MSING0097 Group Coursework Notebook:

Time Series Forecasting

Group Name: Team Panda

Jinghong Li
Yukun Zhang
Thanh Ha Trinh
Bo Yu
Emma Murphy

Word Count In Markups: 1181

Overview - Power Consumption Dataset

In this notebook, the dataset 2, the "UCI Individual household electric power consumption" dataset, is selected to conduct a time series forecasting project from end-to-end. The dataset is a multivariate time series dataset and it contains measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. The description of the dataset is as follow:

Data Set Information:

This archive contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).
Notes:
1.(global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3) represents the active energy consumed every minute (in watt hour) in the household by electrical equipment not measured in sub-meterings 1, 2 and 3.
2.The dataset contains some missing values in the measurements (nearly 1,25% of the rows). All calendar timestamps are present in the dataset but for some timestamps, the measurement values are missing: a missing value is represented by the absence of value between two consecutive semi-colon attribute separators. For instance, the dataset shows missing values on April 28, 2007.

Attribute Information:

1.date: Date in format dd/mm/yyyy
2.time: time in format hh:mm:ss
3.global_active_power: household global minute-averaged active power (in kilowatt)
4.global_reactive_power: household global minute-averaged reactive power (in kilowatt)
5.voltage: minute-averaged voltage (in volt)
6.global_intensity: household global minute-averaged current intensity (in ampere)
7.sub_metering_1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
8.sub_metering_2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
9.sub_metering_3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

Note: The global_active_power is the total real power consumed by the household, whereas the global_reactive_power is the total unused power in the lines.

The structure of the notebook is as follows：

1. Problem Description
2. Experimental Setup
3. Naive forecast
4. Data Analysis
5. Linear Models
6. Supervised Learning Formulation
7. Supervised Learning And Ensemble Models
8. Advanced Methods
9. Evaluation

---

1. Problem Description

PROBLEM DESCRIPTION

The problem is to predict the total active power consumption for one-day ahead. This requires to build a predictive model producing one-step forecast for the daily total active power consumption on the day t+1.

Accordingly, the study will be focused on the total real power consumption in the dataset **(Global_active_power)**. It is also useful to downsample the per-minute obervations of power consumption to daily totals.

In [1]:

```
# Import libraries

import pandas as pd
import numpy as np

from sklearn.metrics import mean_squared_error
from math import sqrt

import matplotlib.pyplot as plt
%matplotlib inline

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

import warnings
warnings.simplefilter('ignore')

from statsmodels.tsa.arima_model import ARIMA
```

In [2]:

```
# Import the dataset
# Merging date and time columns togeter as one (dt for datetime), and making it index
df = pd.read_csv('household_power_consumption.txt', sep=';',
                 parse_dates={'dt' : ['Date', 'Time']}, infer_datetime_format=True,
                 low_memory=False, index_col='dt', header=0)

# Making sure strings for missing values are converted to numpy nan
df.replace('?', np.nan, inplace=True)

# Make the dataset numeric
df = df.astype('float32')

print("Imported dataset with {0:,d} rows and {1} columns".format(len(df),len(df.columns
)))
```

```
Imported dataset with 2,075,259 rows and 7 columns
```

In [3]:

```
# Save the updated dataset
df.to_csv('household_power_consumption.csv')
```

Data Cleaning - Missing Values

In [4]:

```
# From data description, there are around (1.25%) of rows with all missing data
# Below shows that the missing values are equally across all feature columns
df.isna().sum()
```

Out[4]:

```
Global_active_power      25979
Global_reactive_power    25979
Voltage                  25979
Global_intensity         25979
Sub_metering_1           25979
Sub_metering_2           25979
Sub_metering_3           25979
dtype: int64
```

In [5]:

```
# In order to keep the integrity of the timeline, we need to fill in nan's with a copy
 of the observation from 24 hours previously.
# Function to fill missing values with a value at the same time one day ago
def fill_missing(values):
    one_day = 60 * 24
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if np.isnan(values[row, col]):
                values[row, col] = values[row - one_day, col]
```

In [6]:

```
# Apply the function to the dataset
fill_missing(df.values)
```

In [7]:

```
# Check if this worked well to eliminate all missing values
df.isna().sum()
```

Out[7]:

```
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```

In [8]:

```
# Save the updated dataset
df.to_csv('household_power_consumption_cleaned_up.csv')
```

Resampling The Dataset To Daily

In [9]:

```
# Resample minute data to total for each day

data = pd.read_csv('household_power_consumption_cleaned_up.csv', header=0, infer_dateti
me_format=True,
parse_dates=['dt'], index_col=['dt'])

# Resample data to daily
data_daily = data.resample('D').sum()

# Add in a date column as string for later use as series index
data_daily['Date'] = pd.to_datetime(data_daily.index.astype(str))

data_daily.to_csv('household_power_consumption_days.csv')
```

In [10]:

```
print("data_daily Structure: ", data_daily.shape)
data_daily.head(3)
```

```
data_daily Structure:  (1442, 8)
```

Out[10]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	S
dt					
2006-12-16	1209.176	34.922	93552.53	5180.8	0
2006-12-17	3390.460	226.006	345725.32	14398.6	2
2006-12-18	2203.826	161.792	347373.64	9247.2	1

2. Experimental Setup

For the purposes of the experiment, train and test set splitting is performed on the dataset in this section.

In [11]:

```
print("The Timeframe of data_daily: ", data_daily.index.min(),'-',data_daily.index.max
())
```

```
The Timeframe of data_daily:  2006-12-16 00:00:00 - 2010-11-26 00:00:00
```

In [12]:

```
data_daily.iloc[-330]
```

Out[12]:

```
Global_active_power                    1224.25
Global_reactive_power                  165.336
Voltage                                 349295
Global_intensity                        5093.4
Sub_metering_1                            2304
Sub_metering_2                             327
Sub_metering_3                            3558
Date                       2010-01-01 00:00:00
Name: 2010-01-01 00:00:00, dtype: object
```

In [13]:

```
# Train test split
train, validation = data_daily[0:-330], data_daily[-330:]

print('Train observations: {0}, Validation observations: {1}'.format(len(train), len(va
lidation)))
print('\nTrain date range:',train.index.min(),'-',train.index.max())
print('Validation date range:',validation.index.min(),'-',validation.index.max())
```

```
Train observations: 1112, Validation observations: 330

Train date range: 2006-12-16 00:00:00 - 2009-12-31 00:00:00
Validation date range: 2010-01-01 00:00:00 - 2010-11-26 00:00:00
```

In [14]:

```
train.to_csv('household_power_consumption_days_train.csv')
validation.to_csv('household_power_consumption_days_validation.csv')
```

EXPERIMENTAL SETUP

Since the timeframe of the dataset is almost 4 years, the first 3 years of data (2006-2009) is selected for training predictive models and the final year (2010) is for evaluating models. Accordingly, the train set contains 1112 observations and it ranges from 2006-12-16 to 2009-12-31. The validation set has 330 observations and it ranges from 2010-01-01 to 2010-11-26. The validation dataset is about 23% of the original dataset.

3. Naive Forecast

In this section, Naive forecast is studied in order to provide a quantitative idea of how difficult the forecast problem is and also to provide a baseline performance by which more sophisticated forecast methods can be evaluated.

In [15]:

```
# Creating a series for GlobalActivePower feature
train['Date'] = pd.to_datetime(train['Date'].astype(str))
GAP_series = pd.Series(train['Global_active_power'].values , index=train['Date'])
```

In [16]:

```
len(GAP_series)
```

Out[16]:

```
1112
```

In [17]:

```
# Prepare data
X_naive = GAP_series.values
X_naive = X_naive.astype('float32')
```

In [18]:

```
int(len(X_naive)/2)
```

Out[18]:

```
556
```

In [19]:

```
# Walk-forward validation
train_naive, test_naive = X_naive[0:556], X_naive[556:]

history = [x for x in train_naive]
predictions_naive = list()
for i in range(len(test_naive)):

    # predict
    yhat = history[-1]
    predictions_naive.append(yhat)

    # observation
    obs = test_naive[i]
    history.append(obs)

# Report performance
rmse_Naive = sqrt(mean_squared_error(test_naive, predictions_naive))
print('Naive RMSE: %.3f' % rmse_Naive)
```

```
Naive RMSE: 477.762
```

NAIVE FORECAST

We develop a daily persistence model to conduct the naive forecast. This model takes the active power on day t and uses it as the value of the power on day t+1. The walk-forward validation approach is selected to evaluate the persistence model on **test_naive** to predict the next sequence of values based on the prior one.

Root Mean Squared Error (RMSE) is adopted as the evaluation metric for this notebook because RMSE punishes forecast errors heavily. Based on the validation result, an overall RMSE of 477.762 is obtained and the score suggests that on average, the model was incorrect by about 478 kilowatts active power for each prediction made. The overall RMSE score serves as a benchmark for further predictive methodologies.

---

4. Data Analysis

In this section, summary statistics and plots of the data will be applied to the Global_active_power in the variable **(train)** in order to learn about the structure of the prediction problem. The data analysis will be focused on five perspectives:

1. Summary Statistics
2. Seasonal Line Plots
3. Density Plot
4. Box And Whisker Plot

Summary Statistics

In [20]:

```
train.Global_active_power.describe().round(1)
```

Out[20]:

```
count    1112.0
mean     1580.8
std       627.6
min       250.3
25%      1161.6
50%      1558.8
75%      1935.0
max      4773.4
Name: Global_active_power, dtype: float64
```

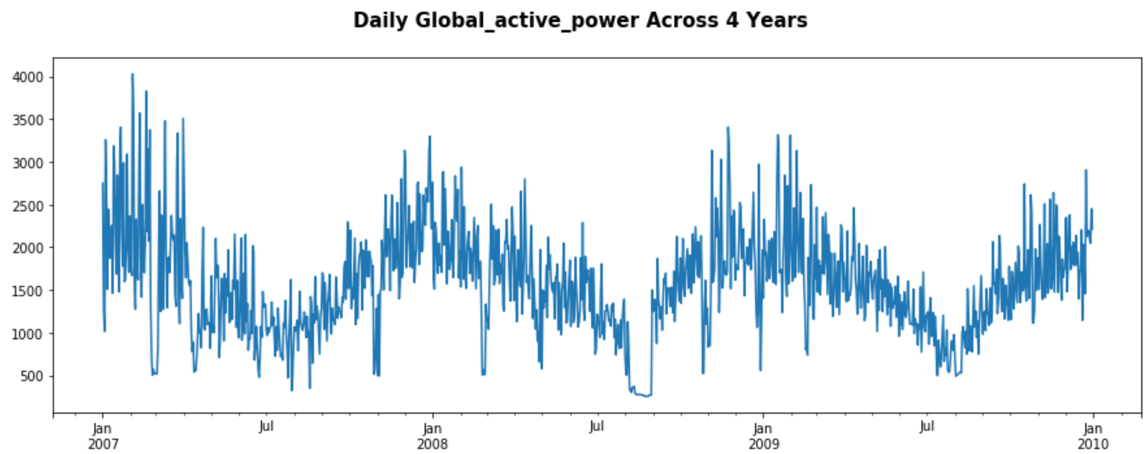
Seasonal Line Plots (Excluding data in 2006)

In [21]:

