In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import tensorflow as tf
import xgboost as xgb
import transformers
from transformers import AutoModel, BertTokenizerFast
import os
import warnings
from tensorflow.keras.layers import Dense, LSTM, Conv1D, MaxPooling
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss, ccf
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, Min
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from math import sqrt

%matplotlib inline
```

In [2]:
```
df_energy = pd.read_csv('/Users/pro/Desktop/energy_dataset.csv')
df_weather= pd.read_csv('/Users/pro/Downloads/weather_features.csv'
```

In [3]:
```
df_energy.head()
```

Out[3]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0

5 rows × 29 columns

In [4]:
```
df_energy = df_energy.drop(['generation fossil coal-derived gas','g
                            'generation fossil peat', 'generation g
                            'generation hydro pumped storage aggreg
                            'generation wind offshore', 'forecast w
                            'total load forecast', 'forecast solar 
                            'forecast wind onshore day ahead'],
                            axis=1)
```

In [5]:
```
df_energy.describe().round(2)
```

Out[5]:

	generation biomass	generation fossil brown coal/lignite	generation fossil gas	generation fossil hard coal	generation fossil oil	generation hydro pumped storage consumption	generation hydro run-of-river and poundage
count	35045.00	35046.00	35046.00	35046.00	35045.00	35045.00	35045.00
mean	383.51	448.06	5622.74	4256.07	298.32	475.58	972.12
std	85.35	354.57	2201.83	1961.60	52.52	792.41	400.78
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	333.00	0.00	4126.00	2527.00	263.00	0.00	637.00
50%	367.00	509.00	4969.00	4474.00	300.00	68.00	906.00
75%	433.00	757.00	6429.00	5838.75	330.00	616.00	1250.00
max	592.00	999.00	20034.00	8359.00	449.00	4523.00	2000.00

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 18 columns):
 #   Column                                       Non-Null Count
Dtype
---  ------                                       --------------
-----
 0   time                                         35064 non-null
object
 1   generation biomass                           35045 non-null
float64
 2   generation fossil brown coal/lignite         35046 non-null
float64
 3   generation fossil gas                        35046 non-null
float64
 4   generation fossil hard coal                  35046 non-null
float64
 5   generation fossil oil                        35045 non-null
float64
 6   generation hydro pumped storage consumption  35045 non-null
float64
 7   generation hydro run-of-river and poundage   35045 non-null
float64
 8   generation hydro water reservoir             35046 non-null
float64
 9   generation nuclear                           35047 non-null
float64
 10  generation other                             35046 non-null
float64
 11  generation other renewable                   35046 non-null
float64
 12  generation solar                             35046 non-null
float64
 13  generation waste                             35045 non-null
float64
 14  generation wind onshore                      35046 non-null
float64
 15  total load actual                            35028 non-null
float64
 16  price day ahead                              35064 non-null
float64
 17  price actual                                 35064 non-null
float64
dtypes: float64(17), object(1)
memory usage: 4.8+ MB
```

In [7]:
```
df_energy['time'] = pd.to_datetime(df_energy['time'], utc=True, inf
df_energy = df_energy.set_index('time')
```

In [8]:
```
# Find NaNs and duplicates in df_energy

print('There are {} missing values or NaNs in df_energy.'
      .format(df_energy.isnull().values.sum()))

temp_energy = df_energy.duplicated(keep='first').sum()

print('There are {} duplicate rows in df_energy based on all column
      .format(temp_energy))
```

```
There are 292 missing values or NaNs in df_energy.
There are 0 duplicate rows in df_energy based on all columns.
```

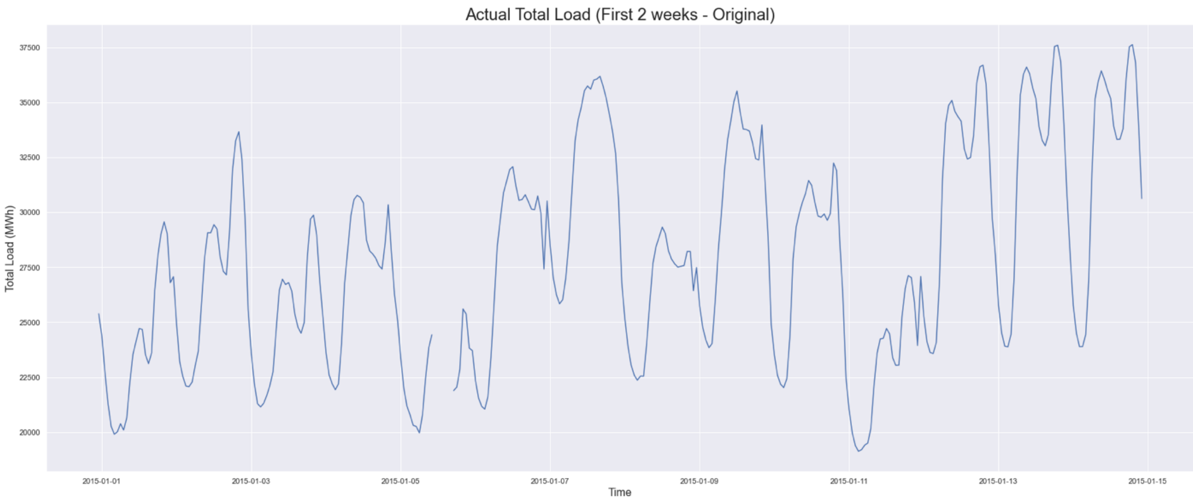
In [9]:
```
# Find the number of NaNs in each column

df_energy.isnull().sum(axis=0)
```

Out[9]:
```
generation biomass                              19
generation fossil brown coal/lignite            18
generation fossil gas                           18
generation fossil hard coal                     18
generation fossil oil                           19
generation hydro pumped storage consumption     19
generation hydro run-of-river and poundage      19
generation hydro water reservoir                18
generation nuclear                              17
generation other                                18
generation other renewable                      18
generation solar                                18
generation waste                                19
generation wind onshore                         18
total load actual                               36
price day ahead                                  0
price actual                                     0
dtype: int64
```

In [10]:
```
# Define a function to plot different types of time-series

def plot_series(df=None, column=None, series=pd.Series([]),
                label=None, ylabel=None, title=None, start=0, end=N
    """
    Plots a certain time-series which has either been loaded in a d
    and which constitutes one of its columns or it a custom pandas 
    created by the user. The user can define either the 'df' and th
    or the 'series' and additionally, can also define the 'label', 
    'ylabel', the 'title', the 'start' and the 'end' of the plot.
    """
    sns.set()
    fig, ax = plt.subplots(figsize=(30, 12))
    ax.set_xlabel('Time', fontsize=16)
    if column:
        ax.plot(df[column][start:end], label=label)
        ax.set_ylabel(ylabel, fontsize=16)
    if series.any():
        ax.plot(series, label=label)
        ax.set_ylabel(ylabel, fontsize=16)
    if label:
        ax.legend(fontsize=16)
    if title:
        ax.set_title(title, fontsize=24)
    ax.grid(True)
    return ax
```

```
/Users/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.p
y:3: DeprecationWarning: The default dtype for empty Series will b
e 'object' instead of 'float64' in a future version. Specify a dty
pe explicitly to silence this warning.
  This is separate from the ipykernel package so we can avoid doin
g imports until
```

In [11]:
```
# Zoom into the plot of the hourly (actual) total load

ax = plot_series(df=df_energy, column='total load actual', ylabel='
                 title='Actual Total Load (First 2 weeks - Original
plt.show()
```



In [12]:
```
# Display the rows with null values

df_energy[df_energy.isnull().any(axis=1)].tail()
```

Out[12]:

time	generation biomass	generation fossil brown coal/lignite	generation fossil gas	generation fossil hard coal	generation fossil oil	generation hydro pumped storage consumption
2016-11-23 03:00:00+00:00	NaN	900.0	4838.0	4547.0	269.0	1413.0
2017-11-14 11:00:00+00:00	0.0	0.0	10064.0	0.0	0.0	0.0
2017-11-14 18:00:00+00:00	0.0	0.0	12336.0	0.0	0.0	0.0
2018-06-11 16:00:00+00:00	331.0	506.0	7538.0	5360.0	300.0	1.0
2018-07-11 07:00:00+00:00	NaN	NaN	NaN	NaN	NaN	NaN

In [13]:
```
# Fill null values using interpolation

df_energy.interpolate(method='linear', limit_direction='forward', i
```

In [14]:
```
# Display the number of non-zero values in each column

print('Non-zero values in each column:\n', df_energy.astype(bool).s
```

```
Non-zero values in each column:

generation biomass                             35060
generation fossil brown coal/lignite           24540
generation fossil gas                          35063
generation fossil hard coal                    35061
generation fossil oil                          35061
generation hydro pumped storage consumption    22450
generation hydro run-of-river and poundage     35061
generation hydro water reservoir               35061
generation nuclear                             35061
generation other                               35060
generation other renewable                     35061
generation solar                               35061
generation waste                               35061
generation wind onshore                        35061
total load actual                              35064
price day ahead                                35064
price actual                                   35064
dtype: int64
```

In [15]:
```
df_weather.head()
```

Out[15]:

	dt_iso	city_name	temp	temp_min	temp_max	pressure	humidity	wind_spee
0	2015-01-01 00:00:00+01:00	Valencia	270.475	270.475	270.475	1001	77	
1	2015-01-01 01:00:00+01:00	Valencia	270.475	270.475	270.475	1001	77	
2	2015-01-01 02:00:00+01:00	Valencia	269.686	269.686	269.686	1002	78	
3	2015-01-01 03:00:00+01:00	Valencia	269.686	269.686	269.686	1002	78	
4	2015-01-01 04:00:00+01:00	Valencia	269.686	269.686	269.686	1002	78	

In [16]:
```
df_weather.describe().round(2)
```

Out[16]:

	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	
count	178396.00	178396.00	178396.00	178396.00	178396.00	178396.00	178396.00	17
mean	289.62	288.33	291.09	1069.26	68.42	2.47	166.59	
std	8.03	7.96	8.61	5969.63	21.90	2.10	116.61	
min	262.24	262.24	262.24	0.00	0.00	0.00	0.00	
25%	283.67	282.48	284.65	1013.00	53.00	1.00	55.00	
50%	289.15	288.15	290.15	1018.00	72.00	2.00	177.00	
75%	295.15	293.73	297.15	1022.00	87.00	4.00	270.00	
max	315.60	315.15	321.15	1008371.00	100.00	133.00	360.00	

In [17]:
```
# Print the type of each variable in df_weather

df_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178396 entries, 0 to 178395
Data columns (total 17 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   dt_iso               178396 non-null  object 
 1   city_name            178396 non-null  object 
 2   temp                 178396 non-null  float64
 3   temp_min             178396 non-null  float64
 4   temp_max             178396 non-null  float64
 5   pressure             178396 non-null  int64  
 6   humidity             178396 non-null  int64  
 7   wind_speed           178396 non-null  int64  
 8   wind_deg             178396 non-null  int64  
 9   rain_1h              178396 non-null  float64
 10  rain_3h              178396 non-null  float64
 11  snow_3h              178396 non-null  float64
 12  clouds_all           178396 non-null  int64  
 13  weather_id           178396 non-null  int64  
 14  weather_main         178396 non-null  object 
 15  weather_description  178396 non-null  object 
 16  weather_icon         178396 non-null  object 
dtypes: float64(6), int64(6), object(5)
memory usage: 23.1+ MB
```

In [18]:
```
def df_convert_dtypes(df, convert_from, convert_to):
    cols = df.select_dtypes(include=[convert_from]).columns
    for col in cols:
        df[col] = df[col].values.astype(convert_to)
    return df
```

In [19]:
```
# Convert columns with int64 type values to float64 type

df_weather = df_convert_dtypes(df_weather, np.int64, np.float64)
```

In [20]:
```
# Convert dt_iso to datetime type, rename it and set it as index

df_weather['time'] = pd.to_datetime(df_weather['dt_iso'], utc=True,
df_weather = df_weather.drop(['dt_iso'], axis=1)
df_weather = df_weather.set_index('time')
```

In [21]:
```
# Display average weather features grouped by each city

mean_weather_by_city = df_weather.groupby('city_name').mean()
mean_weather_by_city
```

Out[21]:

city_name	temp	temp_min	temp_max	pressure	humidity	wind_speed	winc
Barcelona	289.848248	288.594704	291.021987	1284.010486	73.994221	2.786588	187.18
Bilbao	286.378489	284.916661	288.036687	1017.567439	79.089455	1.957470	159.88
Madrid	288.061071	286.824877	289.155600	1011.838448	59.776932	2.441696	173.29
Seville	293.105431	291.184103	295.962431	1018.504711	64.140732	2.483787	151.75
Valencia	290.780780	290.222277	291.355025	1015.973794	65.145113	2.692815	160.75

