Github link : https://github.com/sstorey-nephila/w207_final_project/tree/master (https://github.com/sstorey-nephila/w207_final_project/tree/master)

Temperature Prediction Using Multivariate Time Series Analysis

The objective of this paper is to explain our methodology used to predict the temperature for a range of locations along the coast of mainland USA by applying machine learning with time series analysis. We begin by explaining the different features used today to identify temporal trends and patterns based on data from the “National Oceanic and Atmospheric Administration” (NOAA). We then move on to identify correlations across multiple time series that will be used to generate predicted values (weather forecast). We will explore the stack used to run our solution. Finally, we will implement a number of supervised methods and compare them to traditional techniques using “Autoregressive Integrated Moving Average Models” (ARIMA).

National Oceanic and Atmospheric Administration

“NOAA's National Centers for Environmental Information (NCEI) is responsible for preserving, monitoring, assessing, and providing public access to the Nation's treasure of climate and historical weather data and information” across the United States. it “hosts and provides access to one of the most significant archives on earth, with comprehensive oceanic, atmospheric, and geophysical data. From the depths of the ocean to the surface of the sun and from million-year-old ice core records to near-real-time satellite images, NCEI is the Nation’s leading authority for environmental information.”

Goals of this project

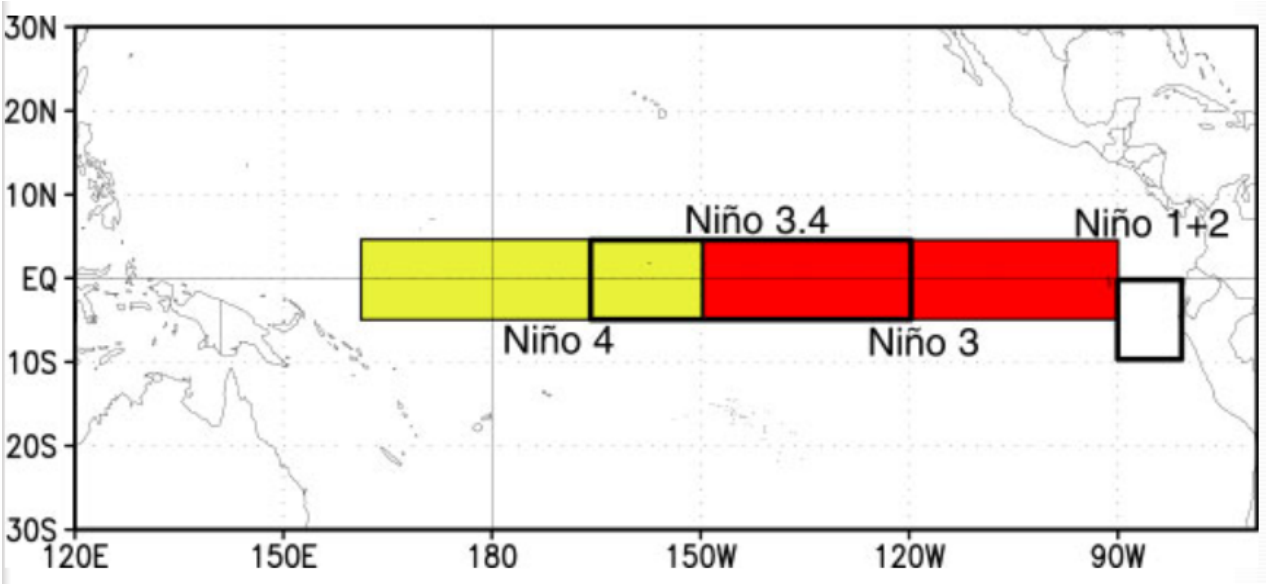
We will be attempting to

- Use teleconnections to help improve our temperature predictions across various locations in the USA
- Use feature engineering to extract key signals for our predictions
- Use a variety of machine learning techniques
- Automatically capture and log data about our experiments to aid repeatibility

Background

Teleconnections in atmospheric science refer to climate anomalies that are related to each other at large distances (typically thousands of kilometers).

A prominent aspect of our weather and climate is its variability. This variability ranges over many time and space scales such as localized thunderstorms and tornadoes, to larger-scale storms, to droughts, to multi-year, multi-decade and even multi-century time scales.



This diagram shows the 4 key zones associated with the determination of if we are in an El Niño or La Nina phase

Some examples of this longer time-scale variability might include a series of abnormally mild or exceptionally severe winters, and even a mild winter followed by a severe winter. Such year-to-year variations in the weather patterns are often associated with changes in the wind, air pressure, storm tracks, and jet streams that encompass areas far larger than that of your particular region. At times, the year-to-year changes in weather patterns are linked to specific weather, temperature and rainfall patterns occurring throughout the world due to the naturally occurring phenomena known as El Niño.

Teleconnections & Weather Prediction

Teleconnection patterns are large-scale changes due to atmospheric waves that influence weather and temperatures as explained in the section above. In our project, we will be using sea surface temperature and 4 main teleconnection patterns from ENSO; AO (Arctic Oscillation), NAO (North Atlantic Oscillation), PNA (Pacific-North American Pattern), AAO (Antarctic Oscillation).

Data Sources

NOAA download tool https://www.ncdc.noaa.gov/cdo-web/datatools (https://www.ncdc.noaa.gov/cdo-web/datatools) https://www.ncdc.noaa.gov/cdo-web/datasets (https://www.ncdc.noaa.gov/cdo-web/datasets)

Data from 2000(by hour) https://www.ncdc.noaa.gov/crn/qcdatasets.html (https://www.ncdc.noaa.gov/crn/qcdatasets.html) ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/hourly02

Daily from 1750(by day) ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/ ftp://ftp.cpc.ncep.noaa.gov/cwlinks/ ftp://ftp.ncdc.noaa.gov/pub/data/normals/1981-2010/source-datasets/

Weather station index ftp://ftp.ncdc.noaa.gov/pub/data/noaa/isd-inventory.txt https://www1.ncdc.noaa.gov/pub/data/ish/country-list.txt (https://www1.ncdc.noaa.gov/pub/data/ish/country-list.txt)

Google BigQuery https://cloud.google.com/bigquery/public-data/noaa-ghcn (https://cloud.google.com/bigquery/public-data/noaa-ghcn)

Hourly data ftp://ftp.ncdc.noaa.gov/pub/data/noaa/isd-lite

Monthly sst for el nino https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/Nino12/ (https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/Nino12/)

AMO https://www.esrl.noaa.gov/psd/data/timeseries/AMO/ (https://www.esrl.noaa.gov/psd/data/timeseries/AMO/)

NINA34 - Anomaly from 1981-2010 https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/Nino34/ (https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/Nino34/)

References used

Information on teleconnections http://www.cpc.ncep.noaa.gov/products/precip/CWlink/daily_ao_index/teleconnections.shtml (http://www.cpc.ncep.noaa.gov/products/precip/CWlink/daily_ao_index/teleconnections.shtml)

Trick to use cosine/sine to encode seasonal components https://www.kaggle.com/avanwyk/encoding-cyclical-features-for-deep-learning (https://www.kaggle.com/avanwyk/encoding-cyclical-features-for-deep-learning)

Example using gated neural net https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/23_Time-Series-Prediction.ipynb (https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/23_Time-Series-Prediction.ipynb)

Feature engineering for time series https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/ (https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/)

In [2]:
```
import nbconvert
```

Data Formatting, Exploratory Data Analysis & Feature Engineering

Notebook Dependencies

We import the necessary library files for our notebook and set the **PATHS** for data, pickle and image files using the helper function python file

In [3]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Introduction

This process cleans the import files and transforms them into a vertical representation that can be used for the timeseries analysis. We also pull data from BigQuery to use as one of the datafeeds for our modeling pipeline.

We use an IPython widget to track the output of our processes

In [2]:
```
out = widgets.Output(layout={'border': '1px solid black'})
```

Next we run two queries we have created on Google BigQuery to extract weather station data for all weather stations available on the US coast in this published dataset from NOAA.

In [ ]:
```
# !pip install --upgrade google-cloud-bigquery
# !pip install --upgrade google-api-python-client
# !pip install --upgrade pandas_gbq
```

Connect to bigQuery via CLI/GCloud and authenticate, set up credentials

In [ ]:
```
# electric-armor-191917
#
# gcloud iam service-accounts create sstoreybda-svcacct
# gcloud projects add-iam-policy-binding electric-armor-191917 --member "serviceAccount:sstoreybda-svcacct@electric-armor-191917.iam.gserviceacc
# gcloud iam service-accounts keys create service-account.json --iam-account sstoreybda-svcacct@electric-armor-191917.iam.gserviceaccount.com
```

Connect to bigQuery using service credentials created previously

In [20]:
```
import pandas_gbq as g

project_id = 'electric-armor-191917'
verbose = False
dialect='standard'
timeout=30
configuration = {
'query': {
"useQueryCache": False
}
}
os.getcwd()

from google.cloud import bigquery

# Explicitly use service account credentials
client = bigquery.Client.from_service_account_json('../electric-armor-191917.json')

sql = ('SELECT CURRENT_DATETIME() as now')
query_job = client.query(sql)
iterator = query_job.result(timeout=timeout)
rows = list(iterator)

print(rows)
```

```
[Row((datetime.datetime(2018, 8, 14, 19, 50, 2, 417331),), {'now': 0})]
```

Extract weather data and save this to a csv

In [29]:
```
# Get weather data based on station ID and show the data available between 1950 and 2018

sql =""" SELECT s.id, s.name,
     min(FORMAT_DATE('%G',t.date)) as min_year,
     max(FORMAT_DATE('%G',t.date)) as max_year,
     count(t.id) as stn_count
   FROM `bigquery-public-data.ghcn_d.ghcnd_stations` s
   LEFT OUTER JOIN (
         SELECT * FROM `bigquery-public-data.ghcn_d.ghcnd_*`
         WHERE _TABLE_SUFFIX BETWEEN '1950' AND '2018'
     ) AS t ON t.id = s.id
 WHERE
   t.id IN ('USW00012919','USW00012924','USW00013970','USW00012912','USW00012960','USW00012917','USW00003937','USW00012916',
     'USW00013894','USW00013899','USW00093805','USW00013889','USW00012816','USW00012834','USW00012836','USW00012839',
     'USW00012844','USW00012835','USW00012842','USW00012843','USW00012815','USW00003822','USW00003820','USW00013880',
     'USW00013883','USW00013748','USW00013722', 'USW00093729','USW00013737','USW00013740','USW00093739','USW00093730',
     'USW00013739','USW00094789','USW00004781','USW00094702','USW00014765','USW00014739','USW00014740','USW00094746',
     'USW00014745','USW00014764','USW00014606','USW00093721') GROUP BY 1, 2"""

df = g.read_gbq(sql, project_id=project_id, verbose=verbose,configuration=configuration)
df.to_csv('../data/temperature_data.csv')
```

In [30]:
```
df.head(5)
```

Out[30]:

	id	name	min_year	max_year	stn_count
0	USW00093805	TALLAHASSEE	1949	2017	296704
1	USW00014740	HARTFORD BRADLEY INTL AP	1949	2017	303222
2	USW00093730	ATLANTIC CITY INTL AP	1949	2017	314872
3	USW00094702	BRIDGEPORT SIKORSKY MEM AP	1949	2017	291238
4	USW00013740	RICHMOND INTL AP	1949	2017	327933

Next we run this query to extract the details about each of these weather stations and save this to a csv

In [36]:
```
stations = ['USW00012919','USW00012924','USW00013970','USW00012912','USW00012960','USW00012917',
     'USW00003937','USW00012916','USW00013894','USW00013899','USW00093805','USW00013889','USW00012816',
     'USW00012834', 'USW00012836','USW00012839','USW00012844','USW00012835','USW00012842','USW00012843',
     'USW00012815','USW00003822','USW00003820','USW00013880''USW00013883','USW00013748','USW00013722',
     'USW00093729','USW00013737','USW00013740','USW00093739','USW00093730','USW00013739','USW00094789',
     'USW00004781','USW00094702','USW00014765','USW00014739','USW00014740','USW00094746','USW00014745',
     'USW00014764','USW00014606','USW00093721']

for station in stations:

    print(f'fetching weather info for {station}')

    sql = f"""SELECT t.*
     FROM `bigquery-public-data.ghcn_d.ghcnd_*` t
     WHERE
     _TABLE_SUFFIX BETWEEN '1950'
       AND '2018'
       AND t.id = '{station}' """

    df = g.read_gbq(sql, project_id=project_id, verbose=verbose,configuration=configuration)
    df.to_csv(f'../data/weather_stations_{station}.csv')
```

```
fetching weather info for USW00012836
fetching weather info for USW00012839
fetching weather info for USW00012844
fetching weather info for USW00012835
fetching weather info for USW00012842
fetching weather info for USW00012843
fetching weather info for USW00012815
fetching weather info for USW00003822
fetching weather info for USW00003820
fetching weather info for USW00013880
fetching weather info for USW00013883
fetching weather info for USW00013748
fetching weather info for USW00013722
fetching weather info for USW00093729
fetching weather info for USW00013737
fetching weather info for USW00013740
fetching weather info for USW00093739
fetching weather info for USW00093730
fetching weather info for USW00013739
fetching weather info for USW00094789
fetching weather info for USW00004781
fetching weather info for USW00094702
fetching weather info for USW00014765
fetching weather info for USW00014739
fetching weather info for USW00014740
fetching weather info for USW00094746
fetching weather info for USW00014745
fetching weather info for USW00014764
fetching weather info for USW00014606
fetching weather info for USW00093721
```

These files are stored using Git-LFS repository (setup instructions not included here).

We then create a function that handles parsing the daily and monthly files organized (without required melting)

In [5]:

```
class FileParser:

#     @out.capture()
    def MonthlyParser(name, filename, column, debug=True):
        """
        Monthly parser that strips datafiles from custom format and creates timeseries
        This module also performs feature engineering to create custom columns for our ML models
        """
        # read raw data file

        raw_df = pd.read_table(filename, header=None, sep=r'\s+')

        if (debug):
            print(f'Raw output - {name}')
            display(raw_df.head(5))

        # Lets unpivot(melt) this layout into a vertical representation

        source_df = pd.melt(raw_df, id_vars=[0], value_vars=[1,2,3,4,5,6,7,8,9,10,11,12])
        source_df.rename(columns = {0:'datetime','variable':'month'}, inplace = True)

        if (debug):
            print(f'Unmelted output - {name}')
            display(source_df.head(5))

        # Remove any entries before min_year (1900) and then concat columns together to create a column
        # that represents the end of the month date

        # filter out dates < 1900 and remove any values not provided ( -99.99 )

        source_df = source_df[source_df.datetime >= 1900 ]
        source_df = source_df[source_df.value != -99.99 ]

        # format and build up date

        source_df['month'] = source_df['month'].apply(lambda x: f'{x:0>2}')
        source_df['datetime'] = source_df['datetime'].astype(str)
        source_df['month'] = source_df['month'].astype(str)

        if (debug):
            print(f'Build up year/month - {name}')
            display(source_df.head(5))

        # Now shred the year/month and create datetime.
        # Move the date to the end of the month ( as values captured and averaged to the end of the month )

        source_df['datetime'] = source_df['datetime'] + '-' + source_df['month'] + '-01 00:00:00'
        source_df['datetime'] = source_df['datetime'].apply(lambda x: dt.datetime.strptime(x,'%Y-%m-%d %H:%M:%S')) + MonthEnd(1)
        source_df.drop('month', axis=1)

        if (debug):
            print(f'Post shredding year/month - {name}')
            display(source_df.head(5))
```

```
            display(source_df.tail(5))

        # Drop month column and set up dataframe index

        source_df = source_df.drop('month', axis=1)
        source_df = source_df.sort_values(by=['datetime'])
        source_df = source_df.set_index('datetime')

        # Next lets back fill, then subsequently remove any NAs and return results

        source_df = source_df.fillna(method = 'bfill', axis=0).dropna()

        if (debug):
            print(f'Final result - {name}')
            display(source_df.head(5))

        return source_df

#     @out.capture()
    def DailyParser(name,filename,column,debug):
        """
        Daily parser that strips datafiles from custom format and creates timeseries
        This module also performs feature engineering to create custom columns for our ML models
        """
        # read raw data file

        raw_df = pd.read_table(filename, header=None, sep=r'\s+')

        if (debug):
            print(f'Raw output - {name}')
            display(raw_df.head(5))

        source_df = raw_df.rename(columns = {0:'datetime',1:'month',2:'day',3:'value'})

        # Remove any entries before min_year (1900) and then concat columns together to create a column
        # that represents the end of the month date

        # filter out dates < 1900 and remove any values not provided ( -99.99 )

        source_df = source_df[source_df.datetime >= 1900 ]
        source_df = source_df[source_df.value != -99.99 ]

        # format data types (for date creation)

        source_df['day'] = source_df['day'].apply(lambda x: f'{x:0>2}')
        source_df['month'] = source_df['month'].apply(lambda x: f'{x:0>2}')
        source_df['day'] = source_df['day'].astype(str)
        source_df['month'] = source_df['month'].astype(str)
        source_df['datetime'] = source_df['datetime'].astype(str)

        if (debug):
            print(f'Build up year/month - {name}')
            display(source_df.head(5))
```

```
    # Now shred the year/month and create datetime.
    # Move the date to the end of the month ( as values captured and averaged to the end of the month )

    source_df['datetime'] = source_df['datetime'] + '-' + source_df['month'] + '-' + source_df['day'] + ' 00:00:00'
    source_df['datetime'] = source_df['datetime'].apply(lambda x: dt.datetime.strptime(x,'%Y-%m-%d %H:%M:%S'))

    source_df = source_df.drop('month', axis=1)
    source_df = source_df.drop('day', axis=1)

    if (debug):
        print(f'Post shredding year/month - {name}')
        display(source_df.head(5))
        display(source_df.tail(5))

    # Drop month column and set up dataframe index

    source_df = source_df.sort_values(by=['datetime'])
    source_df = source_df.set_index('datetime')

    # Next lets back fill, then subsequently remove any NAs and return results

    source_df = source_df.fillna(method = 'bfill', axis=0).dropna()

    if (debug):
        print(f'Final result - {name}')
        display(source_df.head(5))

    return source_df
```

Next, lets create a function that will create all of the artifacts for our EDA. Note, this was done manually and then we converted our EDA steps into a function so we could archive to disk all of our results.

In [6]:
```
def EDA(source_df, name):

    # seasonal decompose graphs for the entire data
    from statsmodels.tsa.seasonal import seasonal_decompose

    # decompose

    result = seasonal_decompose(source_df, model='additive', freq=12)
    result.plot()

    plt.savefig(f'{EDA_IMAGE_PATH}/{name}_decompose.png')
    plt.show()

    # histogram

    ax = sns.distplot(source_df.value)
    plt.show()
    plt.savefig(f'{EDA_IMAGE_PATH}/{name}_histogram.png')

    # Line plot

    layout = go.Layout(
    #     yaxis=dict(
    #         range=[-10, 10]
    #     )
         title = f'{name}'
    )
    fig = go.Figure(data=[{
        'x': source_df.index,
        'y': source_df[col],
        'name': col
    }  for col in source_df.columns], layout=layout)

    iplot(fig)

    py.image.save_as(fig, f'{EDA_IMAGE_PATH}/{name}_trend.png')
```

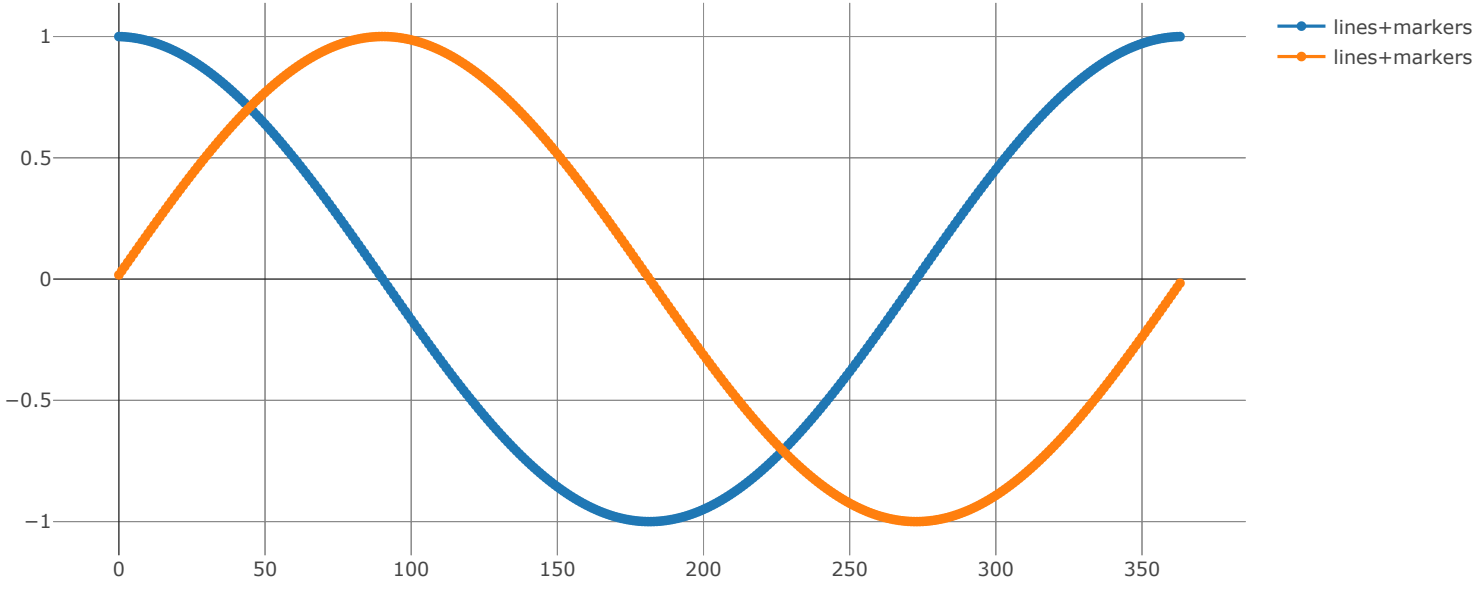
Lets create a couple of functions to help with feature engineering

- Calculate a numerical value and turn it into a cyclical value (i.e. dayOfYear, dayOfMonth) and append onto the dataframe
- Calculate a numerical set of lags / moving averages and append these onto the dataframe

In [10]:
```python
def create_cyclical_signal(new_df, col, max_val):
    """
    Create cyclical signal from column
    """
    new_df[col + '_sin'] = np.sin(2 * np.pi * new_df[col]/max_val)
    new_df[col + '_cos'] = np.cos(2 * np.pi * new_df[col]/max_val)
    return new_df


def create_enhanced_features(df, column, lags, mavgs, include_column=False):
    """
    Create lags and moving averages
    """
    new_df = pd.DataFrame();
    if(include_column):
        new_df[column] = df[column]

    for lag in lags:
        new_df[column+f'_lag{lag}']=df[column].shift(lag)

    for mavg in mavgs:
        new_df[column+f'_mavg{mavg}']=df[column].rolling(window=mavg).mean()

    return new_df.dropna()
```

Lets test our cyclical signal

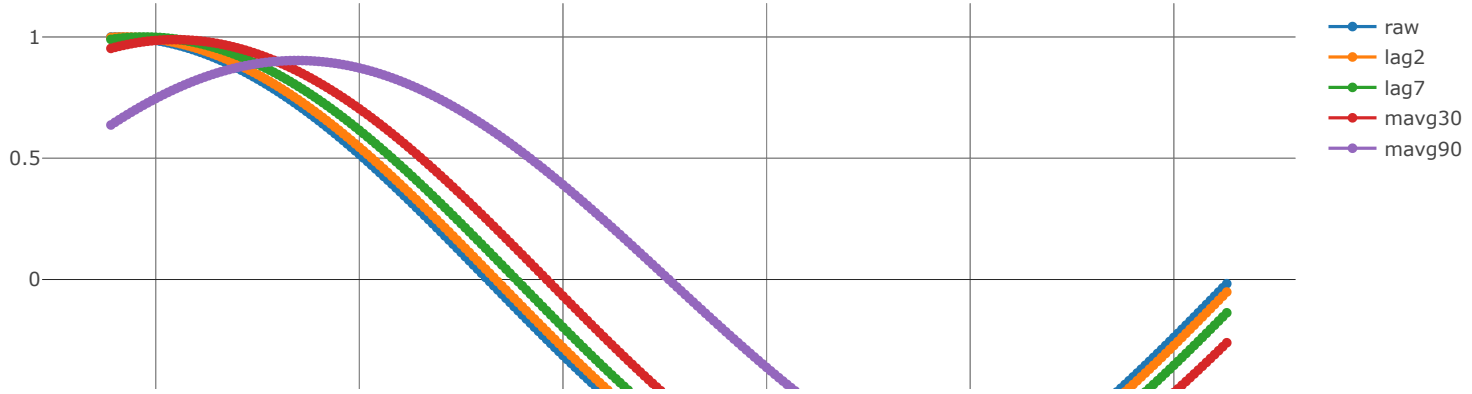
In [8]:
```
x2 = pd.DataFrame(data=pd.Series(np.arange(1,365)), columns=['value'])
x2 = create_cyclical_signal(x2, 'value', 365)

trace1 = go.Scatter(
    x = x2.index,
    y = x2['value_cos'],
    mode = 'lines+markers',
    name = 'lines+markers'
)
trace2 = go.Scatter(
    x = x2.index,
    y = x2['value_sin'],
    mode = 'lines+markers',
    name = 'lines+markers'
)

iplot([trace1,trace2])
```

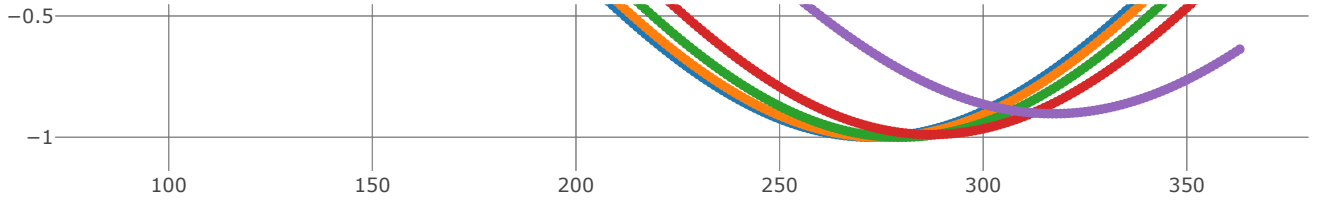


Export to plotly.att.com »

Lets test our lags

In [9]:
```
x_lags = create_enhanced_features(x2, 'value_sin', [2,7], [30,90], include_column=True)

trace1 = go.Scatter(
    x = x_lags.index,
    y = x_lags['value_sin'],
    mode = 'lines+markers',
    name = 'raw'
)
trace2 = go.Scatter(
    x = x_lags.index,
    y = x_lags['value_sin_lag2'],
    mode = 'lines+markers',
    name = 'lag2'
)
trace3 = go.Scatter(
    x = x_lags.index,
    y = x_lags['value_sin_lag7'],
    mode = 'lines+markers',
    name = 'lag7'
)
trace4 = go.Scatter(
    x = x_lags.index,
    y = x_lags['value_sin_mavg30'],
    mode = 'lines+markers',
    name = 'mavg30'
)
trace5 = go.Scatter(
    x = x_lags.index,
    y = x_lags['value_sin_mavg90'],
    mode = 'lines+markers',
    name = 'mavg90'
)
iplot([trace1,trace2,trace3,trace4,trace5])
```





Export to plotly.att.com »

Now lets build a dictionary which represents a bag of parameters for each of the files we will batch process. Then run the artifact parser and EDA routine to generate the outputs. Here we use the ipython widget 'out' to capture the output of the parsing

In [10]:
```
out.clear_output()
```

In [9]:
```
def parse_files(files, runEDA = False):
    """
    Iterate over files, parse, create EDA artifacts and then save as pickles
    """
    for key,value in files.items():

        print('------------------------- starting {0} ------------------------- '.format(key))
        filename = f'{DATA_PATH}/' + value['filename']
        parser = getattr(FileParser, value['parser'])
        df = parser(key, filename, value['column'], value['debug'])
        df.to_pickle(f'{PICKLE_PATH}/clean_{key}.pkl')

        # Run EDA?
        if (runEDA):
            EDA(df, key)

        # Create enhanced features
        enhanced_df = create_enhanced_features(df, value['column'], value['lags'], value['mavgs'], include_column=True)
        enhanced_df.to_pickle(f'{PICKLE_PATH}/enhanced_{key}.pkl')
        print('------------------------- finished {0} ------------------------- '.format(key))
```

In [25]:
```
# Used to track progress
out
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

The data files provided by *NOAA* have already been standardized about their mean and any missing data points have been imputed using interpolation.

We load each time series based on a daily or monthly parsing routing. We then subsequently plot * a decomposition of each time series showing the observed, trend, seasonal and residual components * a histogram of the entire series * a plot of the time series (allowing for the identification of any radical outliers)

In [13]:
```
#Shred files that need unwrapping

# Parameter file, contains all the details of which parser, file name, and features for enhancement
files = {'ao': {'parser': 'DailyParser','filename': 'norm.daily.ao.index.19500101.current.ascii', 'column': 'value', 'debug': False, 'lags':[2,7
         'nao': {'parser': 'DailyParser','filename': 'norm.daily.nao.index.19500101.current.ascii', 'column': 'value', 'debug': False, 'lags':[2
         'pna': {'parser': 'DailyParser','filename': 'norm.daily.pna.index.19500101.current.ascii', 'column': 'value', 'debug': False, 'lags':[2
        }

parse_files(files)
# To run with EDA
#parse_files(files, True)
```

```
-------------------------- starting ao --------------------------
-------------------------- finished ao --------------------------
-------------------------- starting nao --------------------------
-------------------------- finished nao --------------------------
-------------------------- starting pna --------------------------
-------------------------- finished pna --------------------------
```

In [14]:
```
# Parameter file, contains all the details of which parser, file name, and features for enhancement
files = {'nino3': {'parser': 'MonthlyParser','filename': 'nino3.long.anom.data.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],'m
         'nino4': {'parser': 'MonthlyParser','filename': 'nino4.long.anom.data.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],'m
         'nino12': {'parser': 'MonthlyParser','filename': 'nino12.long.anom.data.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],
         'nino34': {'parser': 'MonthlyParser','filename': 'nino34.long.anom.data.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],
         'np': {'parser': 'MonthlyParser','filename': 'np.monthly.long.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],'mavgs':[3
         'pdo': {'parser': 'MonthlyParser','filename': 'pdo.monthly.long.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],'mavgs':
         'soi': {'parser': 'MonthlyParser','filename': 'soi.monthly.long.ascii', 'column': 'value', 'debug': False, 'lags':[2,7,30,365],'mavgs':
           }

parse_files(files)
# To run with EDA
#parse_files(files, True)
```

```
-------------------------- starting nino3 --------------------------
-------------------------- finished nino3 --------------------------
-------------------------- starting nino4 --------------------------
-------------------------- finished nino4 --------------------------
-------------------------- starting nino12 --------------------------
-------------------------- finished nino12 --------------------------
-------------------------- starting nino34 --------------------------
-------------------------- finished nino34 --------------------------
-------------------------- starting np --------------------------
-------------------------- finished np --------------------------
-------------------------- starting pdo --------------------------
-------------------------- finished pdo --------------------------
-------------------------- starting soi --------------------------
-------------------------- finished soi --------------------------
```

Lets examine a couple of our loaded files

In [15]:
```
path = f'{PICKLE_PATH}/clean_nino3.pkl'
df = pd.read_pickle(path)
df.head(5)
```

Out[15]:

datetime	value
1900-01-31	1.40
1900-02-28	1.35
1900-03-31	1.04
1900-04-30	0.59
1900-05-31	0.51

In [16]:
```
path = f'{PICKLE_PATH}/clean_nao.pkl'
df = pd.read_pickle(path)
df.head(5)
```

Out[16]:

datetime	value
1950-01-01	0.365
1950-01-02	0.096
1950-01-03	-0.416
1950-01-04	-0.616
1950-01-05	-0.261

Now lets create a function that can import our temperatures from our *temperature.csv* file

In [7]:
```
def load_temperature(city):
    temperature_df = pd.read_csv(f'{DATA_PATH}/temperature.csv', parse_dates=['datetime'], header =0)
    temperature_df
    data_all = temperature_df[['datetime', f'{city}']]
    data_all = data_all.rename(columns={f'{city}': 'temperature'})
    data_all['temperature'] = data_all['temperature'] - 273.15
    data_all= data_all.fillna(method = 'bfill', axis=0).dropna()
    data_all.Timestamp = pd.to_datetime(data_all.datetime,format='%d-%m-%Y %H:%M')
    data_all.index = data_all.Timestamp
    data_all = data_all.resample('D').mean()
#     display(data_all)
    return data_all
```

Lets test this for New York, here we also enhance the feature set by adding 2 lags and a 30 day moving average.

In [24]:
```
source_df = load_temperature('New York')
enhanced_df = create_enhanced_features(source_df, 'temperature', [1,2], [30] , include_column=True)
enhanced_df.head(10)
```

Out[24]:

datetime	temperature	temperature_lag1	temperature_lag2	temperature_mavg30
2012-10-30	13.630417	14.319583	16.733750	14.452486
2012-10-31	13.050833	13.630417	14.319583	14.374503
2012-11-01	9.213750	13.050833	13.630417	14.090215
2012-11-02	8.306250	9.213750	13.050833	13.803236
2012-11-03	9.368750	8.306250	9.213750	13.487625
2012-11-04	7.492917	9.368750	8.306250	13.040938
2012-11-05	6.564167	7.492917	9.368750	12.599618
2012-11-06	6.031667	6.564167	7.492917	12.171451
2012-11-07	1.910000	6.031667	6.564167	11.885215
2012-11-08	2.433333	1.910000	6.031667	11.656444

Now lets create a function that can create composite files of multiple feeds, these will act as the data cubes for the rest of our analysis.

In [5]:
```
# Combine datasets to create an UBER cube of features and train/test splits

def blender(city, lags, mavgs, feature_files,feature_set_type, split=False, trace=False):

    source_df = load_temperature(city)

    # Add new columns onto df (lags/mavgs)

    enhanced_df = create_enhanced_features(source_df, 'temperature', lags, mavgs , include_column=True)

    # Add signals onto the enhanced df

    for feature_file in feature_files:
        path = f'{PICKLE_PATH}/{feature_file}.pkl'
        df = pd.read_pickle(path)
        for c in df.columns:
            df[c].astype(float)
            df.rename(index=str, columns={c: feature_file + '_' + c}, inplace=True)
        enhanced_df = enhanced_df.join(df, how='left')

    for c in  enhanced_df.columns:
        enhanced_df[c].astype(float)
        enhanced_df[c] = enhanced_df[c].fillna(method='bfill')
        enhanced_df[c] = enhanced_df[c].fillna(method='ffill')

    if (trace):
        print(enhanced_df.head(5))

    # Save enhanced city df

    city = city.replace(' ','_')
    enhanced_df.to_pickle(f'{PICKLE_PATH}/enhanced_{city}.pkl')

    # Create train/test split

    if (split):
        train_size = enhanced_df.shape[0] - 90
        train = enhanced_df[0:train_size]
        test = enhanced_df[train_size:]

        X_train = train.drop('temperature', 1)
        X_train._metadata = {"feature_set_type": feature_set_type,'city':city.replace("_"," "),'lags':lags, 'mavgs': mavgs, 'feature_files': fea
        Y_train = pd.DataFrame(train.temperature)
        Y_train._metadata = {"feature_set_type": feature_set_type,'city':city.replace("_"," "),'lags':lags, 'mavgs': mavgs, 'feature_files': fea
        X_test  = test.drop('temperature', 1)
        Y_test  = pd.DataFrame(test.temperature)

        X_train.to_pickle(f'{PICKLE_PATH}/X_train_{city}_{feature_set_type}.pkl')
        Y_train.to_pickle(f'{PICKLE_PATH}/Y_train_{city}_{feature_set_type}.pkl')
        X_test.to_pickle(f'{PICKLE_PATH}/X_test_{city}_{feature_set_type}.pkl')
        Y_test.to_pickle(f'{PICKLE_PATH}/Y_test_{city}_{feature_set_type}.pkl')

        print("Shape of training Data",X_train.shape)
```

```
        print("Shape of testing Data",X_test.shape)
```

Now lets test this for *New York* combining it with our NAO data

In [20]:
```
blender('Houston',[],[],['clean_nao'],'Basic',False,True)
```

```
Shape of training Data (1797, 1)
Shape of testing Data (90, 1)
```

Lets test it again with two extra data feeds

In [21]:
```
blender('New York',[],[],['enhanced_nino3'],'Basic',False,True)
```

```
Shape of training Data (1764, 7)
Shape of testing Data (90, 7)
```

Finally, now as everything is working lets compile our data-cubes for each of our locations / weather stations

- Basic - City + lags
- Inc - City + lags + Signal
- Enhanced - City + lags + Signal + lags

Adjustments we have made can be described as follows

- by lags we mean prior values from 1 day, 2 days, 7 days, 30 days, 90 days and 1 year ago
- by Moving averages we mean average over 1 week, 30 days, 60 days etc

In [11]:
```
# City datasets + lags/mavgs ~ BASIC datasets

feature_sets = []
lags = [1,2,7,30,90,365]
mavgs = [7,30,60,90,180]

locations = ['Los Angeles','Miami','New York','Dallas','Houston','Boston','Atlanta']

for location in locations:
    blender(location,lags,mavgs,feature_sets,'Basic',True,False)
    print(f'exported {location} data pickle')
```

```
Shape of training Data (1432, 11)
Shape of testing Data (90, 11)
exported Los Angeles data pickle
Shape of training Data (1399, 11)
Shape of testing Data (90, 11)
exported Miami data pickle
Shape of training Data (1399, 11)
Shape of testing Data (90, 11)
exported New York data pickle
Shape of training Data (1432, 11)
Shape of testing Data (90, 11)
exported Dallas data pickle
Shape of training Data (1432, 11)
Shape of testing Data (90, 11)
exported Houston data pickle
Shape of training Data (1432, 11)
Shape of testing Data (90, 11)
exported Boston data pickle
Shape of training Data (1432, 11)
Shape of testing Data (90, 11)
exported Atlanta data pickle
```

In [12]:
```
# City datasets + lags/mavgs ~ INC_SIGNALS datasets
feature_sets = ['clean_ao','clean_nao','clean_nino3','clean_nino4','clean_nino12','clean_nino34']
lags = [1,2,7,30,90,365]
mavgs = [7,30,60,90,180]
locations = ['Los Angeles','Miami','New York','Dallas','Houston','Boston','Atlanta']

for location in locations:
    blender(location,lags,mavgs,feature_sets,'Inc_signals',True,False)
    print(f'exported {location} data pickle')
```

```
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Los Angeles data pickle
Shape of training Data (1399, 17)
Shape of testing Data (90, 17)
exported Miami data pickle
Shape of training Data (1399, 17)
Shape of testing Data (90, 17)
exported New York data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Dallas data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Houston data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Boston data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Atlanta data pickle
```

In [13]:
```
# City datasets + lags/mavgs ~ ENHANCED_SIGNALS datasets
feature_sets = ['clean_ao','clean_nao','clean_nino3','clean_nino4','clean_nino12','clean_nino34']
lags = [1,2,7,30,90,365]
mavgs = [7,30,60,90,180]
locations = ['Los Angeles','Miami','New York','Dallas','Houston','Boston','Atlanta']

for location in locations:
    blender(location,lags,mavgs,feature_sets,'Enhanced_signals',True,False)
    print(f'exported {location} data pickle')
```

```
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Los Angeles data pickle
Shape of training Data (1399, 17)
Shape of testing Data (90, 17)
exported Miami data pickle
Shape of training Data (1399, 17)
Shape of testing Data (90, 17)
exported New York data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Dallas data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Houston data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Boston data pickle
Shape of training Data (1432, 17)
Shape of testing Data (90, 17)
exported Atlanta data pickle
```

In [ ]:

Exploratory Data Analysis - Temperature Data Viewer

Open the menu item Cell and click Run All to see a summary of the data feed passed into this notebook from the URL.

In [11]:
```
%%javascript
function getQueryStringValue (key)
{
    return unescape(window.location.search.replace(new RegExp("^(?:.*[&\\?]" + escape(key).replace(/[\.\+\*]/g, "\\$&") + "(?:\\=([^&]*))?)?.*$", "i"), "$1"));
}
IPython.notebook.kernel.execute("DATA='".concat(getQueryStringValue("DATA")).concat("'"));
```

Load libraries for charting

In [12]:
```
%run DataViewerHelper.py
```

Get chart data and review head to see features (and those we have enhanced)

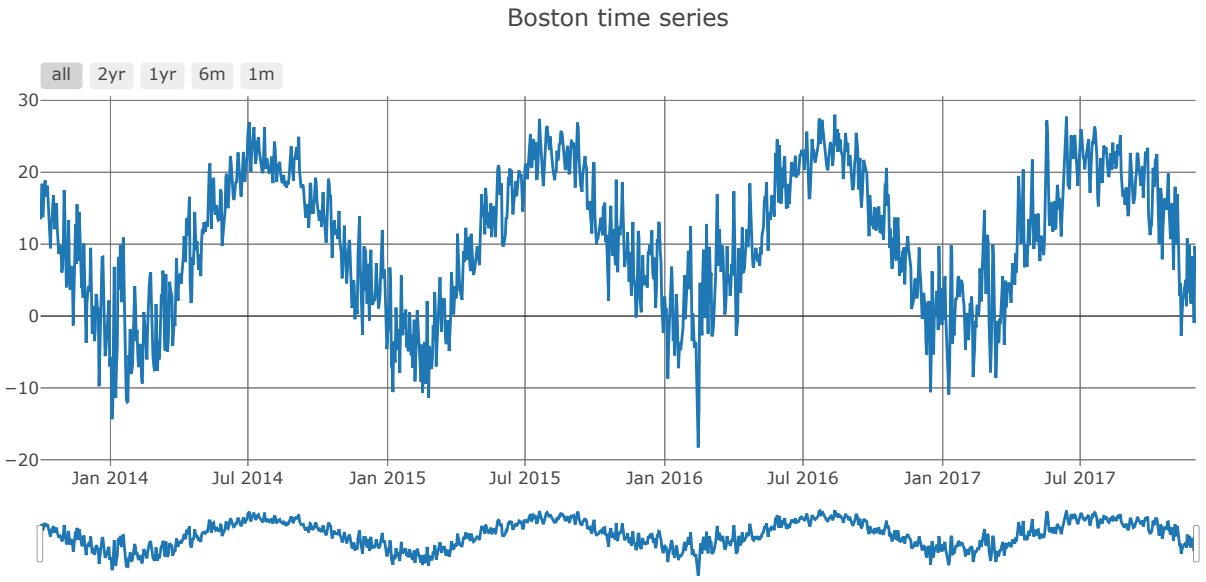
In [13]:
```
df = get_data(DATA)
df.head(10)
```

Out[13]:

datetime	temperature	temperature_lag1	temperature_lag2	temperature_lag7	temperature_lag30	temperature_lag90	temperature_lag365	temperature_mavg7	temperature_mavg30	temperature_mavg60	tempe
2013-10-01	13.435553	12.082927	13.775979	11.581382	22.374375	24.321125	14.204333	13.418761	17.882211	19.420925	
2013-10-02	17.426121	13.435553	12.082927	12.606729	23.005167	24.445833	15.863090	14.107246	17.696243	19.352180	
2013-10-03	18.405358	17.426121	13.435553	14.247764	21.077704	27.494375	15.870833	14.701187	17.607165	19.306616	
2013-10-04	13.742917	18.405358	17.426121	14.693125	20.946385	28.105375	16.893750	14.565443	17.367049	19.190891	
2013-10-05	17.659250	13.742917	18.405358	13.089250	18.253493	27.980833	16.367292	15.218301	17.347241	19.175545	
2013-10-06	17.948667	17.659250	13.742917	13.775979	19.216074	26.404583	17.998958	15.814399	17.304994	19.190334	
2013-10-07	18.860583	17.948667	17.659250	12.082927	18.642200	25.204167	18.042917	16.782635	17.312273	19.185328	
2013-10-08	16.668610	18.860583	17.948667	13.435553	19.479515	20.030417	10.657917	17.244501	17.218577	19.084884	
2013-10-09	17.186250	16.668610	18.860583	17.426121	13.941667	22.366250	9.864375	17.210233	17.326729	18.984527	
2013-10-10	18.134583	17.186250	16.668610	18.405358	17.676625	24.219167	11.317917	17.171551	17.341995	18.889336	

Lets now plot the time series

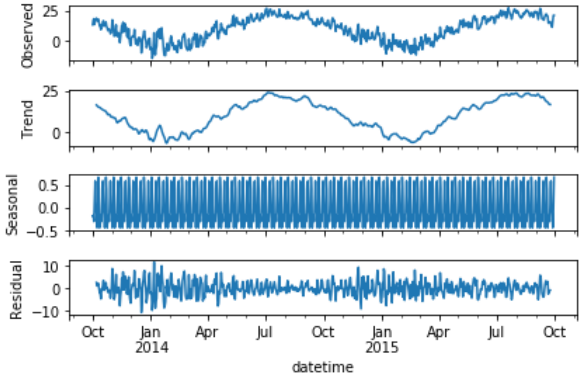
In [14]:
```
fig = chart(f'{DATA} time series', df)
iplot(fig)
```



Export to plot.ly »

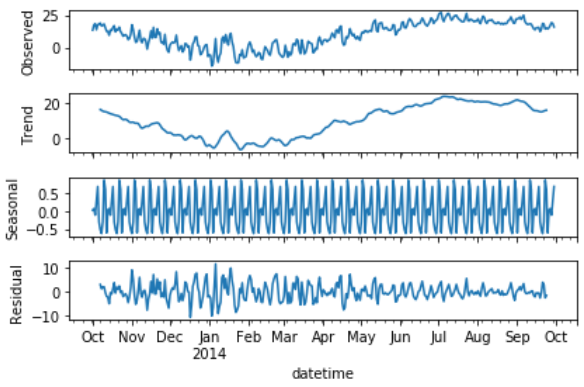
Lets review multiple years and examine the observed, trends, seasonality and residuals using decomposition with additive differencing.

In [15]:
```
dsd = seasonal_decompose(df[:730]['temperature'], model='additive',freq=12)
dsd.plot()
plt.show()
```



Lets zoom in an review a single calendar year

In [17]:
```
dsd = seasonal_decompose(df[:365]['temperature'], model='additive',freq=12)
dsd.plot()
plt.show()
```



We can see each temperature time series has a well structured oscillating trend pattern coinciding with the seasonal patterns.

In [ ]:

Exploratory Data Analysis - SIGNAL Viewer

Open the menu item Cell and click Run All to see a summary of the data feed passed into this notebook from the URL.

In [4]:
```
%%javascript
function getQueryStringValue (key)
{
    return unescape(window.location.search.replace(new RegExp("^(?:.*[&\\?]" + escape(key).replace(/[\.\+\*]/g, "\\$&") + "(?:\\=([^&]*))?)?.*$", "i"), "$1"));
}
IPython.notebook.kernel.execute("DATA='".concat(getQueryStringValue("DATA")).concat("'"));
```

Load libraries for charting

In [16]:
```
%run SignalViewerHelper.py
```

```
Get chart data and review the features we have created.
```

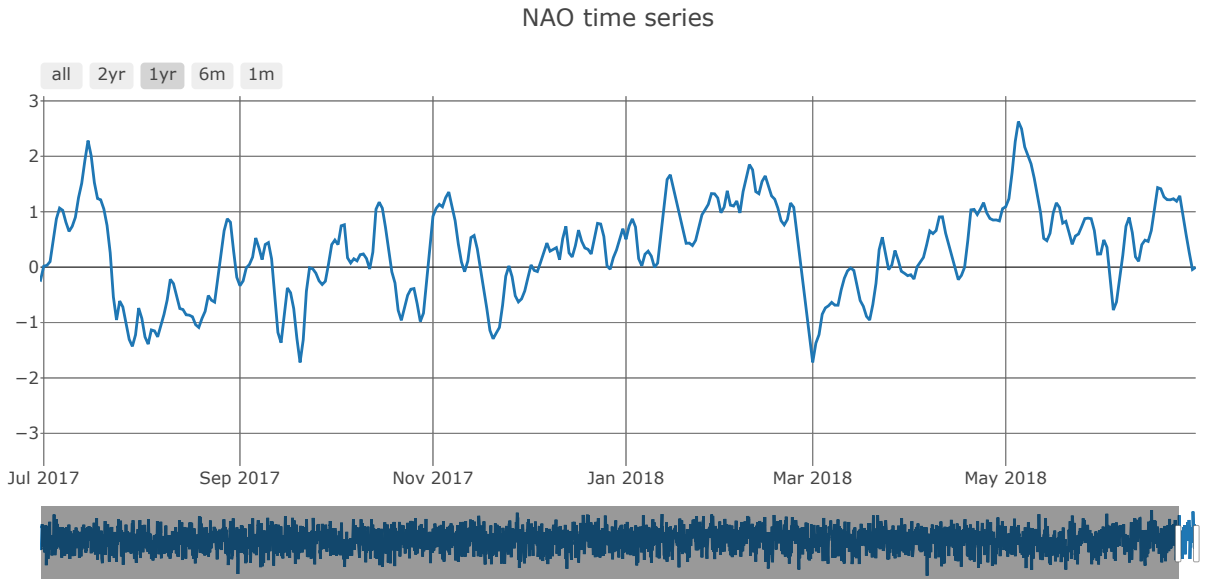
In [8]:
```
df = get_data(DATA)
df.head(10)
```

Out[8]:

datetime	value	value_lag2	value_lag7	value_lag30	value_lag365	value_mavg30	value_mavg90
1951-01-01	-0.745	-0.700	-1.642	-0.216	0.365	-0.613067	-0.279522
1951-01-02	-0.529	-0.646	-1.671	-0.564	0.096	-0.611900	-0.298333
1951-01-03	-0.217	-0.745	-1.454	-1.055	-0.416	-0.583967	-0.314622
1951-01-04	0.057	-0.529	-1.212	-0.976	-0.616	-0.549533	-0.327822
1951-01-05	-0.045	-0.217	-0.923	-0.058	-0.261	-0.549100	-0.339600
1951-01-06	0.013	0.057	-0.700	0.529	0.052	-0.566300	-0.347178
1951-01-07	0.022	-0.045	-0.646	0.282	-0.156	-0.574967	-0.352167
1951-01-08	-0.314	0.013	-0.745	-0.136	-0.440	-0.580900	-0.355489
1951-01-09	-0.434	0.022	-0.529	-0.121	-0.497	-0.591333	-0.357289
1951-01-10	-0.092	-0.314	-0.217	0.058	-0.227	-0.596333	-0.355000

Lets review the time series and explore the series using the interactive chart.

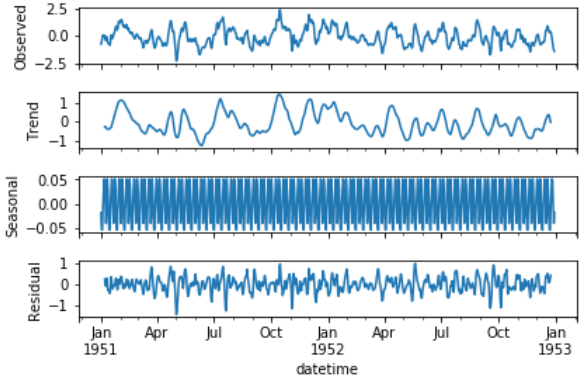
In [9]:
```
fig = chart(f'{DATA} time series', df)
iplot(fig)
```



Export to plot.ly »

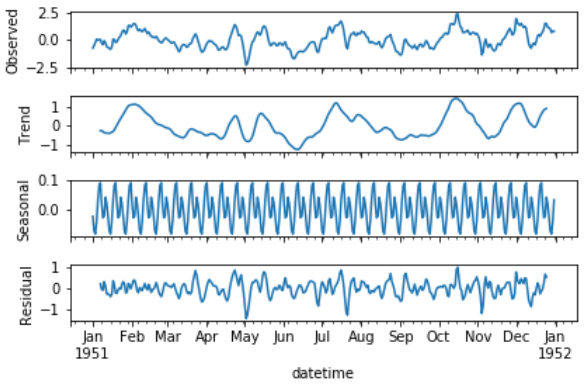
Lets review multiple years and examine the observed, trends, seasonality and residuals using decomposition with additive differencing.

In [18]:
```
dsd = seasonal_decompose(df[:730]['value'], model='additive',freq=12)
dsd.plot()
plt.show()
```



Lets zoom in an review a single calendar year

In [19]:
```
dsd = seasonal_decompose(df[:365]['value'], model='additive',freq=12)
dsd.plot()
plt.show()
```



In [ ]:

Time Series to observe DAILY temperature variations

Daily temperature prediction using ARIMA

Autoregressive Integrated Moving Average Model

An ARIMA model is a class of statistical models for analyzing and forecasting time series data.

It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations. **I**: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary. **MA**: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

p: The number of lag observations included in the model, also called the lag order.
d: The number of times that the raw observations are differenced, also called the degree of differencing.
q: The size of the moving average window, also called the order of moving average.

A linear regression model is constructed including the specified number and type of terms, and the data is prepared by a degree of differencing in order to make it stationary, i.e. to remove trend and seasonal structures that negatively affect the regression model.

A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model.

Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process. This may seem obvious, but helps to motivate the need to confirm the assumptions of the model in the raw observations and in the residual errors of forecasts from the model.

In [38]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Install custom lib for hyper-parameter searching for ARIMA

In [376]:
```
# !pip install pyramid-arima
```

Create a folder for every run of the RNN to store images

In [6]:
```
EXPERIMENT_DIR = create_results_perrun()
print("Path of the results directory",EXPERIMENT_DIR )
```

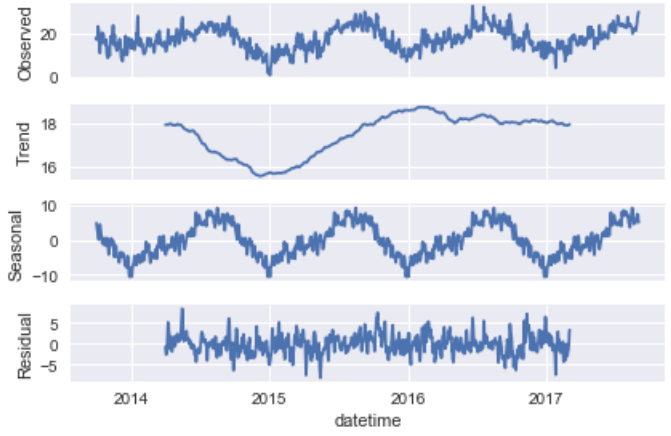
```
Path of the results Image directory ../Images/RESULTS/RUN-17
```

In [204]:
```
city='Los_Angeles'
Y_train = pd.read_pickle(f'{PICKLE_PATH}/Y_train_{city}.pkl')
Y_test  = pd.read_pickle(f'{PICKLE_PATH}/Y_test_{city}.pkl')

print("Shape of Training Dataset " , Y_train.shape)
print("Shape of Testing Dataset " , Y_test.shape)

freq=365
```

```
Shape of Training Dataset  (1432, 1)
Shape of Testing Dataset  (90, 1)
```

Lets also do a decomposition, to identify the trend, seasonal and residuals of the timeseries.

In [133]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
dsd = seasonal_decompose(Y_train, model='additive',freq=365)
dsd.plot()
plt.show()
```

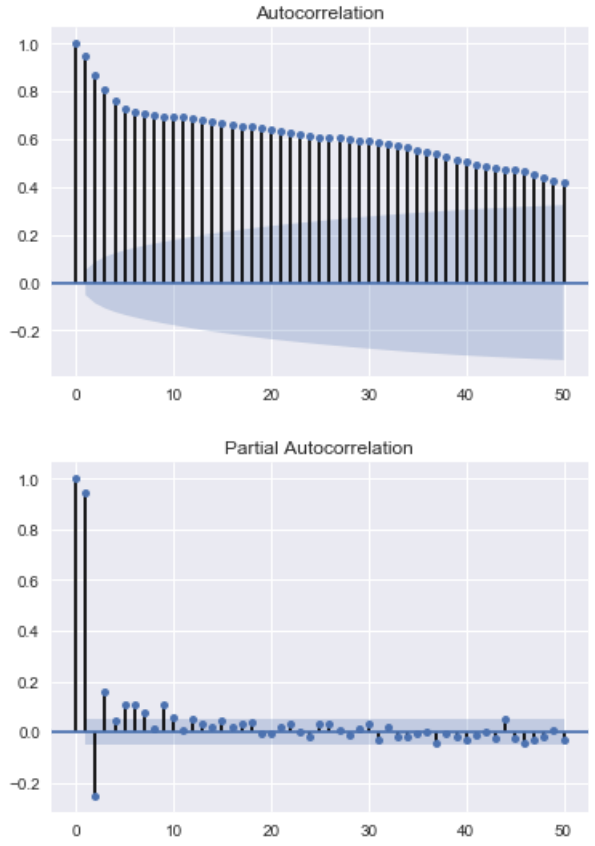


We can see a seasonal component and a horizontal trend that is linear +/- 1

Differencing the series

If a graphical plot of the data indicates nonstationarity, then you should "difference" the series. So, lets plot an autocorrelation and partial autocorrelation plot to examine the series.

In [131]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

plot_acf(Y_train, lags=50)
plot_pacf(Y_train, lags=50)
plt.show()
```



In this example, we can see an decaying pattern, indicating we need to difference the series.

Differencing is an excellent way of transforming a nonstationary series to a stationary one. This is done by subtracting the observation in the current period from the previous one. If this transformation is done only once to a series, you say that the data has been "first differenced". This process essentially eliminates the trend if your series is growing at a fairly constant rate. If it is growing at an increasing rate, you can apply the same procedure and difference the data again. Your data would then be "second differenced".

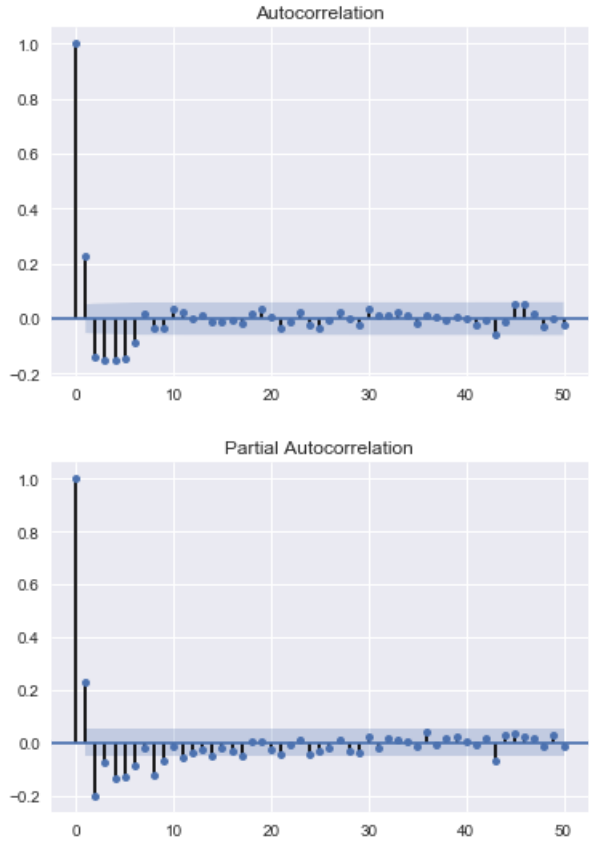
In [134]:
```
#first difference
diff_test_set = Y_train.diff().fillna(method='bfill')
diff_test_set.head(5)
```

Out[134]:

	temperature
datetime	
2013-10-01	-0.798487
2013-10-02	-0.798487
2013-10-03	1.473410
2013-10-04	0.844677
2013-10-05	3.810417

Running the autocorrelation plot again, shows the differencing has removed the cyclic pattern and the time series is however still not stationary.

In [135]:
```
plot_acf(diff_test_set, lags=50)
plot_pacf(diff_test_set, lags=50)
plt.show()
```



Next lets run an augmented dickey fuller test, to see if we have stationarity

In [147]:
```
from statsmodels.tsa.stattools import adfuller
adf_df = adfuller(diff_test_set['temperature'], autolag='AIC')

for key,value in adf_df[4].items():
    print(f'Critical Value ({key})) - {value}')
```

```
Critical Value (1%)) - -3.434979825137732
Critical Value (5%)) - -2.8635847436211317
Critical Value (10%)) - -2.5678586114197954
```

The values indicate the series is still not stationary and it requires a more complex model using an autoregressive or moving average (or both).

We have learned how to to decompose and test a time series for stationarity, now lets use an auto arima function that will perform hyper-parameter tuning.

In [246]:
```
from pyramid.arima import auto_arima
model_fit = auto_arima(Y_train, start_p=1, start_q=1, max_p=2, max_q=2, m=12, max_P=1, max_Q=1,
                           start_P=0, seasonal=True, d=1, D=1, trace=True,
                           error_action='ignore',  # don't want to know if an order does not work
                           suppress_warnings=True,  # don't want convergence warnings
                           stepwise=False, random=True, random_state=42,  # we can fit a random search (not exhaustive)
                             n_fits=20)
```

```
Fit ARIMA: order=(1, 1, 1) seasonal_order=(1, 1, 1, 12); AIC=5414.099, BIC=5445.700, Fit time=6.419 seconds
Fit ARIMA: order=(2, 1, 1) seasonal_order=(1, 1, 1, 12); AIC=5302.123, BIC=5338.991, Fit time=11.679 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 1, 12); AIC=5412.152, BIC=5438.486, Fit time=5.501 seconds
Fit ARIMA: order=(2, 1, 2) seasonal_order=(1, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(1, 1, 2) seasonal_order=(0, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(2, 1, 1) seasonal_order=(0, 1, 1, 12); AIC=5300.370, BIC=5331.971, Fit time=6.308 seconds
Fit ARIMA: order=(1, 1, 2) seasonal_order=(1, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(2, 1, 2) seasonal_order=(0, 1, 1, 12); AIC=nan, BIC=nan, Fit time=nan seconds
Total fit time: 29.922 seconds
```

Lets review the model and the summary of the auto fit routine

In [247]:
```
model_fit.summary()
```

Out[247]:

Statespace Model Results

Dep. Variable:	y	**No. Observations:**	1432
Model:	SARIMAX(2, 1, 1)x(0, 1, 1, 12)	**Log Likelihood**	-2644.185
Date:	Tue, 14 Aug 2018	**AIC**	5300.370
Time:	13:34:20	**BIC**	5331.971
Sample:	0	**HQIC**	5312.170
	- 1432		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	8.273e-05	0.000	0.592	0.554	-0.000	0.000
ar.L1	1.0710	0.027	40.020	0.000	1.019	1.123
ar.L2	-0.3537	0.023	-15.532	0.000	-0.398	-0.309
ma.L1	-0.8949	0.019	-46.174	0.000	-0.933	-0.857
ma.S.L12	-0.9939	0.028	-36.041	0.000	-1.048	-0.940
sigma2	2.3466	0.082	28.448	0.000	2.185	2.508

Ljung-Box (Q):	45.84	**Jarque-Bera (JB):**	179.78
Prob(Q):	0.24	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.73	**Skew:**	0.18
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.71

Lets create an in-sample prediction (using the data we fit with model with)

In [269]:
```
in_sample_preds = model_fit.predict_in_sample()
```

Out[269]:
```
array([ 0.        , 17.83861878, 17.0402024 , ..., 27.14699541,
       27.23811781, 28.52676073])
```

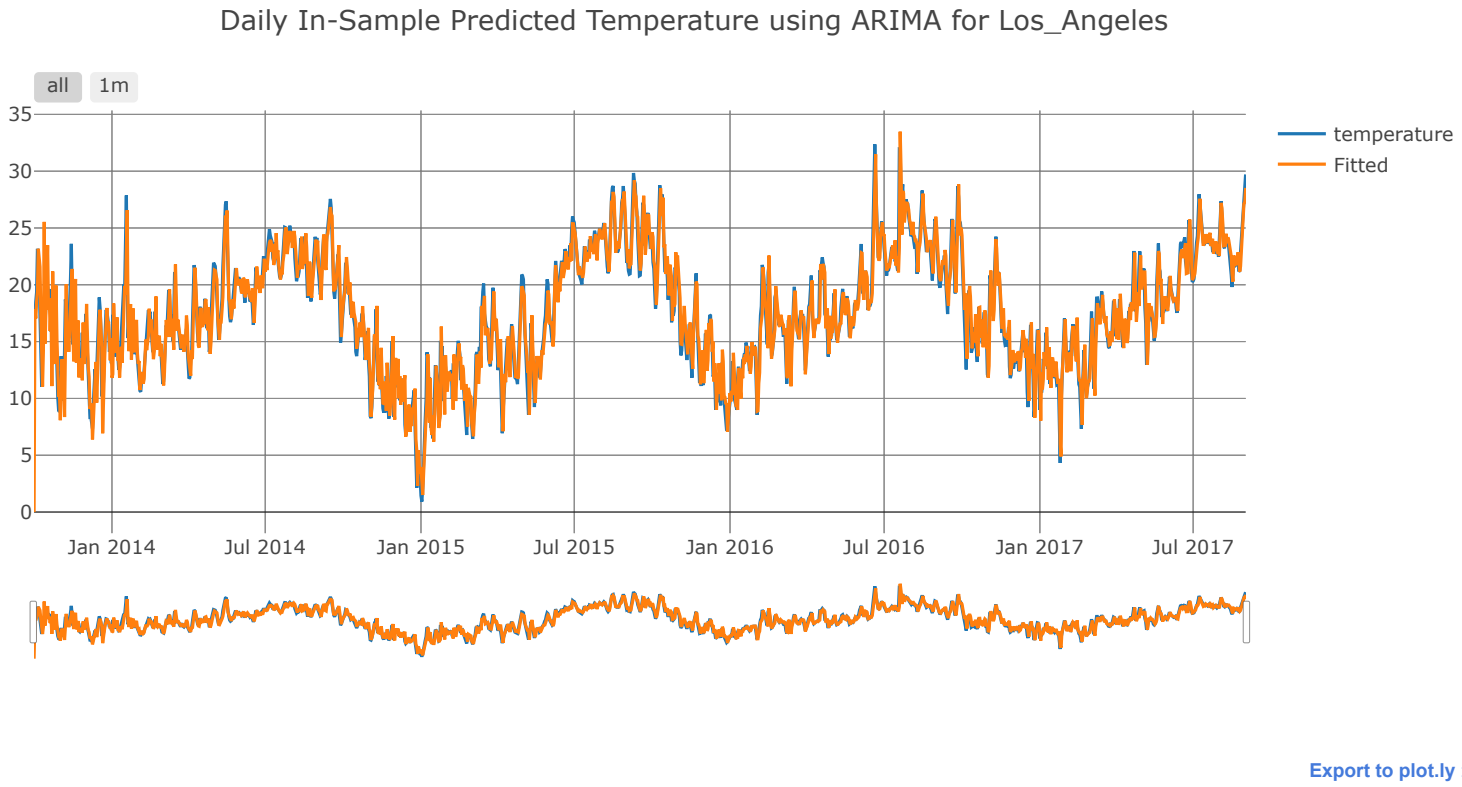
Then lets review the MSE of this

In [284]:
```
mse_train = mean_squared_error(Y_train, in_sample_preds)
print('Mean Squared Error for the testing dataset: %.3f' % mse_train)
```

```
Mean Squared Error for the testing dataset: 2.821
```

Now let us plot this in-sample-prediction so we can see how our model fits to our datasetv

In [327]:
```
predictions_df = pd.DataFrame(in_sample_preds, index=Y_train.index, columns=['Fitted'])
predictions_df

combined_df = Y_train.join(predictions_df, how="left")

fig = charter_helper_fitted("Daily In-Sample Predicted Temperature using ARIMA for " + city, combined_df)
iplot(fig)

py.image.save_as(fig, filename=f'{EXPERIMENT_DIR}/Daily_ARIMA_actual_vs_predict.png')
```



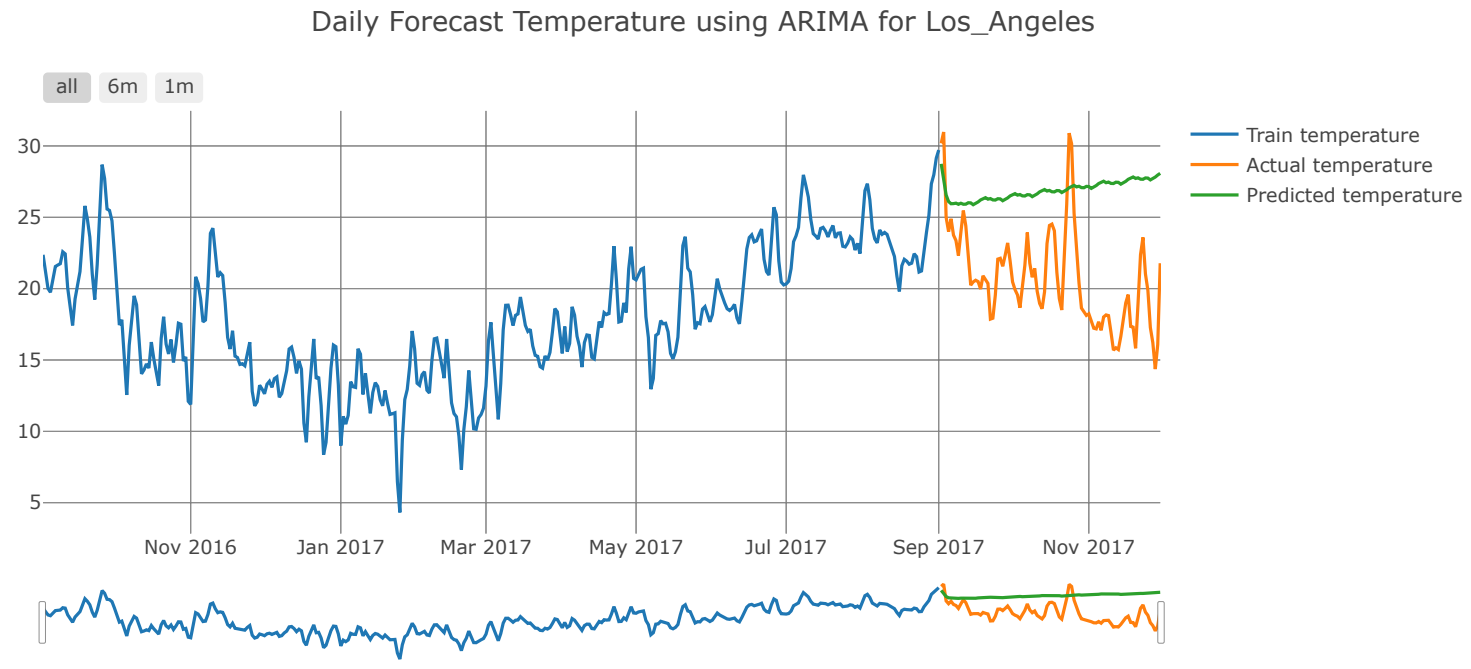
This in-sample prediction looks impressive and the MSE indicates a good fit vs the underlying dataset. Now, lets forecast into the future (based on the Y_test dataset)

In [321]:
```
model_forecast = model_fit.predict(n_periods=len(Y_test))

forecast_df = pd.DataFrame(model_forecast, index=Y_test.index, columns=['Fitted'])
combined_forecast_df = Y_test.join(forecast_df, how="left")
```

In [322]:
```
mse_test = mean_squared_error(Y_test, forecast_df)
print('Mean Squared Error for the testing dataset: %.3f' % mse_test)
```