```
plt.figure(figsize=(15, 8))
groups = train["2007":].Global_active_power.groupby(pd.Grouper(freq='A'))

years = pd.DataFrame()
i=1
n_groups = len(groups)
for name, group in groups:
    plt.subplot(n_groups, 1, i)
    i += 1
    plt.plot(group)
plt.show()
```

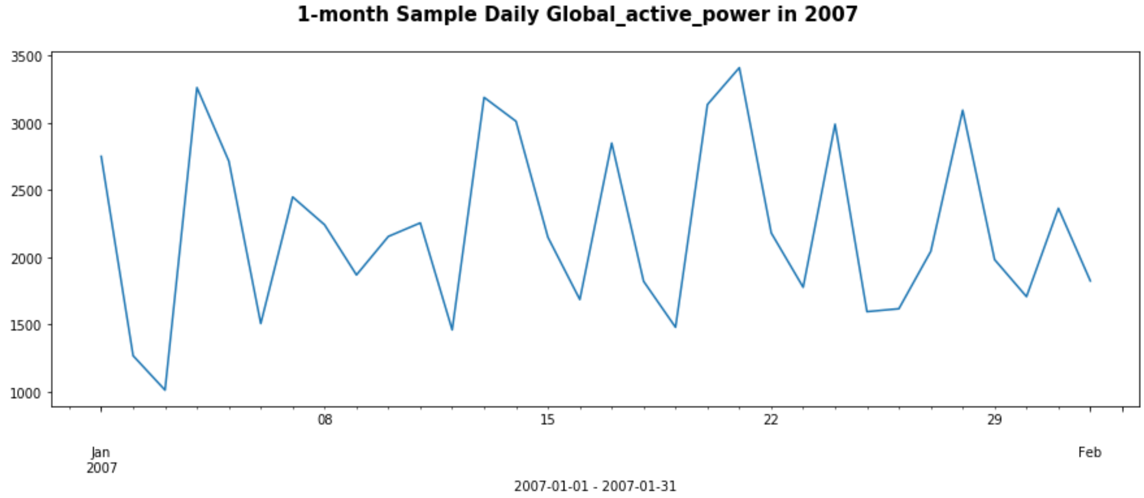
In [22]:

```
fig = plt.figure(figsize=(15,5))
fig.suptitle('Daily Global_active_power Across 4 Years', fontsize=15, fontweight='bold'
)
train["2007":].Global_active_power.plot()
plt.xlabel('')
plt.show()
```



In [23]:

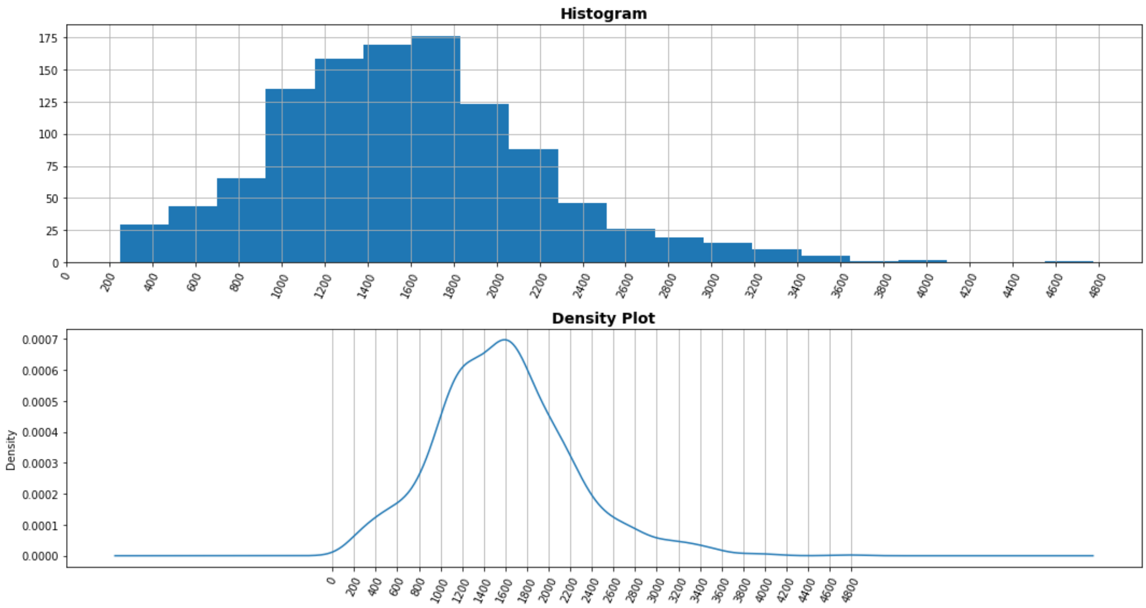
```
fig = plt.figure(figsize=(15,5))
fig.suptitle('1-month Sample Daily Global_active_power in 2007', fontsize=15, fontweigh
t='bold')
train['2007-01-01': '2007-02-01'].Global_active_power.plot()
plt.xlabel('2007-01-01 - 2007-01-31')
plt.show()
```



Density Plot

In [24]:

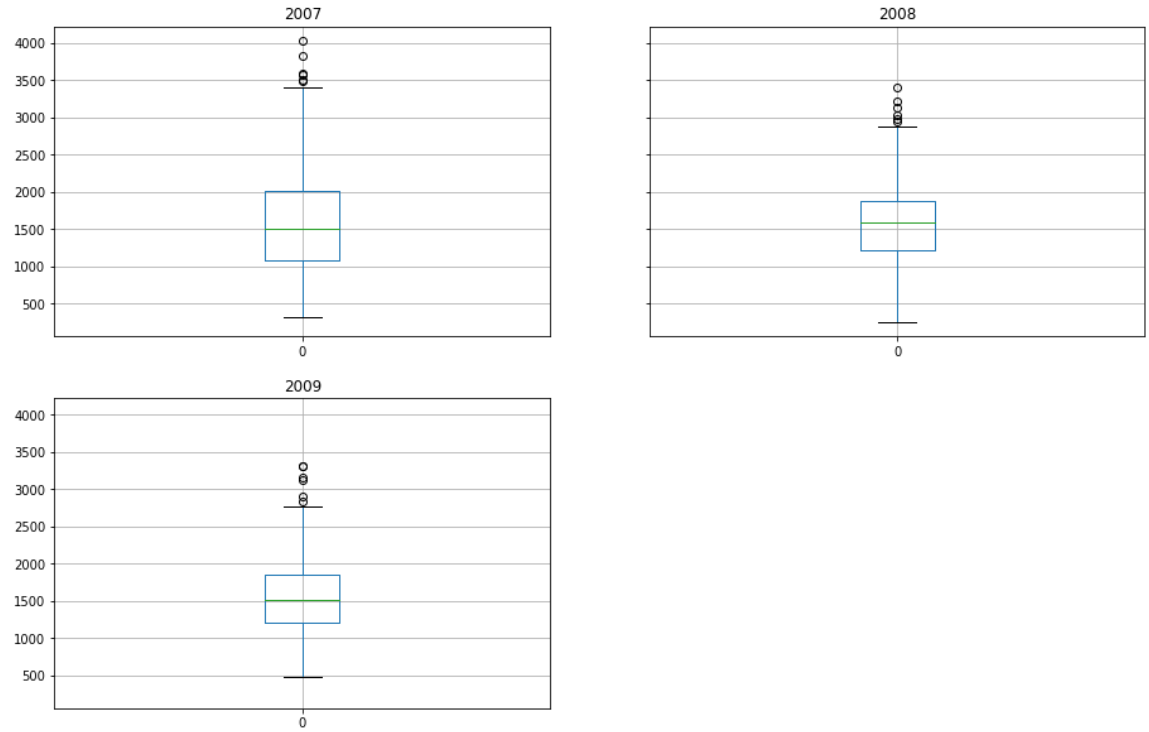
```
plt.figure(figsize=(15, 8))

plt.subplot(211)
train.Global_active_power.hist(bins=20)
plt.title('Histogram', fontsize=14, fontweight='bold')
plt.xticks(np.arange(0, 5000, 200), rotation=65)

plt.subplot(212)
train.Global_active_power.plot(kind='kde')
plt.title('Density Plot', fontsize=14, fontweight='bold')
plt.xticks(np.arange(0, 5000, 200), rotation=65)
plt.grid(axis='x')

plt.tight_layout()
plt.show()
```



Box And Whisker Plot (Excluding data in 2006)

In [25]:

```
series = pd.Series(train['2007':].Global_active_power.values, index=train['2007':].inde
x.year)
years= pd.DataFrame(series)
year_groups = years.groupby('dt')
year_groups.boxplot(figsize=(15, 10))
plt.show()
```



DATA ANALYSIS

Summary Statistics

The number of observations (count) matches our expectation, meaning we are handling the data correctly.
The daily mean is 1580.8 kw, which we might consider our level in this series.
The standard deviation (average spread from the mean) is relatively large at 627.6 sales.
The percentiles along with the standard deviation do suggest a large spread to the data.

Seasonal Line Plots (Excluding data in 2006)

There looks to be a 'v-shaped' patttern in each year, with higher consumption at the start and end of the year, with a dip in the middle. This could coincide with higher consumption in winter months due to cold weather and spending more time at home, as well as possible non-occupancy of the home during holiday period.

Both "Daily Global_active_power Across 4 Years" and "1-month Sample Daily Global_active_power in 2007" plots indicate a day trend in the dataset. Hence, it is worth detrending the dataset by differencing the data day to day.

Density Plot

Distributions are centred around 1600, somewhat longer tail on the right, but overall looks like a reasonably symmetrical distribution, not perfectly Gaussian but a resemblance to a bell-shaped curve.

Box And Whisker Plot (Excluding data in 2006)

The median values for each year is stable across the years.
The spread or middle 50% of the data decreases across the years.
There are outliers each year; these may be some extreme readings during winter months across the years.

---

5. Linear Models

In this section, automatic configuration of the ARIMA model using Grid Search will be performed. This will be followed by investigating the residual errors of the chosen model.

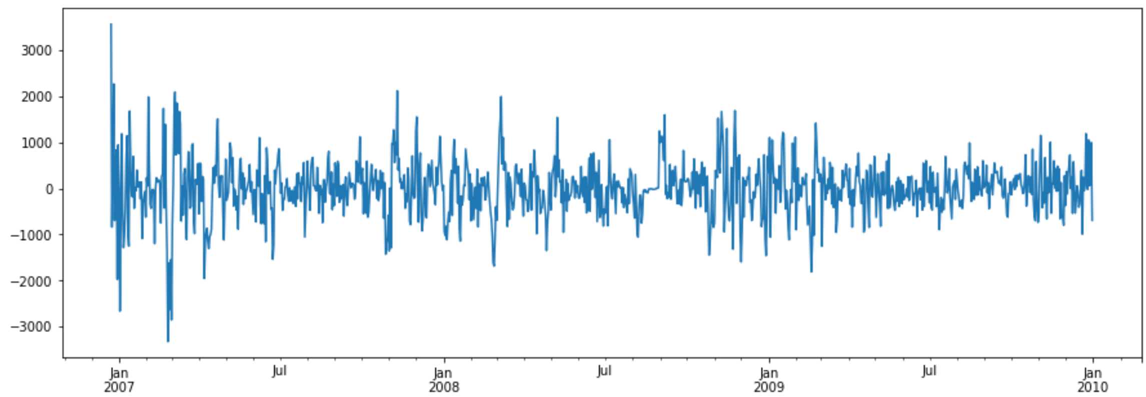
As such, this section is broken down into the following steps:

1. Check Stationarity
2. Automatic Configuration Of The ARIMA Model
3. Review Residual Errors Of The Chosen ARIMA Model
4. Finalize The Model
5. Make Predictions

Check Stationarity

In [370]:

```
# create and summarize stationary version of time series
from statsmodels.tsa.stattools import adfuller

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)

X_arima = GAP_series.values
X_arima = X_arima.astype('float32')

# difference data
days_in_week = 7
stationary = difference(X_arima, days_in_week)
stationary.index = GAP_series.index[days_in_week:]

plt.figure(5,figsize=(15, 5))
stationary.plot()
plt.xlabel("")
plt.show(5)
```



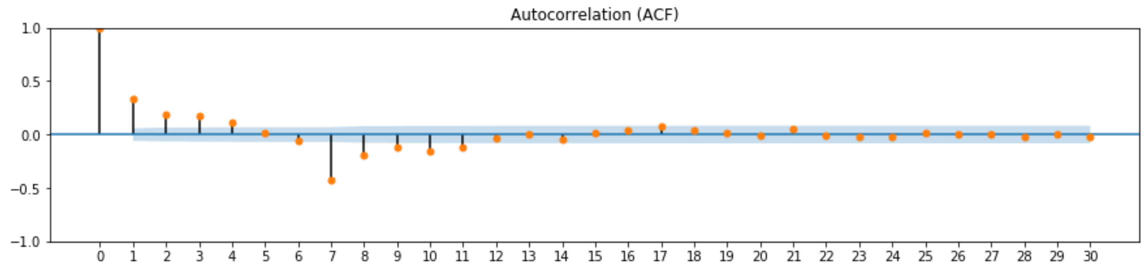
In [371]:

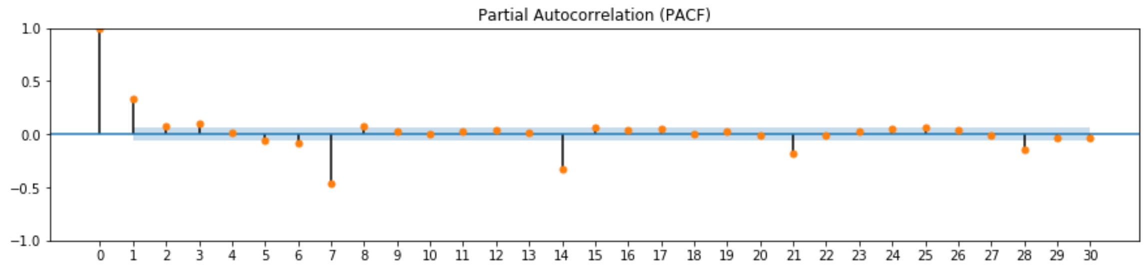
```
# check if stationary
result = adfuller(stationary)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')

for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -9.544446
p-value: 0.000000
Critical Values:
	1%: -3.436
	5%: -2.864
	10%: -2.568
```

In [372]:

```
# ACF and PACF plots of time series

plt.figure(6,figsize=(15, 3))
plt.plot(211)
pylab.ylim([-1,1])
plot_acf(stationary, ax=plt.gca(),lags=30, title='Autocorrelation (ACF)')
plt.xticks(np.arange(0,31,1))
plt.yticks(np.arange(-1,1.5,0.5))
plt.show()

plt.figure(7,figsize=(15, 3))
plt.plot(212)
pylab.ylim([-1,1])
plot_pacf(stationary, method='ywm', ax=plt.gca(), lags=30, title='Partial Autocorrelati
on (PACF)')
plt.xticks(np.arange(0,31,1))
plt.yticks(np.arange(-1,1.5,0.5))
plt.show()
```





Automatic Configuration Of The ARIMA Model

In [373]:

```
# evaluate manually configured ARIMA model

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# load data
df = pd.read_csv('household_power_consumption_days_train.csv', index_col=False, header=
0);
df['dt'] = pd.to_datetime(df['dt'].astype(str))
GAP_series = pd.Series(df['Global_active_power'].values , index=df['dt']) # here we con
vert the DataFrame into a Series
```

In [374]:

```
# Grid Search ARIMA Hyperparameters
# evaluate an ARIMA model for a given order (p,d,q) and return RMSE
def evaluate_arima_model(V, arima_order): # prepare training dataset
    V = V.astype('float32')
    train_size = int(len(V) * 0.50)
    train_arima, test_arima = V[0:train_size], V[train_size:]
    history_arima = [n for n in train_arima]
    # make predictions
    predictions_arima = list()
    for t in range(len(test_arima)):
    # difference data
        days_in_week = 7
        diff = difference(history_arima, days_in_week)
        model = ARIMA(diff, order=arima_order)
        model_fit = model.fit(trend='nc', disp=0)
        yhat = model_fit.forecast()[0]
        yhat = inverse_difference(history_arima, yhat, days_in_week)
        predictions_arima.append(yhat)
        history_arima.append(test_arima[t])
    # calculate out of sample error
    rmse = sqrt(mean_squared_error(test_arima, predictions_arima))
    return rmse

# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    rmse = evaluate_arima_model(dataset, order)
                    if rmse < best_score:
                        best_score, best_cfg = rmse, order
                    print('ARIMA%s RMSE=%.3f' % (order,rmse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))
```

In [36]:

```
# Evaluate ARIMA parameters (Please note this cell has extremely long run time. The cor
responding result is provided as a picture below.)
p_values = range(0, 6)
d_values = range(0, 2)
q_values = range(0, 6)
evaluate_models(GAP_series.values, p_values, d_values, q_values)
```

```
ARIMA(0, 0, 1) RMSE=480.008
ARIMA(0, 0, 2) RMSE=478.823
ARIMA(0, 0, 3) RMSE=480.862
ARIMA(0, 0, 4) RMSE=465.004
ARIMA(0, 0, 5) RMSE=466.760
ARIMA(0, 1, 1) RMSE=511.195
ARIMA(0, 1, 2) RMSE=480.361
ARIMA(1, 0, 0) RMSE=478.414
ARIMA(1, 0, 1) RMSE=478.977
ARIMA(1, 0, 2) RMSE=479.086
ARIMA(1, 0, 3) RMSE=482.573
ARIMA(1, 0, 4) RMSE=473.490
ARIMA(1, 1, 0) RMSE=551.900
ARIMA(1, 1, 1) RMSE=478.739
ARIMA(1, 1, 2) RMSE=481.478
ARIMA(1, 1, 3) RMSE=479.512
ARIMA(1, 1, 4) RMSE=483.143
ARIMA(1, 1, 5) RMSE=465.048
ARIMA(2, 0, 0) RMSE=479.553
ARIMA(2, 0, 1) RMSE=479.003
ARIMA(2, 0, 2) RMSE=478.470
ARIMA(2, 0, 3) RMSE=451.086
ARIMA(2, 0, 4) RMSE=446.271
ARIMA(2, 0, 5) RMSE=446.141
ARIMA(2, 1, 0) RMSE=526.670
ARIMA(2, 1, 1) RMSE=492.434
ARIMA(2, 1, 4) RMSE=453.958
ARIMA(2, 1, 5) RMSE=449.891
ARIMA(3, 0, 0) RMSE=479.141
ARIMA(3, 0, 1) RMSE=479.247
ARIMA(3, 0, 2) RMSE=443.653
ARIMA(3, 0, 4) RMSE=437.493
ARIMA(3, 1, 0) RMSE=513.207
ARIMA(3, 1, 1) RMSE=506.544
ARIMA(3, 1, 2) RMSE=504.424
ARIMA(3, 1, 5) RMSE=437.878
ARIMA(4, 0, 0) RMSE=479.409
ARIMA(4, 0, 2) RMSE=442.817
ARIMA(4, 1, 0) RMSE=511.030
ARIMA(4, 1, 1) RMSE=510.948
ARIMA(4, 1, 2) RMSE=472.282
ARIMA(4, 1, 3) RMSE=443.036
ARIMA(5, 0, 0) RMSE=480.821
ARIMA(5, 0, 1) RMSE=478.614
ARIMA(5, 0, 2) RMSE=471.987
ARIMA(5, 0, 3) RMSE=439.508
ARIMA(5, 0, 4) RMSE=415.630
ARIMA(5, 0, 5) RMSE=413.251
ARIMA(5, 1, 0) RMSE=510.743
ARIMA(5, 1, 1) RMSE=495.952
ARIMA(5, 1, 2) RMSE=474.710
ARIMA(5, 1, 3) RMSE=443.299
ARIMA(5, 1, 5) RMSE=414.393
Best ARIMA(5, 0, 5) RMSE=413.251
```

```
ARIMA(0, 0, 1) RMSE=480.008
ARIMA(0, 0, 2) RMSE=478.823
ARIMA(0, 0, 3) RMSE=480.862
ARIMA(0, 0, 4) RMSE=465.004
ARIMA(0, 0, 5) RMSE=466.760
ARIMA(0, 1, 1) RMSE=511.195
ARIMA(0, 1, 2) RMSE=480.361
ARIMA(1, 0, 0) RMSE=478.414
ARIMA(1, 0, 1) RMSE=478.977
ARIMA(1, 0, 2) RMSE=479.086
ARIMA(1, 0, 3) RMSE=482.573
ARIMA(1, 0, 4) RMSE=473.490
ARIMA(1, 1, 0) RMSE=551.900
ARIMA(1, 1, 1) RMSE=478.739
ARIMA(1, 1, 2) RMSE=481.478
ARIMA(1, 1, 3) RMSE=479.512
ARIMA(1, 1, 4) RMSE=483.143
ARIMA(1, 1, 5) RMSE=465.048
ARIMA(2, 0, 0) RMSE=479.553
ARIMA(2, 0, 1) RMSE=479.003
ARIMA(2, 0, 2) RMSE=478.470
ARIMA(2, 0, 3) RMSE=451.086
ARIMA(2, 0, 4) RMSE=446.271
ARIMA(2, 0, 5) RMSE=446.141
ARIMA(2, 1, 0) RMSE=526.670
ARIMA(2, 1, 1) RMSE=492.434
ARIMA(2, 1, 4) RMSE=453.958
ARIMA(2, 1, 5) RMSE=449.891
ARIMA(3, 0, 0) RMSE=479.141
ARIMA(3, 0, 1) RMSE=479.247
ARIMA(3, 0, 2) RMSE=443.653
ARIMA(3, 0, 4) RMSE=437.493
ARIMA(3, 1, 0) RMSE=513.207
ARIMA(3, 1, 1) RMSE=506.544
ARIMA(3, 1, 2) RMSE=504.424
ARIMA(3, 1, 5) RMSE=437.878
ARIMA(4, 0, 0) RMSE=479.409
ARIMA(4, 0, 2) RMSE=442.817
ARIMA(4, 1, 0) RMSE=511.030
ARIMA(4, 1, 1) RMSE=510.948
ARIMA(4, 1, 2) RMSE=472.282
ARIMA(4, 1, 3) RMSE=443.036
ARIMA(5, 0, 0) RMSE=480.821
ARIMA(5, 0, 1) RMSE=478.614
ARIMA(5, 0, 2) RMSE=471.987
ARIMA(5, 0, 3) RMSE=439.508
ARIMA(5, 0, 4) RMSE=415.630
ARIMA(5, 0, 5) RMSE=413.251
ARIMA(5, 1, 0) RMSE=510.743
ARIMA(5, 1, 1) RMSE=495.952
ARIMA(5, 1, 2) RMSE=474.710
ARIMA(5, 1, 3) RMSE=443.299
ARIMA(5, 1, 5) RMSE=414.393
Best ARIMA(5, 0, 5) RMSE=413.251
```

Review Residual Errors Of The Chosen ARIMA Model

In [78]:

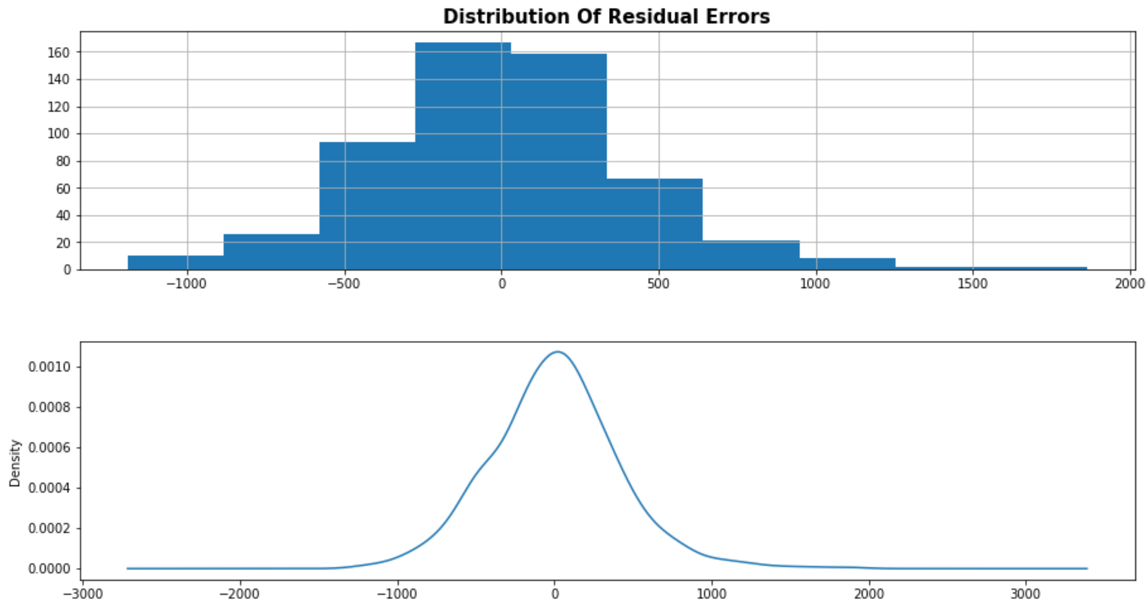
```
# Review Residual Errors on ARIMA (5,0,5) (Long run time, result is reported as a pictu
re)

# prepare data
X_arima505 = GAP_series.values
X_arima505 = X_arima505.astype('float32')
train_size = int(len(X_arima505) * 0.50)
train_arima505, test_arima505 = X_arima505[0:train_size], X_arima505[train_size:]

# walk-forward validation
history_arima505 = [f for f in train_arima505]
predictions_arima505 = list()

for i in range(len(test_arima505)):
    # difference data
    diff = difference(history_arima505, days_in_week)

    # predict
    model = ARIMA(diff, order=(5,0,5))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = inverse_difference(history_arima505, yhat, days_in_week)
    predictions_arima505.append(yhat)

    # observation
    obs = test_arima505[i]
    history_arima505.append(obs)

# errors
residuals_arima505 = [test_arima505[i]-predictions_arima505[i] for i in range(len(test_
arima505))]
residuals_arima505 = pd.DataFrame(residuals_arima505)
print(residuals_arima505.describe())
```

```
                 0
count   556.000000
mean      3.500330
std     413.608315
min   -1189.436318
25%    -238.525012
50%       2.804148
75%     247.038147
max    1863.371672
```

	0
count	556.000000
mean	3.500330
std	413.608315
min	-1189.436318
25%	-238.525012
50%	2.804148
75%	247.038147
max	1863.371672

In [377]:

```
# Plot the distribution of residual errors
plt.figure(figsize=(15, 8))
plt.subplot(211)
residuals_arima505.hist(ax=plt.gca())
plt.title("Distribution Of Residual Errors", fontweight='bold', size=15)
plt.subplot(212)
residuals_arima505.plot(kind='kde', ax=plt.gca(), legend=False)
plt.show()
```



In prinicple we can use this information to bias-correct predictions by adding the mean residual error of to each forecast made. This step is skipped for more efficient notebook processing.

Finalize The Model - ARIMA (5,0,5) with bias = 3.500330

In [378]:

```
# load data
X_Finalize = GAP_series.values
X_Finalize = X_Finalize.astype('float32')

# difference data
days_in_week = 7
diff = difference(X_Finalize, days_in_week)

# fit model
model = ARIMA(diff, order=(5,0,5))
model_fit = model.fit(trend='nc', disp=0)

# bias constant, in-sample mean residual
bias = 3.500330

# save model
model_fit.save('model_GAP.pkl')
np.save('model_bias_GAP', [bias])
```

Make Predictions

In [83]:

```
# EM, from statsmodel version 0.9 onwards, the monkey patch below isn't needed, comment
ed out.
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)
```

```
statsmodels: 0.9.0
```

In [84]:

```
from statsmodels.tsa.arima_model import ARIMAResults

# monkey patch around bug in ARIMA class *** IGNORE THIS ***
#def __getnewargs__(self):
#    return ((self.endog),(self.k_lags, self.k_diff, self.k_ma)) #https://en.wikipedia.
org/wiki/Monkey_patch

#ARIMA.__getnewargs__ = __getnewargs__

# load finalized model and make a prediction

days_in_week = 7
model_fit = ARIMAResults.load('model_GAP.pkl')
bias = np.load('model_bias_GAP.npy')

yhat = float(model_fit.forecast()[0])
yhat = bias + inverse_difference(GAP_series.values, yhat, days_in_week)
print('Predicted: %.3f' % yhat)
```

```
Predicted: 2020.220
```

In [87]:

```
# Load and evaluate the finalized model on the validation dataset

# Load data
X_evaluate = GAP_series.values
X_evaluate = X_evaluate.astype('float32')

# Creating a series for GlobalActivePower feature in validation dataset
validation['Date'] = pd.to_datetime(validation['Date'].astype(str))
GAP_series_validation = pd.Series(validation['Global_active_power'].values , index=vali
dation['Date'])
y_evaluate = GAP_series_validation.values.astype('float32')

history_evaluate = [x for x in X_evaluate]
days_in_week = 7

# Load model
model_fit = ARIMAResults.load('model_GAP.pkl')
bias = np.load('model_bias_GAP.npy')

# make first prediction
predictions_evaluate = list()
yhat = float(model_fit.forecast()[0])
yhat = bias + inverse_difference(history_evaluate, yhat, days_in_week)
predictions_evaluate.append(yhat)
history_evaluate.append(y_evaluate[0])
print('>Predicted=%.3f, Expected=%3.f' % (yhat, y_evaluate[0]))

# rolling forecasts
for i in range(1, len(y_evaluate)):
    # difference data
    days_in_week = 7
    diff = difference(history_evaluate, days_in_week)

    # predict
    model = ARIMA(diff, order=(5,0,5))
    model_fit = model.fit(trend='nc', disp=0)
    yhat = model_fit.forecast()[0]
    yhat = bias + inverse_difference(history_evaluate, yhat, days_in_week)
    predictions_evaluate.append(yhat)

    # observation
    obs = y_evaluate[i]
    history_evaluate.append(obs)

    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))

# report performance
rmse_evaluate = sqrt(mean_squared_error(y_evaluate, predictions_evaluate))
print('RMSE: %.3f' % rmse_evaluate)
```

```
>Predicted=2020.220, Expected=1224
>Predicted=2028.168, Expected=2028
>Predicted=2443.753, Expected=2182
>Predicted=1707.504, Expected=1755
>Predicted=1641.171, Expected=1326
>Predicted=2064.459, Expected=2169
>Predicted=2010.451, Expected=1827
>Predicted=1269.850, Expected=1528
>Predicted=2017.422, Expected=1980
>Predicted=2306.015, Expected=2137
>Predicted=1582.476, Expected=1918
>Predicted=1621.491, Expected=1973
>Predicted=2150.920, Expected=2127
>Predicted=1936.269, Expected=1804
>Predicted=1726.727, Expected=1823
>Predicted=2209.225, Expected=2455
>Predicted=2283.767, Expected=2265
>Predicted=1834.431, Expected=1777
>Predicted=2024.075, Expected=2095
>Predicted=2228.298, Expected=2019
>Predicted=1776.947, Expected=2034
>Predicted=1956.958, Expected=2255
>Predicted=2502.448, Expected=2282
>Predicted=2161.824, Expected=2344
>Predicted=2073.176, Expected=1889
>Predicted=2108.720, Expected=2288
>Predicted=2143.927, Expected=2302
>Predicted=2103.716, Expected=2157
>Predicted=2161.965, Expected=1888
>Predicted=2377.640, Expected=2692
>Predicted=2495.972, Expected=2644
>Predicted=2012.513, Expected=1631
>Predicted=2092.056, Expected=2346
>Predicted=2588.652, Expected=2381
>Predicted=1982.858, Expected=1908
>Predicted=1847.007, Expected=2137
>Predicted=2816.173, Expected=2782
>Predicted=2444.967, Expected=2440
>Predicted=1797.638, Expected=1604
>Predicted=2326.653, Expected=2681
>Predicted=2595.398, Expected=2359
>Predicted=1782.340, Expected=1913
>Predicted=2148.787, Expected=2244
>Predicted=2922.543, Expected=2285
>Predicted=2096.938, Expected=1682
>Predicted=1674.798, Expected=1911
>Predicted=2490.119, Expected=2350
>Predicted=1933.208, Expected=1653
>Predicted=1585.933, Expected=1696
>Predicted=2254.044, Expected=2009
>Predicted=2149.352, Expected=2167
>Predicted=1702.214, Expected=1867
>Predicted=1931.666, Expected=2218
>Predicted=2299.090, Expected=1661
>Predicted=1512.403, Expected=2069
>Predicted=2014.107, Expected=1970
>Predicted=2032.427, Expected=1474
>Predicted=1846.738, Expected=2163
>Predicted=2252.616, Expected=2209
>Predicted=1850.384, Expected=1234
>Predicted=1458.541, Expected=1708
```

```
>Predicted=2150.651, Expected=1008
>Predicted=1337.907, Expected=1182
>Predicted=1308.989, Expected=1298
>Predicted=1791.926, Expected=1076
>Predicted=1285.170, Expected=652
>Predicted=971.531, Expected=1211
>Predicted=1345.009, Expected=1359
>Predicted=699.557, Expected=1029
>Predicted=1016.264, Expected=1873
>Predicted=1687.807, Expected=2002
>Predicted=1371.060, Expected=1386
>Predicted=1278.521, Expected=2247
>Predicted=2156.835, Expected=1995
>Predicted=1547.789, Expected=1808
>Predicted=1478.801, Expected=1763
>Predicted=2253.013, Expected=1734
>Predicted=1796.625, Expected=1628
>Predicted=1672.463, Expected=2090
>Predicted=2205.056, Expected=1651
>Predicted=1608.028, Expected=1667
>Predicted=1799.591, Expected=1570
>Predicted=1626.563, Expected=1995
>Predicted=1710.546, Expected=1680
>Predicted=1640.490, Expected=1651
>Predicted=2001.865, Expected=1848
>Predicted=1888.367, Expected=1798
>Predicted=1539.644, Expected=1623
>Predicted=1683.795, Expected=1906
>Predicted=1838.507, Expected=1924
>Predicted=1686.714, Expected=1687
>Predicted=1726.624, Expected=1649
>Predicted=2007.659, Expected=1805
>Predicted=1851.337, Expected=1690
>Predicted=1558.743, Expected=1119
>Predicted=1608.688, Expected=1826
>Predicted=1770.900, Expected=1597
>Predicted=1362.254, Expected=1649
>Predicted=1585.726, Expected=1233
>Predicted=1784.404, Expected=1190
>Predicted=1498.813, Expected=1603
>Predicted=1325.521, Expected=1439
>Predicted=1394.834, Expected=1513
>Predicted=1414.050, Expected=1660
>Predicted=1646.911, Expected=1872
>Predicted=1444.114, Expected=1562
>Predicted=1566.302, Expected=1558
>Predicted=1858.885, Expected=1476
>Predicted=1516.033, Expected=1709
>Predicted=1576.271, Expected=1834
>Predicted=1571.308, Expected=1580
>Predicted=1683.646, Expected=1461
>Predicted=1652.672, Expected=1181
>Predicted=1530.144, Expected=1518
>Predicted=1560.221, Expected=1543
>Predicted=1432.182, Expected=1083
>Predicted=1384.376, Expected=1251
>Predicted=1417.127, Expected=868
>Predicted=1093.806, Expected=1104
>Predicted=1096.650, Expected=1119
>Predicted=1288.331, Expected=1320
>Predicted=1250.185, Expected=2188
```

```
>Predicted=1433.547, Expected=1411
>Predicted=1222.907, Expected=1517
>Predicted=1481.950, Expected=1805
>Predicted=1731.095, Expected=1299
>Predicted=1143.120, Expected=1644
>Predicted=1859.063, Expected=1533
>Predicted=1862.252, Expected=1567
>Predicted=1311.627, Expected=1468
>Predicted=1631.642, Expected=1571
>Predicted=1488.461, Expected=2035
>Predicted=1594.049, Expected=1555
>Predicted=1533.691, Expected=1722
>Predicted=1875.477, Expected=1945
>Predicted=1897.566, Expected=2100
>Predicted=1610.542, Expected=1628
>Predicted=1817.309, Expected=1421
>Predicted=1836.971, Expected=1427
>Predicted=1580.812, Expected=1154
>Predicted=1367.617, Expected=1294
>Predicted=1630.768, Expected=2096
>Predicted=1849.799, Expected=1488
>Predicted=1181.294, Expected=1341
>Predicted=1639.057, Expected=1199
>Predicted=1422.741, Expected=1388
>Predicted=1168.698, Expected=1323
>Predicted=1336.337, Expected=1398
>Predicted=1698.656, Expected=2050
>Predicted=1677.840, Expected=1666
>Predicted=1303.698, Expected=1340
>Predicted=1566.997, Expected=1269
>Predicted=1532.278, Expected=981
>Predicted=1157.693, Expected=1394
>Predicted=1415.135, Expected=1421
>Predicted=1557.547, Expected=2010
>Predicted=1624.552, Expected=1747
>Predicted=1431.492, Expected=1012
>Predicted=1363.400, Expected=1362
>Predicted=1466.129, Expected=1535
>Predicted=1331.494, Expected=1215
>Predicted=1225.557, Expected=1471
>Predicted=1914.939, Expected=1973
>Predicted=1668.015, Expected=1674
>Predicted=1140.782, Expected=1218
>Predicted=1607.999, Expected=1224
>Predicted=1505.009, Expected=1768
>Predicted=1376.658, Expected=1028
>Predicted=1178.932, Expected=1332
>Predicted=1878.171, Expected=1416
>Predicted=1502.734, Expected=1673
>Predicted=1114.228, Expected=1661
>Predicted=1510.979, Expected=1521
>Predicted=1511.450, Expected=1442
>Predicted=1404.321, Expected=1096
>Predicted=1396.674, Expected=1099
>Predicted=1465.183, Expected=982
>Predicted=1371.782, Expected=1140
>Predicted=1178.517, Expected=1471
>Predicted=1250.589, Expected=1369
>Predicted=1048.953, Expected=1384
>Predicted=1291.395, Expected=1349
>Predicted=1354.167, Expected=1141
```

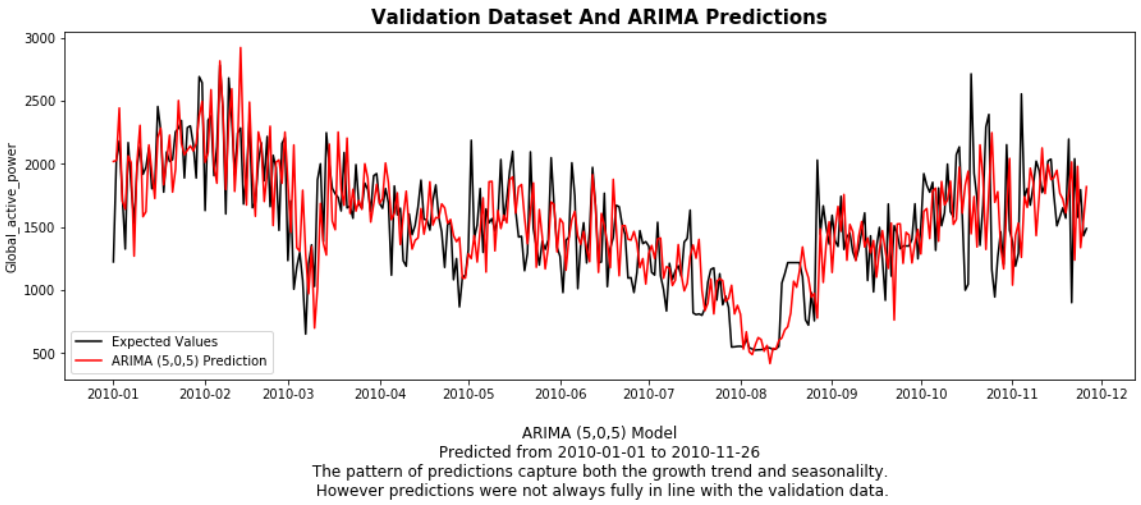
```
>Predicted=1251.153, Expected=1120
>Predicted=1406.053, Expected=1538
>Predicted=1413.078, Expected=1114
>Predicted=1094.838, Expected=1006
>Predicted=1186.106, Expected=835
>Predicted=1174.663, Expected=1213
>Predicted=1036.970, Expected=1083
>Predicted=1080.314, Expected=1155
>Predicted=1361.437, Expected=1194
>Predicted=1123.979, Expected=1110
>Predicted=994.145, Expected=1382
>Predicted=1053.027, Expected=1409
>Predicted=1269.156, Expected=1634
>Predicted=1357.688, Expected=820
>Predicted=1253.094, Expected=806
>Predicted=1402.581, Expected=812
>Predicted=997.807, Expected=801
>Predicted=838.403, Expected=856
>Predicted=898.326, Expected=1064
>Predicted=1088.958, Expected=1164
>Predicted=811.875, Expected=1176
>Predicted=1069.916, Expected=922
>Predicted=1091.743, Expected=1130
>Predicted=1075.366, Expected=884
>Predicted=911.906, Expected=952
>Predicted=934.860, Expected=863
>Predicted=1038.774, Expected=549
>Predicted=811.485, Expected=551
>Predicted=879.789, Expected=555
>Predicted=810.204, Expected=556
>Predicted=533.791, Expected=549
>Predicted=669.992, Expected=615
>Predicted=511.150, Expected=543
>Predicted=489.886, Expected=533
>Predicted=564.519, Expected=525
>Predicted=624.439, Expected=527
>Predicted=608.848, Expected=528
>Predicted=516.243, Expected=535
>Predicted=562.414, Expected=545
>Predicted=418.876, Expected=542
>Predicted=539.544, Expected=530
>Predicted=533.193, Expected=534
>Predicted=601.400, Expected=554
>Predicted=621.578, Expected=1056
>Predicted=684.927, Expected=1130
>Predicted=710.876, Expected=1218
>Predicted=818.706, Expected=1218
>Predicted=1070.828, Expected=1218
>Predicted=1023.546, Expected=1218
>Predicted=1155.038, Expected=1218
>Predicted=1342.584, Expected=1105
>Predicted=1171.592, Expected=766
>Predicted=1097.166, Expected=723
>Predicted=943.277, Expected=980
>Predicted=974.604, Expected=757
>Predicted=778.538, Expected=2031
>Predicted=1485.246, Expected=1443
>Predicted=1061.654, Expected=1668
>Predicted=1369.139, Expected=1495
>Predicted=1541.655, Expected=1365
>Predicted=1140.212, Expected=1593
```

```
>Predicted=1483.658, Expected=1392
>Predicted=1663.774, Expected=1346
>Predicted=1466.440, Expected=1744
>Predicted=1758.325, Expected=1323
>Predicted=1236.782, Expected=1423
>Predicted=1521.109, Expected=1452
>Predicted=1461.887, Expected=1304
>Predicted=1234.236, Expected=1239
>Predicted=1425.959, Expected=1328
>Predicted=1543.841, Expected=1463
>Predicted=1342.225, Expected=1612
>Predicted=1414.146, Expected=1076
>Predicted=1294.901, Expected=1431
>Predicted=1393.611, Expected=986
>Predicted=1103.092, Expected=1340
>Predicted=1323.396, Expected=1498
>Predicted=1471.153, Expected=1248
>Predicted=1309.421, Expected=920
>Predicted=1172.350, Expected=1683
>Predicted=1531.542, Expected=1111
>Predicted=763.103, Expected=1510
>Predicted=1524.077, Expected=1450
>Predicted=1526.058, Expected=1328
>Predicted=1212.446, Expected=1350
>Predicted=1458.266, Expected=1350
>Predicted=1436.842, Expected=1350
>Predicted=1215.746, Expected=1404
>Predicted=1373.005, Expected=1684
>Predicted=1480.848, Expected=1248
>Predicted=1286.541, Expected=1505
>Predicted=1624.131, Expected=1923
>Predicted=1647.061, Expected=1831
>Predicted=1408.717, Expected=1776
>Predicted=1784.227, Expected=1855
>Predicted=1808.644, Expected=1315
>Predicted=1387.873, Expected=1810
>Predicted=1861.265, Expected=1510
>Predicted=1675.178, Expected=1612
>Predicted=1714.168, Expected=1999
>Predicted=1863.914, Expected=1623
>Predicted=1520.612, Expected=1580
>Predicted=1560.965, Expected=2074
>Predicted=1974.645, Expected=2135
>Predicted=1610.070, Expected=1596
>Predicted=1832.027, Expected=1000
>Predicted=1942.634, Expected=1047
>Predicted=1446.605, Expected=2714
>Predicted=1741.003, Expected=1926
>Predicted=1342.177, Expected=1728
>Predicted=2151.382, Expected=1355
>Predicted=1900.338, Expected=1669
>Predicted=1322.760, Expected=2293
>Predicted=1775.917, Expected=2392
>Predicted=2247.688, Expected=1163
>Predicted=1696.284, Expected=946
>Predicted=1780.414, Expected=1260
>Predicted=1341.683, Expected=1463
>Predicted=1169.117, Expected=1204
>Predicted=1637.540, Expected=2153
>Predicted=2044.022, Expected=1476
>Predicted=1040.146, Expected=1383
```

```
>Predicted=1434.625, Expected=1191
>Predicted=1531.056, Expected=1293
>Predicted=1260.278, Expected=2555
>Predicted=1740.383, Expected=1736
>Predicted=1655.669, Expected=1805
>Predicted=1966.497, Expected=1674
>Predicted=1859.316, Expected=1826
>Predicted=1432.449, Expected=2022
>Predicted=1742.307, Expected=1947
>Predicted=2126.775, Expected=1774
>Predicted=1765.349, Expected=1837
>Predicted=1994.593, Expected=2023
>Predicted=1875.407, Expected=2038
>Predicted=1892.811, Expected=1747
>Predicted=1950.269, Expected=1510
>Predicted=1767.539, Expected=1582
>Predicted=1720.419, Expected=1652
>Predicted=1611.170, Expected=1570
>Predicted=1762.917, Expected=2197
>Predicted=2015.744, Expected=901
>Predicted=1238.415, Expected=2042
>Predicted=1980.916, Expected=1578
>Predicted=1336.379, Expected=1796
>Predicted=1559.763, Expected=1431
>Predicted=1819.605, Expected=1488
RMSE: 352.803
```

In [332]:

```
predictions_evaluate1=pd.DataFrame(predictions_evaluate, index=validation['Date'])
y_evaluate1=pd.DataFrame(y_evaluate, index=validation["Date"])

plt.figure(figsize=(15, 5))
plt.plot(y_evaluate1, color='black', label = 'Expected Values')
plt.plot(predictions_evaluate1, color='red', label='ARIMA (5,0,5) Prediction')
plt.title("Validation Dataset And ARIMA Predictions", fontsize=15, fontweight='bold')
plt.xlabel('\n ARIMA (5,0,5) Model \n Predicted from 2010-01-01 to 2010-11-26 \n The pa
ttern of predictions capture both the growth trend and seasonalilty. \n However predict
ions were not always fully in line with the validation data.', fontsize=12)
plt.ylabel('Global_active_power')
plt.legend()
plt.show()
```



LINEAR MODELS

From the data analysis, we can find that there is a trend over weeks, so we choose 7 days to get the difference. And the stationary is proved by the p-value (0.0000).

Then we get p=5 and q=5, from ACF and PACF plots of time series. We try to find out the best model by Grid Search. The result is the best ARIMA model is (5,0,5) with RMSE = 413.251. After applying the best model on the validation dataset, we get RMSE = 352.803. This indicates a much better performance of the ARIMA (5,0,5) model on the validation dataset. The ARIMA (5,0,5) model outperforms the Naïve Model by producing a much smaller RMSE (413.251 vs 477.762).

Finally, the plot of validation dataset and ARIMA (5,0,5) predictions shows ARIMA is a meaningful linear model with insignificant errors.

---

6. Supervised Learning Formulation

In this section, time series forecasting is framed as a supervised learning problem. This re-framing of the time series data gives the access to the suite of standard linear and nonlinear machine learning algorithms on the problem.

In [110]:

```
df = pd.read_csv('household_power_consumption_days.csv', index_col=False, header=0)
df['Date'] = pd.to_datetime(df['Date'].astype(str))
GAP2_series = pd.Series(df['Global_active_power'].values , index=df['Date'])

def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)

days_in_week = 7
diff = pd.DataFrame(difference(GAP2_series, days_in_week))

# Creating lag features
dataframe = pd.concat([diff.shift(3), diff.shift(2), diff.shift(1), diff], axis=1)
dataframe.columns = ['t-2', 't-1', 't', 't+1']
print('The number of rows in the dataframe:',len(dataframe))
dataframe.head()
```

The number of rows in the dataframe: 1435

Out[110]:

	t-2	t-1	t	t+1
0	NaN	NaN	NaN	3564.210
1	NaN	NaN	3564.210	-840.448
2	NaN	3564.210	-840.448	539.294
3	3564.210	-840.448	539.294	2267.916
4	-840.448	539.294	2267.916	-696.988

In [111]:

```
# Split feature and target arrays
XX = dataframe.values[3:,0:-1] #features
yy = dataframe.values[3:,-1] #target
```

In [116]:

```
# Train test split for both feature data and target data
split_point = int(len(GAP2_series) * 0.50) - 3 - 7
XX_train = XX[0:split_point]
XX_test = XX[split_point:]
yy_train = yy[0:split_point]
yy_test = yy[split_point:]
```

In [118]:

```
print('Size of XX_train: ', len(XX_train))
print('Size of yy_train: ', len(yy_train))
print("")
print('Size of XX_test: ', len(XX_test))
print('Size of yy_test: ', len(yy_test))
```

```
Size of XX_train:  711
Size of yy_train:  711

Size of XX_test:  721
Size of yy_test:  721
```

SUPERVISED LEARNING

To apply supervised learning model on this dataset, the sliding window method is adopted and we created the lag features by using diff. shift (), getting four features, which are t-2, t-1, t and t+1. The label feature **(yy)** is t+1, and this will be predicted by the other three features **(XX)**.

We define the training and test sets in a way such that the 1st element in the supervised learning test set corresponds to the 1st element in the previous time series test set.

We compare the sizes of train and test sets with what we had before, and check that the size of the test set is the same.

7. Supervised Learning And Ensemble Models

In this section, 3 different machine learning models are tested on the dataset and their validation RMSEs are reported. Moreover, ensemble techniques are applied on the best model.

1. LinearRegression
2. RandomForestRegressor
3. Gradient Boosting Regressor
4. Ensemble models - Linear Regression with Bagging

In [150]:

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

lr = LinearRegression()
rf = RandomForestRegressor(random_state=42)
GBR = GradientBoostingRegressor(random_state=42)

GBR.fit(XX_train, yy_train)
lr.fit(XX_train, yy_train)
rf.fit(XX_train, yy_train)
```

Out[150]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Linear Regression

In [127]:

```
# LR walk-forward validation

# Load data
df_LR = pd.read_csv('household_power_consumption_days_validation.csv', index_col=False,
header=0)
df_LR['Date'] = pd.to_datetime(df_LR['Date'].astype(str))
prediction_LR  = pd.Series(df_LR['Global_active_power'].values , index=df_LR['Date'])

validation_LR = prediction_LR.values.astype('float32')
train_LR = GAP_series.values.astype('float32')

history_LR = [x for x in train_LR]
predictions_LR = list()
for i in range(len(validation_LR)):
    yhat = lr.predict(XX_test[i,:].reshape(1, -1))[0]
    yhat = inverse_difference(history_LR, yhat, days_in_week)
    predictions_LR.append(yhat)

    # observation
    obs = validation_LR[i]
    history_LR.append(obs)

    prediction_LR[i]=yhat

rmse_LR = sqrt(mean_squared_error(validation_LR, predictions_LR))
print('LR RMSE: %.3f' % rmse_LR)
```

```
LR RMSE: 513.998
```

Random Forest Regressor

In [125]:

```
# RF walk-forward validation

# load data
df_FR = pd.read_csv('household_power_consumption_days_validation.csv', index_col=False,
header=0)
df_FR['Date'] = pd.to_datetime(df_FR['Date'].astype(str))
prediction_FR  = pd.Series(df_FR['Global_active_power'].values , index=df_FR['Date'])

validation_FR = prediction_FR.values.astype('float32')
train_FR = GAP_series.values.astype('float32')

history_FR = [x for x in train_FR]
predictions_FR = list()
for i in range(len(validation_FR)):
    yhat = rf.predict(XX_test[i,:].reshape(1, -1))[0]
    yhat = inverse_difference(history_FR, yhat, days_in_week)
    predictions_FR.append(yhat)

    # observation
    obs = validation_FR[i]
    history_FR.append(obs)

    prediction_FR[i]=yhat

rmse_FR = sqrt(mean_squared_error(validation_FR, predictions_FR))
print('FR RMSE: %.3f' % rmse_FR)
```

```
FR RMSE: 573.099
```

Gradient Boosting Regressor

In [123]:

```
# GBR walk-forward validation

# load data
df_GBR = pd.read_csv('household_power_consumption_days_validation.csv', index_col=False
, header=0)
df_GBR['Date'] = pd.to_datetime(df_GBR['Date'].astype(str))
prediction_GBR = pd.Series(df_GBR['Global_active_power'].values , index=df_GBR['Date'])

validation_GBR = prediction_GBR.values.astype('float32')
train_GBR = GAP_series.values.astype('float32')

history_GBR = [x for x in train_GBR]
predictions_GBR = list()
for i in range(len(validation_GBR)):
    yhat = GBR.predict(XX_test[i,:].reshape(1, -1))[0]
    yhat = inverse_difference(history_GBR, yhat, days_in_week)
    predictions_GBR.append(yhat)

    # observation
    obs = validation_GBR[i]
    history_GBR.append(obs)

    prediction_GBR[i]=yhat

rmse_GBR = sqrt(mean_squared_error(validation_GBR, predictions_GBR))
print('GBR RMSE: %.3f' % rmse_GBR)
```

```
GBR RMSE: 537.319
```

In [130]:

```
print('LR RMSE: %.3f' % rmse_LR)
print('FR RMSE: %.3f' % rmse_FR)
print('GBR RMSE: %.3f' % rmse_GBR)
print('\nLR model is the best one.')
```

```
LR RMSE: 513.998
FR RMSE: 573.099
GBR RMSE: 537.319

LR model is the best one.
```

Ensemble models - Linear Regression with Bagging

In [131]:

```
# Ensemble models--Bagging
from sklearn.ensemble import BaggingRegressor

# Instantiate bc
bc = BaggingRegressor(base_estimator=lr, n_estimators=250, random_state=42)

# Fit bc to the training set
bc.fit(XX_train, yy_train)

# Load data
df_bc = pd.read_csv('household_power_consumption_days_validation.csv', index_col=False,
header=0)
df_bc['Date'] = pd.to_datetime(df_bc['Date'].astype(str))
prediction_bc = pd.Series(df_bc['Global_active_power'].values , index=df_bc['Date'])

validation_bc = prediction_bc.values.astype('float32')
train_bc = GAP_series.values.astype('float32')

history_bc = [x for x in train_bc]
predictions_bc = list()
for i in range(len(validation_bc)):
    yhat = bc.predict(XX_test[i,:].reshape(1, -1))[0]
    yhat = inverse_difference(history_bc, yhat, days_in_week)
    predictions_bc.append(yhat)

    # observation
    obs = validation_bc[i]
    history_bc.append(obs)
    prediction_bc[i]=yhat
rmse_bc = sqrt(mean_squared_error(validation_bc, predictions_bc))
print('Linear Regression with Bagging RMSE: %.3f' % rmse_bc)
```

```
Linear Regression with Bagging RMSE: 513.903
```

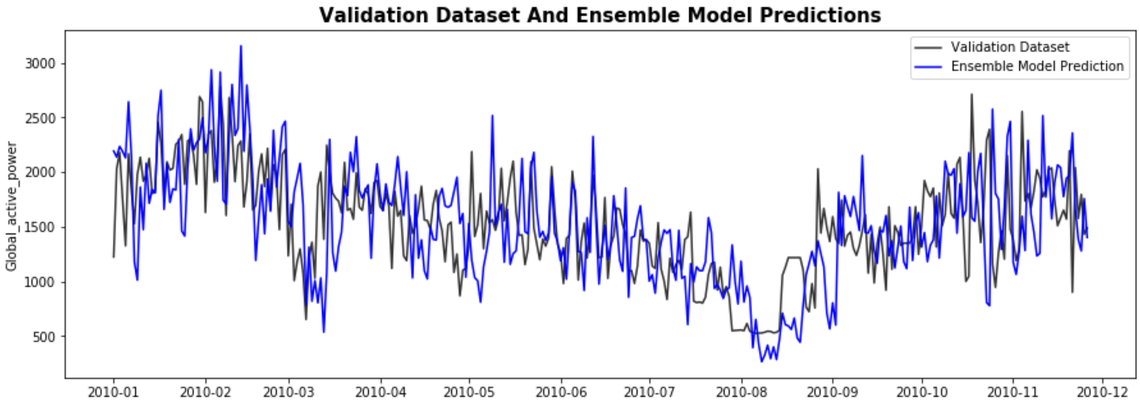
In [134]:

```
print('LR RMSE: %.3f' % rmse_LR)
print('FR RMSE: %.3f' % rmse_FR)
print('GBR RMSE: %.3f' % rmse_GBR)
print('Linear Regression with Bagging RMSE: %.3f' % rmse_bc)
print("")
print('Linear Regression with Bagging model is the best one.')
```

```
LR RMSE: 513.998
FR RMSE: 573.099
GBR RMSE: 537.319
Linear Regression with Bagging RMSE: 513.903

Linear Regression with Bagging model is the best one.
```

In [335]:

```
predictions_bc1=pd.DataFrame(predictions_bc, index=validation['Date'])

plt.figure(figsize=(15, 5))
plt.plot(y_evaluate1, alpha = 0.8, color='black', label = 'Validation Dataset')
plt.plot(predictions_bc1, color='blue', label='Ensemble Model Prediction')
plt.title("Validation Dataset And Ensemble Model Predictions", fontsize=15, fontweight=
'bold')
plt.ylabel('Global_active_power')
plt.legend()
plt.show()
```



ENSEMBLE MODELS

The selected models for supervised learning are Linear Regressor, Random Forest Regressor and Gradient Boosting Regressor. Model spot check indicates the best model is Linear Regressor, thus the ensemble technique Bagging is applied on Linear Regression to obtain the optimal RMSE of 513.903.

From the line plot above, it can be concluded that the Ensemble Model forecasts similar trend as the validation dataset presents.

8. Advanced Methods

In this section, we are applying a recurrent neural network on our dataset. Specifically, we are applying Long Short Term Memory networks, called "LSTM", which is special type of RNN. LSTMs are capable of learning long-term dependencies and they are suitable for time-series problems.

In [310]:

```
#define a function which helps us to define a supervised learning problem which predict
s the global active power
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    dff = pd.DataFrame(data)
    cols, names = list(), list()
    for i in range(n_in, 0, -1): # for loop for our input sequence is t-n,...t-1
        cols.append(dff.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    for i in range(0, n_out): # for loop to forecast sequence (t, t+1, ...t+n)
        cols.append(dff.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    agg = pd.concat(cols, axis=1) # aggregate them
    agg.columns = names
    if dropnan: # drop rows with missing values
        agg.dropna(inplace=True)
    return agg
```

In [311]:

```
# Use the household power consumption csv and analyze the shape
df_daily = pd.read_csv('household_power_consumption_days.csv', index_col=False, header=
0)
df_daily.shape
```

Out[311]:

```
(1442, 9)
```

In [312]:

```
# only use the following columns for the prediction
df_daily=df_daily[['Global_active_power',
                   'Global_reactive_power',
                   'Voltage',
                   'Global_intensity',
                   'Sub_metering_1',
                   'Sub_metering_2',
                   'Sub_metering_3']]
```

In [313]:

```
# normalize all features to make them more comparable
from sklearn.preprocessing import MinMaxScaler
values = df_daily.values

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame it as a supervised learning problem
reframed = series_to_supervised(scaled, 1, 1)

#drop columns do not want to use for prediction
reframed.drop(reframed.columns[[8,9,10,11,12,13]], axis=1, inplace=True)
reframed.head()
```

Out[313]:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var1(t)
1	0.211996	0.000000	0.000000	0.211006	0.000000	0.045090	0.162013	0.694252
2	0.694252	0.499028	0.959730	0.695226	0.181875	0.345776	0.536762	0.431901
3	0.431901	0.331329	0.966003	0.424618	0.095098	0.216451	0.566912	0.313037
4	0.313037	0.302994	0.970210	0.311508	0.075058	0.627798	0.218615	0.436748
5	0.436748	0.329256	0.971902	0.428075	0.000000	0.218680	0.568916	0.325660

In [314]:

```
#reshape training and test data to be able to process the data to the LSTM
values = reframed.values

# use a timeframe of three years (365*3)
n_train_time = 365*3
train_LSTM = values[:n_train_time, :]
test_LSTM = values[n_train_time:, :]

# split into input and outputs
train_LSTM_X, train_LSTM_y = train_LSTM[:, :-1], train_LSTM[:, -1]
test_LSTM_X, test_LSTM_y = test_LSTM[:, :-1], test_LSTM[:, -1]

# reshape data into a 3D format (samples, timesteps, features)
train_LSTM_X = train_LSTM_X.reshape((train_LSTM_X.shape[0], 1, train_LSTM_X.shape[1]))
test_LSTM_X = test_LSTM_X.reshape((test_LSTM_X.shape[0], 1, test_LSTM_X.shape[1]))
print(train_LSTM_X.shape, train_LSTM_y.shape, test_LSTM_X.shape, test_LSTM_y.shape)
```

```
(1095, 1, 7) (1095,) (346, 1, 7) (346,)
```

In [315]:

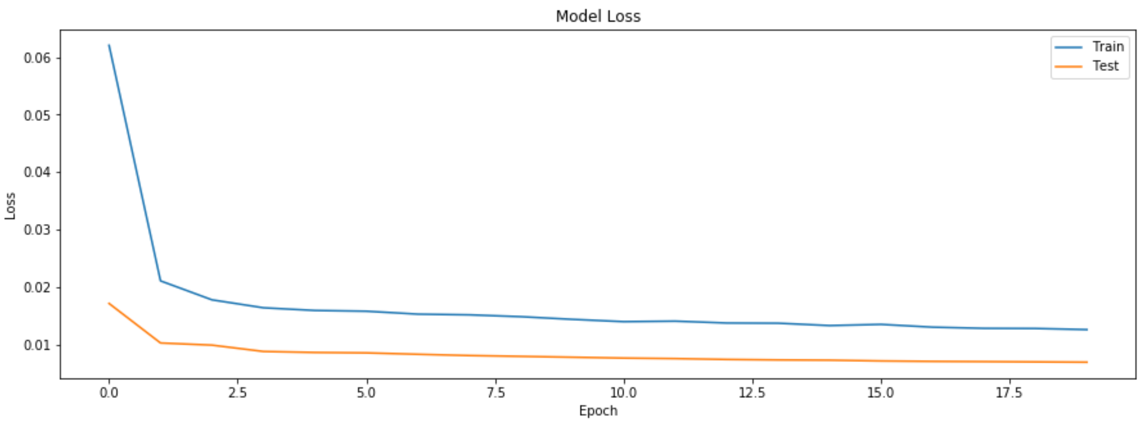
```
#install keras
!pip install keras
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
import itertools
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import Dropout
```

```
Requirement already satisfied: keras in /opt/anaconda/envs/Python3/lib/pyt
hon3.6/site-packages (2.2.4)
Requirement already satisfied: numpy>=1.9.1 in /opt/anaconda/envs/Python3/
lib/python3.6/site-packages (from keras) (1.15.4)
Requirement already satisfied: h5py in /opt/anaconda/envs/Python3/lib/pyth
on3.6/site-packages (from keras) (2.8.0)
Requirement already satisfied: scipy>=0.14 in /opt/anaconda/envs/Python3/l
ib/python3.6/site-packages (from keras) (1.1.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /opt/anacond
a/envs/Python3/lib/python3.6/site-packages (from keras) (1.0.9)
Requirement already satisfied: keras-applications>=1.0.6 in /opt/anaconda/
envs/Python3/lib/python3.6/site-packages (from keras) (1.0.7)
Requirement already satisfied: pyyaml in /opt/anaconda/envs/Python3/lib/py
thon3.6/site-packages (from keras) (3.13)
Requirement already satisfied: six>=1.9.0 in /opt/anaconda/envs/Python3/li
b/python3.6/site-packages (from keras) (1.12.0)
```

In [316]:

```
# create a sequential model
model = Sequential()

# LSTM model with 100 neurons in the first visible layer
model.add(LSTM(100, input_shape=(train_LSTM_X.shape[1], train_LSTM_X.shape[2])))

# dropout of 20%
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

# fit model with 20 training epochs and a batch size
history = model.fit(train_LSTM_X, train_LSTM_y, epochs=20, batch_size=70, validation_da
ta=(test_LSTM_X, test_LSTM_y),
                    verbose=2, shuffle=False)

# summarize history for loss
plt.figure(figsize=(15, 5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

# predict y hat
yhat = model.predict(test_LSTM_X)
test_LSTM_X = test_LSTM_X.reshape((test_LSTM_X.shape[0], 7))

# invert the scaling for forecast
inv_yhat = np.concatenate((yhat, test_LSTM_X[:, -6:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_LSTM_y = test_LSTM_y.reshape((len(test_LSTM_y), 1))
inv_y = np.concatenate((test_LSTM_y, test_LSTM_X[:, -6:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

# calculate the rmse to be able to compare it with the performance of the other models
rmse_LSTM = sqrt(mean_squared_error(inv_y, inv_yhat))
print('LSTM RMSE: %.3f' % rmse_LSTM)
```

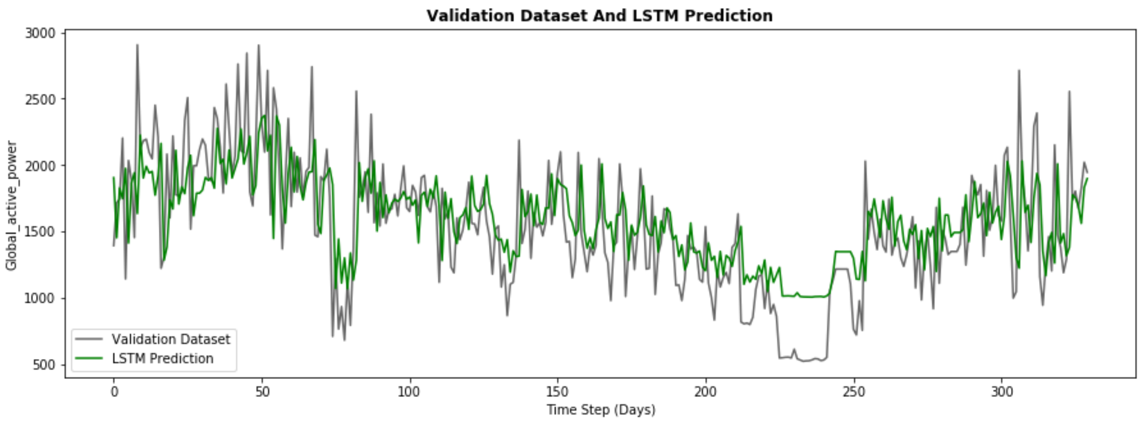
```
Train on 1095 samples, validate on 346 samples
Epoch 1/20
 - 3s - loss: 0.0620 - val_loss: 0.0172
Epoch 2/20
 - 0s - loss: 0.0211 - val_loss: 0.0103
Epoch 3/20
 - 0s - loss: 0.0178 - val_loss: 0.0099
Epoch 4/20
 - 0s - loss: 0.0164 - val_loss: 0.0088
Epoch 5/20
 - 0s - loss: 0.0159 - val_loss: 0.0086
Epoch 6/20
 - 0s - loss: 0.0158 - val_loss: 0.0086
Epoch 7/20
 - 0s - loss: 0.0153 - val_loss: 0.0083
Epoch 8/20
 - 0s - loss: 0.0152 - val_loss: 0.0081
Epoch 9/20
 - 0s - loss: 0.0149 - val_loss: 0.0080
Epoch 10/20
 - 0s - loss: 0.0144 - val_loss: 0.0078
Epoch 11/20
 - 0s - loss: 0.0140 - val_loss: 0.0077
Epoch 12/20
 - 0s - loss: 0.0141 - val_loss: 0.0076
Epoch 13/20
 - 0s - loss: 0.0138 - val_loss: 0.0074
Epoch 14/20
 - 0s - loss: 0.0137 - val_loss: 0.0073
Epoch 15/20
 - 0s - loss: 0.0133 - val_loss: 0.0073
Epoch 16/20
 - 0s - loss: 0.0135 - val_loss: 0.0072
Epoch 17/20
 - 0s - loss: 0.0130 - val_loss: 0.0071
Epoch 18/20
 - 0s - loss: 0.0128 - val_loss: 0.0070
Epoch 19/20
 - 0s - loss: 0.0128 - val_loss: 0.0070
Epoch 20/20
 - 0s - loss: 0.0126 - val_loss: 0.0069
```




```
LSTM RMSE: 376.871
```

In [340]:

```
## time steps, every step is one day
## only compare the predictions in 330 days

plt.figure(figsize=(15, 5))
aa=[x for x in range(330)]
plt.plot(aa, inv_y[:330], alpha=0.6, color='black', label="Validation Dataset")
plt.plot(aa, inv_yhat[:330], color='green', label="LSTM Prediction")
plt.ylabel('Global_active_power')
plt.xlabel('Time Step (Days)')
plt.title('Validation Dataset And LSTM Prediction', fontweight='bold')
plt.legend()
plt.show()
```



OPTIONAL ASSESSMENT - ADVANCED MODELS

The **series_to_supervised** function helps us to define a supervised learning problem which predicts the global active power at the current time (t) using the global active power measurement and other features at the prior time step.

We are only using seven columns from the **df_daily** dataframe. We then normalize the data and drop the columns we do not want to predict. The data is splitted into train and test data and reshaped into a 3D format (samples, timesteps, features) making it possible to be processed by LSTMs.

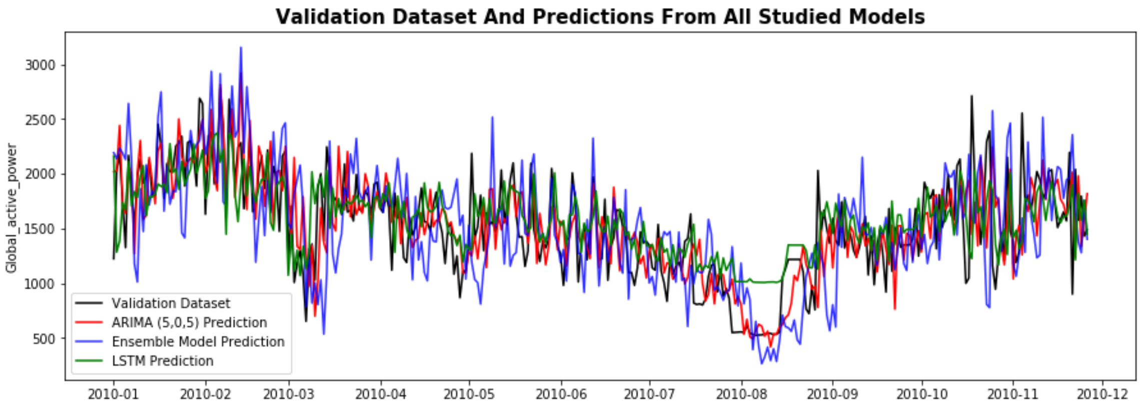
Our LSTM model shall have 100 neurons in the first visible layer, a dropout of 20%. The input shall shall be a one time step with seven features. We define one neuron in the output layer to predict the global active power. The model will be fitted for 20 training epochs with a batch size of 70.

Using a Long Short Term Memory network (LSTM) model, which is a special Recurrent Neural Network model, we achieved a RMSE score of 376.871.

9. Evaluation

In [339]:

```
plt.figure(figsize=(15, 5))
plt.plot(y_evaluate1, color='black', label = 'Validation Dataset')
plt.plot(predictions_evaluate1, color='red', label='ARIMA (5,0,5) Prediction')
plt.plot(predictions_bc1, alpha = 0.8, color='blue', label='Ensemble Model Prediction')
plt.plot(validation['Date'], inv_yhat[16:], color='green', label="LSTM Prediction")
plt.title("Validation Dataset And Predictions From All Studied Models", fontsize=15, fo
ntweight='bold')
plt.ylabel('Global_active_power')
plt.legend()
plt.show()
```



In [328]:

```
print('Naive RMSE: %.3f' % rmse_Naive)
print('ARIMA (5,0,5) RMSE: %.3f' % rmse_evaluate)
print('Linear Regression with Bagging RMSE: %.3f' % rmse_bc)
print('LSTM RMSE: %.3f' % rmse_LSTM)
print("")
print('ARIMA (5,0,5) is the best one.')
```

```
Naive RMSE: 477.762
ARIMA (5,0,5) RMSE: 352.803
Linear Regression with Bagging RMSE: 513.903
LSTM RMSE: 376.871

ARIMA (5,0,5) is the best one.
```

EVALUATION

The main motivation of this project was to define a time series forecasting problem using the "UCI Individual household electric power consumption" dataset. We therefore predicted the total active power consumption for one day ahead. We resampled the dataset to daily observations and trained several different regressors using different methods and tested their performance.

A naïve forecast model, where we obtained a RMSE score of 477.762, provided a baseline performance by which more sophisticated models were evaluated. We then trained an ARIMA model which was automatically configured by GridSearch. Our best ARIMA model (5,0,5) outperformed the naïve forecast model with a RMSE score of 352.803. We applied different supervised learning models (linear regression, random forest, gradient boosting regression) and combined the linear regression model with bagging. From all supervised learning models, we achieved the best RMSE score from the linear regressor combined with bagging (RMSE score validation= 513.903). We applied Long Short Term Memory networks (LSTM), a special type of recurrent neural network, where we obtained a RMSE score of 376.871.

Comparing all trained models, we can see that the ARIMA (5,0,5) model clearly outperformed all other models.

Credits:

Individual household electric power consumption Data Set

http://rstudio-pubs-static.s3.amazonaws.com/239446_1bfac3dbda4b45e9baf1c282791f7664.html (http://rstudio-pubs-static.s3.amazonaws.com/239446_1bfac3dbda4b45e9baf1c282791f7664.html)

Time-series data analysis using LSTM (Tutorial)

https://www.kaggle.com/amirrezaeian/time-series-data-analysis-using-lstm-tutorial (https://www.kaggle.com/amirrezaeian/time-series-data-analysis-using-lstm-tutorial)

Individual Household Electric Power Consumption Analysis

http://rstudio-pubs-static.s3.amazonaws.com/239446_1bfac3dbda4b45e9baf1c282791f7664.html (http://rstudio-pubs-static.s3.amazonaws.com/239446_1bfac3dbda4b45e9baf1c282791f7664.html)

Machine Learning Mastery

https://machinelearningmastery.com/how-to-load-and-explore-household-electricity-usage-data/ (https://machinelearningmastery.com/how-to-load-and-explore-household-electricity-usage-data/)

https://machinelearningmastery.com/multi-step-time-series-forecasting-with-machine-learning-models-for-household-electricity-consumption/ (https://machinelearningmastery.com/multi-step-time-series-forecasting-with-machine-learning-models-for-household-electricity-consumption/)

https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/ (https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/)