In [22]:
```
# Find NaNs and duplicates in df_weather

print('There are {} missing values or NaNs in df_weather.'
      .format(df_weather.isnull().values.sum()))

temp_weather = df_weather.duplicated(keep='first').sum()

print('There are {} duplicate rows in df_weather based on all colum
      .format(temp_weather))
```

```
There are 0 missing values or NaNs in df_weather.
There are 8622 duplicate rows in df_weather based on all columns.
```

In [23]:
```
# Display the number of rows in each dataframe

print('There are {} observations in df_energy.'.format(df_energy.sh

cities = df_weather['city_name'].unique()
grouped_weather = df_weather.groupby('city_name')

for city in cities:
    print('There are {} observations in df_weather'
          .format(grouped_weather.get_group('{}'.format(city)).shap
          'about city: {}.'.format(city))
```

```
There are 35064 observations in df_energy.
There are 35145 observations in df_weather about city: Valencia.
There are 36267 observations in df_weather about city: Madrid.
There are 35951 observations in df_weather about city: Bilbao.
There are 35476 observations in df_weather about city:  Barcelona.
There are 35557 observations in df_weather about city: Seville.
```

In [24]:
```
# Create df_weather_2 and drop duplicate rows in df_weather

df_weather_2 = df_weather.reset_index().drop_duplicates(subset=['ti
                                                        keep='last'

df_weather = df_weather.reset_index().drop_duplicates(subset=['time
                                                      keep='first')
```

In [25]:
```
# Display the number of rows in each dataframe again

print('There are {} observations in df_energy.'.format(df_energy.sh

grouped_weather = df_weather.groupby('city_name')

for city in cities:
    print('There are {} observations in df_weather'
          .format(grouped_weather.get_group('{}'.format(city)).shap
          'about city: {}.'.format(city))
```

```
There are 35064 observations in df_energy.
There are 35064 observations in df_weather about city: Valencia.
There are 35064 observations in df_weather about city: Madrid.
There are 35064 observations in df_weather about city: Bilbao.
There are 35064 observations in df_weather about city:  Barcelona.
There are 35064 observations in df_weather about city: Seville.
```

In [26]:
```
# Display all the unique values in the column 'weather_description'

weather_description_unique = df_weather['weather_description'].uniq
weather_description_unique
```

Out[26]:
```
array(['sky is clear', 'few clouds', 'scattered clouds', 'broken c
louds',
       'overcast clouds', 'light rain', 'moderate rain',
       'heavy intensity rain', 'mist', 'heavy intensity shower rai
n',
       'shower rain', 'very heavy rain', 'thunderstorm with heavy
rain',
       'thunderstorm with light rain', 'proximity thunderstorm',
       'thunderstorm', 'light intensity shower rain',
       'light intensity drizzle', 'thunderstorm with rain', 'fog',
       'smoke', 'drizzle', 'heavy intensity drizzle', 'haze',
       'proximity shower rain', 'light snow', 'rain and snow',
       'light rain and snow', 'snow', 'sleet', 'rain and drizzle',
       'light intensity drizzle rain', 'light shower snow',
       'proximity moderate rain', 'ragged shower rain', 'heavy sno
w',
       'sand dust whirls', 'proximity drizzle', 'dust',
       'light thunderstorm', 'squalls'], dtype=object)
```

In [27]:
```
# Display all the unique values in the column 'weather_main'

weather_main_unique = df_weather['weather_main'].unique()
weather_main_unique
```

Out[27]:
```
array(['clear', 'clouds', 'rain', 'mist', 'thunderstorm', 'drizzl
e',
       'fog', 'smoke', 'haze', 'snow', 'dust', 'squall'], dtype=ob
ject)
```

In [28]:
```
# Display all the unique values in the column 'weather_id'

weather_id_unique = df_weather['weather_id'].unique()
weather_id_unique
```

Out[28]:
```
array([800., 801., 802., 803., 804., 500., 501., 502., 701., 522.,
521.,
       503., 202., 200., 211., 520., 300., 201., 741., 711., 301.,
302.,
       721., 600., 616., 615., 601., 611., 311., 310., 620., 531.,
602.,
       731., 761., 210., 771.])
```

In [29]:
```
# Define a function which will calculate R-squared score for the sa

def encode_and_display_r2_score(df_1, df_2, column, categorical=Fal
    dfs = [df_1, df_2]
    if categorical:
        for df in dfs:
            le = LabelEncoder()
            df[column] = le.fit_transform(df[column])
    r2 = r2_score(df_1[column], df_2[column])
    print("R-Squared score of {} is {}".format(column, r2.round(3)))
```

In [30]:
```
# Display the R-squared scores for the columns with qualitative wea

encode_and_display_r2_score(df_weather, df_weather_2, 'weather_desc
encode_and_display_r2_score(df_weather, df_weather_2, 'weather_main
encode_and_display_r2_score(df_weather, df_weather_2, 'weather_id')
```

```
R-Squared score of weather_description is 0.973
R-Squared score of weather_main is 0.963
R-Squared score of weather_id is 0.921
```

In [31]:
```
# Drop columns with qualitative weather information
df_weather = df_weather.drop(['weather_main', 'weather_id',
                              'weather_description', 'weather_icon'
```

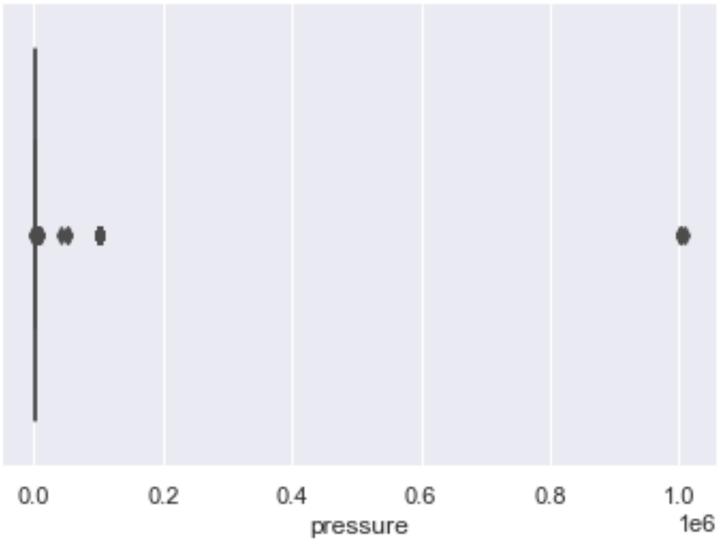
In [32]:
```
# Display the R-squared for all the columns in df_weather and df_we

df_weather_cols = df_weather.columns.drop('city_name')

for col in df_weather_cols:
    encode_and_display_r2_score(df_weather, df_weather_2, col)
```

```
R-Squared score of temp is 1.0
R-Squared score of temp_min is 1.0
R-Squared score of temp_max is 1.0
R-Squared score of pressure is 1.0
R-Squared score of humidity is 1.0
R-Squared score of wind_speed is 1.0
R-Squared score of wind_deg is 1.0
R-Squared score of rain_1h is 1.0
R-Squared score of rain_3h is 1.0
R-Squared score of snow_3h is 1.0
R-Squared score of clouds_all is 1.0
```

In [33]:
```
# Display the number of duplicates in df_weather

temp_weather = df_weather.reset_index().duplicated(subset=['time',
                                                   keep='first').su
print('There are {} duplicate rows in df_weather ' \
      'based on all columns except "time" and "city_name".'.format(
```

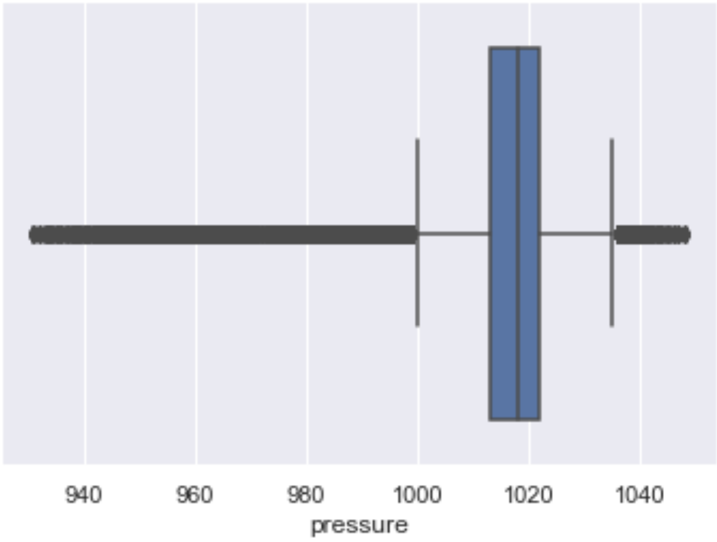
```
There are 0 duplicate rows in df_weather based on all columns exce
pt "time" and "city_name".
```

In [34]:
```
# Check for outliers in 'pressure' column

sns.boxplot(x=df_weather['pressure'])
plt.show()
```

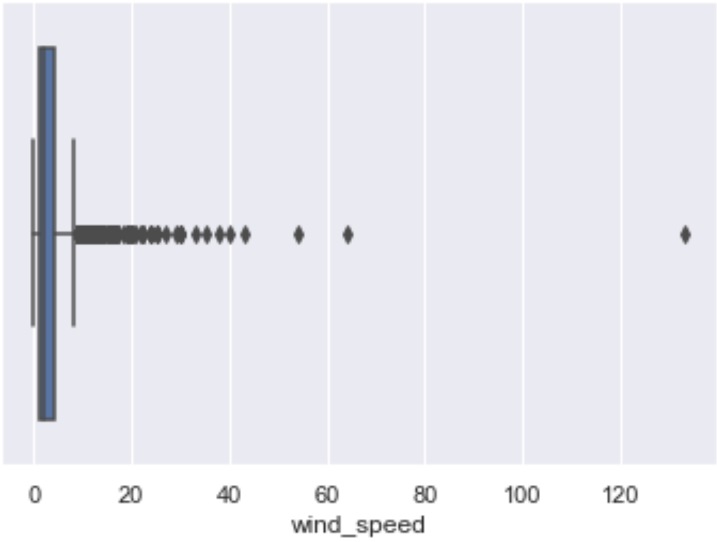


In [35]:
```
# Replace outliers in 'pressure' with NaNs

df_weather.loc[df_weather.pressure > 1051, 'pressure'] = np.nan
df_weather.loc[df_weather.pressure < 931, 'pressure'] = np.nan
```

In [36]:
```
# Check for outliers in 'pressure' column again

sns.boxplot(x=df_weather['pressure'])
plt.show()
```

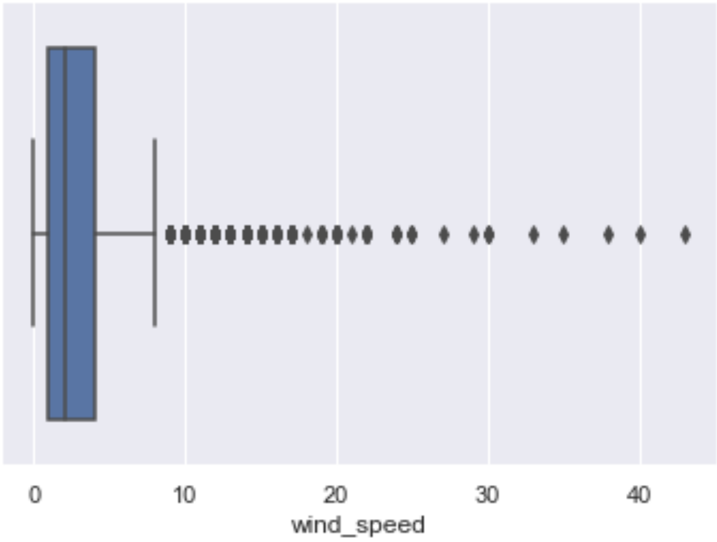


In [37]:
```
# Check for outliers in 'wind_speed' column

sns.boxplot(x=df_weather['wind_speed'])
plt.show()
```



In [38]:
```
# Replace outliers in 'wind_speed' with NaNs

df_weather.loc[df_weather.wind_speed > 50, 'wind_speed'] = np.nan
```

In [39]:
```
# Check for outliers in 'wind_speed' column again

sns.boxplot(x=df_weather['wind_speed'])
plt.show()
```



In [40]:
```
# Fill null values using interpolation

df_weather.interpolate(method='linear', limit_direction='forward',
```

In [41]:
```
# Split the df_weather into 5 dataframes (one for each city)

df_1, df_2, df_3, df_4, df_5 = [x for _, x in df_weather.groupby('c
dfs = [df_1, df_2, df_3, df_4, df_5]
```

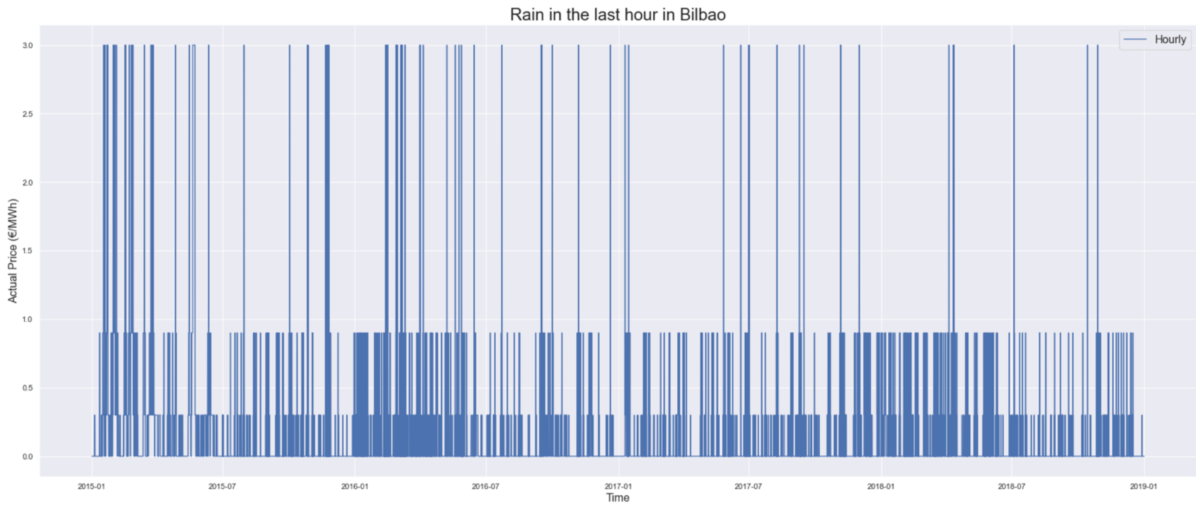
In [42]:
```
# Merge all dataframes into the final dataframe

df_final = df_energy

for df in dfs:
    city = df['city_name'].unique()
    city_str = str(city).replace("'", "").replace('[', '').replace(
    df = df.add_suffix('_{}'.format(city_str))
    df_final = df_final.merge(df, on=['time'], how='outer')
    df_final = df_final.drop('city_name_{}'.format(city_str), axis=

df_final.columns
```

Out[42]:
```
Index(['generation biomass', 'generation fossil brown coal/lignit
e',
       'generation fossil gas', 'generation fossil hard coal',
       'generation fossil oil', 'generation hydro pumped storage c
onsumption',
       'generation hydro run-of-river and poundage',
       'generation hydro water reservoir', 'generation nuclear',
       'generation other', 'generation other renewable', 'generati
on solar',
       'generation waste', 'generation wind onshore', 'total load
actual',
       'price day ahead', 'price actual', 'temp_Barcelona',
       'temp_min_Barcelona', 'temp_max_Barcelona', 'pressure_Barce
lona',
       'humidity_Barcelona', 'wind_speed_Barcelona', 'wind_deg_Bar
celona',
       'rain_1h_Barcelona', 'rain_3h_Barcelona', 'snow_3h_Barcelon
a',
       'clouds_all_Barcelona', 'temp_Bilbao', 'temp_min_Bilbao',
       'temp_max_Bilbao', 'pressure_Bilbao', 'humidity_Bilbao',
       'wind_speed_Bilbao', 'wind_deg_Bilbao', 'rain_1h_Bilbao',
       'rain_3h_Bilbao', 'snow_3h_Bilbao', 'clouds_all_Bilbao', 't
emp_Madrid',
       'temp_min_Madrid', 'temp_max_Madrid', 'pressure_Madrid',
       'humidity_Madrid', 'wind_speed_Madrid', 'wind_deg_Madrid',
       'rain_1h_Madrid', 'rain_3h_Madrid', 'snow_3h_Madrid',
       'clouds_all_Madrid', 'temp_Seville', 'temp_min_Seville',
       'temp_max_Seville', 'pressure_Seville', 'humidity_Seville',
       'wind_speed_Seville', 'wind_deg_Seville', 'rain_1h_Sevill
e',
       'rain_3h_Seville', 'snow_3h_Seville', 'clouds_all_Seville',
       'temp_Valencia', 'temp_min_Valencia', 'temp_max_Valencia',
       'pressure_Valencia', 'humidity_Valencia', 'wind_speed_Valen
cia',
       'wind_deg_Valencia', 'rain_1h_Valencia', 'rain_3h_Valenci
a',
       'snow_3h_Valencia', 'clouds_all_Valencia'],
      dtype='object')
```

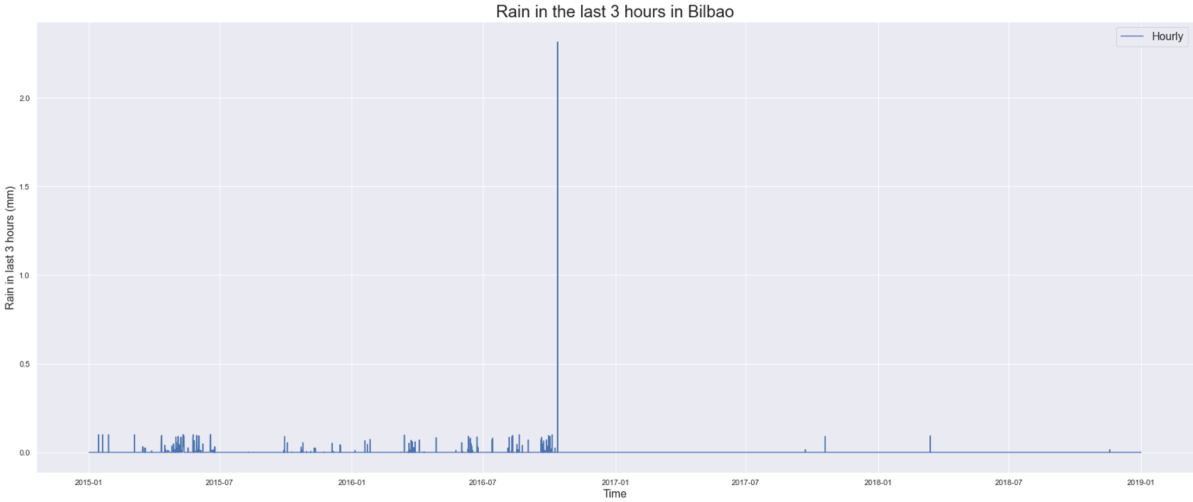
In [43]:
```
# Display the number of NaNs and duplicates in the final dataframe

print('There are {} missing values or NaNs in df_final.'
      .format(df_final.isnull().values.sum()))

temp_final = df_final.duplicated(keep='first').sum()

print('\nThere are {} duplicate rows in df_energy based on all colu
      .format(temp_final))
```

```
There are 0 missing values or NaNs in df_final.

There are 0 duplicate rows in df_energy based on all columns.
```

In [44]:
```
# Plot the 'rain_1h' for Bilbao

ax = plot_series(df_final, 'rain_1h_Bilbao',
                 label='Hourly', ylabel='Actual Price (€/MWh)',
                 title='Rain in the last hour in Bilbao')
plt.show()
```

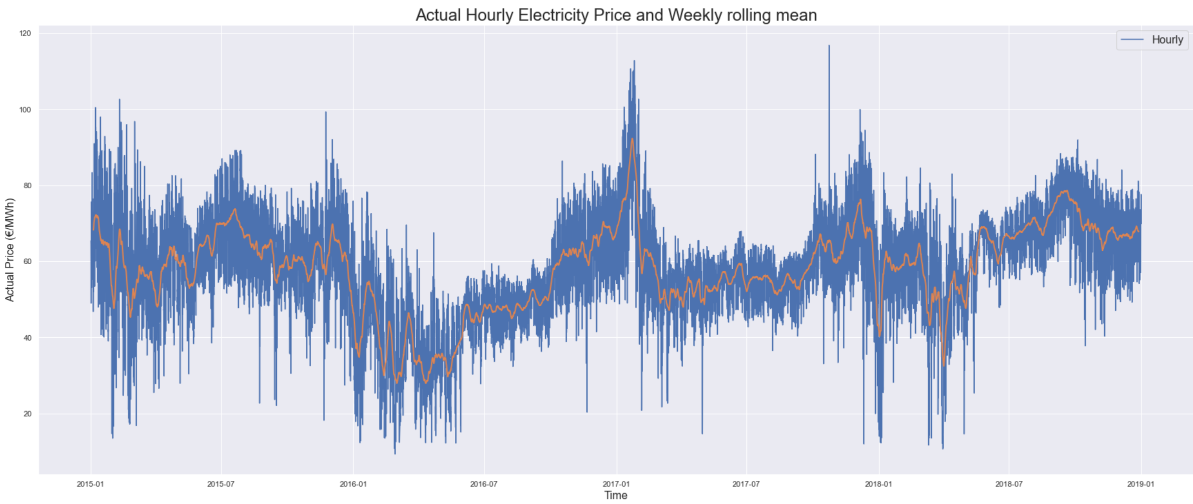


In [45]:
```
# Plot the 'rain_3h' for Bilbao

ax = plot_series(df_final, 'rain_3h_Bilbao',
                 label='Hourly', ylabel='Rain in last 3 hours (mm)'
                 title='Rain in the last 3 hours in Bilbao')
plt.show()
```

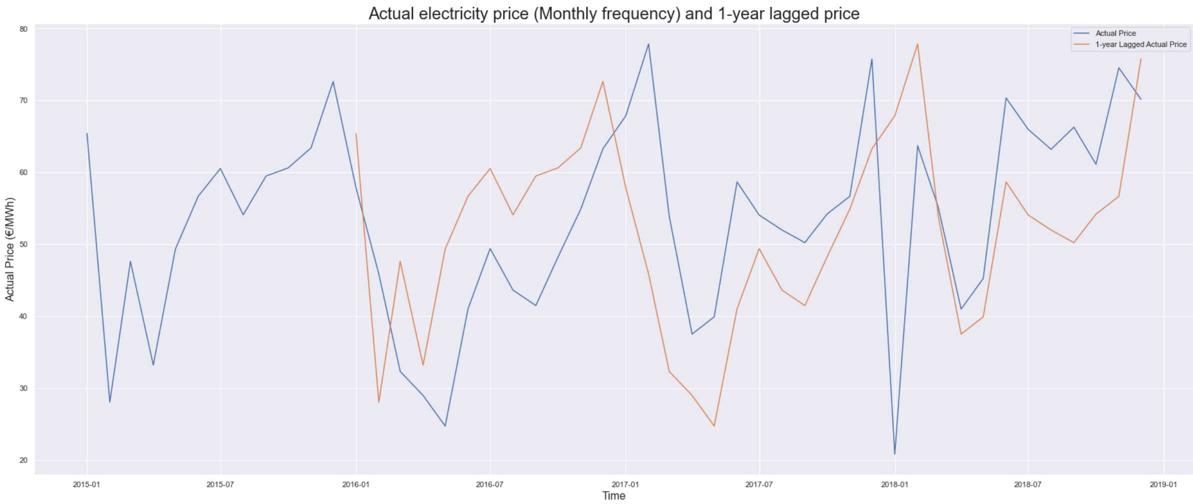


In [46]:
```
cities = ['Barcelona', 'Bilbao', 'Madrid', 'Seville', 'Valencia']

for city in cities:
    df_final = df_final.drop(['rain_3h_{}'.format(city)], axis=1)
```