```
Mean Squared Error for the testing dataset: 51.305
```

In [328]:
```
# Plotting the training data for past year, Actual/test data and predicted temperature - Decision Tree
fig = charter_helper_prediction("Daily Forecast Temperature using ARIMA for " + city, Y_train, Y_train, combined_forecast_df,combined_forecas
t_df,combined_forecast_df)

iplot(fig)
py.image.save_as(fig, filename=f'{EXPERIMENT_DIR}/Daily_ARIMA_predict.png')
```



Export to plot.ly »

Conclusion

The ARIMA model forecast isnt particularly impressive (after all this work), the forecast initially trends correctly but then the forecasts of the future values rapidly damped out and subsequently have no predictive power. We would either have to introduce the forecasted values onto the end of the timeseries and have the model update its training or introduce a more powerful form of model such as SARIMAX (which is a state space model).

Time Series to observe DAILY temperature variations

Daily temperature prediction using Decision Tree

Following ARIMA, we use in the notebook below our first classifier:

Decision Tree (DT) : Similarly to ARIMA, We begin by loading all necessary libraries and paths to read the "pickles" as well as store image for the graph towards the end of our code. The pickles are read and the data is fed into an DT model.

Finally, we have two graphs showing the DT results vs. the fitted model as well as predicted results vs. actuals and test data

In [430]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Create a folder for every run of the Decision tree to store our experiments

In [431]:
```
city='Houston' # New_York Atlanta Boston Dallas Houston Miami
analysis_type = 'Enhanced_Signals' # Basic, Inc_Signals, Enhanced_Signals
```

In [432]:
```
EXPERIMENT_DIR, EXPERIMENT_ID  = create_results_perrun()
print(f"Experiment ID: {EXPERIMENT_ID}")
print(f"Path of the results directory:{EXPERIMENT_DIR}")
```

```
Experiment ID: 18
Path of the results directory:../experiment_results/RUN-18
```

Here we are importing the train and test Data from pickle files created through the EDA file

In [433]:
```
X_train = pd.read_pickle(f'{PICKLE_PATH}/X_train_{city}_{analysis_type}.pkl')
Y_train = pd.read_pickle(f'{PICKLE_PATH}/Y_train_{city}_{analysis_type}.pkl')

X_test  = pd.read_pickle(f'{PICKLE_PATH}/X_test_{city}_{analysis_type}.pkl')
Y_test  = pd.read_pickle(f'{PICKLE_PATH}/Y_test_{city}_{analysis_type}.pkl')

print("Shape of Training Dataset " , X_train.shape)
print("Shape of Testing Dataset " , X_test.shape)
```

```
Shape of Training Dataset  (1432, 17)
Shape of Testing Dataset  (90, 17)
```

In [434]:
```
# Fitting a decision tree regressor with max depth
max_depth = 8
fitted_model = tree.DecisionTreeRegressor(max_depth=max_depth)
fitted_model.fit(X_train,Y_train)

# Dataframe to show features and their importances
top_features = len(fitted_model.feature_importances_)
features_importances_df= show_feature_importances(X_train.columns.values.tolist(),
                                                   fitted_model.feature_importances_,top_features)
features_importances_df.head(10)

# Store results
features_importances_df.to_csv(f'{EXPERIMENT_DIR}/feature_importances.csv')
```

In [435]:
```
# Run the model on the training dataset
Y_train_pred = fitted_model.predict(X_train)

# Calculate mean squared error for the predicted values
mse_train = mean_squared_error(Y_train, Y_train_pred)
print('Mean Squared Error for the training dataset: %.3f' % mse_train)
```

```
Mean Squared Error for the training dataset: 2.785
```

In [436]:
```
# Run the model on the testing dataset
Y_test_pred = fitted_model.predict(X_test)

# Calculate mean squared error for the test vs predicted values
mse_test = mean_squared_error(Y_test, Y_test_pred)
print('Mean Squared Error for the testing dataset: %.3f' % mse_test)
```

```
Mean Squared Error for the testing dataset: 10.205
```

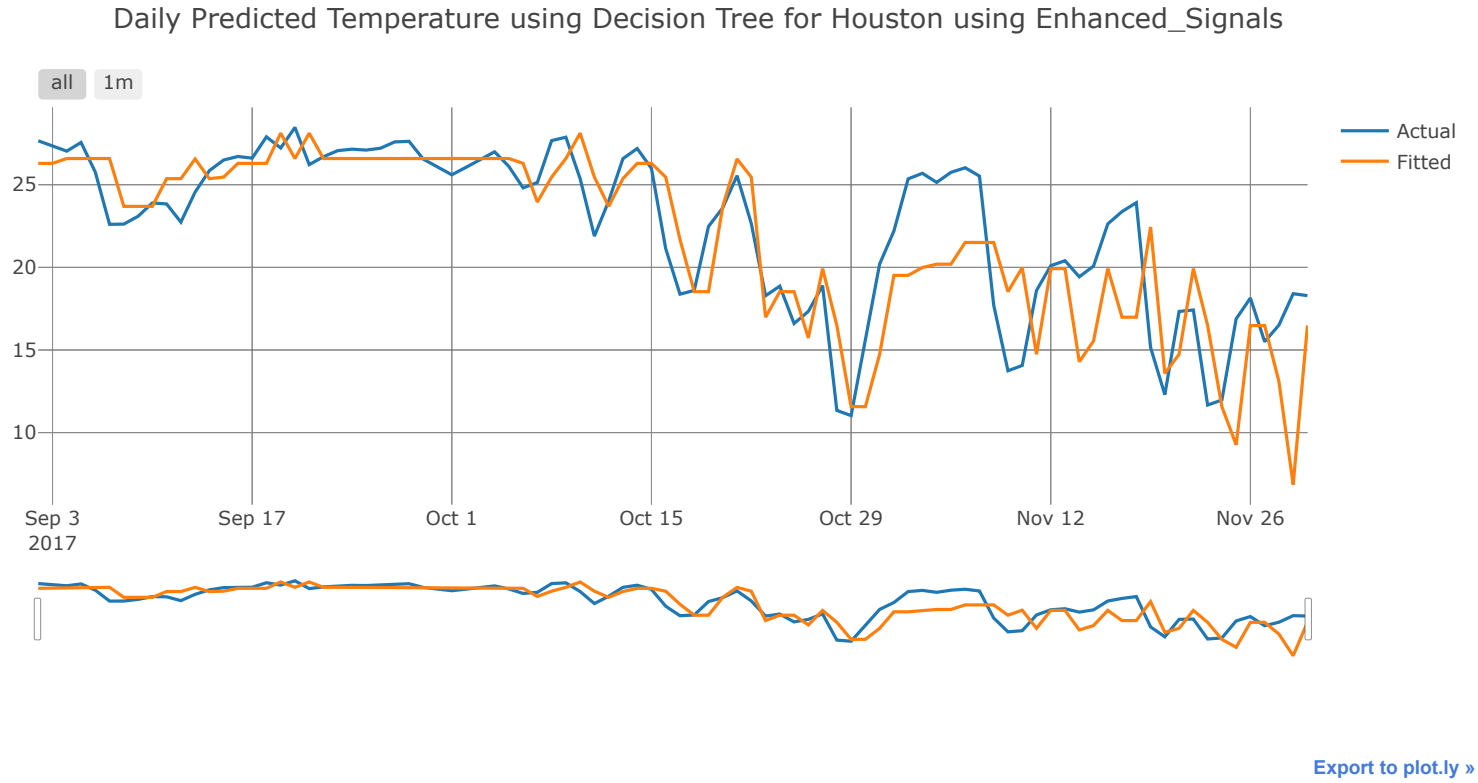
In [437]:
```
# Creating a dataframe for predicted/fitted values
future_forecast = pd.DataFrame(Y_test_pred,index = Y_test.index,columns=['Fitted'])

# Concatenate the predicted/fitted values with actual values to display graphs
predictions = pd.concat([Y_test,future_forecast],axis=1)
predictions.columns = ["Actual","Fitted"]

# Displaying few of the predicted values
predictions.head(10)
```

Out[437]:

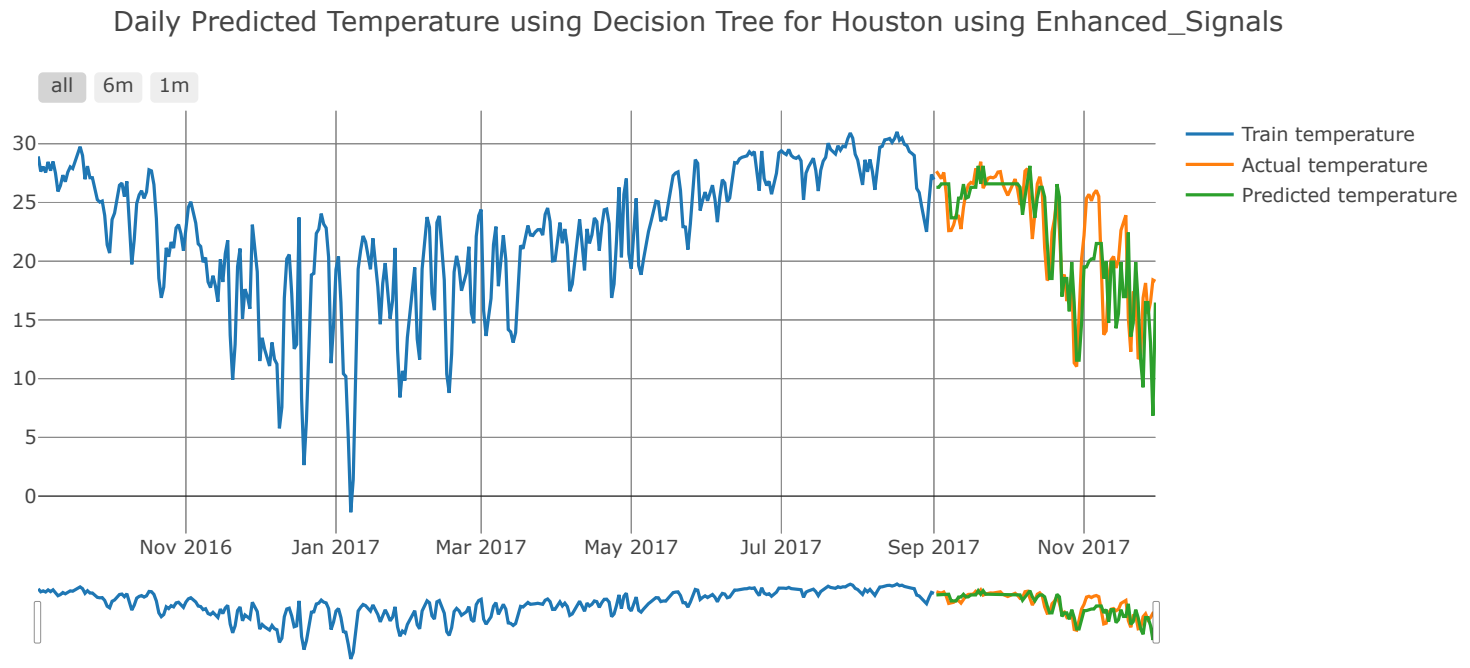
datetime	Actual	Fitted
2017-09-02	27.658333	26.292523
2017-09-03	27.312917	26.292523
2017-09-04	27.032500	26.585889
2017-09-05	27.558333	26.585889
2017-09-06	25.762500	26.585889
2017-09-07	22.602917	26.585889
2017-09-08	22.617917	23.691244
2017-09-09	23.091667	23.691244
2017-09-10	23.897917	23.691244
2017-09-11	23.837500	25.365880

Mean Squared error (MAE), would be easier to interpret as they use the same scale as the data itself.

In [438]:
```
city = city.replace('_',' ')
# Plotting the daily predicted temperature vs Actual Temperature - Decision Tree
fig = charter_helper_fitted(f"Daily Predicted Temperature using Decision Tree for {city} using {analysis_type}", predictions)
iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_DT_actual_vs_predict.png')
```



In [439]:
```
# Plotting the training data for past year, Actual/test data and predicted temperature - Decision Tree
fig = charter_helper_prediction(f"Daily Predicted Temperature using Decision Tree for {city} using {analysis_type}",
                      X_train,Y_train,X_test,Y_test,future_forecast)

iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_DT_predict.png')
```



Export to plot.ly »

In [440]:
```
results = update_results_function(EXPERIMENT_ID, 'DECISION TREE', city, analysis_type,
                                  {'max_depth': max_depth, 'Info': X_train._metadata },
                                  {'features' : X_train.columns.values.tolist(),
                                   'importances':fitted_model.feature_importances_,
                                   'mse_train' : mse_train},
                                    mse_test)
```

In [441]:
```
results.tail(1)
```

Out[441]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
17	18	2018-08-14 19:48:11.736850	DECISION TREE	Houston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	10.205057

In [442]:
```
results = pd.read_pickle('../pickles/results.pkl')
results
```

Out[442]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
0	1	2018-08-14 19:41:45.275297	DECISION TREE	New York	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	5.799872
1	2	2018-08-14 19:42:13.654060	DECISION TREE	Atlanta	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	8.436149
2	3	2018-08-14 19:42:23.328525	DECISION TREE	Boston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	16.504857
3	4	2018-08-14 19:42:30.715058	DECISION TREE	Dallas	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	12.523335
4	5	2018-08-14 19:42:41.506483	DECISION TREE	Houston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.705077
5	6	2018-08-14 19:42:49.124681	DECISION TREE	Miami	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.833555
6	7	2018-08-14 19:43:04.972617	DECISION TREE	New York	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	6.518000
7	8	2018-08-14 19:43:12.930159	DECISION TREE	Atlanta	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.992946
8	9	2018-08-14 19:43:20.383683	DECISION TREE	Boston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	22.967065
9	10	2018-08-14 19:43:32.825549	DECISION TREE	Dallas	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	14.223161
10	11	2018-08-14 19:43:39.374706	DECISION TREE	Houston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	10.519872
11	12	2018-08-14 19:43:48.176029	DECISION TREE	Miami	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.577366
12	13	2018-08-14 19:44:00.243495	DECISION TREE	New York	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	6.767755
13	14	2018-08-14 19:44:07.609795	DECISION TREE	Atlanta	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.783687
14	15	2018-08-14 19:44:14.731752	DECISION TREE	Boston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	25.109055

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
15	16	2018-08-14 19:44:20.754761	DECISION TREE	Dallas	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	15.432839
16	17	2018-08-14 19:44:46.805353	DECISION TREE	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.556905
17	18	2018-08-14 19:48:11.736850	DECISION TREE	Houston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	10.205057

In [ ]:

Time Series to observe DAILY temperature variations

Daily temperature prediction using Random Forest

Going one step deeper than a tree, we use in the notebook below

Random Forests : We begin by loading all necessary libraries and paths to read the "pickles" as well as store image for the graph towards the end of our code. The pickles are read and the data is fed into an RF model.

Finally, we have two graphs showing the RF results vs. the fitted model as well as predicted results vs. actuals and test data

In [224]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Create a folder for every run of the Random forest to store images

In [225]:
```
city='Miami' # New_York Atlanta Boston Dallas Houston Miami
analysis_type = 'Enhanced_Signals' # Basic, Inc_Signals, Enhanced_Signals
```

In [226]:
```
EXPERIMENT_DIR, EXPERIMENT_ID  = create_results_perrun()
print(f"Experiment ID: {EXPERIMENT_ID}")
print(f"Path of the results directory:{EXPERIMENT_DIR}")
```

```
Experiment ID: 36
Path of the results directory:../experiment_results/RUN-36
```

Here we are importing the train and test Data from pickle files created through the EDA file

In [227]:
```
X_train = pd.read_pickle(f'{PICKLE_PATH}/X_train_{city}_{analysis_type}.pkl')
Y_train = pd.read_pickle(f'{PICKLE_PATH}/Y_train_{city}_{analysis_type}.pkl')

X_test  = pd.read_pickle(f'{PICKLE_PATH}/X_test_{city}_{analysis_type}.pkl')
Y_test  = pd.read_pickle(f'{PICKLE_PATH}/Y_test_{city}_{analysis_type}.pkl')

print("Shape of Training Dataset " , X_train.shape)
print("Shape of Testing Dataset " , X_test.shape)
```

```
Shape of Training Dataset  (1399, 17)
Shape of Testing Dataset  (90, 17)
```

In [228]:
```
# Fitting a decision tree regressor with max depth and n_estimators
max_depth = 8
n_estimators = 50
fitted_model = RandomForestRegressor(max_depth=max_depth, random_state=0, n_estimators=n_estimators)
fitted_model.fit(X_train, Y_train)

# Dataframe to show features and their importances
top_features = 10
features_importances_df= show_feature_importances(X_train.columns.values.tolist(),
                                                   fitted_model.feature_importances_,top_features)

# Top 10 features
features_importances_df.head(10)

# Store results
features_importances_df.to_csv(f'{EXPERIMENT_DIR}/feature_importances.csv')
```

In [229]:
```
# Run the model on the training dataset
Y_train_pred = fitted_model.predict(X_train)
# Calculate mean squared error for the predicted values
mse_train = mean_squared_error(Y_train, Y_train_pred)
print('Mean Squared Error for the training dataset: %.3f' % mse_train)
```

```
Mean Squared Error for the training dataset: 0.757
```

In [230]:
```
# Run the model on the testing dataset
Y_test_pred = fitted_model.predict(X_test)
# Calculate mean squared error for the test vs predicted values
mse_test = mean_squared_error(Y_test, Y_test_pred)
print('Mean Squared Error for the testing dataset: %.3f' % mse_test)
```

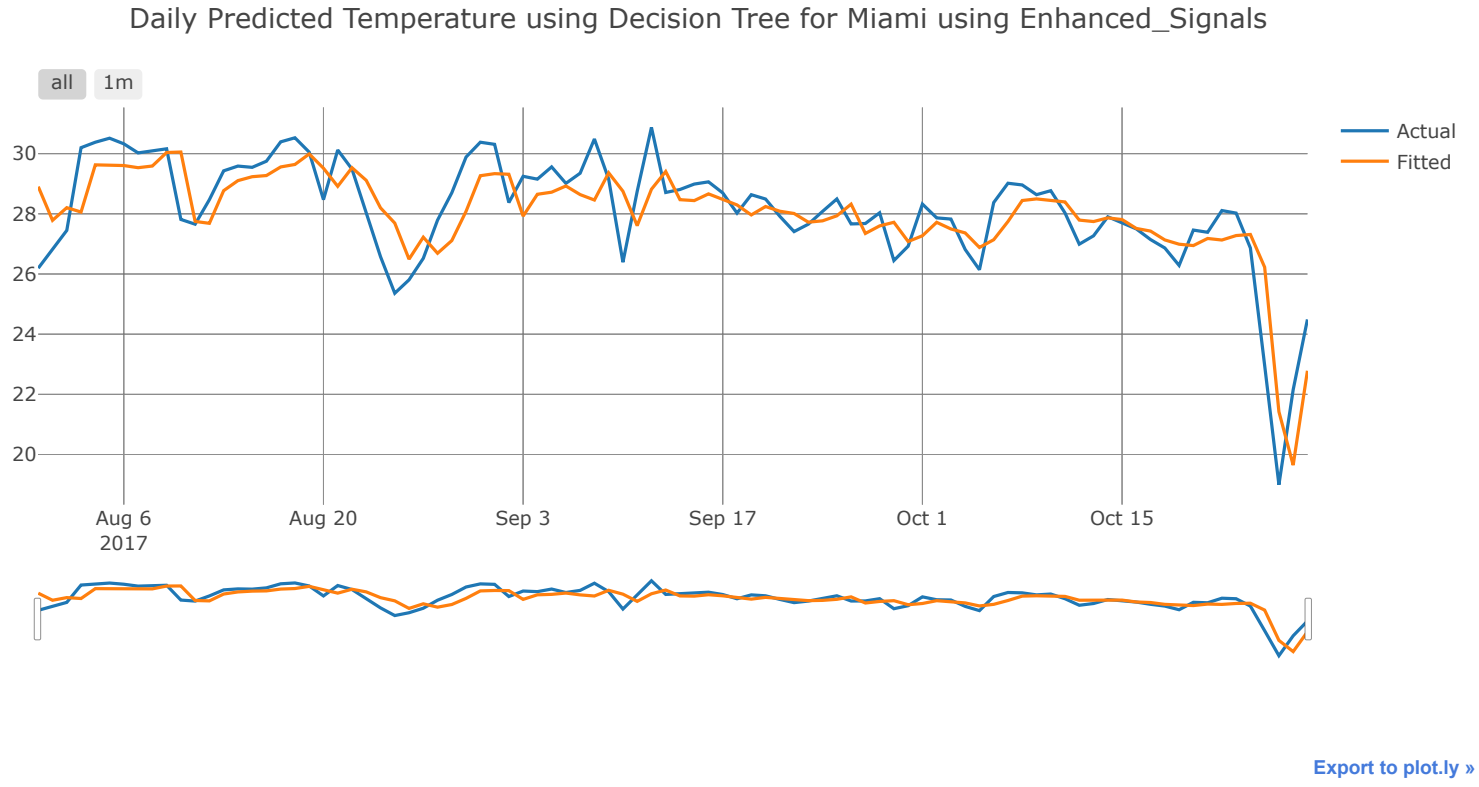
```
Mean Squared Error for the testing dataset: 1.169
```

In [231]:
```
# Creating a dataframe for predicted/fitted values
future_forecast = pd.DataFrame(Y_test_pred,index = Y_test.index,columns=['Fitted'])

# Concatenate the predicted/fitted values with actual values to display graphs
predictions = pd.concat([Y_test,future_forecast],axis=1)
predictions.columns = ["Actual","Fitted"]

# Displaying few of the predicted values
predictions.head(10)
```

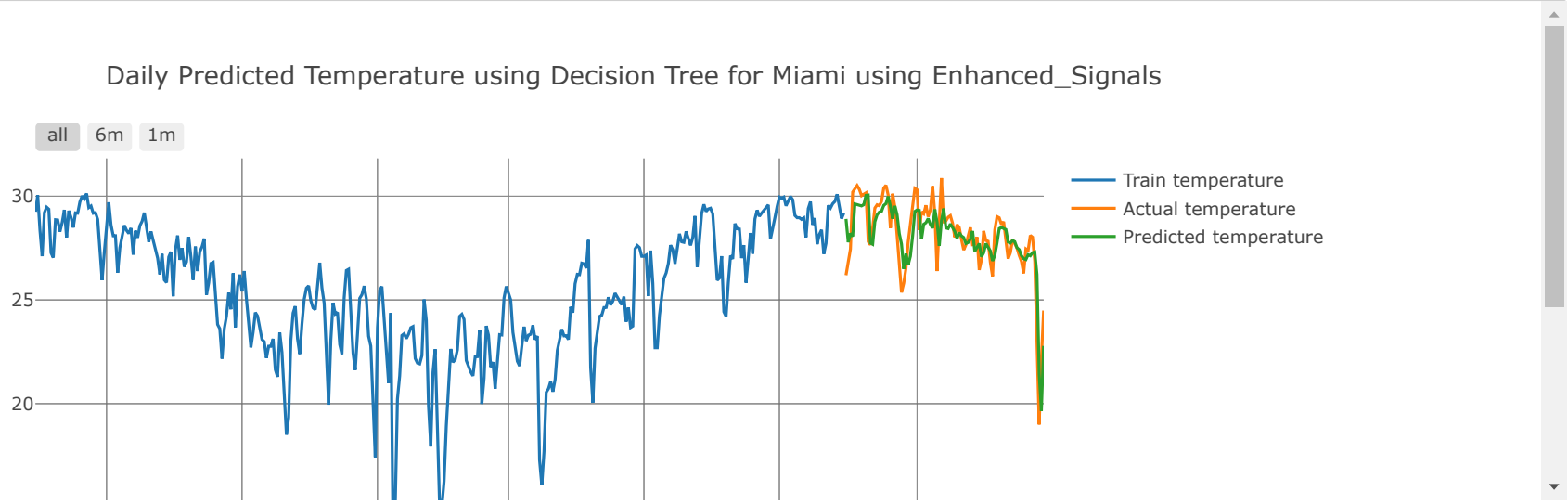
Out[231]:

datetime	Actual	Fitted
2017-07-31	26.192917	28.903332
2017-08-01	26.835000	27.783733
2017-08-02	27.449583	28.204148
2017-08-03	30.198333	28.059736
2017-08-04	30.380000	29.627431
2017-08-05	30.513333	29.596322
2017-08-06	30.326667	29.604831
2017-08-07	30.024583	29.532938
2017-08-08	30.094583	29.588812
2017-08-09	30.160833	30.038199

In [232]:
```
city = city.replace('_',' ')
# Plotting the daily predicted temperature vs Actual Temperature - Decision Tree
fig = charter_helper_fitted(f"Daily Predicted Temperature using Decision Tree for {city} using {analysis_type}", predictions)
iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_actual_vs_predict.png')
```



In [233]:
```
# Plotting the training data for past year, Actual/test data and predicted temperature - Decision Tree
fig = charter_helper_prediction(f"Daily Predicted Temperature using Decision Tree for {city} using {analysis_type}",
                     X_train,Y_train,X_test,Y_test,future_forecast)

iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_predict.png')
```



In [234]:
```
results = update_results_function(EXPERIMENT_ID, 'RANDOM FOREST', city, analysis_type,
                                  {'max_depth': max_depth , 'n_estimators':n_estimators, 'Info': X_train._metadata},
                                  {'features' : X_train.columns.values.tolist(),
                                   'importances':fitted_model.feature_importances_,
                                   'mse_train' : mse_train},
                                    mse_test)
```

In [235]:
```
results.tail(1)
```

Out[235]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
35	36	2018-08-14 19:56:51.638468	RANDOM FOREST	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	1.16879

In [236]:
```
results = pd.read_pickle('../pickles/results.pkl')
results
```

Out[236]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
0	1	2018-08-14 19:41:45.275297	DECISION TREE	New York	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	5.799872
1	2	2018-08-14 19:42:13.654060	DECISION TREE	Atlanta	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	8.436149
2	3	2018-08-14 19:42:23.328525	DECISION TREE	Boston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	16.504857
3	4	2018-08-14 19:42:30.715058	DECISION TREE	Dallas	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	12.523335
4	5	2018-08-14 19:42:41.506483	DECISION TREE	Houston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.705077
		2018-08-14	DECISION			DESKTOP-	{'max_depth': 8, 'Info':	{'features':	

In [ ]:

Time Series to observe DAILY temperature variations

Daily temperature prediction using RNN

Moving from predictive Machine Learning classifier to Unpredicitve Neural Nets, we use Sequential **Recurrent Neural Net (RNN)** in the notebook below.

You need to ensure that you have the right environment installed on top of your python3 to run Keras and Tensorflow. Two libraries needed to successfully run (RNN).

Just like in the other models, we begin by loading all necessary libraries and paths to read the "pickles" as well as store image for the graph towards the end of our code. The pickles are read and the data is fed into an RNN model. Finally, we have two graphs showing the DT results vs. the fitted model as well as predicted results vs. actuals and test data

Here we are importing the train and test Data from pickle files created through the EDA file

In [274]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Create a folder for every run of the RNN to store images

In [275]:
```
city='Miami' # New_York Atlanta Boston Dallas Houston Miami
analysis_type = 'Enhanced_Signals' # Basic, Inc_Signals, Enhanced_Signals
```

In [276]:
```
EXPERIMENT_DIR, EXPERIMENT_ID  = create_results_perrun()
print("Path of the results directory",EXPERIMENT_DIR )
```

```
Path of the results directory ../experiment_results/RUN-54
```

In [277]:
```
#EXPERIMENT_DIR = '../experiment_results/RUN-37'
#EXPERIMENT_ID = 37
```

In [278]:
```
X_train = pd.read_pickle(f'{PICKLE_PATH}/X_train_{city}_{analysis_type}.pkl')
Y_train = pd.read_pickle(f'{PICKLE_PATH}/Y_train_{city}_{analysis_type}.pkl')

X_test  = pd.read_pickle(f'{PICKLE_PATH}/X_test_{city}_{analysis_type}.pkl')
Y_test  = pd.read_pickle(f'{PICKLE_PATH}/Y_test_{city}_{analysis_type}.pkl')

print("Shape of Training Dataset " , X_train.shape)
print("Shape of Testing Dataset " , X_test.shape)
```

```
Shape of Training Dataset  (1399, 17)
Shape of Testing Dataset  (90, 17)
```

In [279]:
```
# Function to fit a sequential rnn with loss estimate being mean squared error
def train_model(X_train, y_train, X_test, y_test, epochs):
    model = Sequential(
        [
            Dense(10, activation="relu", input_shape=(X_train.shape[1],)),
            Dense(10, activation="relu"),
            Dense(10, activation="relu"),
            Dense(1, activation="linear")
        ]
    )
    model.compile(optimizer=Adam(lr=0.001), loss="mean_squared_error")

    history = model.fit(X_train, y_train, epochs=epochs, shuffle=False)
    return model, history
```

In [280]:
```
# Function to fit a sequential rnn with epochs = 50
epochs = 50
model_encoded, encoded_hist = train_model(
    X_train,
    Y_train,
    X_test,
    Y_test,
    epochs=epochs
)
```

```
Epoch 1/50
1399/1399 [==============================] - 1s 956us/step - loss: 780.4398
Epoch 2/50
1399/1399 [==============================] - 0s 74us/step - loss: 565.0433
Epoch 3/50
1399/1399 [==============================] - 0s 72us/step - loss: 346.5662
Epoch 4/50
1399/1399 [==============================] - 0s 84us/step - loss: 45.1367
Epoch 5/50
1399/1399 [==============================] - 0s 85us/step - loss: 7.4433
Epoch 6/50
1399/1399 [==============================] - 0s 78us/step - loss: 6.7492
Epoch 7/50
1399/1399 [==============================] - 0s 74us/step - loss: 6.6773
Epoch 8/50
1399/1399 [==============================] - 0s 65us/step - loss: 6.6082
Epoch 9/50
1399/1399 [==============================] - 0s 69us/step - loss: 6.5385
Epoch 10/50
```

In [281]:
```
# Run the model on the training dataset
Y_train_pred = model_encoded.predict(X_train)
# Calculate mean squared error for the predicted values
mse_train = mean_squared_error(Y_train, model_encoded.predict(X_train))
print('Mean Squared Error for the training dataset: %.3f' % mse_train)
```

```
Mean Squared Error for the training dataset: 3.901
```

In [282]:
```
# Run the model on the testing dataset
Y_test_pred = model_encoded.predict(X_test)
# Calculate mean squared error for the test vs predicted values
mse_test = mean_squared_error(Y_test, model_encoded.predict(X_test))
print('Mean Squared Error for the testing dataset: %.3f' % mse_test)
```

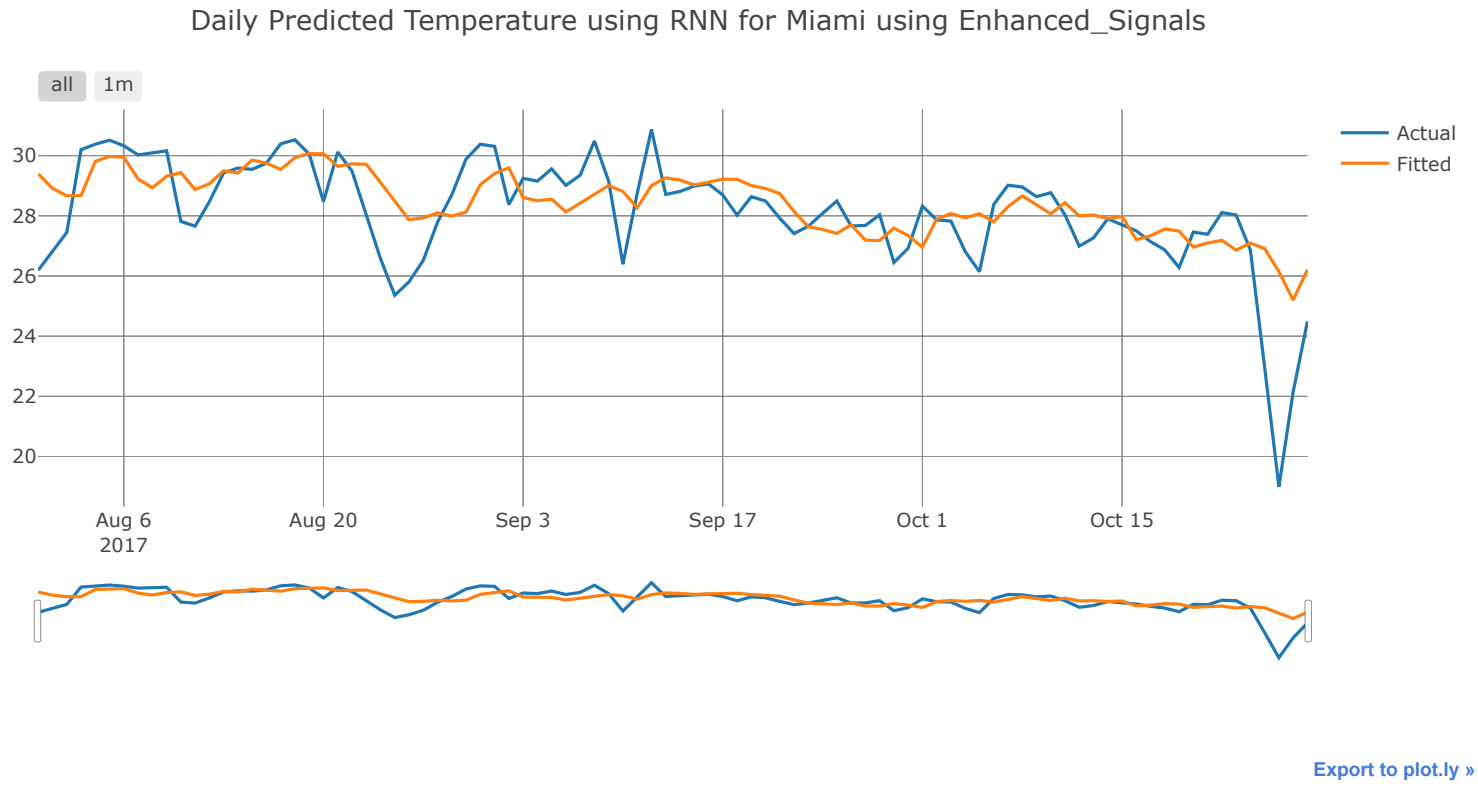
```
Mean Squared Error for the testing dataset: 2.013
```

In [283]:
```
# Creating a dataframe for predicted/fitted values
future_forecast = pd.DataFrame(Y_test_pred,index = Y_test.index,columns=['Fitted'])

# Concatenate the predicted/fitted values with actual values to display graphs
predictions = pd.concat([Y_test,future_forecast],axis=1)
predictions.columns = ["Actual","Fitted"]

# Displaying few of the predicted values
predictions.head(10)
```

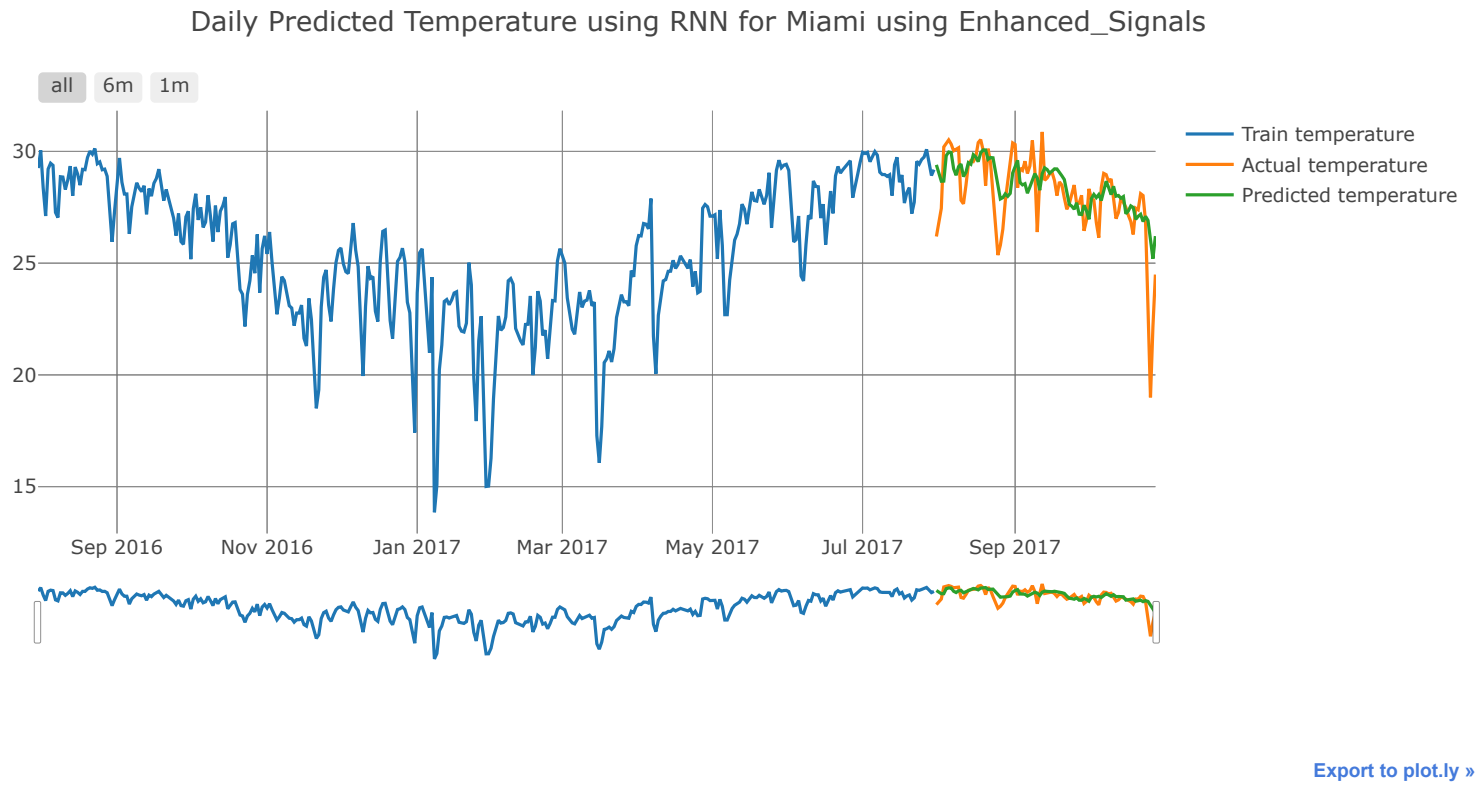
Out[283]:

datetime	Actual	Fitted
2017-07-31	26.192917	29.396700
2017-08-01	26.835000	28.911028
2017-08-02	27.449583	28.666040
2017-08-03	30.198333	28.672266
2017-08-04	30.380000	29.811632
2017-08-05	30.513333	29.973490
2017-08-06	30.326667	29.948656
2017-08-07	30.024583	29.224321
2017-08-08	30.094583	28.931959
2017-08-09	30.160833	29.316683

In [284]:
```
city = city.replace('_',' ')
# Plotting the daily predicted temperature vs Actual Temperature - RNN
fig = charter_helper_fitted(f"Daily Predicted Temperature using RNN for {city} using {analysis_type}", predictions)
iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_actual_vs_predict.png')
```



In [285]:
```
 # Plotting the training data for past year, Actual/test data and predicted temperature - Decision Tree
fig = charter_helper_prediction(f"Daily Predicted Temperature using RNN for {city} using {analysis_type}",
                      X_train,Y_train,X_test,Y_test,future_forecast)

iplot(fig)

py.image.save_as(fig, f'{EXPERIMENT_DIR}/Daily_predict.png')
```



In [286]:
```
results = update_results_function(EXPERIMENT_ID, 'RNN',city,analysis_type,
                                  {'epochs': epochs,'Info': X_train._metadata},
                                  {'features' : X_train.columns.values.tolist(),
                                   'importances':'NA',
                                   'mse_train' : mse_train},
                                    mse_test)
```

In [287]:
```
results.tail(1)
```

Out[287]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
53	54	2018-08-14 20:40:54.336628	RNN	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	2.01286

In [288]:
```
results = pd.read_pickle('../pickles/results.pkl')
results
```

Out[288]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
0	1	2018-08-14 19:41:45.275297	DECISION TREE	New York	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	5.799872
1	2	2018-08-14 19:42:13.654060	DECISION TREE	Atlanta	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	8.436149
2	3	2018-08-14 19:42:23.328525	DECISION TREE	Boston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	16.504857
3	4	2018-08-14 19:42:30.715058	DECISION TREE	Dallas	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	12.523335
4	5	2018-08-14 19:42:41.506483	DECISION TREE	Houston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.705077
		2018-08-14	DECISION			DESKTOP-	{'max_depth': 8, 'Info':	{'features':	

In [ ]:
```
```

Time Series to observe DAILY temperature variations

Daily temperature prediction using Facebook Prophet

For this analysis, to complement our analysis of models we turned to using an open-sourced library by Facebook called Prophet. https://facebook.github.io/prophet/ (https://facebook.github.io/prophet/)

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet is open source software released by Facebook's Core Data Science team.

We have take the original series as provided and using their documentation repeated our forecasting analysis.

In [3]:
```
import pandas as pd
from fbprophet import Prophet
```

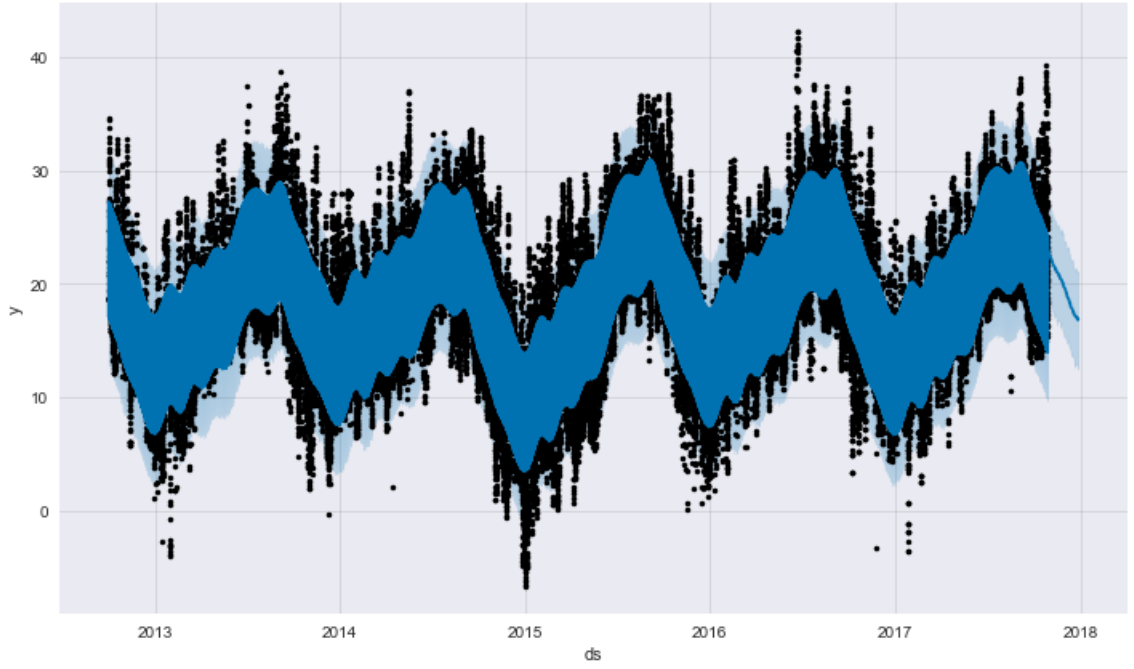
In [4]:
```
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('seaborn')

cities_df = pd.read_csv("../data/temperature.csv", parse_dates=['datetime'])
cities_df.head()

temp_df = cities_df.fillna(method = 'bfill', axis=0).dropna()
temp_df = temp_df.rename(columns={'Los Angeles': 'y', 'datetime': 'ds'})
temp_df['y'] = temp_df['y'] - 273.15
```

In [7]:
```
m = Prophet(changepoint_prior_scale=0.01)
m.fit(temp_df.loc[:,["ds","y"]])
```

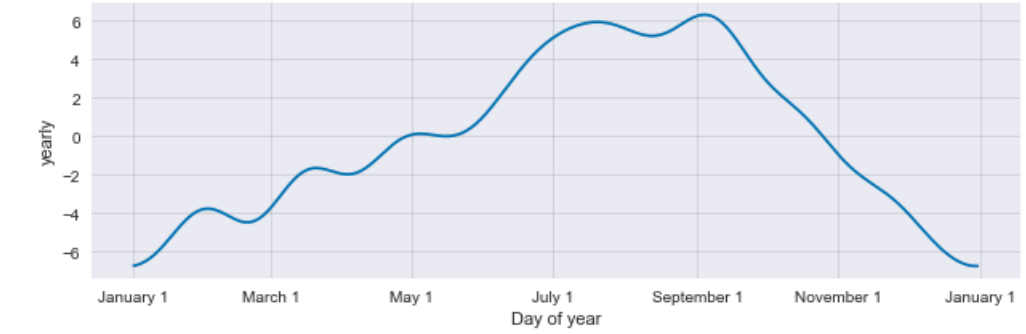
```
C:\ProgramData\Anaconda3\lib\site-packages\pystan\misc.py:399: FutureWarning: Conversion of the second argument of issubdtype from `float` t
o `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  elif np.issubdtype(np.asarray(v).dtype, float):
```

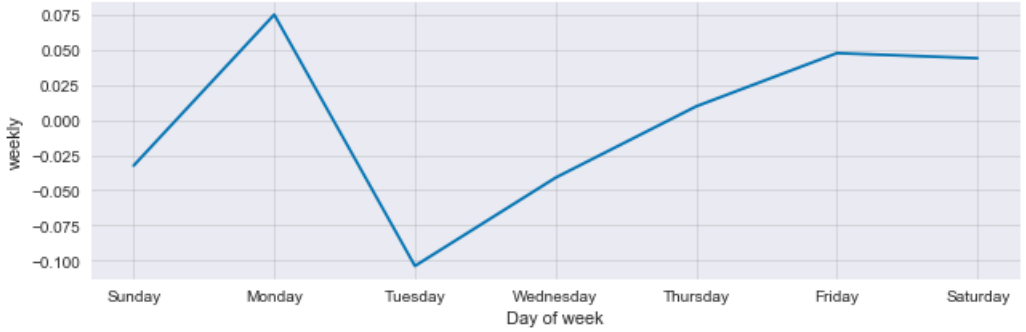
Out[7]:
```
<fbprophet.forecaster.Prophet at 0x129cfbef0>
```

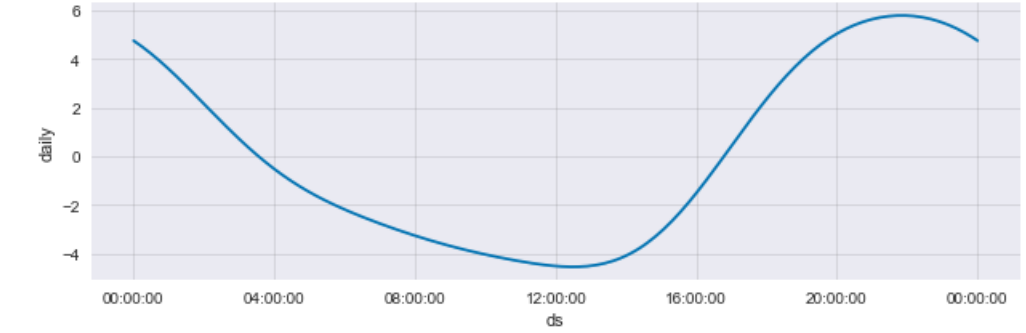
In [8]:
```
future = m.make_future_dataframe(periods=60, freq='D')

forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
fig1 = m.plot(forecast)
fig2 = m.plot_components(forecast)
```


40
30
20
10
0
y
2013
2014
2015
2016
2017
2018
ds


trend
19
18
17
16
15
2013
2014
2015
2016
2017
2018
ds


yearly
6
4
2
0
−2
−4
−6
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year


weekly
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week


daily
6
4
2
0
−2
−4
00:00:00
04:00:00
08:00:00
12:00:00
16:00:00
20:00:00
00:00:00
ds

Experiment Results

In [17]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Here we review the results from all of our experiments across the various ML algorithms we investigated and the datasets we created.

We provide a summary below with a link to each of the notebooks used for the experiments.

Experiment Summary

We created the following datasets as part of our experiment design

Link	Dataset	Cleaned + Lags/Moving Averages	Including Signals	Enhanced Signals
link (DataViewer.ipynb?DATA=Atlanta)	Atlanta	x	x	x
link (DataViewer.ipynb?DATA=Boston)	Boston	x	x	x
link (DataViewer.ipynb?DATA=Dallas)	Dallas	x	x	x
link (DataViewer.ipynb?DATA=Houston)	Houston	x	x	x
link (DataViewer.ipynb?DATA=New_York)	New York	x	x	x
link (DataViewer.ipynb?DATA=Miami)	Miami	x	x	x

We created the following supplementary signals that where used to enhance the datasets above

Link	Dataset	Cleaned	Enhanced with Lags/Moving Averages	Reference
link (SignalViewer.ipynb?DATA=AO)	AO (Artic Oscillation)	x	x	x
link (SignalViewer.ipynb?DATA=NAO)	NAO (North American Oscilliation)	x	x	x
link (SignalViewer.ipynb?DATA=NINO3)	NINO3	x	x	x
link (SignalViewer.ipynb?DATA=NINO4)	NINO4	x	x	x
link (SignalViewer.ipynb?DATA=NINO12)	NINO1/2	x	x	x
link (SignalViewer.ipynb?DATA=NINO34)	NINO3/4	x	x	x

We also prepared other signals and locations to consider as part of our experimental design. We however limited ourselves to these datasets due to time constraints.

We ran the following models as part of our experiment design, these notebooks we used repeatedly to run each variant of the experiments and we included hooks to track the artifacts and results.

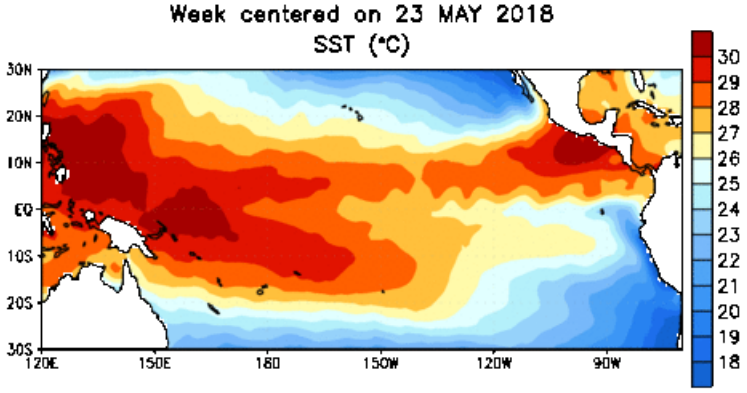
Link	Experiments

Link	Experiments
link (Daily_Temp_Analysis_ARIMA.ipynb)	ARIMA statistical models and techniques
link (Daily_Temp_Analysis_DT.ipynb)	Decision Tree
link (Daily_Temp_Analysis_RF.ipynb)	Random Forest
link (Daily_Temp_Analysis_RNN.ipynb)	Sequential Recurrent Neural Net

We systematically collected and archived all of the data created by each of the experiments and saved this assigning each experiment its own unique identifier. We also recorded who/when the experiment was run. We also collected various artifacts created in each experiement so we could repeat the experiment at a later date should it be required.

This diagram shows the NINO area of the ocean (courtesy of Climate Prediction Center, National Weather Service) and it shows how the temperature of the ocean is localized.

In [70]:
```
%%HTML
<img src="http://www.cpc.ncep.noaa.gov/products/analysis_monitoring/enso_update/sstanim.gif"></img>
```



Summary of results by location

Here is some quick links to our results

Link	Experiments Results for cities
link (Experiment_Results-Atlanta.ipynb)	Atlanta
link (Experiment_Results-Boston.ipynb)	Boston
link (Experiment_Results-Dallas.ipynb)	Dallas
link (Experiment_Results-Houston.ipynb)	Houston
link (Experiment_Results-New_York.ipynb)	New York
link (Experiment_Results-Miami.ipynb)	Miami

We have provided some pivot tables and charts to allow the reader to see the results of our experiments. We will focus initially on the Mean Squared Error metric as a means of illustrating how each model performed.

ARIMA has been excluded from these comparisons as we were unable to create a forecast that would extend any more than 6 days before the signal reverted to the mean trend.

On investigation, we would need to use an alternative model called SARIMAX which would allow us to extend the forecasts over a longer time period and thus we would be able to make a meaningful comparison.

In [4]:
```
# All runs presented in pivot table
results_df = get_results()
results_df.pivot_table(index=['CITY'], columns=['MODEL_NAME','FEATURE_TYPE'], values="MEAN_SQUARED_ERROR")
```

Out[4]:

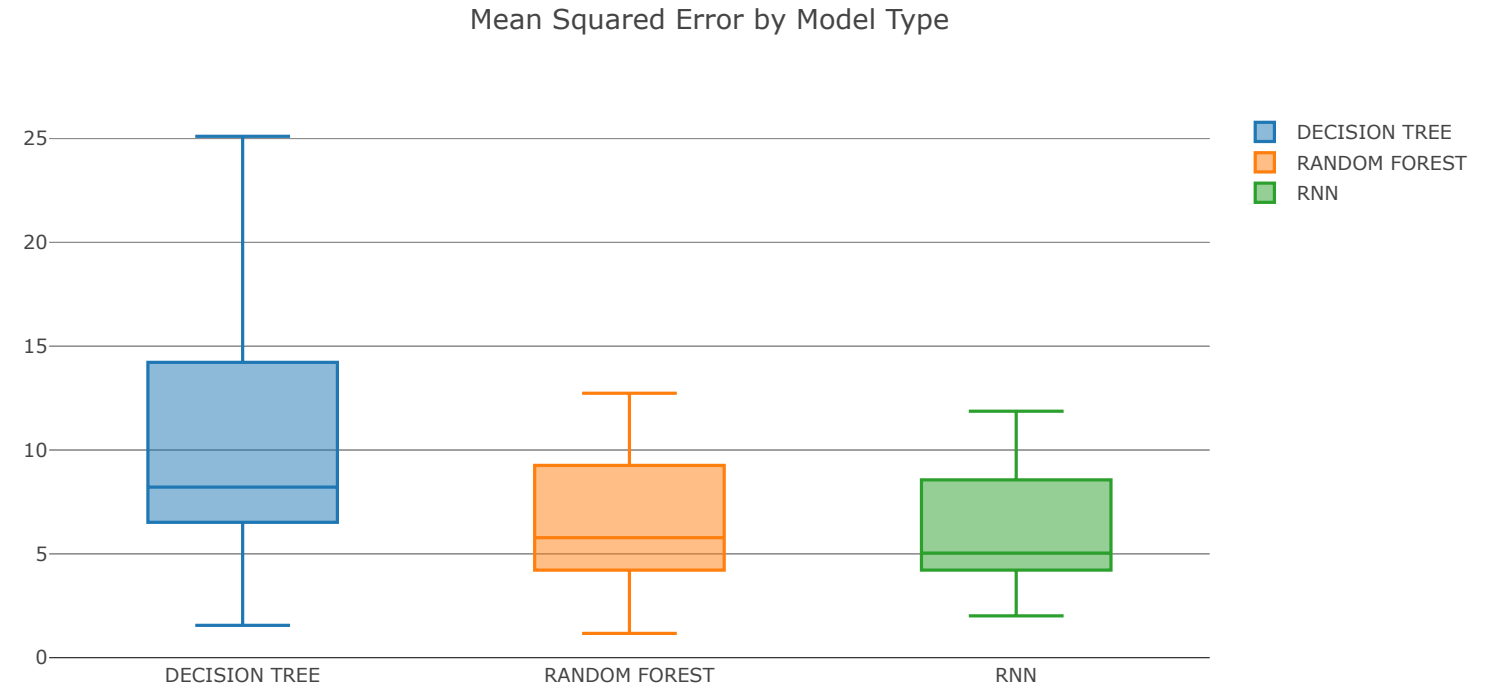
MODEL_NAME	DECISION TREE			RANDOM FOREST			RNN		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
CITY									
Atlanta	8.436149	7.783687	7.992946	5.516615	5.434763	5.434763	5.105819	4.966304	5.628276
Boston	16.504857	25.109055	22.967065	11.877780	12.735153	12.735153	11.869203	11.645510	11.682041
Dallas	12.523335	15.432839	14.223161	8.983822	9.260334	9.260334	8.563057	8.632428	8.336575
Houston	7.705077	10.205057	10.519872	6.637421	6.041626	6.041626	5.581542	4.217506	4.588875
Miami	1.833555	1.556905	1.577366	1.206302	1.168790	1.168790	2.016882	2.012860	2.093093
New York	5.799872	6.767755	6.518000	4.369905	4.216172	4.216172	4.839433	4.095801	4.700272

As we can see above, the enhanced signals appear in some instances to improve the forecasts. However this is not always the case, this could be because the cluster of signals choosen should be selected specifically for each location vs. just applying the same data set to each.

We will also review those features flagged in each city's analysis to determine the top 10 features used by the models.

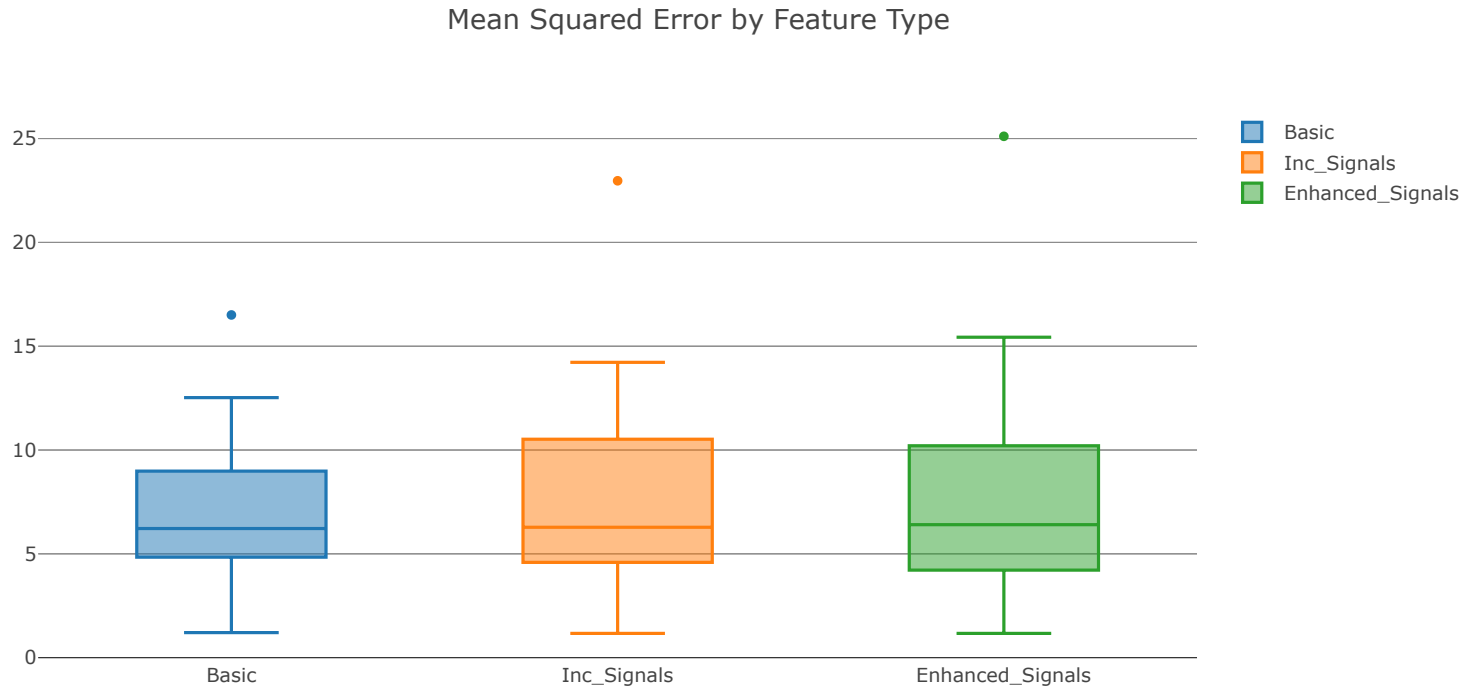
Let's use some boxplots to review the performance of our models over all of our experiments

In [19]:
```
# All runs presented in pivot table
create_boxplot_traces_for_results(results_df,'MODEL_NAME','MEAN_SQUARED_ERROR',"Mean Squared Error by Model Type")
```



As expected the **RNN** performs the best, **Random forest** is the second, **Decision Tree** is the last

In [20]:
```
# All runs presented in pivot table
create_boxplot_traces_for_results(results_df,'FEATURE_TYPE','MEAN_SQUARED_ERROR',"Mean Squared Error by Feature Type")
```



In contrary to what we saw above the signals are not having a big of an effect on our prediction, this may be because we are missing signals or we have not tuned the hyper parameters of the models enough

Conclusion

The Arima model was unsuitable for forecasting as detailed in summary switching to a more advanced model for example SARIMAX or THETA would help us to forecast for a longer period of time

We also incorporated more signals like lags which is signal 1,2,7,30,90 and 365 days ago and moving averages which is signal averaged over 1 week, 30 days, 60 days etc from alternate locations (more locations for SST) and more weather measurement types like AO (Arctic Oscillation), NAO (North Atlantic Oscillation) and their lags and moving averages, also removed signals that are found to have no predictive power.

As per the results detailed in summary RNN performs better of all models and we intend to use LSTM model as a progression from our sequential RNN

Takeaways

- We used GIT as the version control tool, as we reached come to the end of the project we realised being organized with git came really handy with code merges and large csv or pickle file checkins.
- Serializing data, capturing results(pickle files) and images as we ran multiple experiments provided indispensible which helped us to quickly change artifacts / notebooks as we wanted to answer new questions or compare results.
- Notebooks is a great tool for quick analysis but are less efficient when you have to perform multiple iterations with different input parameters and datasets, A hybrid approach is recommended.
- We needed more time to tune our hyperparameters using grid searching or any other approaches.
- Now that we have good amount of results, It would be a good time to get the results reviewed by a domain expert so that we can identify weaknesses in our approach towards data and redefine our next steps, for example adding more signals or removing unnecessary signals, tune hyperparameters of our models.
- We could scale our data and no of experiments if we have faster compute resources.
- We would like to investigate a framework (i.e. MLFlow) or similiar to help manage the end to end process of data capture, running experiences, instrumentation and presenting our results.

Experiment Results

Results for Atlanta

In [17]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results for Atlanta, ordered by Mean Square Error (ascending)

In [5]:
```
city='Atlanta'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

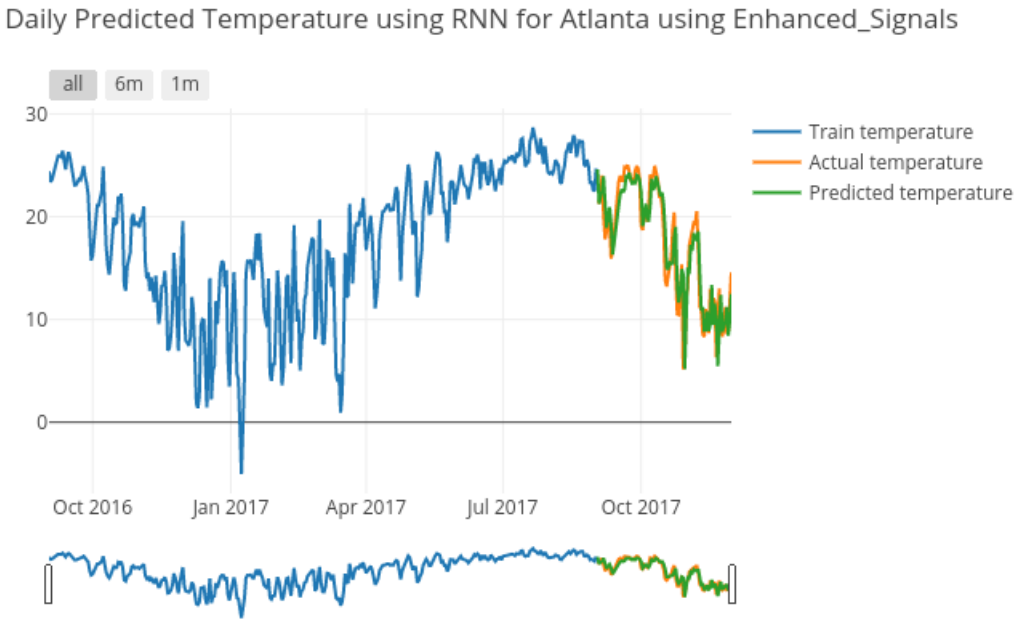
Out[5]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	50	2018-08-14 20:39:13.461675	RNN	Atlanta	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	4.966304
2	38	2018-08-14 20:34:21.973872	RNN	Atlanta	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	5.105819
3	26	2018-08-14 19:55:31.041998	RANDOM FOREST	Atlanta	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	5.434763
4	32	2018-08-14 19:56:23.951216	RANDOM FOREST	Atlanta	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	5.434763
5	20	2018-08-14 19:54:40.149363	RANDOM FOREST	Atlanta	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	5.516615
6	44	2018-08-14 20:36:38.538124	RNN	Atlanta	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	5.628276
7	14	2018-08-14 19:44:07.609795	DECISION TREE	Atlanta	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.783687
8	8	2018-08-14 19:43:12.930159	DECISION TREE	Atlanta	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.992946
9	2	2018-08-14 19:42:13.654060	DECISION TREE	Atlanta	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	8.436149

RNN has done better than Random forest but the results are close so there is no clear winner, lets take a look at that chart.

In [6]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #50 - run by DESKTOP-KN40C32 on 2018-08-14 20:39:13.461675



We can see the forecast follows the trend and it does a better job vs the prior example of Miami with respect to following the peaks and troughs of the actual temperature pattern experiences.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

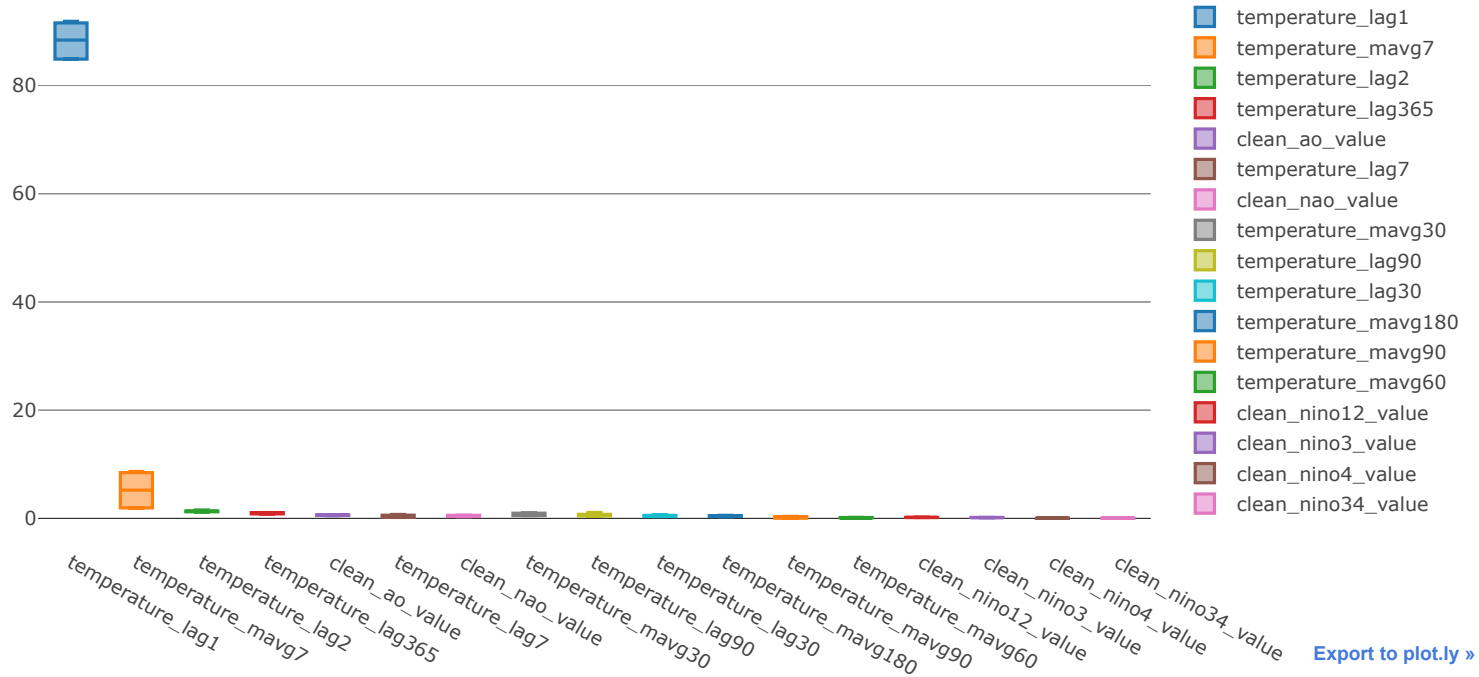
In [7]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[7]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.484511	0.586696	NaN	0.667402	0.667402
clean_nao_value	NaN	0.387084	0.319784	NaN	0.570843	0.570843
clean_nino12_value	NaN	0.176243	0.171550	NaN	NaN	NaN
clean_nino34_value	NaN	0.014362	0.075990	NaN	NaN	NaN
clean_nino3_value	NaN	0.155625	0.162820	NaN	NaN	NaN
clean_nino4_value	NaN	0.050394	0.000808	NaN	NaN	NaN
temperature_lag1	91.866531	91.563345	91.569282	85.242322	84.878510	84.878510
temperature_lag2	1.239663	1.220060	1.194826	1.553500	1.365561	1.365561
temperature_lag30	0.551914	0.641604	0.527405	0.532966	0.353330	0.353330
temperature_lag365	1.069122	0.947275	0.986347	1.080280	0.836627	0.836627
temperature_lag7	0.186437	0.173366	0.185205	0.730078	0.593266	0.593266
temperature_lag90	1.104324	0.751362	0.739947	0.587154	0.454484	0.454484
temperature_mavg180	0.570359	0.293828	0.448047	0.404703	NaN	NaN
temperature_mavg30	0.980080	0.966142	0.957687	0.671558	0.515406	0.515406
temperature_mavg60	0.107090	0.180291	0.096090	NaN	NaN	NaN
temperature_mavg7	1.944451	1.978938	1.905487	8.675354	8.460200	8.460200
temperature_mavg90	0.380028	0.015570	0.072030	0.274919	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [19]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [10]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[10]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.601503
clean_nao_value	0.462139
clean_nino12_value	0.173896
clean_nino34_value	0.045176
clean_nino3_value	0.159222
clean_nino4_value	0.025601
temperature_lag1	88.333084
temperature_lag2	1.323195
temperature_lag30	0.493425
temperature_lag365	0.959380
temperature_lag7	0.410270
temperature_lag90	0.681959
temperature_mavg180	0.429234
temperature_mavg30	0.767713
temperature_mavg60	0.127823
temperature_mavg7	5.237438
temperature_mavg90	0.185637

Now lets review the top 5 only

In [11]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[11]:

FEATURE	mean
temperature_lag1	88.333084
temperature_mavg7	5.237438
temperature_lag2	1.323195
temperature_lag365	0.959380
temperature_mavg30	0.767713

In [ ]:

Experiment Results

Results for Boston

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results, ordered by Mean Square Error (ascending)

In [3]:
```
city='Boston'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

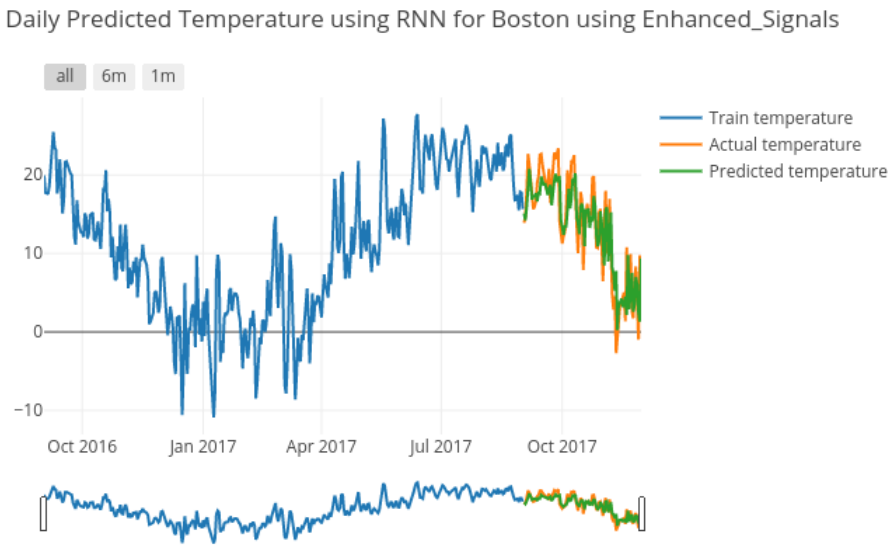
Out[3]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	51	2018-08-14 20:39:52.178626	RNN	Boston	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	11.645510
2	45	2018-08-14 20:37:08.003250	RNN	Boston	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	11.682041
3	39	2018-08-14 20:34:39.550435	RNN	Boston	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	11.869203
4	21	2018-08-14 19:54:48.599391	RANDOM FOREST	Boston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	11.877780
5	27	2018-08-14 19:55:39.652542	RANDOM FOREST	Boston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	12.735153
6	33	2018-08-14 19:56:30.584670	RANDOM FOREST	Boston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	12.735153
7	3	2018-08-14 19:42:23.328525	DECISION TREE	Boston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	16.504857
8	9	2018-08-14 19:43:20.383683	DECISION TREE	Boston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	22.967065
9	15	2018-08-14 19:44:14.731752	DECISION TREE	Boston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	25.109055

RNN has done better than Random forest but the results are close so there is no clear winner, lets take a look at that chart.

In [4]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #51 - run by DESKTOP-KN40C32 on 2018-08-14 20:39:52.178626



We can see the forecast follows the trend and it does a better job but like the prior example of Miami it has some difficulty following the peaks and troughs of the actual temperature pattern experiences.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

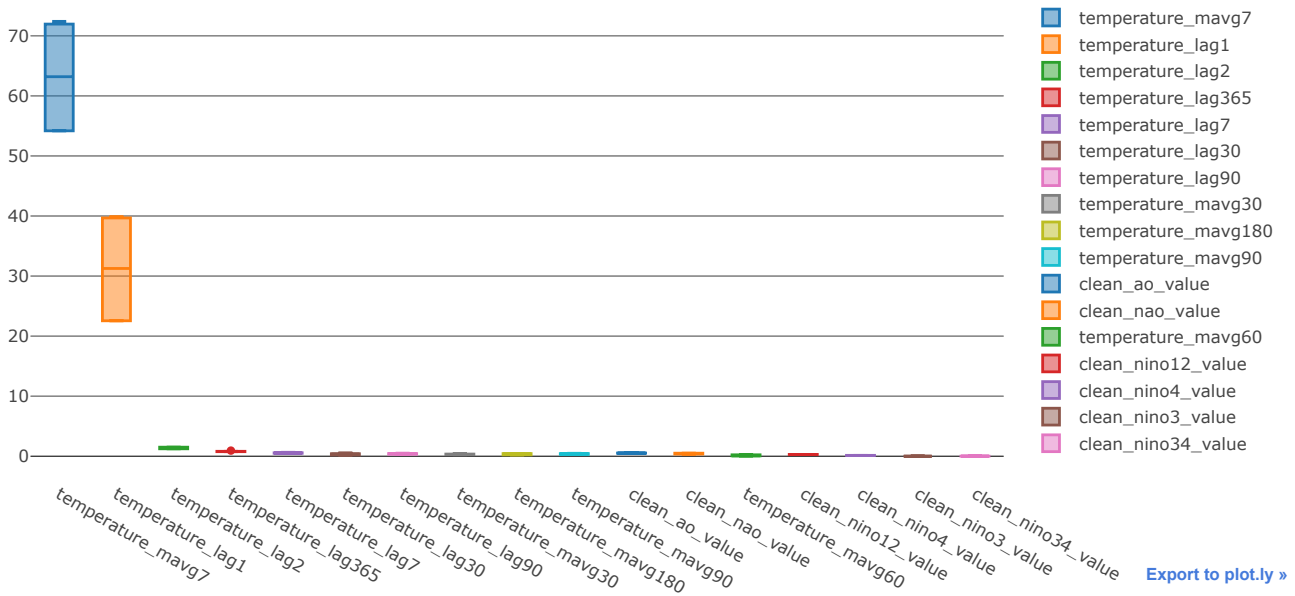
In [5]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[5]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.494070	0.547344	NaN	0.519692	0.519692
clean_nao_value	NaN	0.365090	0.362958	NaN	0.509393	0.509393
clean_nino12_value	NaN	0.287805	0.302207	NaN	NaN	NaN
clean_nino34_value	NaN	0.021185	0.006980	NaN	NaN	NaN
clean_nino3_value	NaN	0.012564	0.012307	NaN	NaN	NaN
clean_nino4_value	NaN	0.146795	0.132922	NaN	NaN	NaN
temperature_lag1	22.853599	22.547786	22.555122	39.903638	39.676008	39.676008
temperature_lag2	1.512019	1.278805	1.293871	1.508032	1.242142	1.242142
temperature_lag30	0.261568	0.202941	0.197705	0.572096	0.415173	0.415173
temperature_lag365	0.821443	0.820219	0.772308	0.938950	0.762632	0.762632
temperature_lag7	0.475197	0.497790	0.543488	0.657224	0.573611	0.573611
temperature_lag90	0.296958	0.351368	0.361960	0.514342	0.442467	0.442467
temperature_mavg180	0.472520	0.218633	0.242651	0.417609	NaN	NaN
temperature_mavg30	0.255383	0.256516	0.265126	0.472850	0.361452	0.361452
temperature_mavg60	0.291385	0.051112	0.079570	NaN	NaN	NaN
temperature_mavg7	72.406163	71.954276	71.946289	54.442588	54.181522	54.181522
temperature_mavg90	0.353767	0.493047	0.377192	0.295478	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [6]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [7]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[7]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.520199
clean_nao_value	0.436709
clean_nino12_value	0.295006
clean_nino34_value	0.014083
clean_nino3_value	0.012436
clean_nino4_value	0.139858
temperature_lag1	31.202027
temperature_lag2	1.346168
temperature_lag30	0.344109
temperature_lag365	0.813031
temperature_lag7	0.553487
temperature_lag90	0.401594
temperature_mavg180	0.337853
temperature_mavg30	0.328797
temperature_mavg60	0.140689
temperature_mavg7	63.185393
temperature_mavg90	0.379871

Now lets review the top 5 only

In [8]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[8]:

FEATURE	mean
temperature_mavg7	63.185393
temperature_lag1	31.202027
temperature_lag2	1.346168
temperature_lag365	0.813031
temperature_lag7	0.553487

This is different than the prior cases as we see the moving average taking a lead in the predictions, vs. the temperature of the prior day.

Experiment Results

Results for Dallas

In [8]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results, ordered by Mean Square Error (ascending)

In [2]:
```
city='Dallas'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

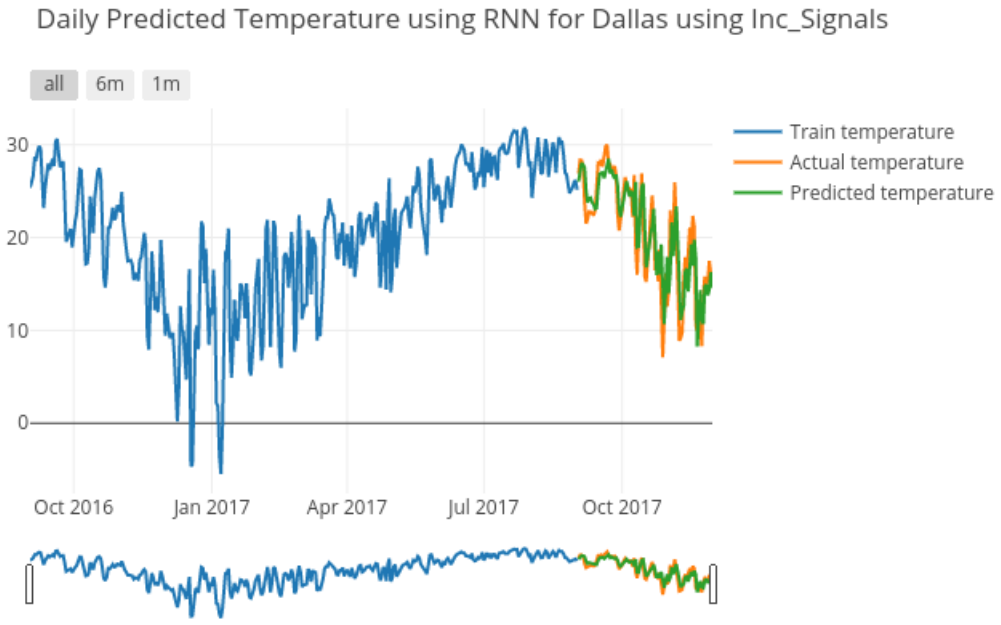
Out[2]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	46	2018-08-14 20:37:30.002193	RNN	Dallas	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	8.336575
2	40	2018-08-14 20:34:55.307517	RNN	Dallas	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	8.563057
3	52	2018-08-14 20:40:06.720627	RNN	Dallas	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	8.632428
4	22	2018-08-14 19:54:57.660172	RANDOM FOREST	Dallas	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	8.983822
5	28	2018-08-14 19:55:47.002009	RANDOM FOREST	Dallas	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	9.260334
6	34	2018-08-14 19:56:37.269205	RANDOM FOREST	Dallas	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	9.260334
7	4	2018-08-14 19:42:30.715058	DECISION TREE	Dallas	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	12.523335
8	10	2018-08-14 19:43:32.825549	DECISION TREE	Dallas	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	14.223161
9	16	2018-08-14 19:44:20.754761	DECISION TREE	Dallas	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	15.432839

RNN has done better than Random forest but the results are close so there is no clear winner, lets take a look at that chart.

In [3]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #46 - run by DESKTOP-KN40C32 on 2018-08-14 20:37:30.002193



We can see the forecast follows the trend and it does a better job but like the prior example of Miami it has some difficulty following the peaks and troughs of the actual temperature pattern experiences.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

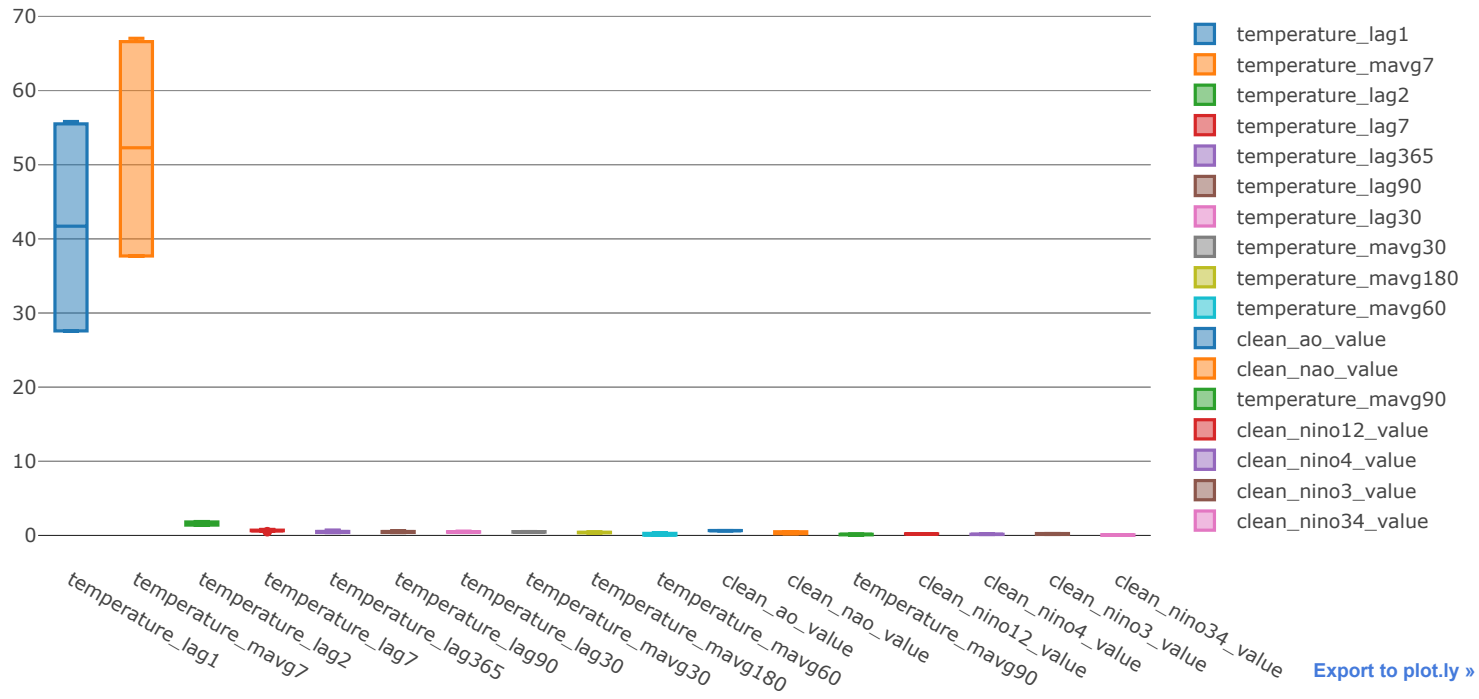
In [4]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[4]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.651503	0.656238	NaN	0.658562	0.658562
clean_nao_value	NaN	0.226906	0.249332	NaN	0.499492	0.499492
clean_nino12_value	NaN	0.215984	0.212701	NaN	NaN	NaN
clean_nino34_value	NaN	0.006323	0.095549	NaN	NaN	NaN
clean_nino3_value	NaN	0.257658	0.133802	NaN	NaN	NaN
clean_nino4_value	NaN	0.125707	0.157822	NaN	NaN	NaN
temperature_lag1	27.909283	27.582810	27.584224	55.826530	55.508373	55.508373
temperature_lag2	1.816951	1.397116	1.382537	1.875478	1.629844	1.629844
temperature_lag30	0.497862	0.437565	0.403289	0.583795	0.477568	0.477568
temperature_lag365	0.516834	0.400962	0.375246	0.728918	0.538135	0.538135
temperature_lag7	0.503527	0.638941	0.711708	0.814543	0.634003	0.634003
temperature_lag90	0.530253	0.401933	0.383665	0.635825	0.522790	0.522790
temperature_mavg180	0.453587	0.378973	0.344204	0.405867	NaN	NaN
temperature_mavg30	0.486033	0.516936	0.514952	0.526882	0.447634	0.447634
temperature_mavg60	0.181417	0.044381	0.043627	0.373975	NaN	NaN
temperature_mavg7	67.048224	66.559364	66.600341	38.009765	37.689387	37.689387
temperature_mavg90	0.056029	0.156939	0.150763	NaN	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [9]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [6]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[6]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.656216
clean_nao_value	0.368805
clean_nino12_value	0.214342
clean_nino34_value	0.050936
clean_nino3_value	0.195730
clean_nino4_value	0.141765
temperature_lag1	41.653265
temperature_lag2	1.621962
temperature_lag30	0.479608
temperature_lag365	0.516371
temperature_lag7	0.656121
temperature_lag90	0.499543
temperature_mavg180	0.395658
temperature_mavg30	0.490012
temperature_mavg60	0.160850
temperature_mavg7	52.266078
temperature_mavg90	0.121244

Now lets review the top 5 only

In [7]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[7]:

FEATURE	mean
temperature_mavg7	52.266078
temperature_lag1	41.653265
temperature_lag2	1.621962
clean_ao_value	0.656216
temperature_lag7	0.656121

This is different than the prior cases as we see the moving average taking the lead in the predictions.

In [ ]:

Experiment Results

Results for Houston

In [8]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results, ordered by Mean Square Error (ascending)

In [2]:
```
city='Houston'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

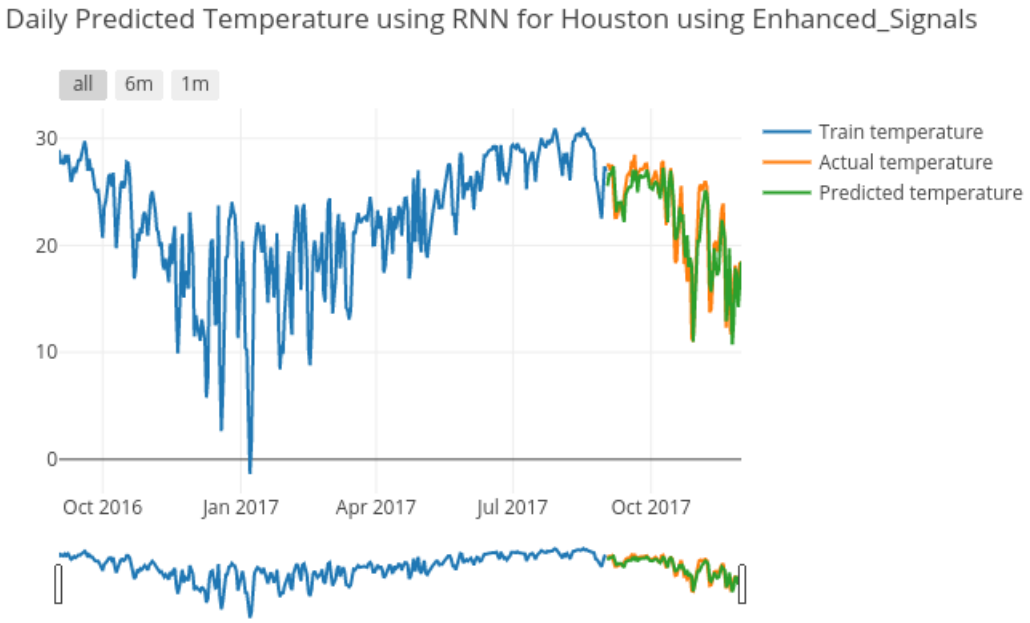
Out[2]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	53	2018-08-14 20:40:23.868627	RNN	Houston	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	4.217506
2	47	2018-08-14 20:37:47.827488	RNN	Houston	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	4.588875
3	41	2018-08-14 20:35:12.061515	RNN	Houston	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	5.581542
4	29	2018-08-14 19:55:55.798404	RANDOM FOREST	Houston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	6.041626
5	35	2018-08-14 19:56:44.111200	RANDOM FOREST	Houston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	6.041626
6	23	2018-08-14 19:55:04.570078	RANDOM FOREST	Houston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	6.637421
7	5	2018-08-14 19:42:41.506483	DECISION TREE	Houston	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	7.705077
8	18	2018-08-14 19:48:11.736850	DECISION TREE	Houston	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	10.205057
9	11	2018-08-14 19:43:39.374706	DECISION TREE	Houston	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	10.519872

RNN has done much better than Random forest so in this instance it is the winner, lets take a look at that chart.

In [3]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #53 - run by DESKTOP-KN40C32 on 2018-08-14 20:40:23.868627



We can see the forecast follows the trend and it does a better job but like the prior example of Miami it has some difficulty following the peaks and troughs of the actual temperature pattern experiences.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

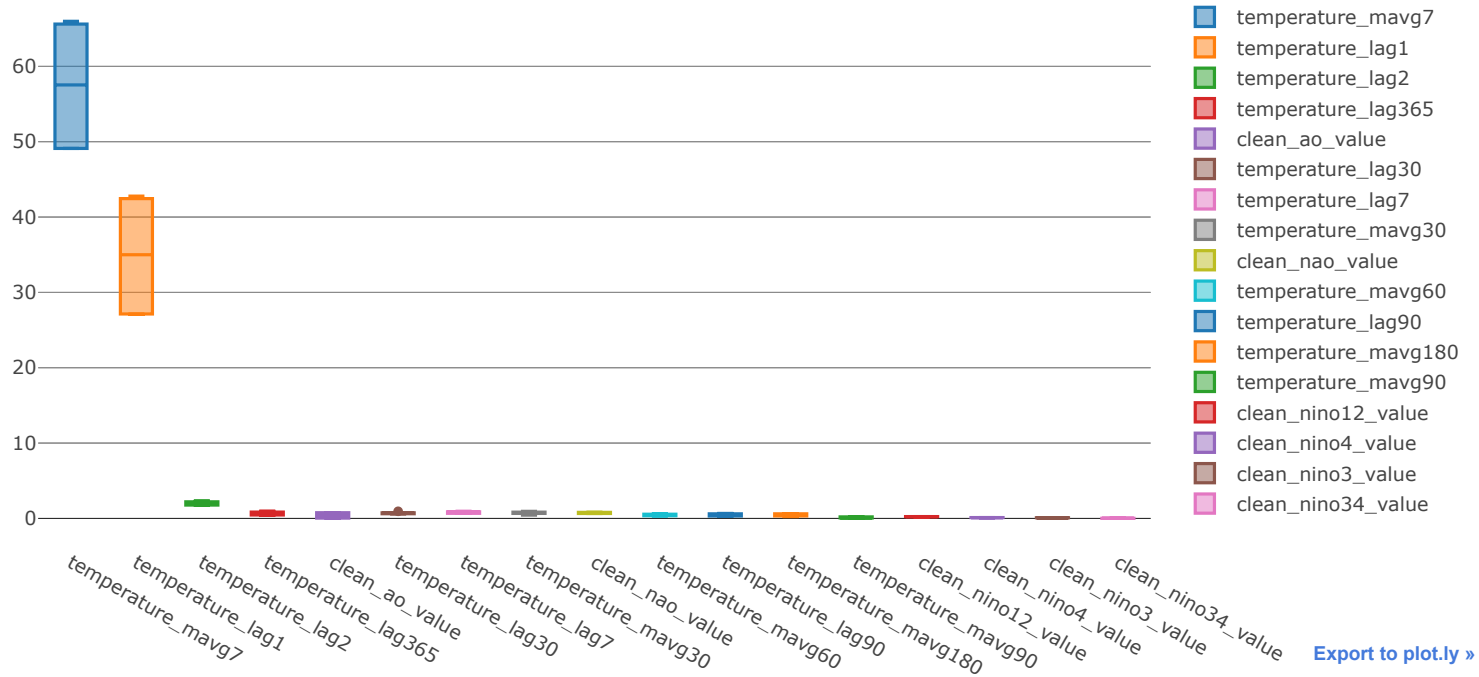
In [4]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[4]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.043560	0.041918	NaN	0.756749	0.756749
clean_nao_value	NaN	0.816215	0.794028	NaN	0.656157	0.656157
clean_nino12_value	NaN	0.235922	0.244526	NaN	NaN	NaN
clean_nino34_value	NaN	0.003483	0.018493	NaN	NaN	NaN
clean_nino3_value	NaN	0.087301	0.090106	NaN	NaN	NaN
clean_nino4_value	NaN	0.109168	0.112347	NaN	NaN	NaN
temperature_lag1	27.548477	27.136183	27.082286	42.775543	42.447002	42.447002
temperature_lag2	2.202069	1.773779	1.774231	2.347156	2.064882	2.064882
temperature_lag30	0.609669	0.646106	0.621661	0.944930	0.742389	0.742389
temperature_lag365	0.753962	0.440651	0.455842	0.975389	0.840931	0.840931
temperature_lag7	0.954081	0.859909	0.875849	0.863763	0.675088	0.675088
temperature_lag90	0.662133	0.330657	0.305711	0.556063	NaN	NaN
temperature_mavg180	0.241951	0.605832	0.616706	0.407410	NaN	NaN
temperature_mavg30	0.437132	0.722921	0.940497	0.831117	0.664705	0.664705
temperature_mavg60	0.558999	0.354695	0.328816	0.598374	0.469204	0.469204
temperature_mavg7	65.979684	65.591639	65.612385	49.480824	49.105328	49.105328
temperature_mavg90	0.051843	0.241981	0.084600	NaN	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [5]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [6]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[6]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.399744
clean_nao_value	0.730639
clean_nino12_value	0.240224
clean_nino34_value	0.010988
clean_nino3_value	0.088704
clean_nino4_value	0.110757
temperature_lag1	34.906082
temperature_lag2	2.037833
temperature_lag30	0.717857
temperature_lag365	0.717951
temperature_lag7	0.817296
temperature_lag90	0.463641
temperature_mavg180	0.467975
temperature_mavg30	0.710179
temperature_mavg60	0.463216
temperature_mavg7	57.479198
temperature_mavg90	0.126141

Now lets review the top 5 only

In [7]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[7]:

FEATURE	mean
temperature_mavg7	57.479198
temperature_lag1	34.906082
temperature_lag2	2.037833
temperature_lag7	0.817296
clean_nao_value	0.730639

This is different than the prior cases as the moving average takes a clear lead in the predictions vs. the prior lags of temperature.

In [ ]:

Experiment Results

Results for Miami

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results for Miami, ordered by Mean Square Error (ascending)

In [125]:
```
city='Miami'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

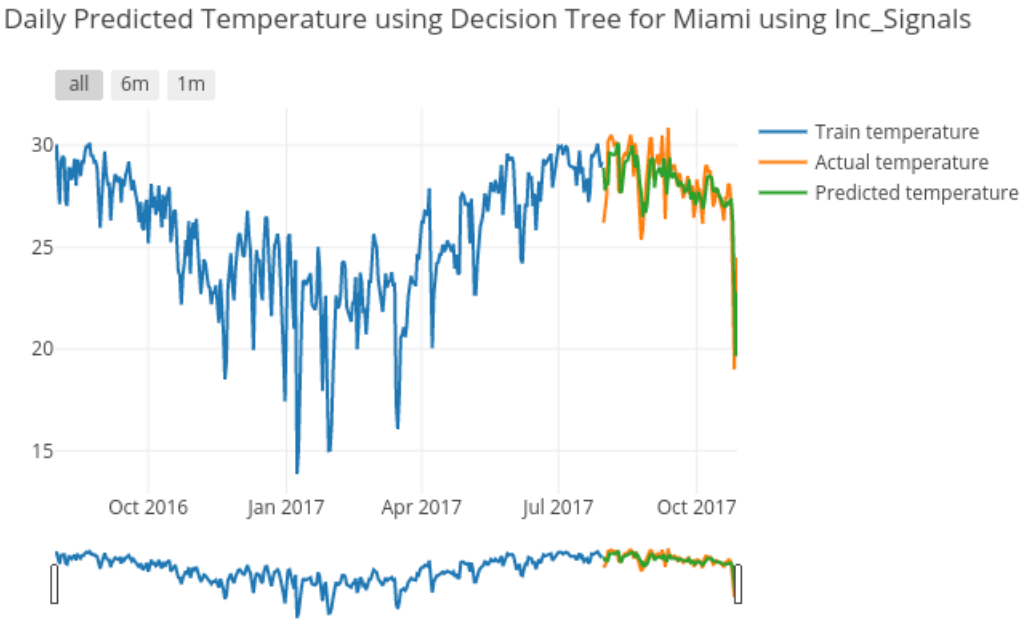
Out[125]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	30	2018-08-14 19:56:00.644376	RANDOM FOREST	Miami	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	1.168790
2	36	2018-08-14 19:56:51.638468	RANDOM FOREST	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	1.168790
3	24	2018-08-14 19:55:11.164461	RANDOM FOREST	Miami	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	1.206302
4	17	2018-08-14 19:44:46.805353	DECISION TREE	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.556905
5	12	2018-08-14 19:43:48.176029	DECISION TREE	Miami	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.577366
6	6	2018-08-14 19:42:49.124681	DECISION TREE	Miami	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	1.833555
7	54	2018-08-14 20:40:54.336628	RNN	Miami	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	2.012860
8	42	2018-08-14 20:35:28.587513	RNN	Miami	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	2.016882
9	48	2018-08-14 20:38:20.610680	RNN	Miami	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	2.093093

Random forest appears to be the clear winner, lets take a look at that chart.

In [73]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #30 - run by DESKTOP-KN40C32 on 2018-08-14 19:56:00.644376



We can see the forecast follows the trend but does not have the predictive power to track some of the more volatile swings in the signals.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

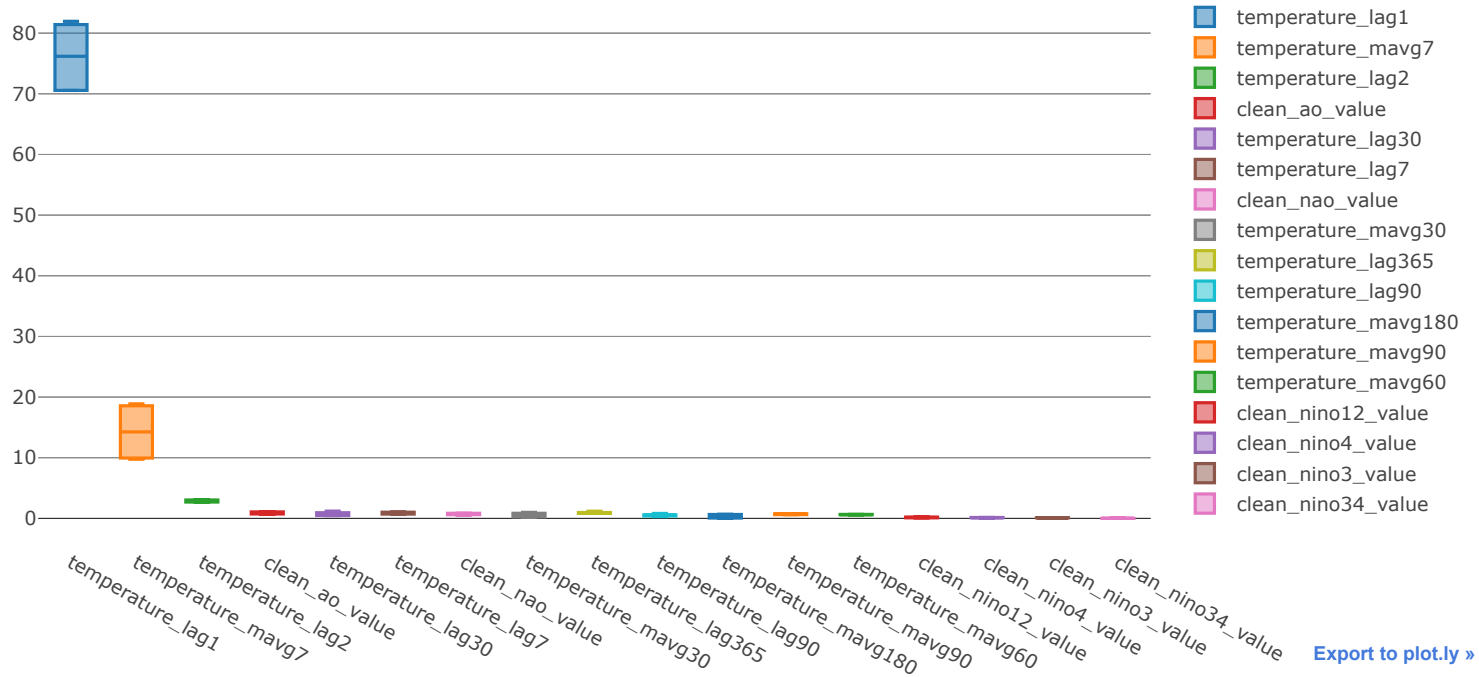
In [138]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[138]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.762803	0.697029	NaN	1.069394	1.069394
clean_nao_value	NaN	0.705866	0.518667	NaN	0.844053	0.844053
clean_nino12_value	NaN	0.205030	0.101543	NaN	NaN	NaN
clean_nino34_value	NaN	0.002508	0.000472	NaN	NaN	NaN
clean_nino3_value	NaN	0.113445	0.100821	NaN	NaN	NaN
clean_nino4_value	NaN	0.149838	0.105239	NaN	NaN	NaN
temperature_lag1	81.951355	81.333226	81.412405	71.035461	70.556534	70.556534
temperature_lag2	2.944449	2.956676	3.089420	3.020370	2.698790	2.698790
temperature_lag30	0.612916	0.475372	0.456209	1.198484	0.934467	0.934467
temperature_lag365	0.842632	0.837027	0.969628	1.184014	0.823453	0.823453
temperature_lag7	1.021827	0.703784	0.710618	1.039751	0.860722	0.860722
temperature_lag90	0.399428	0.296508	0.412470	0.818218	0.580242	0.580242
temperature_mavg180	0.607840	0.055693	0.061311	0.679769	NaN	NaN
temperature_mavg30	0.235605	0.239946	0.236853	0.995305	0.842836	0.842836
temperature_mavg60	0.630974	0.594589	0.659230	NaN	NaN	NaN
temperature_mavg7	9.960203	9.951108	9.751084	18.888060	18.563162	18.563162
temperature_mavg90	0.792771	0.616582	0.717001	0.631938	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [123]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [153]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[153]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.899655
clean_nao_value	0.728160
clean_nino12_value	0.153286
clean_nino34_value	0.001490
clean_nino3_value	0.107133
clean_nino4_value	0.127538
temperature_lag1	76.140919
temperature_lag2	2.901416
temperature_lag30	0.768653
temperature_lag365	0.913368
temperature_lag7	0.866237
temperature_lag90	0.514518
temperature_mavg180	0.351153
temperature_mavg30	0.565563
temperature_mavg60	0.628264
temperature_mavg7	14.279463
temperature_mavg90	0.689573

Now lets review the top 5 only

In [155]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[155]:

FEATURE	mean
temperature_lag1	76.140919
temperature_mavg7	14.279463
temperature_lag2	2.901416
temperature_lag365	0.913368
clean_ao_value	0.899655

In [ ]:

Experiment Results

Results for New York

In [6]:
```
import warnings
warnings.filterwarnings('ignore')

%run helper_functions.py
%matplotlib inline
```

Lets review the top 10 results, ordered by Mean Square Error (ascending)

In [7]:
```
city='New York'
results_top_10 = get_results(city=city, top_hm_results=10)
results_top_10
```

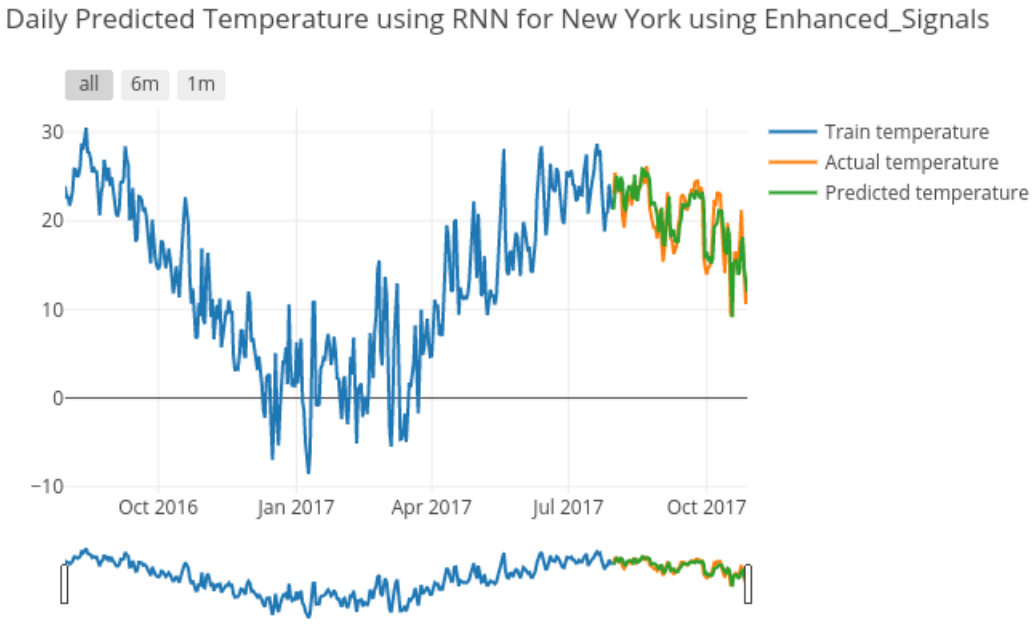
Out[7]:

	RUN_ID	DATETIME	MODEL_NAME	CITY	FEATURE_TYPE	HOST_MACHINE	MODEL_PARAMETERS	MODEL_RESULTS	MEAN_SQUARED_ERROR
1	49	2018-08-14 20:38:45.591679	RNN	New York	Enhanced_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'E...	{'features': ['temperature_lag1', 'temperature...	4.095801
2	25	2018-08-14 19:55:22.399240	RANDOM FOREST	New York	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	4.216172
3	31	2018-08-14 19:56:16.424827	RANDOM FOREST	New York	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	4.216172
4	19	2018-08-14 19:53:51.786200	RANDOM FOREST	New York	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'n_estimators': 50, 'Info': {...	{'features': ['temperature_lag1', 'temperature...	4.369905
5	43	2018-08-14 20:35:48.643558	RNN	New York	Inc_Signals	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'I...	{'features': ['temperature_lag1', 'temperature...	4.700272
6	37	2018-08-14 20:33:25.810145	RNN	New York	Basic	DESKTOP-KN40C32	{'epochs': 50, 'Info': {'feature_set_type': 'B...	{'features': ['temperature_lag1', 'temperature...	4.839433
7	1	2018-08-14 19:41:45.275297	DECISION TREE	New York	Basic	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	5.799872
8	7	2018-08-14 19:43:04.972617	DECISION TREE	New York	Inc_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	6.518000
9	13	2018-08-14 19:44:00.243495	DECISION TREE	New York	Enhanced_Signals	DESKTOP-KN40C32	{'max_depth': 8, 'Info': {'feature_set_type': ...	{'features': ['temperature_lag1', 'temperature...	6.767755

RNN has done much better than Random forest so in this instance it is the winner, lets take a look at that chart.

In [8]:
```
display_results(results_top_10.head(1), chart_type='predict')
```

Experiment #49 - run by DESKTOP-KN40C32 on 2018-08-14 20:38:45.591679



We can see the forecast follows the trend and it does a good job following the peaks and troughs of the actual temperature patterns.

Lets examine the top 10 features across the model runs for this location in a pivot table form.

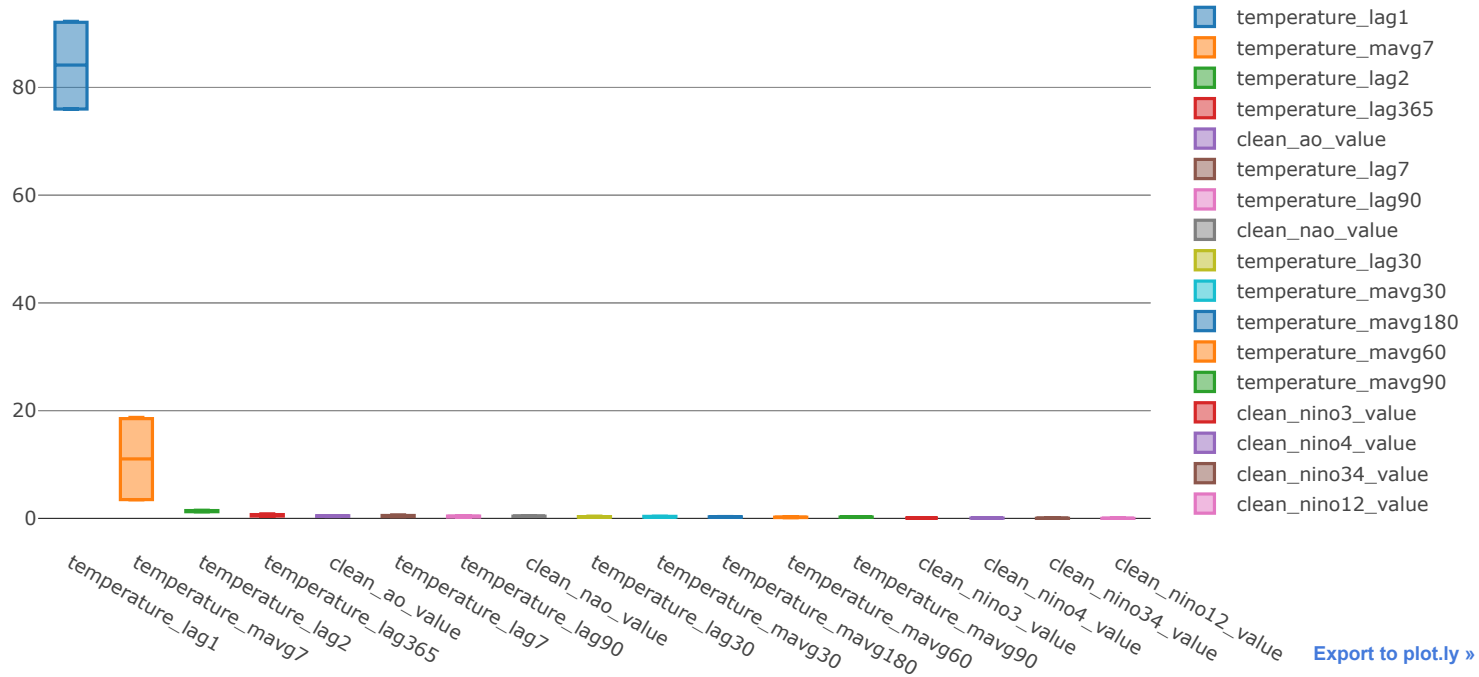
In [13]:
```
features_df = get_feature_importances(results_top_10)
features_df.pivot_table(index=['FEATURE'], columns=['MODEL_NAME','FEATURE_TYPE'], values="IMPORTANCE")
```

Out[13]:

MODEL_NAME	DECISION TREE			RANDOM FOREST		
FEATURE_TYPE	Basic	Enhanced_Signals	Inc_Signals	Basic	Enhanced_Signals	Inc_Signals
FEATURE						
clean_ao_value	NaN	0.342321	0.405794	NaN	0.495784	0.495784
clean_nao_value	NaN	0.413380	0.409221	NaN	0.362524	0.362524
clean_nino12_value	NaN	0.004266	0.000000	NaN	NaN	NaN
clean_nino34_value	NaN	0.016905	0.004148	NaN	NaN	NaN
clean_nino3_value	NaN	0.066890	0.067450	NaN	NaN	NaN
clean_nino4_value	NaN	0.039745	0.062396	NaN	NaN	NaN
temperature_lag1	92.275579	92.092587	92.086427	76.219275	75.992544	75.992544
temperature_lag2	1.509527	1.452181	1.434437	1.444294	1.242860	1.242860
temperature_lag30	0.222598	0.209085	0.215933	0.418034	0.329091	0.329091
temperature_lag365	0.592770	0.433306	0.411147	0.840668	0.707942	0.707942
temperature_lag7	0.540986	0.326928	0.326868	0.634435	0.483584	0.483584
temperature_lag90	0.315943	0.247336	0.219708	0.507118	0.429263	0.429263
temperature_mavg180	0.252067	0.197706	0.240038	0.345253	NaN	NaN
temperature_mavg30	0.376335	0.234558	0.236628	0.385914	0.316122	0.316122
temperature_mavg60	0.179414	0.206461	0.135184	0.240559	NaN	NaN
temperature_mavg7	3.558182	3.466374	3.489681	18.744591	18.520411	18.520411
temperature_mavg90	0.176598	0.249971	0.254940	NaN	NaN	NaN

Lets look a boxplot of this data to see how the distributions look across our experiments for this location.

In [14]:
```
traces = create_boxplot_traces_for_features(features_df)
iplot(traces)
```



Lets review the means for the features

In [15]:
```
features_df = get_feature_importances(results_top_10)
pivot_df = features_df.pivot_table(index=['FEATURE'], columns=[], values="IMPORTANCE", aggfunc= [np.mean])
pivot_df
```

Out[15]:

FEATURE	mean IMPORTANCE
clean_ao_value	0.434921
clean_nao_value	0.386912
clean_nino12_value	0.002133
clean_nino34_value	0.010526
clean_nino3_value	0.067170
clean_nino4_value	0.051071
temperature_lag1	84.109826
temperature_lag2	1.387693
temperature_lag30	0.287305
temperature_lag365	0.615629
temperature_lag7	0.466064
temperature_lag90	0.358105
temperature_mavg180	0.258766
temperature_mavg30	0.310946
temperature_mavg60	0.190405
temperature_mavg7	11.049942
temperature_mavg90	0.227170

Now lets review the top 5 only

In [16]:
```
pivot_df.columns = pivot_df.columns.get_level_values(0)
pivot_df.sort_values(['mean'], ascending=False).head(5)
```

Out[16]:

FEATURE	mean
temperature_lag1	84.109826
temperature_mavg7	11.049942
temperature_lag2	1.387693
temperature_lag365	0.615629
temperature_lag7	0.466064

This is again totally different than all the prior cases as the prior day temperature is the significant factor in the prediction of the temperature.

In [ ]: