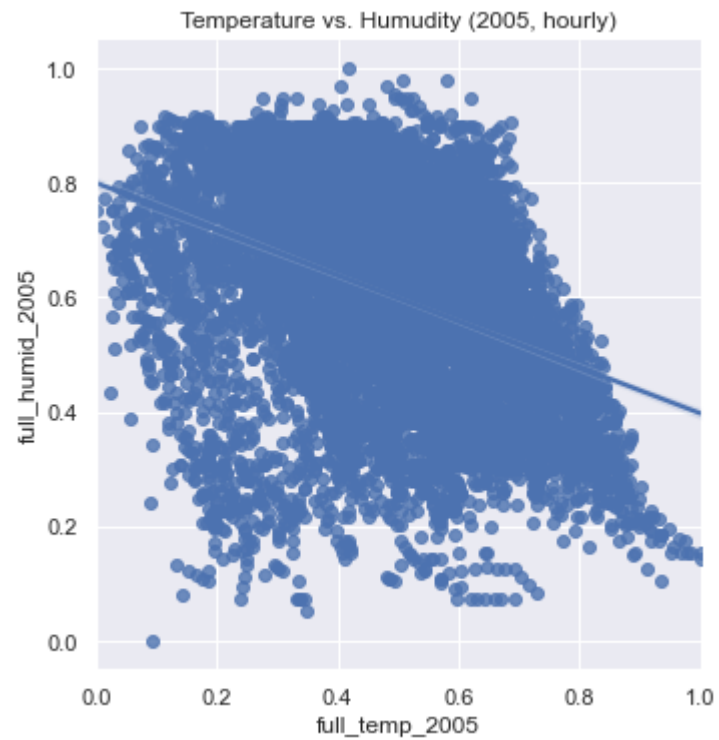

```
In [46]: sns.lmplot(x='energy_consumpt_2005', y='full_temp_2005', data=merged_df)
plt.title('Energy consumption vs. Temperature (2005, hourly)')
plt.show()
```



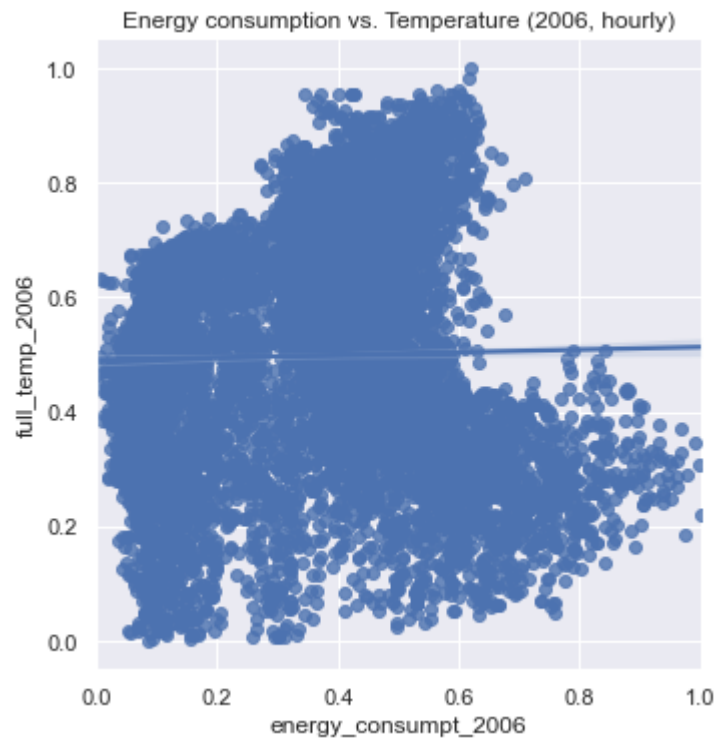
```
In [47]: sns.lmplot(x='energy_consumpt_2005', y='full_humid_2005', data=merged_df)
plt.title('Energy consumption vs. Humidity (2005, hourly)')
plt.show()
```



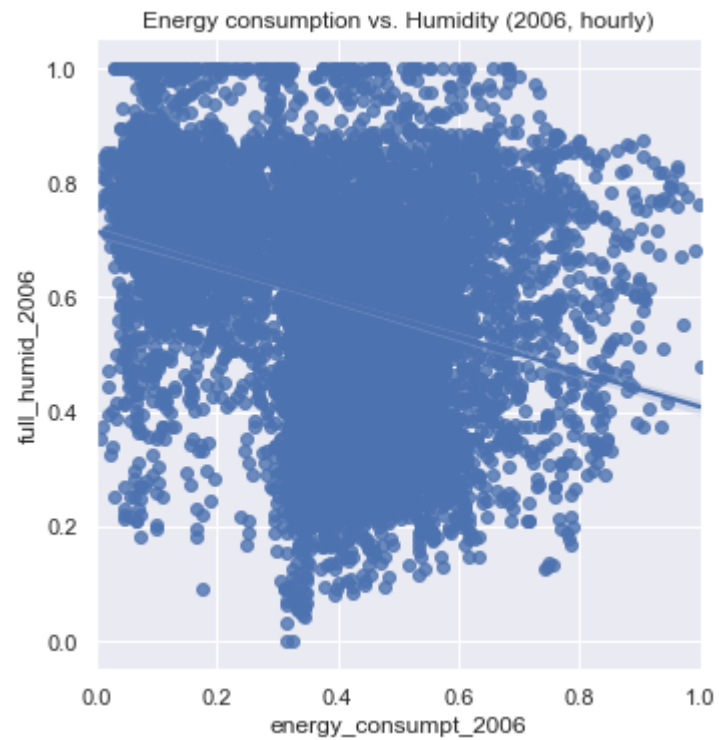
```
In [48]: sns.lmplot(x='full_temp_2005', y='full_humid_2005', data=merged_df)
plt.title('Temperature vs. Humidity (2005, hourly)')
plt.show()
```



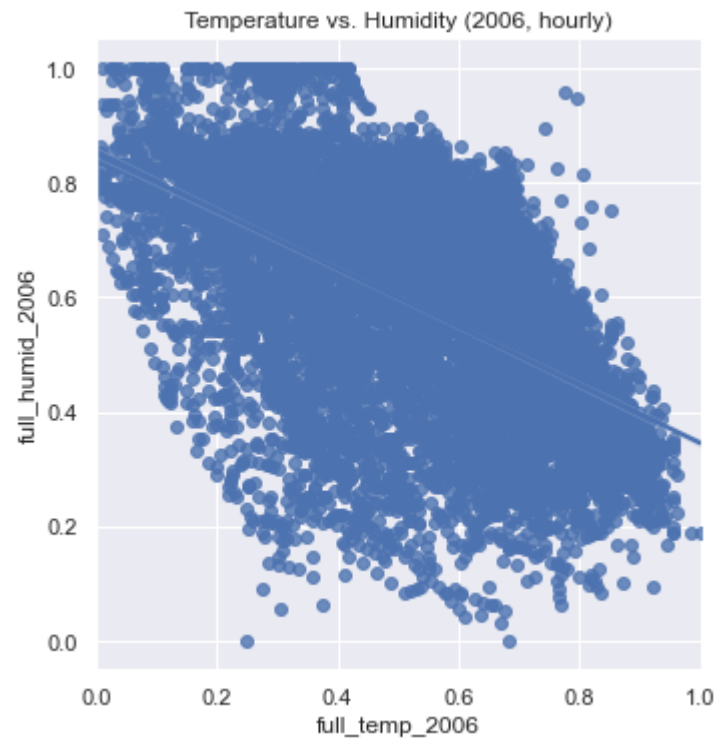
```
In [49]: sns.lmplot(x='energy_consumpt_2006', y='full_temp_2006', data=merged_df)
plt.title('Energy consumption vs. Temperature (2006, hourly)')
plt.show()
```



```
In [50]: sns.lmplot(x='energy_consumpt_2006', y='full_humid_2006', data=merged_df)
plt.title('Energy consumption vs. Humidity (2006, hourly)')
plt.show()
```



```
In [51]: sns.lmplot(x='full_temp_2006', y='full_humid_2006', data=merged_df)
plt.title('Temperature vs. Humidity (2006, hourly)')
plt.show()
```

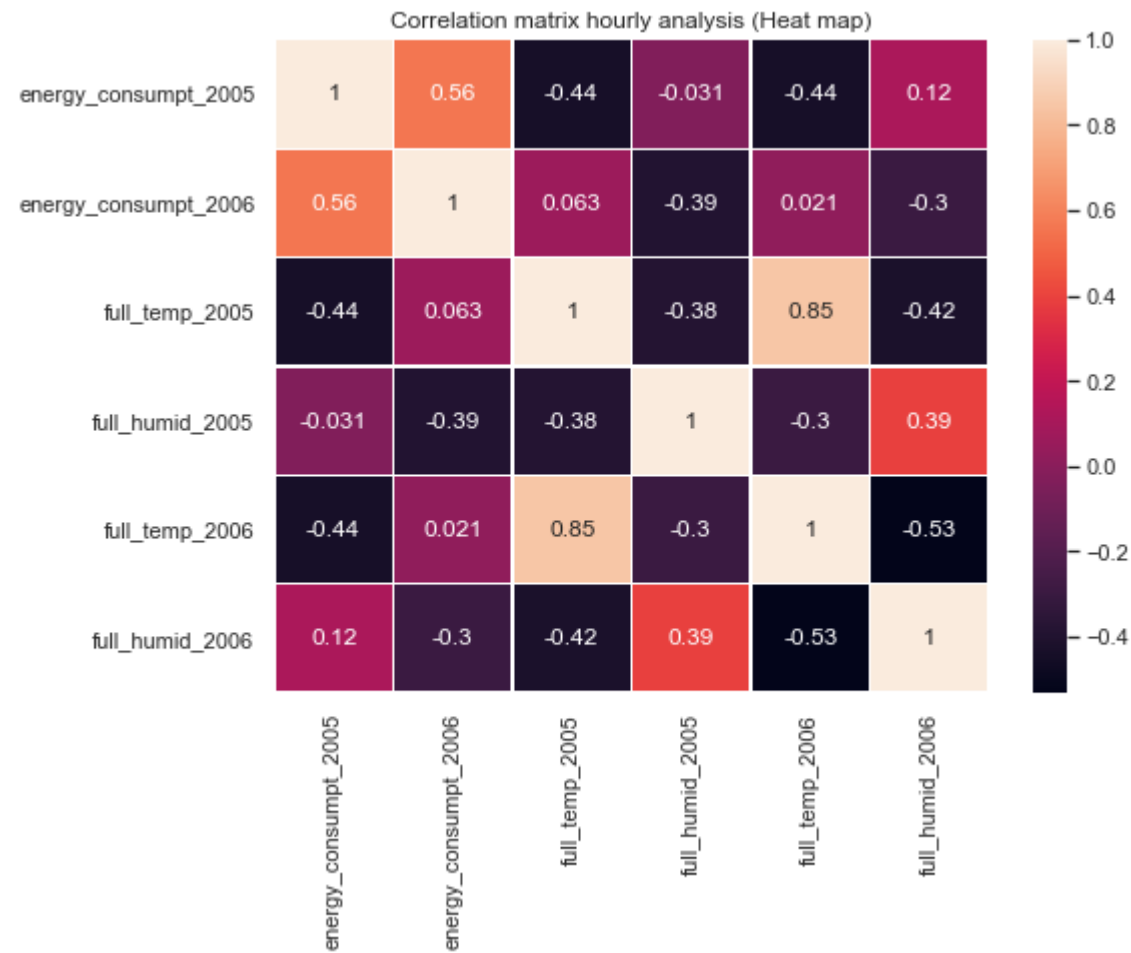


```
In [52]: correlation_matrix = merged_df[['energy_consumpt_2005', 'energy_consumpt_2006', 'full_temp_2005',  
                                         'full_humid_2005', 'full_temp_2006', 'full_humid_2006']].corr()  
round(correlation_matrix,2)
```

Out[52]:

	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
energy_consumpt_2005	1.00	0.56	-0.44	-0.03	-0.44	0.12
energy_consumpt_2006	0.56	1.00	0.06	-0.39	0.02	-0.30
full_temp_2005	-0.44	0.06	1.00	-0.38	0.85	-0.42
full_humid_2005	-0.03	-0.39	-0.38	1.00	-0.30	0.39
full_temp_2006	-0.44	0.02	0.85	-0.30	1.00	-0.53
full_humid_2006	0.12	-0.30	-0.42	0.39	-0.53	1.00

```
In [53]: plt.figure(figsize = (8,6))
sns.heatmap(correlation_matrix, annot = True, linewidths = 0.2);
plt.title('Correlation matrix hourly analysis (Heat map)')
plt.show()
```



Daily Analysis

```
In [54]: daily_df = pd.DataFrame(columns = ['Day', 'energy_consumpt_2005', 'energy_consumpt_2006',  
                                             'full_temp_2005', 'full_humid_2005', 'full_temp_2006', 'full_humid_2006'])  
daily_df
```

Out[54]:

Day	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
-----	----------------------	----------------------	----------------	-----------------	----------------	-----------------

```

In [55]: inferior = 1
         superior = 25

         for day in range(1, 367):

             sub_df = merged_df[(merged_df['Hour'] >= inferior) & (merged_df['Hour'] < superior)]

             a = sub_df['energy_consumpt_2005'].sum()
             b = sub_df['energy_consumpt_2006'].sum()
             c = sub_df['full_temp_2005'].mean()
             d = sub_df['full_humid_2005'].mean()
             e = sub_df['full_temp_2006'].mean()
             f = sub_df['full_humid_2006'].mean()

             new_row = {'Day': day, 'energy_consumpt_2005':a, 'energy_consumpt_2006':b,
                        'full_temp_2005':c, 'full_humid_2005':d, 'full_temp_2006':e, 'full_humid_2006':f}

             daily_df = daily_df.append(new_row, ignore_index=True)

             inferior += 24
             superior += 24

daily_df

```

Out[55]:

	Day	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
0	1.0	6.465117	10.354534	0.143288	0.408935	0.280907	0.671875
1	2.0	6.250899	10.904980	0.239913	0.606100	0.259158	0.802083
2	3.0	6.692253	13.078194	0.287142	0.846649	0.234203	0.599392
3	4.0	8.700107	14.602220	0.289781	0.834192	0.223443	0.634115
4	5.0	7.806567	14.510661	0.337010	0.815722	0.278617	0.524740
...
361	362.0	8.809112	12.982791	0.226810	0.712629	0.162088	0.722222
362	363.0	8.871168	13.223763	0.199472	0.400773	0.201694	0.783854
363	364.0	9.004471	13.651571	0.192873	0.270189	0.150641	0.737413

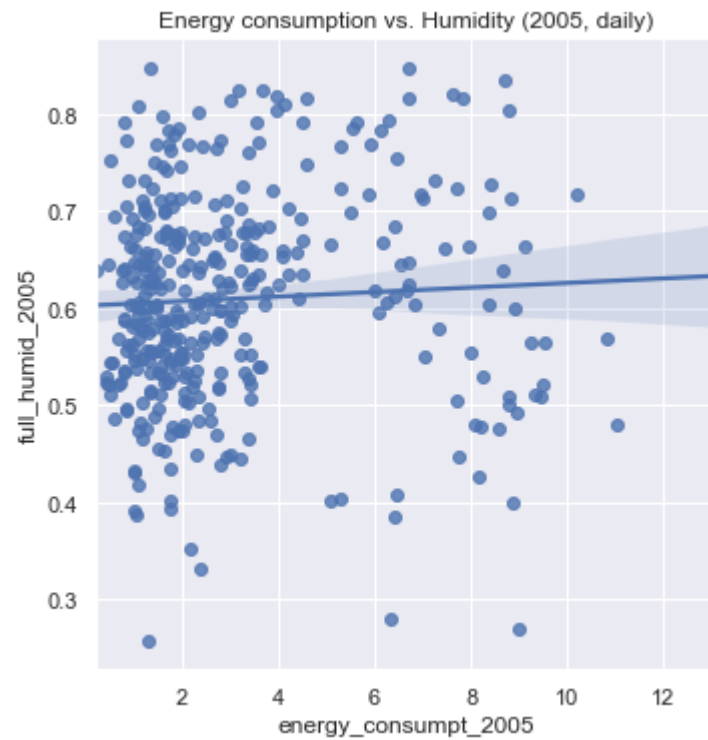
	Day	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
364	365.0	11.030877	14.421458	0.119910	0.479811	0.131181	0.719184
365	366.0	13.080026	12.764708	0.135558	0.570017	0.067766	0.812500

366 rows × 7 columns

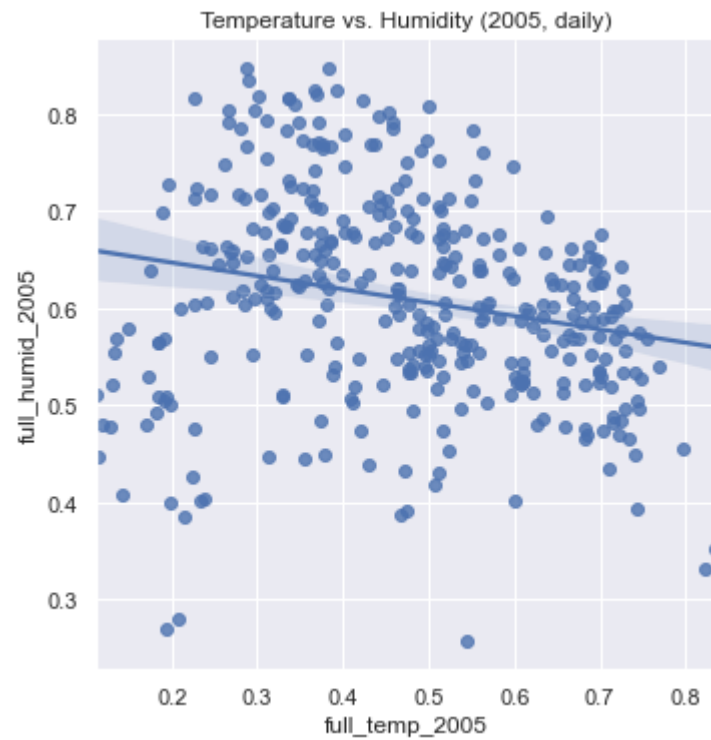
```
In [56]: sns.lmplot(x = 'energy_consumpt_2005', y = 'full_temp_2005', data=daily_df)
plt.title('Energy consumption vs. Temperature (2005, daily)')
plt.show()
```



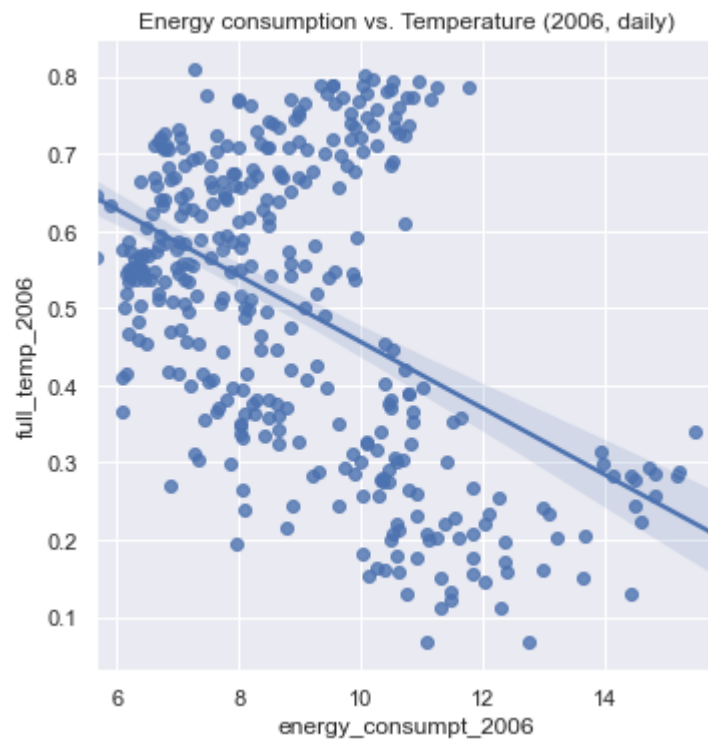
```
In [57]: sns.lmplot(x='energy_consumpt_2005', y='full_humid_2005', data=daily_df)
plt.title('Energy consumption vs. Humidity (2005, daily)')
plt.show()
```



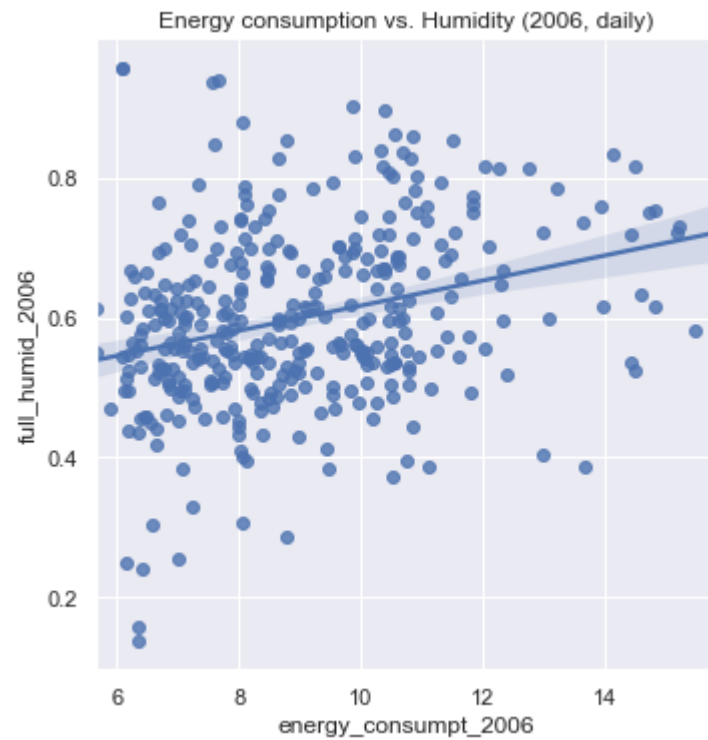
```
In [58]: sns.lmplot(x='full_temp_2005', y='full_humid_2005', data=daily_df)
plt.title('Temperature vs. Humidity (2005, daily)')
plt.show()
```



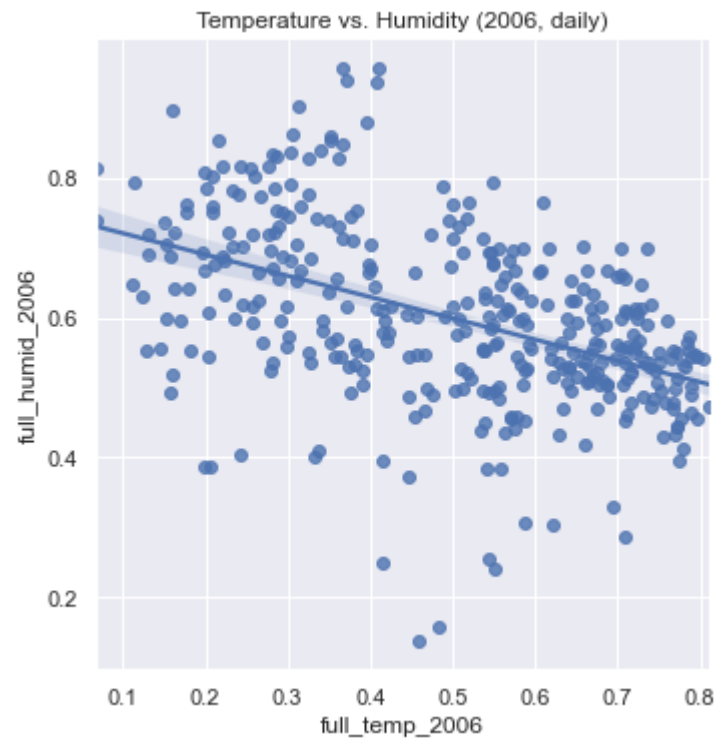
```
In [59]: sns.lmplot(x='energy_consumpt_2006', y='full_temp_2006', data=daily_df)
plt.title('Energy consumption vs. Temperature (2006, daily)')
plt.show()
```



```
In [60]: sns.lmplot(x='energy_consumpt_2006', y='full_humid_2006', data=daily_df)
plt.title('Energy consumption vs. Humidity (2006, daily)')
plt.show()
```



```
In [61]: sns.lmplot(x='full_temp_2006', y='full_humid_2006', data=daily_df)
plt.title('Temperature vs. Humidity (2006, daily)')
plt.show()
```




```
In [62]: correlation_matrix = daily_df.corr()  
round(correlation_matrix,2)
```

Out[62]:

	Day	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid
Day	1.00	-0.00	-0.38	0.12	-0.06	0.18	
energy_consumpt_2005	-0.00	1.00	0.67	-0.72	0.05	-0.71	
energy_consumpt_2006	-0.38	0.67	1.00	-0.34	-0.01	-0.46	
full_temp_2005	0.12	-0.72	-0.34	1.00	-0.21	0.87	
full_humid_2005	-0.06	0.05	-0.01	-0.21	1.00	-0.15	
full_temp_2006	0.18	-0.71	-0.46	0.87	-0.15	1.00	
full_humid_2006	0.03	0.40	0.30	-0.38	0.00	-0.46	



```
In [63]: plt.figure(figsize = (8,6))
sns.heatmap(correlation_matrix, annot = True, linewidths = 0.2);
plt.title('Correlation matrix daily analysis (Heat map)')
plt.show()
```



Weekly

```
In [64]: weekly_df = pd.DataFrame(columns = ['Week', 'energy_consumpt_2005', 'energy_consumpt_2006',  
                                             'full_temp_2005', 'full_humid_2005', 'full_temp_2006', 'full_humid_2006'])  
weekly_df
```

```
Out[64]:
```

Week	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
------	----------------------	----------------------	----------------	-----------------	----------------	-----------------

In [65]: **for** week **in** range(1, 54):

```
    sub_df = merged_df[merged_df['Semana'] == week]
```

```
    a = sub_df['energy_consumpt_2005'].sum()
```

```
    b = sub_df['energy_consumpt_2006'].sum()
```

```
    c = sub_df['full_temp_2005'].mean()
```

```
    d = sub_df['full_humid_2005'].mean()
```

```
    e = sub_df['full_temp_2006'].mean()
```

```
    f = sub_df['full_humid_2006'].mean()
```

```
    new_row = {'Week': week, 'energy_consumpt_2005':a, 'energy_consumpt_2006':b,  
              'full_temp_2005':c, 'full_humid_2005':d, 'full_temp_2006':e, 'full_humid_2006':f}
```

```
    weekly_df = weekly_df.append(new_row, ignore_index=True)
```

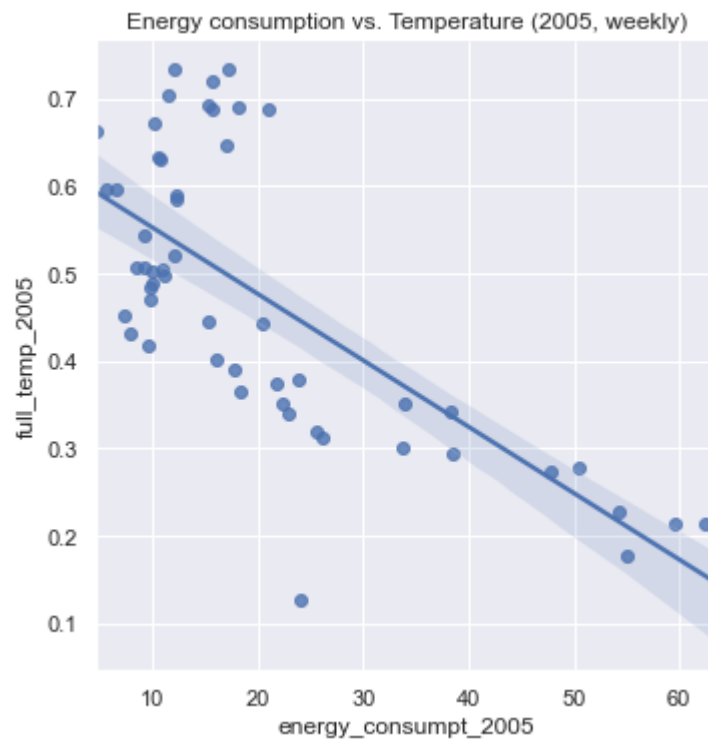
weekly_df

Out[65]:

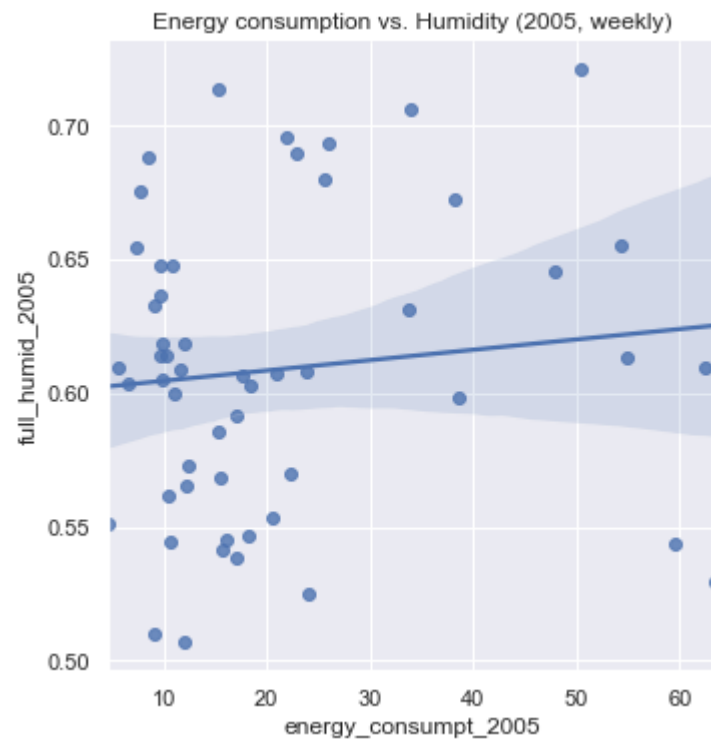
	Week	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
0	1.0	50.503140	93.467373	0.278765	0.720913	0.263507	0.672433
1	2.0	38.271922	103.868542	0.342626	0.672251	0.294414	0.659722
2	3.0	54.951655	93.704788	0.177763	0.613095	0.255200	0.672619
3	4.0	62.367058	83.856671	0.214393	0.609168	0.164737	0.593130
4	5.0	33.960903	74.504284	0.350463	0.705817	0.282624	0.675905
...
48	49.0	47.862028	69.790682	0.272975	0.645250	0.261120	0.698661
49	50.0	54.303694	73.925921	0.227241	0.655437	0.226648	0.714224
50	51.0	63.453021	74.085259	0.195809	0.529823	0.251603	0.820375
51	52.0	59.588217	85.403866	0.214663	0.543692	0.188628	0.686756
52	53.0	24.110903	27.186166	0.127734	0.524914	0.099473	0.765842

53 rows × 7 columns

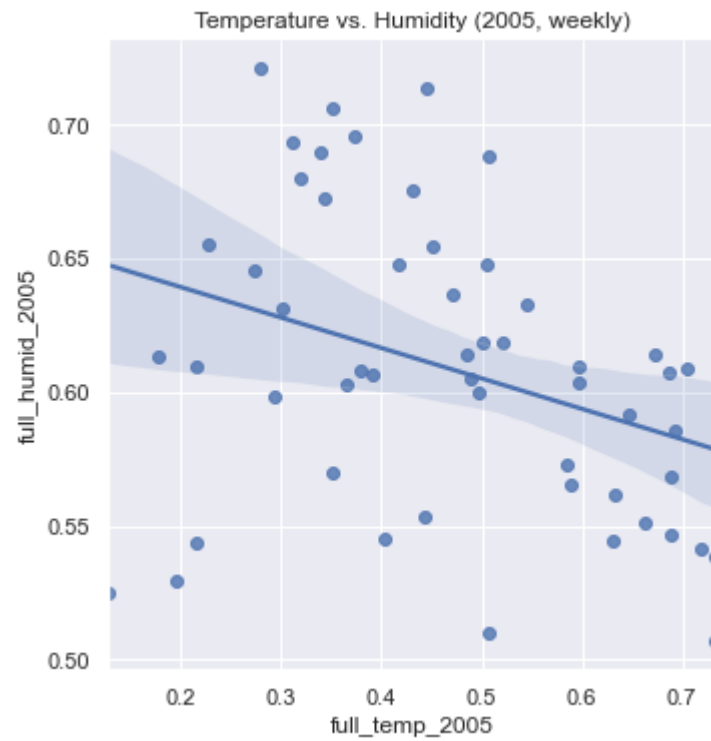
```
In [66]: sns.lmplot(x='energy_consumpt_2005', y='full_temp_2005', data=weekly_df)
plt.title('Energy consumption vs. Temperature (2005, weekly)')
plt.show()
```



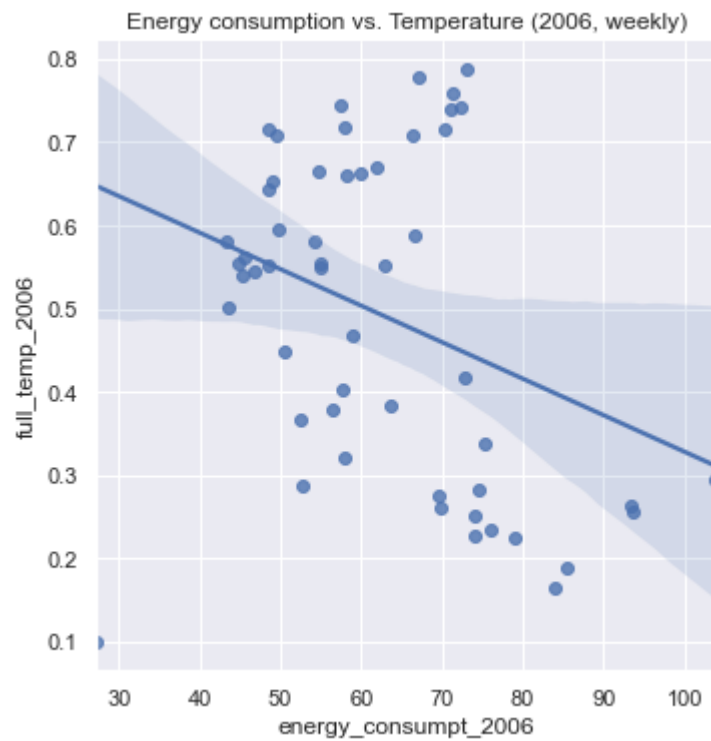
```
In [67]: sns.lmplot(x='energy_consumpt_2005', y='full_humid_2005', data=weekly_df)
plt.title('Energy consumption vs. Humidity (2005, weekly)')
plt.show()
```



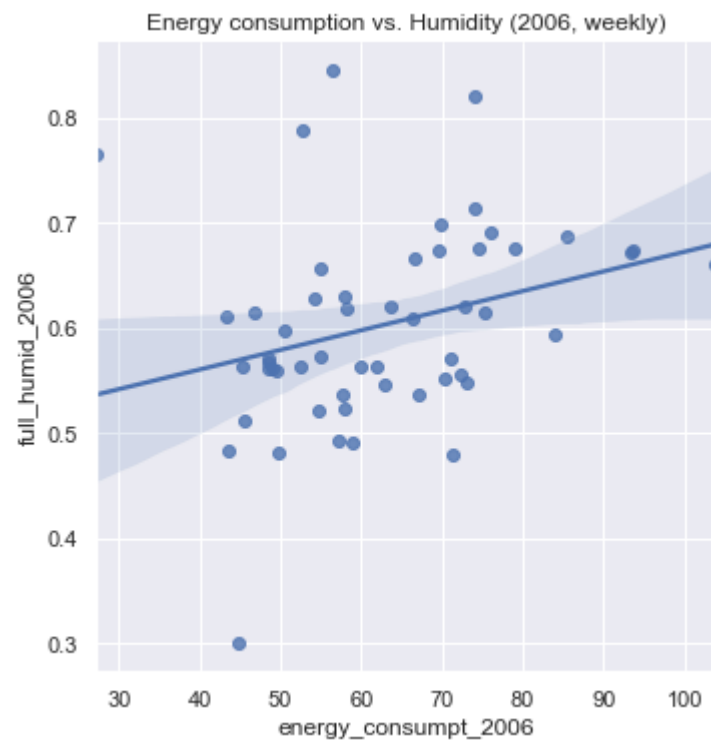
```
In [68]: sns.lmplot(x='full_temp_2005', y='full_humid_2005', data=weekly_df)
plt.title('Temperature vs. Humidity (2005, weekly)')
plt.show()
```



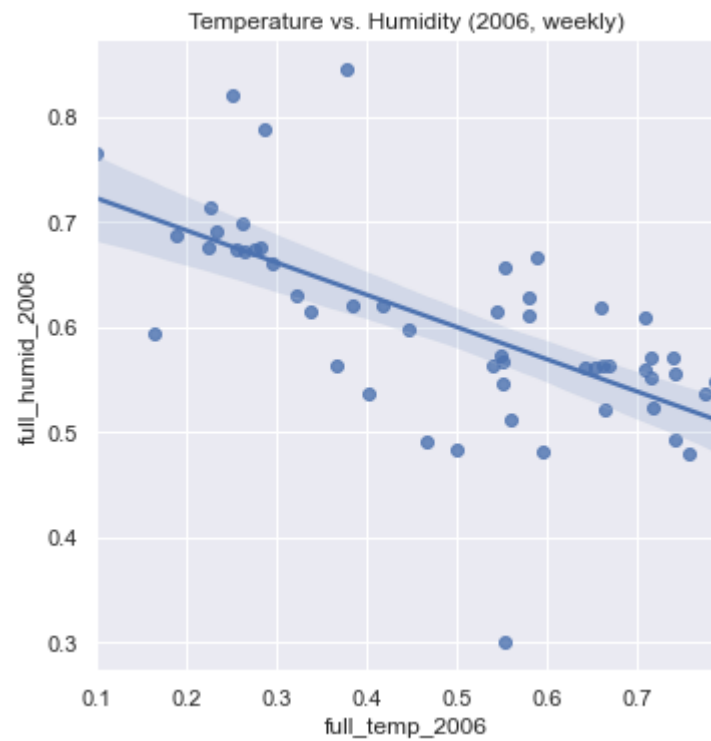
```
In [69]: sns.lmplot(x='energy_consumpt_2006', y='full_temp_2006', data=weekly_df)
plt.title('Energy consumption vs. Temperature (2006, weekly)')
plt.show()
```




```
In [70]: sns.lmplot(x='energy_consumpt_2006', y='full_humid_2006', data=weekly_df)
plt.title('Energy consumption vs. Humidity (2006, weekly)')
plt.show()
```



```
In [71]: sns.lmplot(x='full_temp_2006', y='full_humid_2006', data=weekly_df)
plt.title('Temperature vs. Humidity (2006, weekly)')
plt.show()
```

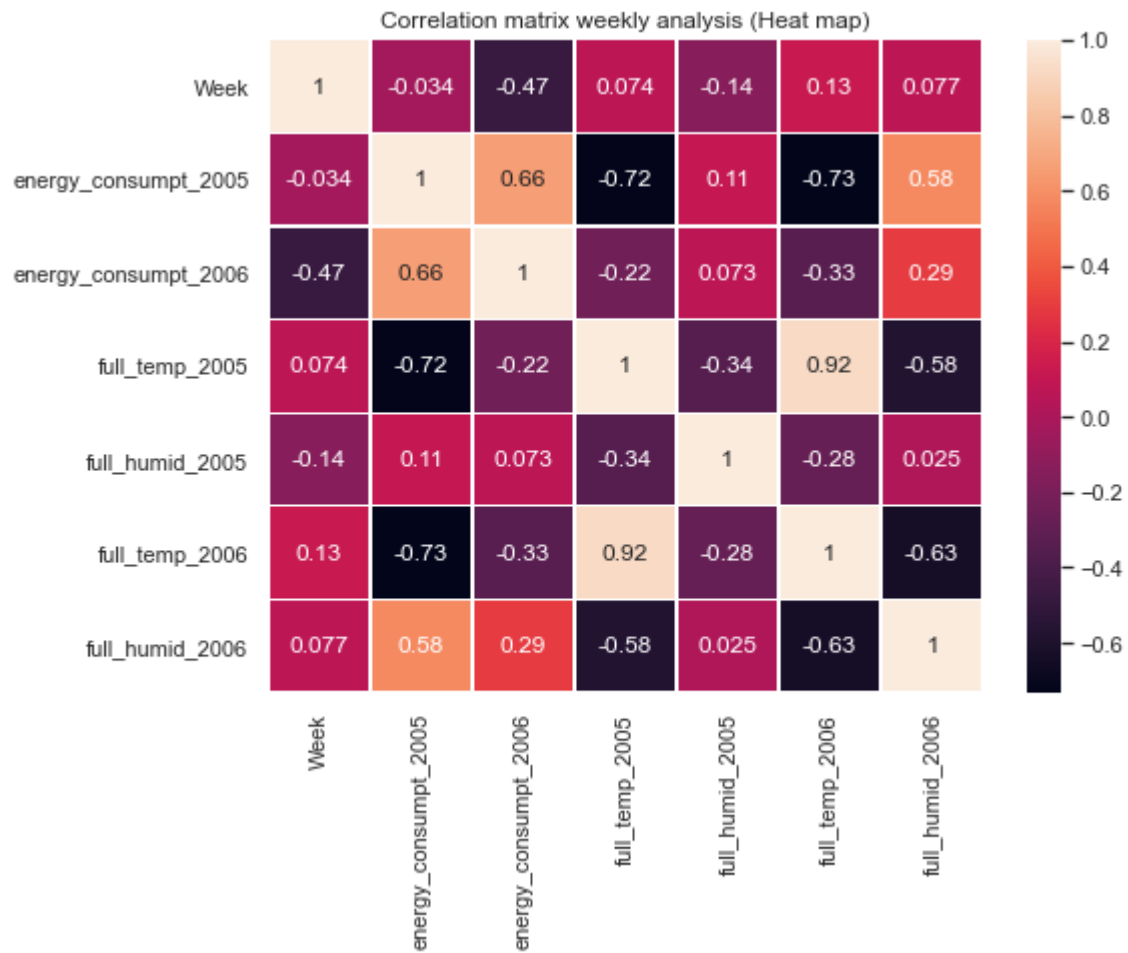


```
In [72]: correlation_matrix = weekly_df.corr()  
round(correlation_matrix,2)
```

Out[72]:

	Week	energy_consumpt_2005	energy_consumpt_2006	full_temp_2005	full_humid_2005	full_temp_2006	full_humid_2006
Week	1.00	-0.03	-0.47	0.07	-0.14	0.13	0.08
energy_consumpt_2005	-0.03	1.00	0.66	-0.72	0.11	-0.73	0.58
energy_consumpt_2006	-0.47	0.66	1.00	-0.22	0.07	-0.33	0.29
full_temp_2005	0.07	-0.72	-0.22	1.00	-0.34	0.92	-0.58
full_humid_2005	-0.14	0.11	0.07	-0.34	1.00	-0.28	0.03
full_temp_2006	0.13	-0.73	-0.33	0.92	-0.28	1.00	-0.63
full_humid_2006	0.08	0.58	0.29	-0.58	0.03	-0.63	1.00

```
In [73]: plt.figure(figsize = (8,6))
sns.heatmap(correlation_matrix, annot = True, linewidths = 0.2);
plt.title('Correlation matrix weekly analysis (Heat map)')
plt.show()
```



Data Concatenation

Weekly

```
In [74]: columnas2005=['Week', 'energy_consumpt_2005', 'full_temp_2005', 'full_humid_2005']
columnas2006=['Week', 'energy_consumpt_2006', 'full_temp_2006', 'full_humid_2006']

weekly_df2005=weekly_df[columnas2005]
weekly_df2006=weekly_df[columnas2006]

weekly_df2005=weekly_df2005.rename(columns={'energy_consumpt_2005':'energy_consumpt',
                                             'full_temp_2005':'full_temp',
                                             'full_humid_2005':'full_humid'})

weekly_df2006=weekly_df2006.rename(columns={'energy_consumpt_2006':'energy_consumpt',
                                             'full_temp_2006':'full_temp',
                                             'full_humid_2006':'full_humid'})

weekly_f = pd.concat([weekly_df2005, weekly_df2006])
weekly_f=weekly_f.reset_index(drop=True)
weekly_f=weekly_f.drop(['Week'], axis = 'columns')
```

Daily

```
In [75]: columnas2005=['Day', 'energy_consumpt_2005', 'full_temp_2005', 'full_humid_2005']
columnas2006=['Day', 'energy_consumpt_2006', 'full_temp_2006', 'full_humid_2006']

daily_df2005 = daily_df[columnas2005]
daily_df2006 = daily_df[columnas2006]

daily_df2005 = daily_df2005.rename(columns={'energy_consumpt_2005': 'energy_consumpt',
                                             'full_temp_2005': 'full_temp',
                                             'full_humid_2005': 'full_humid'})

daily_df2006 = daily_df2006.rename(columns={'energy_consumpt_2006': 'energy_consumpt',
                                             'full_temp_2006': 'full_temp',
                                             'full_humid_2006': 'full_humid'})

daily_f = pd.concat([daily_df2005, daily_df2006])
daily_f = daily_f.reset_index(drop=True)
daily_f = daily_f.drop(['Day'], axis = 'columns')
```

Hourly

```
In [76]: columnas2005=['Hora', 'energy_consumpt_2005', 'full_temp_2005', 'full_humid_2005']
columnas2006=['Hora', 'energy_consumpt_2006', 'full_temp_2006', 'full_humid_2006']

hourly_df2005 = merged_df[columnas2005]
hourly_df2006 = merged_df[columnas2006]

hourly_df2005 = hourly_df2005.rename(columns={'energy_consumpt_2005': 'energy_consumpt',
                                             'full_temp_2005': 'full_temp',
                                             'full_humid_2005': 'full_humid'})

hourly_df2006 = hourly_df2006.rename(columns={'energy_consumpt_2006': 'energy_consumpt',
                                             'full_temp_2006': 'full_temp',
                                             'full_humid_2006': 'full_humid'})

hourly_f = pd.concat([hourly_df2005, hourly_df2006])
hourly_f = hourly_f.reset_index(drop=True)
hourly_f = hourly_f.drop(['Hora'], axis = 'columns')
```

Data Visualization

Weekly Graphs

```

In [77]: columnas2005=['Week', 'energy_consumpt_2005','full_temp_2005','full_humid_2005']
columnas2006=['Week', 'energy_consumpt_2006','full_temp_2006','full_humid_2006']

weekly_df2005_graficas=weekly_df[columnas2005]
weekly_df2006_graficas=weekly_df[columnas2006]

weekly_df2005_graficas=weekly_df2005_graficas.rename(columns={'energy_consumpt_2005':'energy_consumpt',
                                                             'full_temp_2005':'full_temp',
                                                             'full_humid_2005':'full_humid'})

weekly_df2006_graficas=weekly_df2006_graficas.rename(columns={'energy_consumpt_2006':'energy_consumpt',
                                                             'full_temp_2006':'full_temp',
                                                             'full_humid_2006':'full_humid'})

weekly_f_graficas = pd.concat([weekly_df2005_graficas, weekly_df2006_graficas])
weekly_f_graficas=weekly_f.reset_index(drop=True)
weekly_f_graficas['Week'] = 0

for i in range(len(weekly_f_graficas['Week'])):
    weekly_f_graficas.Week[i]=i+1

weekly_f_graficas

```

Out[77]:

	energy_consumpt	full_temp	full_humid	Week
0	50.503140	0.278765	0.720913	1
1	38.271922	0.342626	0.672251	2
2	54.951655	0.177763	0.613095	3
3	62.367058	0.214393	0.609168	4
4	33.960903	0.350463	0.705817	5
...
101	69.790682	0.261120	0.698661	102
102	73.925921	0.226648	0.714224	103
103	74.085259	0.251603	0.820375	104
104	85.403866	0.188628	0.686756	105
105	27.186166	0.099473	0.765842	106

106 rows × 4 columns

```

In [78]: import plotly.graph_objects as go

energy_consumpt_fig = weekly_f_graficas['energy_consumpt']
full_temp_fig = weekly_f_graficas['full_temp']
full_humid_fig = weekly_f_graficas['full_humid']

x = weekly_f_graficas['Week']

fig = go.Figure()

fig.add_trace(go.Scatter(x=x, y=energy_consumpt_fig,
                        mode='lines',
                        name='energy_consumpt', marker=dict(color="DarkOrange"))),
fig.add_trace(go.Scatter(x=x, y=full_temp_fig,
                        mode='lines',
                        name='full_temp', marker=dict(color="Crimson"))),
fig.add_trace(go.Scatter(x=x, y=full_humid_fig,
                        mode='lines',
                        name='full_humid', marker=dict(color="RebeccaPurple")))

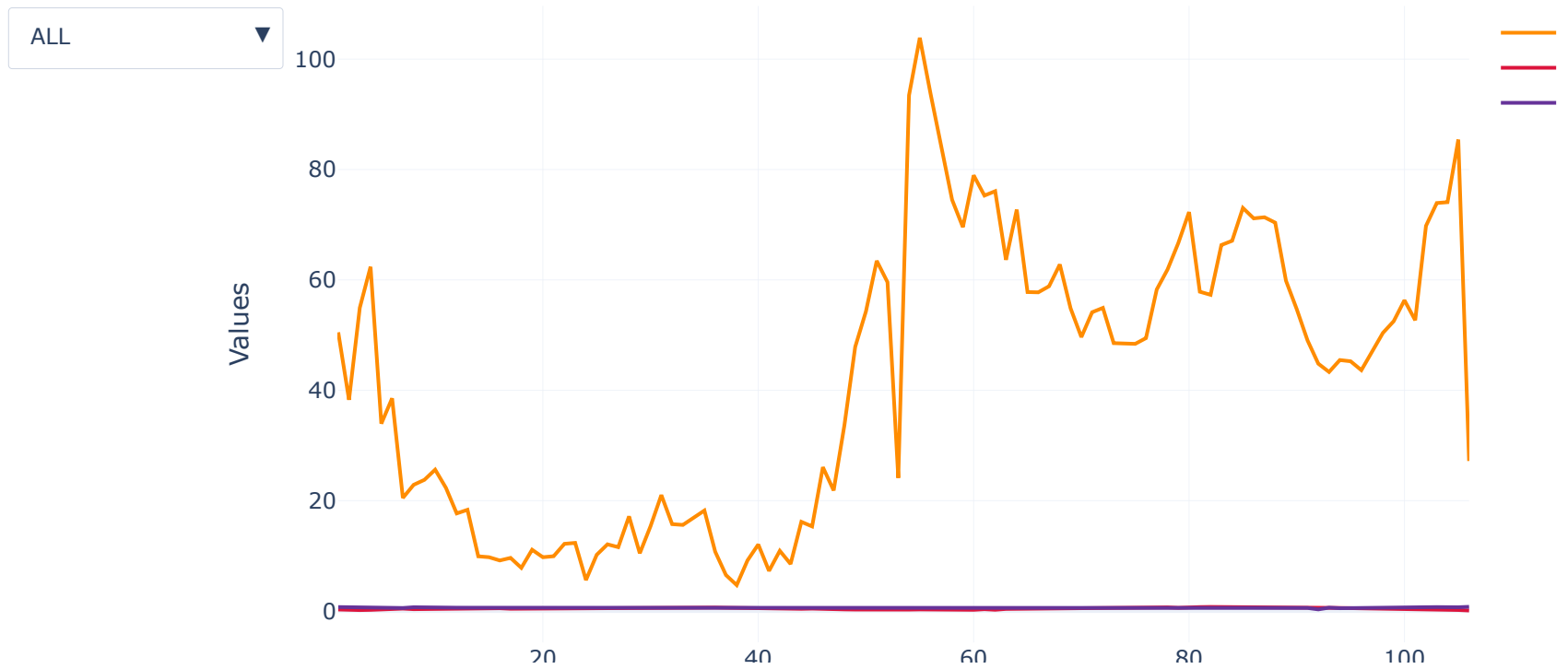
fig.update_layout(title_text='Consumo por semana',
                  xaxis_title='Semana', yaxis_title='Values')

fig.update_layout(template = 'plotly_white',
                  updatemenus=[
                      dict(active=0,
                          buttons=list([
                              dict(label="ALL",
                                  method="update",
                                  args=[{"visible": [True, True, True]},
                                      {"title": "All"}])),
                              dict(label="energy_consumpt",
                                  method="update",
                                  args=[{"visible": [True, False, False]},
                                      {"title": "energy_consumpt"}])),
                              dict(label="full_temp",
                                  method="update",
                                  args=[{"visible": [False, True, False]},
                                      {"title": "full_temp"}])),
                              dict(label="full_humid",
                                  method="update",
                                  args=[{"visible": [False, False, True]}],

```

```
    [{"title": "full_humid"}])  
    ],  
    )  
)  
fig.show()
```

Consumo por semana



Daily Graphs


```

In [79]: columnas2005=['Day', 'energy_consumpt_2005','full_temp_2005','full_humid_2005']
columnas2006=['Day', 'energy_consumpt_2006','full_temp_2006','full_humid_2006']

daily_df2005_graficas = daily_df[columnas2005]
daily_df2006_graficas = daily_df[columnas2006]

daily_df2005_graficas = daily_df2005_graficas.rename(columns={'energy_consumpt_2005':'energy_consumpt',
                                                             'full_temp_2005':'full_temp',
                                                             'full_humid_2005':'full_humid'})

daily_df2006_graficas = daily_df2006_graficas.rename(columns={'energy_consumpt_2006':'energy_consumpt',
                                                             'full_temp_2006':'full_temp',
                                                             'full_humid_2006':'full_humid'})

daily_f_graficas = pd.concat([daily_df2005_graficas, daily_df2006_graficas])
daily_f_graficas = daily_f_graficas.reset_index(drop=True)
daily_f_graficas['Day'] = 0

for i in range(len(daily_f_graficas['Day'])):
    daily_f_graficas.Day[i]=i+1

daily_f_graficas

```

Out[79]:

	Day	energy_consumpt	full_temp	full_humid
0	1	6.465117	0.143288	0.408935
1	2	6.250899	0.239913	0.606100
2	3	6.692253	0.287142	0.846649
3	4	8.700107	0.289781	0.834192
4	5	7.806567	0.337010	0.815722
...
727	728	12.982791	0.162088	0.722222
728	729	13.223763	0.201694	0.783854
729	730	13.651571	0.150641	0.737413
730	731	14.421458	0.131181	0.719184
731	732	12.764708	0.067766	0.812500

732 rows × 4 columns

```

In [80]: import plotly.graph_objects as go

energy_consumpt_fig = daily_f_graficas['energy_consumpt']
full_temp_fig = daily_f_graficas['full_temp']
full_humid_fig = daily_f_graficas['full_humid']

x = daily_f_graficas['Day']

fig = go.Figure()

fig.add_trace(go.Scatter(x=x, y=energy_consumpt_fig,
                        mode='lines',
                        name='energy_consumpt', marker=dict(color="DarkOrange"))),
fig.add_trace(go.Scatter(x=x, y=full_temp_fig,
                        mode='lines',
                        name='full_temp', marker=dict(color="Crimson"))),
fig.add_trace(go.Scatter(x=x, y=full_humid_fig,
                        mode='lines',
                        name='full_humid', marker=dict(color="RebeccaPurple")))

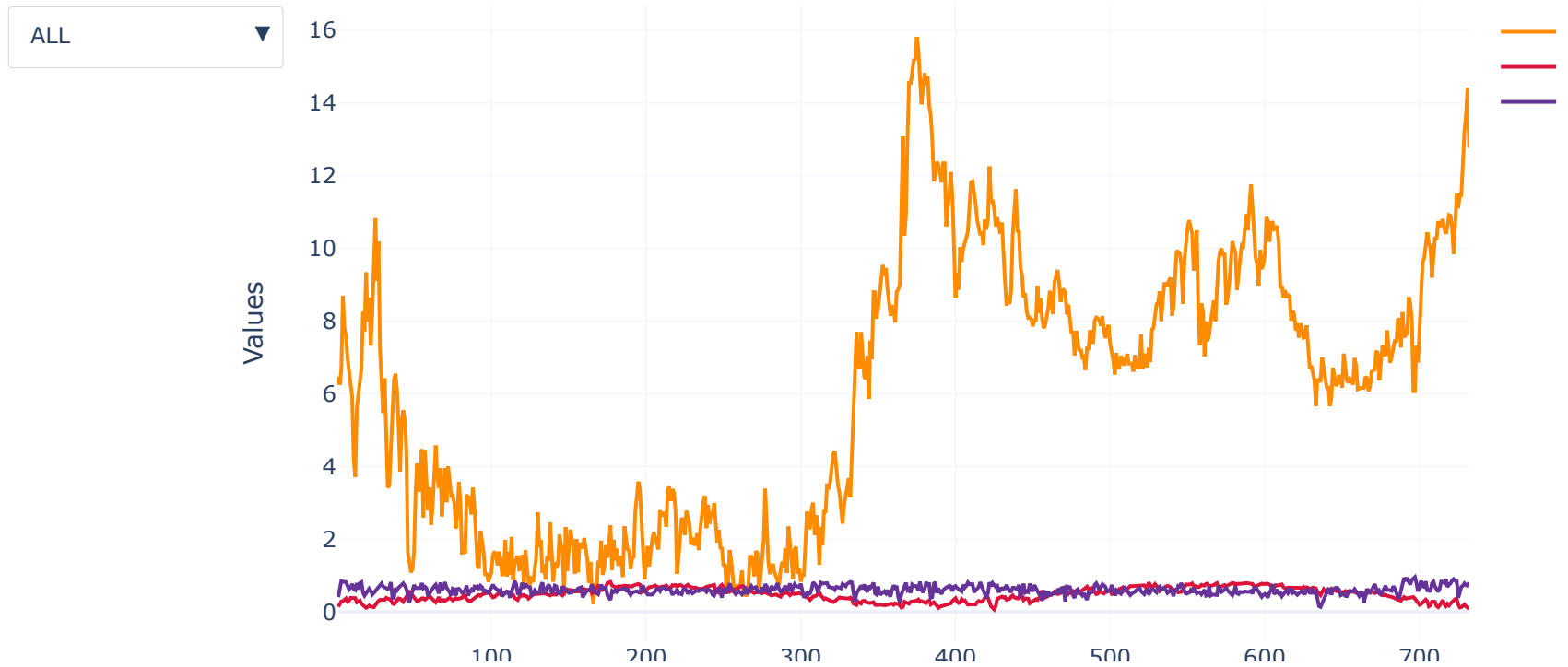
fig.update_layout(title_text='Consumo por día',
                  xaxis_title='Día', yaxis_title='Values')

fig.update_layout(template = 'plotly_white',
                  updatemenus=[
                      dict(active=0,
                          buttons=list([
                              dict(label="ALL",
                                  method="update",
                                  args=[{"visible": [True, True, True]},
                                      {"title": "All"}])),
                              dict(label="energy_consumpt",
                                  method="update",
                                  args=[{"visible": [True, False, False]},
                                      {"title": "energy_consumpt"}])),
                              dict(label="full_temp",
                                  method="update",
                                  args=[{"visible": [False, True, False]},
                                      {"title": "full_temp"}])),
                              dict(label="full_humid",
                                  method="update",
                                  args=[{"visible": [False, False, True]}],

```

```
    ], [{"title": "full_humid"}])  
fig.show()
```

Consumo por día



Hourly Graphs


```

In [81]: columnas2005=['Hora', 'energy_consumpt_2005','full_temp_2005','full_humid_2005']
columnas2006=['Hora', 'energy_consumpt_2006','full_temp_2006','full_humid_2006']

hourly_df2005_graficas = merged_df[columnas2005]
hourly_df2006_graficas = merged_df[columnas2006]

hourly_df2005_graficas = hourly_df2005_graficas.rename(columns={'energy_consumpt_2005':'energy_consumpt',
                                                                'full_temp_2005':'full_temp',
                                                                'full_humid_2005':'full_humid'})

hourly_df2006_graficas = hourly_df2006_graficas.rename(columns={'energy_consumpt_2006':'energy_consumpt',
                                                                'full_temp_2006':'full_temp',
                                                                'full_humid_2006':'full_humid'})

hourly_f_graficas = pd.concat([hourly_df2005_graficas, hourly_df2006_graficas])
hourly_f_graficas = hourly_f_graficas.reset_index(drop=True)
hourly_f_graficas['Hora'] = 0

for i in range(len(hourly_f_graficas['Hora'])):
    hourly_f_graficas.Hora[i]=i+1

hourly_f_graficas

```

Out[81]:

	Hora	energy_consumpt	full_temp	full_humid
0	1	0.258616	0.063348	0.628866
1	2	0.067566	0.055807	0.642612
2	3	0.067566	0.048265	0.656357
3	4	0.067566	0.040724	0.670103
4	5	0.067566	0.034691	0.591065
...
17563	17564	0.531863	0.067766	0.812500
17564	17565	0.531863	0.067766	0.812500
17565	17566	0.531863	0.067766	0.812500
17566	17567	0.531863	0.067766	0.812500
17567	17568	0.531863	0.067766	0.812500

17568 rows × 4 columns

```

In [82]: import plotly.graph_objects as go

energy_consumpt_fig = hourly_f_graficas['energy_consumpt']
full_temp_fig = hourly_f_graficas['full_temp']
full_humid_fig = hourly_f_graficas['full_humid']

x = hourly_f_graficas['Hora']

fig = go.Figure()

fig.add_trace(go.Scatter(x=x, y=energy_consumpt_fig,
                        mode='lines',
                        name='energy_consumpt', marker=dict(color="DarkOrange"))),
fig.add_trace(go.Scatter(x=x, y=full_temp_fig,
                        mode='lines',
                        name='full_temp', marker=dict(color="Crimson"))),
fig.add_trace(go.Scatter(x=x, y=full_humid_fig,
                        mode='lines',
                        name='full_humid', marker=dict(color="RebeccaPurple")))

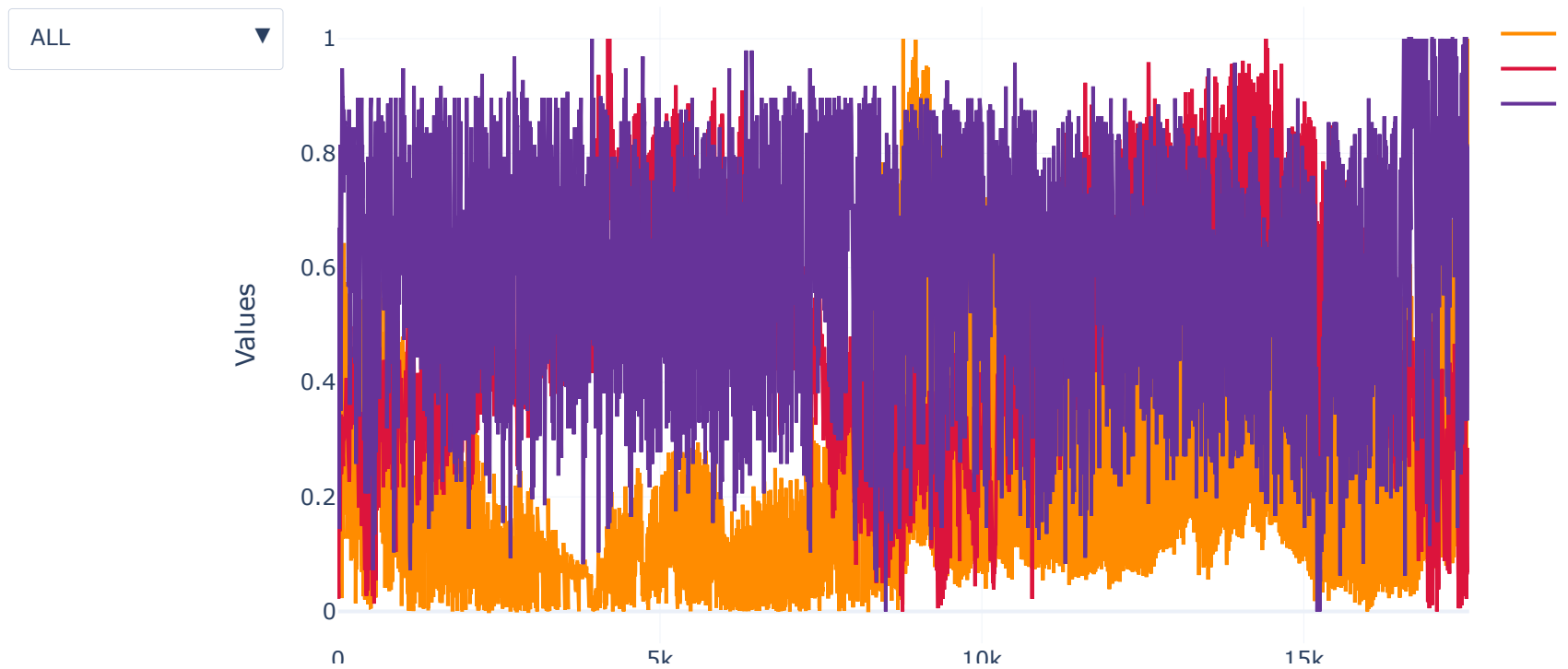
fig.update_layout(title_text='Consumo por hora',
                  xaxis_title='Hora', yaxis_title='Values')

fig.update_layout(template = 'plotly_white',
                  updatemenus=[
                      dict(active=0,
                          buttons=list([
                              dict(label="ALL",
                                  method="update",
                                  args=[{"visible": [True, True, True]},
                                      {"title": "All"}])),
                              dict(label="energy_consumpt",
                                  method="update",
                                  args=[{"visible": [True, False, False]},
                                      {"title": "energy_consumpt"}])),
                              dict(label="full_temp",
                                  method="update",
                                  args=[{"visible": [False, True, False]},
                                      {"title": "full_temp"}])),
                              dict(label="full_humid",
                                  method="update",
                                  args=[{"visible": [False, False, True]},
                                      {"title": "full_humid"}])),
                          ]))

```

```
fig, ax = plt.subplots(1, 1, {"title": "full_humid"})  
ax.plot(humid_data)  # Assuming humid_data is the variable name  
fig.show()
```

Consumo por hora



Prediction models

```
In [83]: !pip install pyarrow
```

```
Requirement already satisfied: pyarrow in c:\users\paomo\anaconda3\lib\site-packages (2.0.0)  
Requirement already satisfied: numpy>=1.14 in c:\users\paomo\anaconda3\lib\site-packages (from pyarrow) (1.22.2)
```

```
In [84]: from sklearn.model_selection import train_test_split  
from sklearn.metrics import precision_score  
from sklearn.metrics import accuracy_score  
from sklearn.model_selection import cross_val_score  
import warnings  
warnings.filterwarnings('ignore')  
from statsmodels.tsa.stattools import adfuller  
import pyarrow.parquet as pq  
import datetime  
import matplotlib.pyplot as plt  
from sklearn.linear_model import Ridge  
from sklearn.linear_model import LogisticRegression  
from sklearn.linear_model import ElasticNet  
from sklearn.tree import DecisionTreeRegressor  
from xgboost import XGBRegressor  
from sklearn.preprocessing import StandardScaler  
from sklearn.svm import SVR
```

Train, test, validation data

```
In [85]: X = hourly_f.drop('energy_consumpt', axis=1) # Features
y = hourly_f.energy_consumpt # Target variable
train_ratio = 0.65
validation_ratio = 0.25
test_ratio = 0.10

# train is now 65% of the entire data set
# the _junk suffix means that we drop that variable completely
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_ratio, shuffle=False)

# test is now 25% of the initial data set
# validation is now 10% of the initial data set
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, shuffle=False, test_size=test_ratio/(test_ratio
```

Arima Model

Time Series Forecasting based on past values.

```
In [86]: print("p-value:", adfuller(hourly_f['energy_consumpt']))
```

p-value: (-1.6473449396268942, 0.45842548275796274, 44, 17523, {'1%': -3.430723238497956, '5%': -2.861704957013964, '10%': -2.5668578023342805}, -78559.63348913024)

Since the p-value is less than .05, we reject the null hypothesis.

This means the time series is stationary. In other words, it has some time-dependent structure and does have constant variance over time.

```
In [87]: #!pip install pmdarima
```

```
In [88]: #!pip install statsmodels
#!pip install pmdarima
```

```
In [90]: from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
import pmdarima as pmd
'''import Lasso regression from sklearn library
'''
from sklearn.linear_model import Lasso
```

Evaluación de modelos

Support Vector

Support Vector Machine (SVM) is a supervised learning algorithm that can be used for classification or regression problems. It's based on creating hyperplanes that are used to segregate the data in different groups. These planes can be configured with different kernel functions, being able to classificate different objects with non-linear hyper-planes.

RandomForest

Random Forest is a supervised learning algorithm that can be used for either classification or regression problems, it is also an ensemble method. The way it works is really simple, the algorithm creates N number of different Classification And Regression Trees (CART), with different samples and initial values. All these models are later evaluated and the final prediction is a function of all the models.

DecisionTreeRegressor

Supervised learning algorithm that can be used for both classification and regression. The algorithm breaks down the dataset into smaller subsets, while it tries to gain insights and information from each one of the features.

Decision trees are represented graphically, where a 'branch-like' structure can be appreciated. There are two types of nodes; decision nodes, which represent the features and attributes; leaf nodes represent the outcome. Branches among the nodes represent the decision rules that are applied.

One of the main advantages of this model is that it allows the user to follow thoroughly the process of deciding and determining an outcome. It can be appreciated in a simple way which are the key features for determining the outcome of the algorithm. Trees can be transformed into a set of rules, where each path from the root to a leaf node is a certain rule.

XGBRegressor

The Extra Gradient Boosting model is actually a technique applied to Decision Trees, where the main objective is to 'boost' the results that are generated through modeling. These results seek to enhance a certain metric that wants to be either minimized or maximized, depending on what the metric is measuring (for example, if we consider the mean squared error, enhancing the result of the metric refers to reducing its value). Then, the algorithm repeats itself several iterations, enhancing the metric; this way, the regressor gains power and performs better. Only the models that actually get to enhance the metric will be considered for building the next one; this way, we make sure that results will improve eventually.

In order to boost the algorithm, the main idea is to generate several weaker models, which iterate themselves and consider the previous results; by doing this, the current model will enhance the previous results and, eventually, better and more stable results will be obtained. In order to generate better models, an optimization algorithm must be employed (such as gradient descent).

Hourly

```
In [91]: from sklearn.ensemble import RandomForestRegressor
```

```
In [92]: models_h = []
models_h.append(('Support Vector', SVR(kernel = 'linear')))
#models.append(('LR', LogisticRegression(max_iter = 300000)))
models_h.append(('Random Forest', RandomForestRegressor(n_estimators = 100, random_state = 0)))
models_h.append(('XGB Regressor', XGBRegressor()))
models_h.append(('Decision Tree Regressor', DecisionTreeRegressor(max_depth=5)))
models_h.append(('Elastic Net', ElasticNet(alpha = 1)))
models_h.append(('Ridge Regression', Ridge(alpha = 1)))
```

```
In [93]: from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
In [94]: def mae(y_true, predictions):
y_true, predictions = np.array(y_true), np.array(predictions)
return np.mean(np.abs(y_true - predictions))
```



```

In [95]: import time

names = []
times = []
mseError = []
maeError = []
rmseError = []
fits = []
preds = []

for name, model in models_h:
    start_time = time.time()
    fits.append(model.fit(x_train, y_train))
    print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
    times.append(round((time.time() - start_time),6))
    y_pred = model.predict(x_test)
    preds.append(y_pred)
    mseError.append(mean_squared_error(y_test,y_pred))
    rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
    maeError.append(mae(y_test,y_pred))
    names.append(name)

```

Execution time: 1.230997 ms for model: SVR(kernel='linear')

Execution time: 1.469089 ms for model: RandomForestRegressor(random_state=0)

Execution time: 0.262914 ms for model: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)

Execution time: 0.008 ms for model: DecisionTreeRegressor(max_depth=5)

Execution time: 0.069 ms for model: ElasticNet(alpha=1)

Execution time: 0.014 ms for model: Ridge(alpha=1)

```

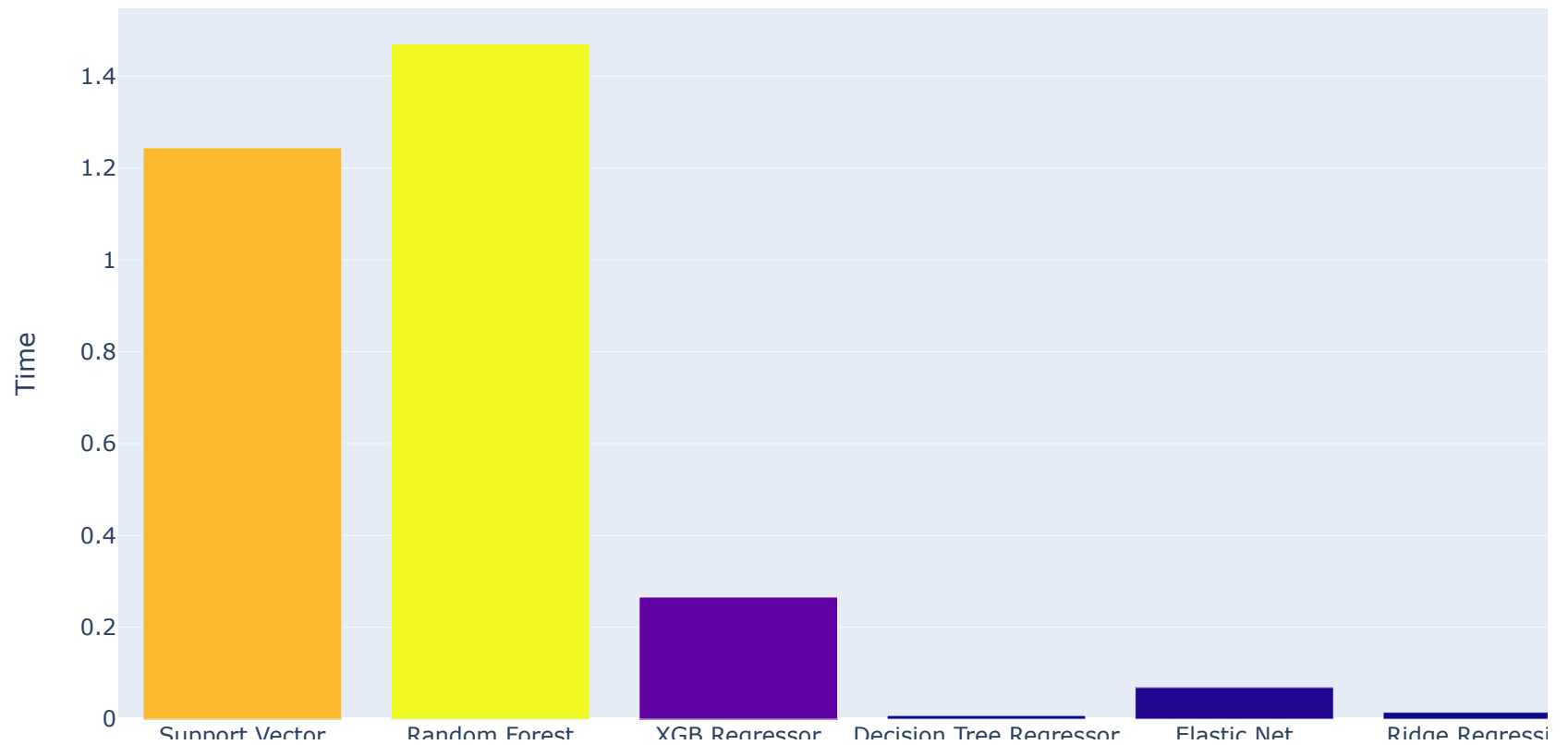
In [96]: start_time = time.time()
         arima_model = pmd.auto_arima(y_train,
                                     start_p=0,d = 1,start_q=0,
                                     test="adf", supress_warnings = True)
         y_pred = arima_model.predict(n_periods = len(x_test))
         preds.append(y_pred)
         mseError.append(mean_squared_error(y_test,y_pred))
         rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
         maeError.append(mae(y_test,y_pred))
         names.append('Arima Model')
         print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
         names2=['Support Vector','Random Forest','XGB Regressor','Decision Tree Regressor','Elastic Net','Ridge Regressi

```

Execution time: 141.027156 ms for model: Ridge(alpha=1)

Execution Time

```
In [97]: time = pd.DataFrame({'Name':names2, 'Time': times})  
  
fig = px.bar(time, x = "Name", y = "Time", color = "Time")  
fig.show()
```

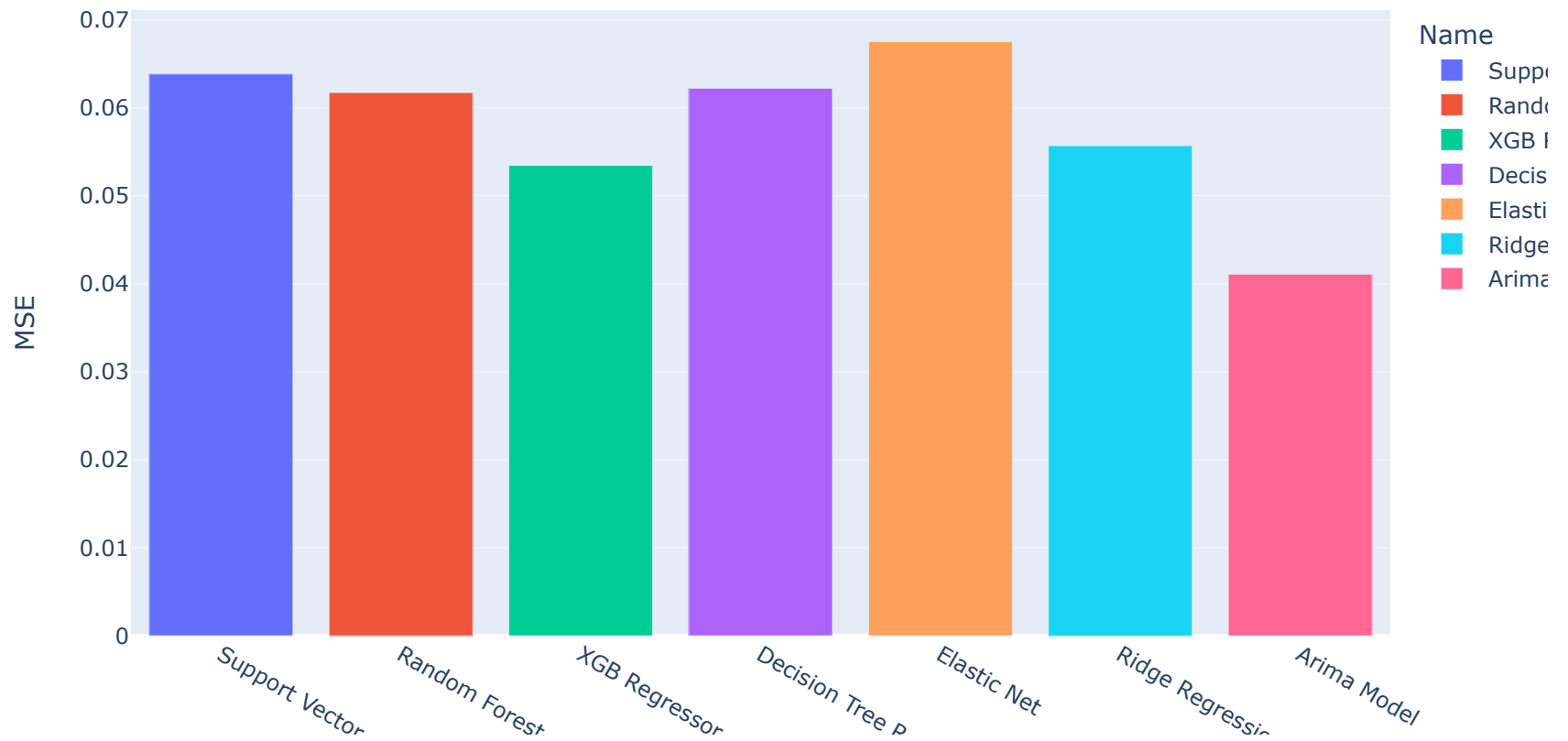


In this graph we do not include the execution time of the Arima Model because it was bigger in comparison with the other models.

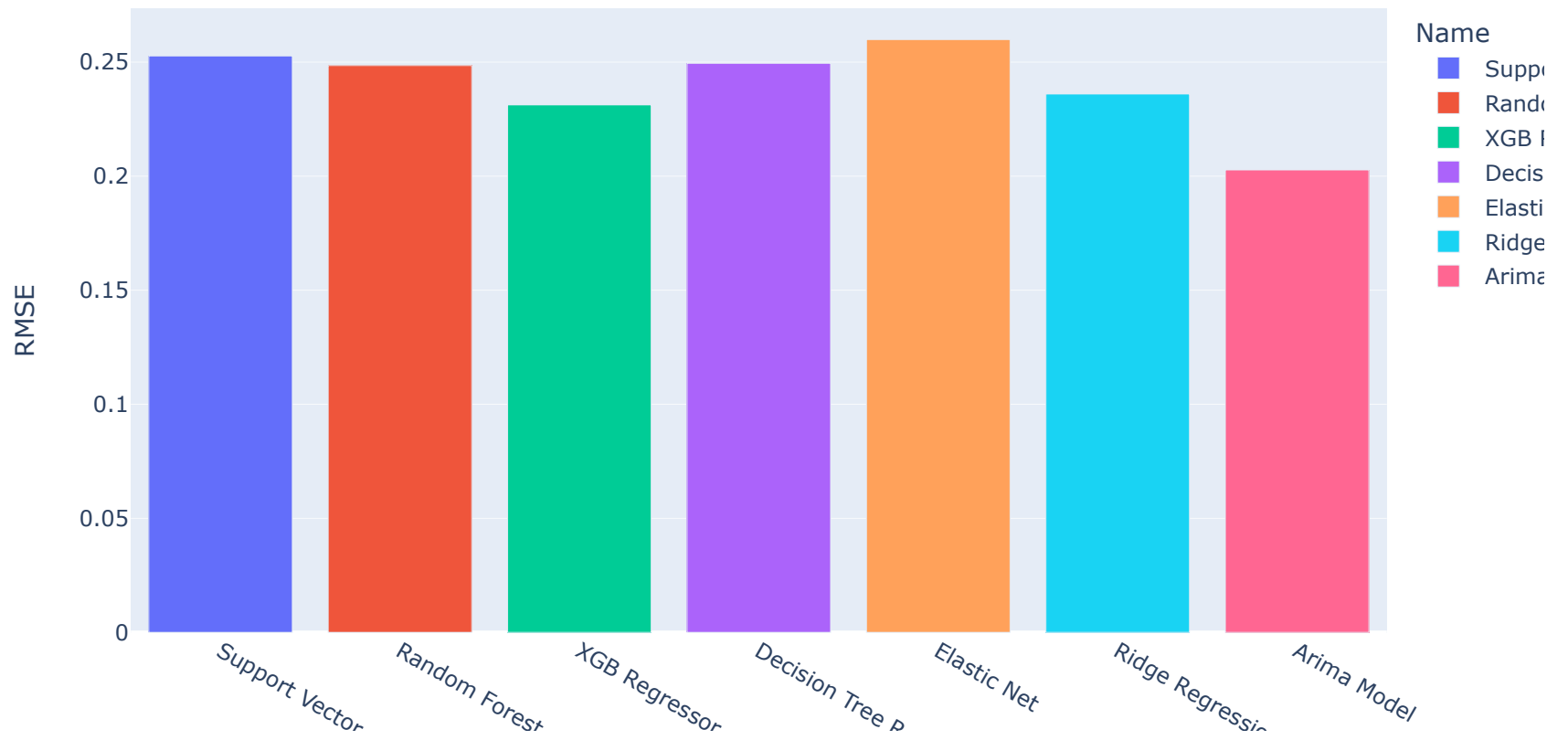
Error Visualization

```
In [98]: error1 = pd.DataFrame({'Name':names, 'MSE': mseError})
error2 = pd.DataFrame({'Name':names, 'RMSE': rmseError})
error3 = pd.DataFrame({'Name':names, 'MAE': maeError})
```

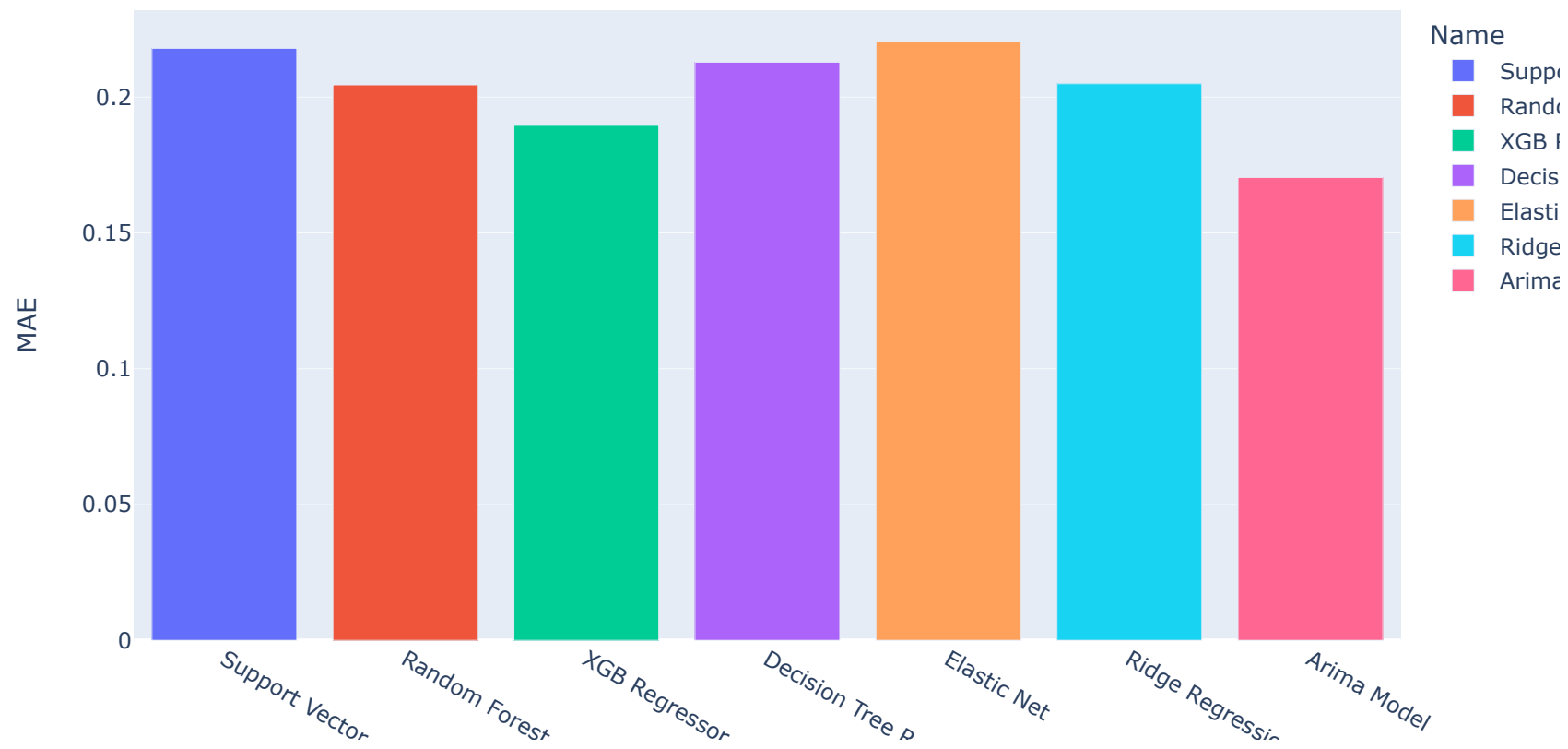
```
In [99]: fig = px.bar(error1, x = "Name", y = "MSE", color = "Name")
fig.show()
```



```
In [100]: fig = px.bar(error2, x = "Name", y = "RMSE", color = "Name")  
fig.show()
```



```
In [101]: fig = px.bar(error3, x = "Name", y = "MAE", color = "Name")  
fig.show()
```



Daily

```
In [102]: models_d = []
models_d.append(('Support Vector', SVR(kernel = 'linear')))
#models_d.append(('LR', LogisticRegression(max_iter = 300000)))
models_d.append(('Random Forest', RandomForestRegressor(n_estimators = 100, random_state = 0)))
models_d.append(('XGB Regressor', XGBRegressor()))
models_d.append(('Decision Tree Regressor', DecisionTreeRegressor(max_depth=5)))
models_d.append(('Elastic Net', ElasticNet(alpha = 1)))
models_d.append(('Ridge Regression', Ridge(alpha = 1)))
```

```
In [103]: X = daily_f.drop('energy_consumpt', axis=1) # Features
y = daily_f.energy_consumpt # Target variable
train_ratio = 0.65
validation_ratio = 0.25
test_ratio = 0.10

# train is now 65% of the entire data set
# the _junk suffix means that we drop that variable completely
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_ratio, shuffle=False)

# test is now 25% of the initial data set
# validation is now 10% of the initial data set
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, shuffle=False, test_size=test_ratio/(test_ratio
```

```

In [104]: import time

names = []
times = []
mseError = []
maeError = []
rmseError = []
fits = []
preds = []

for name, model in models_d:
    start_time = time.time()
    fits.append(model.fit(x_train, y_train))
    print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
    times.append(round((time.time() - start_time),6))
    y_pred = model.predict(x_test)
    preds.append(y_pred)
    mseError.append(mean_squared_error(y_test,y_pred))
    rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
    maeError.append(mae(y_test,y_pred))
    names.append(name)

```

```

Execution time: 0.006996 ms for model: SVR(kernel='linear')
Execution time: 0.141021 ms for model: RandomForestRegressor(random_state=0)
Execution time: 0.082977 ms for model: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
           colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
           importance_type='gain', interaction_constraints='',
           learning_rate=0.300000012, max_delta_step=0, max_depth=6,
           min_child_weight=1, missing=nan, monotone_constraints='()',
           n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
           reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
           tree_method='exact', validate_parameters=1, verbosity=None)
Execution time: 0.001998 ms for model: DecisionTreeRegressor(max_depth=5)
Execution time: 0.001 ms for model: ElasticNet(alpha=1)
Execution time: 0.001998 ms for model: Ridge(alpha=1)

```



```

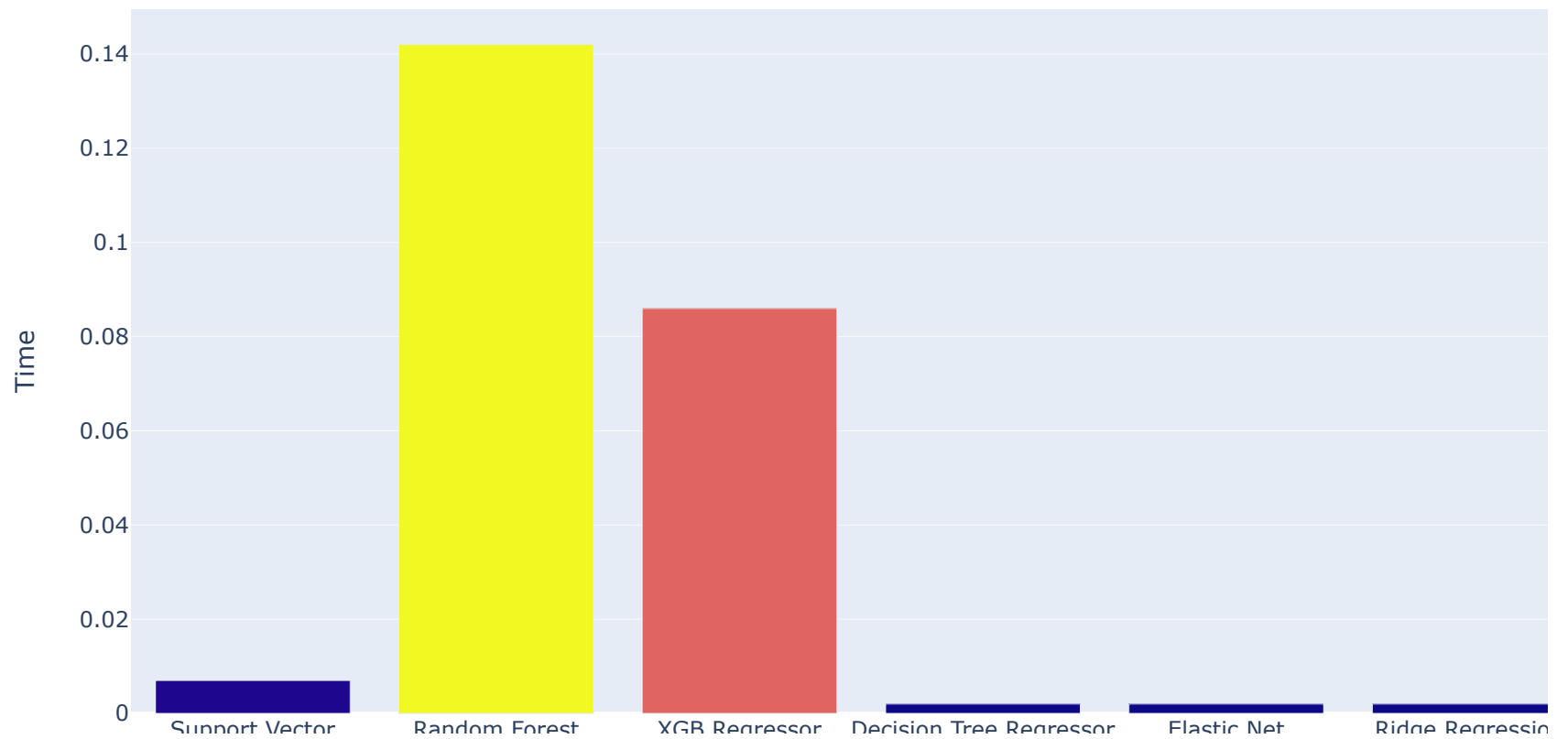
In [105]: start_time = time.time()
          arima_model = pmd.auto_arima(y_train,
                                     start_p=0,d = 1,start_q=0,
                                     test="adf", supress_warnings = True)
          y_pred = arima_model.predict(n_periods = len(x_test))
          preds.append(y_pred)
          mseError.append(mean_squared_error(y_test,y_pred))
          rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
          maeError.append(mae(y_test,y_pred))
          names.append('Arima Model')
          print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
          names2=['Support Vector','Random Forest','XGB Regressor','Decision Tree Regressor','Elastic Net','Ridge Regressi

```

Execution time: 0.974001 ms for model: Ridge(alpha=1)

Execution Time

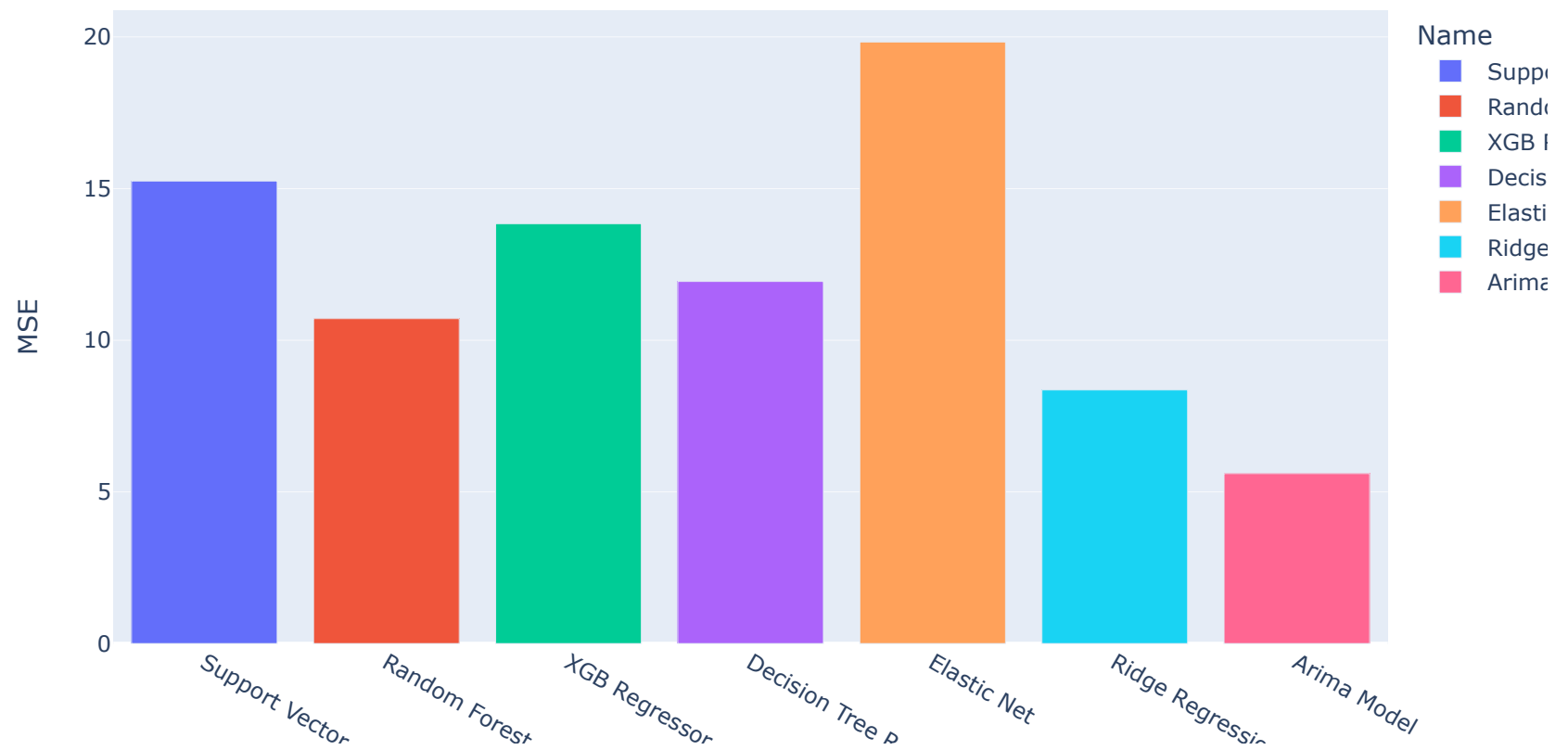
```
In [106]: time = pd.DataFrame({'Name':names2, 'Time': times})  
  
fig = px.bar(time, x = "Name", y = "Time", color = "Time")  
fig.show()
```



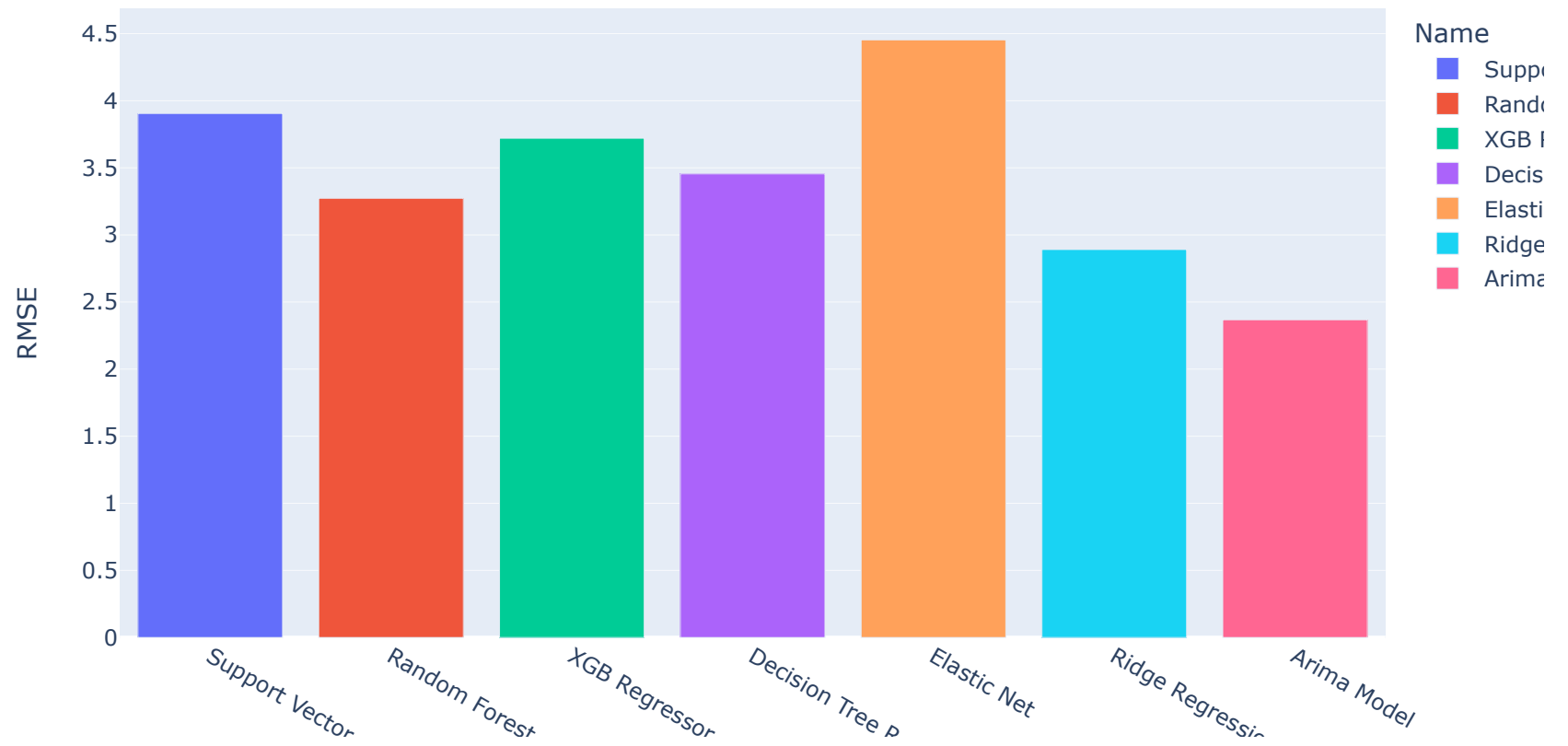
Error Visualization

```
In [107]: error1 = pd.DataFrame({'Name':names, 'MSE': mseError})
error2 = pd.DataFrame({'Name':names, 'RMSE': rmseError})
error3 = pd.DataFrame({'Name':names, 'MAE': maeError})
```

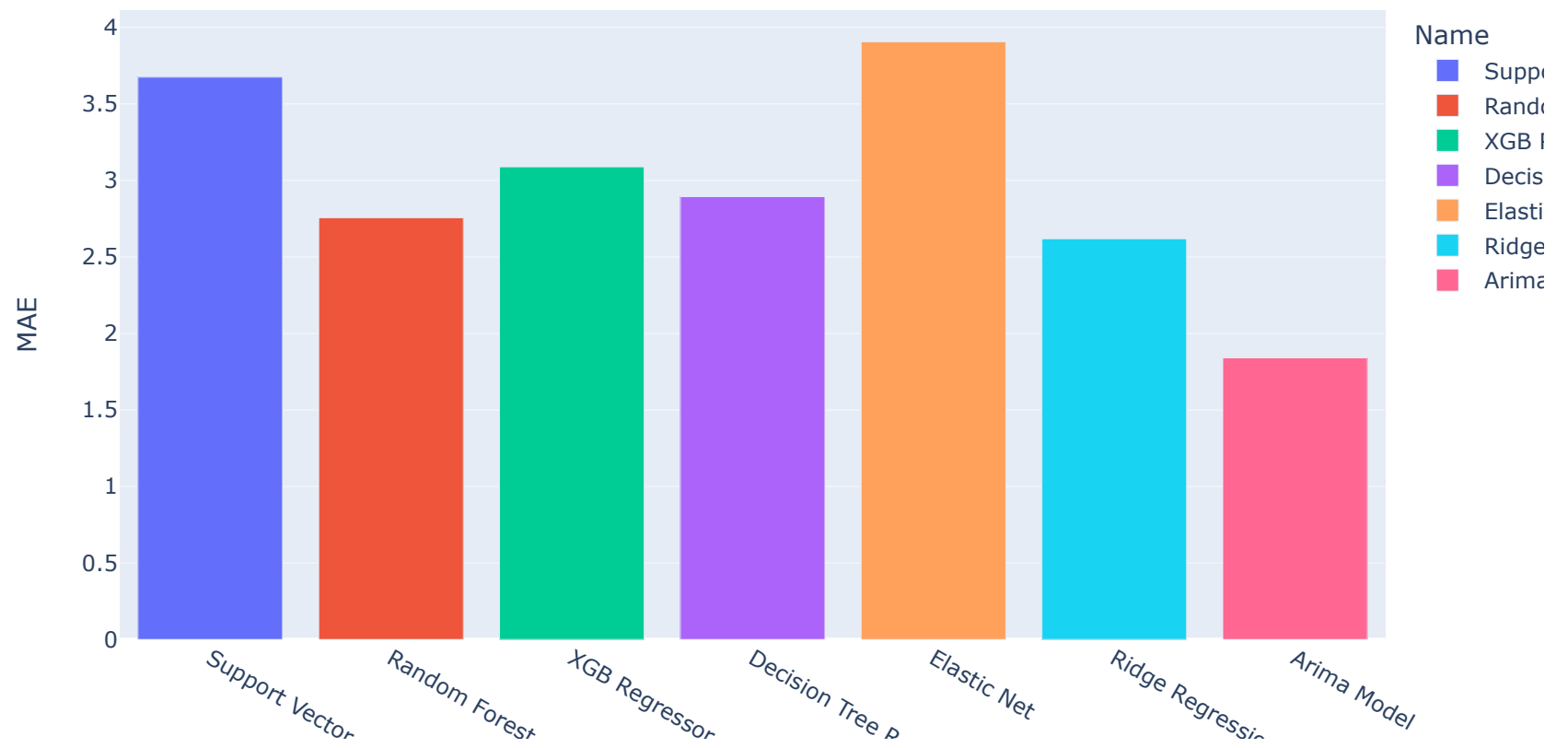
```
In [108]: fig = px.bar(error1, x = "Name", y = "MSE", color = "Name")
fig.show()
```



```
In [109]: fig = px.bar(error2, x = "Name", y = "RMSE", color = "Name")  
fig.show()
```



```
In [110]: fig = px.bar(error3, x = "Name", y = "MAE", color = "Name")  
fig.show()
```



Weekly

```
In [111]: models_w = []
models_w.append(('Support Vector', SVR(kernel = 'linear')))
#models.append(('LR', LogisticRegression(max_iter = 300000)))
models_w.append(('Random Forest', RandomForestRegressor(n_estimators = 100, random_state = 0)))
models_w.append(('XGB Regressor', XGBRegressor()))
models_w.append(('Decision Tree Regressor', DecisionTreeRegressor(max_depth=5)))
models_w.append(('Elastic Net', ElasticNet(alpha = 1)))
models_w.append(('Ridge Regression', Ridge(alpha = 1)))
```

```
In [112]: X = weekly_f.drop('energy_consumpt', axis=1) # Features
y = weekly_f.energy_consumpt # Target variable
train_ratio = 0.65
validation_ratio = 0.25
test_ratio = 0.10

# train is now 65% of the entire data set
# the _junk suffix means that we drop that variable completely
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_ratio, shuffle=False)

# test is now 25% of the initial data set
# validation is now 10% of the initial data set
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, shuffle=False, test_size=test_ratio/(test_ratio
```

In [113]: `import time`

```
names = []
times = []
mseError = []
maeError = []
rmseError = []
fits = []
preds = []

for name, model in models_w:
    start_time = time.time()
    fits.append(model.fit(x_train, y_train))
    print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
    times.append(round((time.time() - start_time),6))
    y_pred = model.predict(x_test)
    preds.append(y_pred)
    mseError.append(mean_squared_error(y_test,y_pred))
    rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
    maeError.append(mae(y_test,y_pred))
    names.append(name)
```

Execution time: 0.002059 ms for model: SVR(kernel='linear')

Execution time: 0.078986 ms for model: RandomForestRegressor(random_state=0)

Execution time: 0.049998 ms for model: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)

Execution time: 0.001001 ms for model: DecisionTreeRegressor(max_depth=5)

Execution time: 0.007 ms for model: ElasticNet(alpha=1)

Execution time: 0.000999 ms for model: Ridge(alpha=1)

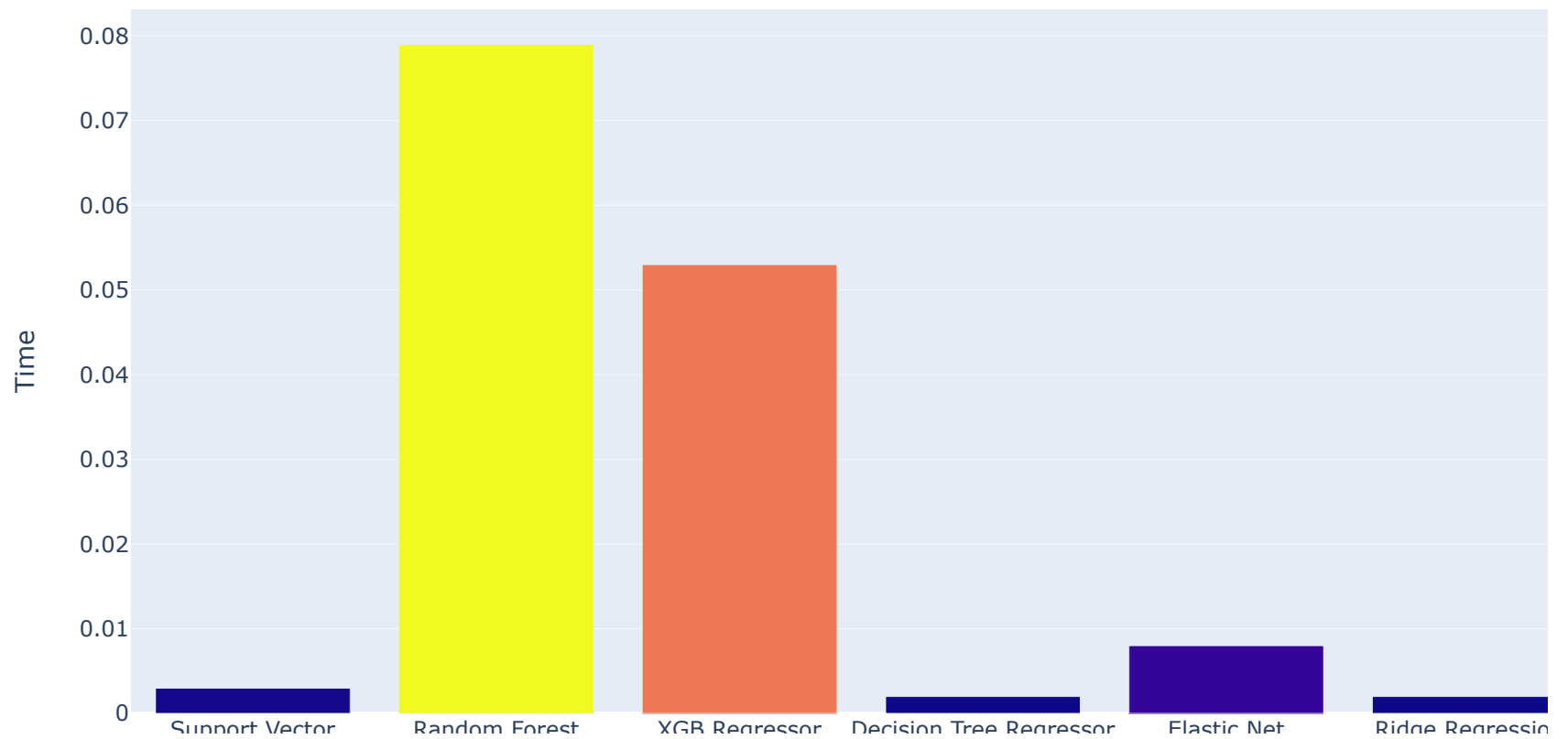
```
In [114]: arima_model = pmd.auto_arima(y_train,
                                     start_p=0,d = 1,start_q=0,
                                     test="adf", supress_warnings = True)
y_pred = arima_model.predict(n_periods = len(x_test))
preds.append(y_pred)
mseError.append(mean_squared_error(y_test,y_pred))
rmseError.append(sqrt(mean_squared_error(y_test,y_pred)))
maeError.append(mae(y_test,y_pred))
names.append('Arima Model')
print("Execution time: " + str(round((time.time() - start_time),6)) + ' ms' + ' for model: ' + str(model) )
```

Execution time: 0.284001 ms for model: Ridge(alpha=1)

Execution Time


```
In [115]: time = pd.DataFrame({'Name':names2, 'Time': times})

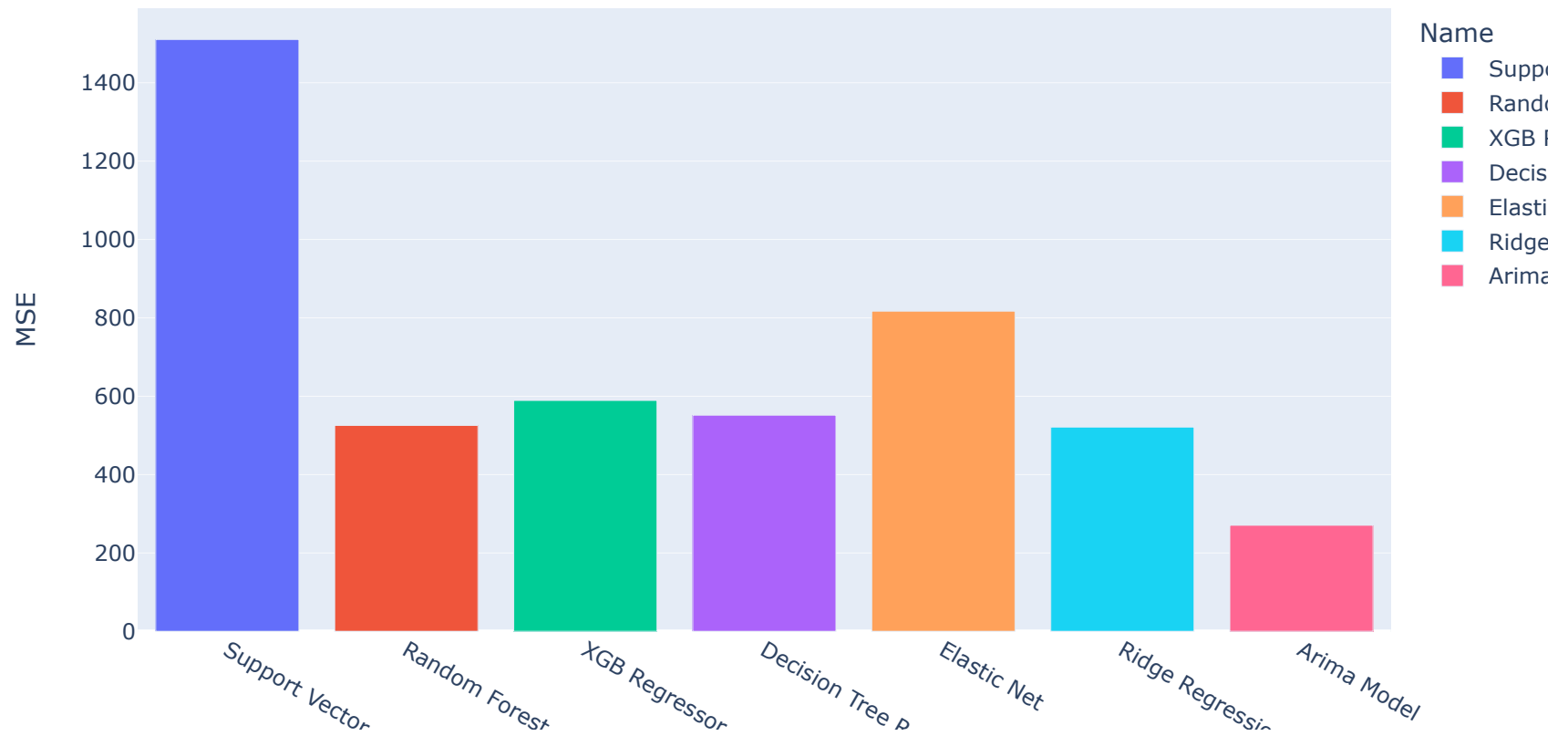
fig = px.bar(time, x = "Name", y = "Time", color = "Time")
fig.show()
```



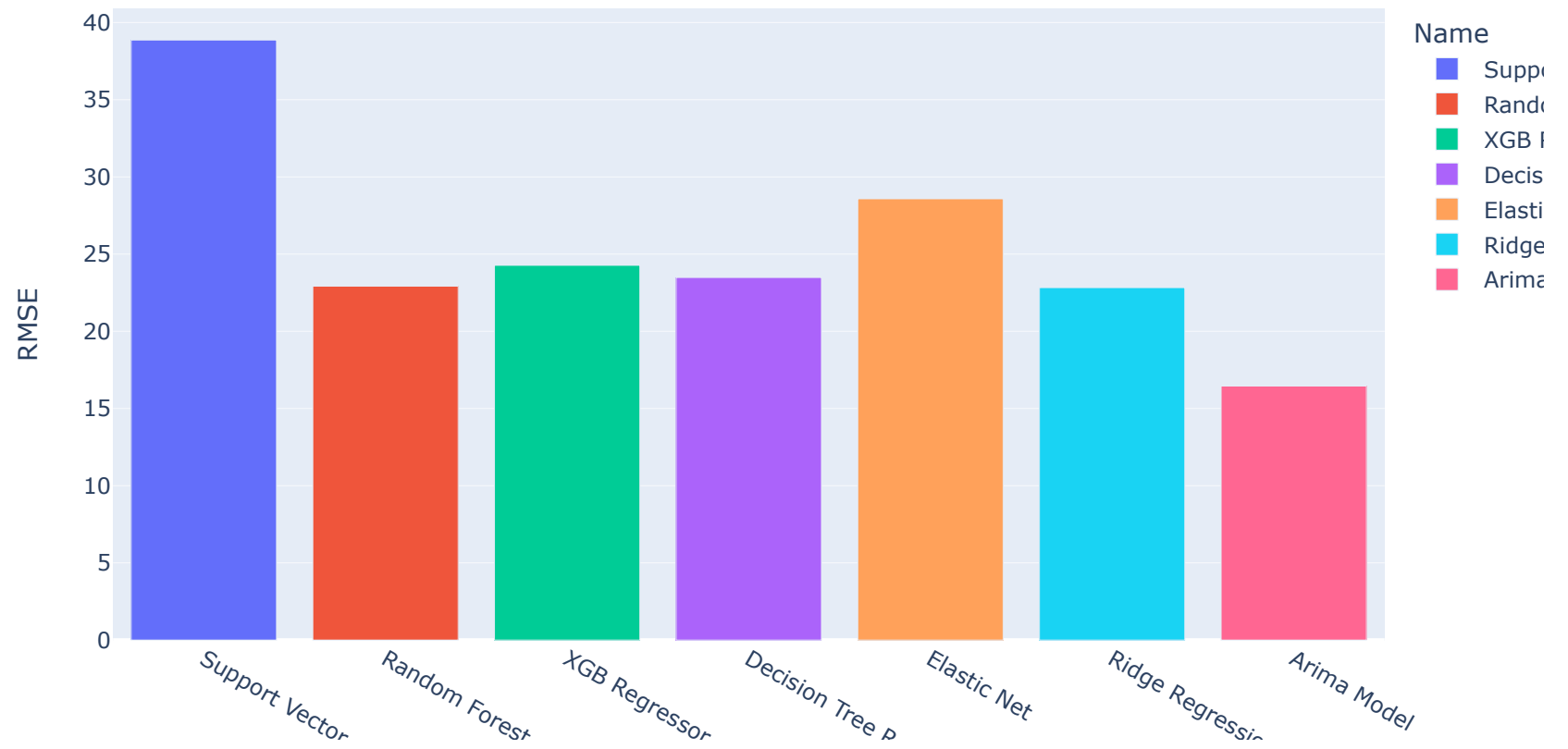
Error Visualization

```
In [116]: error1 = pd.DataFrame({'Name':names, 'MSE': mseError})
          error2 = pd.DataFrame({'Name':names, 'RMSE': rmseError})
          error3 = pd.DataFrame({'Name':names, 'MAE': maeError})
```

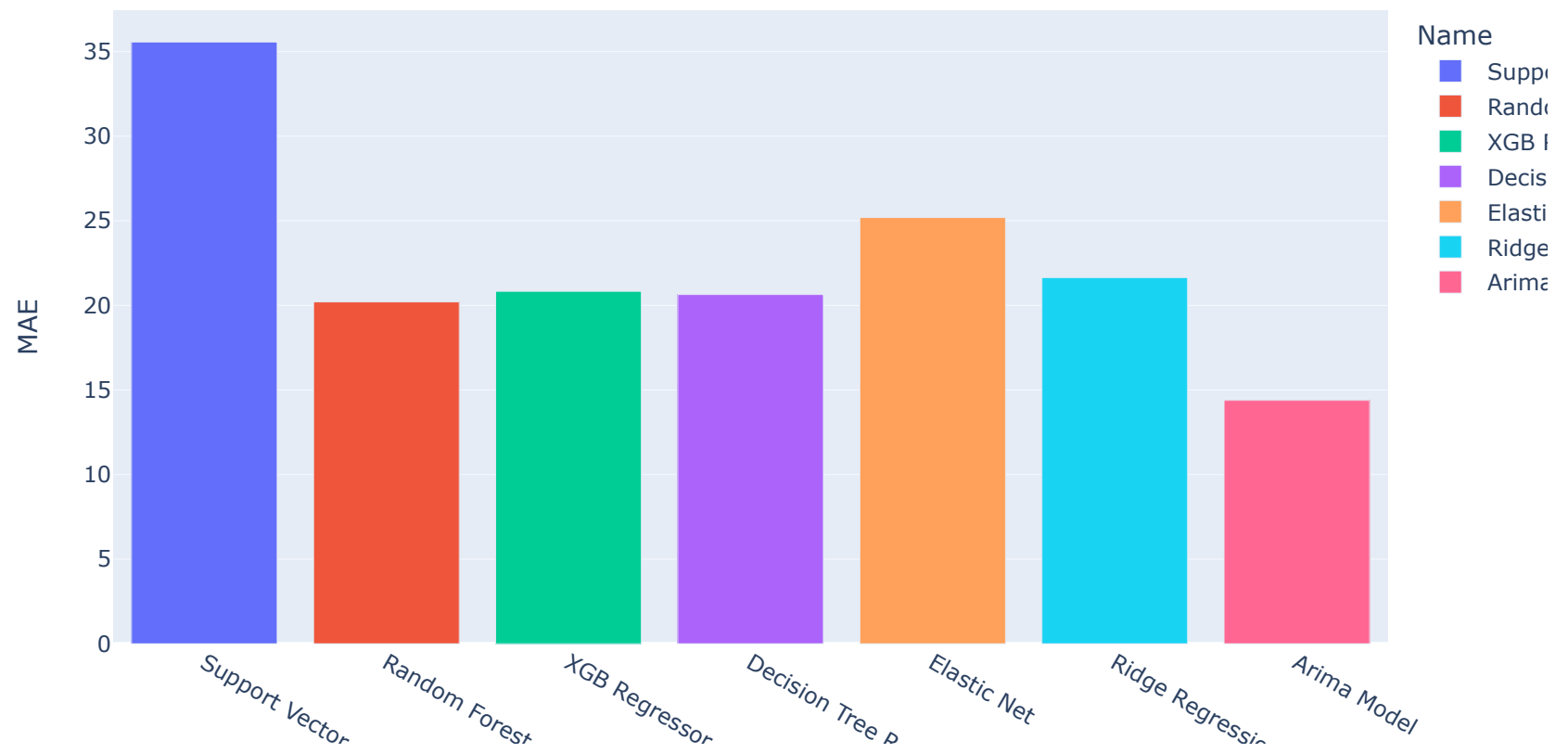
```
In [117]: fig = px.bar(error1, x = "Name", y = "MSE", color = "Name")
          fig.show()
```



```
In [118]: fig = px.bar(error2, x = "Name", y = "RMSE", color = "Name")  
fig.show()
```



```
In [119]: fig = px.bar(error3, x = "Name", y = "MAE", color = "Name")  
fig.show()
```



Optimization of Hyperparameters

Decision Tree Regressor

```
In [120]: parameters={"splitter":["best","random"],
    "max_depth" : [1,3,5,7,9],
    "min_samples_leaf":[1,2,4,5,8,9],
    "min_weight_fraction_leaf":[0.1,0.3,0.5,0.7,0.9],
    "max_features":["auto","sqrt",None],
    "max_leaf_nodes":[None,10,30,50,70] }
```

```
In [121]: # calculating different regression metrics

from sklearn.model_selection import GridSearchCV

DT = DecisionTreeRegressor()
#lr_grid = GridSearchCV(estimator=Lr

tuning_model = GridSearchCV(DT,param_grid=parameters,scoring='neg_mean_squared_error',cv=3,verbose=3)
```

In [122]: `import time`

```
start_time = time.time()
tuning_model_result = tuning_model.fit(X, y)
# Summarize results
print("Best: %f using %s" % (tuning_model_result.best_score_, tuning_model_result.best_params_))
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=best, score=-654.465, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=best
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=best, score=-1133.043, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random, score=-1035.318, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random, score=-1036.067, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.1,
splitter=random, score=-790.129, total= 0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.3,
splitter=best
```

```
In [123]: tuning_model_result.fit(x_train,y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   0.0s remaining:   0.0s  
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   0.0s remaining:   0.0s
```

```
In [124]: tuned_pred = tuning_model_result.predict(x_test)
```

```
In [125]: # without hyperparameter tuning
```

```
from sklearn import metrics  
  
print('MAE:', metrics.mean_absolute_error(y_test,preds[3]))  
print('MSE:', metrics.mean_squared_error(y_test, preds[3]))  
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, preds[3])))
```

```
MAE: 20.641310906854418  
MSE: 551.5408667775441  
RMSE: 23.484907212453365
```

In [126]: *# With hyperparameter tuned*

```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test,tuned_pred))
print('MSE:', metrics.mean_squared_error(y_test, tuned_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, tuned_pred)))
```

```
MAE: 20.27853301763724
MSE: 598.3121745241366
RMSE: 24.46042057128488
```

We do not see a extraordinary improvement in the metrics, so we didn't use the tuned hyperparameters

Conclusion

Regarding the analysis and modeling generated, the team considers that we could work adequately with the information and data, and that we could generate appropriate and accurate models that can be employed successfully for predicting potential values and information. Through the development of the case study, we had to fully understand how to label information and how the climatic variables change throughout different periods of time, in order to develop a deep-analysis.

The analysis took into consideration several transformations, variables, labels, time periods; in order to achieve a complete study of the case. To prepare the data for modeling, the team studied the correlation among variables, performed transformations, considered the metrics, visualized insights through several graphs. Once we had completed the modifications, we could understand fully the data, so that we could continue with modeling and prediction generation.

The modeling was generated using a wide variety of 'models', each one of them considers unique parameters that aim to adjust in a more efficient way the data, so that its trends and insights remain represented through the model. All of the models performed in a distinct way, and they were adjusted with their unique hyperparameters to improve results. The main goal of creating a full set of models that use different regression methods was to compare the performance, in order to determine which model is better aligned with the objectives of the study.

For determining better performance, the team took into consideration the time in which the models get to study data and determines the 'formula' for predictions and new values. In that aspect, all of the models actually have an acceptable performance, as they do not have a quite significant difference in the final running time; all of them take less than 10 seconds. Then, for measuring the differences (error) between the predicted values and the actual values, three metrics were evaluated; the mean squared error MSE, the root mean squared

error RMSE, the mean absolute error MAE. Those metrics have a particular focus; two of them care more about outliers and bigger distances, while the other considers more the absolute difference; they also function in a particular manner for certain types of values, therefore it was worth it to take them all into account. The error visualization graphs actually indicated that, depending on the frequency in which data is evaluated (either hourly, daily or weekly), differences between predictions and measured values change drastically. Actually, all of the models perform similarly, with none of them establishing a dominance and considerably lower error metrics.

After finishing working on the case study, the team considers that all models were successfully train, as they work correctly when compared to the actual values that were measured. However, it is important to take into account that some additional training and measures can be taken so that an optimal model is found. Working with some other data transformation, eliminating missing values, creating more variables, working with less components and applying principal componen analysis, among other options, could potentially help for finding better modeling ways, and should be applied for further studies.

References

REFERENCES

S. (2021, 6 febrero). Ridge Regression: Simple Definition. Statistics How To. [https://www.statisticshowto.com/ridge-regression/#:%7E:text=Share%20on,\(correlations%20between%20predictor%20variables\),\(https://www.statisticshowto.com/ridge-regression/#:%7E:text=Share%20on,\(correlations%20between%20predictor%20variables\)\)](https://www.statisticshowto.com/ridge-regression/#:%7E:text=Share%20on,(correlations%20between%20predictor%20variables),(https://www.statisticshowto.com/ridge-regression/#:%7E:text=Share%20on,(correlations%20between%20predictor%20variables))).

Shumway, R. H., & Stoffer, D. S. (2017). Time Series Analysis and Its Applications: With R Examples (4th 2017 ed.). Springer.

Sunil. (2017). Understanding Support Vector Machine(SVM) algorithm from examples (along with code). March 1st 2022, from Analytics Vidhya Site: <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/> (<https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>).

Analytics Vidhya. (2016). Tree Based Algorithms: A Complete Tutorial from Scratch (in R & Python). March 1st 2022, From Analytics Vidhya: <https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/> (<https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/>).

G. Choueiry. (2022). Square Root Transformation: A Beginner's Guide. March 1st 2022, from Quantifying Health: <https://quantifyinghealth.com/square-root-transformation/> (<https://quantifyinghealth.com/square-root-transformation/>).

Wang, Li; Zhu, Ji; Zou, Hui (2006). "The doubly regularized support vector machine" (PDF). *Statistica Sinica*. 16: 589–615.