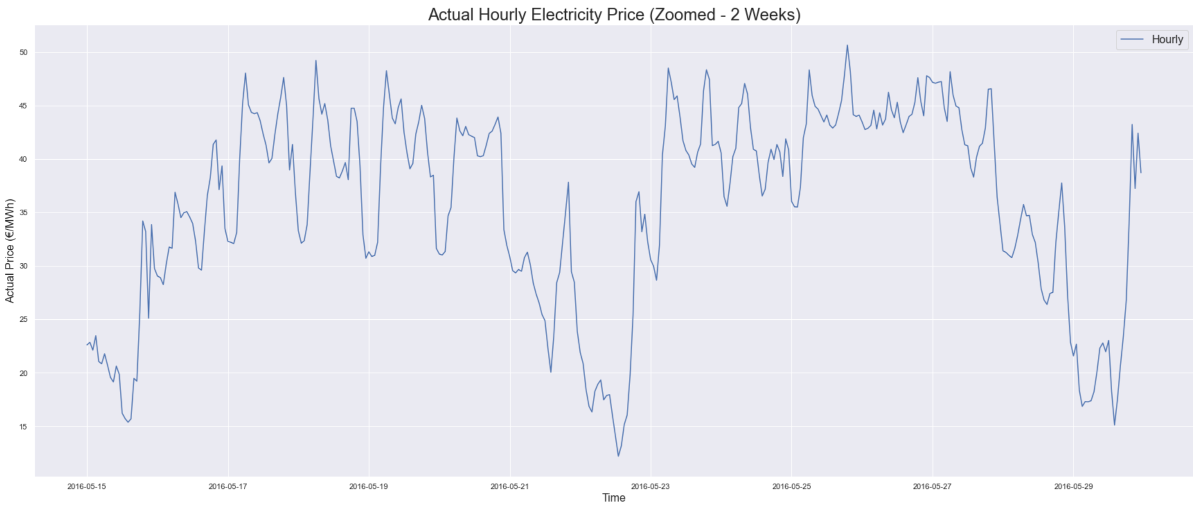
In [47]:
```
# Plot the hourly actual electricity price, along with the weekly r

rolling = df_final['price actual'].rolling(24*7, center=True).mean(
ax = plot_series(df_final, 'price actual', label='Hourly', ylabel='
                 title='Actual Hourly Electricity Price and Weekly
ax.plot(rolling, linestyle='-', linewidth=2, label='Weekly rolling 
plt.show()
```



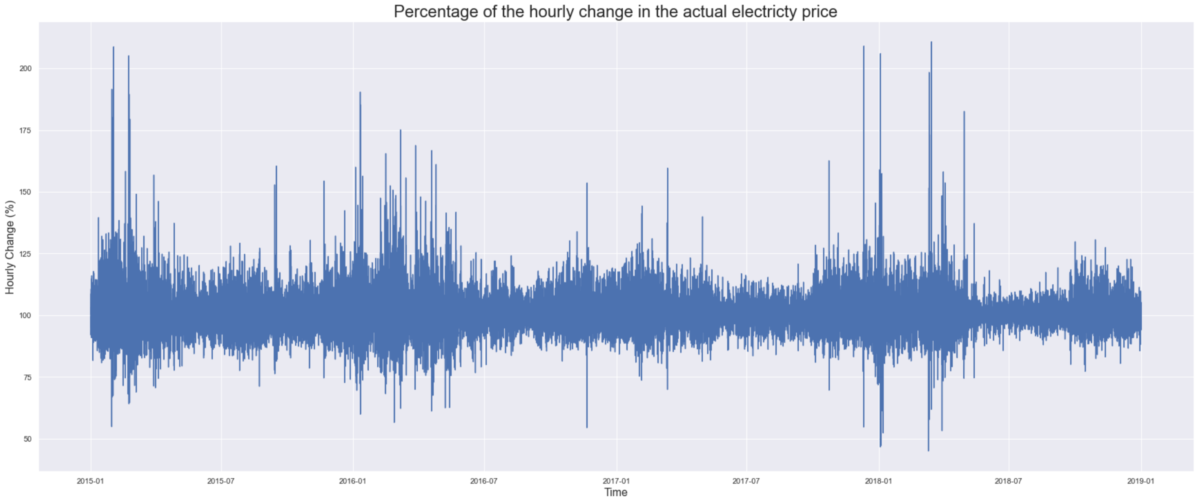
In [48]:
```
# Plot the electricity price (monthly frequence) along with its 1-ye

monthly_price = df_final['price actual'].asfreq('M')
ax = plot_series(series=monthly_price, ylabel='Actual Price (€/MWh)
                 title='Actual electricity price (Monthly frequency
shifted = df_final['price actual'].asfreq('M').shift(12)
ax.plot(shifted, label='Hourly')
ax.legend(['Actual Price', '1-year Lagged Actual Price'])
plt.show()
```



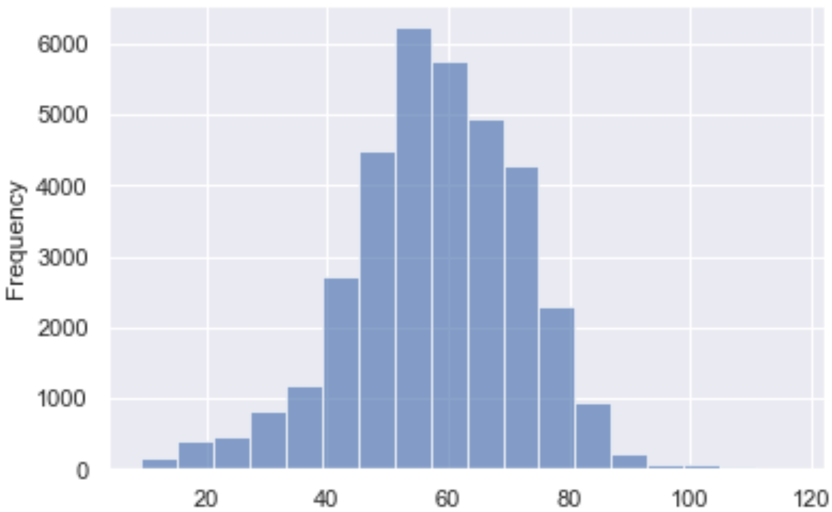
In [49]:
```
# Plot the actual electricity price at a daily/weekly scale

ax = plot_series(df_final, 'price actual', label='Hourly', ylabel='A
                 start=1 + 24 * 500, end=1 + 24 * 515,
                 title='Actual Hourly Electricity Price (Zoomed - 2
plt.show()
```



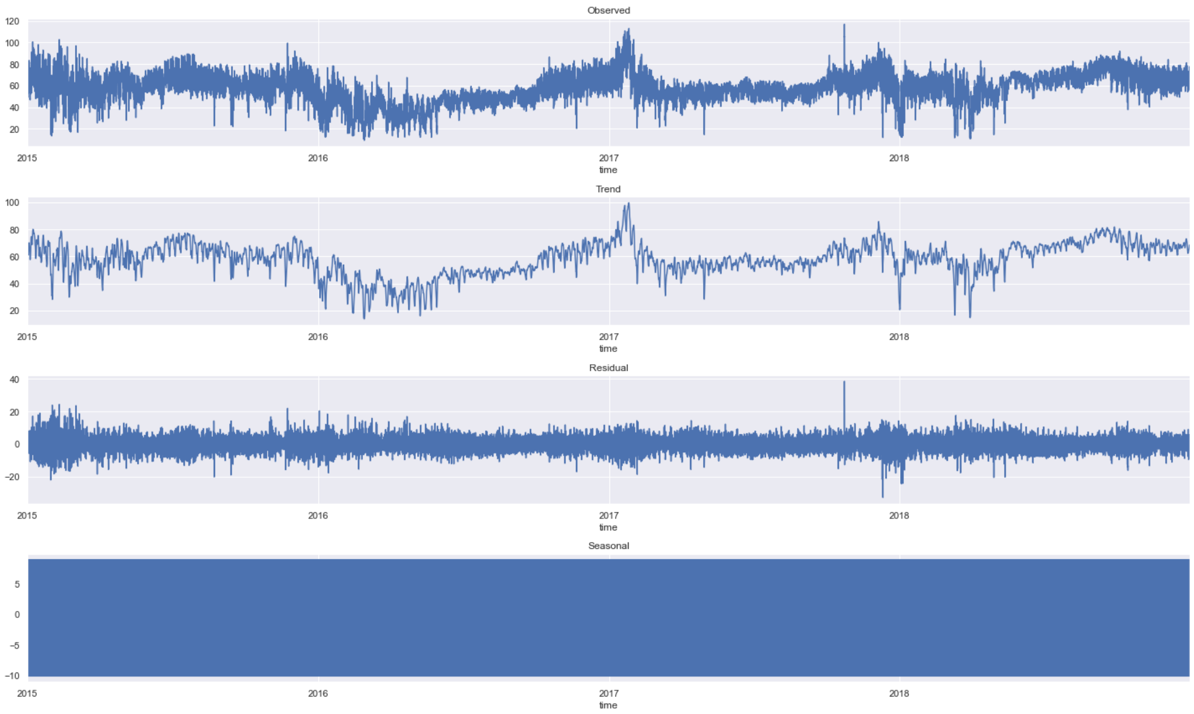
In [50]:
```
# Plot the percentage of the hourly change in the actual electricit

change = df_energy['price actual'].div(df_energy['price actual'].sh
ax = plot_series(series=change, ylabel='Hourly Change (%)',
                 title='Percentage of the hourly change in the actu
plt.show()
```



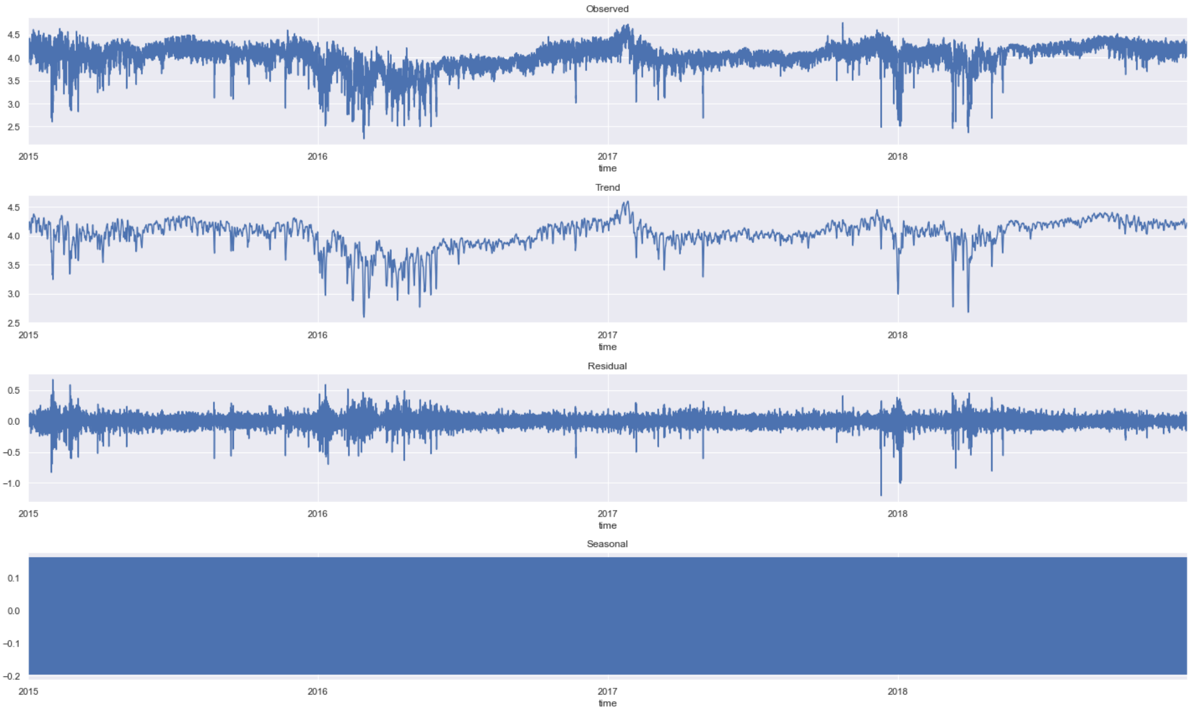
In [51]:
```
# Plot the histogram of the actual electricity price

ax = df_energy['price actual'].plot.hist(bins=18, alpha=0.65)
```



In [52]:
```
# Decompose the electricity price time series

res = sm.tsa.seasonal_decompose(df_energy['price actual'], model='a

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(20, 12))
res.observed.plot(ax=ax1, title='Observed')
res.trend.plot(ax=ax2, title='Trend')
res.resid.plot(ax=ax3, title='Residual')
res.seasonal.plot(ax=ax4, title='Seasonal')
plt.tight_layout()
plt.show()
```



In [53]:
```
# Decompose the log electricity price time-series

res = sm.tsa.seasonal_decompose(np.log(df_energy['price actual']), 

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(20, 12))
res.observed.plot(ax=ax1, title='Observed')
res.trend.plot(ax=ax2, title='Trend')
res.resid.plot(ax=ax3, title='Residual')
res.seasonal.plot(ax=ax4, title='Seasonal')
plt.tight_layout()
plt.show()
```

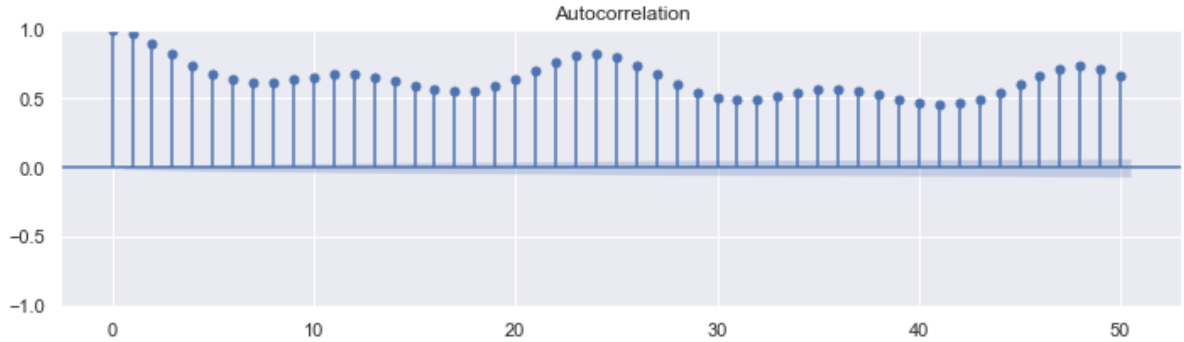


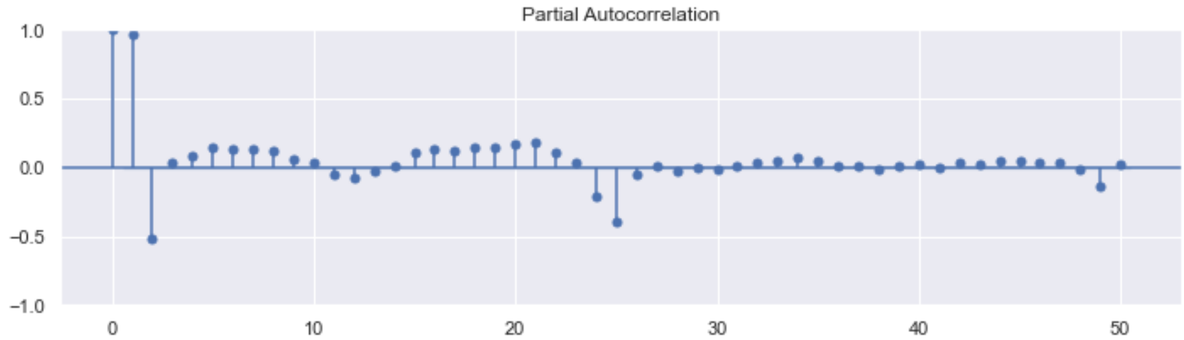
In [54]:
```
y = df_final['price actual']
adf_test = adfuller(y, regression='c')
print('ADF Statistic: {:.6f}\np-value: {:.6f}\n#Lags used: {}'
      .format(adf_test[0], adf_test[1], adf_test[2]))
for key, value in adf_test[4].items():
    print('Critical Value ({}): {:.6f}'.format(key, value))
```

```
ADF Statistic: -9.147016
p-value: 0.000000
#Lags used: 50
Critical Value (1%): -3.430537
Critical Value (5%): -2.861623
Critical Value (10%): -2.566814
```

In [55]:
```
# Plot autocorrelation and partial autocorrelation plots

fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(10, 6))
plot_acf(df_final['price actual'], lags=50, ax=ax1)
plot_pacf(df_final['price actual'], lags=50, ax=ax2)
plt.tight_layout()
plt.show()
```

```
/Users/anaconda3/lib/python3.7/site-packages/statsmodels/graphics/
tsaplots.py:353: FutureWarning: The default method 'yw' can produc
e PACF values outside of the [-1,1] interval. After 0.13, the defa
ult will change tounadjusted Yule-Walker ('ywm'). You can use this
method now by setting method='ywm'.
  FutureWarning,
```





In [56]:
```
# Find the correlations between the electricity price and the rest 

correlations = df_final.corr(method='pearson')
print(correlations['price actual'].sort_values(ascending=False).to_
```

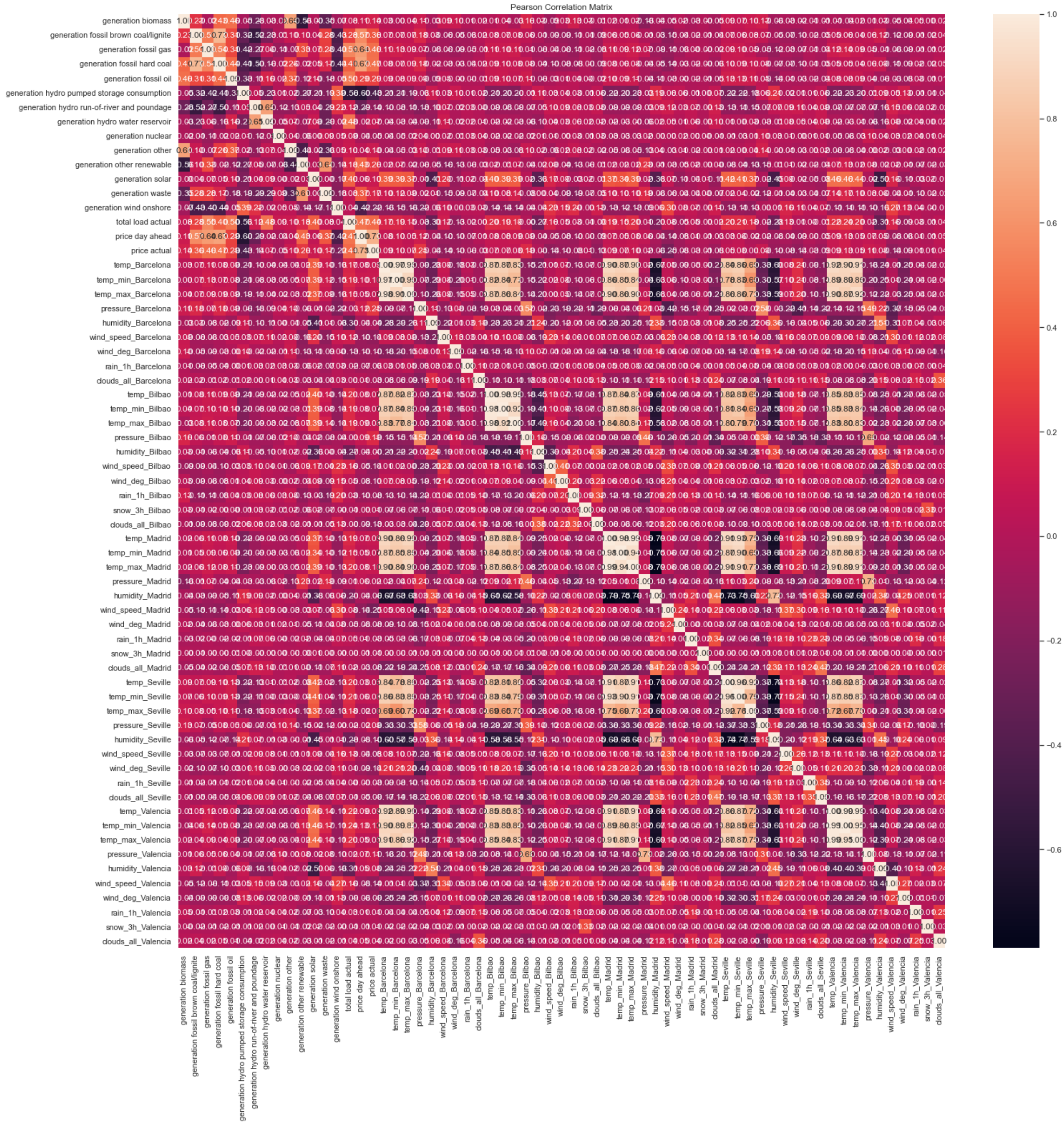
```
price actual                                1.000000
price day ahead                             0.732155
generation fossil hard coal                 0.465637
generation fossil gas                       0.461452
total load actual                           0.435253
generation fossil brown coal/lignite        0.363993
generation fossil oil                       0.285050
generation other renewable                  0.255551
pressure_Barcelona                          0.249177
pressure_Bilbao                             0.194063
generation waste                            0.168710
generation biomass                          0.142671
```

```
temp_min_Valencia                              0.133141
pressure_Valencia                              0.109812
temp_min_Barcelona                             0.103726
generation other                               0.099914
generation solar                               0.098529
temp_max_Madrid                                0.096279
temp_Valencia                                  0.090505
pressure_Seville                               0.090162
temp_Madrid                                    0.087995
temp_Barcelona                                 0.085857
humidity_Valencia                              0.078819
temp_min_Seville                               0.077296
temp_max_Bilbao                                0.076766
temp_min_Bilbao                                0.074776
temp_Bilbao                                    0.073018
generation hydro water reservoir               0.071910
temp_max_Barcelona                             0.068936
temp_min_Madrid                                0.066777
temp_Seville                                   0.050274
temp_max_Valencia                              0.047478
clouds_all_Valencia                            0.040055
pressure_Madrid                                0.018756
snow_3h_Bilbao                                 0.014920
rain_1h_Valencia                               0.012049
snow_3h_Valencia                               0.007461
temp_max_Seville                               0.003253
humidity_Bilbao                               -0.000450
snow_3h_Madrid                                -0.008427
rain_1h_Madrid                                -0.027137
clouds_all_Barcelona                          -0.027599
rain_1h_Seville                               -0.034887
humidity_Barcelona                            -0.037682
generation nuclear                            -0.053016
rain_1h_Barcelona                             -0.055130
humidity_Madrid                               -0.064668
wind_speed_Seville                            -0.078469
rain_1h_Bilbao                                -0.078806
clouds_all_Madrid                             -0.079415
wind_deg_Madrid                               -0.082756
clouds_all_Seville                            -0.086233
wind_deg_Valencia                             -0.092710
wind_deg_Barcelona                            -0.096248
humidity_Seville                              -0.103004
wind_deg_Bilbao                               -0.103097
clouds_all_Bilbao                             -0.132669
generation hydro run-of-river and poundage    -0.136659
wind_deg_Seville                              -0.137099
wind_speed_Barcelona                          -0.138658
wind_speed_Valencia                           -0.142360
wind_speed_Bilbao                             -0.143327
generation wind onshore                       -0.220497
wind_speed_Madrid                             -0.245861
generation hydro pumped storage consumption   -0.426196
snow_3h_Barcelona                                   NaN
```

```
snow_3h_Seville                                         NaN
```

In [57]:
```
df_final = df_final.drop(['snow_3h_Barcelona', 'snow_3h_Seville'], a
```

In [58]:
```
# Plot Pearson correlation matrix

correlations = df_final.corr(method='pearson')
fig = plt.figure(figsize=(24, 24))
sns.heatmap(correlations, annot=True, fmt='.2f')
plt.title('Pearson Correlation Matrix')
plt.show()
```



In [59]:
```
highly_correlated = abs(correlations[correlations > 0.75])
print(highly_correlated[highly_correlated < 1.0].stack().to_string(
```

```
generation fossil brown coal/lignite  generation fossil hard coal
0.768831
generation fossil hard coal           generation fossil brown coa
l/lignite    0.768831
temp_Barcelona                        temp_min_Barcelona
0.970264
                                      temp_max_Barcelona
0.976904
                                      temp_Bilbao
0.866727
                                      temp_min_Bilbao
0.867970
                                      temp_max_Bilbao
0.828347
                                      temp_Madrid
0.903996
                                      temp_min_Madrid
0.874548
                                      temp_max_Madrid
```

feature enginering

In [60]:
```
# Generate 'hour', 'weekday' and 'month' features

for i in range(len(df_final)):
    position = df_final.index[i]
    hour = position.hour
    weekday = position.weekday()
    month = position.month
    df_final.loc[position, 'hour'] = hour
    df_final.loc[position, 'weekday'] = weekday
    df_final.loc[position, 'month'] = month
```

In [61]:
```
# Generate 'business hour' feature

for i in range(len(df_final)):
    position = df_final.index[i]
    hour = position.hour
    if ((hour > 8 and hour < 14) or (hour > 16 and hour < 21)):
        df_final.loc[position, 'business hour'] = 2
    elif (hour >= 14 and hour <= 16):
        df_final.loc[position, 'business hour'] = 1
    else:
        df_final.loc[position, 'business hour'] = 0
```

In [62]:
```
# Generate 'weekend' feature

for i in range(len(df_final)):
    position = df_final.index[i]
    weekday = position.weekday()
    if (weekday == 6):
        df_final.loc[position, 'weekday'] = 2
    elif (weekday == 5):
        df_final.loc[position, 'weekday'] = 1
    else:
        df_final.loc[position, 'weekday'] = 0
```

In [63]:
```
# Generate 'temp_range' for each city

cities = ['Barcelona', 'Bilbao', 'Madrid', 'Seville', 'Valencia']

for i in range(len(df_final)):
    position = df_final.index[i]
    for city in cities:
        temp_max = df_final.loc[position, 'temp_max_{}'.format(city
        temp_min = df_final.loc[position, 'temp_min_{}'.format(city
        df_final.loc[position, 'temp_range_{}'.format(city)] = abs(
```

In [64]:
```
# Calculate the weight of every city

# THE POPULATION OF PEOPLE AND THEIR CITIES 6155116 + 5179243 + 164

total_pop = 6155116 + 5179243 + 1645342 + 1305342 + 987000

weight_Madrid = 6155116 / total_pop
weight_Barcelona = 5179243 / total_pop
weight_Valencia = 1645342 / total_pop
weight_Seville = 1305342 / total_pop
weight_Bilbao = 987000 / total_pop
```

In [65]:
```
cities_weights = {'Madrid': weight_Madrid,
                  'Barcelona': weight_Barcelona,
                  'Valencia': weight_Valencia,
                  'Seville': weight_Seville,
                  'Bilbao': weight_Bilbao}
```

In [66]:
```
# Generate 'temp_weighted' feature

for i in range(len(df_final)):
    position = df_final.index[i]
    temp_weighted = 0
    for city in cities:
        temp = df_final.loc[position, 'temp_{}'.format(city)]
        temp_weighted += temp * cities_weights.get('{}'.format(city
    df_final.loc[position, 'temp_weighted'] = temp_weighted
```

In [67]:
```
df_final['generation coal all'] = df_final['generation fossil hard 
```

In [68]:
```
def multivariate_data(dataset, target, start_index, end_index, hist
                      target_size, step, single_step=False):
    data = []
    labels = []

    start_index = start_index + history_size
    if end_index is None:
        end_index = len(dataset) - target_size

    for i in range(start_index, end_index):
        indices = range(i-history_size, i, step)
        data.append(dataset[indices])

        if single_step:
            labels.append(target[i + target_size])
        else:
            labels.append(target[i : i + target_size])

    return np.array(data), np.array(labels)
```

In [69]:
```
train_end_idx = 27048
cv_end_idx = 31056
test_end_idx = 35064
```

In [70]:
```
X = df_final[df_final.columns.drop('price actual')].values
y = df_final['price actual'].values

y = y.reshape(-1, 1)
```

In [71]:
```
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))
```

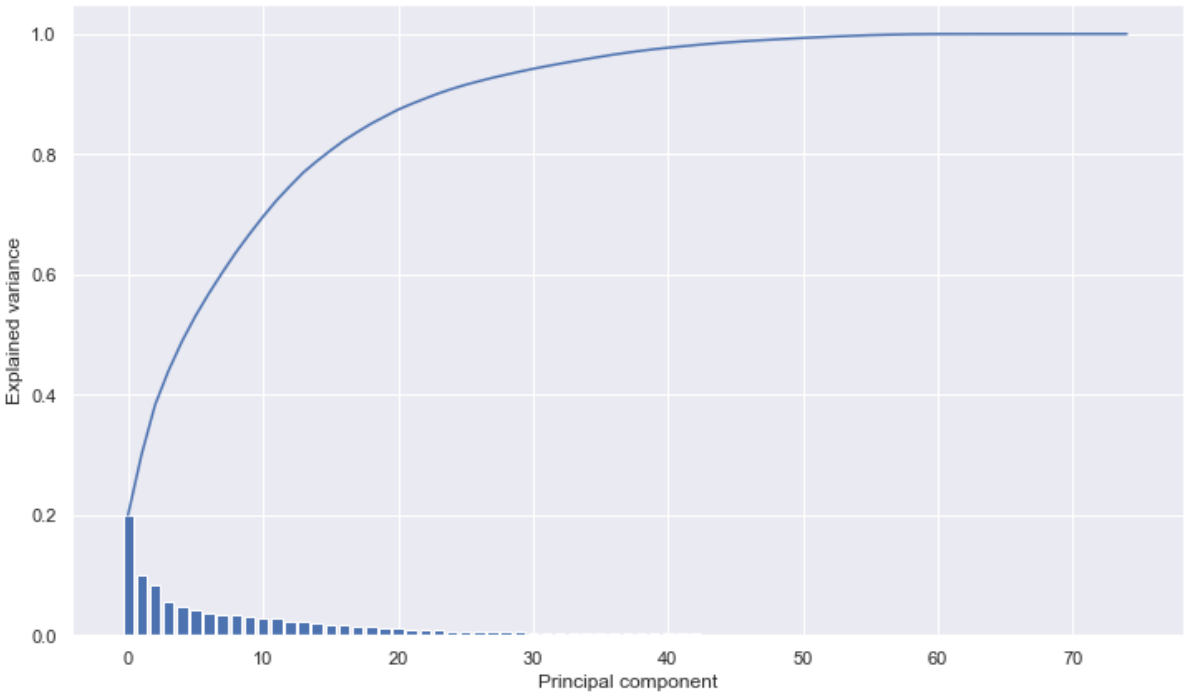
In [72]:
```
scaler_X.fit(X[:train_end_idx])
scaler_y.fit(y[:train_end_idx])
```

Out[72]: MinMaxScaler()

In [73]:
```
X_norm = scaler_X.transform(X)
y_norm = scaler_y.transform(y)
```

In [74]:
```
pca = PCA()
X_pca = pca.fit(X_norm[:train_end_idx])
```

In [75]:
```
num_components = len(pca.explained_variance_ratio_)
plt.figure(figsize=(12, 7))
plt.bar(np.arange(num_components), pca.explained_variance_ratio_)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Principal component')
plt.ylabel('Explained variance')
plt.show()
```



In [76]:
```
pca = PCA(n_components=0.80)
pca.fit(X_norm[:train_end_idx])
X_pca = pca.transform(X_norm)
```

In [77]:
```
X_pca.shape
```

Out[77]:
```
(35064, 16)
```

In [78]:
```
dataset_norm = np.concatenate((X_pca, y_norm), axis=1)

past_history = 24
future_target = 0
```

In [79]:
```
X_train, y_train = multivariate_data(dataset_norm, dataset_norm[:,
                                    0, train_end_idx, past_history
                                    future_target, step=1, single_
```

In [80]:
```
X_val, y_val = multivariate_data(dataset_norm, dataset_norm[:, -1],
                                train_end_idx, cv_end_idx, past_hi
                                future_target, step=1, single_step
```

In [81]:
```
X_test, y_test = multivariate_data(dataset_norm, dataset_norm[:, -1
                                   cv_end_idx, test_end_idx, past_h
                                   future_target, step=1, single_st
```

In [82]:
```
batch_size = 32
buffer_size = 1000
```

In [83]:
```
train = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train = train.cache().shuffle(buffer_size).batch(batch_size).prefet

validation = tf.data.Dataset.from_tensor_slices((X_val, y_val))
validation = validation.batch(batch_size).prefetch(1)
```

In [84]:
```
# Define some common parameters

input_shape = X_train.shape[-2:]
loss = tf.keras.losses.MeanSquaredError()
metric = [tf.keras.metrics.RootMeanSquaredError()]
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
              lambda epoch: 1e-4 * 10**(epoch / 10))
early_stopping = tf.keras.callbacks.EarlyStopping(patience=10)
```

In [85]:
```
y_test = y_test.reshape(-1, 1)
y_test_inv = scaler_y.inverse_transform(y_test)
```

ELECTRICITY PRICE FORECASTING

In [86]:
```
def plot_model_rmse_and_loss(history):

    # Evaluate train and validation accuracies and losses

    train_rmse = history.history['root_mean_squared_error']
    val_rmse = history.history['val_root_mean_squared_error']

    train_loss = history.history['loss']
    val_loss = history.history['val_loss']

    # Visualize epochs vs. train and validation accuracies and loss
    plt.style.use('fivethirtyeight')
    plt.figure(figsize=(30, 15))
    plt.subplot(1, 2, 1)
    plt.plot(train_rmse, label='Training RMSE')
    plt.plot(val_rmse, label='Validation RMSE')
    plt.legend()
    plt.title('Epochs vs. Training and Validation RMSE')

    plt.subplot(1, 2, 2)
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.legend()
    plt.title('Epochs vs. Training and Validation Loss')

    plt.show()
    plt.savefig('LSTM-CNN')
```

XGBOOST

In [87]:
```
X_train_xgb = X_train.reshape(-1, X_train.shape[1] * X_train.shape[
X_val_xgb = X_val.reshape(-1, X_val.shape[1] * X_val.shape[2])
X_test_xgb = X_test.reshape(-1, X_test.shape[1] * X_test.shape[2])
```

In [88]:
```
param = {'eta': 0.03, 'max_depth': 180,
         'subsample': 1.0, 'colsample_bytree': 0.95,
         'alpha': 0.1, 'lambda': 0.15, 'gamma': 0.1,
         'objective': 'reg:linear', 'eval_metric': 'rmse',
         'silent': 1, 'min_child_weight': 0.1, 'n_jobs': -1}

dtrain = xgb.DMatrix(X_train_xgb, y_train)
dval = xgb.DMatrix(X_val_xgb, y_val)
dtest = xgb.DMatrix(X_test_xgb, y_test)
eval_list = [(dtrain, 'train'), (dval, 'eval')]

xgb_model = xgb.train(param, dtrain, 180, eval_list, early_stopping
```

```
/Users/anaconda3/lib/python3.7/site-packages/xgboost/core.py:571:
FutureWarning: Pass `evals` as keyword args.  Passing these as pos
itional arguments will be considered as error in future releases.
  format(", ".join(args_msg)), FutureWarning

[22:40:47] WARNING: /Users/runner/work/xgboost/xgboost/python-pack
age/build/temp.macosx-10.9-x86_64-cpython-37/xgboost/src/objectiv
e/regression_obj.cu:203: reg:linear is now deprecated in favor of
reg:squarederror.
[22:40:47] WARNING: /Users/runner/work/xgboost/xgboost/python-pack
age/build/temp.macosx-10.9-x86_64-cpython-37/xgboost/src/learner.c
c:627:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used b
y language bindings but
  then being mistakenly passed down to XGBoost core, or some param
eter actually being used
  but getting flagged wrongly here. Please open an issue if you fi
nd any such cases
```

In [89]:
```
forecast = xgb_model.predict(dtest)
xgb_forecast = forecast.reshape(-1, 1)

xgb_forecast_inv = scaler_y.inverse_transform(xgb_forecast)

rmse_xgb = sqrt(mean_squared_error(y_test_inv, xgb_forecast_inv))
print('RMSE of hour-ahead electricity price XGBoost forecast: {}'
      .format(round(rmse_xgb, 3)))

# Printing the RMSE score
print('Model Score (RMSE):', round(rmse_xgb, 3))
```

```
RMSE of hour-ahead electricity price XGBoost forecast: 2.216
Model Score (RMSE): 2.216
```

In [90]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squa

# Assuming you have predictions (xgb_forecast_inv) and actual value
# Calculate R-squared
r_squared = r2_score(y_test_inv, xgb_forecast_inv)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test_inv, xgb_forecast_inv)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_inv, xgb_forecast_inv)

# Print evaluation metrics
print('R-squared (R²): {:.3f}'.format(r_squared))
print('Mean Absolute Error (MAE): {:.3f}'.format(mae))
print('Mean Squared Error (MSE): {:.3f}'.format(mse))
```

```
R-squared (R²): 0.928
Mean Absolute Error (MAE): 1.653
Mean Squared Error (MSE): 4.911
```

LSTM

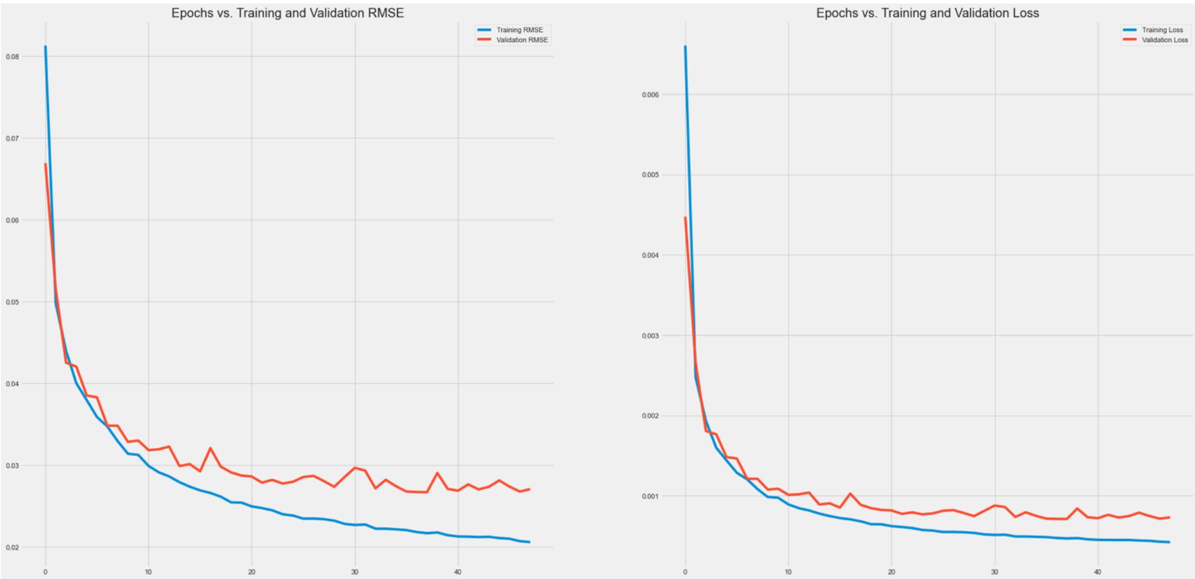
In [91]:
```
tf.keras.backend.clear_session()

multivariate_lstm = tf.keras.models.Sequential([
    LSTM(100, input_shape=input_shape,
         return_sequences=True),
    Flatten(),
    Dense(200, activation='relu'),
    Dropout(0.1),
    Dense(1)
])

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
                   'multivariate_lstm.h5', monitor=('val_loss'), sa
optimizer = tf.keras.optimizers.Adam(lr=6e-3, amsgrad=True)

multivariate_lstm.compile(loss=loss,
                          optimizer=optimizer,
                          metrics=metric)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instea
d, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ad
am.
```

In [92]:
```
history = multivariate_lstm.fit(train, epochs=120,
                                validation_data=validation,
                                callbacks=[early_stopping,
                                           model_checkpoint])
```

```
Epoch 1/120
845/845 [==============================] - 17s 18ms/step - loss:
0.0066 - root_mean_squared_error: 0.0813 - val_loss: 0.0045 - val_
root_mean_squared_error: 0.0669
Epoch 2/120
845/845 [==============================] - 15s 17ms/step - loss:
0.0025 - root_mean_squared_error: 0.0498 - val_loss: 0.0027 - val_
root_mean_squared_error: 0.0515
Epoch 3/120
845/845 [==============================] - 15s 18ms/step - loss:
0.0019 - root_mean_squared_error: 0.0440 - val_loss: 0.0018 - val_
root_mean_squared_error: 0.0425
Epoch 4/120
845/845 [==============================] - 16s 18ms/step - loss:
0.0016 - root_mean_squared_error: 0.0400 - val_loss: 0.0018 - val_
root_mean_squared_error: 0.0420
Epoch 5/120
845/845 [==============================] - 15s 18ms/step - loss:
0.0014 - root_mean_squared_error: 0.0379 - val_loss: 0.0015 - val_
root_mean_squared_error: 0.0385
```

In [93]:
```
plot_model_rmse_and_loss(history)
```



<Figure size 432x288 with 0 Axes>

In [94]:
```
multivariate_lstm = tf.keras.models.load_model('multivariate_lstm.h5

forecast = multivariate_lstm.predict(X_test)
lstm_forecast = scaler_y.inverse_transform(forecast)

rmse_lstm = sqrt(mean_squared_error(y_test_inv,
                                    lstm_forecast))
print('RMSE of hour-ahead electricity price LSTM forecast: {}'
      .format(round(rmse_lstm, 3)))
# Printing the RMSE score
print('Model Score (RMSE):', round(rmse_lstm, 3))
```

```
125/125 [==============================] - 1s 6ms/step
RMSE of hour-ahead electricity price LSTM forecast: 2.464
Model Score (RMSE): 2.464
```

In [95]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squa

# Assuming you have predictions (xgb_forecast_inv) and actual value
# Calculate R-squared
r_squared = r2_score(y_test_inv, lstm_forecast)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test_inv, lstm_forecast)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_inv, lstm_forecast)

# Print evaluation metrics
print('R-squared (R²): {:.3f}'.format(r_squared))
print('Mean Absolute Error (MAE): {:.3f}'.format(mae))
print('Mean Squared Error (MSE): {:.3f}'.format(mse))
```

```
R-squared (R²): 0.911
Mean Absolute Error (MAE): 1.926
Mean Squared Error (MSE): 6.073
```

CNN

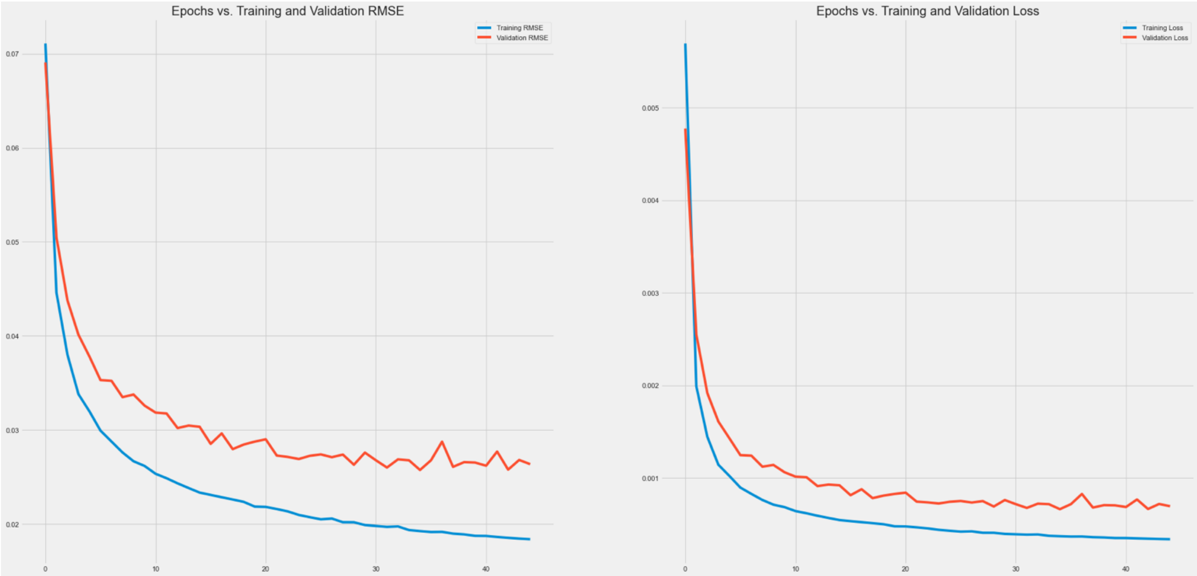
In [96]:
```
tf.keras.backend.clear_session()

multivariate_cnn = tf.keras.models.Sequential([
    Conv1D(filters=48, kernel_size=2,
           strides=1, padding='causal',
           activation='relu',
           input_shape=input_shape),
    Flatten(),
    Dense(48, activation='relu'),
    Dense(1)
])

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
                   'multivariate_cnn.h5', save_best_only=True)
optimizer = tf.keras.optimizers.Adam(lr=6e-3, amsgrad=True)

multivariate_cnn.compile(loss=loss,
                         optimizer=optimizer,
                         metrics=metric)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instea
d, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ad
am.
```

In [97]:
```
history = multivariate_cnn.fit(train, epochs=120,
                               validation_data=validation,
                               callbacks=[early_stopping,
                                          model_checkpoint])
```

```
Epoch 1/120
845/845 [==============================] - 3s 2ms/step - loss: 0.0
057 - root_mean_squared_error: 0.0711 - val_loss: 0.0048 - val_roo
t_mean_squared_error: 0.0691
Epoch 2/120
845/845 [==============================] - 2s 2ms/step - loss: 0.0
020 - root_mean_squared_error: 0.0446 - val_loss: 0.0025 - val_roo
t_mean_squared_error: 0.0505
Epoch 3/120
845/845 [==============================] - 2s 2ms/step - loss: 0.0
014 - root_mean_squared_error: 0.0380 - val_loss: 0.0019 - val_roo
t_mean_squared_error: 0.0438
Epoch 4/120
845/845 [==============================] - 2s 2ms/step - loss: 0.0
011 - root_mean_squared_error: 0.0338 - val_loss: 0.0016 - val_roo
t_mean_squared_error: 0.0401
Epoch 5/120
845/845 [==============================] - 2s 2ms/step - loss: 0.0
010 - root_mean_squared_error: 0.0320 - val_loss: 0.0014 - val_roo
t_mean_squared_error: 0.0378
```

In [98]:
```
plot_model_rmse_and_loss(history)
```




```
<Figure size 432x288 with 0 Axes>
```

In [99]:
```
multivariate_cnn = tf.keras.models.load_model('multivariate_cnn.h5'

forecast = multivariate_cnn.predict(X_test)
multivariate_cnn_forecast = scaler_y.inverse_transform(forecast)

rmse_mult_cnn = sqrt(mean_squared_error(y_test_inv,
                                        multivariate_cnn_forecast))
print('RMSE of hour-ahead electricity price multivariate CNN foreca
      .format(round(rmse_mult_cnn, 3)))
```

```
125/125 [==============================] - 0s 1ms/step
RMSE of hour-ahead electricity price multivariate CNN forecast: 2.
358
```

In [100]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squa

# Assuming you have predictions (xgb_forecast_inv) and actual value
# Calculate R-squared
r_squared = r2_score(y_test_inv, multivariate_cnn_forecast)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test_inv, multivariate_cnn_forecast)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_inv, multivariate_cnn_forecast)

# Print evaluation metrics
print('R-squared (R²): {:.3f}'.format(r_squared))
print('Mean Absolute Error (MAE): {:.3f}'.format(mae))
print('Mean Squared Error (MSE): {:.3f}'.format(mse))
```

```
R-squared (R²): 0.919
Mean Absolute Error (MAE): 1.797
Mean Squared Error (MSE): 5.560
```

time distributed MLP

In [101]:
```
tf.keras.backend.clear_session()

multivariate_mlp = tf.keras.models.Sequential([
    TimeDistributed(Dense(200, activation='relu'),
                    input_shape=input_shape),
    TimeDistributed(Dense(150, activation='relu')),
    TimeDistributed(Dense(100, activation='relu')),
    TimeDistributed(Dense(50, activation='relu')),
    Flatten(),
    Dense(150, activation='relu'),
    Dropout(0.1),
    Dense(1)
])

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
                    'multivariate_mlp.h5', save_best_only=True)
optimizer = tf.keras.optimizers.Adam(lr=2e-3, amsgrad=True)

multivariate_mlp.compile(loss=loss,
                          optimizer=optimizer,
                          metrics=metric)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instea
d, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ad
am.
```

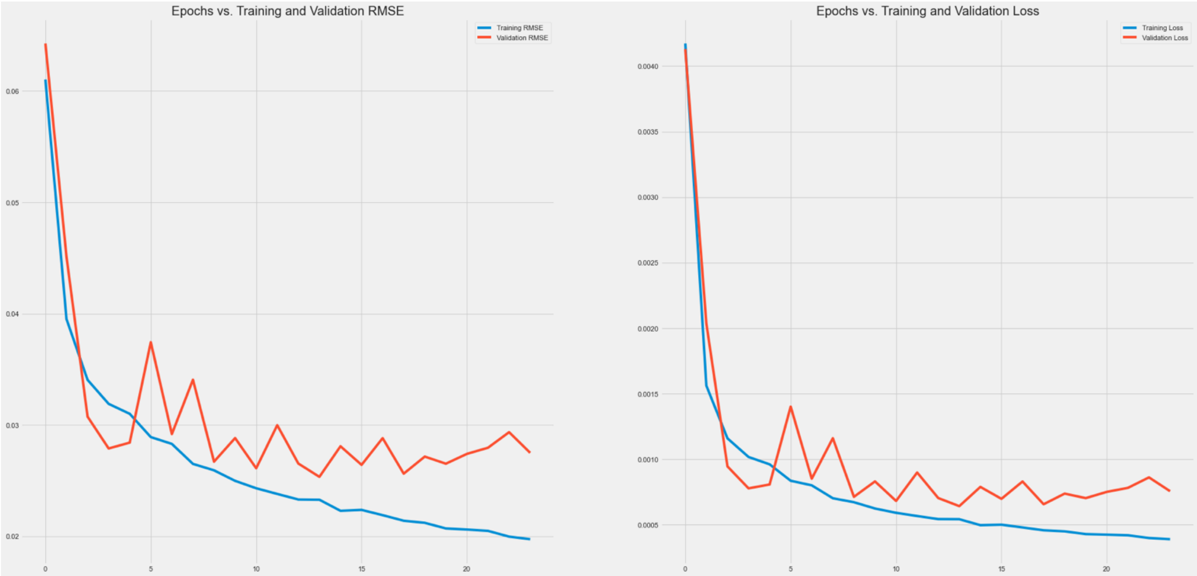
In [102]:
```
history = multivariate_mlp.fit(train, epochs=120,
                               validation_data=validation,
```

```
                                   callbacks=[early_stopping,
                                              model_checkpoint])
```

```
Epoch 1/120
845/845 [==============================] - 7s 6ms/step - loss: 0.0
042 - root_mean_squared_error: 0.0610 - val_loss: 0.0041 - val_roo
t_mean_squared_error: 0.0643
Epoch 2/120
845/845 [==============================] - 5s 6ms/step - loss: 0.0
016 - root_mean_squared_error: 0.0395 - val_loss: 0.0020 - val_roo
t_mean_squared_error: 0.0451
Epoch 3/120
845/845 [==============================] - 5s 6ms/step - loss: 0.0
012 - root_mean_squared_error: 0.0340 - val_loss: 9.4495e-04 - val
_root_mean_squared_error: 0.0307
Epoch 4/120
845/845 [==============================] - 11s 13ms/step - loss:
0.0010 - root_mean_squared_error: 0.0319 - val_loss: 7.7730e-04 -
val_root_mean_squared_error: 0.0279
Epoch 5/120
845/845 [==============================] - 6s 7ms/step - loss: 9.6
062e-04 - root_mean_squared_error: 0.0310 - val_loss: 8.0678e-04 -
val_root_mean_squared_error: 0.0284
Epoch 6/120
845/845 [==============================] - 6s 7ms/step - loss: 8.3
540e-04 - root_mean_squared_error: 0.0289 - val_loss: 0.0014 - val
_root_mean_squared_error: 0.0374
Epoch 7/120
845/845 [==============================] - 5s 6ms/step - loss: 8.0
030e-04 - root_mean_squared_error: 0.0283 - val_loss: 8.5097e-04 -
val_root_mean_squared_error: 0.0292
Epoch 8/120
845/845 [==============================] - 4s 5ms/step - loss: 7.0
193e-04 - root_mean_squared_error: 0.0265 - val_loss: 0.0012 - val
_root_mean_squared_error: 0.0341
Epoch 9/120
845/845 [==============================] - 5s 6ms/step - loss: 6.7
164e-04 - root_mean_squared_error: 0.0259 - val_loss: 7.1259e-04 -
val_root_mean_squared_error: 0.0267
Epoch 10/120
845/845 [==============================] - 5s 6ms/step - loss: 6.2
359e-04 - root_mean_squared_error: 0.0250 - val_loss: 8.3031e-04 -
val_root_mean_squared_error: 0.0288
Epoch 11/120
845/845 [==============================] - 6s 7ms/step - loss: 5.9
076e-04 - root_mean_squared_error: 0.0243 - val_loss: 6.8138e-04 -
val_root_mean_squared_error: 0.0261
Epoch 12/120
845/845 [==============================] - 6s 7ms/step - loss: 5.6
615e-04 - root_mean_squared_error: 0.0238 - val_loss: 8.9829e-04 -
val_root_mean_squared_error: 0.0300
Epoch 13/120
845/845 [==============================] - 6s 7ms/step - loss: 5.4
270e-04 - root_mean_squared_error: 0.0233 - val_loss: 7.0359e-04 -
```

```
val_root_mean_squared_error: 0.0265
Epoch 14/120
845/845 [==============================] - 6s 7ms/step - loss: 5.4
180e-04 - root_mean_squared_error: 0.0233 - val_loss: 6.4164e-04 -
val_root_mean_squared_error: 0.0253
Epoch 15/120
845/845 [==============================] - 5s 6ms/step - loss: 4.9
633e-04 - root_mean_squared_error: 0.0223 - val_loss: 7.8841e-04 -
val_root_mean_squared_error: 0.0281
Epoch 16/120
845/845 [==============================] - 5s 6ms/step - loss: 5.0
001e-04 - root_mean_squared_error: 0.0224 - val_loss: 6.9735e-04 -
val_root_mean_squared_error: 0.0264
Epoch 17/120
845/845 [==============================] - 5s 6ms/step - loss: 4.7
910e-04 - root_mean_squared_error: 0.0219 - val_loss: 8.2972e-04 -
val_root_mean_squared_error: 0.0288
Epoch 18/120
845/845 [==============================] - 6s 7ms/step - loss: 4.5
738e-04 - root_mean_squared_error: 0.0214 - val_loss: 6.5622e-04 -
val_root_mean_squared_error: 0.0256
Epoch 19/120
845/845 [==============================] - 6s 7ms/step - loss: 4.4
934e-04 - root_mean_squared_error: 0.0212 - val_loss: 7.3718e-04 -
val_root_mean_squared_error: 0.0272
Epoch 20/120
845/845 [==============================] - 6s 7ms/step - loss: 4.2
830e-04 - root_mean_squared_error: 0.0207 - val_loss: 7.0280e-04 -
val_root_mean_squared_error: 0.0265
Epoch 21/120
845/845 [==============================] - 6s 7ms/step - loss: 4.2
420e-04 - root_mean_squared_error: 0.0206 - val_loss: 7.5048e-04 -
val_root_mean_squared_error: 0.0274
Epoch 22/120
845/845 [==============================] - 5s 6ms/step - loss: 4.1
930e-04 - root_mean_squared_error: 0.0205 - val_loss: 7.8105e-04 -
val_root_mean_squared_error: 0.0279
Epoch 23/120
845/845 [==============================] - 6s 7ms/step - loss: 3.9
842e-04 - root_mean_squared_error: 0.0200 - val_loss: 8.6099e-04 -
val_root_mean_squared_error: 0.0293
Epoch 24/120
845/845 [==============================] - 6s 7ms/step - loss: 3.8
910e-04 - root_mean_squared_error: 0.0197 - val_loss: 7.5570e-04 -
val_root_mean_squared_error: 0.0275
```

In [103]:
```
plot_model_rmse_and_loss(history)
```




```
<Figure size 432x288 with 0 Axes>
```

In [104]:
```
multivariate_mlp = tf.keras.models.load_model('multivariate_mlp.h5'

forecast = multivariate_mlp.predict(X_test)
multivariate_mlp_forecast = scaler_y.inverse_transform(forecast)

rmse_mult_mlp = sqrt(mean_squared_error(y_test_inv,
                                        multivariate_mlp_forecast))
print('RMSE of hour-ahead electricity price multivariate MLP foreca
      .format(round(rmse_mult_mlp, 3)))
```

```
125/125 [==============================] - 0s 3ms/step
RMSE of hour-ahead electricity price multivariate MLP forecast: 2.
421
```

In [105]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squ

#  (mlp_forecast_inv) and actual values (y_test_inv)
# Calculate R-squared
r_squared = r2_score(y_test_inv, multivariate_mlp_forecast)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test_inv, multivariate_mlp_forecast)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_inv, multivariate_mlp_forecast)

# Print evaluation metrics
print('R-squared (R²): {:.3f}'.format(r_squared))
print('Mean Absolute Error (MAE): {:.3f}'.format(mae))
print('Mean Squared Error (MSE): {:.3f}'.format(mse))
```

```
R-squared (R²): 0.914
Mean Absolute Error (MAE): 1.854
Mean Squared Error (MSE): 5.862
```

Encoder-Decoder

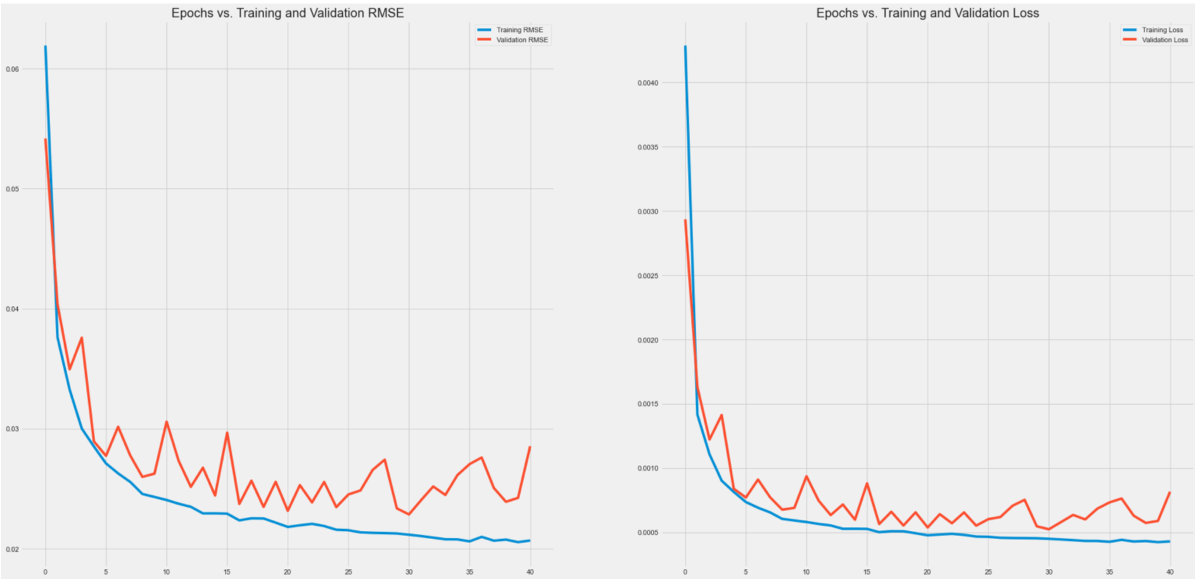
In [106]:
```
tf.keras.backend.clear_session()

encoder_decoder = tf.keras.models.Sequential([
    LSTM(50, activation='relu', input_shape=input_shape),
    RepeatVector(past_history),
    LSTM(50, activation='relu', return_sequences=True),
    TimeDistributed(Dense(50, activation='relu')),
    Flatten(),
    Dense(25, activation='relu'),
    Dense(1)
])



model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
                   'encoder_decoder.h5', save_best_only=True)

optimizer = tf.keras.optimizers.Adam(lr=1e-3, amsgrad=True)

encoder_decoder.compile(loss=loss,
                    optimizer=optimizer,
                    metrics=metric)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instea
d, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Ad
am.
```

In [107]:
```
history = encoder_decoder.fit(train, epochs=50,
                              validation_data=validation,
                              callbacks=[early_stopping,
                                         model_checkpoint])
```

```
Epoch 1/50
845/845 [==============================] - 16s 15ms/step - loss:
0.0043 - root_mean_squared_error: 0.0619 - val_loss: 0.0029 - val_
root_mean_squared_error: 0.0542
Epoch 2/50
845/845 [==============================] - 13s 15ms/step - loss:
0.0014 - root_mean_squared_error: 0.0376 - val_loss: 0.0016 - val_
root_mean_squared_error: 0.0404
Epoch 3/50
845/845 [==============================] - 15s 17ms/step - loss:
0.0011 - root_mean_squared_error: 0.0333 - val_loss: 0.0012 - val_
root_mean_squared_error: 0.0349
Epoch 4/50
845/845 [==============================] - 13s 16ms/step - loss:
9.0073e-04 - root_mean_squared_error: 0.0300 - val_loss: 0.0014 -
val_root_mean_squared_error: 0.0376
Epoch 5/50
845/845 [==============================] - 14s 16ms/step - loss:
8.1322e-04 - root_mean_squared_error: 0.0285 - val_loss: 8.3758e-0
4 - val_root_mean_squared_error: 0.0289
```

In [108]:
```
plot_model_rmse_and_loss(history)
```




```
&lt;Figure size 432x288 with 0 Axes&gt;
```

In [109]:
```
encoder_decoder = tf.keras.models.load_model('encoder_decoder.h5')

forecast = encoder_decoder.predict(X_test)
encoder_decoder_forecast = scaler_y.inverse_transform(forecast)

rmse_encoder_decoder = sqrt(mean_squared_error(y_test_inv,
                                               encoder_decoder_fore
print('RMSE of hour-ahead electricity price Encoder-Decoder forecas
      .format(round(rmse_encoder_decoder, 3)))
```

```
125/125 [==============================] - 1s 6ms/step
RMSE of hour-ahead electricity price Encoder-Decoder forecast: 2.3
46
```

In [110]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squa

#  (encoder_decoder_forecast) and actual values (y_test_inv)
# Calculate R-squared
r_squared = r2_score(y_test_inv, encoder_decoder_forecast)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test_inv, encoder_decoder_forecast)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_inv, encoder_decoder_forecast)

# Print evaluation metrics
print('R-squared (R²): {:.3f}'.format(r_squared))
print('Mean Absolute Error (MAE): {:.3f}'.format(mae))
print('Mean Squared Error (MSE): {:.3f}'.format(mse))
```

```
R-squared (R²): 0.919
Mean Absolute Error (MAE): 1.770
Mean Squared Error (MSE): 5.506
```

In [ ]: