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]:
```python
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]:
```python
# 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]:
```python
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]:
```python
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))
```

```python
            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]:
```python
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')
```

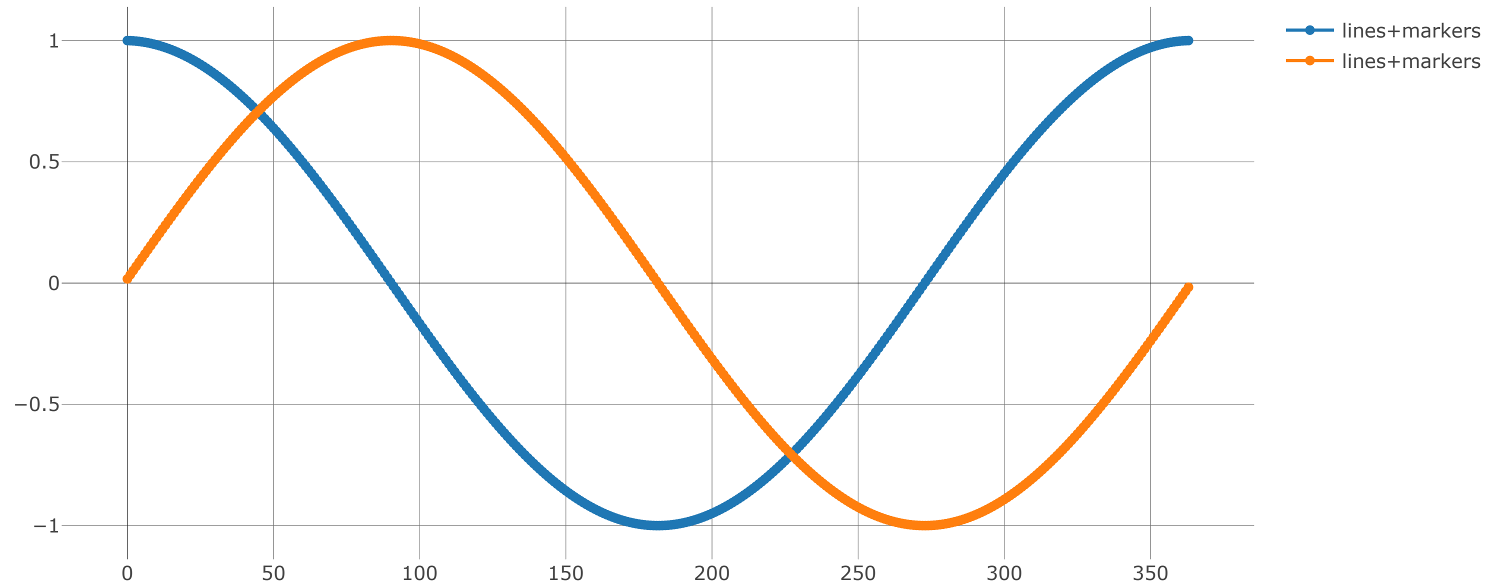
Lets create a couple of functions to help with feature engineering

- 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]:
```
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])
```

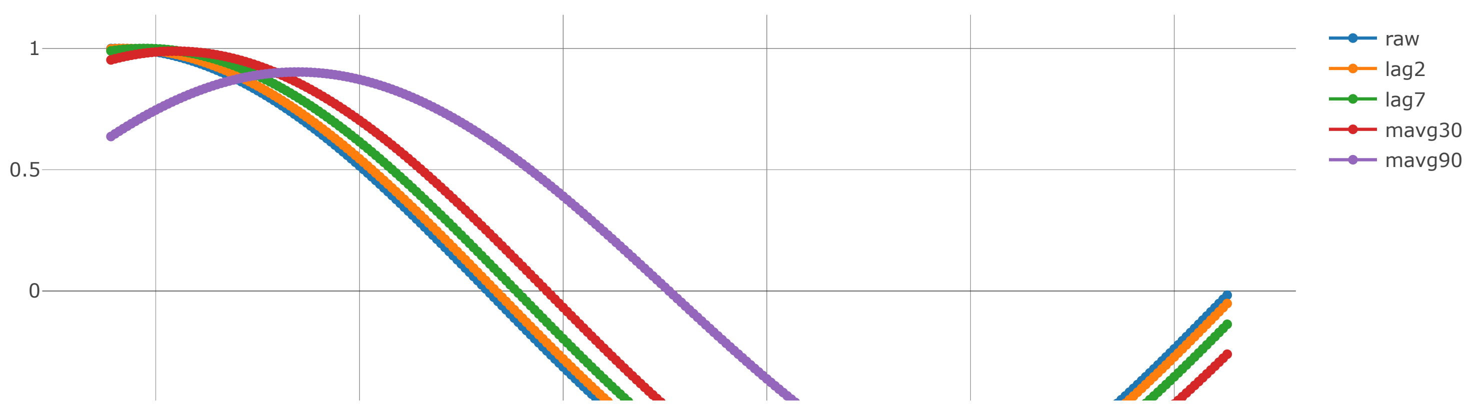


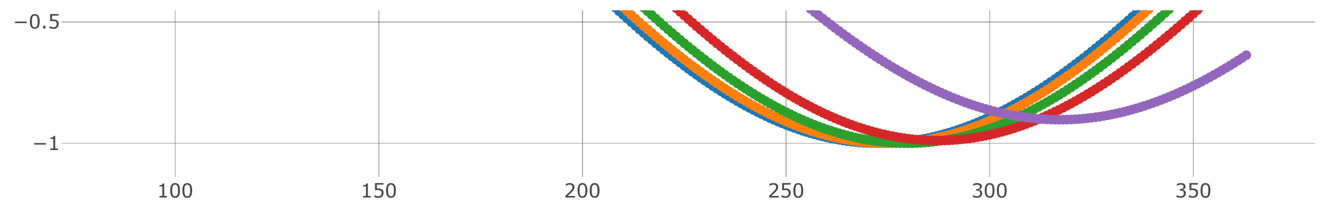
Export to plotly.att.com »

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]:

	value
datetime	
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]:

	value
datetime	
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]:

```python
# 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)
```

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

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

In [20]:
```python
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]:
```python
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]:

```python
# 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]:
```python
# 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]:
```python
# 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 [ ]: