Load Analysis

Casa Grande Main Facility

Developed by: Jeff MacKinnon

Reviewed by: Sam Terry

- Project Number 2325
- November 28, 2023

Dependancies

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

Source Information

The first step is to combine the meter information provided by Lucid. This data was provided by Chris Kim of Lucid. The information was uploaded to Sharefile on November 22, 2023.

There were a number of .CSV files provided. The MeterInformation.csv is a single file with all the lines from the provided files.

```
In [ ]: # step through all the meter files and create a single CSV. This should only be don

        import os
        import glob
        meter_info = os.path.exists('MeterInformation.csv')
        if meter_info == False:
            import pandas as pd
            files = glob.glob('rawmeter/*.csv')
            df = pd.concat(
                [pd.read_csv(fp,) for fp in files], ignore_index=True)
            df.to_csv('MeterInformation.csv')
            print('MeterInformation.csv created.')
        else:
            print('The meter information file already exists. If it needs to be updated, re
```

```
The meter information file already exists. If it needs to be updated, rename or dele
te the existing file.
```

```
In [ ]: raw = pd.read_csv('MeterInformation.csv',
                          #encoding = 'cp1252', # This is the encoding that PVsyst spits
                          skip_blank_lines=True, # I don't want blank rows.
                          #header = header_row,
                          #skiprows = (1-9,11),
                          parse_dates=['Timestamp UTC','Timestamp'],
                          #index_col='date',
      )
```

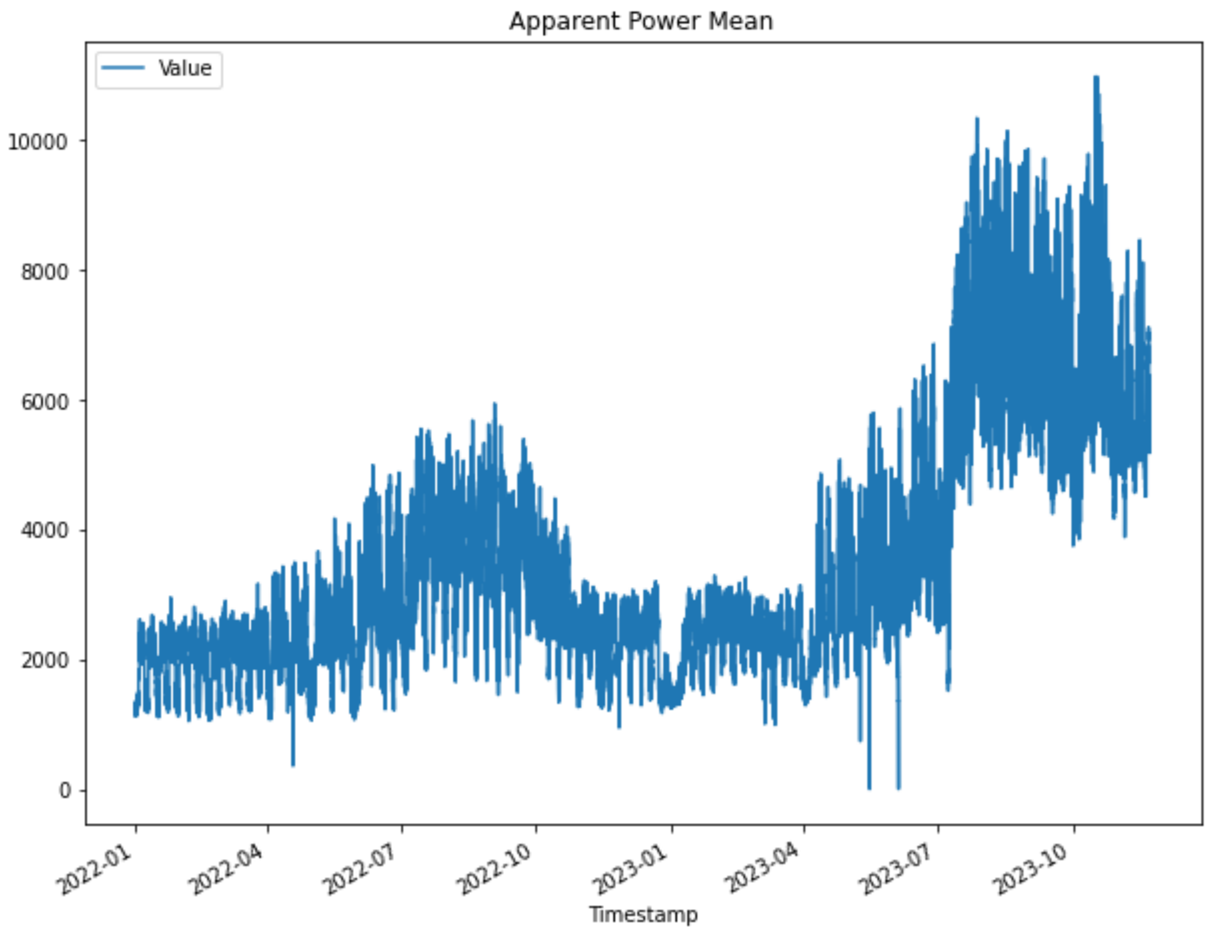
The critical information that we are looking at are in the "Measurement" column.

In [ ]:
```
measurements = raw.drop_duplicates(subset=['Measurement'])
#print(measurements.Measurement)
```

Initial check to see that the data shows

To ensure that there are no gaps in the information we select a measurment, in this case Apparent Power Mean, and plot.

In [ ]:
```
grouped = raw.groupby(raw['Measurement'])
Apparent_Power_raw = grouped.get_group('Apparent Power Mean')

Apparent_Power_raw.plot(
        x="Timestamp",
        y="Value",
        title='Apparent Power Mean',
        #xlabel= 'day',
        #ylabel='kW to the grid',
        kind="line",
        figsize=(10,8),
        )
plt.savefig('report/images/provided_values_apparent_power.png', format='png')
plt.show()
```



Initial data cleaning

Start cleaning the raw data by removing the columns we won't be using and then clearing any duplicates.

```
In [ ]: df = raw.drop(['Timestamp UTC','Unit'], axis=1) # I want to remove a timestamp to h
        df = df.drop_duplicates(subset=['Timestamp','Value','Source','Measurement'], keep='
```

Creating a Pivot Table

Pivot the dataframe so that everything has columns and then make the timestamp the index.

Next, create a zipfile of the data that will be used to as part of this analysis

```
In [ ]: df = df.pivot(index='Timestamp', columns=['Source','Measurement']) # Pivot so that

        compression_opts = dict(method='zip',
                                archive_name='2325-pivot_data.csv')
        df.to_csv('2325-pivot_data.zip', index=True,
                  compression=compression_opts)
        df
```

Out[ ]:

Source							
Measurement	Voltage A-N	Voltage B-N	Voltage C-N	Voltage A-B	Voltage B-C	Voltage C-A	C1 Current A
Timestamp							
2022-01-01 00:15:00	0.0	1.0	2.0	3.0	4.0	5.0	6.0
2022-01-01 00:30:00	38.0	39.0	40.0	41.0	42.0	43.0	44.0
2022-01-01 00:45:00	76.0	77.0	78.0	79.0	80.0	81.0	82.0
2022-01-01 01:00:00	114.0	115.0	116.0	117.0	118.0	119.0	120.0
2022-01-01 01:15:00	152.0	153.0	154.0	155.0	156.0	157.0	158.0
...	...	...	...	...	...	...	...
2023-11-22 11:15:00	9418034.0	9418035.0	9418036.0	9418037.0	9418038.0	9418039.0	9418040.0
2023-11-22 11:30:00	9418072.0	9418073.0	9418074.0	9418075.0	9418076.0	9418077.0	9418078.0
2023-11-22 11:45:00	9418110.0	9418111.0	9418112.0	9418113.0	9418114.0	9418115.0	9418116.0
2023-11-22 12:00:00	9418148.0	9418149.0	9418150.0	9418151.0	9418152.0	9418153.0	9418154.0
2023-11-22 12:15:00	9418186.0	9418187.0	9418188.0	9418189.0	9418190.0	9418191.0	9418192.0

66764 rows × 562 columns

Removing the Data that we won't be using

From here we will continue to condition the dataframe to make it easier to work with.

Clean the dataframe to include the values we want, and rename the "high" values from the main meter to be "instantaneous" to match the shark meter values

In [ ]:
```
df_conditioned = df.drop(columns=[
                             #'Apparent Power Low',
                            'Apparent Power Mean',
                            #'Block Demand Apparent Power Total',
 'Reactive Power Low',
 'Reactive Power Mean',
 #'Block Demand Reactive Power',
 #'Block Demand Real Power',
 'Real Power Low',
 'Power Factor Lagging High',
 'Power Factor Lagging Low',
 'Power Factor Lagging Mean',
 'Power Factor Leading High',
 'Power Factor Leading Low',
 'Power Factor Leading Mean',
 'Voltage A-B High',
 'Voltage A-B Mean',
 'Voltage A-B Low', # Will use the "mean" low as the base volt
 'Voltage B-C High',
 'Voltage B-C Low',
 'Voltage B-C Mean',
 'Voltage C-A High',
 'Voltage C-A Low',
 'Voltage C-A Mean',
 'Voltage L-L Avg High',
'Voltage L-L Avg Mean',
# 'Voltage L-L Avg Low',
 'Voltage A-N', # The shark meters included both L-N and L-L v
 'Voltage B-N',
 'Voltage C-N',
 'Real Power A',
 'Real Power B',
 'Real Power C',
 'Voltage B-C',
 'Voltage C-A',
 'Current A',
 'Current B',
 'Current C',
 'Current N',
 'Current A High',
 'Current A Low',
 'Current A Mean',
 'Current B High',
 'Current B Low',
 'Current B Mean',
 'Current C High',
 'Current C Low',
 'Current C Mean',
 'C1 Current A',
 'C1 Current B',
 'C1 Current C',
 'C2 Current A',
 'C2 Current B',
 'C2 Current C',
 'C3 Curent A', # Spelled wrong because that's how its spelled
 'C3 Current B',
 'C3 Current C',
 'C4 Curent A', # Spelled wrong because that's how its spelled
 'C4 Current B',
 'C4 Current C',
                      'C5 Curent A', # Spelled wrong because that's how its spelled
                      'C5 Current B',
                      'C5 Current C',
                      'C6 Curent A', # Spelled wrong because that's how its spelled
                      'C6 Current B',
                      'C6 Current C',
                      'C7 Curent A',
                      'C7 Current B',
                      'C7 Current C',
                      'C8 Curent A',
                      'C8 Current B',
                      'C8 Current C',
                      ],
                      level=2)


df_conditioned = df_conditioned.rename({'Apparent Power High':'Apparent Power',
                'Reactive Power High':'Reactive Power',
                'Real Power High':'Real Power',
                'Voltage L-L Avg Low':'Voltage',       # Making the Voltage the Sam
                'Voltage A-B':'Voltage',                #
                },
                level=2,
                axis=1)

df_conditioned = df_conditioned['Value'] # Strips the value row making the datafram
df_conditioned = df_conditioned.loc['2022-07-01 00:00:00': '2023-11-21 23:45:00']

#
#   Create a .zip file of the conditioned table.
compression_opts = dict(method='zip',
                        archive_name='2325-conditioned_data.csv')
df_conditioned.to_csv('2325-conditioned_data.zip', index=True,
          compression=compression_opts)

df_conditioned
```

Out[ ]:

Source							HANN
Measurement	Voltage	C1 kW 3 Phase Total	C2 kW 3 Phase Total	C3 kW 3 Phase Total	C4 kW 3 Phase Total	C5 kW 3 Phase Total	C6 kW Pha To
Timestamp							
2022-07-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	N
2022-07-01 00:15:00	NaN	NaN	NaN	NaN	NaN	NaN	N
2022-07-01 00:30:00	NaN	NaN	NaN	NaN	NaN	NaN	N
2022-07-01 00:45:00	NaN	NaN	NaN	NaN	NaN	NaN	N
2022-07-01 01:00:00	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	
2023-11-21 22:45:00	484.199005	90.561203	143.102703	34.741922	316.257719	24.934873	31.4525
2023-11-21 23:00:00	486.479492	39.883285	64.000871	27.148957	210.999234	25.022152	27.4638
2023-11-21 23:15:00	483.316833	83.499773	53.611617	20.290480	76.538195	11.174855	29.2439
2023-11-21 23:30:00	483.531036	84.635617	136.490484	29.779871	71.920156	11.120963	23.7882
2023-11-21 23:45:00	482.226196	42.134477	128.760461	74.590539	225.815656	11.130894	23.8565

49211 rows × 78 columns

In [ ]:
```
df_HANNA = df_conditioned['HANNA_ROAD.SHARK_PM200_01']
df_MainA = df_conditioned['LUCID01.PRIMARY_SWGR_MAIN']
df_1302 = df_conditioned['NORTH_PAINT.1302_SPM']
df_1201 = df_conditioned['BIW.1201_SPM']
df_1304 = df_conditioned['WEST_PAINT.1304_SPM']
df_13SP = df_conditioned['NORTH_PAINT.13SP_SPM']
df_1202 = df_conditioned['BIW.1202_SPM']
df_1305 = df_conditioned['WEST_PAINT.1305_SPM']
df_1902 = df_conditioned['CUP.1902_SPM']
df_1903 = df_conditioned['CUP.1903_SPM']
df_1901 = df_conditioned['CUP.1901_SPM']
df_19SP = df_conditioned['CUP.19SP_SPM']
df_1301 = df_conditioned['NORTH_PAINT.1301_SPM']
df_1303 = df_conditioned['NORTH_PAINT.1303_SPM']
```

```
#df_MainA
```

Q4'23/Q1'24 Load Analysis

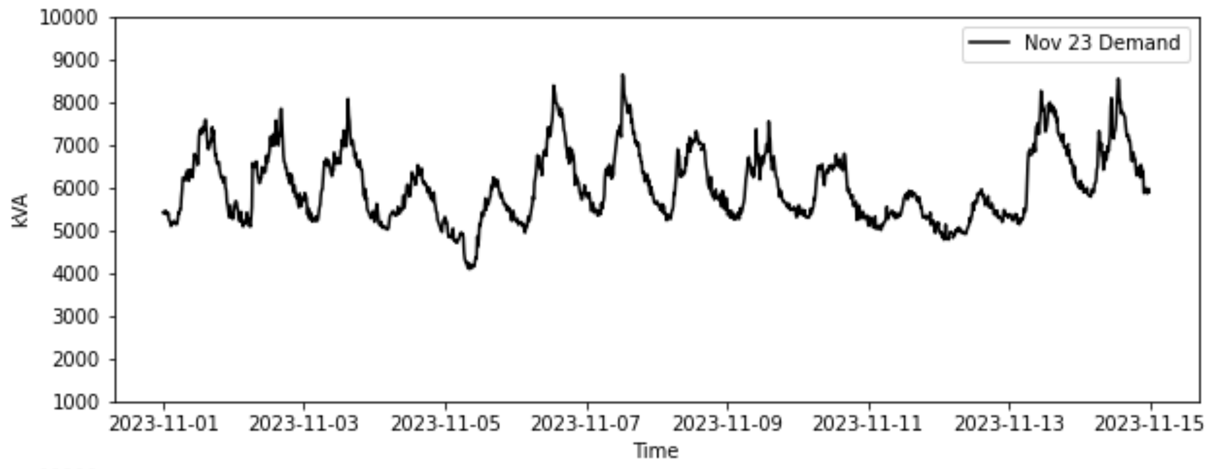
The factors that we are accounting for with regards to the load growth from January 2023 to January 2024 are:

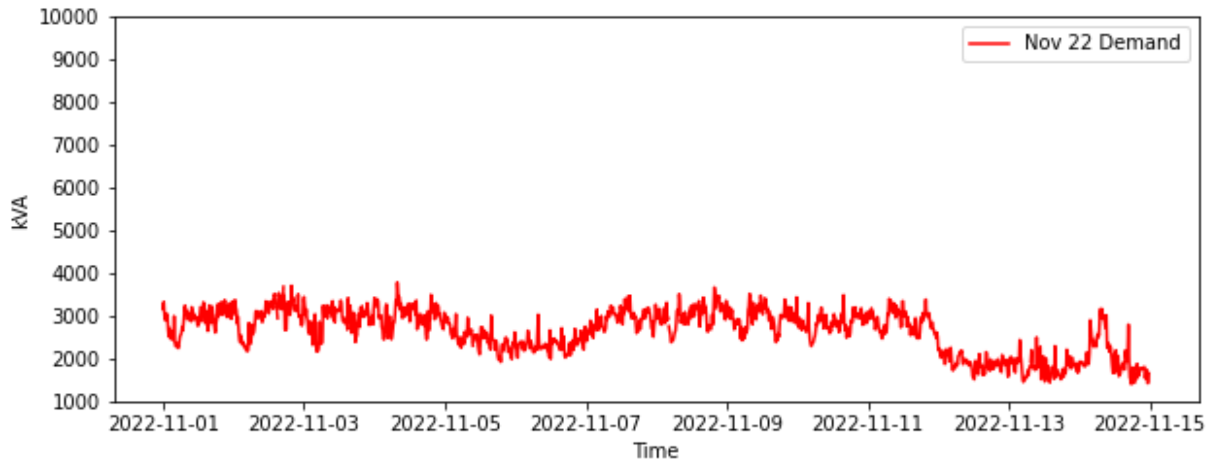
- The YoY growth from the first two weeks in November 2022 and November 2023.
- The expected additional load for Stamping
- Additional HVAC Demand for North Building

For this part of the analysis we will only look at the main meter.

In [ ]:
```
# This code is specifically to ensure that the graphs all have the same axis and Lo

plot_y_min = 1000
plot_y_max_base = 10000
plot_y_max_fans = 14000
plot_y_abs_max = 20000
```

In [ ]:
```
df_nov22 = df_conditioned.loc['2022-11-01 00:00:00': '2022-11-14 23:45:00']
df_nov23 = df_conditioned.loc['2023-11-01 00:00:00': '2023-11-14 23:45:00']

df_MainA_nov22 = df_nov22['LUCID01.PRIMARY_SWGR_MAIN']
df_MainA_nov23 = df_nov23['LUCID01.PRIMARY_SWGR_MAIN']

# Make the comparison plot
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,8))

ax1.plot(df_nov23.index, df_MainA_nov23['Apparent Power'], color='black', label='No

ax1.set_xlabel('Time')
ax1.set_ylabel('kVA')
ax1.set_ylim(plot_y_min,plot_y_max_base)
ax1.legend()

ax2.plot(df_nov22.index, df_MainA_nov22['Apparent Power'], color='red', label='Nov
ax2.set_xlabel('Time')
ax2.set_ylabel('kVA')
ax2.set_ylim(plot_y_min,plot_y_max_base)
ax2.legend()

plt.savefig('report/images/yoy-load_growth.png', format='png')
```





At this point we will assume that the day of the week average increase can be applied to the values from December '22 and January '23.

From there we will add the expected new loads for December, January and February to have a prediction of the new expected peaks.

Temperature and its affect on demand

Temperature has an extreme impact on the demand profile at the plant. There is a very large HVAC load to account for the cooling throughout the facility.

To determine the base load without the HVAC, without having the building automation system, or an operational SCADA system that monitors this information is difficult.

To get a sense of the magnitude of this load we will compare the temperature and demand during the first two weeks in September 2023 and the first two weeks in November.

The exact method is to determine the difference in demand and compare that to the difference in temperature. We will then determine if there is a way to add this information to the forecasted demand model.

```
In [ ]: meter_info = os.path.exists('weatherinformation.csv')
```

```
if meter_info == False:
    import pandas as pd
    files = glob.glob('rawweather/*.csv')
    df = pd.concat(
        [pd.read_csv(fp,) for fp in files], ignore_index=True)
    df.to_csv('weatherinformation.csv')
    print('weatherinformation.csv created.')
else:
    print('The weather information file already exists. If it needs to be updated,

weather = pd.read_csv('weatherinformation.csv',
                    #encoding = 'cp1252', # This is the encoding that PVsyst spits
                    skip_blank_lines=True, # I don't want blank rows.
                    #header = header_row,
                    #skiprows = (1-9,11),
                    parse_dates=['datetime'],
                    index_col='datetime',
)

weather = weather.loc[:, ~weather.columns.str.contains('^Unnamed')] # drops any Unn


#weather
```

```
The weather information file already exists. If it needs to be updated, rename or de
lete the existing file.
```

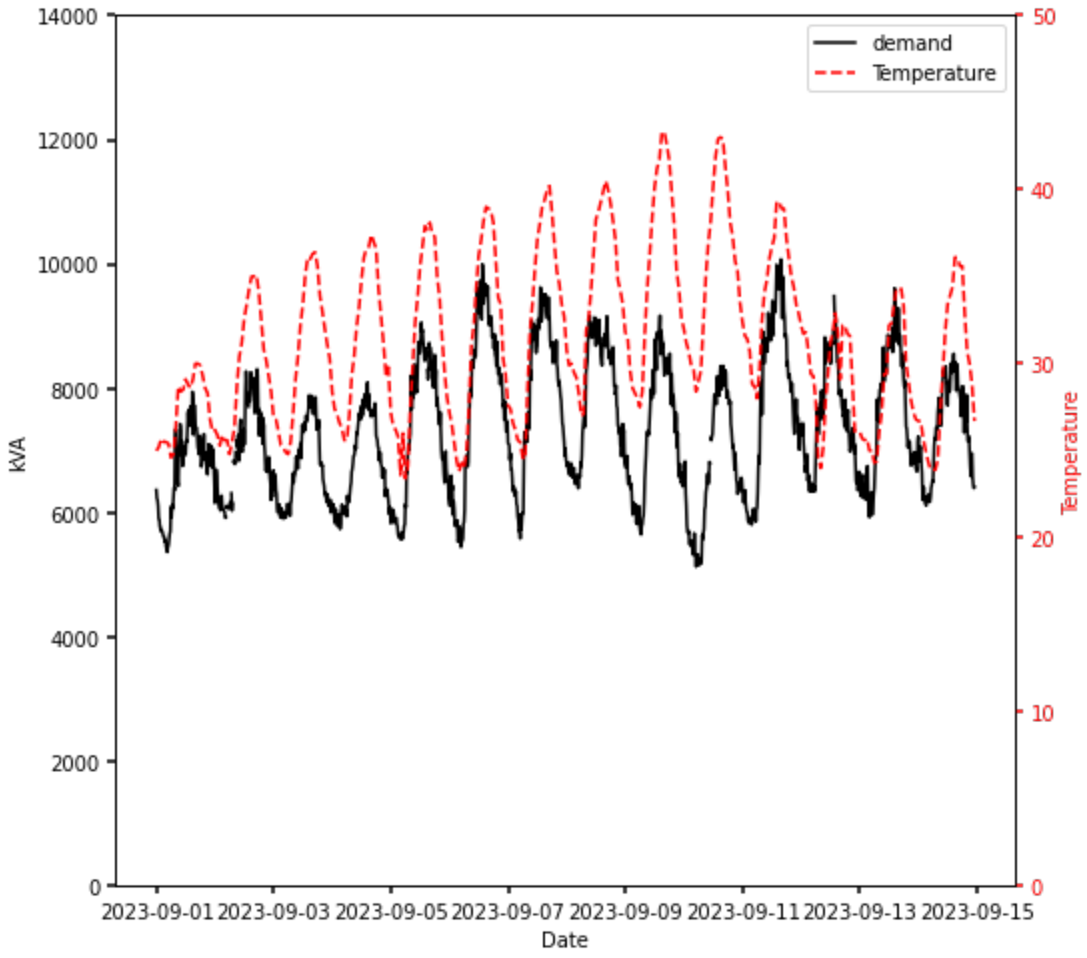
In [ ]:

```
# Set up the warm analysis frame

warm_analysis_start = '2023-09-01 00:00:00'
warm_analysis_end = '2023-09-14 23:00:00'

warm_analysis_power = df_MainA.loc[warm_analysis_start : warm_analysis_end]
warm_analysis_weather = weather.loc[warm_analysis_start : warm_analysis_end]
```

In [ ]:

```
fig, ax = plt.subplots(figsize=(10,8))
fig.subplots_adjust(right=0.75)

twin1 = ax.twinx()
#twin2 = ax.twinx()

# Offset the right spine of twin2.  The ticks and label have already been
# placed on the right by twinx above.
#twin2.spines.right.set_position(("axes", 1.2))

p1, = ax.plot(warm_analysis_power.index, warm_analysis_power['Apparent Power'], col
p2, = twin1.plot(warm_analysis_weather.index, warm_analysis_weather['temp'], color=

#ax.set_xlim(0, 2)
ax.set_ylim(0, 14000)
twin1.set_ylim(0, 50)

ax.set_xlabel("Date")
ax.set_ylabel("kVA")
twin1.set_ylabel("Temperature")
```

```
ax.yaxis.label.set_color(p1.get_color())
twin1.yaxis.label.set_color(p2.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
twin1.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax.tick_params(axis='x', **tkw)

ax.legend(handles=[p1, p2])

plt.savefig('report/images/power_demand_vs_temperature-Sept.png', format='png')

plt.show()
```

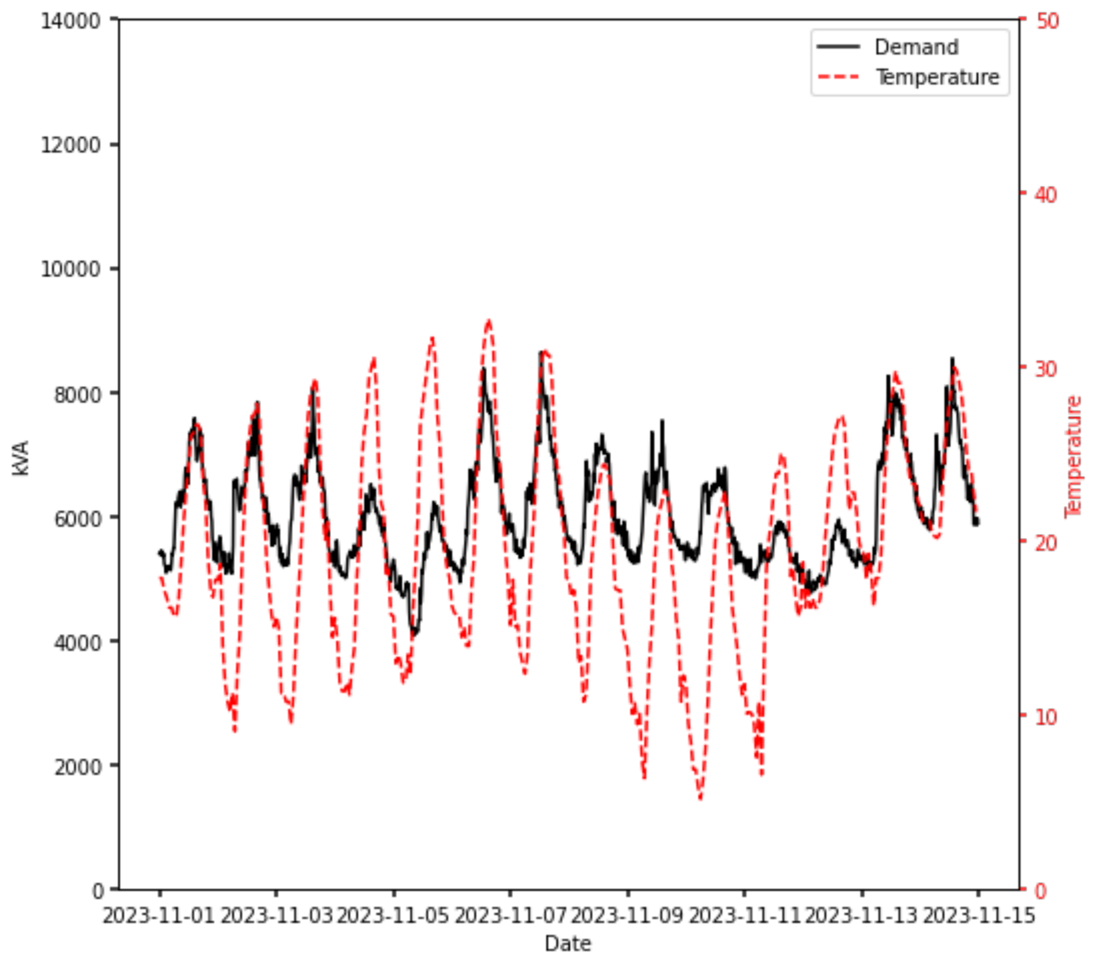


This shows a strong correlation between temperature and power. Lets do the same during a cooler period where a similar amount of equipment is connected.

```
In [ ]: # setup the cool analysis frame

cool_analysis_start = '2023-11-01 00:00:00'
cool_analysis_end = '2023-11-14 23:45:00'

cool_analysis_power = df_MainA.loc[cool_analysis_start : cool_analysis_end]
cool_analysis_weather = weather.loc[cool_analysis_start : cool_analysis_end]
```

```
In [ ]: fig, ax = plt.subplots(figsize=(10,8))
```

```
fig.subplots_adjust(right=0.75)

twin1 = ax.twinx()
#twin2 = ax.twinx()

# Offset the right spine of twin2.  The ticks and label have already been
# placed on the right by twinx above.
#twin2.spines.right.set_position(("axes", 1.2))

p1, = ax.plot(cool_analysis_power.index, cool_analysis_power['Apparent Power'], col
p2, = twin1.plot(cool_analysis_weather.index, cool_analysis_weather['temp'], color=

#ax.set_xlim(0, 2)
ax.set_ylim(0, 14000)
twin1.set_ylim(0, 50)

ax.set_xlabel("Date")
ax.set_ylabel("kVA")
twin1.set_ylabel("Temperature")

ax.yaxis.label.set_color(p1.get_color())
twin1.yaxis.label.set_color(p2.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
twin1.tick_params(axis='y', colors=p2.get_color(), **tkw)
ax.tick_params(axis='x', **tkw)

ax.legend(handles=[p1, p2])
plt.savefig('report/images/power_demand_vs_temperature-Nov.png', format='png')


plt.show()
```



There is still a correlation. On cooler days it appears that the demand peak is earlier in the day compared to the peak temperature, and then it drops with the temperature to a minimum base load between 4 and 5 MVA.

From these graphs we can determine the base load, calculated below to be the average of the bottom 10% of readings.

```
In [ ]: bottom_10per = len(warm_analysis_power.index)*0.1
        warm_base_n = warm_analysis_power.nsmallest(n=int(bottom_10per),columns=['Apparent
        warm_base = warm_base_n['Apparent Power'].mean()

        bottom_10per = len(cool_analysis_power.index)*0.1
        cool_base_n = cool_analysis_power.nsmallest(n=int(bottom_10per),columns=['Apparent
        cool_base = cool_base_n['Apparent Power'].mean()

        print('The base load during the warm period is ' + str(round(warm_base,2)) + 'kVA.'
        print('The base load during the cool period is ' + str(round(cool_base,2)) + 'kVA.'
```

```
The base load during the warm period is 5702.26kVA.
The base load during the cool period is 4868.41kVA.
```

During the cool period, we can expect that this is the least amount that the HVAC can be operating.

In fact, the days that the temperature peaks around 25C, there is very little peak to the demand graph. Therefore we could assume that it is during those days that the base production load can be calculated. For this analysis we will use November 8, 9, and 10th. These are a Wednesday, Thursday and Friday.

To get a clean number we will take the average of the top 10% of the demand during this period. This will be lower than the actual demand and therefore should be more conservative, while also smoothing any startup vs running load requirements.

In [ ]:
```
cool_production_peak = cool_analysis_power.loc['2023-11-08 00:00:00' : '2023-11-10
top_10per = len(cool_production_peak.index)*0.1
cool_production_peak_n = cool_production_peak.nlargest(n=int(top_10per), columns=['

cool_production_peak_demand = cool_production_peak_n['Apparent Power'].mean()

print('The max production demand during this period was ' + str(round(cool_producti

production_demand = cool_production_peak_demand
```

```
The max production demand during this period was 7075.9kVA.
```

With this value we can assume that any demand over this value is based on the outside temperature. To calculate this we will subtract the production base demand from the total demand. Then divide that by the temperature. That will give use a kVA/C value that can be used in the demand forecast looking further into the system.

In [ ]:
```
warm_analysis_power['Calc HVAC'] = warm_analysis_power['Apparent Power'] - producti
warm_analysis_power['temp'] = warm_analysis_weather['temp']
temp_warm_analysis = warm_analysis_power[warm_analysis_power['temp'].notna()]
temp_warm_analysis['kVA_C'] = temp_warm_analysis['Apparent Power'] / temp_warm_anal
#temp_warm_analysis['kVA_C']

print(temp_warm_analysis['kVA_C'].mean())
```

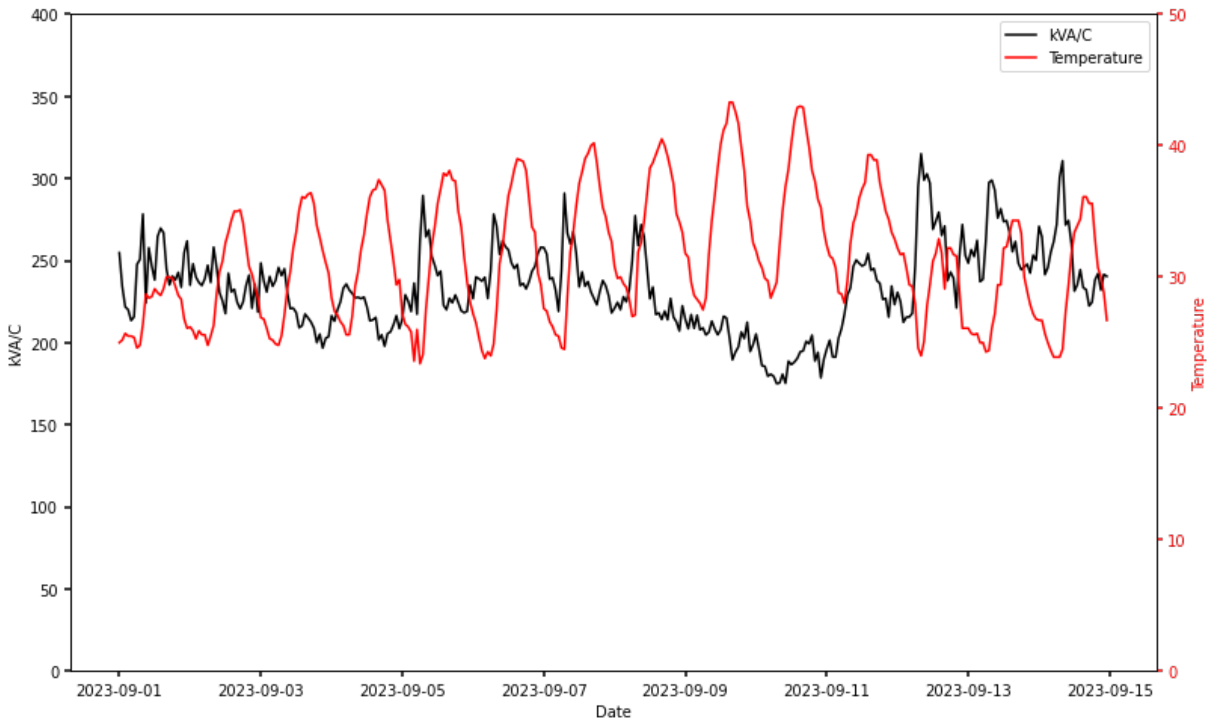
```
233.21688423398362
```

```
C:\Users\jeff\AppData\Local\Temp\ipykernel_15796\2331834001.py:1: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  warm_analysis_power['Calc HVAC'] = warm_analysis_power['Apparent Power'] - product
ion_demand
C:\Users\jeff\AppData\Local\Temp\ipykernel_15796\2331834001.py:2: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  warm_analysis_power['temp'] = warm_analysis_weather['temp']
C:\Users\jeff\AppData\Local\Temp\ipykernel_15796\2331834001.py:4: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  temp_warm_analysis['kVA_C'] = temp_warm_analysis['Apparent Power'] / temp_warm_ana
lysis['temp']
```

In [ ]:

```
# graph the new analysis values

fig, ax = plt.subplots(figsize=(16,8))
fig.subplots_adjust(right=0.75)

twin1 = ax.twinx()
#twin2 = ax.twinx()

# Offset the right spine of twin2.  The ticks and label have already been
# placed on the right by twinx above.
#twin2.spines.right.set_position(("axes", 1.2))

p1, = ax.plot(temp_warm_analysis.index, temp_warm_analysis['kVA_C'], color='black',
p2, = twin1.plot(temp_warm_analysis.index, temp_warm_analysis['temp'], color='red',

#ax.set_xlim(0, 2)
ax.set_ylim(0, 400)
twin1.set_ylim(0, 50)

ax.set_xlabel("Date")
ax.set_ylabel("kVA/C")
twin1.set_ylabel("Temperature")

ax.yaxis.label.set_color(p1.get_color())
twin1.yaxis.label.set_color(p2.get_color())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
twin1.tick_params(axis='y', colors=p2.get_color(), **tkw)
```

```
ax.tick_params(axis='x', **tkw)

ax.legend(handles=[p1, p2])

plt.savefig('report/images/KVA_per_C-Analysis-Sept.png', format='png')


plt.show()
```



In [ ]:
```
print('The kVA/C skew is ' + str(temp_warm_analysis['kVA_C'].skew()))
print('The kVA/C median is ' + str(temp_warm_analysis['kVA_C'].median()))
print('The kVA/C mean is ' + str(temp_warm_analysis['kVA_C'].mean()))
print('The kVA/C standard deviation is ' + str(temp_warm_analysis['kVA_C'].std()))
print('The kVA/C min is ' + str(temp_warm_analysis['kVA_C'].min()))
print('The kVA/C max is ' + str(temp_warm_analysis['kVA_C'].max()))
#print(temp_warm_analysis['kVA_C'].max() - temp_warm_analysis['kVA_C'].mean())
#print(temp_warm_analysis['kVA_C'].mean() - temp_warm_analysis['kVA_C'].min())
```

```
The kVA/C skew is 0.33272830900838163
The kVA/C median is 233.2273214710076
The kVA/C mean is 233.21688423398362
The kVA/C standard deviation is 25.947847945182424
The kVA/C min is 175.0811767578125
The kVA/C max is 315.0097249348958
```

Without more information to this is the closest that we can get to. The median and mean values are close and the max/min value is within 50% of the value. For our forecasting purposes, I think that it makes sense to consider that the HVAC demand will be approximately 250kVA/degree C over 25C. That means that when the outside temperature is over 25C then we should consider adding additional demand to our forecasted demand

Calculate expected new loads

Currently we are expecting that most of South will be coming online.

First we calculate the existing diveristy factor. This is based on the "Final" panel schedules provided during the Phase 1 Arc Flash Analysis. They were sent from Lucid by email on or around April 9, 2021

We are assuming that South will be 100% online during August.

In [ ]:
```
# Calculate load increase, between 2023 and 2022
safety_factor = 0.2

load_increase = (df_MainA_nov23['Apparent Power'].mean()/df_MainA_nov22['Apparent P

print("The mean average in the first two weeks of November 2022 is " + str(df_MainA
```

```
The mean average in the first two weeks of November 2022 is 2640.9917756035234kVA, a
nd the mean average in November 2023 is 6030.846934363955kVA. This represents an inc
rease of 2.7402646188032644.
```

Calculation method

The first order calculation will be to consider the average demand in November in 2022 and 2023. This will help start to give an understanding if there is a risk of overload in the coming months, however it will not take into account the obvious cyclical nature in the November 2023 demand.

We are assuming that the South Loads reflected in the Nov'23 graph will not increase during the study period.

The North is the Stamping system, their expected demand is for commissioning starting in November.

Stamping Load

The Stamping load has been provided by Chris in an email on May 12:

Tandem Line:

- Needed 11/8 (11SO Control Power) - Approx. 650kVA (This is 50% of 1284 kVA)
- Needed 12/6 (12SO, Drive Power) – Approx. 800kVA (This is 50% of 1550kVA
- Needed 12/8 (CBF, Automation) – Approx. 309kVA
- Needed 11/17 (FOL, Automation) – Approx. 240kVA
- Needed 11/24 (EOL, Automation) – Approx. 145kVA

Tryout Press:

- Needed 11/18 General) – Approx. 36kVA
- Needed 11/14 (Cushion) – Approx. 91kVA
- Needed 12/1 12SO Drives) – Approx. 520kVA)

Laser Blanking

- Needed 12/1 Coil Line & Stacker) – Approx. 508kVA
- Needed 12/1 (Laser) – Approx. 241kVA

**Tool & die area **

- Needed 1/1/2024 (General) – Approx. 300kVA

Currently we are assuming that these loads are additive, so everything that is needed in November will be assumed that it will be online all month.

For ease of calculation, and to assume worse case, we are assuming that all the stamping demand will be online starting December 2022.

In [ ]:
```
stamping_connected = 650+800+309+240+145+36+91+520+508+241+300
stamping_diversity = 1

north_connected = stamping_connected * stamping_diversity
```

With the YoY `load_increase` and expected new load in `north_connected` we can estimate the demand in Dec'23 through to March'24

In [ ]:
```
analysis_period_str = '2022-12-01 00:00:00'
analysis_period_end = '2023-03-30 23:45:00'

df_MainA_analysis = df_conditioned.loc[analysis_period_str:analysis_period_end  ]['
df_MainA_analysis['Forecast Demand'] = df_MainA_analysis['Apparent Power'] * load_i

df_MainA_analysis['Forecast Demand'].max()
```

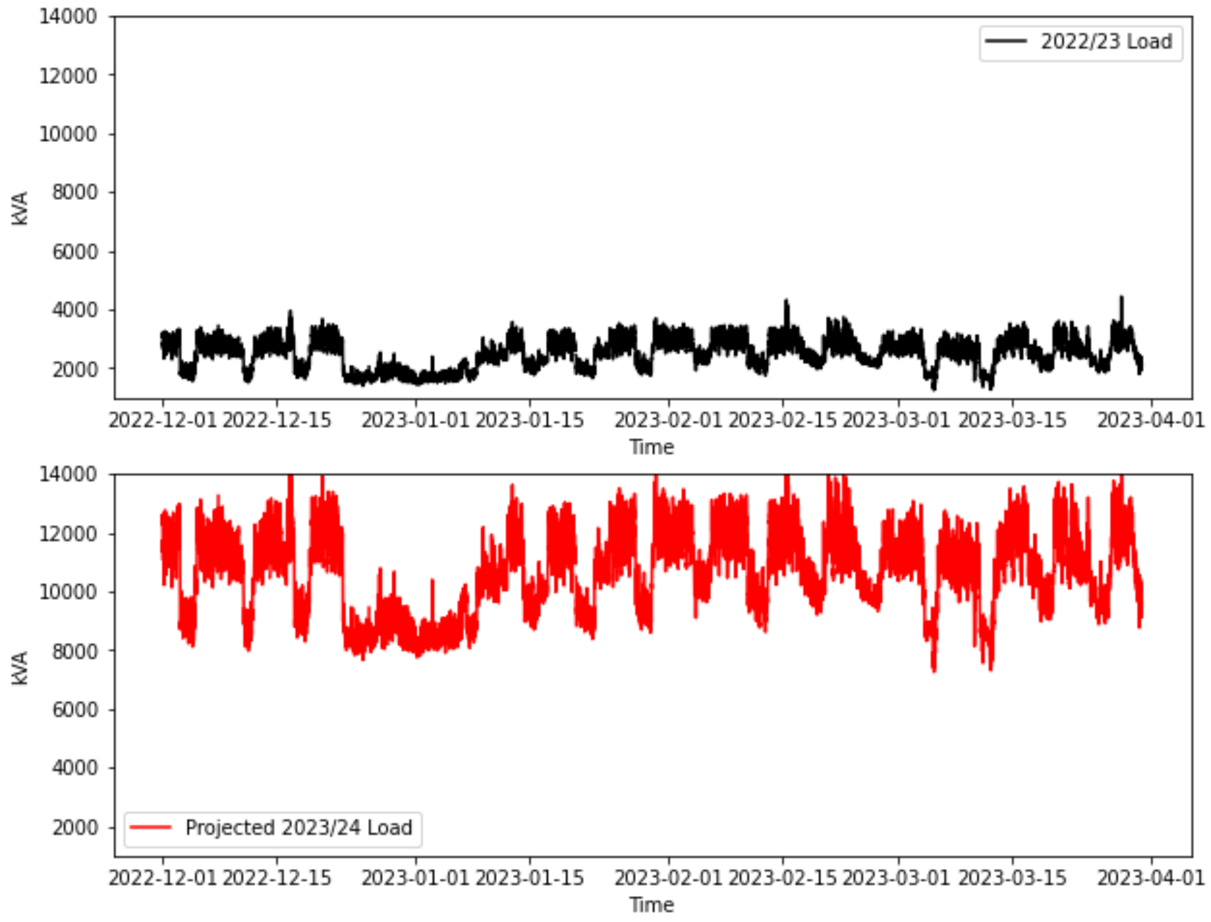
Out[ ]:
```
16008.660177431755
```

In [ ]:
```

```

Comparing the 2022/23 demand with the forecasted 2023/24 Demand

Looking into Q1'24 we can show what we are forecasting the demand will be based on the model developed above. For Q1 the temperature doesn't typically go above 25C, and we will not consider increasing the expected demand at this time.

In [ ]:
```
# Make the comparison plot
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,8))

ax1.plot(df_MainA_analysis.index, df_MainA_analysis['Apparent Power'], color='black

ax1.set_xlabel('Time')
ax1.set_ylabel('kVA')
ax1.set_ylim(plot_y_min,plot_y_max_fans)
ax1.legend()

ax2.plot(df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'], color='red'
ax2.set_xlabel('Time')
ax2.set_ylabel('kVA')
ax2.set_ylim(plot_y_min,plot_y_max_fans)
ax2.legend()
plt.savefig('report/images/Load_Projection_Q124.png', format='png')
```



With the first order load forecast it appears that there are no real issues until April '23.

To be careful, we will check the largest difference between the 15min Low and High Apparent Power, only the high has been used for the analysis thus far. This difference will represent the largest load change that we can expect on the system.

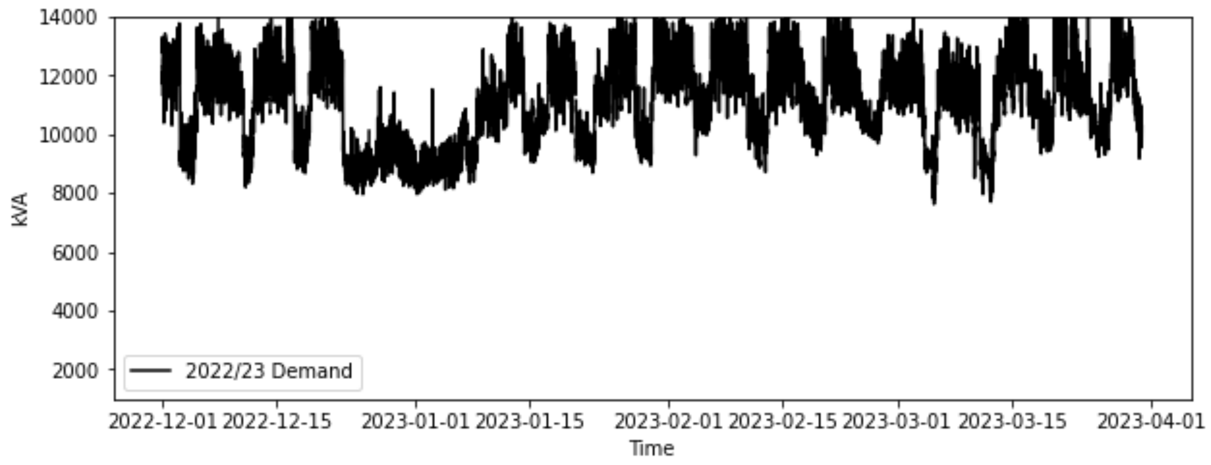
In [ ]:
```
df_MainA_analysis['Difference'] = df_MainA_analysis['Apparent Power'] - df_MainA_an
largest_load = df_MainA_analysis['Difference'].max()
max_demand = df_MainA_analysis['Forecast Demand'].max() + largest_load
```

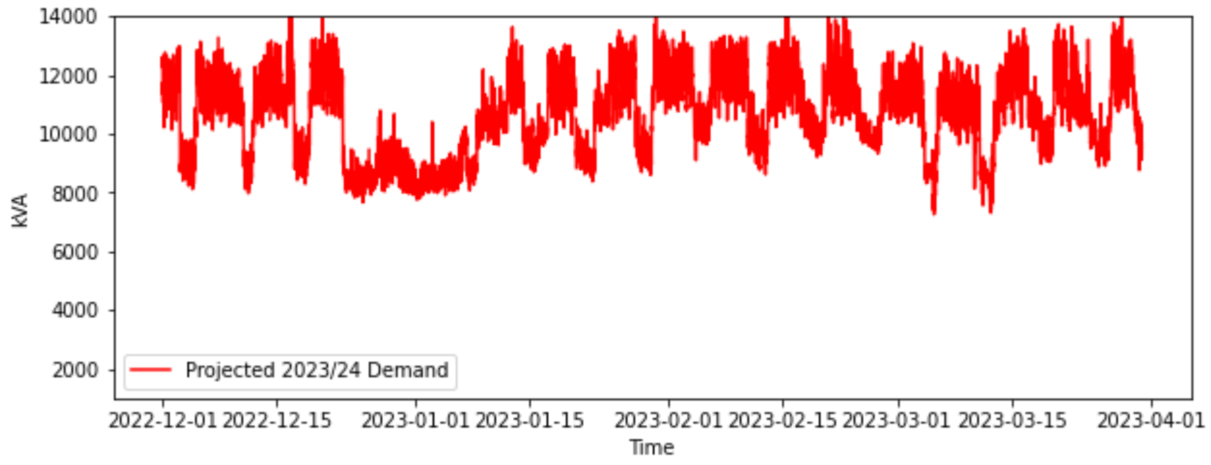
```
print('The max load increase that is represented in the difference between the 15mi

if max_demand > plot_y_max_fans:
    print('This new demand is ' + str(round(max_demand - plot_y_max_fans, 2)) + 'kV

else:
    print('This new demand is within the fan rating of the existing transformer.')
```

```
The max load increase that is represented in the difference between the 15min high a
nd low is 1988.14kVA. When this is added to the peak forecasted demand, the new maxi
mum forecasted demand is 17996.8kVA.
This new demand is 3996.8kVA greater than the fan rating of the existing transforme
r.
```

Using the new max load increase as the new demand, we can graph them to see if this will be a challenge.

In [ ]:
```
# Make the comparison plot
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,8))

ax1.plot(df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'] + df_MainA_a

ax1.set_xlabel('Time')
ax1.set_ylabel('kVA')
ax1.set_ylim(plot_y_min,plot_y_max_fans)
ax1.legend()

ax2.plot(df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'], color='red'
ax2.set_xlabel('Time')
ax2.set_ylabel('kVA')
ax2.set_ylim(plot_y_min,plot_y_max_fans)
ax2.legend()
plt.savefig('report/images/Demand_Projection_Q124.png', format='png')
```



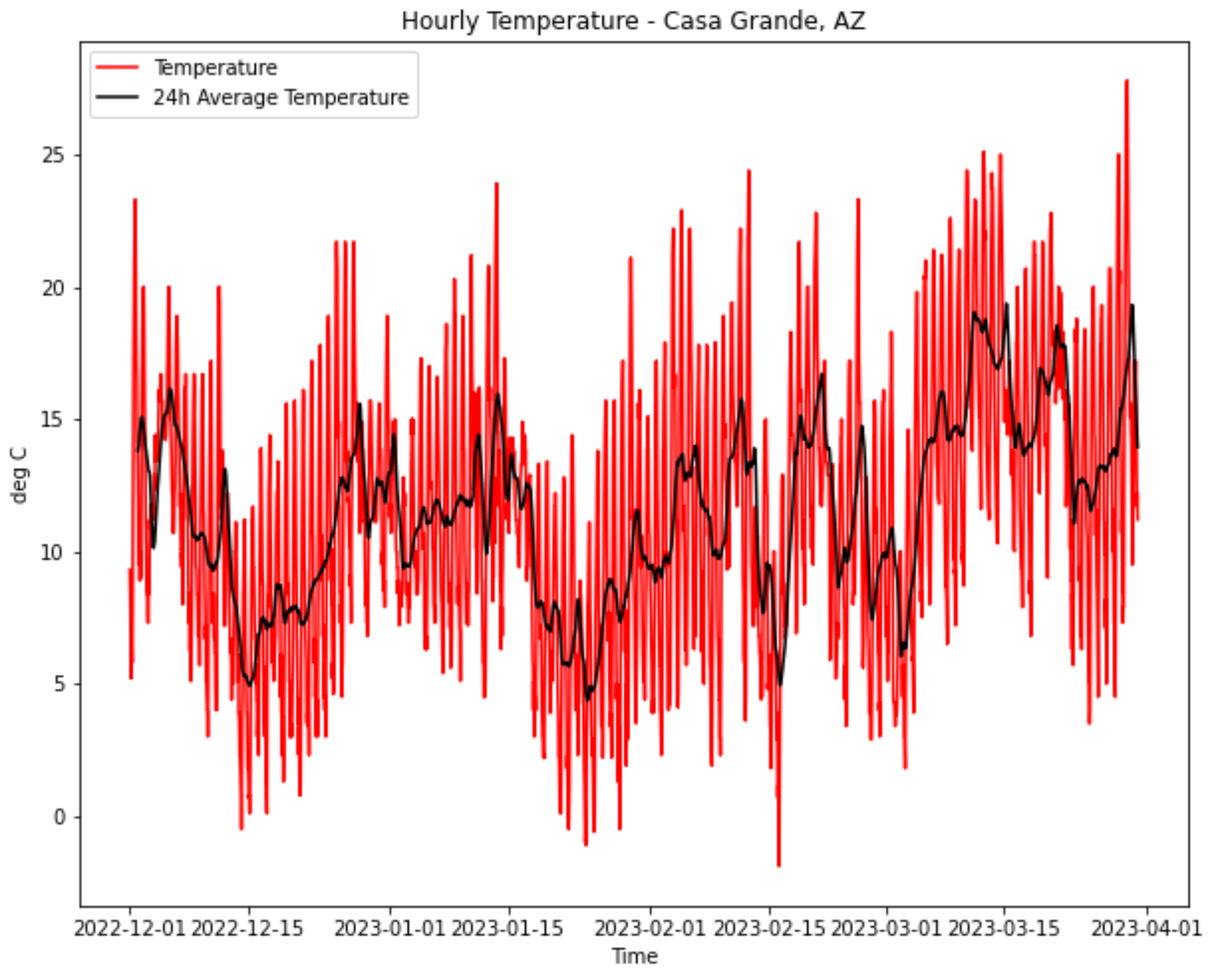


In the black graph above, we can see the areas that the transformer will be over its fan rating, however it is only for very short periods of time, which is likely to be managed with production scheduling.

Transformer Capacity and Temperature Derating

To ensure that the liklihood of derating based on temperature during the analysis period is low, we have completed an hourly temperature analysis using data from VisualCrossing.

In [ ]:
```
# Weather data manipulation

df_weather = weather[analysis_period_str:analysis_period_end].copy()

df_weather['24h_temp'] = df_weather['temp'].rolling(24).mean()

fig, ax1 = plt.subplots(figsize=(10,8))
plt.plot(df_weather.index, df_weather['temp'], color='red', label='Temperature')
plt.plot(df_weather.index, df_weather['24h_temp'], color='black', label='24h Averag

plt.legend()

plt.title('Hourly Temperature - Casa Grande, AZ')
plt.xlabel('Time')
plt.ylabel('deg C')
plt.savefig('report/images/hourly_temperature_casa_grande_2023_Q1.png', format='png
plt.show()
print('The max temperature is ' + str(df_weather['temp'].max()) + ' degC, ' + 'with
```



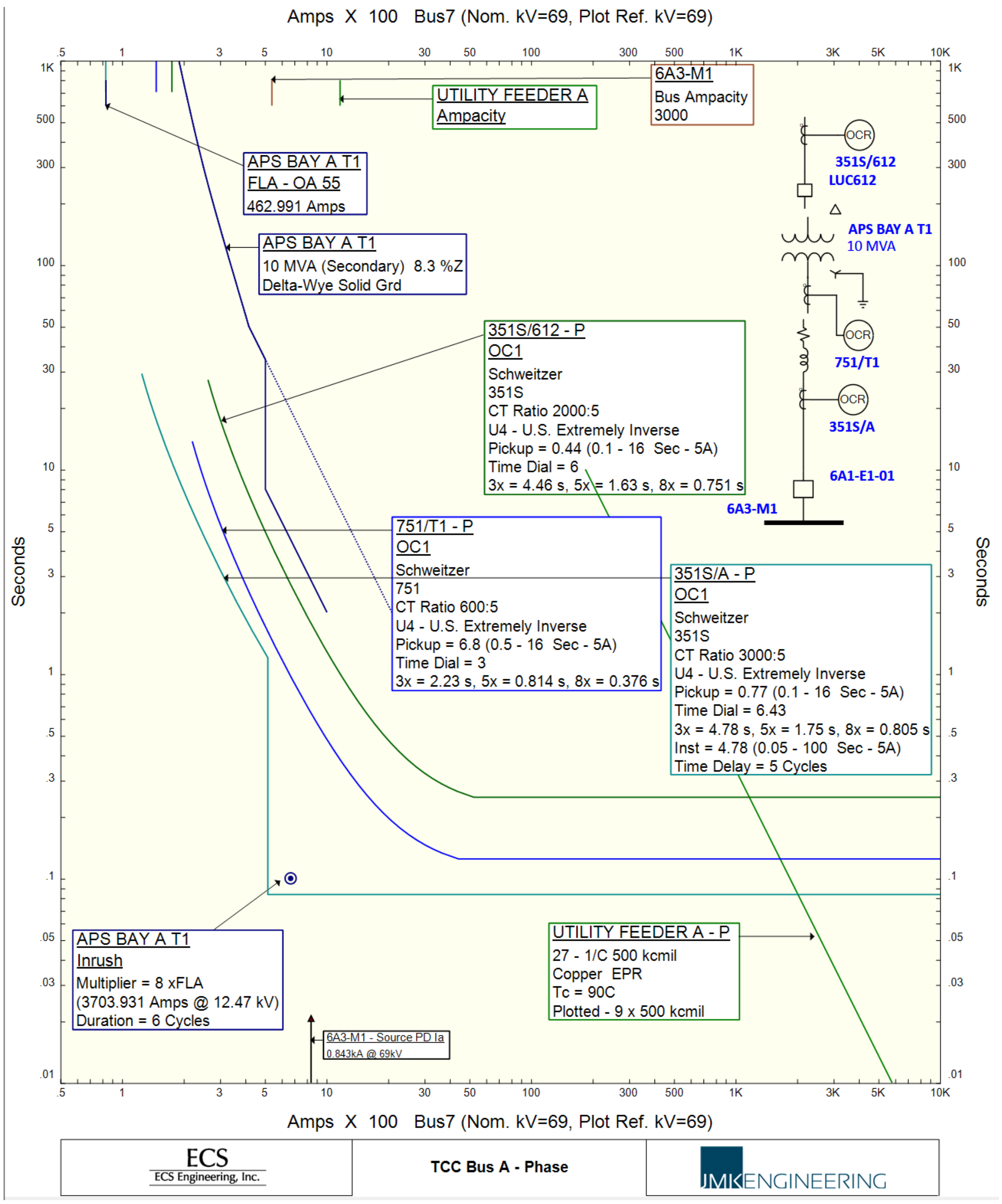
```
The max temperature is 27.8 degC, with the highest 24h average being 19.0 degC.
```

The highest rolling 24h average temperature is less than 30C and therefore the transformer is unlikely to need to be derated as part of the analysis over the analysis period.

Transformer Protection and Tripping

Something that we haven't addressed yet is the existing overload and overcurrent (50/51) protection for the Main service and service transformer. There is a chance that even if the transformer isn't overloaded, being close to the transformer rating, for a long period, may cause the protection to activate.

This next part of the analysis will review that.



Above is the TCC for the switchyard transformer protection and the main 12.47kV breaker in E-House A.

The pickup for these breakers are:

- 351S - 174A @ 69kV
- 751T - 816A @ 12.47kV
- 351S/A - 462A @ 12.47kV (E-House A Main)

The 10MVA transformer full-load amps (FLA) is 462.99A @ 12.47kV, this is the limiting factor for the E-House A main breaker. The other breakers are set to both protect that transformer from damage and coordinate with the E-House A breaker.

These settings are based on the sensed current, and will represent a range of MVA.

```
In [ ]: print('The recorded min voltage is ' + str(round(df_MainA_analysis['Voltage'].min()
```

```
The recorded min voltage is 12291.07V and the max is 12889.1V.
```

To calculate the least MVA values, that match the current settings, we will use the minimum recorded voltage.

```
In [ ]: # Calculate the MVA rating of the protection settings, at low system voltage, to ma

        import math

        voltage = df_MainA_analysis['Voltage'].min()

        mva_351S = math.sqrt(3) * voltage * (69/12.47) * 174 # need to reflect the 69kV to
        mva_751T = math.sqrt(3) * voltage * 816
        mva_EHouseA = math.sqrt(3) * voltage * 462

        print(mva_351S/1000000)
        print(mva_751T/1000000)
        print(mva_EHouseA/1000000)

        # Calculate the max demand currents at low voltage for later in the analysis

        max_load_current = (df_MainA_analysis['Forecast Demand'].max()*1000)/(math.sqrt(3)*
        max_demand_current = ((df_MainA_analysis['Forecast Demand'].max() + largest_load)*1

        print(max_demand_current)
        print(max_load_current)
```

```
20.496621675430628
17.37162950114758
9.835407879326203
845.3662710819922
751.97704993198
```

With the max MVA that the current breaker settings will accomodate calculated, we will compare them to the forecasted values, starting with the E-House A Main Breaker.

```
In [ ]: # Evaluate the E-House A Breaker at low voltage

        if mva_EHouseA/1000 < df_MainA_analysis['Forecast Demand'].max():
            print('The E-House A breaker has a high risk of nuance tripping. The forecasted
        elif mva_EHouseA/1000 < max_demand:
            print('The E-House A breaker has a risk of nuance tripping. The forecasted dema
        else:
            print('The E-House A breaker has a low risk of nuance tripping.')

        # Evaluate the 751T breaker at low voltage
        if mva_751T/1000 < df_MainA_analysis['Forecast Demand'].max():
            print('The 751T breaker has a high risk of nuance tripping. The forecasted dema
        elif mva_751T/1000 < max_demand:
            print('The 751T breaker has a risk of nuance tripping. The forecasted demand is
        else:
```

```
    print('The 751T breaker has a low risk of nuance tripping.')

#Evaluate the 351S breaker at low voltage
if mva_351S/1000 < df_MainA_analysis['Forecast Demand'].max():
    print('The 351S breaker has a high risk of nuance tripping. The forecasted dema
elif mva_351S/1000 < max_demand:
    print('The 351S breaker has a risk of nuance tripping. The forecasted demand is
else:
    print('The 351S breaker has a low risk of nuance tripping.')
```

```
The E-House A breaker has a high risk of nuance tripping. The forecasted demand is h
igher than the breaker setting at 12.29kV.
The 751T breaker has a risk of nuance tripping. The forecasted demand is higher than
the breaker setting at 12.29kV.
The 351S breaker has a low risk of nuance tripping.
```

Based on this analysis we need to determine strategies to either modify the breaker settings, or control the load during low voltage events.

To determine if that is a viable option, lets do the same analysis, but with the high voltage to determine if the risk of nuance tripping lowers.

In [ ]:
```
#Evaluate the breakers at high voltage.
voltage_h = df_MainA_analysis['Voltage'].max()

mva_351S_vh = math.sqrt(3) * voltage_h * (69/12.47) * 174 # need to reflect the 69k
mva_751T_vh = math.sqrt(3) * voltage_h * 816
mva_EHouseA_vh = math.sqrt(3) * voltage_h * 462

# Evaluate the E-House A Breaker at low voltage

if mva_EHouseA_vh/1000 < df_MainA_analysis['Forecast Demand'].max():
    print('The E-House A breaker has a high risk of nuance tripping. The forecasted
elif mva_EHouseA_vh/1000 < df_MainA_analysis['Forecast Demand'].max() + max_demand:
    print('The E-House A breaker has a risk of nuance tripping. The forecasted dema
else:
    print('The E-House A breaker has a low risk of nuance tripping.')

# Evaluate the 751T breaker at low voltage
if mva_751T_vh/1000 < df_MainA_analysis['Forecast Demand'].max():
    print('The 751T breaker has a high risk of nuance tripping. The forecasted dema
elif mva_751T_vh/1000 < df_MainA_analysis['Forecast Demand'].max() + max_demand:
    print('The 751T breaker has a risk of nuance tripping. The forecasted demand is
else:
    print('The 751T breaker has a low risk of nuance tripping.')

#Evaluate the 351S breaker at low voltage
if mva_351S_vh/1000 < df_MainA_analysis['Forecast Demand'].max():
    print('The 351S breaker has a high risk of nuance tripping. The forecasted dema
elif mva_351S_vh/1000 < df_MainA_analysis['Forecast Demand'].max() + max_demand:
    print('The 351S breaker has a risk of nuance tripping. The forecasted demand is
else:
    print('The 351S breaker has a low risk of nuance tripping.')
```

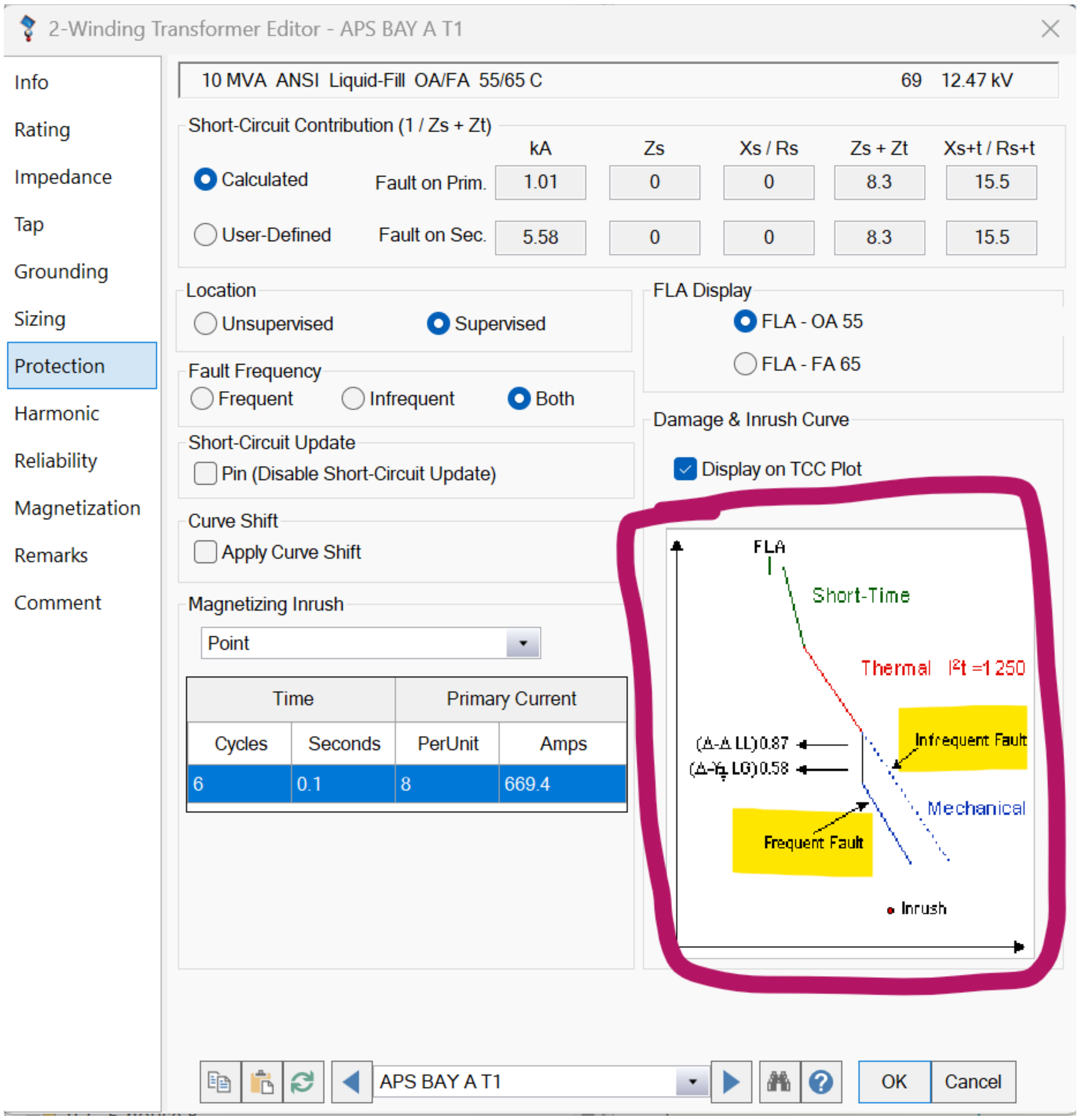
```
The E-House A breaker has a high risk of nuance tripping. The forecasted demand is h
igher than the breaker setting at 12.89kV.
The 751T breaker has a risk of nuance tripping. The forecasted demand is higher than
the breaker setting at 12.89kV.
The 351S breaker has a risk of nuance tripping. The forecasted demand is higher than
the breaker setting at 12.89kV.
```

There is no change in the risk of nuance tripping. The E-House breaker has the highest liklihood of nuance tripping, and it will be the one that we look at first. All the analysis from this point will use the lowest recorded voltage.

Removing Protection Coordination

The simpliest option is to remove the coordination from all the breakers in this part of the system and have them overlap. This doesn't increase the amount of the system that would be affected during an event, and will give the space needed to start-up North while the new transformers are installed.

To do this we would verify that the 12.47kV cabling and bus work can accomodate the added current, and then move all protection settings into the area between the frequent/infrequent fault damage curve of the existing transformer.



2-Winding Transformer Editor - APS BAY A T1

Info | Rating | Impedance | Tap | Grounding | Sizing | Protection | Harmonic | Reliability | Magnetization | Remarks | Comment

10 MVA ANSI Liquid-Fill OA/FA 55/65 C 69 12.47 kV

Short-Circuit Contribution (1 / Zs + Zt)

		kA	Zs	Xs / Rs	Zs + Zt	Xs+t / Rs+t
[✓ Calculated]	Fault on Prim.	1.01	0	0	8.3	15.5
User-Defined	Fault on Sec.	5.58	0	0	8.3	15.5

Location: Unsupervised [✓ Supervised]

FLA Display: [✓ FLA - OA 55] FLA - FA 65

Fault Frequency: Frequent Infrequent [✓ Both]

Damage & Inrush Curve: [✓ Display on TCC Plot]

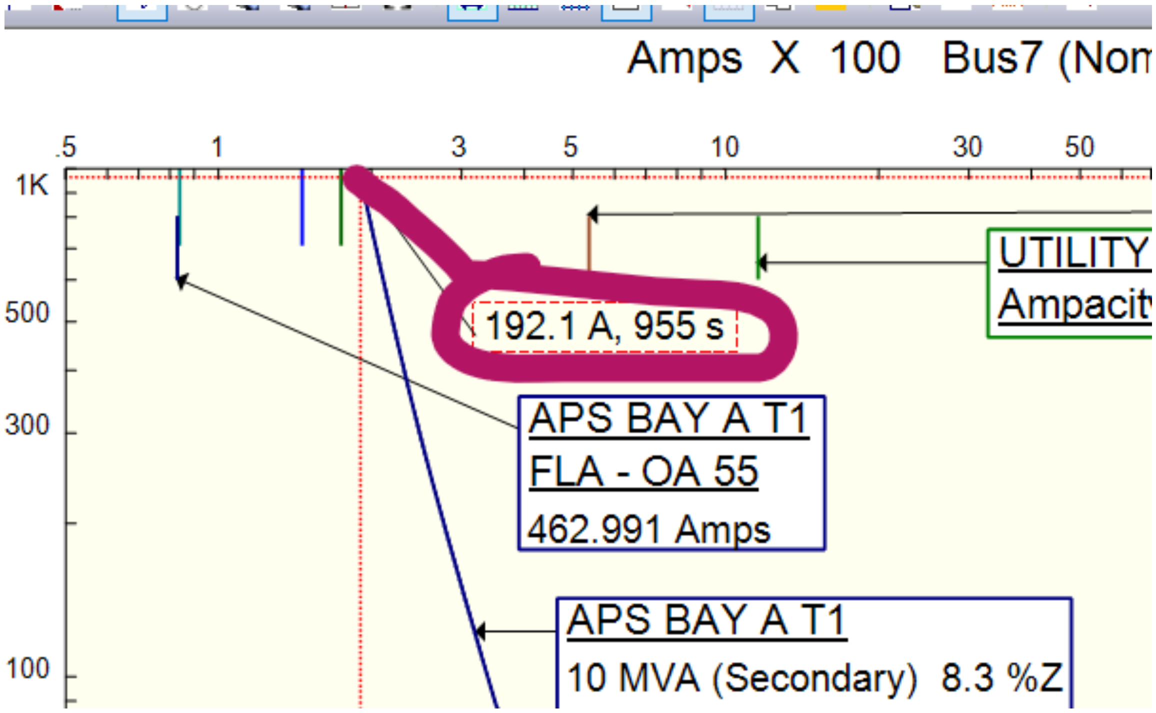
Short-Circuit Update: Pin (Disable Short-Circuit Update)

Curve Shift: Apply Curve Shift

Magnetizing Inrush: Point

Time		Primary Current	
Cycles	Seconds	PerUnit	Amps
6	0.1	8	669.4

We may still have a challenge at the extremes of the transformer damage curve as shown here.



This is the abolsute maximum that we can use for the pick-up on the protection devices.

```
In [ ]: # Determine if the maximum demand is lower than the edge of the damage curve.

        mva_tx_max = math.sqrt(3) * voltage * (69/12.47) * 192

        print('The absolute maximum, at low system voltage, that the protection can be set

        if mva_tx_max > df_MainA_analysis['Forecast Demand'].max() + max_demand:
            print('This demand is higher than the maximum forecasted demand.')
        elif mva_tx_max > df_MainA_analysis['Forecast Demand'].max():
            print('This demand is higher that the forecasted demand, but there may still be
```

```
The absolute maximum, at low system voltage, that the protection can be set at repre
sents 22.62MVA.
This demand is higher than the maximum forecasted demand.
```

Based on the above, the absolute maximum value for the pickup is higher than the Calculated Maximum Demand. From this we can use the max demand to calculate reasonable starting values for the temporary protection settings.

```
In [ ]: # Calculate reasonable max pickup settings

        pickup_safety_factor = 0.000 # 5%

        pickup_1247 = max_demand_current * (1+pickup_safety_factor )
        pickup_69 = (12.47/60) * max_demand_current * (1+pickup_safety_factor )

        print(pickup_1247)
        print(pickup_69)

        new_351S_pickup = pickup_69 * 1.05
        new_751T_pickup = pickup_1247 * 1.1
```

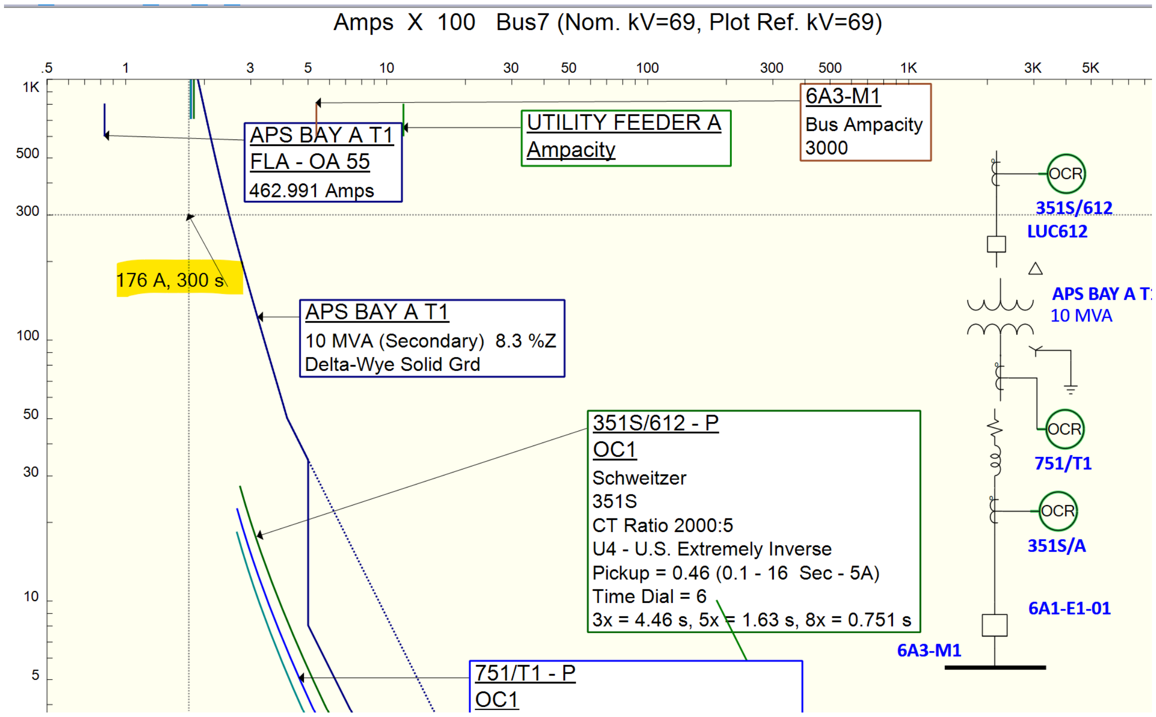
```
new_EHouseA_pickup = pickup_1247

#print(new_351S_pickup)
#print(new_751T_pickup)
#print(new_EHouseA_pickup)

print('The max demand current on the 12.47kV system is projected to be ' + str(roun
print('The temporary pickup values are proposed to be \n - ' + str(round(new_351S_p
```

```
845.3662710819922
175.69529000654072
The max demand current on the 12.47kV system is projected to be 845A, and reflected
on the 69kV system it will be 176A.
The temporary pickup values are proposed to be
 - 184.5A for the primary 351S,
 - 929.9A for the 751T,
 - 845.37A for the E-House A Main.
```

These new values have a risk of pre-maturely aging the transformer, especially during periods when the temperature and demand are high. We recommend increased oil testing, and continuous temperature monitoring through the building automation system, along with active alarming, and/or automatic demand response during high load periods.

These changes may need to be coordinated with the local utility to ensure that the new 69kV breaker settings don't affect any upstream protection devices.

These new settings would look something similar to this:



The highlighted point is the calculated maximum demand, reflected to a 69kV base, for 5 minutes.

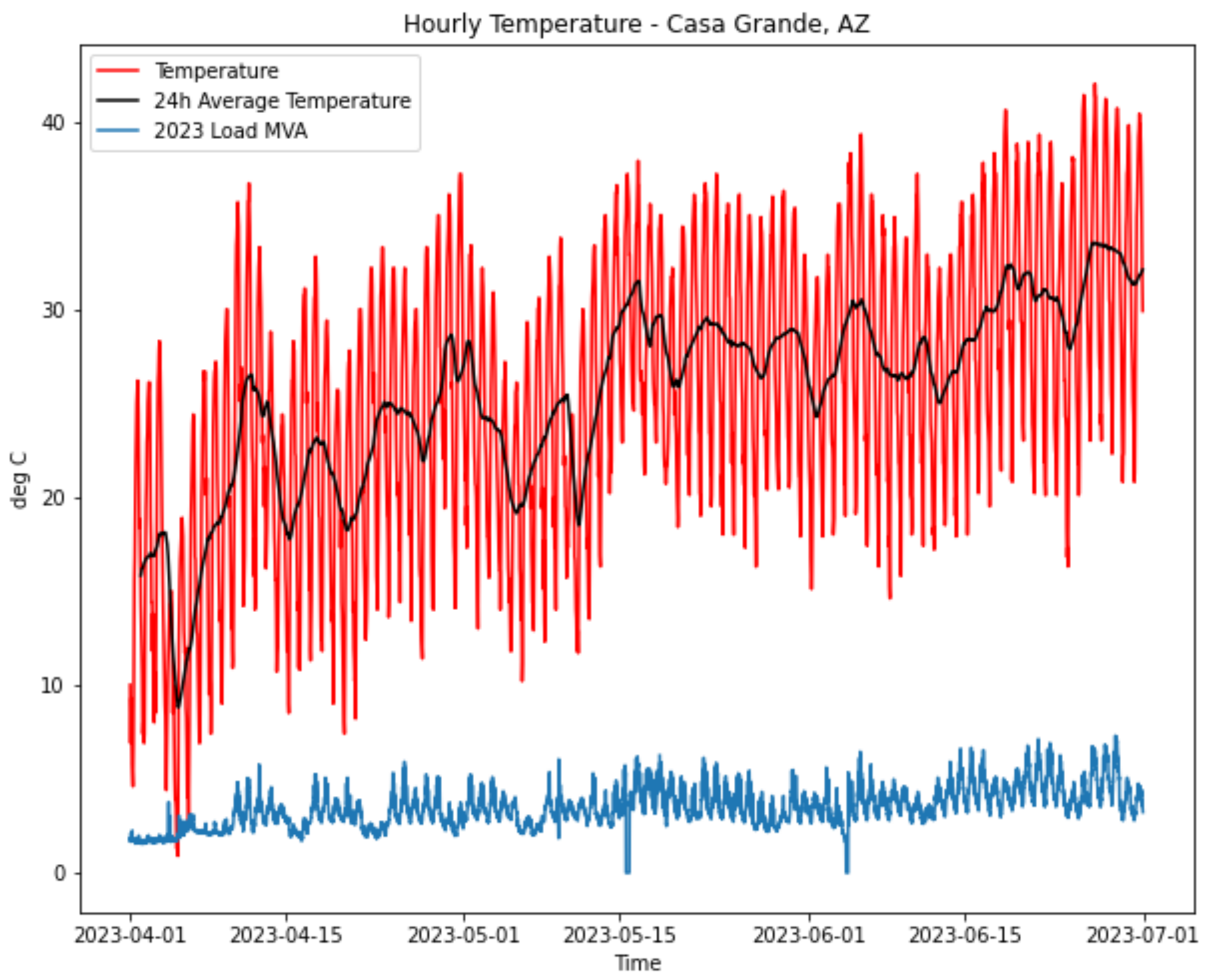
This is TCC has very poor coordination at the pickup, essentially all at the damage curve for the transformer. On warm days this will represent an overload, as such it is critical that oil temperatures are monitored.

Forecasting beyond Q1 into Q2'24

The above analysis is only for Q1'23, extending further into 2023 is difficult without better demand forecasting models, however, there is a benefit to extend the forecasting model that we have developed to determine if there is going to be possible challenges if there are delays.

As we will be getting into the warmer part of the year, we will check to ensure that there isn't a requirement to derate the transformer for this part of the analysis.

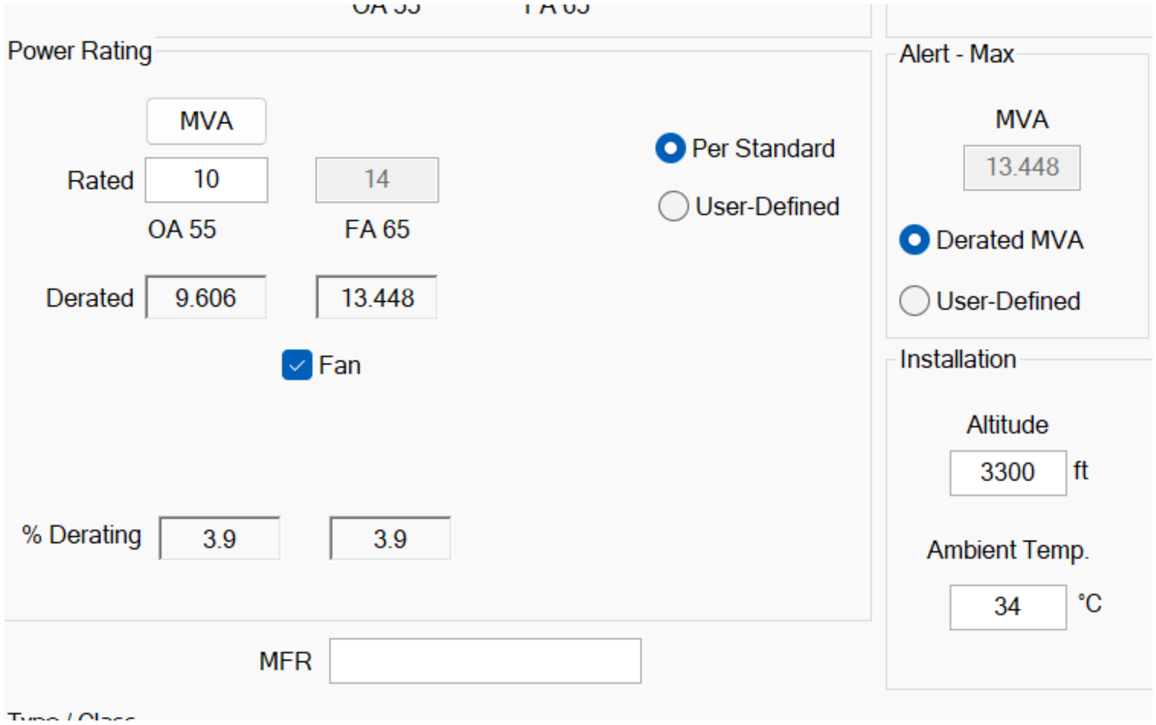
In [ ]:
```
analysis_period_str = '2023-04-01 00:00:00'
analysis_period_end = '2023-06-30 23:45:00'

df_MainA_analysis = df_conditioned.loc[analysis_period_str:analysis_period_end  ]['
df_MainA_analysis['Forecast Demand'] = df_MainA_analysis['Apparent Power'] * load_i
df_MainA_analysis['Difference'] = df_MainA_analysis['Apparent Power'] - df_MainA_an
largest_load = df_MainA_analysis['Difference'].max()
max_demand = df_MainA_analysis['Forecast Demand'].max() + largest_load

df_weather = df_weather = weather[analysis_period_str:analysis_period_end].copy()
df_weather['24h_temp'] = df_weather['temp'].rolling(24).mean()

fig, ax1 = plt.subplots(figsize=(10,8))
plt.plot(df_weather.index, df_weather['temp'], color='red', label='Temperature')
plt.plot(df_weather.index, df_weather['24h_temp'], color='black', label='24h Averag
plt.plot(df_MainA_analysis.index, df_MainA_analysis['Apparent Power']/1000, label='

plt.legend()

plt.title('Hourly Temperature - Casa Grande, AZ')
plt.xlabel('Time')
plt.ylabel('deg C')
plt.savefig('report/images/hourly_temperature_casa_grande_2023_Q2.png', format='png
plt.show()
print('The max temperature is ' + str(df_weather['temp'].max()) + ' degC, ' + 'with
```



```
The max temperature is 42.0 degC, with the highest 24h average being 34.0 degC.
```

Starting in May it appears that the rolling average temperature starts to increase above 30C.

This leads to a derating to 9.6MVA base and 13.44MVA with fans.


Power Rating

	MVA	
Rated	10	14
	OA 55	FA 65
Derated	9.606	13.448
% Derating	3.9	3.9

[✓ Per Standard] User-Defined

[✓ Fan]

MFR

Alert - Max

MVA 13.448

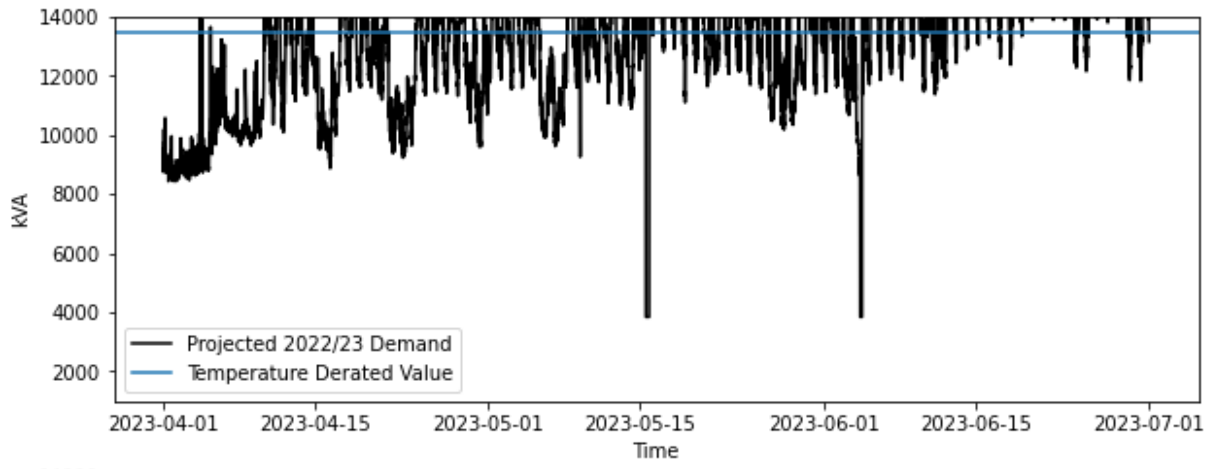
[✓ Derated MVA] User-Defined

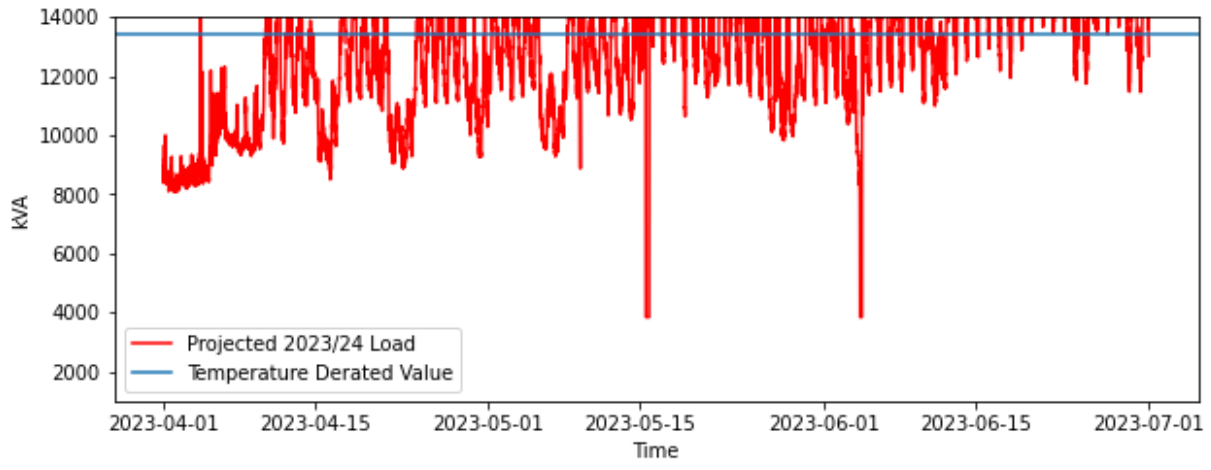
Installation

Altitude 3300 ft

Ambient Temp. 34 °C

```
In [ ]: XF_derated_fan = 13448
        fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,8))

        ax1.plot(df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'] + df_MainA_a

        ax1.set_xlabel('Time')
        ax1.set_ylabel('kVA')
        ax1.set_ylim(plot_y_min,plot_y_max_fans)
        ax1.axhline(y=XF_derated_fan, linestyle='-', label='Temperature Derated Value')
        ax1.legend()

        ax2.plot(df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'], color='red'
        ax2.set_xlabel('Time')
        ax2.set_ylabel('kVA')
        ax2.set_ylim(plot_y_min,plot_y_max_fans)
        ax2.axhline(y=XF_derated_fan, linestyle='-', label='Temperature Derated Value')
        ax2.legend()
        plt.savefig('report/images/Load_Projection_Q224.png', format='png')
```





As you can see above, as the temperature warms up, and assuming that the same YoY increase from 2023-2023 load values in Q2'24 match those that were calculated from Nov'22 and Nov'23, the existing 10MVA transformer will **NOT** accomodate the load, and it is an increase that we don't expect that load shifting will be able to accomodate.

This overload situation will go away when the new transformers are installed.

Additional HVAC load from South and North Campus

These graphs don't take into account the added HVAC load from South or North Campus. For Phase 1 and South we calculated that the kVA/C over 25C was going to be in the range of 250 and 300kVA. Including North in this calculation would increase that.

The values above ONLY account for the HVAC load from Phase 1.

For the below calculation we will assume that the North and South HVAC combined will be similar to the Phase 1 and South that was calculated above.

With that assumption we can calculate what the new HVAC demand, on top of the existing forecasted load, will be when the weather starts to warm in in Early April.

In [ ]:
```
# calculating the hourly demand based on outside temperature.
```

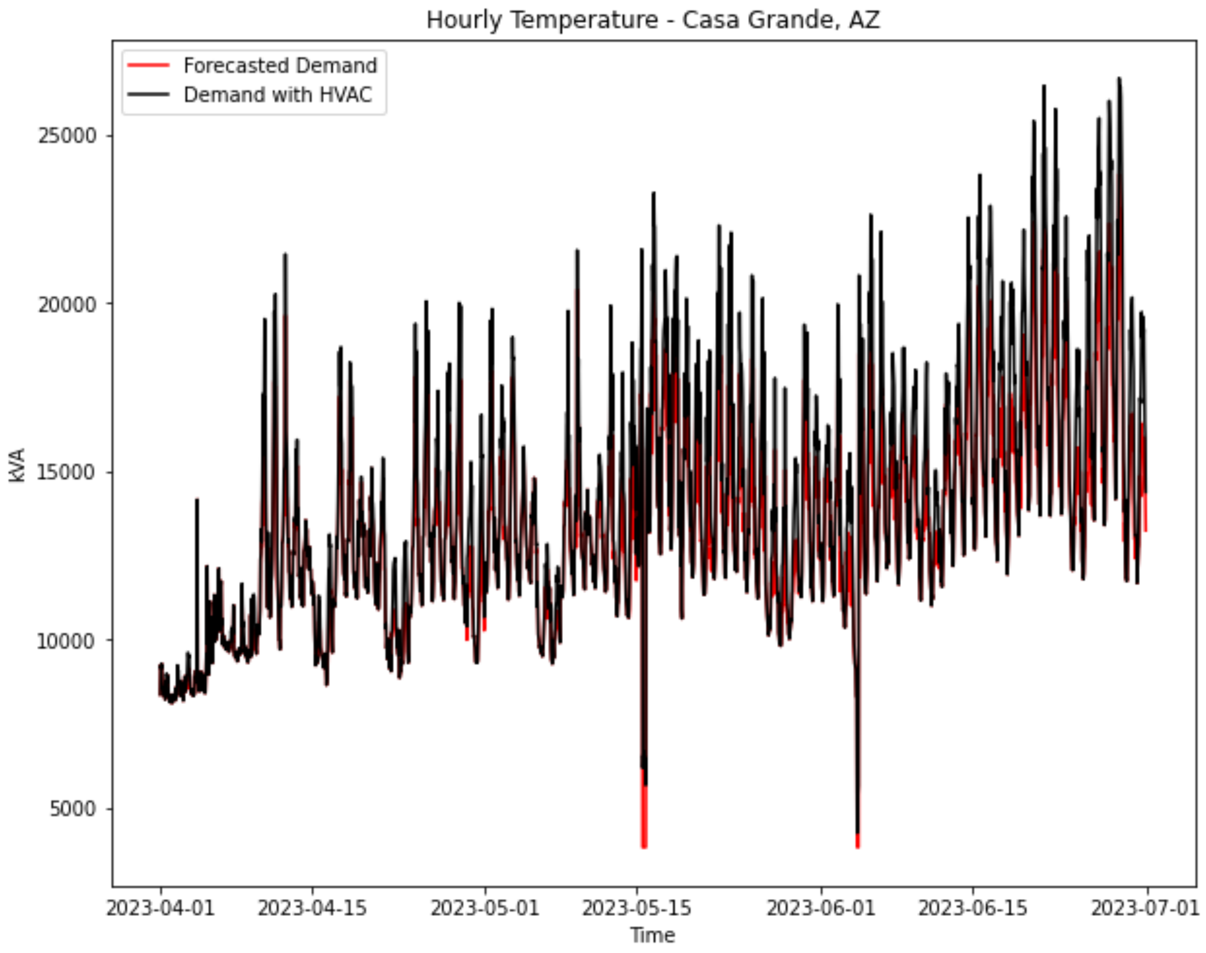
```
MainA_analysis_Q2 = df_MainA_analysis
MainA_analysis_Q2['temp'] = weather['temp']

MainA_analysis_Q2 = MainA_analysis_Q2[MainA_analysis_Q2['temp'].notna()]
MainA_analysis_Q2['HVAC Load'] = np.where(MainA_analysis_Q2['temp'] > 25, (MainA_an
MainA_analysis_Q2['HVAC Load'].max()

print('Based on this analysis there could be up to ' + str(round(MainA_analysis_Q2[
```

```
Based on this analysis there could be up to 3.96MVA of additional HVAC load on the s
ystem in Q2'24
```

```
C:\Users\jeff\AppData\Local\Temp\ipykernel_15796\199403563.py:7: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  MainA_analysis_Q2['HVAC Load'] = np.where(MainA_analysis_Q2['temp'] > 25, (MainA_a
nalysis_Q2['temp']-25)*temp_warm_analysis['kVA_C'].mean(),0)
```

In [ ]:

```
fig, ax1 = plt.subplots(figsize=(10,8))


plt.plot(MainA_analysis_Q2.index, MainA_analysis_Q2['Forecast Demand'], color='red'
plt.plot(MainA_analysis_Q2.index, MainA_analysis_Q2['Forecast Demand'] + MainA_anal

plt.legend()

plt.title('Hourly Temperature - Casa Grande, AZ')
plt.xlabel('Time')
plt.ylabel('kVA')
plt.savefig('report/images/forecasted_demand-_casa_grande_2023_Q2-withHVAC.png', fo
plt.show()
```



As can be seen in the graph, the new peak demand is above 25MVA, and is well over 15MVA most days in later April through to the end of the quarter.

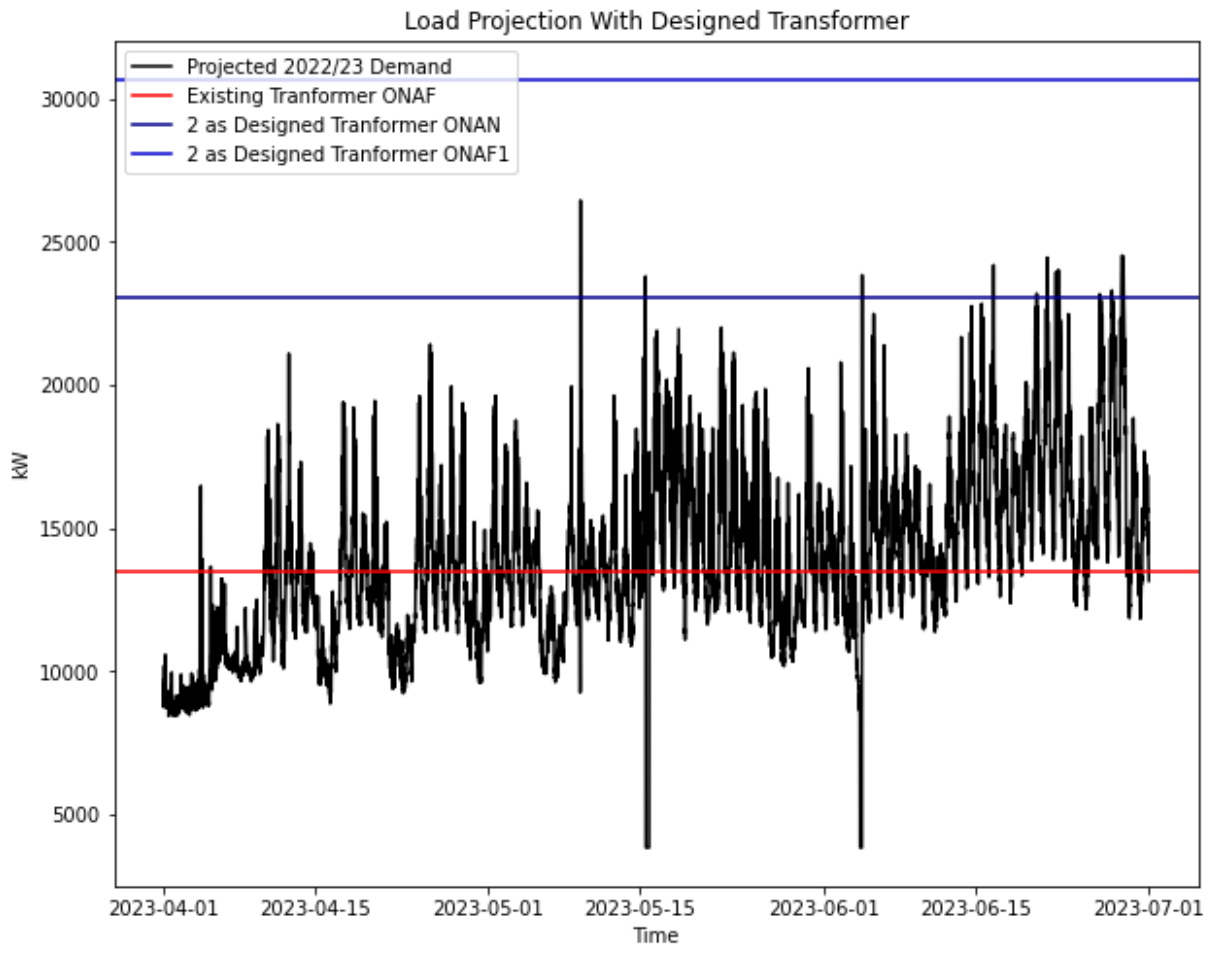
We strongly expect there to be nuance tripping on the 10MVA transformer, along with likely overload damage, if it is not replaced in Q1'24.

Load Projections with Designed Transformer

Below is a graph showing the expected load projection, without the forecasted HVAC, compared to a single 24MVA base transformer. The majority of the time it is not expected for the fans to come on to meet the required demand.

In [ ]:
```
XF_derating_factor = 0.039

XF_base = 10000
XF_fan = 14000
XF_fan_derated = XF_fan * (1-XF_derating_factor)
XF_designed_base = 24000 # Assumed based on the new transformer being purchased.
XF_designed_fan1 = XF_designed_base * 1.33
XF_designed_fan2 = XF_designed_base * 1.66
XF_designed_base_derated = XF_designed_base * (1-XF_derating_factor) # Assumption o
XF_designed_fan1_derated = XF_designed_fan1 * (1-XF_derating_factor)
XF_designed_fan2_derated = XF_designed_fan2 * (1-XF_derating_factor)
```

```
plt.figure(figsize=(10,8))

plt.title('Load Projection With Designed Transformer')
plt.xlabel('Time')
plt.ylabel('kW')

plt.plot (df_MainA_analysis.index, df_MainA_analysis['Forecast Demand'] + df_MainA_

plt.axhline(y=XF_fan_derated, color='red', label='Existing Tranformer ONAF')
plt.axhline(y=XF_designed_base_derated, color='darkblue', label='2 as Designed Tran
plt.axhline(y=XF_designed_fan1_derated, color='mediumblue', label='2 as Designed Tr


plt.legend(loc='upper left')
plt.savefig('report/images/Load_Projection_with_Designed_Transformer.png', format='
plt.show()
```


Load Projection With Designed Transformer
Projected 2022/23 Demand
Existing Tranformer ONAF
2 as Designed Tranformer ONAN
2 as Designed Tranformer ONAF1
kW
Time