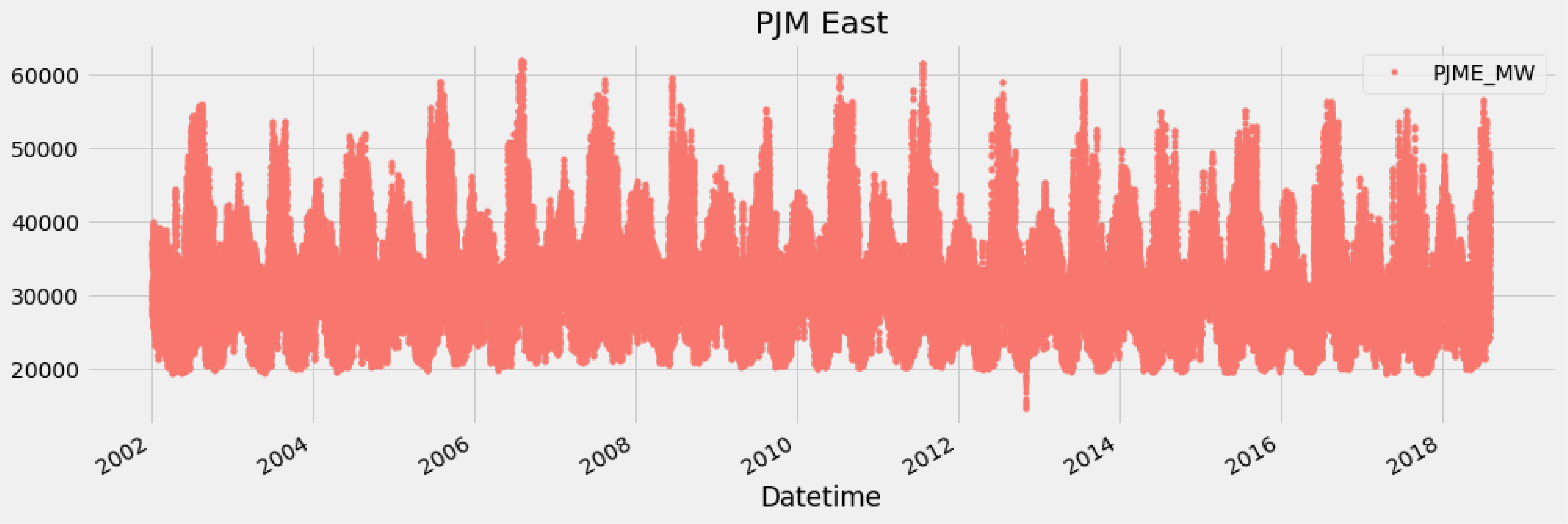
Forecast Model Using Prophet

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from fbprophet import Prophet
from sklearn.metrics import mean_squared_error, mean_absolute_error
plt.style.use('fivethirtyeight') # For plots
```

```
pjme = pd.read_csv('/content/PJME_hourly.csv',
                   index_col=[0], parse_dates=[0]) # We set the index column and know it has dates
```

```
# Color pallete for plotting
color_pal = ["#F8766D", "#D39200", "#93AA00",
             "#00BA38", "#00C19F", "#00B9E3",
             "#619CFF", "#DB72FB"]
pjme.plot(style='.', figsize=(15,5), color=color_pal[0], title='PJM East')
plt.show()
```

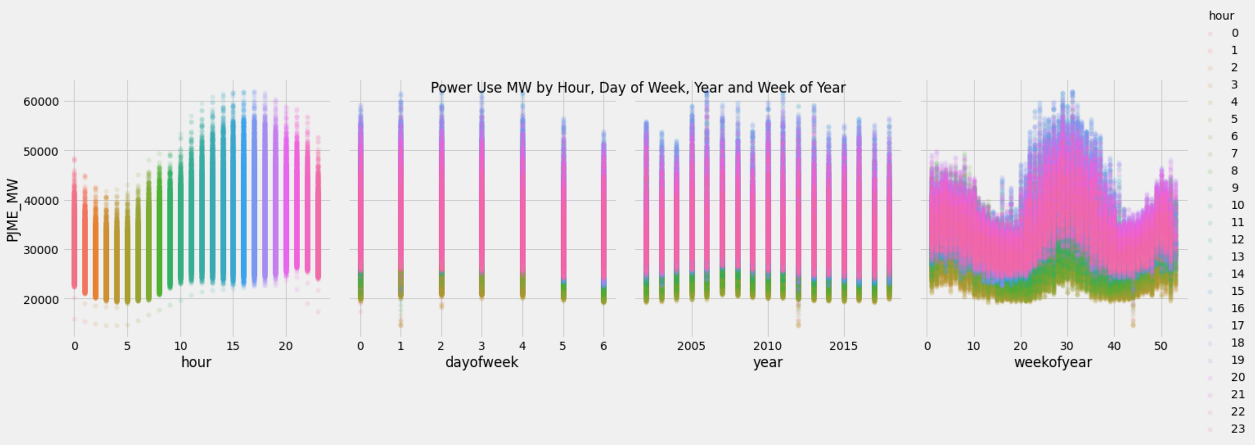


```
def create_features(df, label=None):
    """
    Creates time series features from datetime index.
    """
    df = df.copy()
    df['date'] = df.index
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear

    X = df[['hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear']]
    if label:
        y = df[label]
        return X, y
    return X

X, y = create_features(pjme, label='PJME_MW')

features_and_target = pd.concat([X, y], axis=1)
# See our features and target

```

```
# See our features and target
features_and_target.head()
```

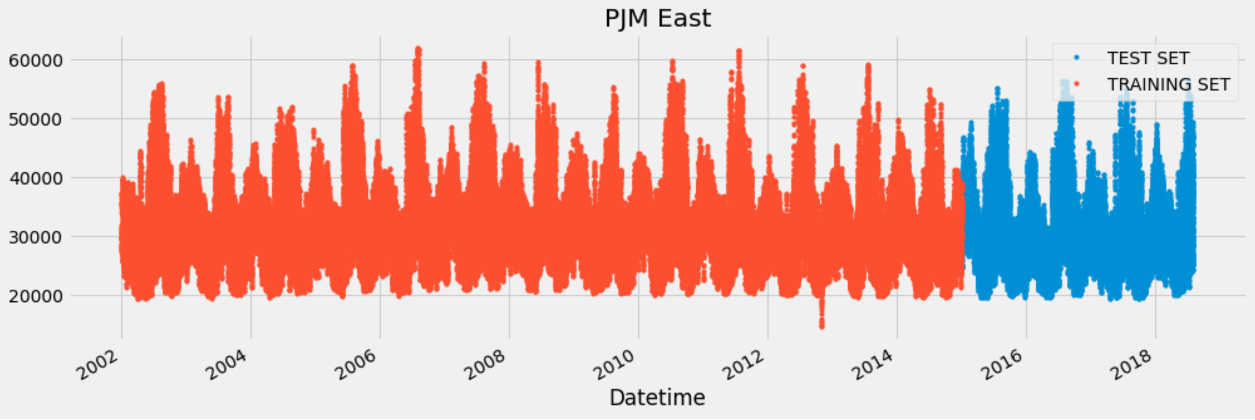
hour dayofweek quarter month year dayofyear dayofmonth weekofyear PJME_MW

```
sns.pairplot(features_and_target.dropna(),
             hue='hour',
             x_vars=['hour','dayofweek',
                     'year','weekofyear'],
             y_vars='PJME_MW',
             height=5,
             plot_kws={'alpha':0.15, 'linewidth':0}
            )
plt.suptitle('Power Use MW by Hour, Day of Week, Year and Week of Year')
plt.show()
```



```
split_date = '01-Jan-2015'
pjme_train = pjme.loc[pjme.index <= split_date].copy()
pjme_test = pjme.loc[pjme.index > split_date].copy()
```

```
# Plot train and test so you can see where we have split
pjme_test \
    .rename(columns={'PJME_MW': 'TEST SET'}) \
    .join(pjme_train.rename(columns={'PJME_MW': 'TRAINING SET'}),
          how='outer') \
    .plot(figsize=(15,5), title='PJM East', style='.')
plt.show()
```



```
# Format data for prophet model using ds and y
pjme_train.reset_index() \
    .rename(columns={'Datetime':'ds',
                     'PJME_MW':'y'}).head()
```

	ds	y
0	2002-12-31 01:00:00	26498.0
1	2002-12-31 02:00:00	25147.0
2	2002-12-31 03:00:00	24574.0
3	2002-12-31 04:00:00	24393.0
4	2002-12-31 05:00:00	24860.0

```
# Setup and train model and fit
model = Prophet()
model.fit(pjme_train.reset_index() \
              .rename(columns={'Datetime':'ds',
                                   'PJME_MW':'y'}))
```

```
<fbprophet.forecaster.Prophet at 0x7fa182a2ee80>
```

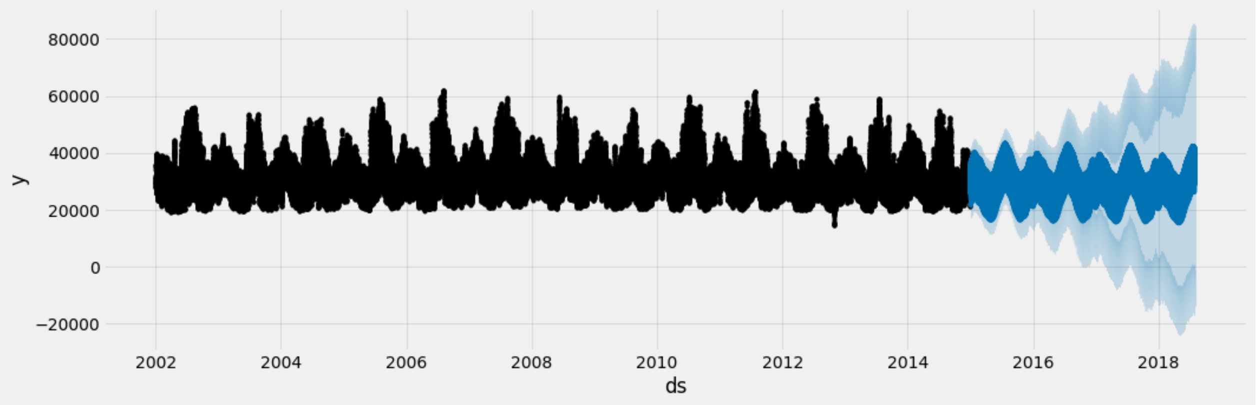
```
# Predict on training set with model
pjme_test_fcst = model.predict(df=pjme_test.reset_index() \
                                   .rename(columns={'Datetime':'ds'}))
```

```
pjme_test_fcst.head()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	addit
0	2015-01-01 01:00:00	31214.768254	24059.550985	33219.415449	31214.768254	31214.768254	-2864.261748	-2864.261748	
1	2015-01-01 02:00:00	31214.731338	22299.213878	31050.073010	31214.731338	31214.731338	-4368.619332	-4368.619332	
2	2015-01-01 03:00:00	31214.694422	21778.269880	30430.150614	31214.694422	31214.694422	-5240.326860	-5240.326860	
3	2015-01-01 04:00:00	31214.657506	21261.359251	30178.233641	31214.657506	31214.657506	-5381.914966	-5381.914966	
4	2015-01-01 05:00:00	31214.620591	22554.406551	31366.301810	31214.620591	31214.620591	-4707.617961	-4707.617961	

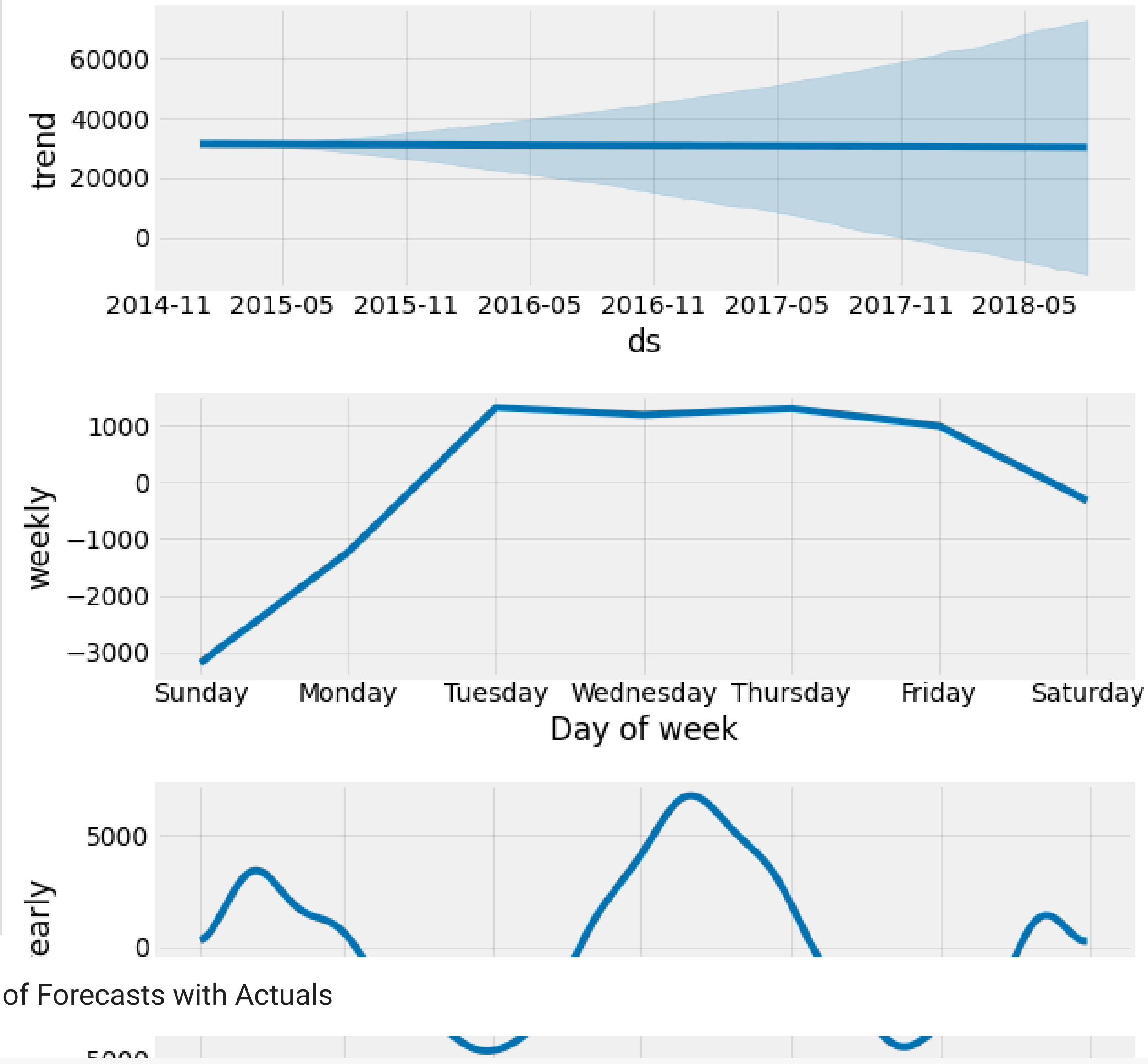
Plot the Forecast

```
# Plot the forecast
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
fig = model.plot(pjme_test_fcst,
                 ax=ax)
plt.show()
```



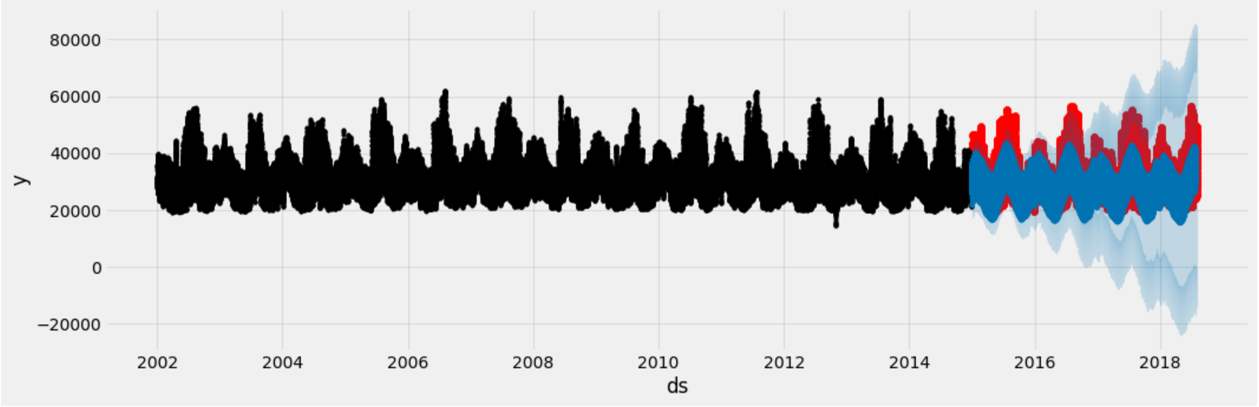
Plot the components of the model: trend, weekly, yearly

```
# Plot the components of the model
fig = model.plot_components(pjme_test_fcst)
```



Plot of Forecasts with Actuals

```
# Plot the forecast with the actuals
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
ax.scatter(pjme_test.index, pjme_test['PJME_MW'], color='r')
fig = model.plot(pjme_test_fcst, ax=ax)
```



Performance Metrics

```
print('Mean Squared Error:\n {}'.format(mean_squared_error(y_true=pjme_test['PJME_MW'],
                   y_pred=pjme_test_fcst['yhat'])))
```

```
Mean Squared Error:
 43761675.09158127
```

```
print("Mean Absolute Error:\n", mean_absolute_error(y_true=pjme_test['PJME_MW'],
                   y_pred=pjme_test_fcst['yhat']))
```

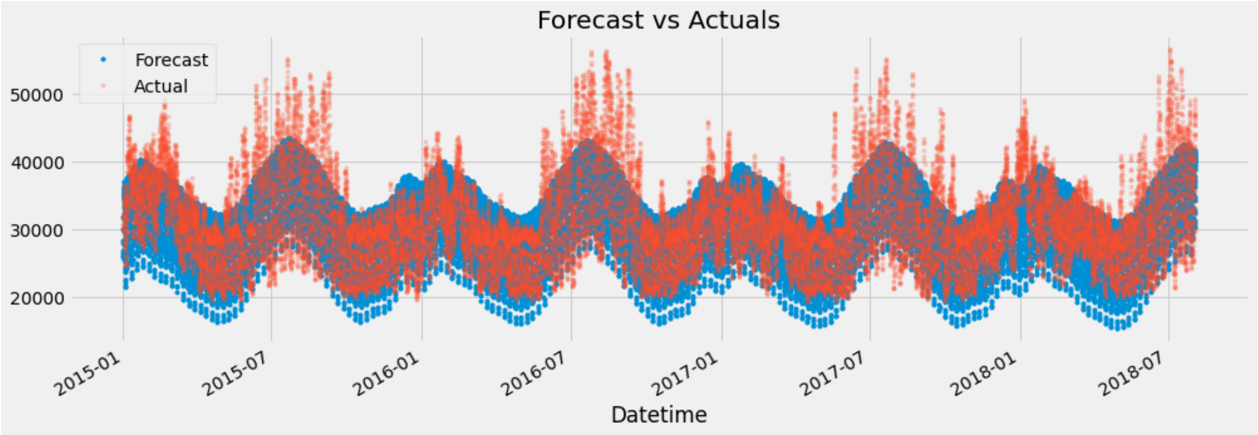
```
Mean Absolute Error:
 5181.782050398612
```

```
def mean_absolute_percentage_error(y_true, y_pred):
    """Calculates MAPE given y_true and y_pred"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

print("Mean Absolute Percentage Error:\n",mean_absolute_percentage_error(y_true=pjme_test['PJME_MW'],
                   y_pred=pjme_test_fcst['yhat']))
```

```
Mean Absolute Percentage Error:
 16.512109913326153
```

Plot Forecasts v. Actuals

```
ax = pjme_test_fcst.set_index('ds')['yhat'].plot(figsize=(15, 5),
                                                 lw=0,
                                                 style='.')
pjme_test['PJME_MW'].plot(ax=ax,
                          style='.',
                          lw=0,
                          alpha=0.2)
plt.legend(['Forecast','Actual'])
plt.title('Forecast vs Actuals')
plt.show()
```


Forecast vs Actuals
Forecast
Actual
Datetime