Weekend Weather Prediction for Outdoor Event Scheduling in Monterey

Background + The Client's Decision (The Problem)
To decide when to schedule events, Monterey peninsula event planners for outdoor weddings and cultural festivals, such as the Monterey Jazz Festival, must often guess whether or not a given weekend will be sunny, sometimes up to a year in advance. But the area's climate can be notoriously volatile and counterintuitive: sunny weather ends sometime in May or June and begins again sometime in September, after a few months of fog throughout the summer season, with gaps of sun between

The Action
I propose to develop a predictive model based on weather data for the city of Monterey for the last ten years, to assign a sunniness probability for each calendar weekend. I will need to align calendar weekends across years, i.e., "the first weekend in March," and create an average sunniness score for a given weekend that takes into account three component days (Friday, Saturday, and Sunday) and averages across ten different corresponding weekends through time; a "calendar weekend" data structure will include data for thirty different calendar days (Friday, Saturday, and Sunday for ten years into the past). The dataset will be ordered from the National Center for Environmental Information, and will include several daily summary fields: total daily precipitation, average wind speed, and (most importantly) daily weather type. The data requested spans a period of May 14th, 2008 to May 14th 2019.

The project requires several considered modeling choices. The reality of climate change problematizes the question of timeframe: how many past years of data really reflect a weather trend relevant to the immediate future, rather than pre-global-warming patterns that no longer apply? But also how many years of data will be necessary to avoid overfitting to this small amount of data? One challenge of interpretation will be the quantification of qualitative, categorical data: how should a "partly cloudy" or "partly sunny" day impact a weekend's overall sunniness score? The model will also require binarization, as the goal is to determine a "sunny/not sunny" binary value for each calendar weekend — should this binarization occur for each day, for each weekend, for each day throughout history, or for each weekend throughout history?

Deliverable
Because the aim of the project is to improve event planners' future event scheduling decisions, I will create a calendar that displays sunniness probabilities for each weekend, an interface that will allow planners to make more informed choices when scheduling outdoor events in the area.

The Value of an Improved Decision
While this won't enable us to predict the future, it should allow event and festival planners in the Monterey area to make more informed event planning choices in the future. This might be especially useful to those planners who have recently moved to the area and aren't yet familiar

with the area’s peculiar weather patterns, and to smaller businesses or individual entrepreneurs who can’t afford data consultants.

Capstone 1 Data Wrangling Report

I obtained my dataset as a CSV file from the freely available National Oceanic and Atmospheric Administration's online database. The data contained hourly sky condition and visibility, collected from the Monterey airport between 2009 and the present. The data arrived via email request as a CSV file, which I imported into Pandas as a dataframe.

Cleaning Steps

1. Converted each row's date-time column to a Python datetime object in a new column.
2. Set this new column to be a datetime index for the dataframe.
3. Convert strings to float across dataset where possible with to_numeric. This coerced suspect values (see below), asterisk, and T values as floats if possible (for single letter numeric value suffixes) or as NaNs (for alphanumeric strings).
4. Backfilled the maximum daily temperature, minimum daily temperature, daily precipitation, and maximum daily wind speed columns throughout the day.
5. For hourly sky condition, learned to interpret the sky obscuration okta scale (1-9) and the abbreviation codes that appear in the hourly sky condition column, and then replaced the string representation of the list of sky conditions with a more accessible list of dictionaries; each dictionary's key is a condition code, and each dictionary's value is a SkyCondition namedtuple containing obscuration and vertical_distance fields.[1]

Missing Values

The presence of a NaN for a particular value means a couple different things in this dataset. Because the hourly weather type column tracks only the presence of rain, mist, or fog, a missing value in this column indicates the absence of one of these three types of weather. Otherwise, a NaN more generally means that no measurement is available.

The DailyPrecipitation column records the day's rainfall in inches on each day at 11:59 PM but otherwise contains all missing values. This reading sometimes contains a "T", "Ts", or "0:00s" values, which appear to indicate "trace amounts" of precipitation;

[1] This structure works better than a single dictionary for this data, because condition codes occur multiple times for a single hours sometimes, which would prevent a single dictionary from having unique keys.

these have been replaced with zero values and then backfilled with the other daily measurements.

The hourly temperature and hourly wind speed columns are missing about four thousand values each. It would be strange to impute these values, as the point of tracking hourly temperature is to characterize a single particular hour, so I've elected to leave these missing values as NaNs. The presence of daily temperature max and min, as well as the ability to calculate a daily average, should provide solutions if this becomes a problem.

The sky condition representation contained an especially challenging missing value: a clear day lacks a following integer to represent the vertical visibility from ground to clouds, but no NaN appeared for this value, as it occurred within a string that required custom parsing. The parsing algorithm handles this by replacing the missing value with a 0, as would normally trail the clear day code to indicate a missing value, as in "CLR:00 00".

The HourlyWindSpeed and HourlyDryBulb temperature columns contained many missing values indicated with an asterisk string ('*'). These have been replaced with NaNs.

To avoid processing errors on the the hourly sky condition column, the NaN values in that column have been converted to empty lists.

Peculiar to this dataset, NOAA defines a value with an "s" suffix indicates a "suspect value." In these cases, the value has nonetheless been coerced as a floating point value without its suspect suffix.

Outliers

There are some outliers in various respects. The maximum and minimum daily temperatures tend to have from four to six spikes in a given year. And any Californian will tell you that daily precipitation has seen some historically surprising highs and lows in the past several years.

Monterey Weather: Exploratory Data Visualization

In [304]:
```
from datetime import date

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [305]:
```
# pandas settings
pd.options.mode.chained_assignment = None  # turn off false positive warnings for editing a copy
```

In [306]:
```
df = pd.read_csv('cleaned_df.csv', parse_dates=['datetime'], index_col=['datetime'])
df.head()
```

Out[306]:

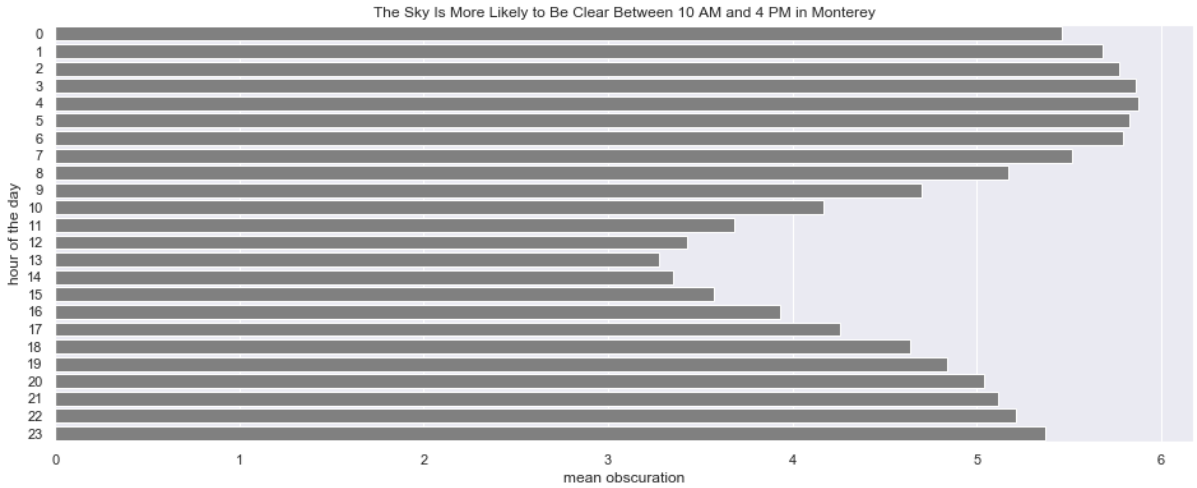
datetime	DATE	HourlyPresentWeatherType	HourlySkyConditions	HourlyVisibility	Hou
2009-04-01 00:08:00	2009-04-01T00:08:00	NaN	[{'BKN': SkyCondition(obscuration=7, vertical_...	10.0	
2009-04-01 00:50:00	2009-04-01T00:50:00	NaN	[{'SCT': SkyCondition(obscuration=4, vertical_...	9.0	
2009-04-01 00:54:00	2009-04-01T00:54:00	NaN	[{'SCT': SkyCondition(obscuration=4, vertical_...	9.0	
2009-04-01 01:54:00	2009-04-01T01:54:00	NaN	[]	9.0	
2009-04-01 02:54:00	2009-04-01T02:54:00	NaN	[]	9.0	

Introduction

Chaos determines weather, and the most powerful weather prediction systems (https://xkcd.com/1885/) depend on models of chaos. With this in mind, when asked to make a prediction for hypothetical event planners based on a decade of hourly weather data using a laptop, it's useful to frame the task of exploratory visualization with the question: from what perspectives is this data least chaotic?

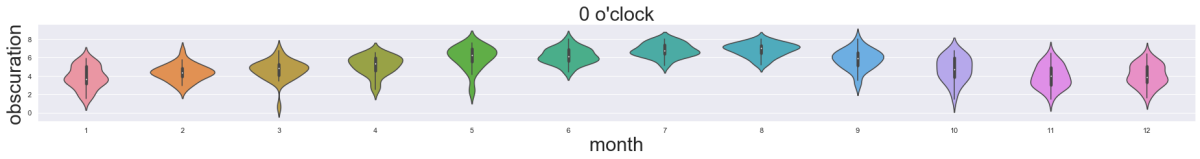
Event planners are often most concerned with the presence or absence of sun, which appears in the dataset as the average sky obscuration value. So a good first question might be,

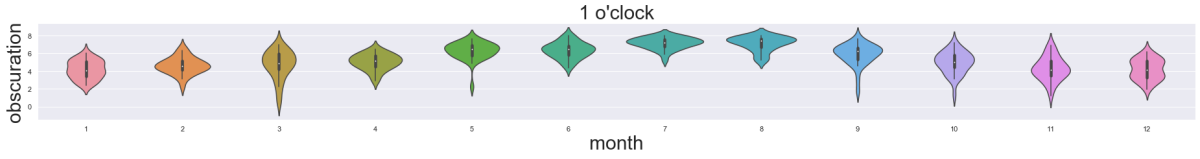
In [307]:
```
sns.set()

hourly_obscuration = pd.DataFrame(df.groupby(df.index.hour).averageObscu
ration.mean())
hourly_obscuration = hourly_obscuration.reset_index()
hourly_obscuration.columns = ['hour of the day', 'mean obscuration']
plt.figure(figsize=(16, 6))
sns.barplot(y='hour of the day', x='mean obscuration', color='grey', ori
ent='h', data=hourly_obscuration).set_title('The Sky Is More Likely to B
e Clear Between 10 AM and 4 PM in Monterey')
plt.show()
```

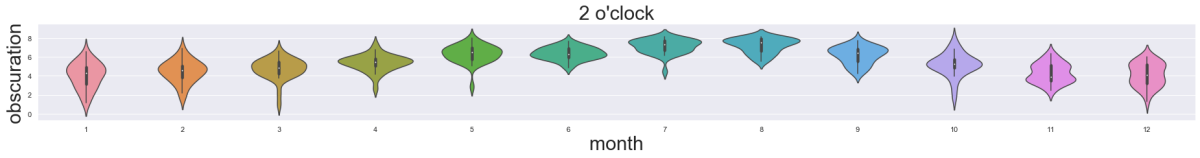


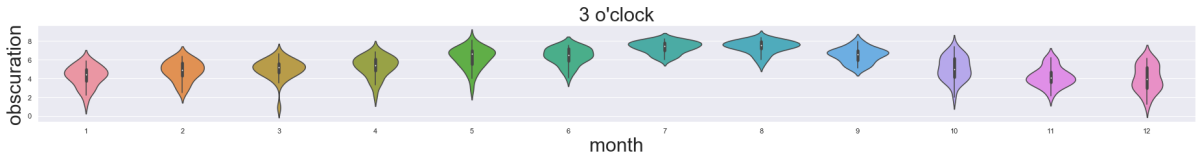
It looks like the hours between 10 AM and 4 PM are the least obscured. But this is averaging out any changes over the course of the usual year. How might the result take into account this sort of variation?:

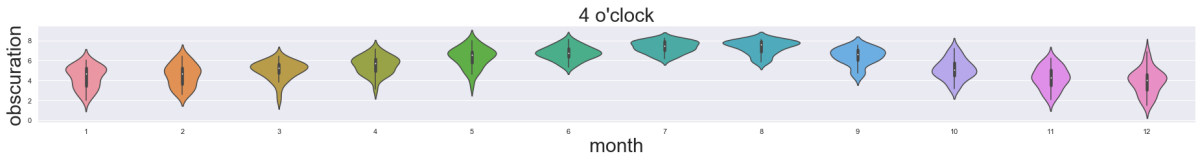
In [308]:
```
f, axes = plt.subplots(24,1)
f.set_size_inches(32,128)
axes = axes.flatten()

def add_subplot(master_frame, index):
    hour_frame = hours.loc[index]
    hour_frame = hour_frame.reset_index()
    v = sns.violinplot(y='averageObscuration', x='month', data=hour_fram
e, ax=axes[index])
    v.set_xlabel("month", fontsize=30)
    v.set_ylabel("obscuration", fontsize=30)
    title_string = str(index) + " o'clock"
    v.set_title(title_string, fontsize=30)

hours = df
hours = pd.DataFrame(df.groupby([df.index.hour, df.index.day, df.index.m
onth]).averageObscuration.mean())
hours.index = hours.index.set_names(['hour', 'day', 'month'])
for x in range(24):
    add_subplot(hours, x)



left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between
 subplots
hspace =  1.1    # the amount of height reserved for white space between
subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace
)
plt.savefig('figures/hourlyAverageObscurationOverYear.png')
plt.show()
```

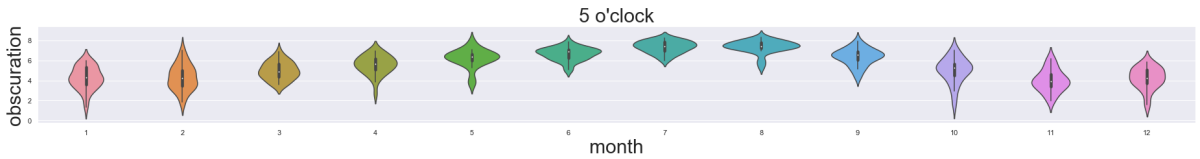

0 o'clock
obscuration
month

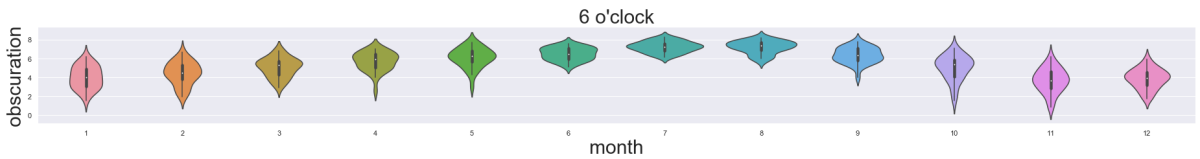

1 o'clock
obscuration
month

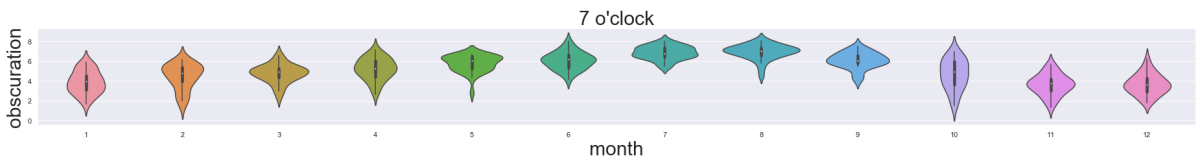

2 o'clock
obscuration
month


3 o'clock
obscuration
month

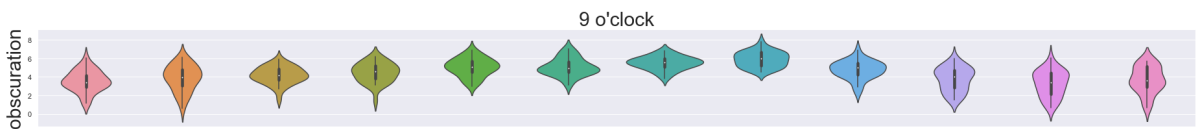

4 o'clock
obscuration
month

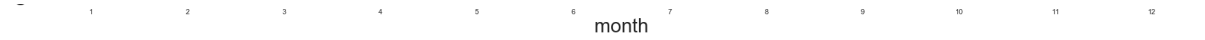

5 o'clock
obscuration
month

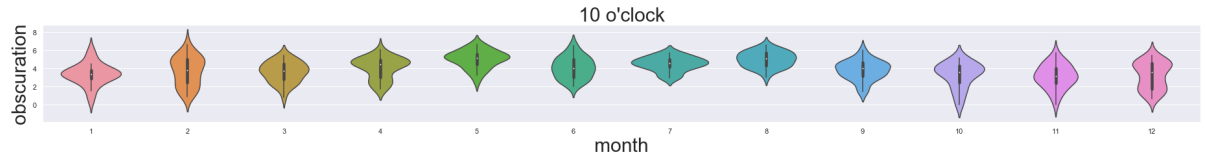

6 o'clock
obscuration
month

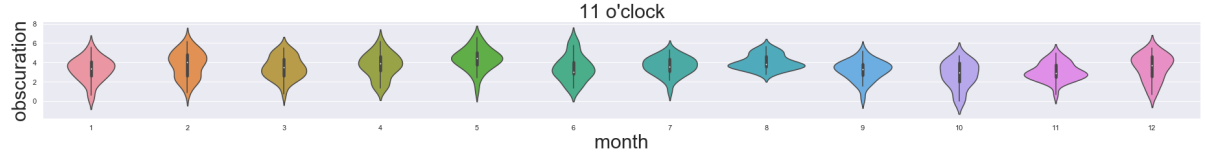

7 o'clock
obscuration
month


8 o'clock
obscuration
month

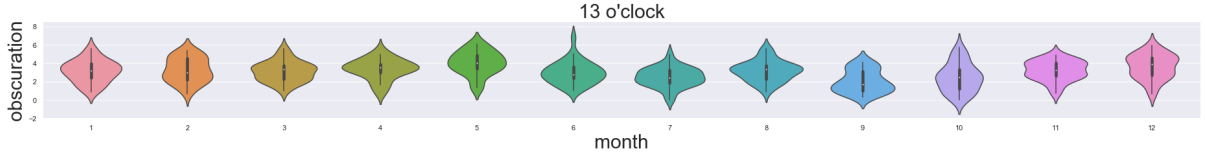

9 o'clock
obscuration

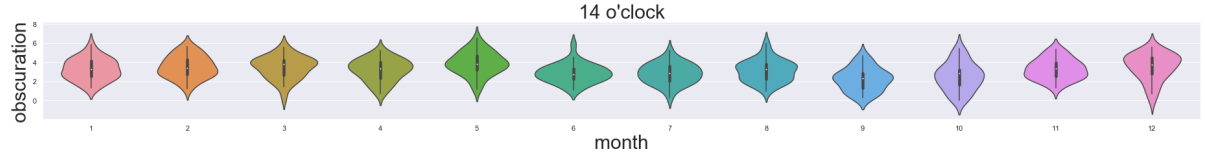

month

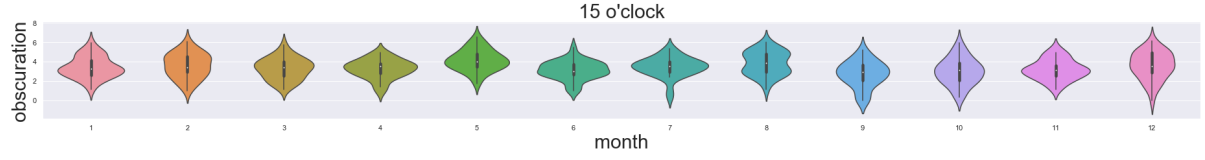

10 o'clock
obscuration
month

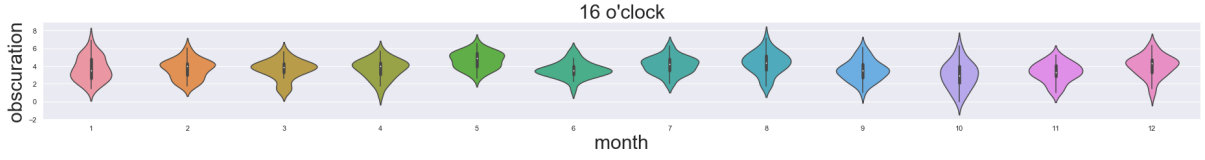

11 o'clock
obscuration
month

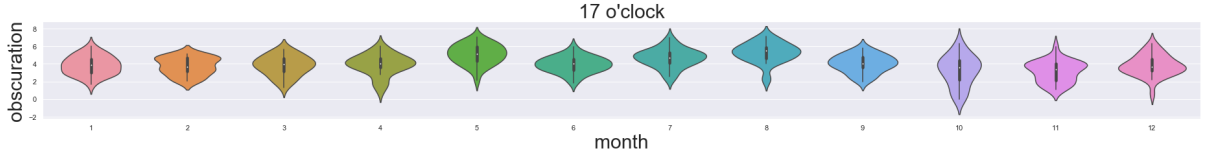

12 o'clock
obscuration
month

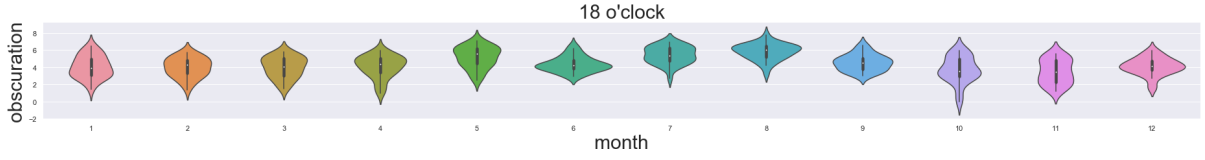

13 o'clock
obscuration
month


14 o'clock
obscuration
month

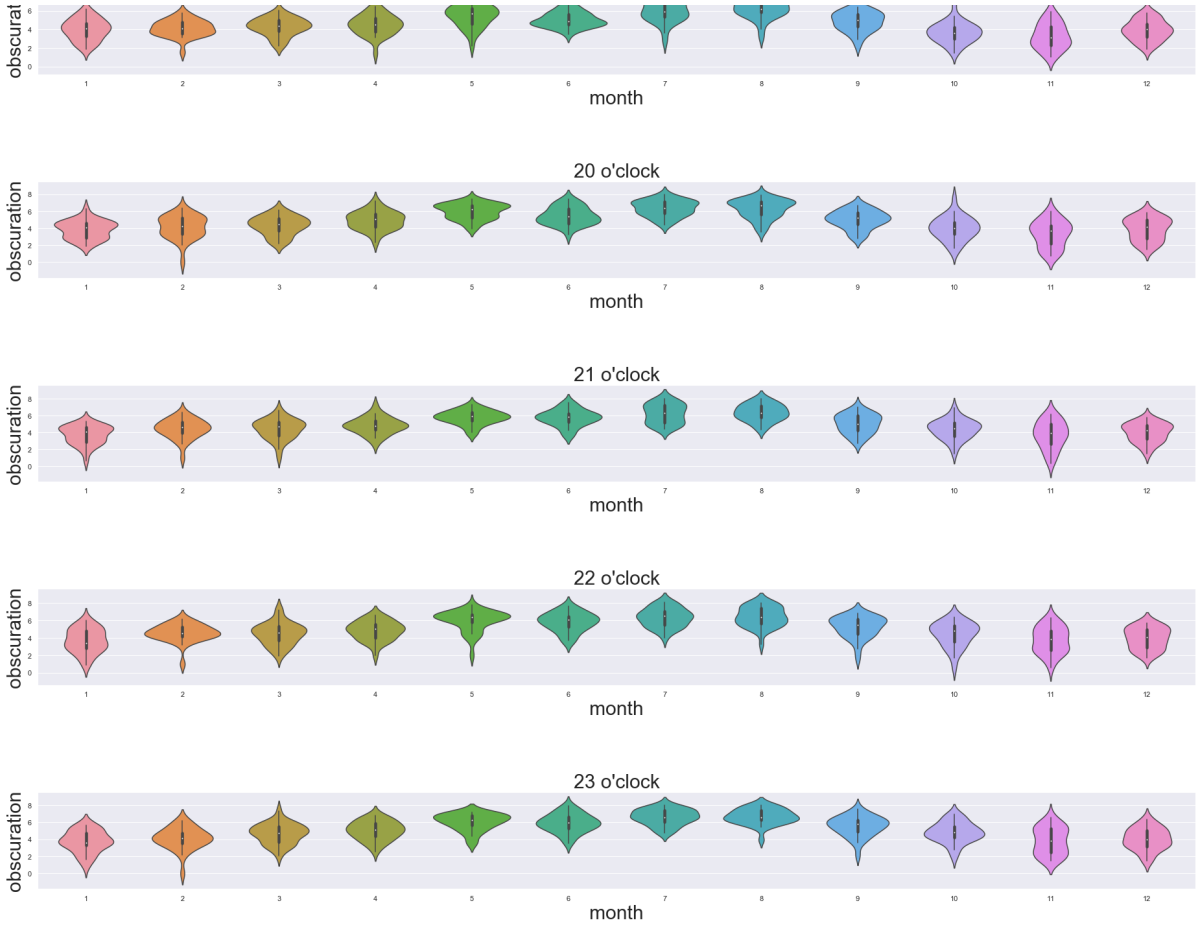

15 o'clock
obscuration
month


16 o'clock
obscuration
month


17 o'clock
obscuration
month


18 o'clock
obscuration
month

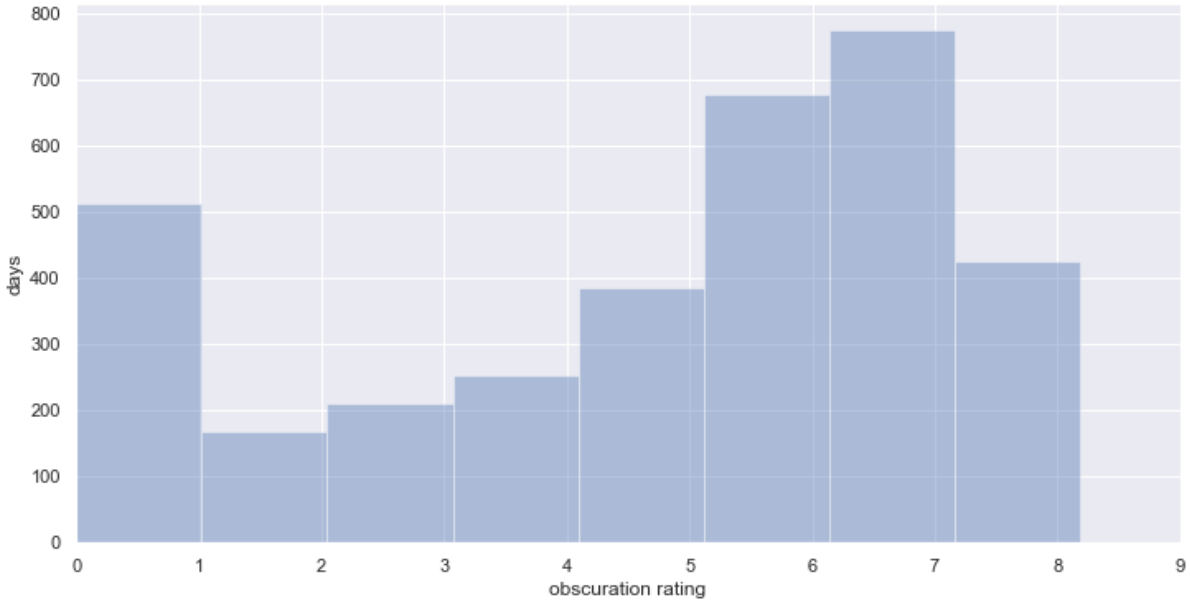

19 o'clock



These plots reveal a couple important features of the area's climate. The nighttime and early morning hours become reliably very foggy in the "summer" months, which is the sort of relatively automatic watering that allowed crops like berries to thrive in this area in the early twentieth century. In contrast, late morning to early afternoon hours remain much more varied throughout the year. 1 PM and 2 PM in September and October show the best (most bottom-heavy) visibility distributions.

Before looking into obscuration by day more speciifcally, it's useful to have a look first at the distribution of daily average obscuration values across the ten years of hourly data. (That is, 365 * 10 daily average obscuration values):

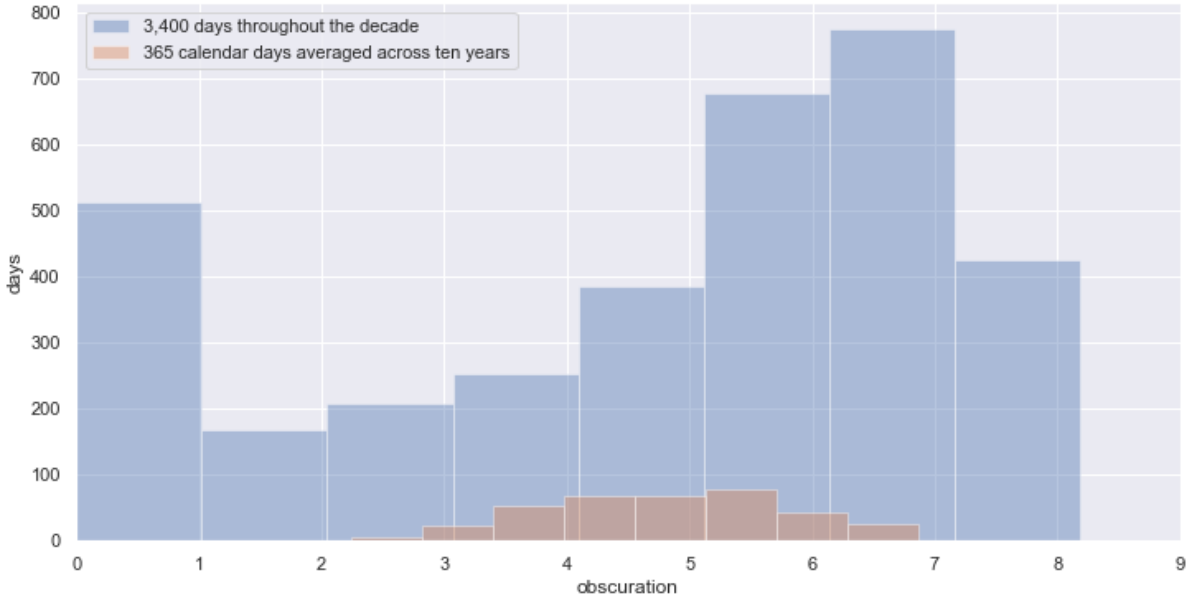
In [309]:
```
plt.figure(figsize=(12, 6))
sns.set_style('darkgrid')

obscuration = pd.DataFrame(df['averageObscuration'].groupby([df.index.da
te]).mean().dropna()) # ten years of individual dates
obscuration.name = 'There are more cloudy days than sunny days in the la
st decade.'

plt.xlim(0, 9)
sns.distplot(obscuration,bins=8, kde=False, norm_hist=False)

plt.xlabel('obscuration rating')
plt.ylabel('days')
plt.show()
```



This eight-bin histogram illustrates the national weather service's mapping between sky condition categories and obscuration values on a scale from 0 to 9: 0-2 would get the code corresponding to "clear," 2-4 "scattered clouds," 4-6 "overcast," and 6-9 "fully obscured." (A rating of 9 is rarely used.) This shows us that weather in the area skews a bit toward cloudiness. The recommendation task here takes a binary view of this histogram: we're mainly interested in the days with an obscuration of less than or equal to around 4, the threshold for "overcast."

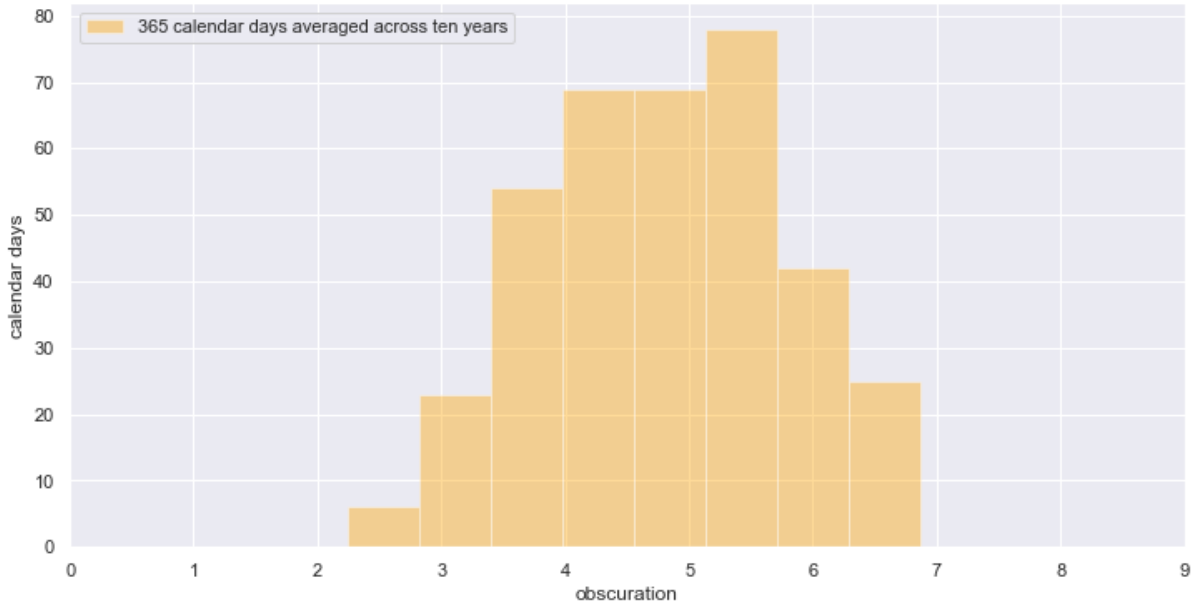
But this distribution takes each date in the last decade as a single datapoint. This view looks a bit different if we average each calendar day across the ten years of data to get 365 data points instead of 3,645:

In [310]:
```
plt.figure(figsize=(12, 6))

obscuration = pd.DataFrame(df['averageObscuration'].groupby([df.index.da
te]).mean().dropna()) # ten years of individual dates
obscuration.name = 'There are more cloudy days than sunny days in the la
st decade.'
sns.distplot(obscuration,bins=8, kde=False, label='3,400 days throughout
the decade')

obscuration_year_averaged_across_decade = pd.DataFrame(df['averageObscur
ation'].groupby([df.index.month, df.index.day]).mean())
obscuration_year_averaged_across_decade.name = 'Averaging Obscuration Ac
ross the Ten Years Compresses the Distribution'
sns.distplot(obscuration_year_averaged_across_decade,bins=8, kde=False,
label='365 calendar days averaged across ten years')


# plt.ylim(0, 500)
plt.xlim(0, 9)
plt.legend()
plt.xlabel('obscuration')
plt.ylabel('days')
plt.show()
# plt.savefig('figures/dailyMeanForDatesAcrossDecade.png') # uncomment t
o write out figure
```



Let's have a closer look at the distribution of obscuration values for 365 calendar days each averaged over 10 yearly values:

In [311]:
```
plt.figure(figsize=(12, 6))
obscuration_year_averaged_across_decade = pd.DataFrame(df['averageObscuration'].groupby([df.index.month, df.index.day]).mean())
obscuration_year_averaged_across_decade.name = 'Averaging Obscuration Across the Ten Years Compresses the Distribution'
sns.distplot(obscuration_year_averaged_across_decade,bins=8, color='orange', kde=False, label='365 calendar days averaged across ten years')


# plt.ylim(0, 500)
plt.xlim(0, 9)
plt.legend()
plt.xlabel('obscuration')
plt.ylabel('calendar days')
plt.show()
# plt.savefig('figures/dailyMeanForDatesAcrossDecade.png') # uncomment to write out figure
```



Averaging each day across its ten yearly values gives a distribution that looks more like a normal distribution, as the clearest and least clear outlier dates are averaged toward less extreme values. It looks like around 75 calendar days have an average obscuration rating below 4, a conservative threshold for a clear sky.

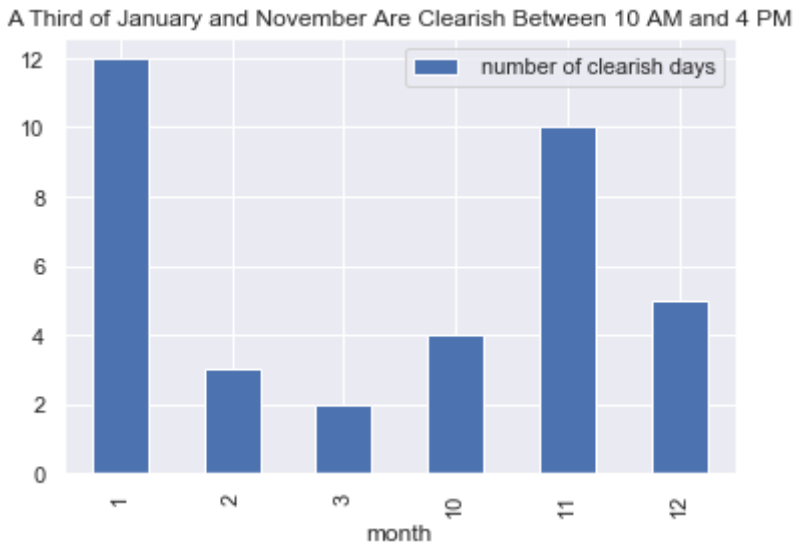
After getting a sense of how daily obscuration distributes, we can focus in on the clear days.

In [312]:
```
by_date = df[(df.index.hour >= 10) & (df.index.hour <= 16)] # get 10 AM
 to 4 PM
by_date = df.groupby([df.index.month, df.index.day]).averageObscuration.
mean() # average by calendar day across decade
by_date = by_date.sort_values()
by_date = by_date[by_date <= 3.5] # 3.5 is a conservative cut-off for a
 clear day: there are at worst "scattered clouds"
print(str(len(by_date)) + " days have had a decade average obscuration r
ating of under 3.5.")
```

```
36 days have had a decade average obscuration rating of under 3.5.
```

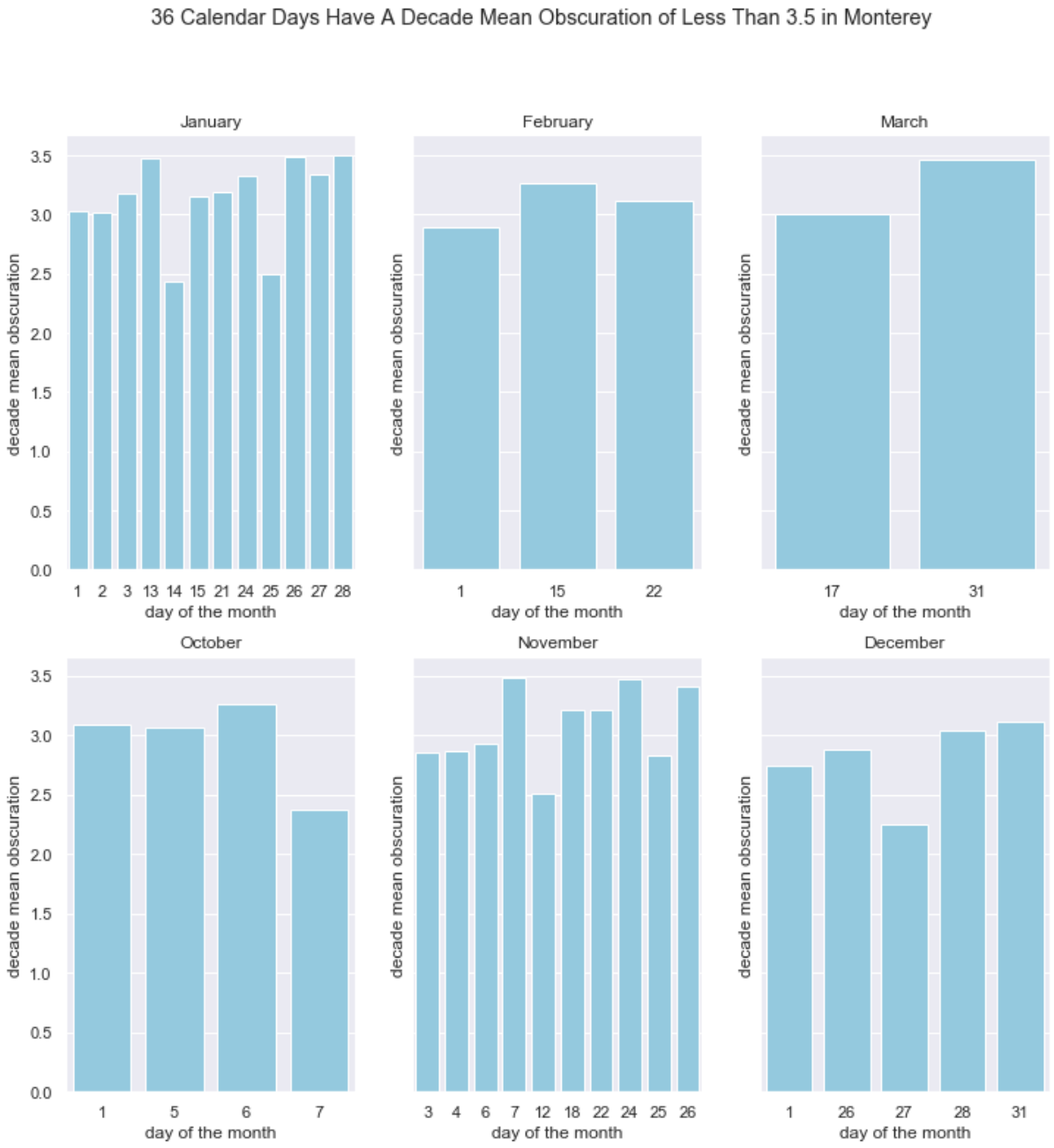
A cut-off of 3.0 cut this number down from 36 days to 12 days, which means that scattered cloud days are especially important to this analysis.

In [313]:
```
by_date = df[(df.index.hour >= 10) & (df.index.hour <= 16)] # get 10 AM
 to 4 PM
by_date = df.groupby([df.index.month, df.index.day]).averageObscuration.
mean()
by_date = by_date.sort_values()
by_date = by_date[by_date <= 3.5] # 3.5 is a conservative cut-off for a
 clear day: there are at worst "scattered clouds"
by_date = pd.DataFrame(by_date)
by_date.index = by_date.index.rename(["month", "day"])
# by_date = by_date.unstack(level=0)
by_date
by_date.groupby('month').count().rename(columns={'averageObscuration':'
 number of clearish days'}).plot(kind='bar', title='A Third of January a
nd November Are Clearish Between 10 AM and 4 PM')
plt.show()
```



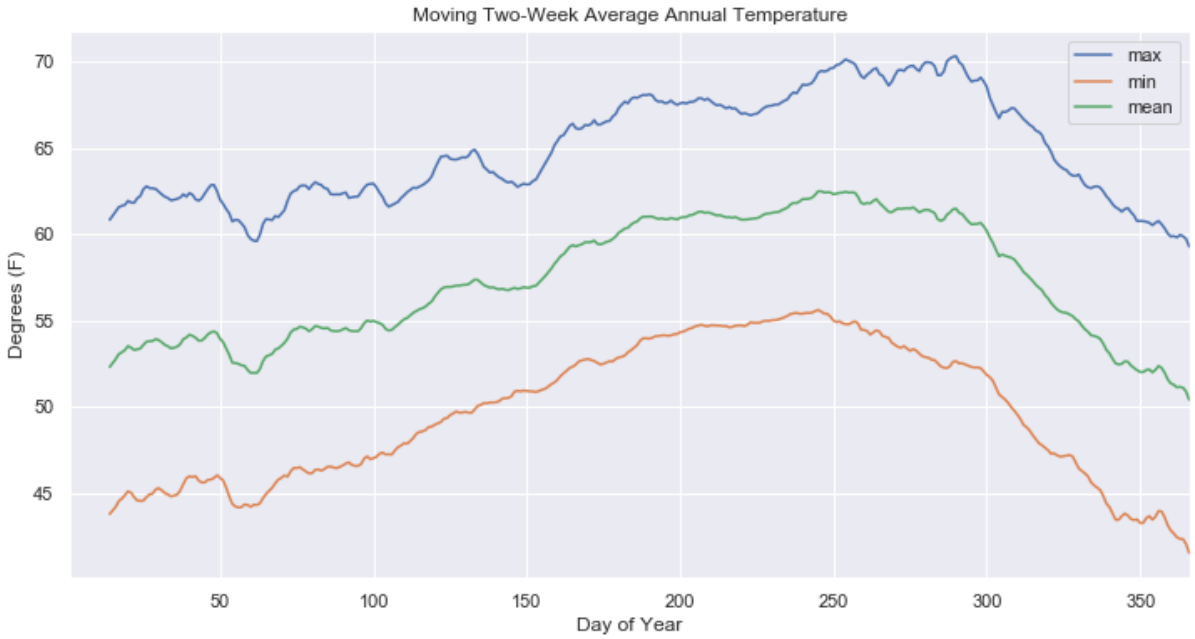
Public consensus in the area is that September and October have the nicest days, but the data reveal that this understanding might be conflating temperature with sky clarity: November and January have the most reliably clearish days. Next, it might be useful to map out all 36 of these days with their obscurations.

In [314]:
```
by_date = df[(df.index.hour >= 10) & (df.index.hour <= 16)] # get 10 AM
 to 4 PM
by_date = df.groupby([df.index.month, df.index.day]).averageObscuration.
mean()
by_date = by_date.sort_values()
by_date = by_date[by_date <= 3.5] # 3.5 is a conservative cut-off for a
 clear day: there are at worst "scattered clouds"
by_date = pd.DataFrame(by_date)
by_date.index = by_date.index.rename(["month", "day"])

# set up subplots
f, axes = plt.subplots(2,3, sharey='row')
f.set_size_inches(12,12)
axes = axes.flatten()

# set up title lookup
month_dict = {1: 'January', 2: 'February', 3: 'March', 10: 'October', 11
: 'November', 12: 'December'}

# plot a month
def plot_month(frame, month_index, plot_index):
    """Plots the decade mean obscuration for the index month's clearish
 days"""
    frame = frame.reset_index()
    frame = frame[frame['month'] == month_index]
    frame = frame.set_index('day')
    frame = frame.sort_index()
    b = sns.barplot(data=frame, x=frame.index, color='skyblue', y='avera
geObscuration', ax=axes[plot_index])
    b.set_xlabel('day of the month')
    b.set_ylabel('decade mean obscuration')
    b.set_title(month_dict[month_index])

for plot, month in enumerate(set(by_date.index.get_level_values(0))):
    plot_month(by_date, month, plot)

plt.suptitle("36 Calendar Days Have A Decade Mean Obscuration of Less Th
an 3.5 in Monterey")
plt.savefig('figures/clearishDaysByMonth.png')
plt.show()
```



This suggests a correlation between temperature and obscuration, as there are so many clear days in winter months. Let's investigate that by plotting temperature range against obscuration. But first, let's have a look at temperature itself.

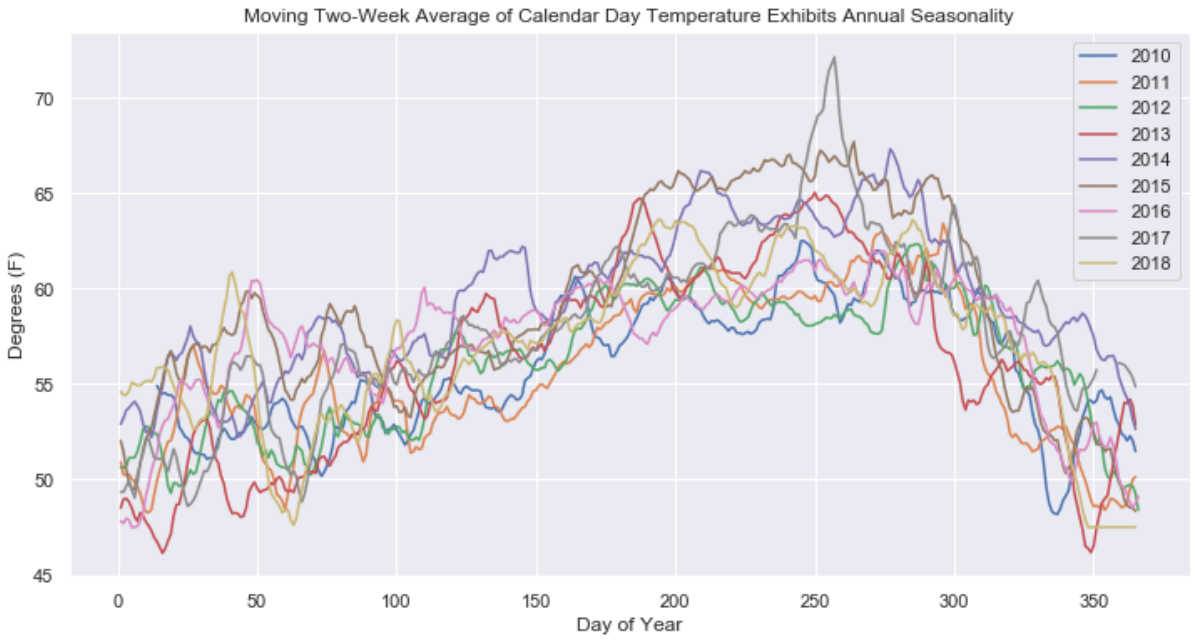
In [315]:
```
plt.figure(figsize=(12, 6))

x = df
max_temp = x.groupby(df.index.dayofyear)['DailyMaximumDryBulbTemperature'].mean().rolling(14).mean().plot(label='max')
min_temp = x.groupby(df.index.dayofyear)['DailyMinimumDryBulbTemperature'].mean().rolling(14).mean().plot(label='min')

x['mean_temp'] = (df['DailyMaximumDryBulbTemperature'] + df['DailyMinimumDryBulbTemperature'])/2
mean_temp = x.groupby(df.index.dayofyear)['mean_temp'].mean().rolling(14).mean().plot(label='mean')

# plot
plt.legend()
plt.title('Moving Two-Week Average Annual Temperature')
plt.xlabel('Day of Year')
plt.ylabel('Degrees (F)')
plt.show()
```



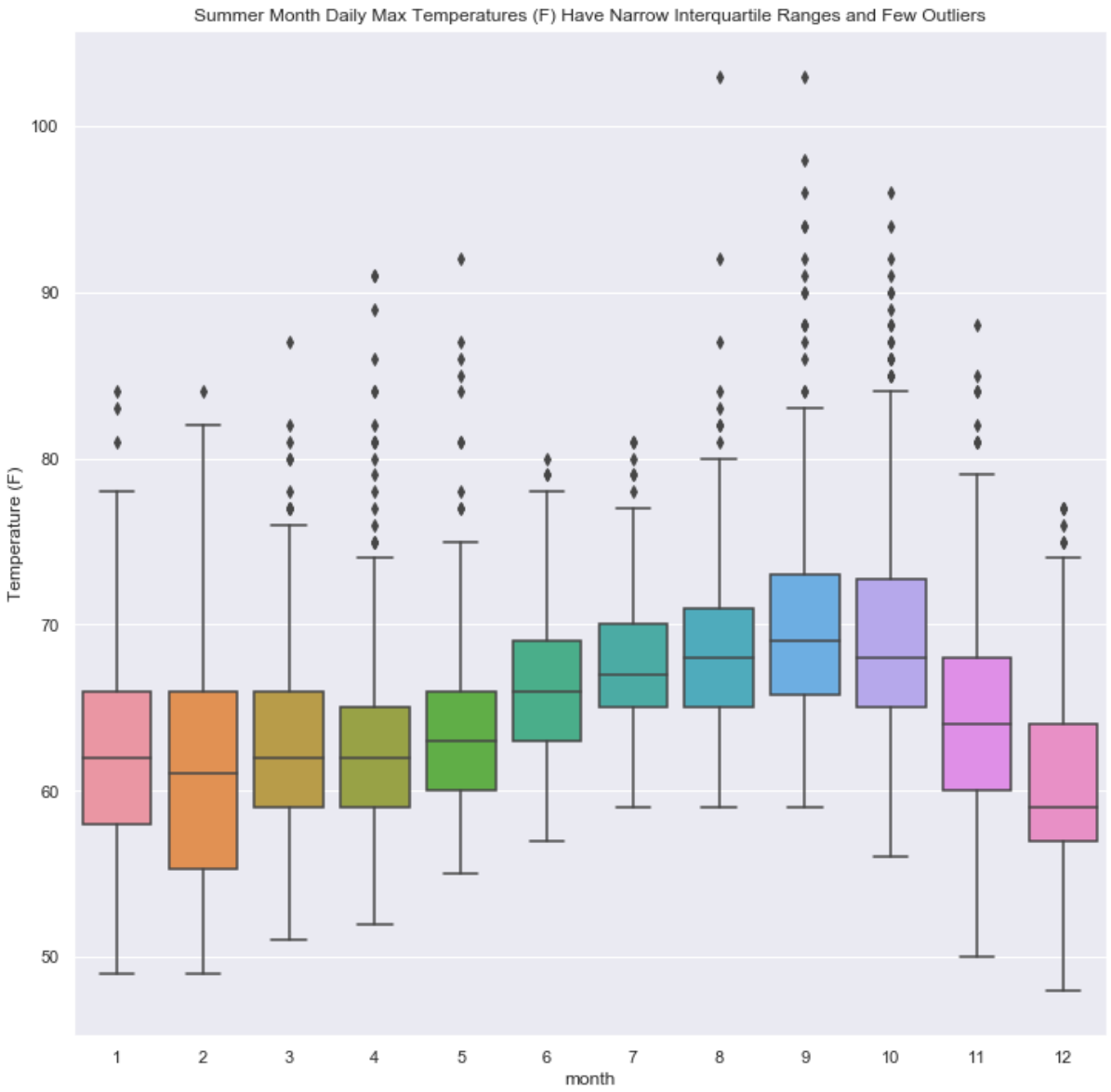
This plot of maximum daily temperature shows that days in the middle of the year have much less choatic variation than the rest of the year. Some boxplots per month should confirm this from the month-to-month view of the year.

Do the last ten years share consistent seasonal patterns?:

In [316]:
```
plt.figure(figsize=(12, 6))

x = df
bool_index = (x.index.year >= 2010) & (x.index.year <= 2018)
x = x[bool_index]
x['mean_temp'] = (x['DailyMaximumDryBulbTemperature'] + x['DailyMinimumDryBulbTemperature'])/2

mean_temp = x.groupby([x.index.year, x.index.dayofyear])['mean_temp'].mean().rolling(14).mean()

mean_temp = mean_temp.unstack(level=0)

# plot
for col in mean_temp.columns:
    plt.plot(mean_temp[col], label=str(col))
plt.legend()
plt.title('Moving Two-Week Average of Calendar Day Temperature Exhibits Annual Seasonality')
plt.xlabel('Day of Year')
plt.ylabel('Degrees (F)')
plt.show()
```

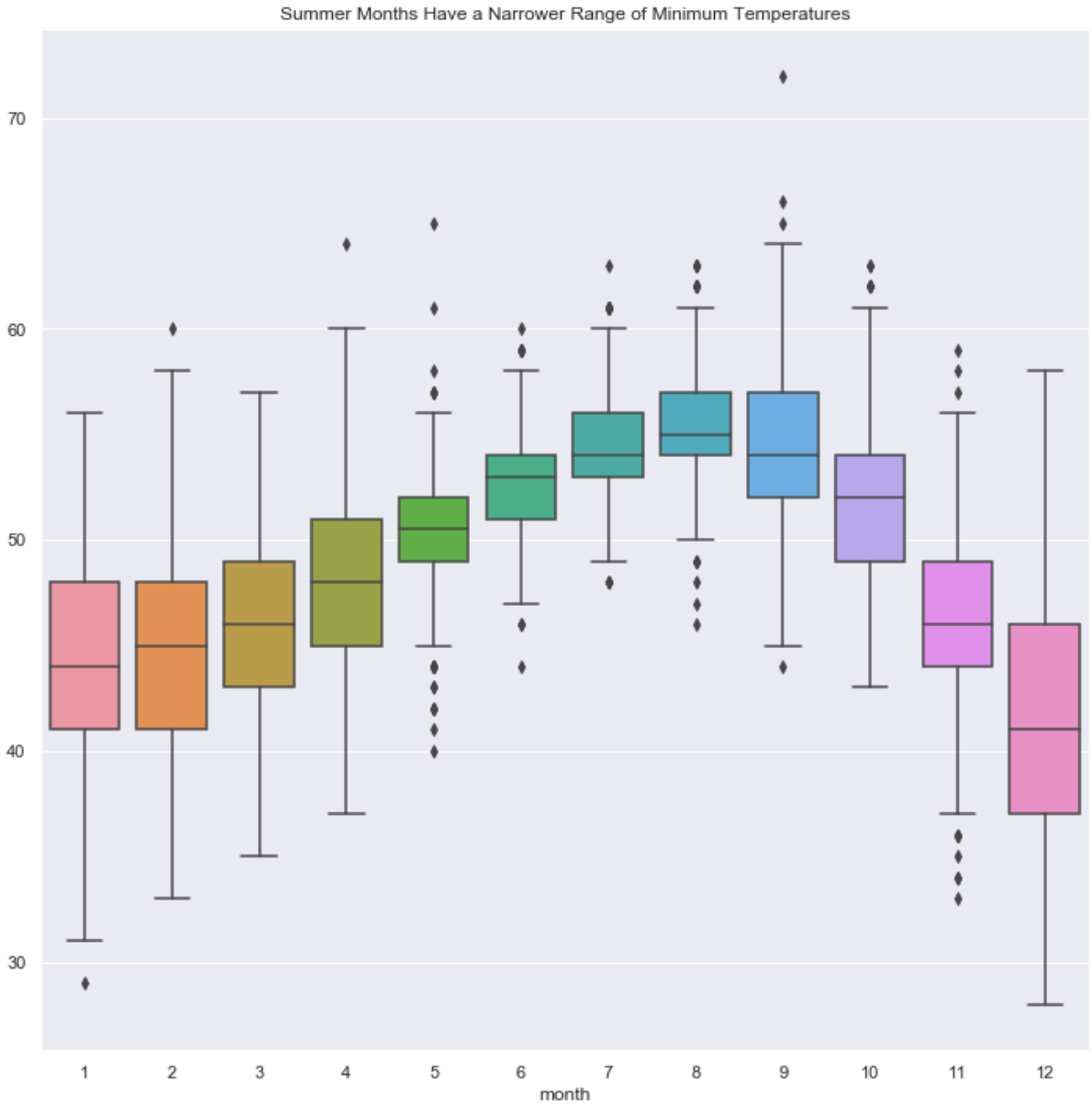


In [317]:
```
plt.figure(figsize=(12, 12))

# processing setup
x = df
x.reset_index()

# wrangle: year columns, day of year index, daily high temperature values
# mean does nothing here: all entries have same max value
x = x.groupby([x.index.month, x.index.date]).DailyMaximumDryBulbTemperature.last() # all hourly entries have same value
x.index = x.index.rename(['month', 'date'])
x = x.unstack(level=0)
x.head(100)


# # plot
b = sns.boxplot(data=x)
b.set_title('Summer Month Daily Max Temperatures (F) Have Narrow Interquartile Ranges and Few Outliers')
b.set_ylabel('Temperature (F)')
plt.show()
```



This plot confirms what we see in the previous line plot: the "summer" months have a much narrower range of maximum temperatures than the other months; June and July have the fewest outlier temperatures. February has an exceptionally wide range of temperatures compared to the other months.

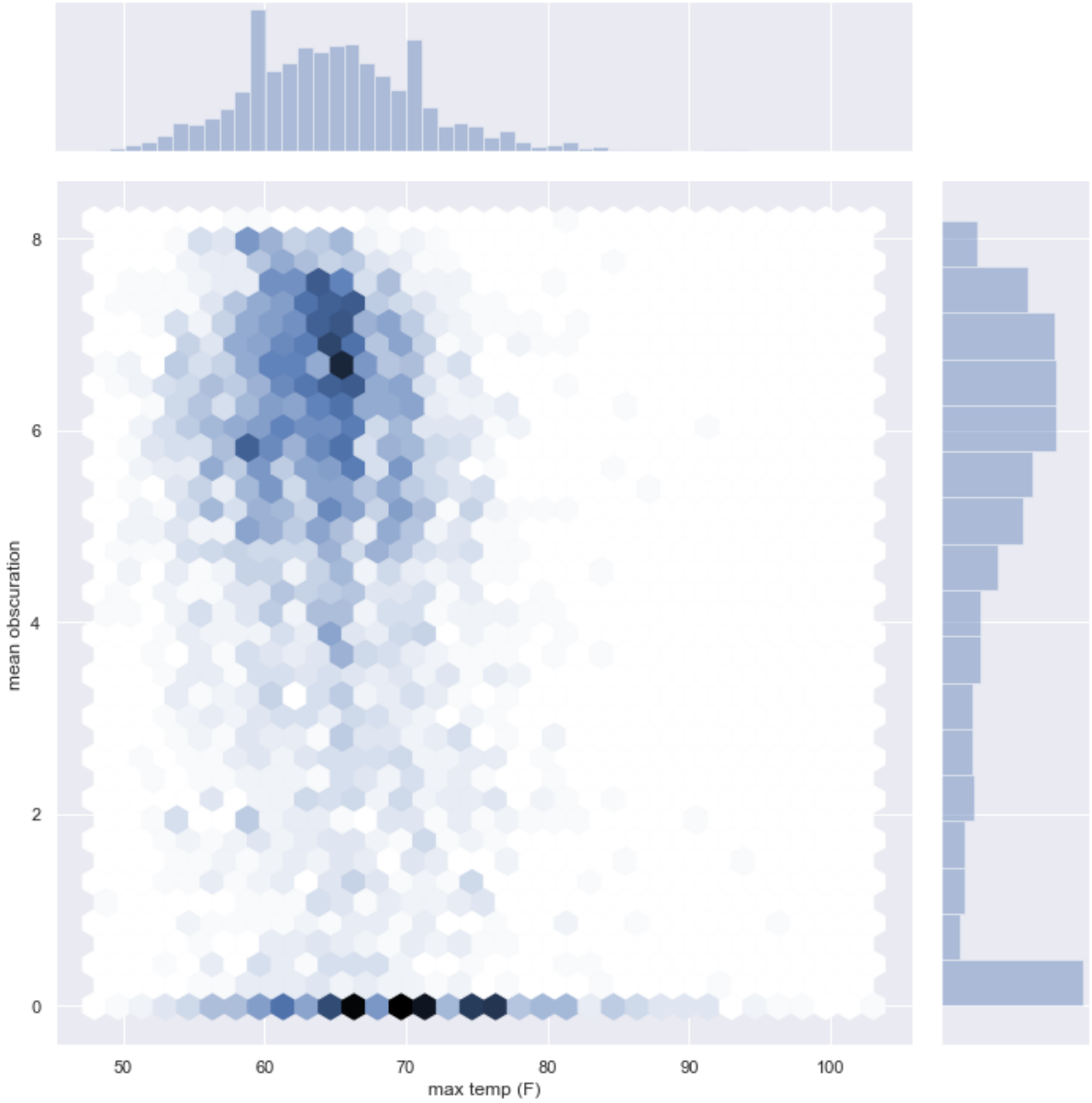
In [318]:
```
plt.figure(figsize=(12, 12))

# processing setup
x = df
x.reset_index()

# wrangle: year columns, day of year index, daily high temperature values
x = x.groupby([x.index.month, x.index.date]).DailyMinimumDryBulbTemperature.first()
x.index = x.index.rename(['month', 'date'])
x = x.unstack(level=0)
x.head(100)


# # plot
sns.boxplot(data=x).set_title('Summer Months Have a Narrower Range of Minimum Temperatures')
plt.show()
```



The minimum temperatures per month show a much more pronounced seasonal variation, with the "summer" months showing notably smaller interquartile ranges than the other months.

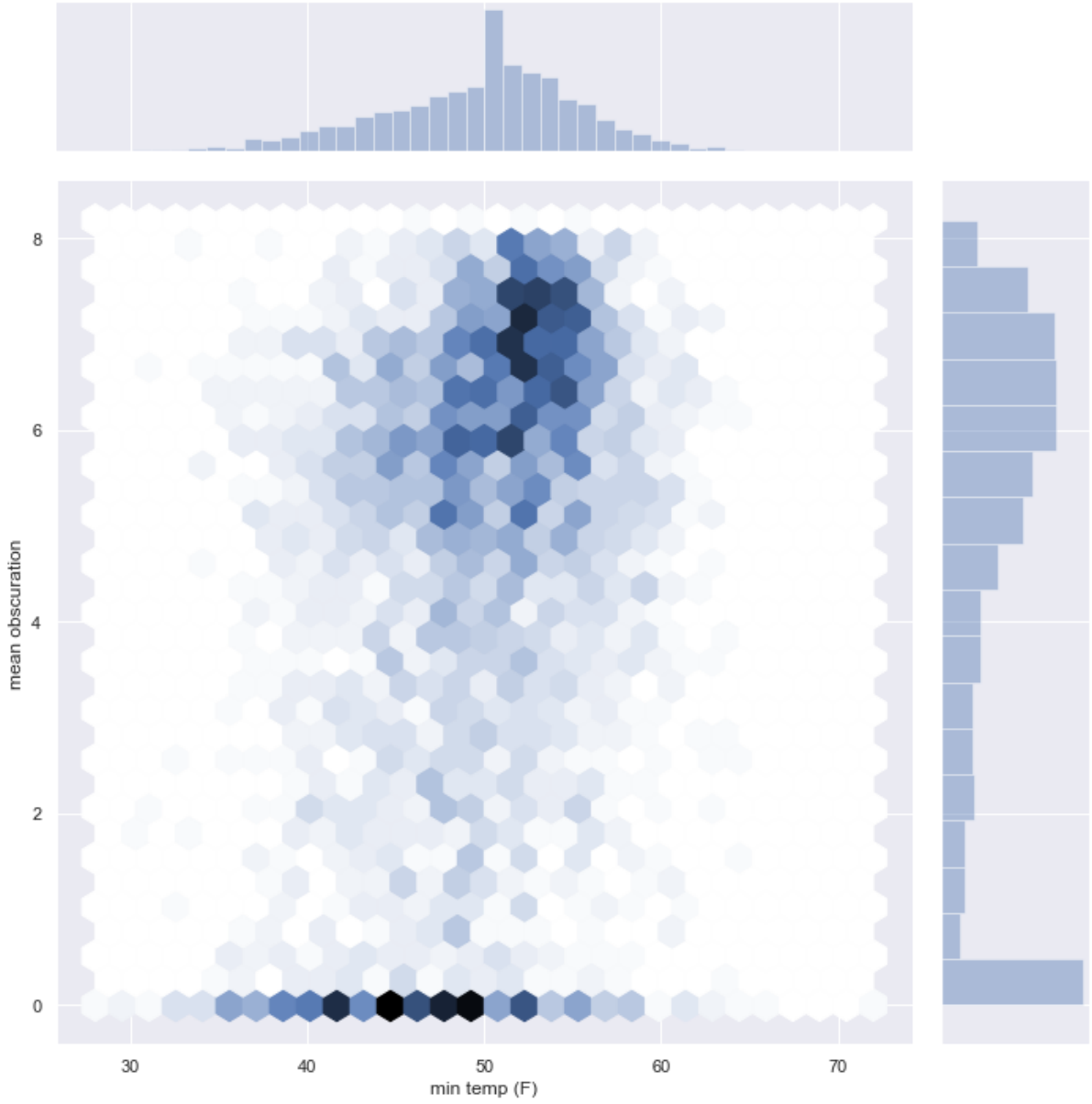
Now let's have a look at the correlation between maximum daily temperature and daily average sky obscuration:

In [319]:
```
# processing setup
x = df
x.reset_index()
x.head()

# wrangle: create average obscuration and max temperature columns
x = pd.DataFrame(x.groupby([x.index.date, x['DailyMaximumDryBulbTemperat
ure']]).averageObscuration.mean())
x.index = x.index.rename(['date', 'max temp (F)'])
x = x.reset_index()
x = x.set_index(['date'])
x.columns = ['max temp (F)', 'mean obscuration']
x.head()

# plot
hexplot = sns.jointplot(x='max temp (F)', y='mean obscuration', height=1
0, data=x, kind='hex')
plt.show()
```



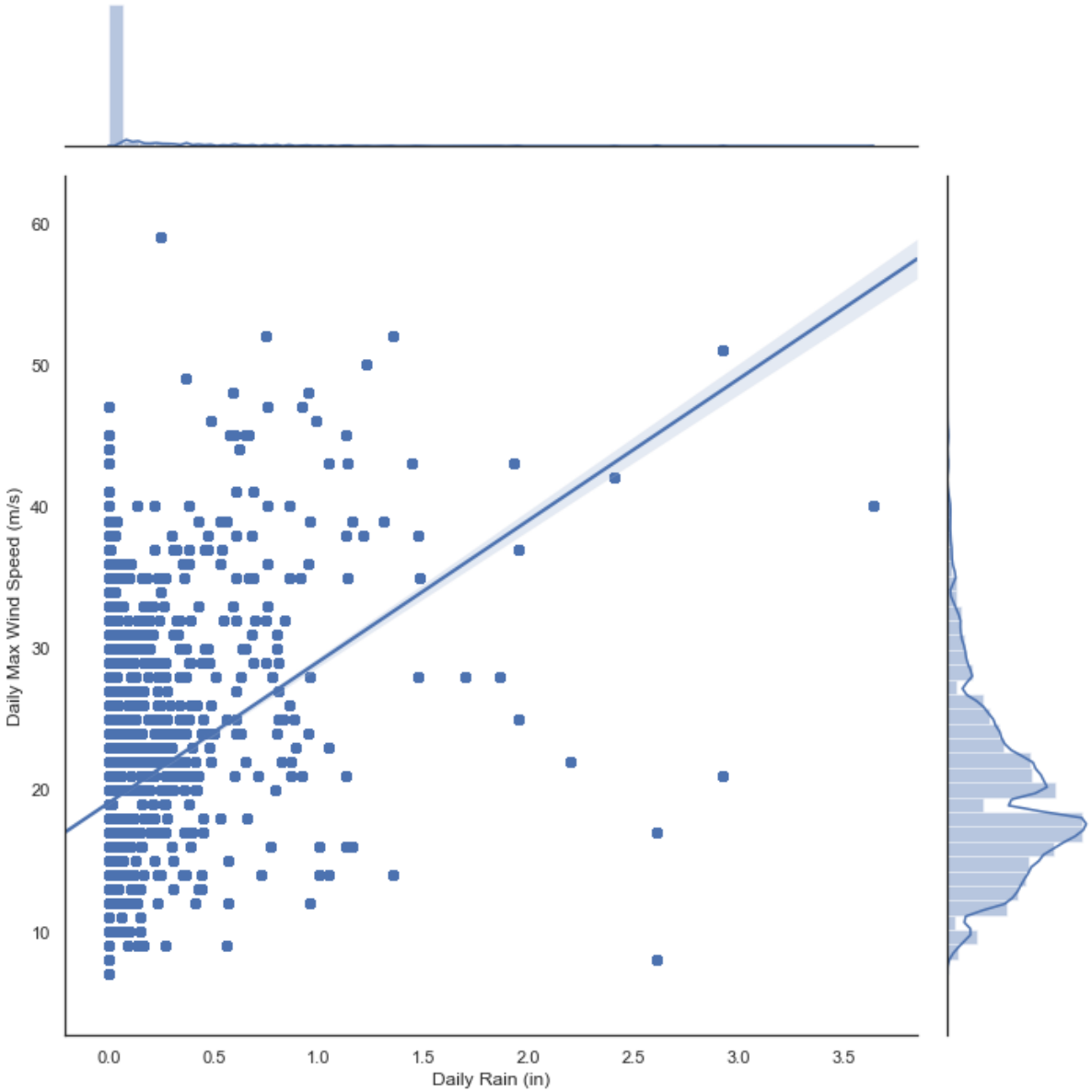
This plot and its accompanying histograms show a strong correlation between a range of temperatures and overcast weather: there are lots of overcast days with a maximum temperature between 60 and 70 degrees.

In [320]:
```
# processing setup
x = df
x.reset_index()
x.head()

# wrangle: create average obscuration and min temperature columns
x = pd.DataFrame(x.groupby([x.index.date, x['DailyMinimumDryBulbTemperat
ure']]).averageObscuration.mean())
x.index = x.index.rename(['date', 'max temp (F)'])
x = x.reset_index()
x = x.set_index(['date'])
x.columns = ['min temp (F)', 'mean obscuration']
x.head()

# plot
hexplot = sns.jointplot(x='min temp (F)', y='mean obscuration', height=1
0, data=x, kind='hex')
plt.show()
```



Lastly, we'll have a look at rainfall and wind patterns.

In [321]:
```
# processing setup
x = df
x.reset_index()
x.head()

# wrangle: rename wind speed and precipitation columns
x = x.rename(columns={'DailyPrecipitation': "Daily Rain (in)", 'DailyPeakWindSpeed': 'Daily Max Wind Speed (m/s)'})

# plot
sns.set_style('white')
j = sns.jointplot(x='Daily Rain (in)', y='Daily Max Wind Speed (m/s)', height=10, data=x, kind='reg')
plt.show()
```



It looks like higher wind speeds correlate with more daily precipitation, but the confidence interval spreads as the amount of rain increases, because we have many fewer data points. It also looks like most days have between 0 and 1 mm of precipitation, which makes sense, as California experienced a historic drought for a large part of the last decade. But in the past few years, California has also seen historic rains, thanks to increasingly common atmospheric river events.

In [322]:
```
x = df[(df.index.year >= 2010) & (df.index.year < 2019)] # choose 2010 t
hrough 2018, because 2009 and 2019 are missing some dates
x = x.groupby([x.index.year, x.index.date]).DailyPrecipitation.sum()
x.index = x.index.rename(['year', 'date'])
x = x[x == 0]
x = x.reset_index()
x = x.groupby('year').count().rename(columns={'DailyPrecipitation':'Days
Without Rain'}).drop(columns=['date'])
x
```

Out[322]:

year	Days Without Rain
2010	265
2011	295
2012	303
2013	341
2014	303
2015	324
2016	295
2017	286
2018	312

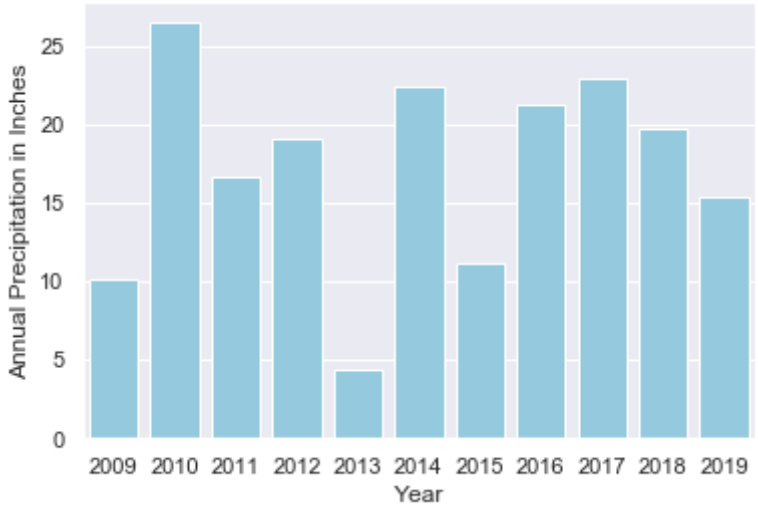
Several online sources report that the average annual rainfall in Monterey is around 500 mm. How do these data align with that figure? The drought and the atmospheric river events should both pull the average lower and higher respectively — which trend wins out?

In [323]:
```
x = df
x = x.groupby([x.index.year, x.index.date])['DailyPrecipitation'].first
()
x.index = x.index.rename(['year', 'date'])
x = x.groupby(['year']).sum().mean()
x
```

Out[323]: 17.19363636363636

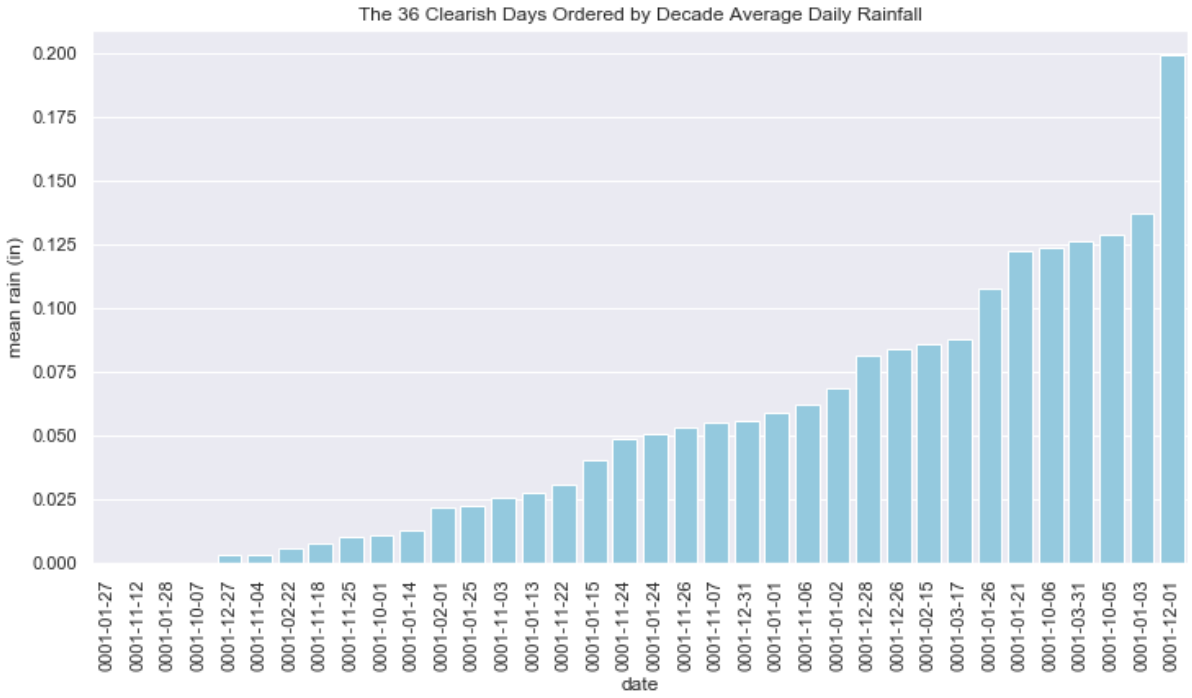
17.2 inches is about 436 mm, which brings in the average a bit below a widely reported average figure.

In [324]:
```
x = df
x = x.groupby([x.index.year, x.index.date])['DailyPrecipitation'].first
()
x.index = x.index.rename(['Year', 'Date'])
x = pd.DataFrame(x.groupby(['Year']).sum())
x.columns = ['Annual Precipitation in Inches']
x = x.reset_index()

# plot
sns.set_style('darkgrid')
sns.barplot(x='Year', y='Annual Precipitation in Inches', color='skyblu
e', data=x)
plt.show()
```

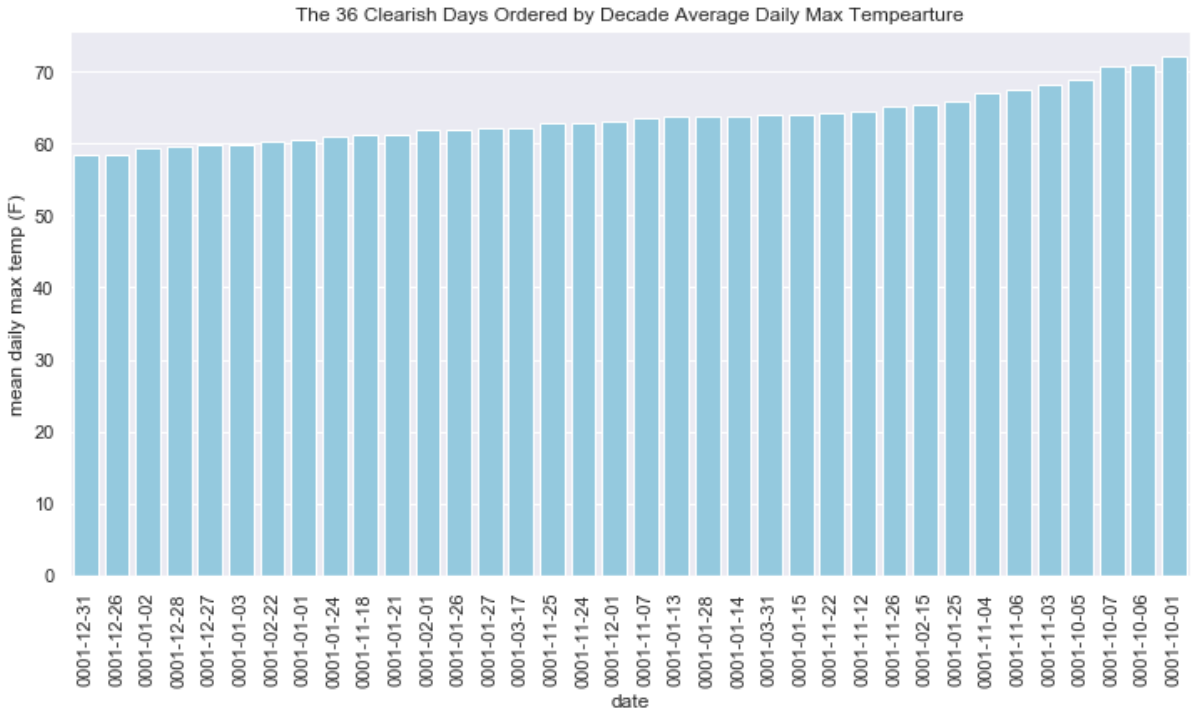


There's been a wide range of annual rainfalls in the last decade, from 4.33 inches in 2013 to 22.85 inches in 2017.

In [325]:
```
plt.figure(figsize=(12, 6))

by_date = df[(df.index.hour >= 10) & (df.index.hour <= 16)] # get 10 AM
 to 4 PM (see definition of 'clearish' above)

# mean the average daily obscuration and keep the first value for daily
 precipitaton for each date throughout decade
by_date = df.groupby([df.index.date]).agg({'DailyPrecipitation': 'first'
, 'averageObscuration': 'mean'})

# further average both obscuration and daily rainfall by calendar day
by_date = df.groupby([df.index.month, df.index.day]).agg({'DailyPrecipit
ation': 'mean', 'averageObscuration': 'mean'})
by_date.index = by_date.index.rename(['month', 'day'])
by_date.columns = ['mean rain (in)', 'mean obscuration']
by_date = by_date.reset_index()

# filter out clearish days
by_date = by_date[by_date['mean obscuration'] <= 3.5] # 3.5 is a conserv
ative cut-off for a clear day: there are at worst "scattered clouds"

# sort by ascending rainfall
by_date = by_date.sort_values(by='mean rain (in)')

# add a date column to serve as the index
by_date['date'] = by_date.apply(lambda x: date(year=1, month=int(x['mont
h']), day=int(x['day'])), axis=1)
by_date


# plot
b = sns.barplot(x='date', y='mean rain (in)', color='skyblue', data=by_d
ate)
b.set_title('The 36 Clearish Days Ordered by Decade Average Daily Rainfa
ll')
plt.xticks(rotation=90)
plt.show()
```

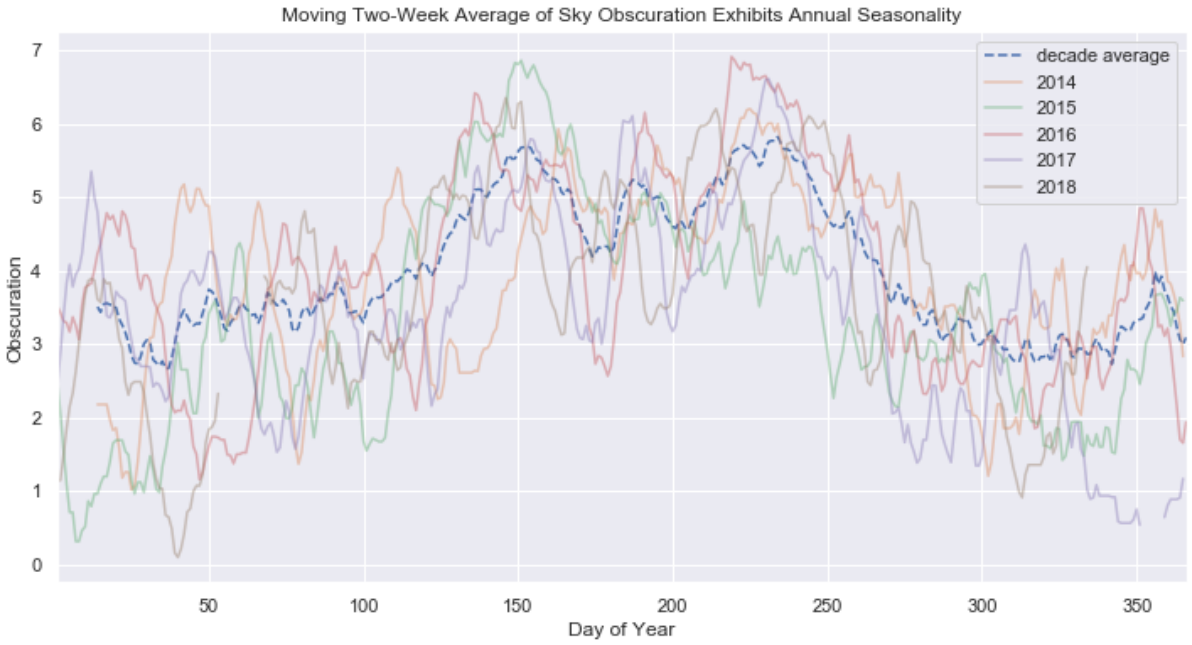

The 36 Clearish Days Ordered by Decade Average Daily Rainfall
mean rain (in)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0001-01-27
0001-11-12
0001-01-28
0001-10-07
0001-12-27
0001-11-04
0001-02-22
0001-11-18
0001-11-25
0001-10-01
0001-01-14
0001-02-01
0001-01-25
0001-11-03
0001-01-13
0001-11-22
0001-01-15
0001-11-24
0001-01-24
0001-11-26
0001-11-07
0001-12-31
0001-01-01
0001-11-06
0001-01-02
0001-12-28
0001-12-26
0001-02-15
0001-03-17
0001-01-26
0001-01-21
0001-10-06
0001-03-31
0001-10-05
0001-01-03
0001-12-01
date

In [326]:
```
plt.figure(figsize=(12, 6))

by_date = df[(df.index.hour >= 10) & (df.index.hour <= 16)] # get 10 AM
 to 4 PM (see definition of 'clearish' above)

# mean the average daily obscuration and keep the first value for daily
 precipitaton for each date throughout decade
by_date = df.groupby([df.index.date]).agg({'DailyMaximumDryBulbTemperatu
re': 'first', 'averageObscuration': 'mean'})

# further average both obscuration and daily rainfall by calendar day
by_date = df.groupby([df.index.month, df.index.day]).agg({'DailyMaximumD
ryBulbTemperature': 'mean', 'averageObscuration': 'mean'})
by_date.index = by_date.index.rename(['month', 'day'])
by_date.columns = ['mean daily max temp (F)', 'mean obscuration']
by_date = by_date.reset_index()

# filter out clearish days
by_date = by_date[by_date['mean obscuration'] <= 3.5] # 3.5 is a conserv
ative cut-off for a clear day: there are at worst "scattered clouds"

# sort by ascending rainfall
by_date = by_date.sort_values(by='mean daily max temp (F)')

# add a date column to serve as the index
by_date['date'] = by_date.apply(lambda x: date(year=1, month=int(x['mont
h']), day=int(x['day'])), axis=1)
by_date


# plot
b = sns.barplot(x='date', y='mean daily max temp (F)', color='skyblue',
data=by_date)
b.set_title('The 36 Clearish Days Ordered by Decade Average Daily Max Te
mpearture')
plt.xticks(rotation=90)
plt.show()
```



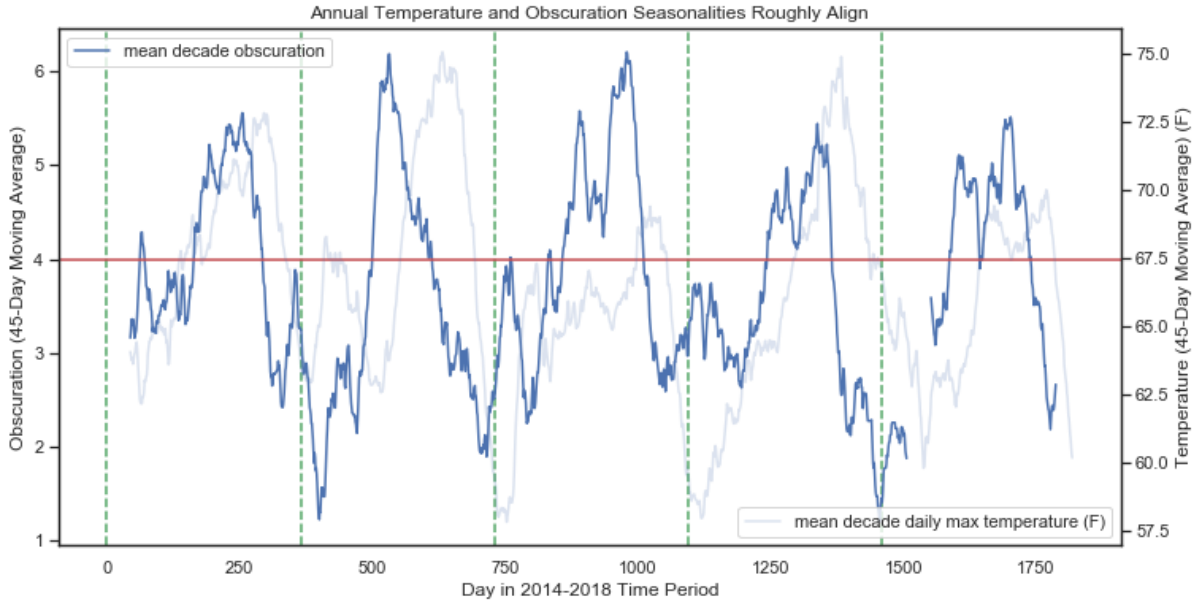
Does average daily sky obscuration act seasonally throughout the year?

In [331]:
```
sns.set()
plt.figure(figsize=(12, 6))

x = df
bool_index = (x.index.year >= 2014) & (x.index.year <= 2018) # only thes
e years have data for all days of the year
x = x[bool_index]

mean_obsc = x.groupby([x.index.year, x.index.dayofyear])['averageObscura
tion'].mean().rolling(14).mean()

mean_obsc_decade = x.groupby([x.index.dayofyear])['averageObscuration'].
mean().rolling(14).mean().plot(label='decade average', linestyle='--')

mean_obsc = mean_obsc.unstack(level=0)

# plot
for col in mean_obsc.columns:
    plt.plot(mean_obsc[col], label=str(col), alpha=.4)
plt.legend()
plt.title('Moving Two-Week Average of Sky Obscuration Exhibits Annual Se
asonality')
plt.xlabel('Day of Year')
plt.ylabel('Obscuration')
plt.show()
```



Sky obscuration seems to roughly follow annual fluctuations in temperature. Let's plot both to see their annual fluctuation:

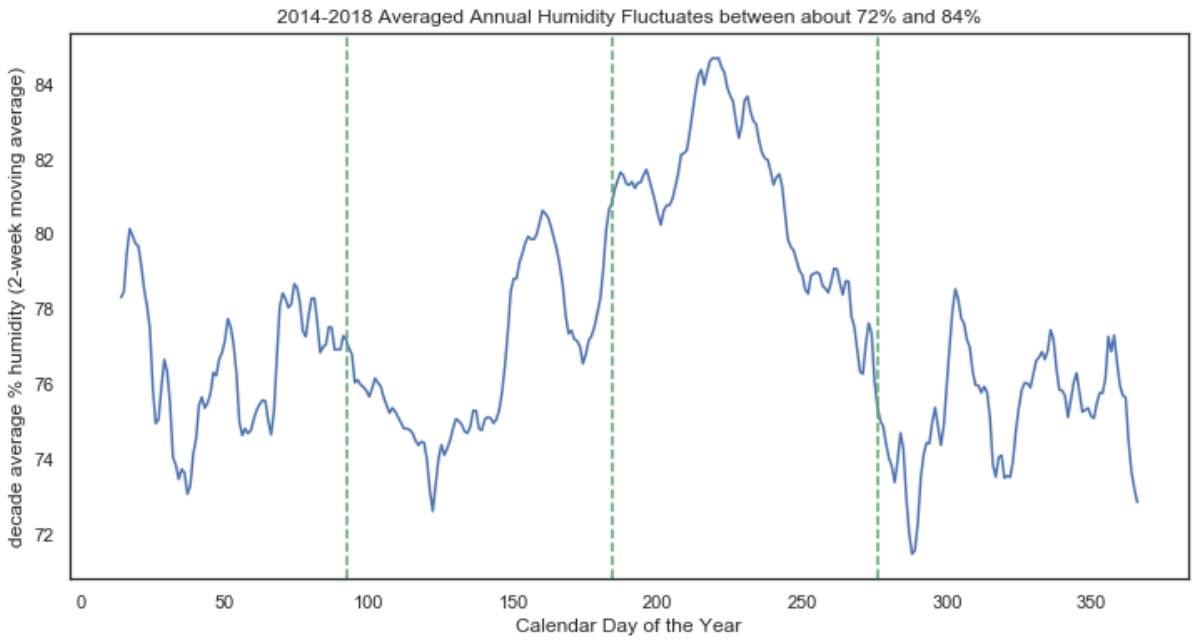
In [333]:
```
fig, ax1 = plt.subplots(figsize=(12, 6))
sns.set_style('white')

x = df
bool_index = (x.index.year >= 2014) & (x.index.year <= 2018) # only thes
e years have data for all days of the year
x = x[bool_index]
x = x[['averageObscuration', 'DailyMaximumDryBulbTemperature']]

days = x.groupby([x.index.year, x.index.dayofyear]).mean().rolling(45).m
ean()
days.index = days.index.rename(['year', 'day'])
days = days.reset_index()
days = days.drop(['year', 'day'], axis=1)

# plot
[ax1.axvline(x, color='g', linestyle='--') for x in [y*365 for y in rang
e(5)]] # show year starts

ax1.plot(days['averageObscuration'], label='mean decade obscuration')
ax1.set_xlabel('Day in 2014-2018 Time Period')
ax1.set_ylabel('Obscuration (45-Day Moving Average)')
ax1.legend(loc='upper left')
ax1.axhline(4, color='r')
ax2 = ax1.twinx() # share x axis, use two separate y axes on left and ri
ght sides
ax2.plot(days['DailyMaximumDryBulbTemperature'], label='mean decade dail
y max temperature (F)', alpha=0.2)
ax2.legend(loc='lower right')
ax2.set_ylabel('Temperature (45-Day Moving Average) (F)')
plt.title('Annual Temperature and Obscuration Seasonalities Roughly Alig
n')
plt.show()
```



While both obscuration and maximum temperature show annually seasonal behavior, their cycles align unpredictably from year to year.

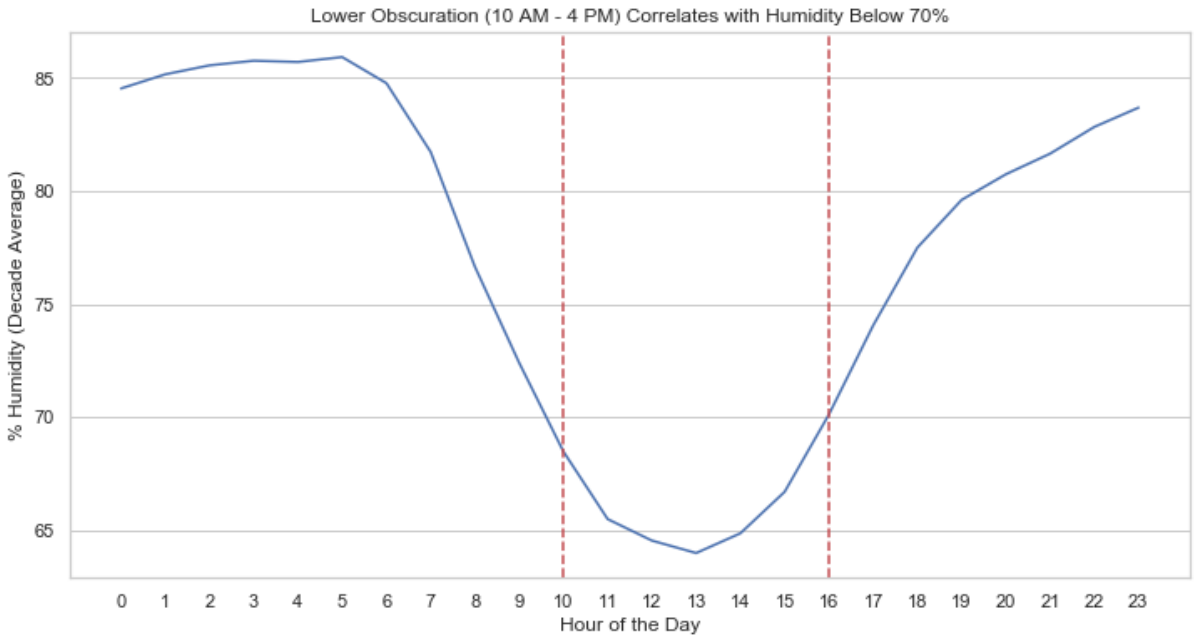
The heat index measures "real feel" temperature as a function of dry bulb temperature and relative percent humidity. Are there substantial humidity fluctuations that should be taken into account with these data?

In [335]:
```
fig, ax1 = plt.subplots(figsize=(12, 6))

x = df
bool_index = (x.index.year >= 2014) & (x.index.year <= 2018) # only thes
e years have data for all days of the year
x = x[bool_index]
x = x.groupby(x.index.dayofyear).mean()['HourlyRelativeHumidity'].rollin
g(14).mean()

plt.plot(x, label='hourly humidity (2-week rolling average)')
[plt.axvline(x, linestyle='--', color='g') for x in [y*92 for y in range
(1, 4)]]
plt.xlabel('Calendar Day of the Year')
plt.ylabel('decade average % humidity (2-week moving average)')
plt.title('2014-2018 Averaged Annual Humidity Fluctuates between about 7
2% and 84%')
plt.show()
```



This plot averages the hourly average humidity per calendar day across the decade and then plots the two-week moving average of the daily result. Given the range of maximum temperatures in the data, this range of humidities (72%-84%) means a heat index felt-temperature fluctuation of one degree or less. It should be suffiicient to equate dry bulb temperature with the felt air temperature.

In [341]:
```
fig, ax1 = plt.subplots(figsize=(12, 6))
sns.set_style('whitegrid')

x = df
bool_index = (x.index.year >= 2014) & (x.index.year <= 2018) # only thes
e years have data for all days of the year
x = x[bool_index]
x = x.groupby([x.index.hour]).mean()['HourlyRelativeHumidity']
x.head()

sns.lineplot(data=x)
plt.xticks(range(24))
plt.xlabel('Hour of the Day')
plt.ylabel('% Humidity (Decade Average)')
ax1.xaxis.grid(which="major")
plt.axvline(10, color='r', linestyle='--')
plt.axvline(16, color='r', linestyle='--')
plt.title('Lower Obscuration (10 AM - 4 PM) Correlates with Humidity Bel
ow 70%')
plt.show()
```



From day to day, the humidity pattern links back to the first plot: the tendency for days to be clearer between 10 AM and 4 PM lines up with the time period during which % humidity tends to be lower than 70%.

Capstone One: Statistical Analysis Report

Interactive documentation of this analysis can be found on Github in this jupyter notebook.

Because the project aims to forecast temperature, obscuration, and humidity, statistical analysis focuses on time series forecasting with three different prediction techniques: ARMA modeling, exponential smoothing, and generalized additive modeling. Models fit daily data for the three variables and then predict future values.

Some preliminary data processing takes place before modeling. Exploratory visualization previously revealed that the hours of 10 AM to 4 PM contained the best sky clarity and temperature for each day; for this reason, analysis considers only hours in this range across the dataset's complete years (2010-2018). Before investigating any of the models, the hourly humidity data needs to be resampled to occur at a daily interval; the resample method applies a mean to each day's 10 AM to 4 PM data to arrive at this figure. The temperature and obscuration data already occur at a daily interval.

Analysis continues from preliminary processing to ARMA process modeling. As ARMA modeling requires time series data to be time stationary, the first step of the analysis investigates the data's stationarity. A Dickey-Fuller test provides a p-value in response to the null hypothesis that the time series is not stationary; the data set's temperature, obscuration, and humidity time series all yield low p-values in the test. This indicates that all three series are stationary and can be fruitfully approached using ARMA modeling.

To get a sense of the proper order of ARMA models, the next stage of analysis examines the three variables' autocorrelation and partial autocorrelation plots. The minimized Bayesian Information Criterion (BIC) determines model orders, after fitting and assessing all possible ARMA models up to a maximum AR order of 4 and a maximum MA order of 2. Lastly, plots compare each variable's ARMA model forecast for the month of January, 2019 against recorded data for that month. Additional techniques that may add more model accuracy may include simple differencing or seasonal differencing in advance of modeling.

Next, exponential smoothing provides alternative forecasting models. Naive STL decomposition provides an initial view of the data's relevant linear and cyclic components. After this, Holt-Winters triple exponential smoothing models each variable's series. For each variable in the data set, plots compare several possible models with changes to the following two variables: (1) damped vs. undamped additive trend modeling, and (2) additive vs. multiplicative seasonal modeling. Damped additive trend and multiplicative seasonality performed best for temperature, while damped additive trend with additive seasonality performed best for humidity data; obscuration model results were ambiguous between damped and undamped additive trend with additive seasonality. An average root mean square error metric helps quantify model assessment. These plots compare forecast 2018 data against observed 2018 data.

The final technique examined utilizes Facebook's Generalized Additive Model, Prophet, to model and predict temperature, obscuration, and humidity values. This technique captures a trend with hierarchical seasonality through a combination of Fourier harmonic analysis with Bayesian smoothing on a piecewise linear model. Prophet's temperature and obscuration

forecasts yielded lower error than Holt-Winters exponential smoothing, while Holt-Winters smoothing narrowly outperforms Prophet’s humidity predictions.

Time Series Analysis of Monterey Airport Weather Data

Contents

1. Setup

In [5]:
```
from datetime import datetime
from random import seed, random

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, coint, arma_order_select
_ic
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing

from fbprophet import Prophet
```

In [6]:
```
# pandas settings
pd.set_option('display.max_columns', 125) # csv contains 124 columns
pd.set_option('display.max_rows', 4000) # display more rows
```

In [7]:
```
df = pd.read_csv('cleaned_df.csv', parse_dates=['datetime'], index_col=[
'datetime'])
df.head()
```

Out[7]:

datetime	DATE	HourlySkyConditions	HourlyVisibility	HourlyDryBulbTemperature	Hou
2009-04-01 00:08:00	2009-04-01T00:08:00	[{'BKN': SkyCondition(obscuration=7, vertical_...	10.0	52.0	
2009-04-01 00:50:00	2009-04-01T00:50:00	[{'SCT': SkyCondition(obscuration=4, vertical_...	9.0	52.0	
2009-04-01 00:54:00	2009-04-01T00:54:00	[{'SCT': SkyCondition(obscuration=4, vertical_...	9.0	50.0	
2009-04-01 01:54:00	2009-04-01T01:54:00	[]	9.0	51.0	
2009-04-01 02:54:00	2009-04-01T02:54:00	[]	9.0	50.0	

I. ARMA Models

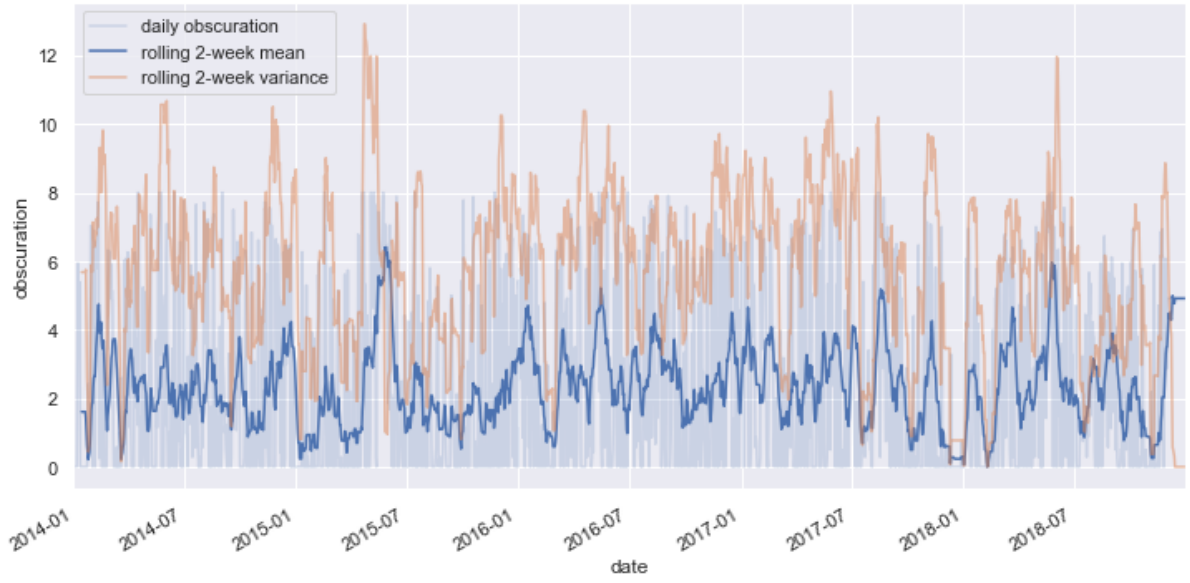
2. Exploring Stationarity

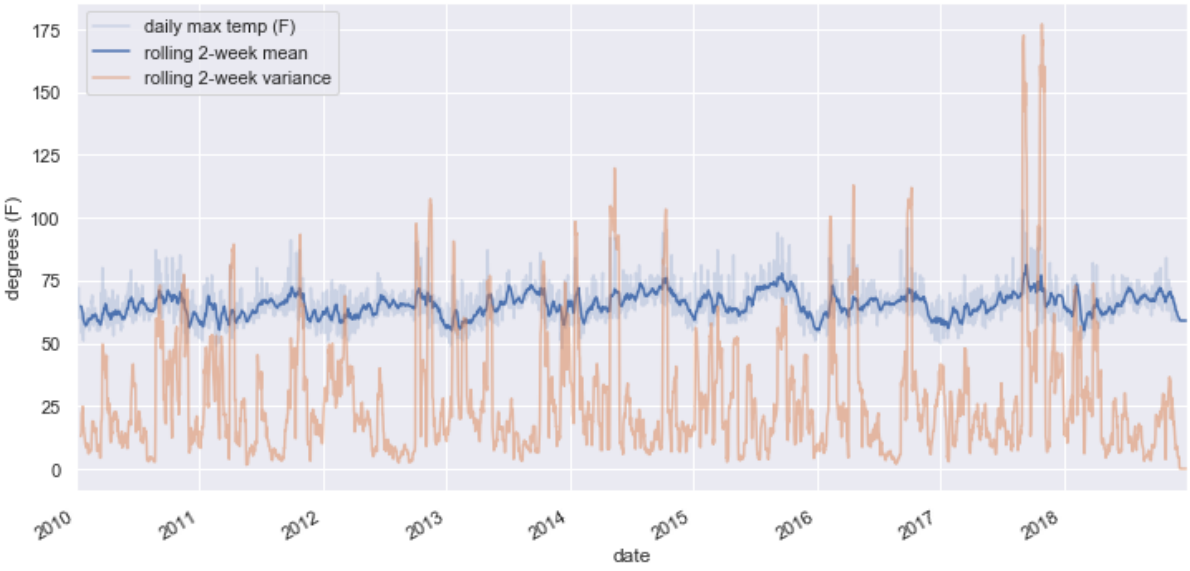
We're primarily modeling the daily maximum temperature, daily average obscuration, and daily humidity variables.

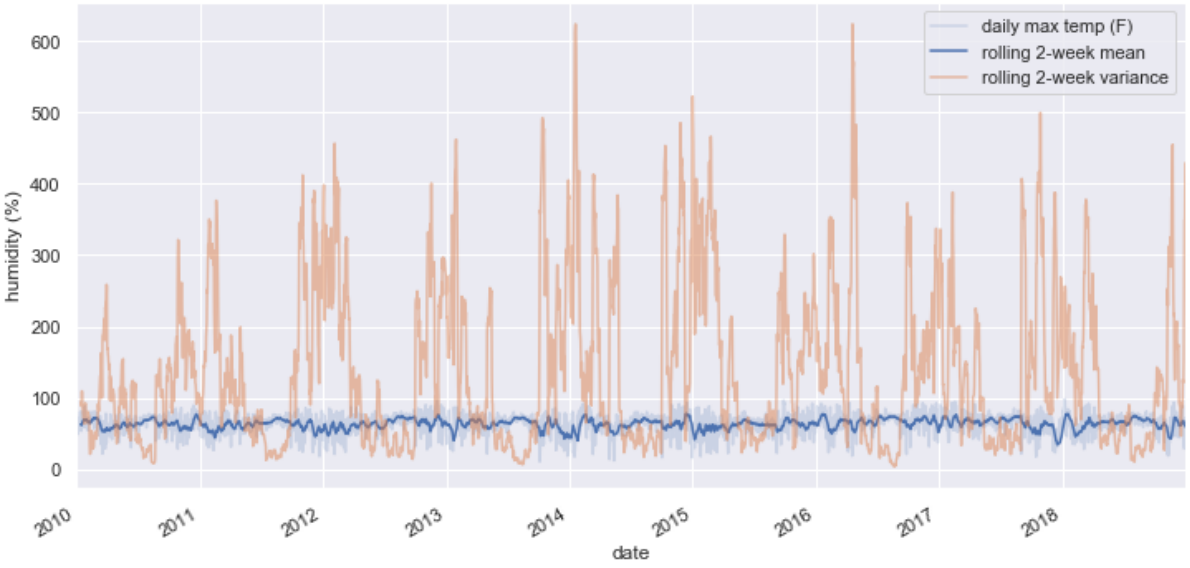
In [8]:
```
x = df
bool_index = (x.index.hour >= 10) & (x.index.hour <= 16) # consider the
 clearest and driest part of each day
x = x[bool_index]

obsc = x['averageObscuration'].resample(rule='D').mean().dropna()
hum = x['HourlyRelativeHumidity'].resample(rule='D').mean().dropna()
temp = x['DailyMaximumDryBulbTemperature'].resample(rule='D').last().dro
pna()
```

In [9]:
```
# we need the 2019 data to assess predictions later
obsc_all = obsc[obsc.index.year == 2019]
hum_all = hum[hum.index.year == 2019]
temp_all = temp[temp.index.year == 2019]
```

In [17]:
```
sns.set()

bool_index = (obsc.index.year >= 2014) & (obsc.index.year <= 2018)
obsc = obsc[bool_index]
bool_index = (temp.index.year >= 2010) & (temp.index.year <= 2018)
temp = temp[bool_index]
bool_index = (hum.index.year >= 2010) & (hum.index.year <= 2018)
hum = hum[bool_index]

plt.figure(figsize=(12, 6))
plt.plot(obsc, color='b', alpha=0.2)
obsc.rolling(14).mean().plot()
obsc.rolling(14).var().plot(alpha=0.5)
plt.xlabel('date')
plt.ylabel('obscuration')
plt.legend(('daily obscuration', 'rolling 2-week mean', 'rolling 2-week
 variance'))
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(temp, color='b', alpha=0.2)
temp.rolling(14).mean().plot()
temp.rolling(14).var().plot(alpha=0.5)
plt.xlabel('date')
plt.ylabel('degrees (F)')
plt.legend(('daily max temp (F)', 'rolling 2-week mean', 'rolling 2-week
variance'))

plt.figure(figsize=(12, 6))
plt.plot(hum, color='b', alpha=0.2)
hum.rolling(14).mean().plot()
hum.rolling(14).var().plot(alpha=0.5)
plt.xlabel('date')
plt.ylabel('humidity (%)')
plt.legend(('daily max temp (F)', 'rolling 2-week mean', 'rolling 2-week
variance'))



plt.show()
```


daily obscuration
rolling 2-week mean
rolling 2-week variance
obscuration
0
2
4
6
8
10
12
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
date


daily max temp (F)
rolling 2-week mean
rolling 2-week variance
degrees (F)
0
25
50
75
100
125
150
175
2010
2011
2012
2013
2014
2015
2016
2017
2018
date


daily max temp (F)
rolling 2-week mean
rolling 2-week variance
humidity (%)
0
100
200
300
400
500
600
2010
2011
2012
2013
2014
2015
2016
2017
2018
date

Does the Dickey-Fuller test think the daily average obscuration variable acts like a random walk? Its null hypothesis is that the series' properties (mean, variance, autocorrelation) change over time (the series is non-stationary).

In [272]:
```
results = adfuller(obsc)
print("p-value is:", results[1])
```

```
p-value is: 0.0
```

We can reject the null hypothesis that the daily average obscuration is non-stationary and have evidence to treat the series as stationary.

How about daily maximum temperature?

In [273]:
```
results = adfuller(temp)
print("p-value is:", results[1])
```

```
p-value is: 1.8819053175423097e-06
```

We can likewise reject the null hypothesis that the Daily Maximum temperature isn't stationary and can treat it as sa stationary series.

How about humidity?

In [274]:
```
# conduct test
results = adfuller(hum)
print("p-value is:", results[1])
```

```
p-value is: 3.488511104743475e-23
```

The humidity is also likely a stationary time series.

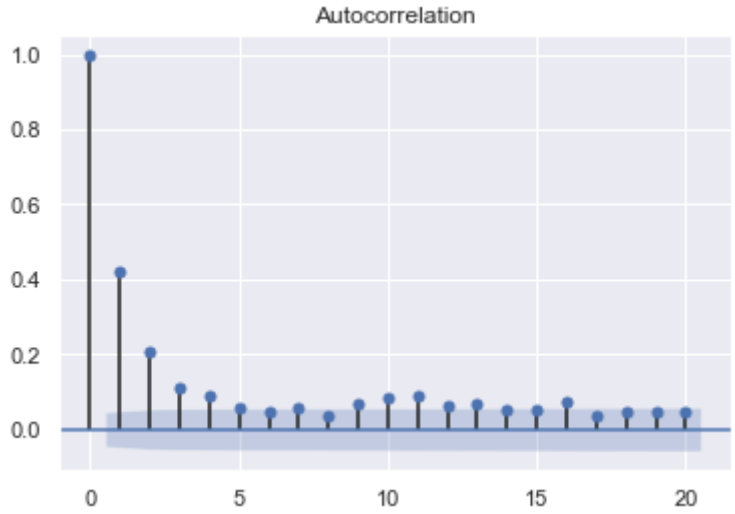
In [275]:
```
# sanity check: does a random walk have a time dependent structure?

# Generate random residuals
np.random.seed(0)
errors = np.random.normal(0, 1, 1000)

# Create AR(1) (random walk) samples for models with and without unit ro
ots
x_unit_root = [0]
x_no_unit_root = [0]
for i in range(len(errors)):
    x_unit_root.append(x_unit_root[-1] + errors[i])
    x_no_unit_root.append(0.9*x_no_unit_root[-1] + errors[i]) # (0.9 is
n't 1, so no unit root)

# Calculate Augmented Dickey-Fuller p-values
adfuller(x_unit_root)[1], adfuller(x_no_unit_root)[1] # good: a random w
alk is non-stationary
```

Out[275]:
```
(0.8925193132739655, 3.8562004970537815e-06)
```

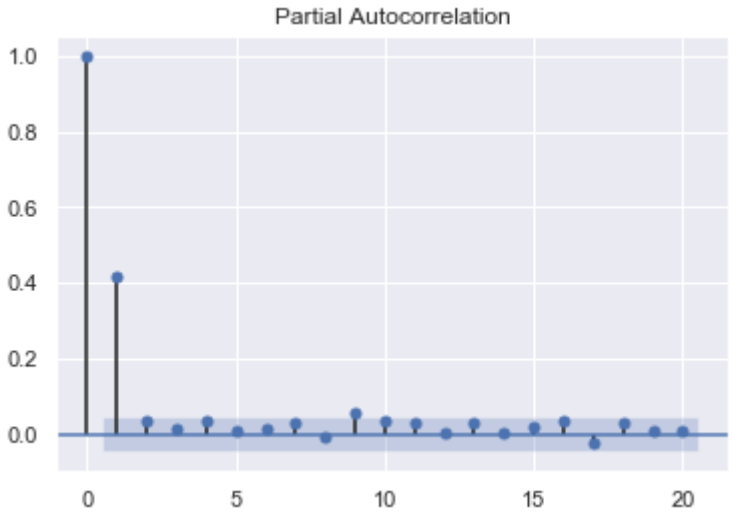
3. Autocorrelation and Partial Autocorrelation

The autocorrelation and partial autocorrelation plots of the variables' lag difference series will help suggest the types of ARMA, ARIMA, or SARIMA models that might be appropriate for modeling the time series.

In [276]:
```
plot_acf(obsc, lags=20)
plt.show()
```



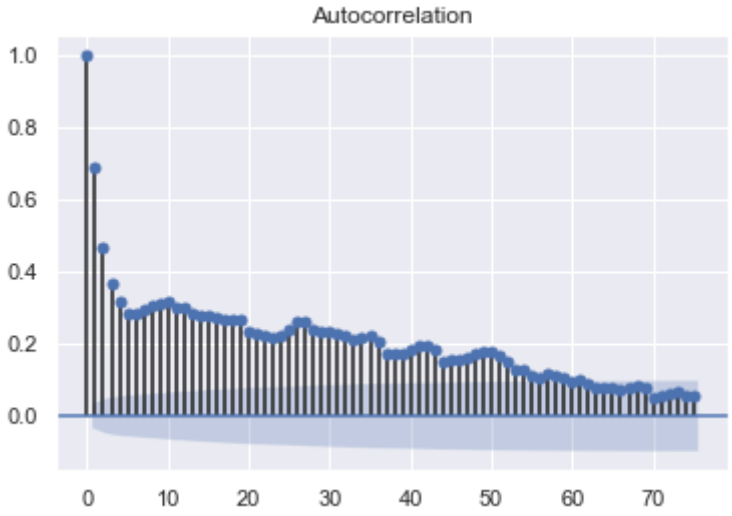
In [277]:
```
plot_pacf(obsc, lags=20)
plt.show()
```



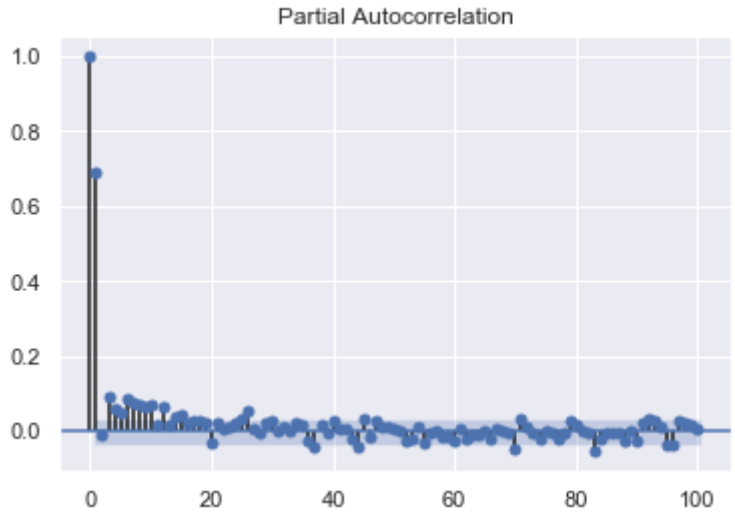
Lags 1-4 show significant values in the auto-correlation plot, which suggests an AR model. The values' positive exponential decay suggests a positive value for ϕ. The partial autocorrelation plot shows a significant value for lag one only, which suggests an AR(1) model.

How about the maximum daily temperature series?

In [278]:
```
plot_acf(temp, lags=75)
plt.show()
```



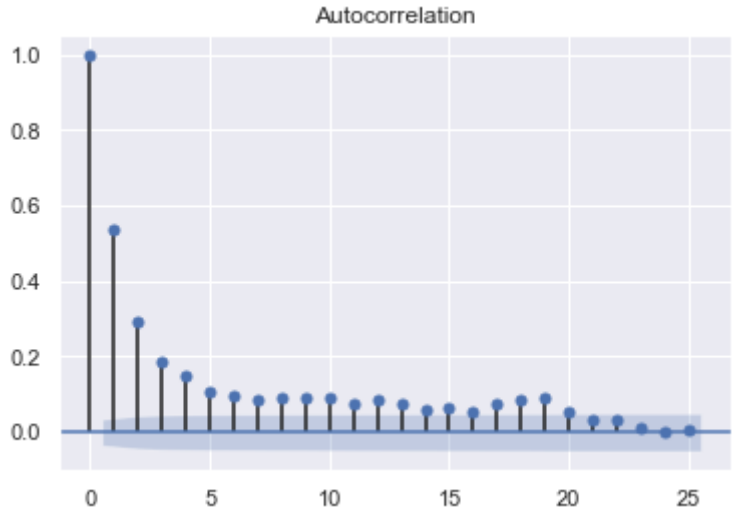
In [279]:
```
plot_pacf(temp, lags=100)
plt.show()
```



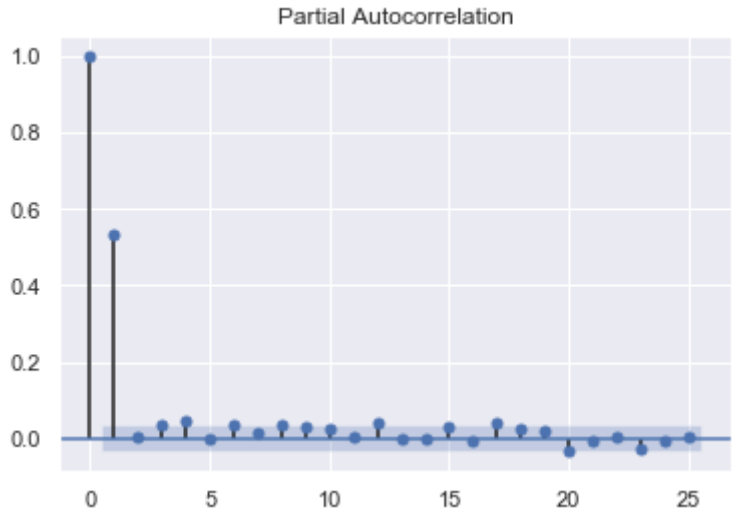
The temperature variable's partial autocorrelation also suggests an AR(1) model.

And humidity?

In [280]:
```
plot_acf(hum, lags=25)
plt.show()
```



In [281]:
```
plot_pacf(hum, lags=25)
plt.show()
```



The humidity variable's partial autocorrelation also suggests an AR(1) model.

We can also use the `arma_order_select_ic` function to try out possible combinations of AR and MA orders (with a default maximum of 4 AR terms and 2 MA terms), to pick the ARMA model with the lowest BIC score:

In [282]:
```
for series in [temp, obsc, hum]:
    result = arma_order_select_ic(series)['bic_min_order']
    print(str(series.name), ": ", result)
```

```
DailyMaximumDryBulbTemperature :  (3, 1)
averageObscuration :  (1, 0)
HourlyRelativeHumidity :  (2, 2)
```

4. ARMA Modeling

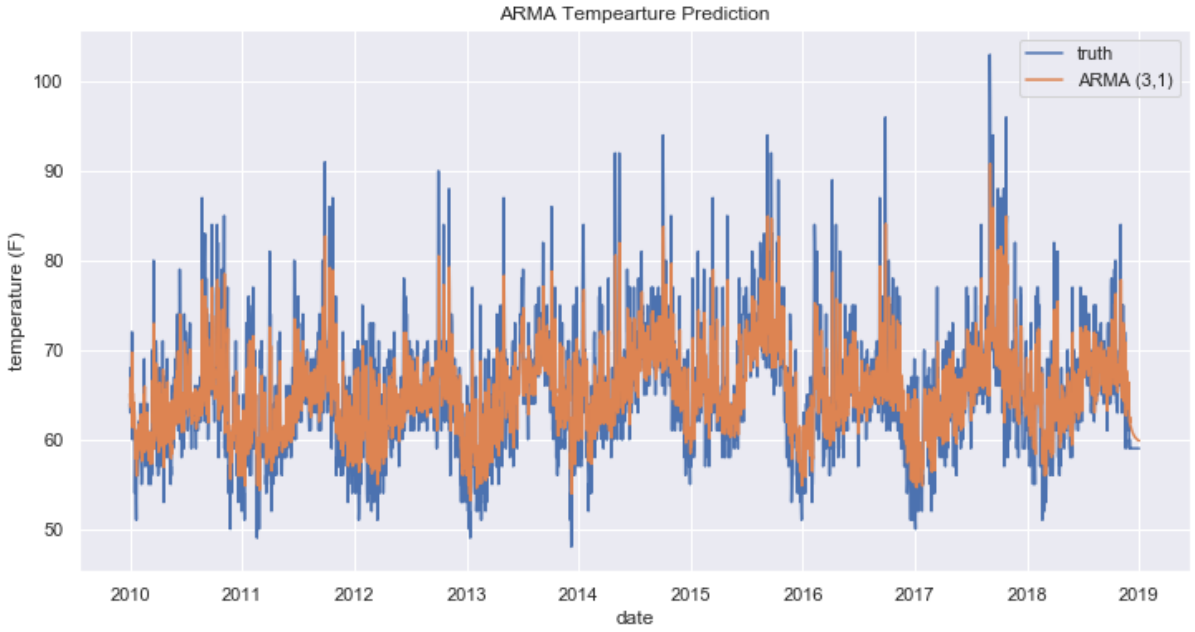
Temperature

In [283]:
```
# Fit an ARMA model to the first simulated data
model = ARMA(temp, order=(3,1)) # fit to ARMA model
fitted = model.fit()

# Print out summary information on the fit
print(fitted.summary())

# Print out the estimate for the constant and for phi
print("The estimate of phi (and the constant) are:")
print(fitted.params)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
                                 ARMA Model Results
==============================================================================
==================
Dep. Variable:     DailyMaximumDryBulbTemperature   No. Observations:
3280
Model:                                ARMA(3, 1)   Log Likelihood
-9736.828
Method:                                   css-mle   S.D. of innovations
4.709
Date:                             Wed, 03 Jul 2019   AIC
19485.656
Time:                                     19:43:17   BIC
19522.230
Sample:                                          0   HQIC
19498.752

==============================================================================
================================
                                                 coef    std err          z
P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
--------------------------------
const                                         65.0788      0.787     82.734
0.000      63.537      66.620
ar.L1.DailyMaximumDryBulbTemperature     1.5797      0.020     78.791
0.000       1.540       1.619
ar.L2.DailyMaximumDryBulbTemperature    -0.6925      0.030    -22.718
0.000      -0.752      -0.633
ar.L3.DailyMaximumDryBulbTemperature     0.1053      0.018      5.856
0.000       0.070       0.141
ma.L1.DailyMaximumDryBulbTemperature    -0.9267      0.011    -87.274
0.000      -0.948      -0.906
                                    Roots
==============================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------------
------
AR.1            1.0149           -0.0000j            1.0149           -
0.0000
AR.2            2.7803           -1.2748j            3.0586           -
0.0684
AR.3            2.7803           +1.2748j            3.0586
0.0684
MA.1            1.0791           +0.0000j            1.0791
0.0000
------------------------------------------------------------------------------
------
The estimate of phi (and the constant) are:
const                                         65.078762
ar.L1.DailyMaximumDryBulbTemperature     1.579654
ar.L2.DailyMaximumDryBulbTemperature    -0.692521
ar.L3.DailyMaximumDryBulbTemperature     0.105318
ma.L1.DailyMaximumDryBulbTemperature    -0.926692
dtype: float64
```

In [289]:
```
# forecast the past...
cast = fitted.predict(start='01-01-2010', end='12-31-2018')
fig, ax = plt.subplots(figsize=(12, 6))
plt.plot(temp, label='truth')
plt.plot(cast, label='ARMA (3,1)')
plt.xlabel('date')
plt.ylabel('temperature (F)')
plt.title('ARMA Tempearture Prediction')
plt.legend()
plt.show()
```



And now we finally use the model to forecast beyond the data. An ARMA forecast will quickly converge to a single value:

In [290]:
```
forecast = fitted.forecast(31)[0]
```

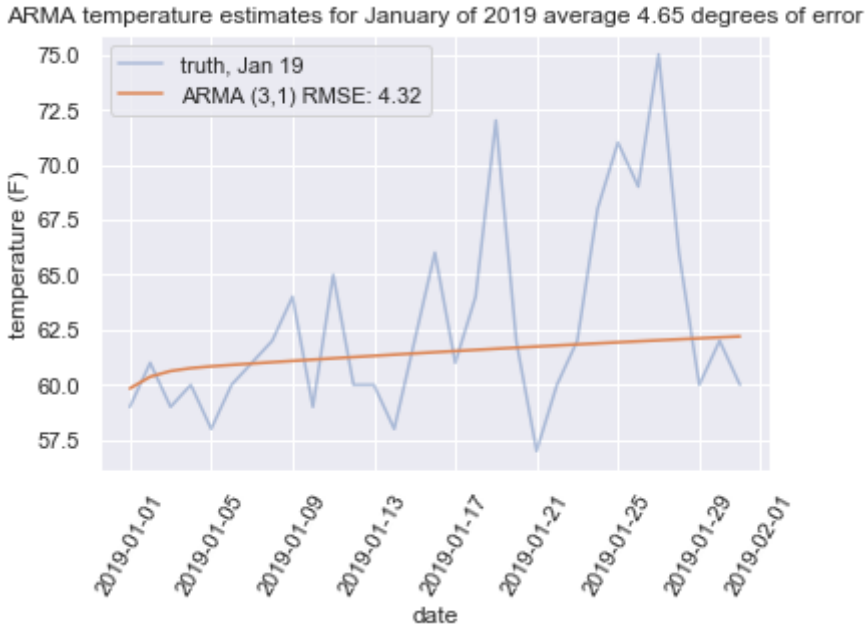
How does this model compare to the real January 2019 data? Let's compute the root mean square error (RMSE) and overlay the values:

In [291]:
```
# get January of 2019 values
x = df
jan_nineteen = x[x.index.year == 2019]
jan_nineteen = jan_nineteen[(jan_nineteen.index.hour >= 10) & (jan_ninet
een.index.hour <= 16)]
jan_nineteen = jan_nineteen['DailyMaximumDryBulbTemperature'].resample(r
ule='D').last().dropna()
jan_nineteen = jan_nineteen[jan_nineteen.index.month == 1]
```

In [292]:
```
# calculate error
rmse = np.sqrt(np.mean(np.square(forecast - jan_nineteen.values)))
```

In [293]:
```
# give predicted values a datetime index
index = pd.date_range(start='01-01-2019', end='01-31-2019')
jan_predicted = pd.DataFrame(forecast)
jan_predicted = jan_predicted.set_index(index)
```

In [294]:
```
# overlay predicted values with measured values
plt.plot(jan_nineteen, alpha=.4, label='truth, Jan 19')
plt.plot(jan_predicted, label='ARMA (3,1) RMSE: {:0.2f}'.format(rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('temperature (F)')
plt.title('ARMA temperature estimates for January of 2019 average 4.65 d
egrees of error')
plt.show()
```



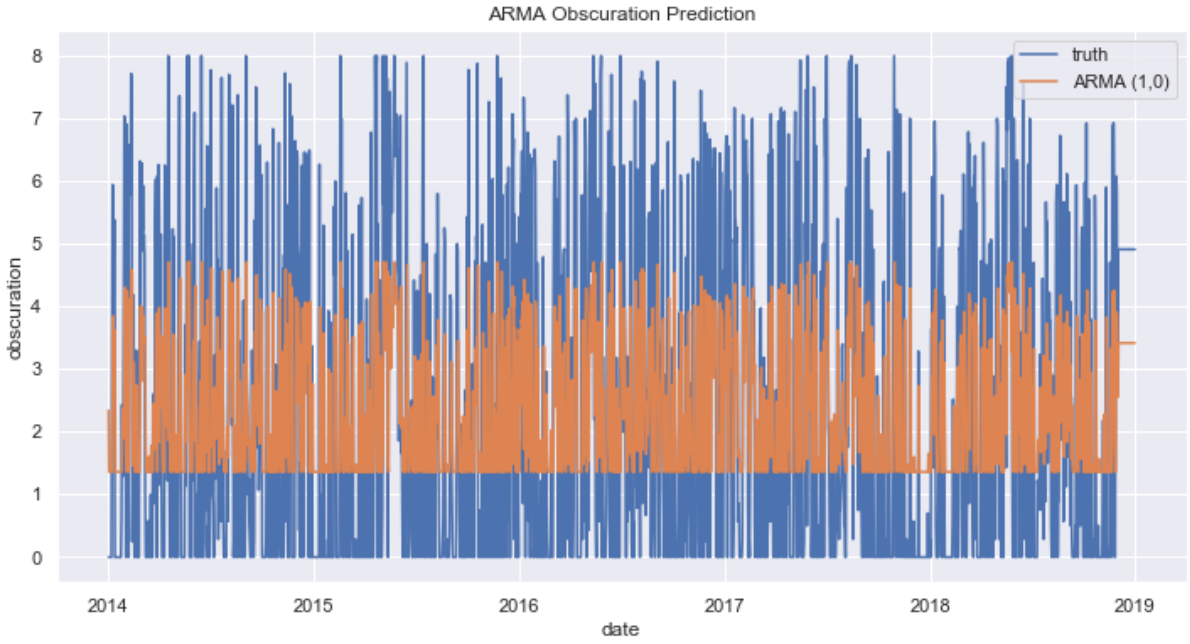
This means predictions will be off by around 25\% of the real value on average.

Obscuration

In [309]:
```
# Fit an ARMA model to the first simulated data
model = ARMA(obsc, order=(1,0)) # fit to ARMA model
fitted = model.fit()

# Print out summary information on the fit
print(fitted.summary())

# Print out the estimate for the constant and for phi
print("The estimate of phi (and the constant) are:")
print(fitted.params)
```

```
                              ARMA Model Results
==============================================================================
=======
Dep. Variable:     averageObscuration   No. Observations:
1819
Model:                     ARMA(1, 0)   Log Likelihood               -4
122.274
Method:                       css-mle   S.D. of innovations
2.333
Date:                Wed, 03 Jul 2019   AIC                           8
250.548
Time:                        19:56:44   BIC                           8
267.066
Sample:                             0   HQIC                          8
256.642

==============================================================================
=====================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
---------------------
const                          2.3394      0.094     24.872      0.000
2.155       2.524
ar.L1.averageObscuration       0.4186      0.021     19.658      0.000
0.377       0.460
                                    Roots
==============================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------------
------
AR.1            2.3888           +0.0000j            2.3888
0.0000
------------------------------------------------------------------------------
------
The estimate of phi (and the constant) are:
const                        2.339438
ar.L1.averageObscuration     0.418623
dtype: float64


/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [310]:
```
# forecast the past...
cast = fitted.predict(start='01-01-2014', end='12-31-2018')
fig, ax = plt.subplots(figsize=(12, 6))
plt.plot(obsc, label='truth')
plt.plot(cast, label= 'ARMA (1,0)')
plt.xlabel('date')
plt.ylabel('obscuration')
plt.title('ARMA Obscuration Prediction')
plt.legend()
plt.show()
plt.show()
```



And now we finally use the model to forecast beyond the data. An ARMA forecast will quickly converge to a single value:

In [311]:
```
forecast = fitted.forecast(31)[0]
```

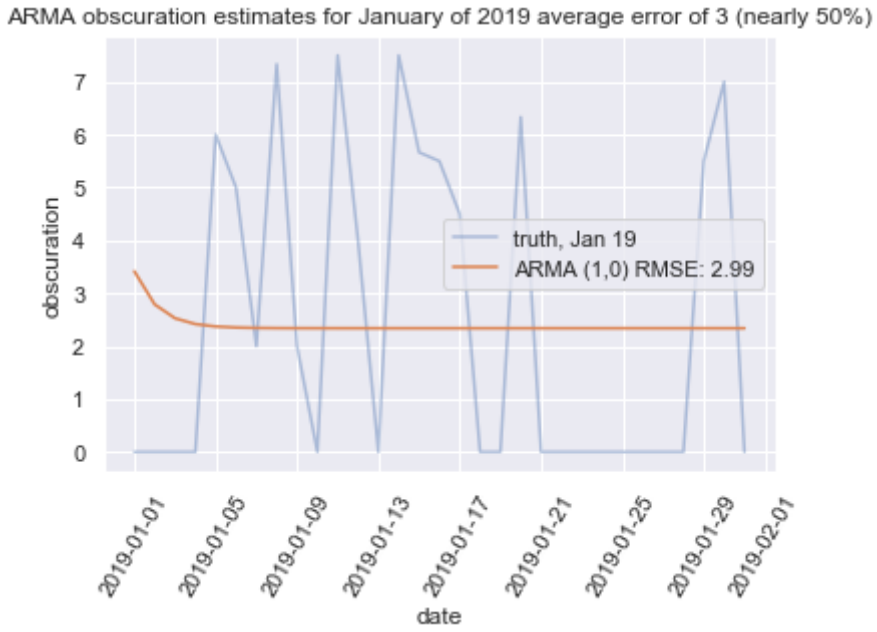
How does this model compare to the real January 2019 data? Let's compute the root mean square error (RMSE) and overlay the values:

In [312]:
```
# get January of 2019 values
x = df
jan_nineteen = x[x.index.year == 2019]
jan_nineteen = jan_nineteen[(jan_nineteen.index.hour >= 10) & (jan_ninet
een.index.hour <= 16)]
jan_nineteen = jan_nineteen['averageObscuration'].resample(rule='D').las
t().dropna()
jan_nineteen = jan_nineteen[jan_nineteen.index.month == 1]
```

In [313]:
```
# calculate error
rmse = np.sqrt(np.mean(np.square(forecast - jan_nineteen.values)))
```

In [314]:
```
# give predicted values a datetime index
index = pd.date_range(start='01-01-2019', end='01-31-2019')
jan_predicted = pd.DataFrame(forecast)
jan_predicted = jan_predicted.set_index(index)
```

In [315]:
```
# overlay predicted values with measured values
plt.plot(jan_nineteen, alpha=.4, label='truth, Jan 19')
plt.plot(jan_predicted, label='ARMA (1,0) RMSE: {:0.2f}'.format(rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('obscuration')
plt.title('ARMA obscuration estimates for January of 2019 average error
 of 3 (nearly 50%)')
plt.show()
```



This means predictions will be off by around 50 \% of the real value on average.

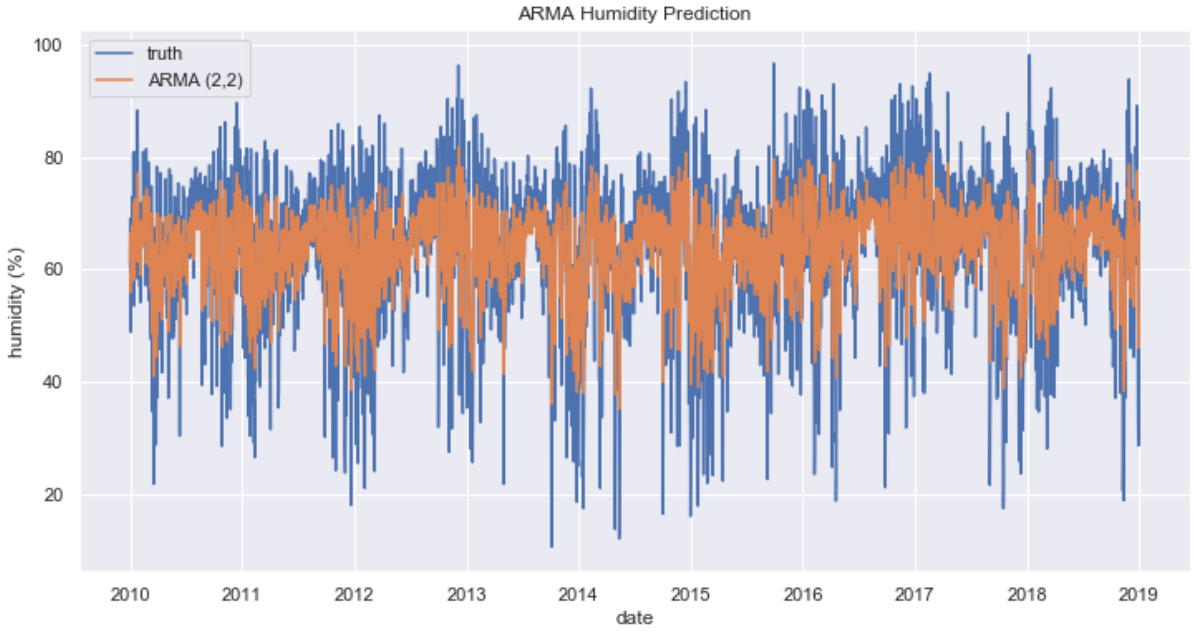
Humidity

In [316]:
```
# Fit an ARMA model to the first simulated data
model = ARMA(hum, order=(2, 2)) # fit to ARMA model
fitted = model.fit()

# Print out summary information on the fit
print(fitted.summary())

# Print out the estimate for the constant and for phi
print("The estimate of phi (and the constant) are:")
print(fitted.params)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
                                ARMA Model Results
==============================================================================
==========
Dep. Variable:     HourlyRelativeHumidity   No. Observations:
3280
Model:                         ARMA(2, 2)   Log Likelihood
-12561.610
Method:                           css-mle   S.D. of innovations
11.142
Date:                    Wed, 03 Jul 2019   AIC
25135.221
Time:                            19:57:17   BIC
25171.794
Sample:                                 0   HQIC
25148.317

==============================================================================
=========================
                                   coef    std err          z      P>|z
|      [0.025      0.975]
------------------------------------------------------------------------------
-------------------------
const                           63.5041      0.667     95.243      0.00
0      62.197      64.811
ar.L1.HourlyRelativeHumidity     1.3828      0.059     23.536      0.00
0       1.268       1.498
ar.L2.HourlyRelativeHumidity    -0.4061      0.049     -8.275      0.00
0      -0.502      -0.310
ma.L1.HourlyRelativeHumidity    -0.8574      0.061    -14.162      0.00
0      -0.976      -0.739
ma.L2.HourlyRelativeHumidity    -0.0625      0.037     -1.677      0.09
4      -0.136       0.011
                                    Roots
==============================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------------
------
AR.1            1.0421           +0.0000j            1.0421
0.0000
AR.2            2.3630           +0.0000j            2.3630
0.0000
MA.1            1.0811           +0.0000j            1.0811
0.0000
MA.2          -14.8042           +0.0000j           14.8042
0.5000
------------------------------------------------------------------------------
------
The estimate of phi (and the constant) are:
const                           63.504110
ar.L1.HourlyRelativeHumidity     1.382831
ar.L2.HourlyRelativeHumidity    -0.406110
ma.L1.HourlyRelativeHumidity    -0.857446
ma.L2.HourlyRelativeHumidity    -0.062482
dtype: float64
```

In [319]:
```
# forecast the past...
cast = fitted.predict(start='01-01-2010', end='12-31-2018')
fig, ax = plt.subplots(figsize=(12, 6))
plt.plot(hum, label='truth')
plt.plot(cast, label='ARMA (2,2)')
plt.xlabel('date')
plt.ylabel('humidity (%)')
plt.title('ARMA Humidity Prediction')
plt.legend()
plt.show()
plt.show()
```



And now we finally use the model to forecast beyond the data. An ARMA forecast will quickly converge to a single value:

In [320]:
```
forecast = fitted.forecast(31)[0]
```

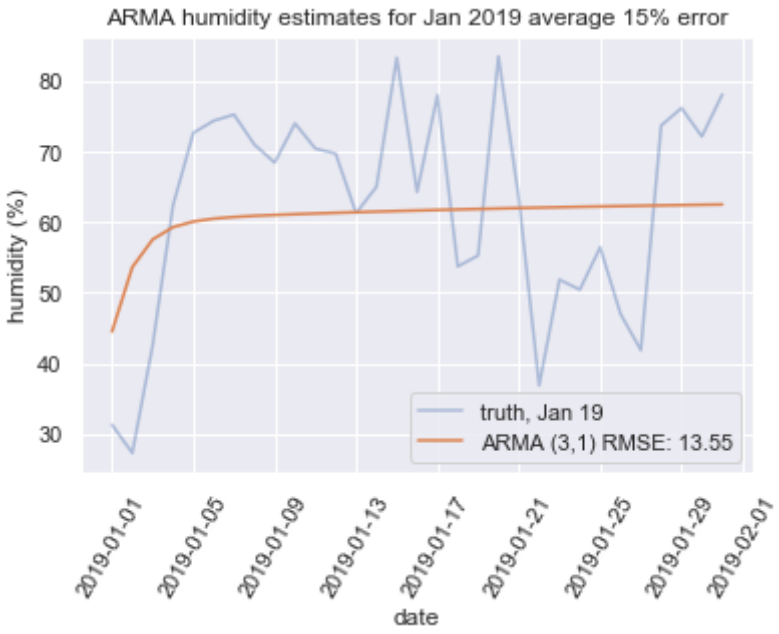
How does this model compare to the real January 2019 data? Let's compute the root mean square error (RMSE) and overlay the values:

In [321]:
```
# get January of 2019 values
x = df
jan_nineteen = x[x.index.year == 2019]
jan_nineteen = jan_nineteen[(jan_nineteen.index.hour >= 10) & (jan_ninet
een.index.hour <= 16)]
jan_nineteen = jan_nineteen['HourlyRelativeHumidity'].resample(rule='D')
.mean().dropna()
jan_nineteen = jan_nineteen[jan_nineteen.index.month == 1]
```

In [322]:
```
# calculate error
rmse = np.sqrt(np.mean(np.square(forecast - jan_nineteen.values)))
```

In [323]:
```
# give predicted values a datetime index
index = pd.date_range(start='01-01-2019', end='01-31-2019')
jan_predicted = pd.DataFrame(forecast)
jan_predicted = jan_predicted.set_index(index)
```

In [324]:
```
# overlay predicted values with measured values
plt.plot(jan_nineteen, alpha=.4, label='truth, Jan 19')
plt.plot(jan_predicted, label='ARMA (3,1) RMSE: {:0.2f}'.format(rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('humidity (%)')
plt.title('ARMA humidity estimates for Jan 2019 average 15% error')
plt.show()
```



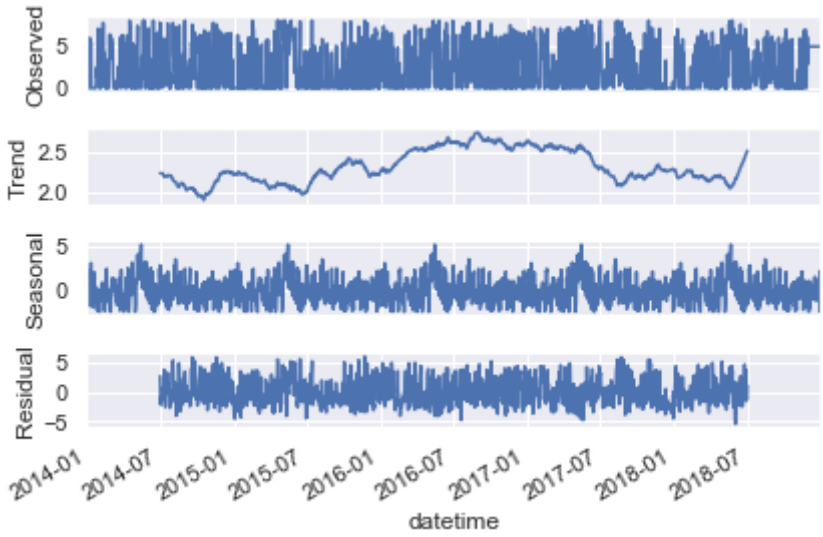
15\% humidity off in a 50\% humidity range means the predictions are on average around 30\% wrong.

II. Exponential Smoothing Models

5. Naive Season-Trend-Level Decomposition

Before applying more sophisticated techniques, the first approach will use a simple moving average technique.

In [325]:
```
results = seasonal_decompose(obsc, model='additive', freq=365)
results.plot()
plt.show()
```



This approach separates out stationary trend, seasonal, and residual components.

In [327]:
```
df_results = adfuller(results.trend.dropna())
print("trend df p-value is:", df_results[1]) # trend isn't stationary (i
t trends)

df_results = adfuller(results.seasonal.dropna())
print("seasonal df p-value is:", df_results[1]) # seasonality is

df_results = adfuller(results.resid.dropna())
print("residuals p-value is:", df_results[1]) # so are residuals
```

```
trend df p-value is: 0.4488899725806956
seasonal df p-value is: 2.7805536590827702e-11
residuals p-value is: 0.0
```

In [329]:
```
results = seasonal_decompose(temp, model='additive', freq=365)
results.plot()
plt.show()
```

In [330]:
```
df_results = adfuller(results.trend.dropna())
print("trend df p-value is:", df_results[1]) # trend isn't stationary (i
t trends)

df_results = adfuller(results.seasonal.dropna())
print("seasonal df p-value is:", df_results[1]) # seasonality is

df_results = adfuller(results.resid.dropna())
print("residuals p-value is:", df_results[1]) # so are residuals
```

```
trend df p-value is: 0.5280017871852764
seasonal df p-value is: 0.0001072283921320691
residuals p-value is: 0.0
```

6. Holt-Winters Seasonal Smoothing

6a. Model Comparison

Next, we fit a Holt-Winters exponential smoothing model to the temperature data. We will use the Akaike Information Criterion to assess performance, and we will plot 2018 predictions over 2018's real values.

In [331]:
```
# separate data into train and test sets
train = temp[:-365]
test = temp.iloc[-365:]
```

In [332]:
```
# initialize models
model1 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365)
model2 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365, damped=True)
model3 = ExponentialSmoothing(train, trend='add', seasonal='mul', season
al_periods=365, damped=True)
```

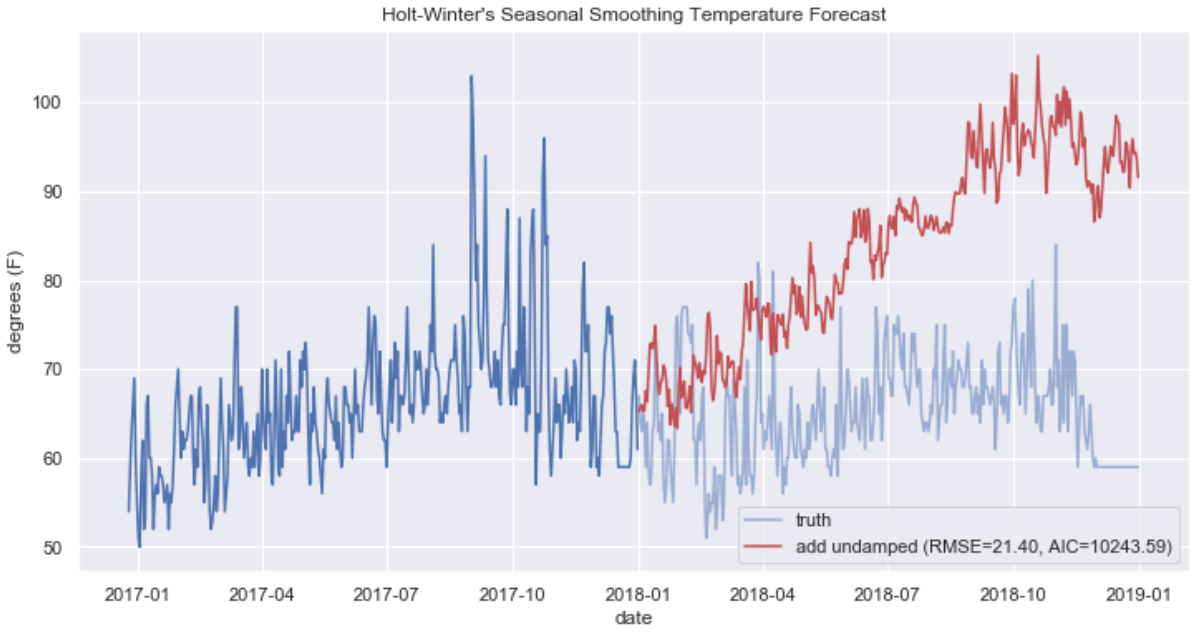
```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

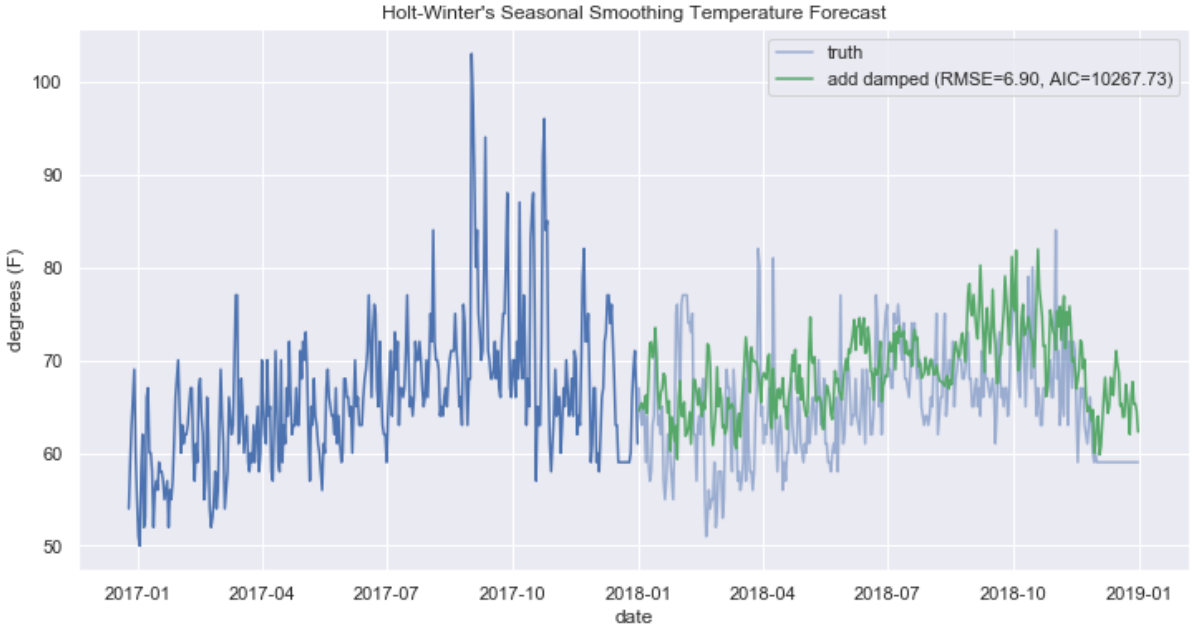
In [333]:
```
# fit models to data
fit1 = model1.fit()
cast1 = fit1.forecast(365)
fit2 = model2.fit()
cast2 = fit2.forecast(365)
fit3 = model3.fit()
cast3 = fit3.forecast(365)
```

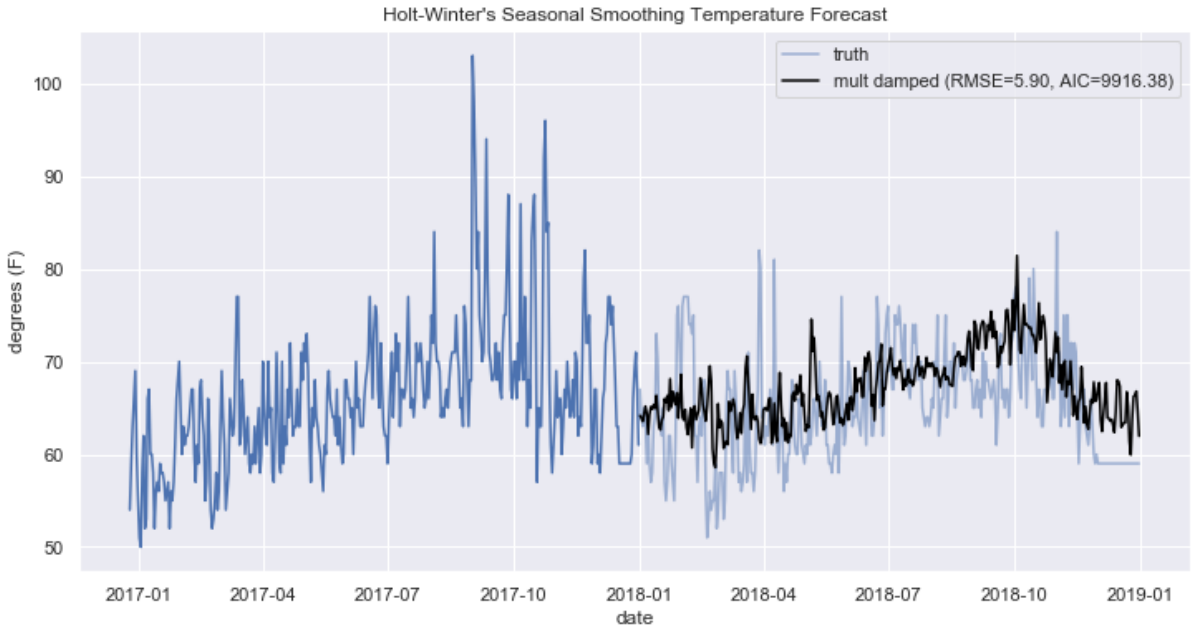
```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [334]:
```
# calculate error
sse1 = np.sqrt(np.mean(np.square(test.values - cast1.values)))
sse2 = np.sqrt(np.mean(np.square(test.values - cast2.values)))
sse3 = np.sqrt(np.mean(np.square(test.values - cast3.values)))
```

In [335]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast1, color='r', label="add undamped (RMSE={:0.2f},
AIC={:0.2f})".format(sse1, fit1.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Temperature Forecast");
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast2, color='g', label="add damped (RMSE={:0.2f}, A
IC={:0.2f})".format(sse2, fit2.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Temperature Forecast");
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast3, color='black', label="mult damped (RMSE={:0.2
f}, AIC={:0.2f})".format(sse3, fit3.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Temperature Forecast");
plt.show()
```


Holt-Winter's Seasonal Smoothing Temperature Forecast
degrees (F)
100
90
80
70
60
50
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01
date
truth
add undamped (RMSE=21.40, AIC=10243.59)


Holt-Winter's Seasonal Smoothing Temperature Forecast
degrees (F)
100
90
80
70
60
50
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01
date
truth
add damped (RMSE=6.90, AIC=10267.73)



The undamped additive model goes off the rails, and both damped models perform better. As predicted by Hyndman and Athanasopoulos (https://otexts.com/fpp2/holt-winters.html), the model with damped trend and multiplicative seasonality performs the best by both error and AIC measures.

We repeat the same process for the obscuration data.

In [336]:
```
# separate data into train and test sets
train = obsc.iloc[:-365]
test = obsc.iloc[-365:]
```

In [337]:
```
# initialize models
model1 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365)
model2 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365, damped=True)
```

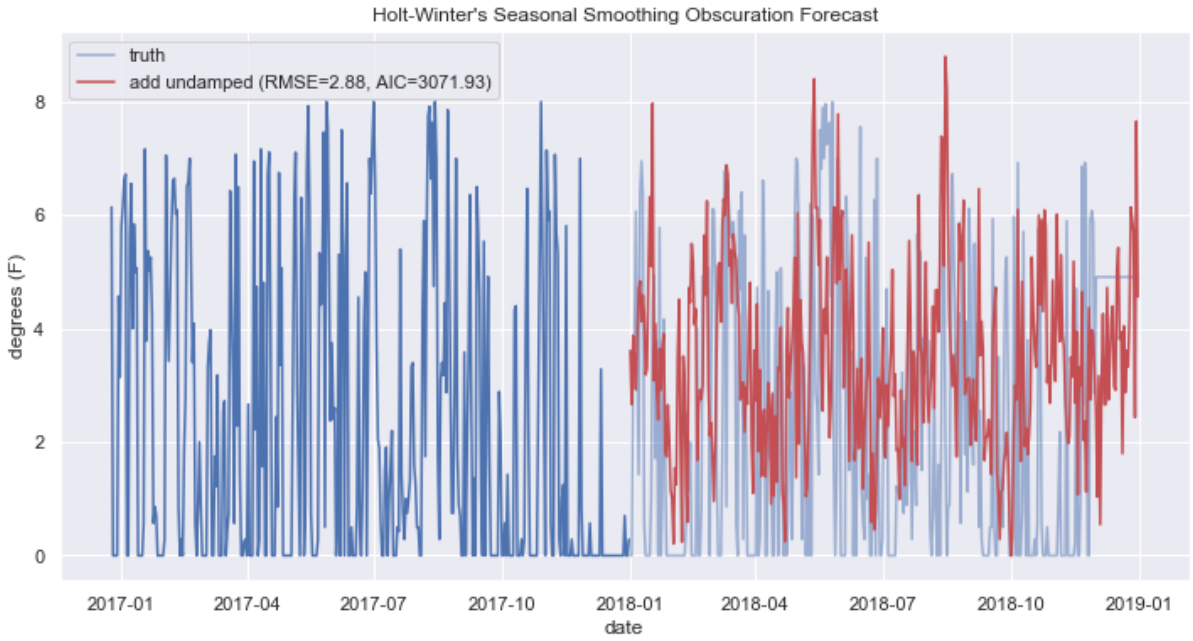
```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

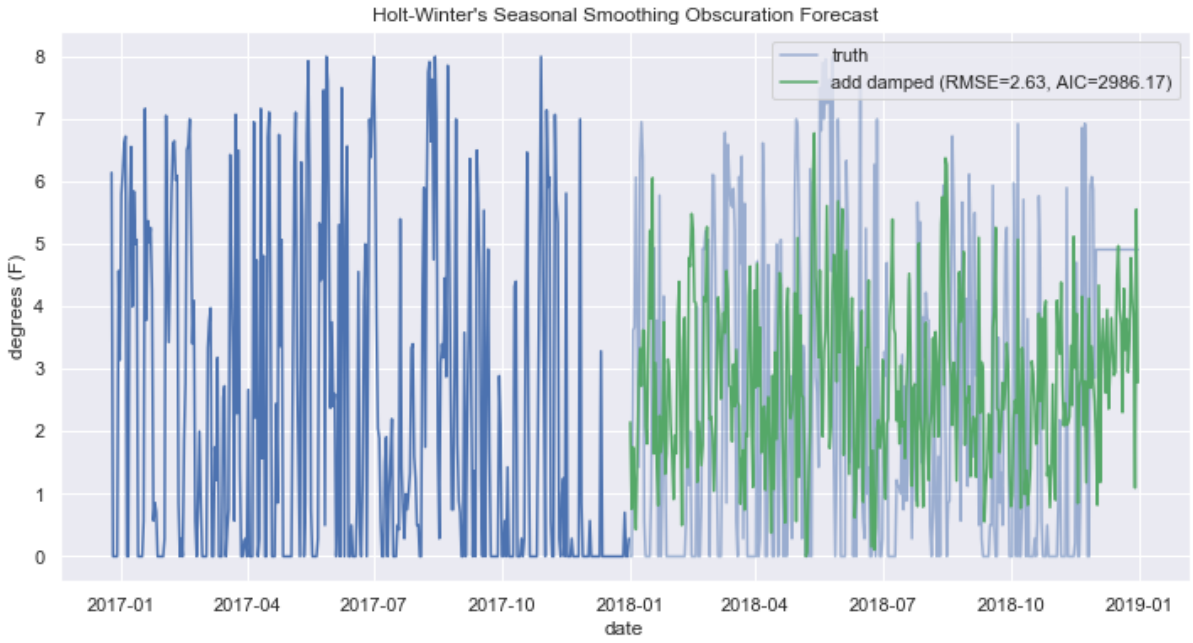
In [338]:
```
# fit models to data
fit1 = model1.fit()
cast1 = fit1.forecast(365)
cast1 = cast1 - cast1.min()

fit2 = model2.fit()
cast2 = fit2.forecast(365)
cast2 = cast2 - cast2.min()
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [339]:
```
# calculate error
sse1 = np.sqrt(np.mean(np.square(test.values - cast1.values)))
sse2 = np.sqrt(np.mean(np.square(test.values - cast2.values)))
```

In [340]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast1, color='r', label="add undamped (RMSE={:0.2f},
AIC={:0.2f})".format(sse1, fit1.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Obscuration Forecast");
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast2, color='g', label="add damped (RMSE={:0.2f}, A
IC={:0.2f})".format(sse2, fit2.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Obscuration Forecast");
plt.show()
```





For obscuration data, the performance difference between damped and undamped additive models is much more ambiguous: the undamped model yielded less error, while the damped model yielded a lower AIC score. (All forecast values have been shifted up by their minimum value to avoid negative values.)

And we fit the same model to the humidity data:

In [341]:
```
# separate data into train and test sets
train = hum[:-365]
test = hum.iloc[-365:]
```

In [342]:
```
# initialize models
model1 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365)
model2 = ExponentialSmoothing(train, trend='add', seasonal='add', season
al_periods=365, damped=True)
model3 = ExponentialSmoothing(train, trend='add', seasonal='mul', season
al_periods=365, damped=True)
```

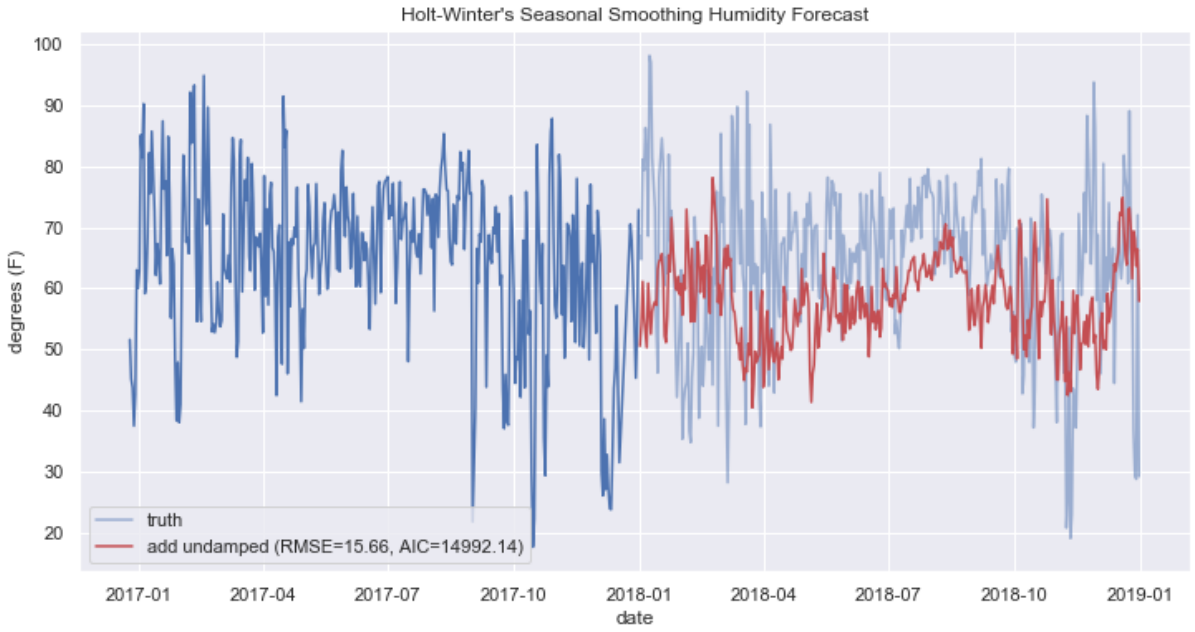
```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

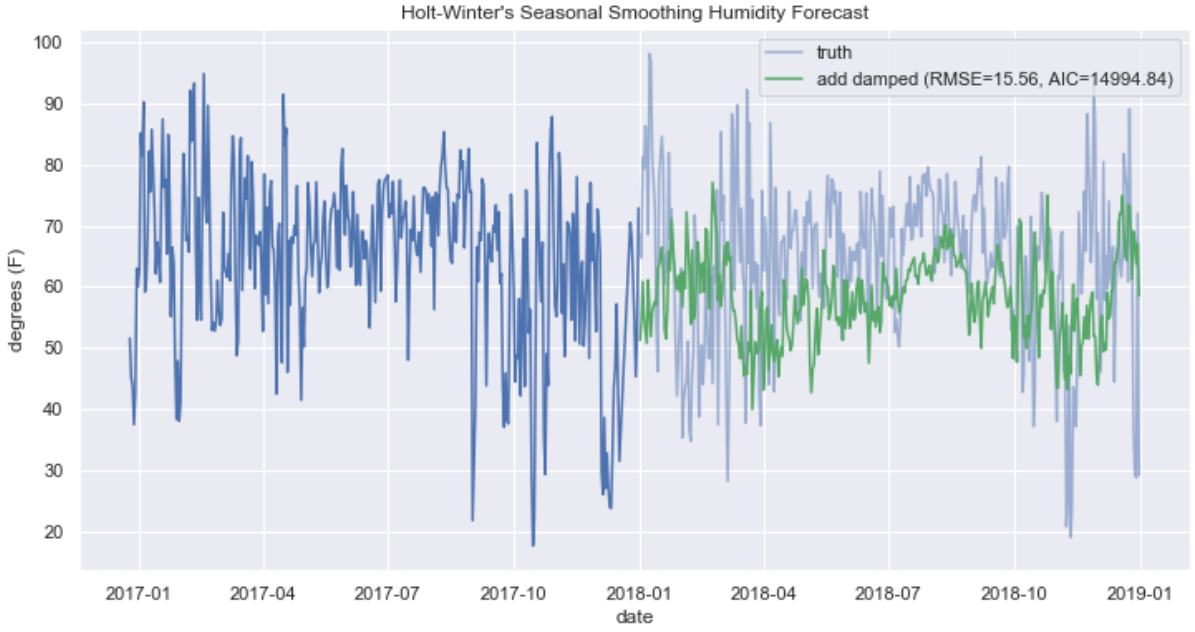
In [343]:
```
# fit models to data
fit1 = model1.fit()
cast1 = fit1.forecast(365)
fit2 = model2.fit()
cast2 = fit2.forecast(365)
fit3 = model3.fit()
cast3 = fit3.forecast(365)
```

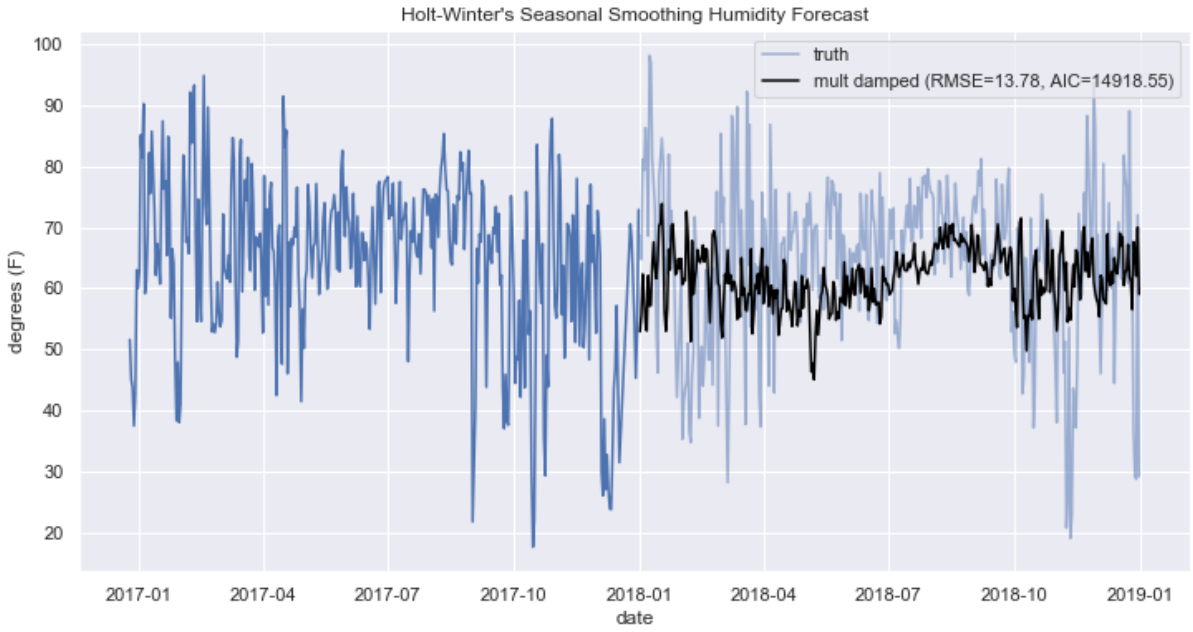
```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [344]:
```
# calculate error
sse1 = np.sqrt(np.mean(np.square(test.values - cast1.values)))
sse2 = np.sqrt(np.mean(np.square(test.values - cast2.values)))
sse3 = np.sqrt(np.mean(np.square(test.values - cast3.values)))
```

In [345]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast1, color='r', label="add undamped (RMSE={:0.2f},
AIC={:0.2f})".format(sse1, fit1.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Humidity Forecast");
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast2, color='g', label="add damped (RMSE={:0.2f}, A
IC={:0.2f})".format(sse2, fit2.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Humidity Forecast");
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index[-365:], train.values[-365:])
ax.plot(test.index, test.values, label='truth',color='b', alpha=.5);
ax.plot(test.index, cast3, color='black', label="mult damped (RMSE={:0.2
f}, AIC={:0.2f})".format(sse3, fit3.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Humidity Forecast");
plt.show()
```


Holt-Winter's Seasonal Smoothing Humidity Forecast
degrees (F)
date
truth
add undamped (RMSE=15.66, AIC=14992.14)
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01


Holt-Winter's Seasonal Smoothing Humidity Forecast
degrees (F)
date
truth
add damped (RMSE=15.56, AIC=14994.84)
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01



In the case of the humidity data, the damped trend with additive trend and seasonality performs best by both error and AIC measures.

6b. 2019 Model Predictions

In [346]:
```
# calculate index for use in all predictions
forecast_index = pd.date_range(start='01-01-2019', end='12-31-2019')
```

A damped, additive trend and multiplicative seaonality model predicts 2019 temperature data:

In [347]:
```
# set training data
train = temp
```

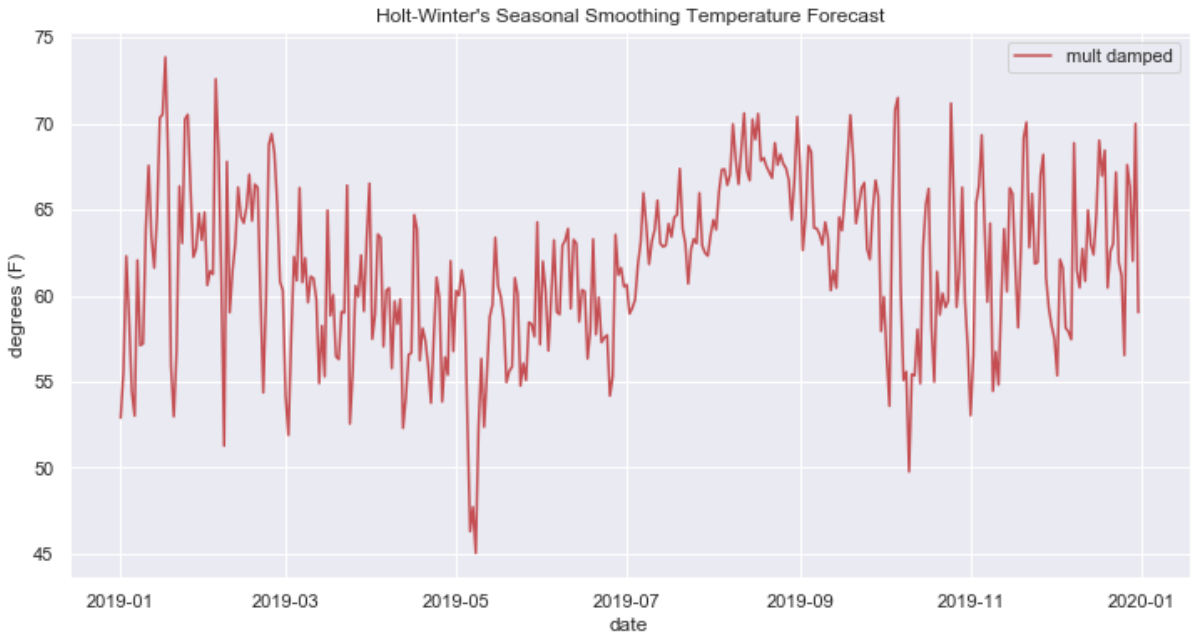
In [348]:
```
# initialize models
temp_model = ExponentialSmoothing(train, trend='add', seasonal='mul', se
asonal_periods=365, damped=True)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [349]:
```
# fit models to data
fit_temp = model3.fit()
temp_cast = fit_temp.forecast(365)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [350]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(forecast_index, temp_cast, color='r', label="mult damped".format
(sse1, fit1.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('degrees (F)')
ax.set_title("Holt-Winter's Seasonal Smoothing Temperature Forecast");
plt.show()
```



A damped, additive trend and additive seaonality model predicts 2019 obscuration data:

In [351]:
```
# set training data
train = obsc
```

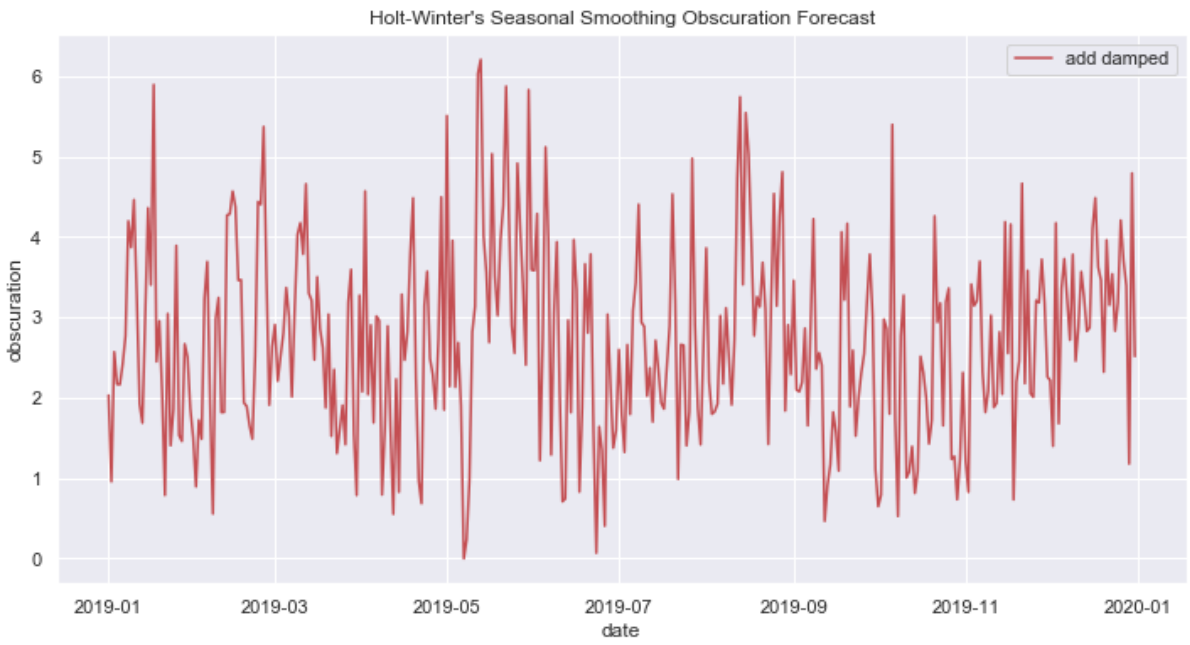
In [352]:
```
# initialize model
model = ExponentialSmoothing(train, trend='add', seasonal='add', seasona
l_periods=365, damped=True)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [353]:
```
# fit models to data
obsc_fit = model.fit()
obsc_cast = obsc_fit.forecast(365)
obsc_cast = obsc_cast - obsc_cast.min()
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [354]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(forecast_index, obsc_cast, color='r', label="add damped".format(
sse1, fit1.aic));
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('obscuration')
ax.set_title("Holt-Winter's Seasonal Smoothing Obscuration Forecast");
plt.show()
```



A damped, additive trend and additive seaonality model predicts 2019 humidity data:

In [355]:
```
# set the training data
train = hum
```

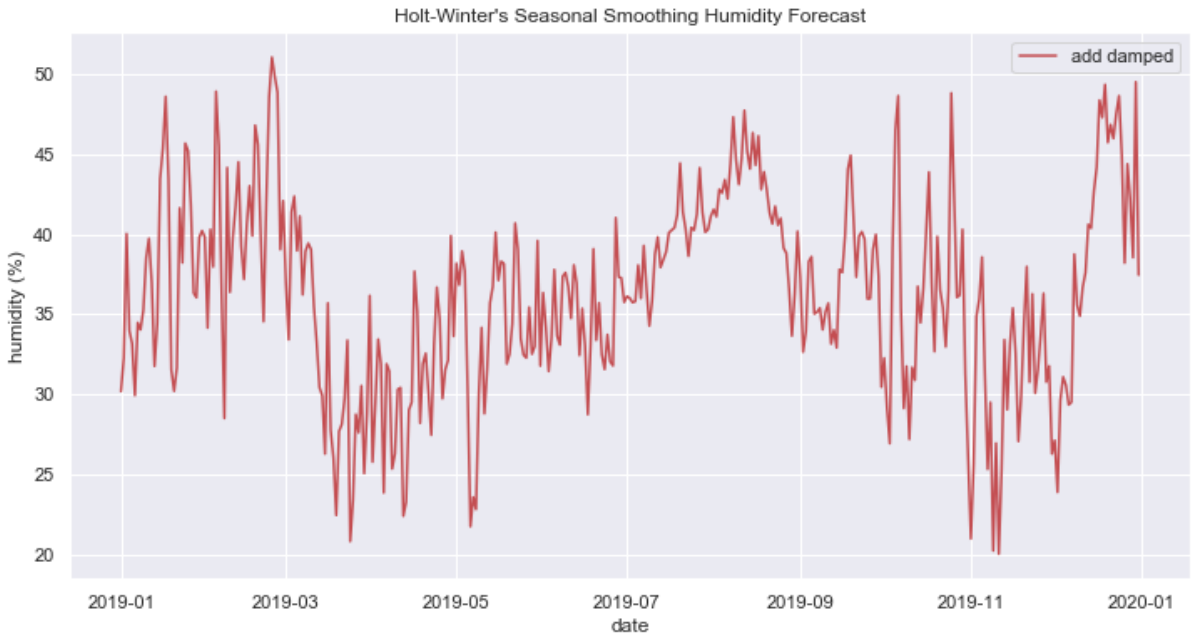
In [356]:
```
# initialize models
model = ExponentialSmoothing(train, trend='add', seasonal='add', seasona
l_periods=365, damped=True)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarnin
g: A date index has been provided, but it has no associated frequency i
nformation and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

In [357]:
```
# fit models to data
fit = model.fit()
hum_cast = fit.forecast(365)
```

```
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/holtwinters.py:711: ConvergenceWarn
ing: Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
/Users/trev7591a/.local/share/virtualenvs/Springboard-_vI4Xo1j/lib/pyth
on3.7/site-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarnin
g: No supported index is available. Prediction results will be given wi
th an integer index beginning at `start`.
  ValueWarning)
```

In [358]:
```
# plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(forecast_index, hum_cast, color='r', label="add damped");
ax.legend();
ax.set_xlabel('date')
ax.set_ylabel('humidity (%)')
ax.set_title("Holt-Winter's Seasonal Smoothing Humidity Forecast");
plt.show()
```



7. Facebook Prophet

Facebook's open-source Prophet forecasting tool is a generalized additive model that captures two hierachically seasonal components through Fourier modeling and a trend component using Bayesian inference on smoothing priors. Details [can be found here (https://peerj.com/preprints/3190/)](https://peerj.com/preprints/3190/).

In [139]:
```
# make dataframes fbprophet likes
def make_prophet_dataframe_from_series(series):
    frame = pd.DataFrame(series).reset_index()
    frame.columns = ['ds', 'y']
    return frame
```

Temperature

In [140]:
```
# make prophet frames
fb_temp = make_prophet_dataframe_from_series(temp)
fb_temp.tail()
```

Out[140]:

	ds	y
3275	2018-12-27	59.0
3276	2018-12-28	59.0
3277	2018-12-29	59.0
3278	2018-12-30	59.0
3279	2018-12-31	59.0

In [141]:
```
# fit the model
m = Prophet()
m.fit(fb_temp)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seas
onality=True to override this.
```

Out[141]:
```
<fbprophet.forecaster.Prophet at 0x11d225ef0>
```

In [142]:
```
# make future column
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[142]:

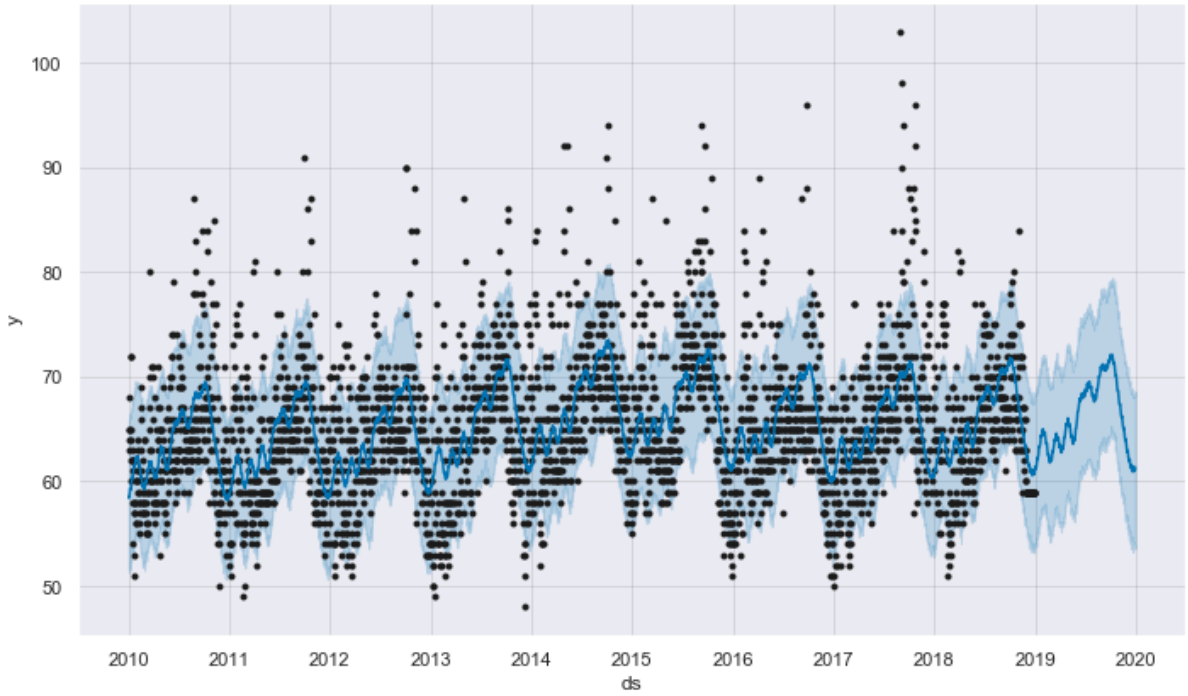
	ds
3640	2019-12-27
3641	2019-12-28
3642	2019-12-29
3643	2019-12-30
3644	2019-12-31

In [143]:
```
# predict
temp_forecast = m.predict(future)
temp_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

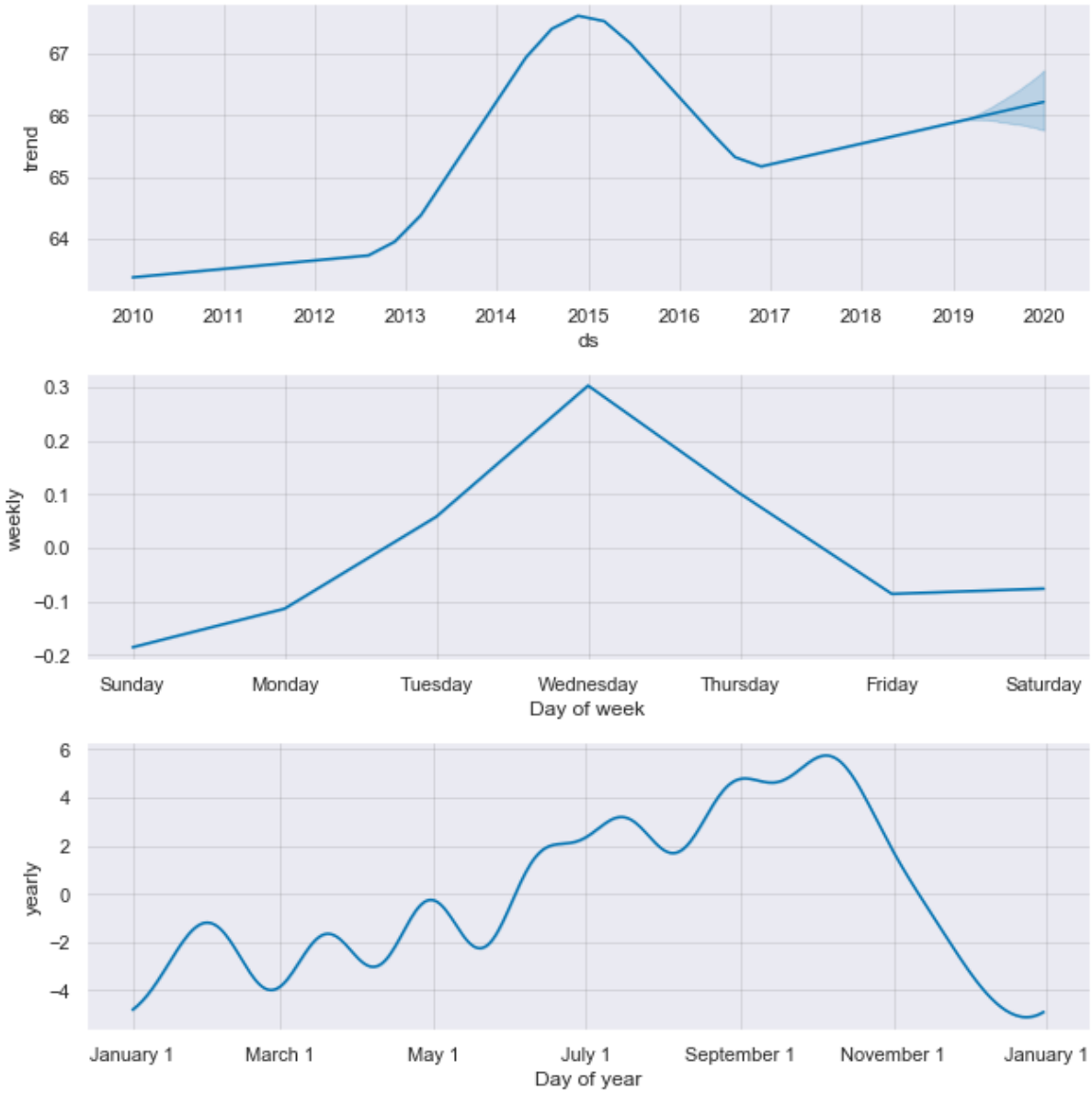
Out[143]:

	ds	yhat	yhat_lower	yhat_upper
3640	2019-12-27	61.021124	53.784638	68.518011
3641	2019-12-28	61.058185	53.900629	68.626621
3642	2019-12-29	60.985157	53.982501	68.185354
3643	2019-12-30	61.103092	53.581407	68.235560
3644	2019-12-31	61.330493	54.125660	68.580220

In [144]:
```
# plot predictions
fig1 = m.plot(temp_forecast)
```



In [145]:
```
# plot model components
fig2 = m.plot_components(temp_forecast)
```



Obscuration

In [146]:
```
# make prophet frames
fb_obsc = make_prophet_dataframe_from_series(obsc)
fb_obsc.tail()
```

Out[146]:

	ds	y
1814	2018-12-27	4.91146
1815	2018-12-28	4.91146
1816	2018-12-29	4.91146
1817	2018-12-30	4.91146
1818	2018-12-31	4.91146

In [147]:
```
# fit the model
m = Prophet()
m.fit(fb_obsc)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seas
onality=True to override this.
```

Out[147]:
```
<fbprophet.forecaster.Prophet at 0x11a347f60>
```

In [148]:
```
# make future column
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[148]:

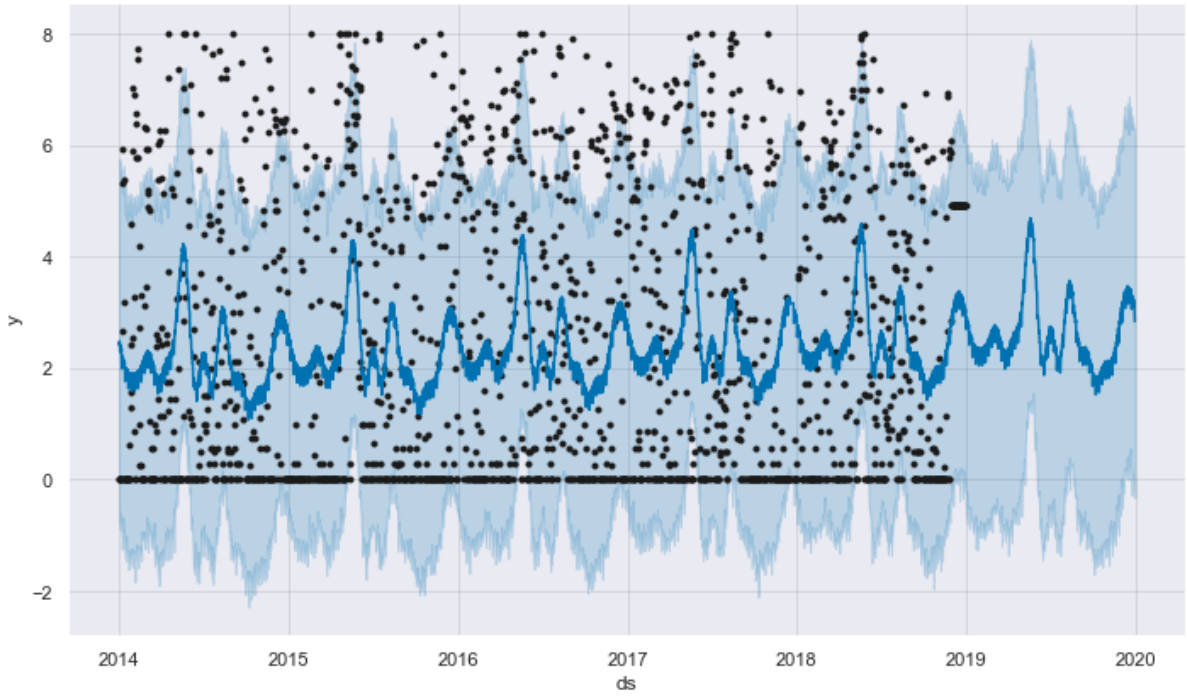
	ds
2179	2019-12-27
2180	2019-12-28
2181	2019-12-29
2182	2019-12-30
2183	2019-12-31

In [149]:
```
# predict
obsc_forecast = m.predict(future)
obsc_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

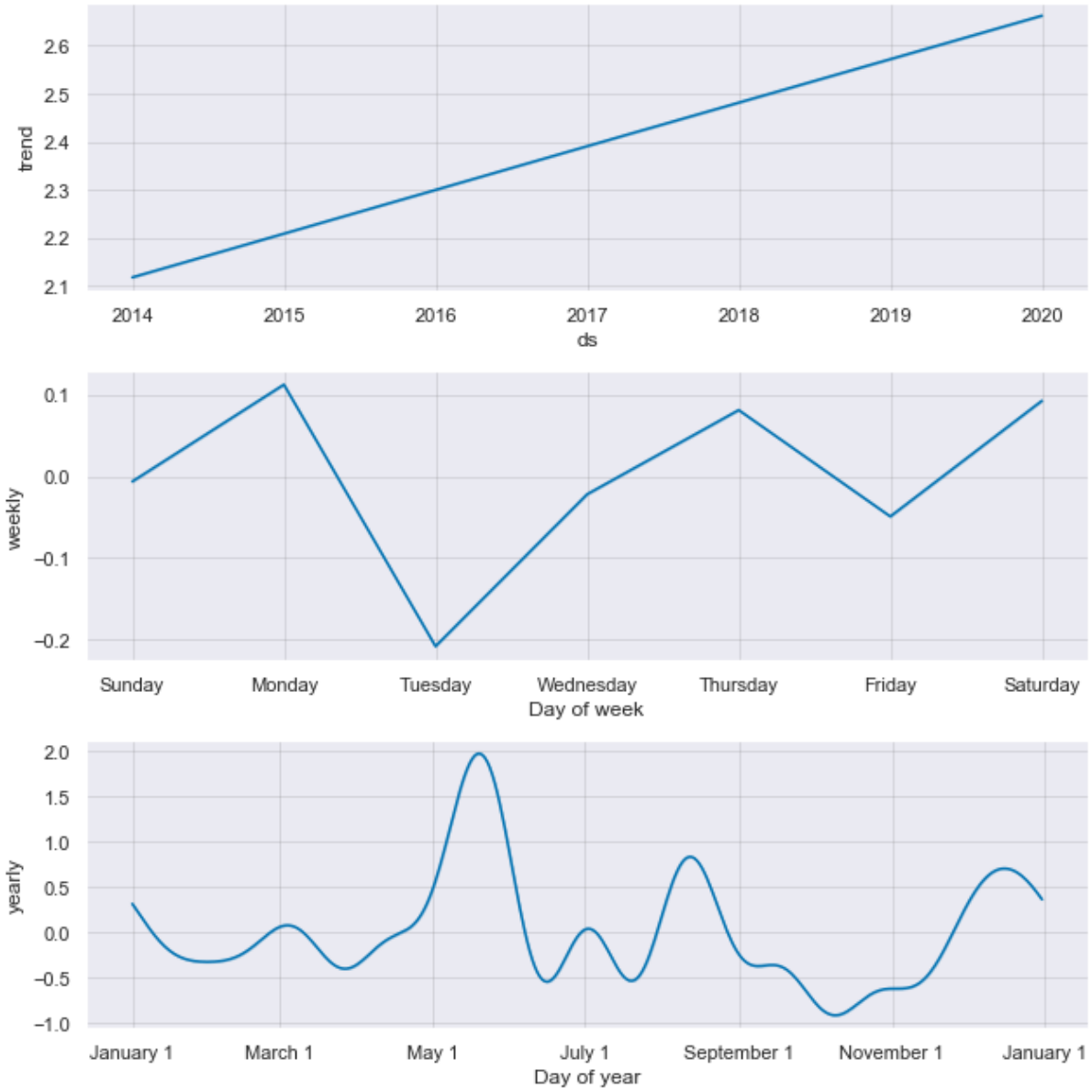
Out[149]:

	ds	yhat	yhat_lower	yhat_upper
2179	2019-12-27	3.136435	0.035086	6.396066
2180	2019-12-28	3.245902	0.055232	6.344792
2181	2019-12-29	3.112395	-0.303804	6.309925
2182	2019-12-30	3.195186	0.050476	6.302576
2183	2019-12-31	2.836476	-0.323144	6.090992

In [150]:
```
# plot predictions
fig1 = m.plot(obsc_forecast)
```



In [151]:
```
# plot model components
fig2 = m.plot_components(obsc_forecast)
```



Humidity

In [152]:
```
# make prophet frames
fb_hum = make_prophet_dataframe_from_series(hum)
fb_hum.tail()
```

Out[152]:

	ds	y
3275	2018-12-27	36.714286
3276	2018-12-28	29.285714
3277	2018-12-29	28.714286
3278	2018-12-30	72.000000
3279	2018-12-31	29.142857

In [153]:
```
# fit the model
m = Prophet()
m.fit(fb_hum)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seas
onality=True to override this.
```

Out[153]:
```
<fbprophet.forecaster.Prophet at 0x11ba04668>
```

In [154]:
```
# make future column
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[154]:

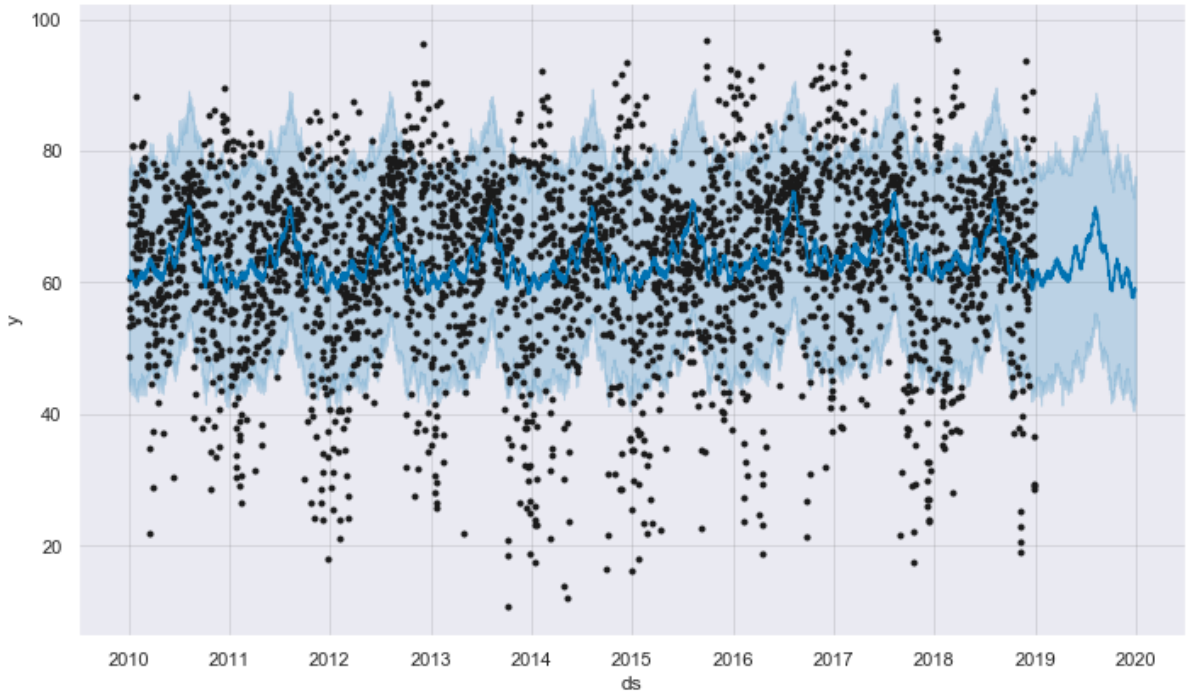
	ds
3640	2019-12-27
3641	2019-12-28
3642	2019-12-29
3643	2019-12-30
3644	2019-12-31

In [155]:
```
# predict
hum_forecast = m.predict(future)
hum_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

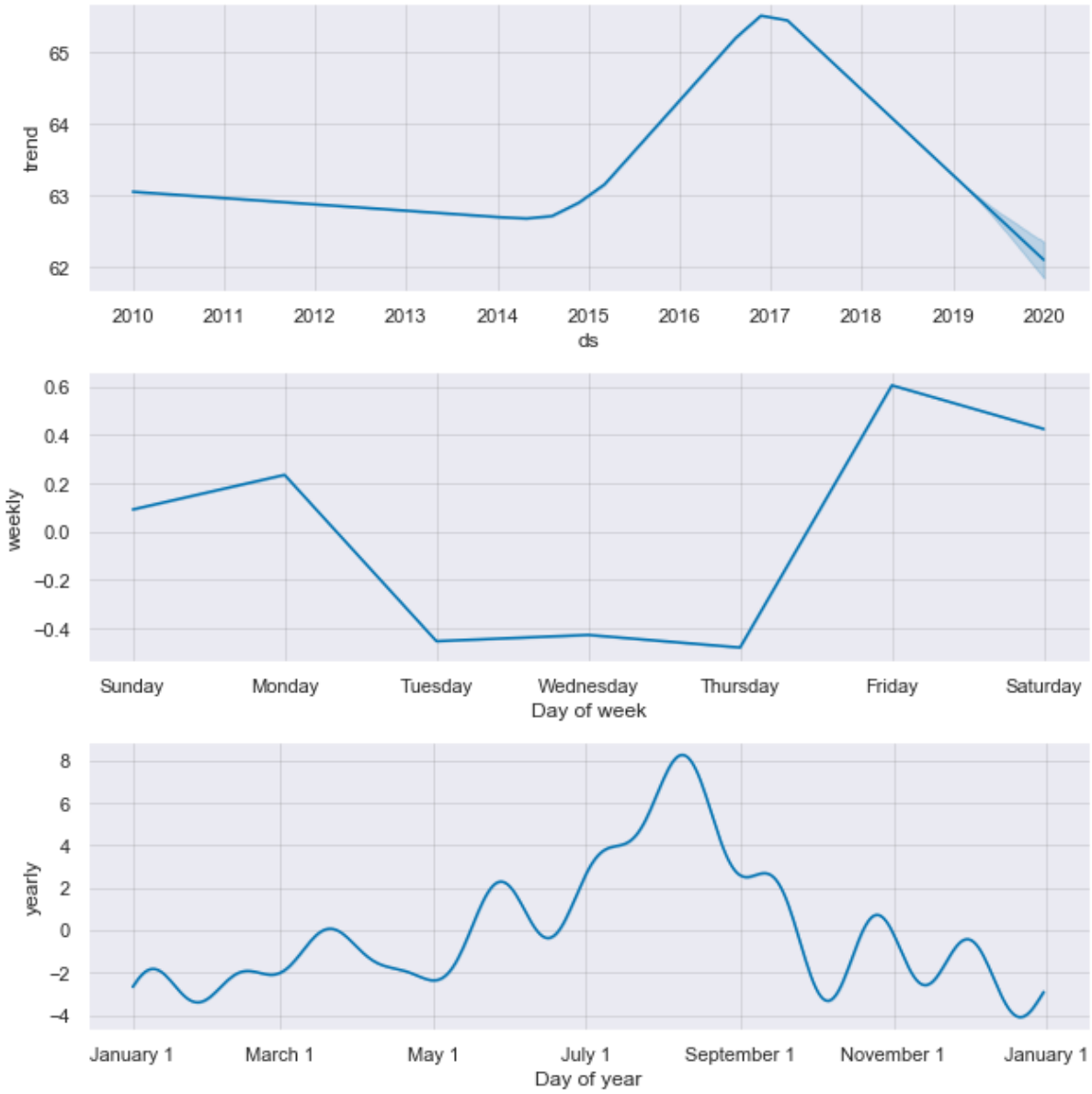
Out[155]:

	ds	yhat	yhat_lower	yhat_upper
3640	2019-12-27	59.002216	42.225497	75.746077
3641	2019-12-28	58.976680	42.608785	74.775363
3642	2019-12-29	58.815097	41.561250	76.076791
3643	2019-12-30	59.140106	42.475000	76.263902
3644	2019-12-31	58.640828	43.268612	74.678885

In [156]:
```
# plot predictions
fig1 = m.plot(hum_forecast)
```



In [157]:
```
# plot model components
fig2 = m.plot_components(hum_forecast)
```



How do the prophet predictions compare to the first three months of 2019?

In [158]:
```
# get 2019 values
temp_nineteen = temp_all[temp_all.index.year == 2019]
temp_nineteen
len(obsc_nineteen)
```

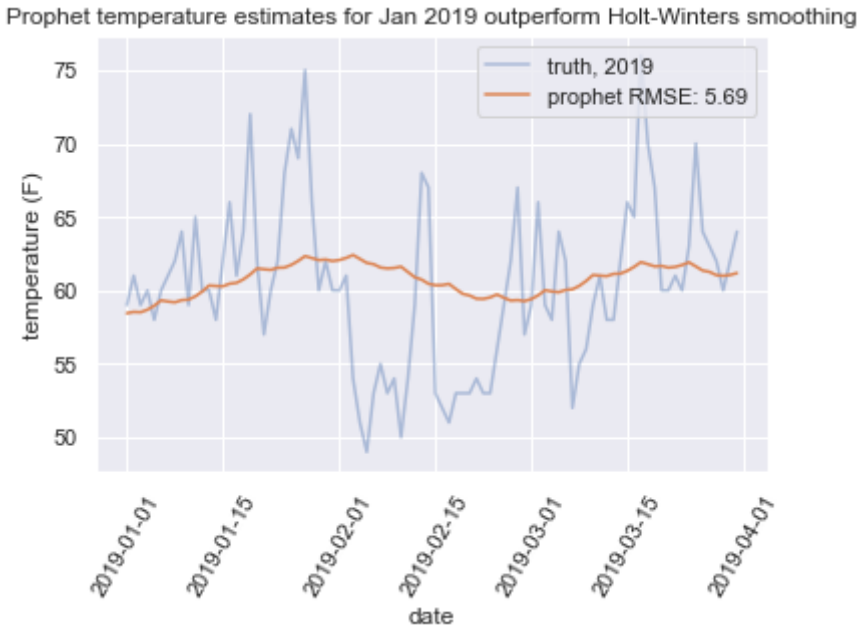
Out[158]: 90

In [159]:
```
# get estimates
guess_df = pd.DataFrame(temp_forecast['yhat'][:90])
i = pd.date_range(start='01/01/19', end='03/31/19')
guess_df = guess_df.set_index(i)
```

In [160]:
```
# calculate error
temp_rmse = np.sqrt(np.mean(np.square(guess_df.values - temp_nineteen.va
lues)))
temp_rmse
```

Out[160]:
```
5.693763973980777
```

In [161]:
```
# overlay predicted values with measured values
plt.plot(temp_nineteen, alpha=.4, label='truth, 2019')
plt.plot(guess_df, label='prophet RMSE: {:0.2f}'.format(temp_rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('temperature (F)')
plt.title('Prophet temperature estimates for Jan 2019 outperform Holt-Wi
nters smoothing')
plt.show()
```



In [171]:
```
# get 2019 values
obsc_nineteen = obsc_all[obsc_all.index.year == 2019]
obsc_nineteen += np.min(obsc_nineteen)
len(obsc_nineteen)
```

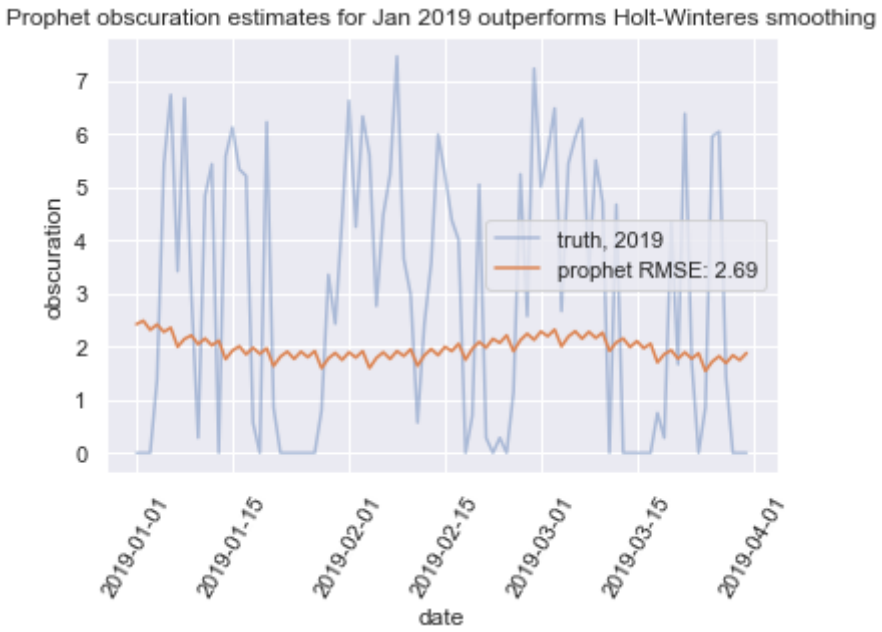
Out[171]:
```
90
```

In [172]:
```
# get estimates
guess_df = pd.DataFrame(obsc_forecast['yhat'][:90])
i = pd.date_range(start='01/01/19', end='03/31/19')
guess_df = guess_df.set_index(i)
```

In [173]:
```
# calculate error
obsc_rmse = np.sqrt(np.mean(np.square(guess_df.values - obsc_nineteen.va
lues)))
obsc_rmse
```

Out[173]: 2.692371232072787

In [174]:
```
# overlay predicted values with measured values
plt.plot(obsc_nineteen, alpha=.4, label='truth, 2019')
plt.plot(guess_df, label='prophet RMSE: {:0.2f}'.format(obsc_rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('obscuration')
plt.title('Prophet obscuration estimates for Jan 2019 outperforms Holt-W
interes smoothing')
plt.show()
```



In [166]:
```
# get 2019 values
hum_nineteen = hum_all[temp_all.index.year == 2019]
len(temp_nineteen)
```

Out[166]: 90

In [167]:
```
# get estimates
guess_df = pd.DataFrame(hum_forecast['yhat'][:90])
i = pd.date_range(start='01/01/19', end='03/31/19')
guess_df = guess_df.set_index(i)
```

In [168]:
```
# calculate error
rmse = np.sqrt(np.mean(np.square(guess_df.values - hum_nineteen.values
)))
rmse
```

Out[168]: 12.98746527763348

In [169]:
```
# overlay predicted values with measured values
plt.plot(hum_nineteen, alpha=.4, label='truth, 2019')
plt.plot(guess_df, label='prophet RMSE: {:0.2f}'.format(rmse))
plt.xticks(rotation=60)
plt.legend()
plt.xlabel('date')
plt.ylabel('humidity (%)')
plt.title('Prophet humidity estimates for Jan 2019 perform slightly unde
r Holt-Winters smoothing')
plt.show()
```