About the Dataset

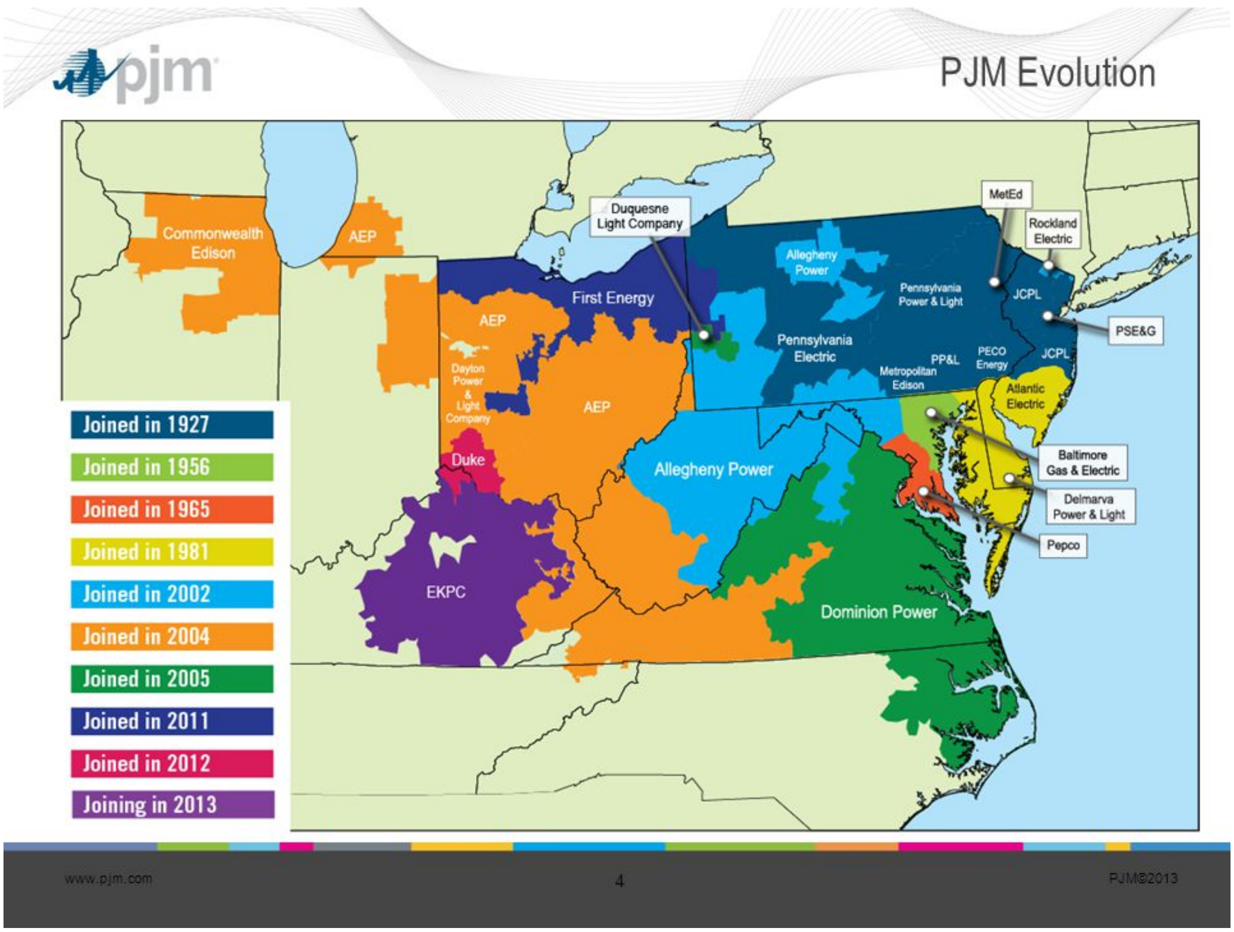
PJM Hourly Energy Consumption Data PJM Interconnection LLC (PJM) is a regional transmission organization (RTO) in the United States. It is part of the Eastern Interconnection grid operating an electric transmission system serving all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia, and the District of Columbia.

The hourly power consumption data comes from PJM's website and are in megawatts (MW).

The regions have changed over the years so data may only appear for certain dates per region.

In [125]:
```
#Show PJM Regions
from IPython.display import Image
Image(url= "http://slideplayer.com/4238181/14/images/4/PJM+Evolution.jpg")
```

Out[125]:



In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [3]:
```
df = pd.read_csv('E:/Data Science with Python/Project/hourly-energy-consumption/I
```

In [4]:
```
df.head()
```

Out[4]:

	Datetime	PJME_MW
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

In [5]:
```
df.describe()
```

Out[5]:

	PJME_MW
count	145366.000000
mean	32080.222831
std	6464.012166
min	14544.000000
25%	27573.000000
50%	31421.000000
75%	35650.000000
max	62009.000000

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145366 entries, 0 to 145365
Data columns (total 2 columns):
Datetime    145366 non-null object
PJME_MW     145366 non-null float64
dtypes: float64(1), object(1)
memory usage: 2.2+ MB
```

Checking for null values

In [7]:
```
df.isna().any()
```

Out[7]:
```
Datetime    False
PJME_MW     False
dtype: bool
```

Drop any duplicate values

In [8]:
```
df.drop_duplicates(subset='Datetime', keep='last', inplace=True)
```

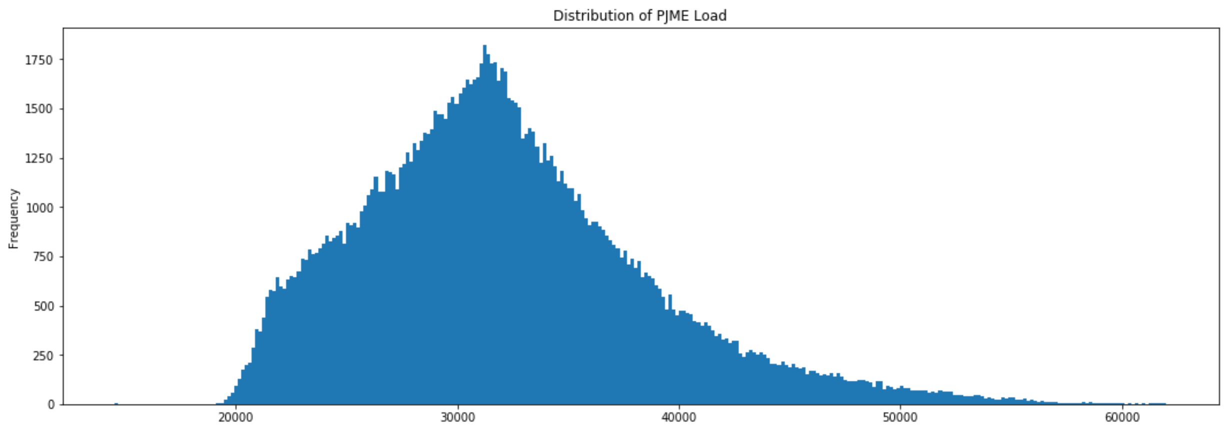
Convert into datetime and set as index

In [9]:
```
df['Datetime']= pd.to_datetime(df['Datetime'])
```

In [10]:
```
df = df.set_index('Datetime') # set the Datetime as our index
```

Histogram plot

In [11]:
```
df['PJME_MW'].plot.hist(figsize=(18, 6), bins=300, title='Distribution of PJME L
plt.show()
```



check if our dataset is continuous

In [12]:
```
print(df.index.freq) # our dataset is not continuous
```

```
None
```

In [13]:
```
# creating a continuous date range with hourly frequency
date_range = pd.date_range(start=min(df.index),
                           end=max(df.index),
                           freq='H')
```

In [14]:
```
df = df.reindex(date_range)
```

In [15]:
```
df.isnull().any()
```

Out[15]:
```
PJME_MW    True
dtype: bool
```

now we have null values lets fill them

In [16]:
```
df = df.fillna(method = 'ffill') # using the ffill technique
```

In [17]:
```
df.isnull().any()
```

Out[17]:
```
PJME_MW    False
dtype: bool
```

In [18]:
```
print(df.index.freq) # now our timeseries is continuous with no missing data
```

```
<Hour>
```

Extracting more features from the time series

In [19]:
```
df['dow'] = df.index.dayofweek
df['doy'] = df.index.dayofyear
df['year'] = df.index.year
df['month'] = df.index.month
df['quarter'] = df.index.quarter
df['hour'] = df.index.hour
df['weekday'] = df.index.weekday_name
df['woy'] = df.index.weekofyear
df['dom'] = df.index.day # Day of Month
df['date'] = df.index.date

# let's add the season number
df['season'] = df['month'].apply(lambda month_number: (month_number%12 + 3)//3)
```

In [20]:
```
df.head()
```

Out[20]:

	PJME_MW	dow	doy	year	month	quarter	hour	weekday	woy	dom	date	season
2002-01-01 01:00:00	30393.0	1	1	2002	1	1	1	Tuesday	1	1	2002-01-01	1
2002-01-01 02:00:00	29265.0	1	1	2002	1	1	2	Tuesday	1	1	2002-01-01	1
2002-01-01 03:00:00	28357.0	1	1	2002	1	1	3	Tuesday	1	1	2002-01-01	1
2002-01-01 04:00:00	27899.0	1	1	2002	1	1	4	Tuesday	1	1	2002-01-01	1
2002-01-01 05:00:00	28057.0	1	1	2002	1	1	5	Tuesday	1	1	2002-01-01	1

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 145392 entries, 2002-01-01 01:00:00 to 2018-08-03 00:00:00
Freq: H
Data columns (total 12 columns):
PJME_MW    145392 non-null float64
dow        145392 non-null int64
doy        145392 non-null int64
year       145392 non-null int64
month      145392 non-null int64
quarter    145392 non-null int64
hour       145392 non-null int64
weekday    145392 non-null object
woy        145392 non-null int64
dom        145392 non-null int64
date       145392 non-null object
season     145392 non-null int64
dtypes: float64(1), int64(9), object(2)
memory usage: 14.4+ MB
```

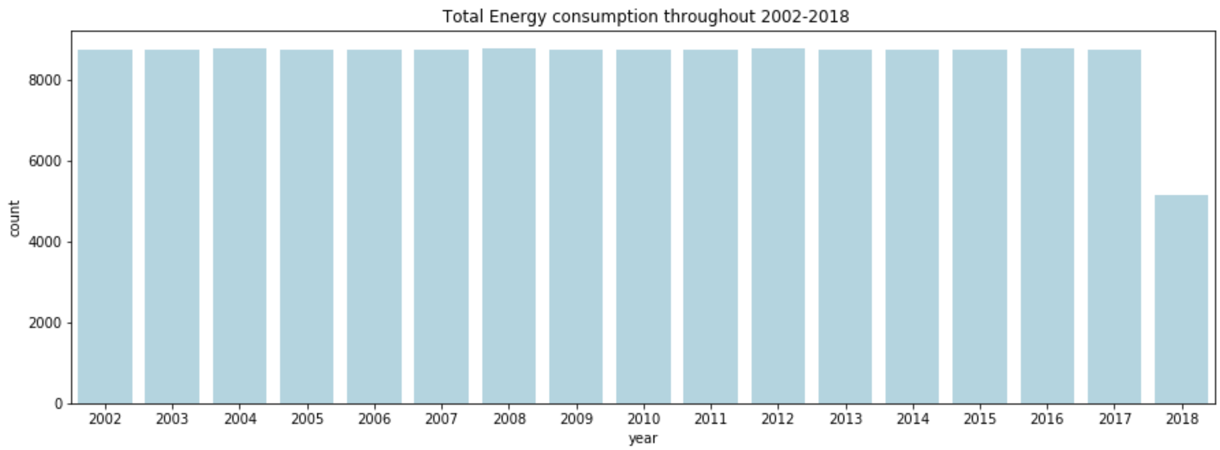
In [22]:
```
df.describe().T
```

Out[22]:

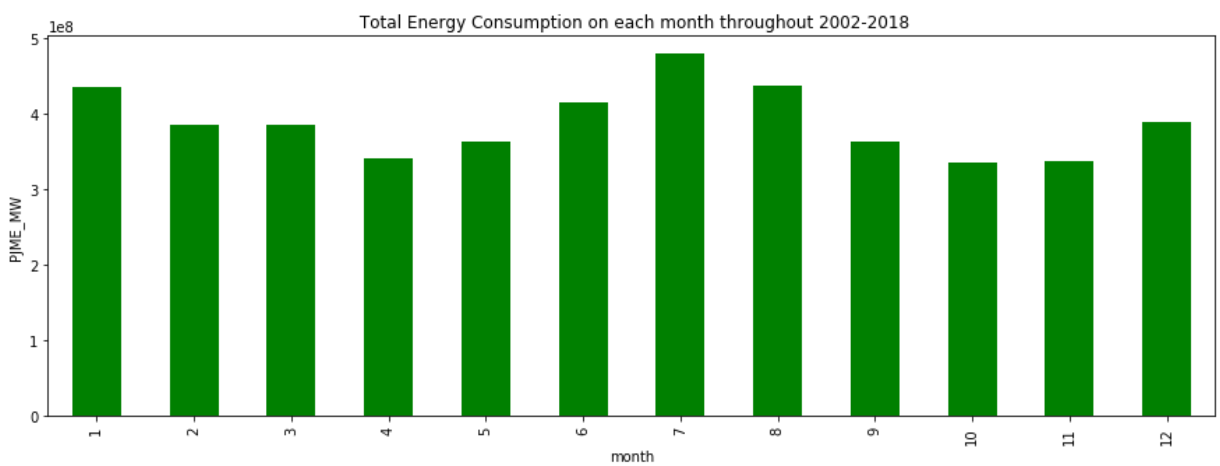
	count	mean	std	min	25%	50%	75%	max
PJME_MW	145392.0	32079.051440	6464.154940	14544.0	27571.00	31420.0	35647.25	62009.0
dow	145392.0	2.999525	1.999713	0.0	1.00	3.0	5.00	6.0
doy	145392.0	180.455252	105.138733	1.0	90.00	179.0	271.00	366.0
year	145392.0	2009.800704	4.791740	2002.0	2006.00	2010.0	2014.00	2018.0
month	145392.0	6.435836	3.438967	1.0	3.00	6.0	9.00	12.0
quarter	145392.0	2.481196	1.114472	1.0	1.00	2.0	3.00	4.0
hour	145392.0	11.500000	6.922210	0.0	5.75	11.5	17.25	23.0
woy	145392.0	26.217935	15.019948	1.0	13.00	26.0	39.00	53.0
dom	145392.0	15.722529	8.801313	1.0	8.00	16.0	23.00	31.0
season	145392.0	2.485983	1.107560	1.0	2.00	2.0	3.00	4.0

Exploratory Data Analysis

In [23]:
```
plt.figure(figsize=(15,5))
plt.title(' Total Energy consumption throughout 2002-2018')
sns.countplot(x='year', data=df, color='lightblue');
```

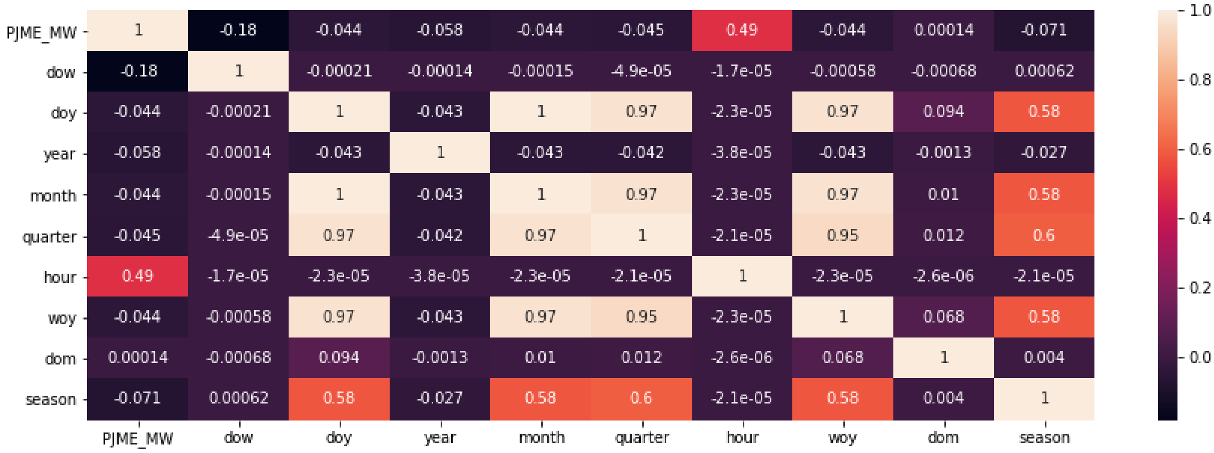


In [24]:
```
plt.figure(figsize=(15,5))
plt.title(' Total Energy Consumption on each month throughout 2002-2018')
plt.ylabel('PJME_MW')
plt=df.groupby('month').PJME_MW.sum().plot(kind='bar',color='green')
```



Correlation plot

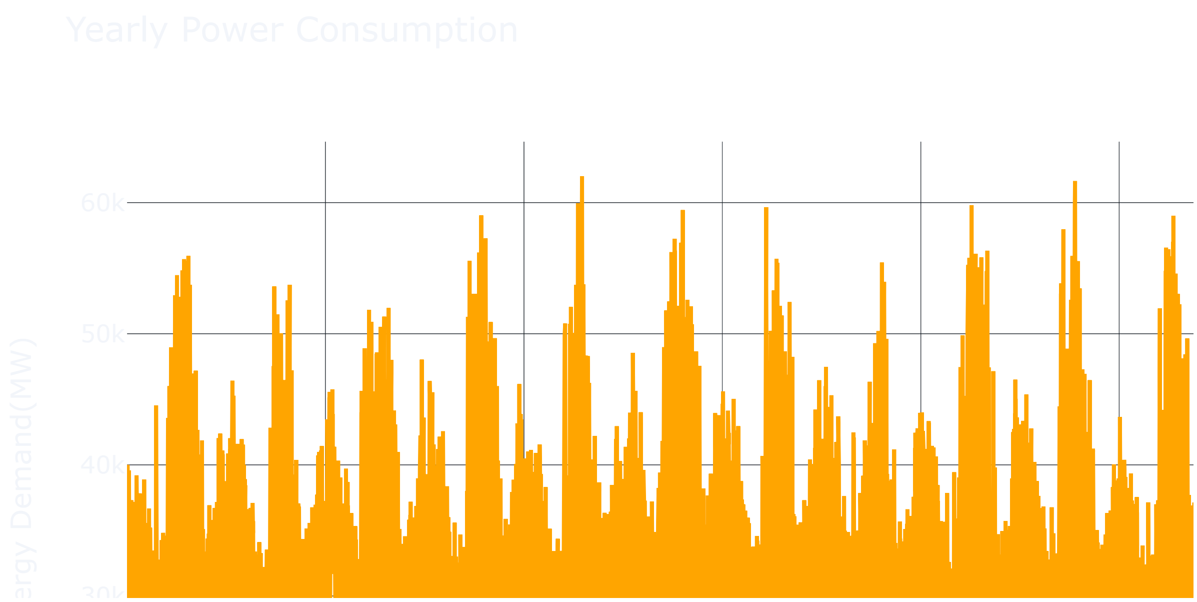
In [25]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



- The correlation matrix indicates the variables "dow" (day of week) and "hour" will be interesting to look at in the context of predicting our target variable.

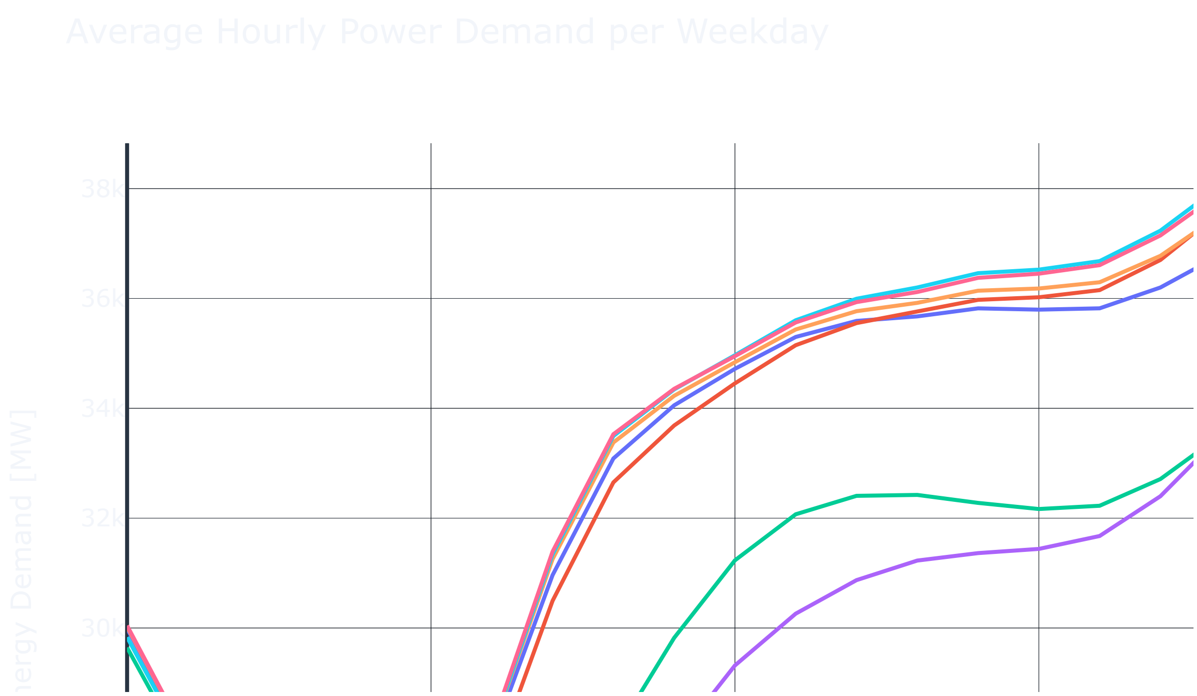
Time series plot

In [26]:
```
import plotly.graph_objects as go
fig = go.Figure([go.Scatter(x=df.index, y=df.PJME_MW,line_color='orange')])
fig.update_layout(title_text='Yearly Power Consumption',template="plotly_dark")
fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Energy Demand(MW)')
fig.show()
```



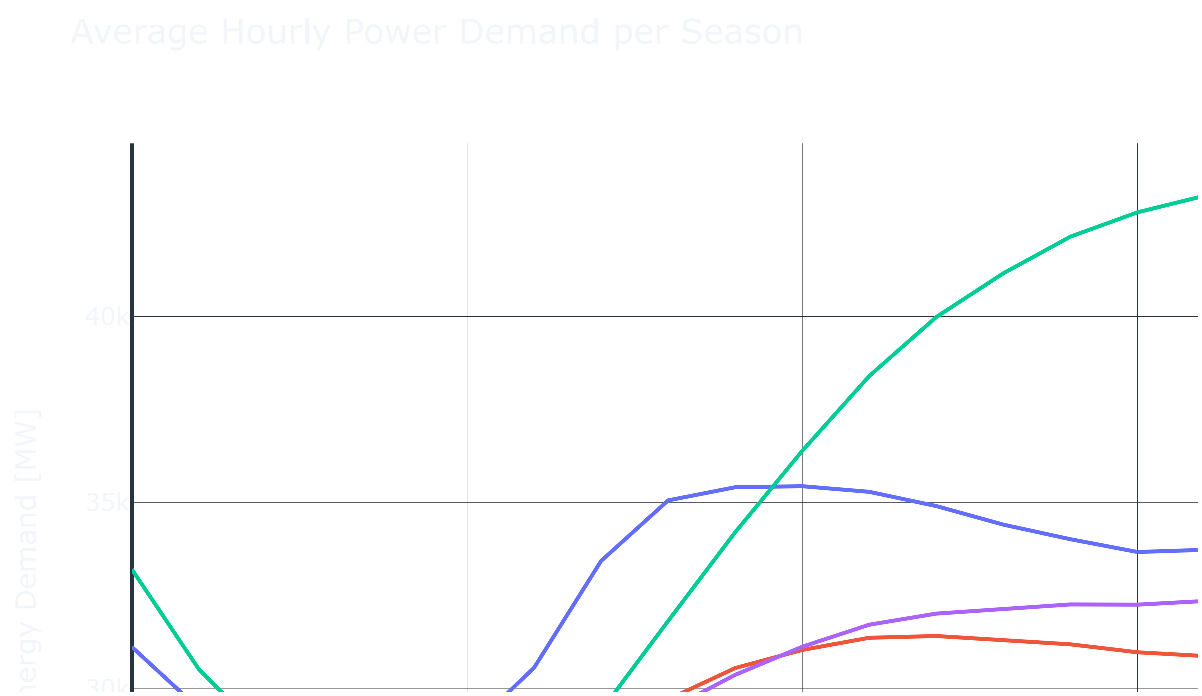
Hourly,Daily,Weekly Timeseries Analysis

In [27]:
```
import plotly.express as px
plot_df=df.groupby(['hour', 'weekday'], as_index=False).agg({'PJME_MW':'mean'})
# plotting
fig = px.line(plot_df, x='hour', y='PJME_MW', color='weekday', title='Average Hou
fig.update_layout(xaxis_title='Hour',yaxis_title='Energy Demand [MW]',template="
fig.show()
```



- demand for electricity is lower during the weekends, and dips a little sooner on friday afternoons.

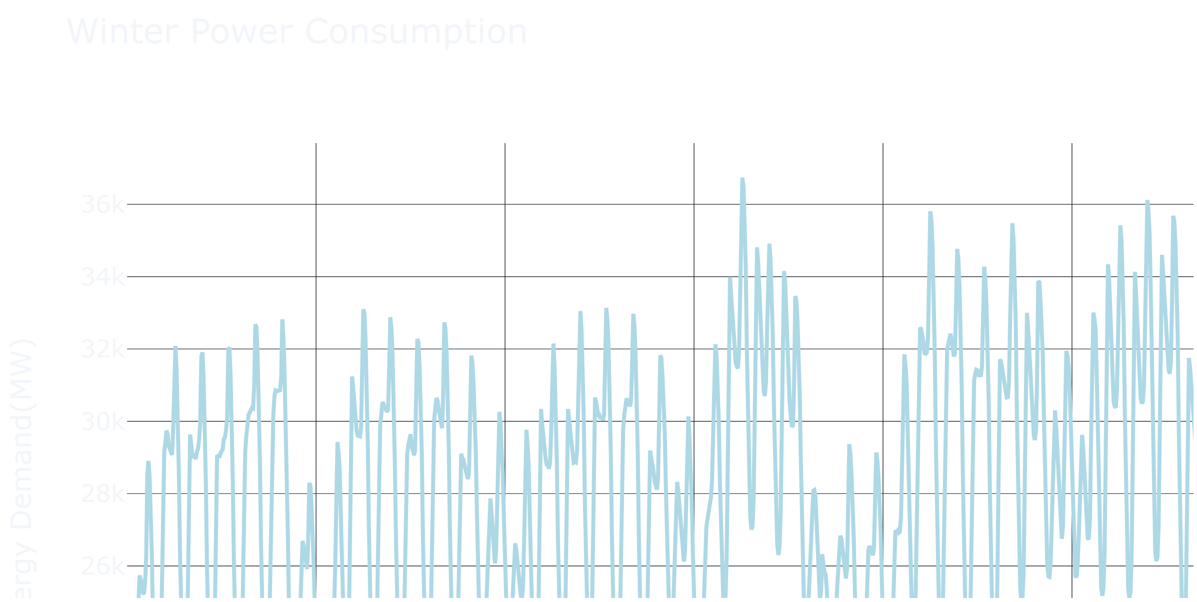
In [28]:
```
plt_df=df.groupby(['hour', 'season'], as_index=False).agg({'PJME_MW':'mean'})

# plotting
fig = px.line(plt_df,x='hour', y='PJME_MW', color='season', title='Average Hourl
fig.update_layout(xaxis_title='Hour',
                  yaxis_title='Energy Demand [MW]',template="plotly_dark")
fig.show()
```



- The Energy Consumption during season 3 i.e Summer is the highest

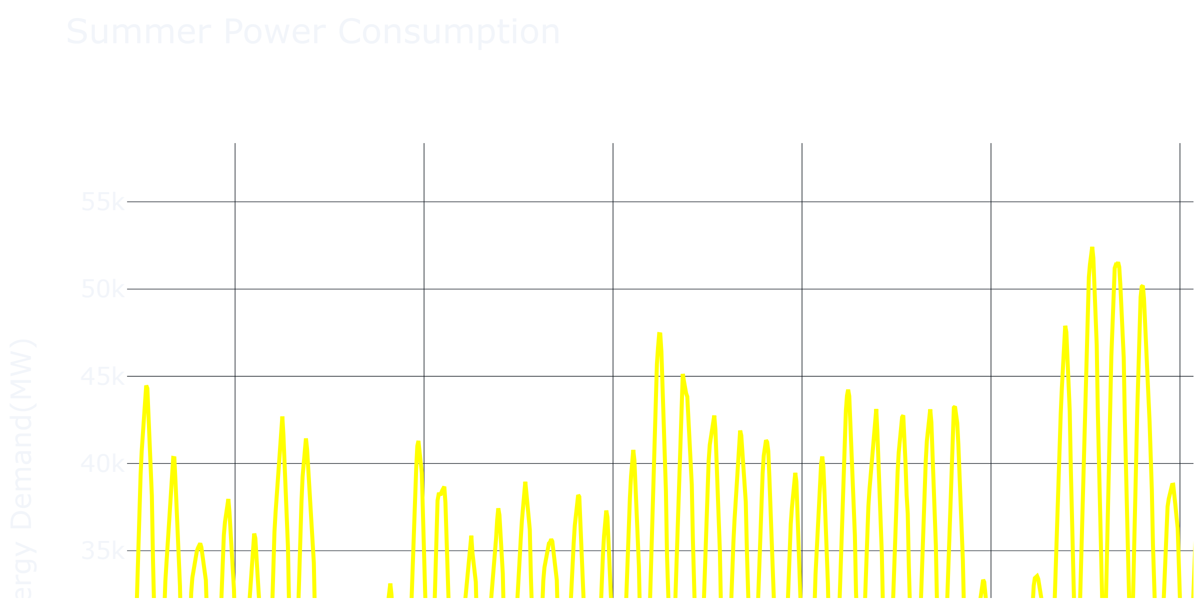
Summer vs Winter Demand

In [29]:
```
plt_df = df.loc[(df.index >= '2015-11-01') & (df.index < '2016-01-01')]
fig = go.Figure([go.Scatter(x=plt_df.index, y=plt_df.PJME_MW,line_color='lightblu
fig.update_layout(title_text='Winter Power Consumption',template="plotly_dark")
fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Energy Demand(MW)')
fig.show()
```



- we notice dips in the energy consumption mid day.
- In winter months people tend to use less energy mid-day.

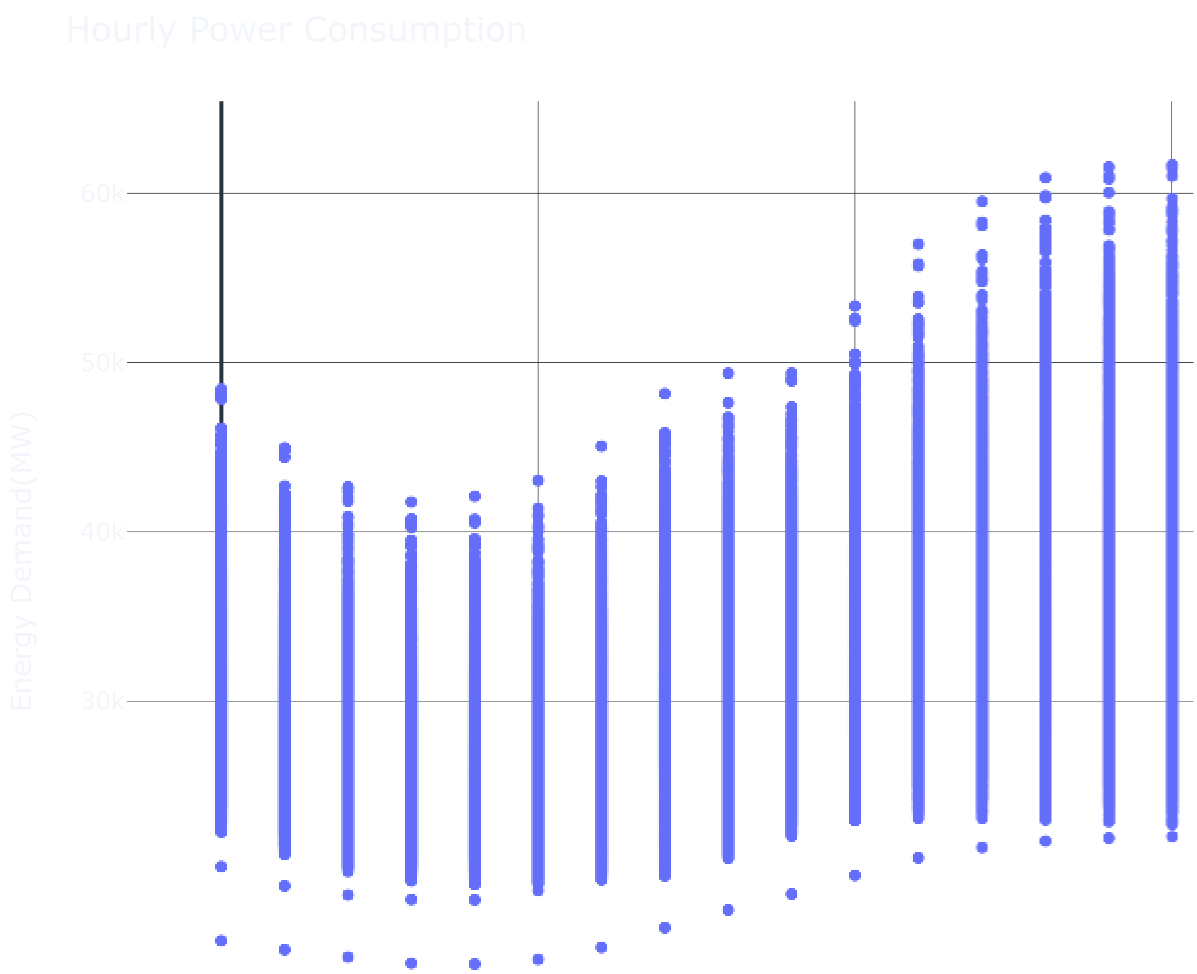
In [30]:
```
plt_df = df.loc[(df.index >= '2016-06-01') & (df.index < '2016-08-01')]
fig = go.Figure([go.Scatter(x=plt_df.index, y=plt_df.PJME_MW,line_color='yellow'
fig.update_layout(title_text='Summer Power Consumption',template="plotly_dark")
fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Energy Demand(MW)')
fig.show()
```



- we notice bell shaped curves all over
- more energy is consumed mid-day.... this maybe due to the use of air conditioners in the summer

Hourly Trend

In [31]:
```
fig = px.scatter(df, x="hour", y="PJME_MW")
fig.update_layout(title_text='Hourly Power Consumption',template="plotly_dark")
fig.update_xaxes(title_text='Hour')
fig.update_yaxes(title_text='Energy Demand(MW)')
fig.show()
```



Pandas Profiling

- This library helps us to quickly get an overview of the data
- lets try to use it and see what we can infer
- To install run pip install pandas_profiling

In [32]:
```
#pip install pandas_profiling
```

In [33]:
```
import pandas_profiling
pandas_profiling.ProfileReport(df)
```

Overview

Dataset info

Number of variables	13
Number of observations	145392
Missing cells	0 (0.0%)
Duplicate rows	0 (0.0%)
Total size in memory	14.4 MiB
Average record size in memory	104.0 B

Variables types

Numeric	6
Categorical	3
Boolean	0
Date	1
URL	0
Text (Unique)	0
Rejected	3
Unsupported	0

Warnings

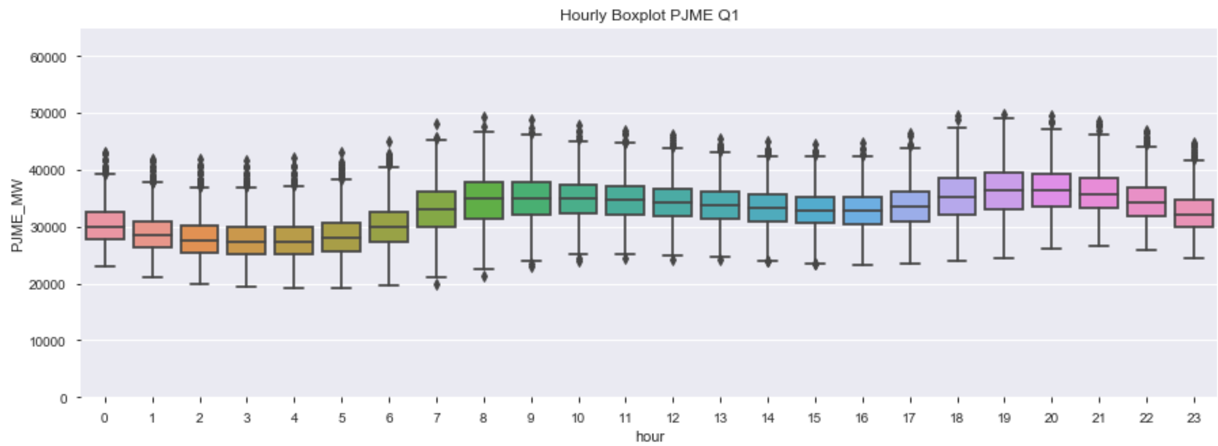
`date` only contains datetime values, but is categorical. Consider applying `pd.to_datetime()` **Type**

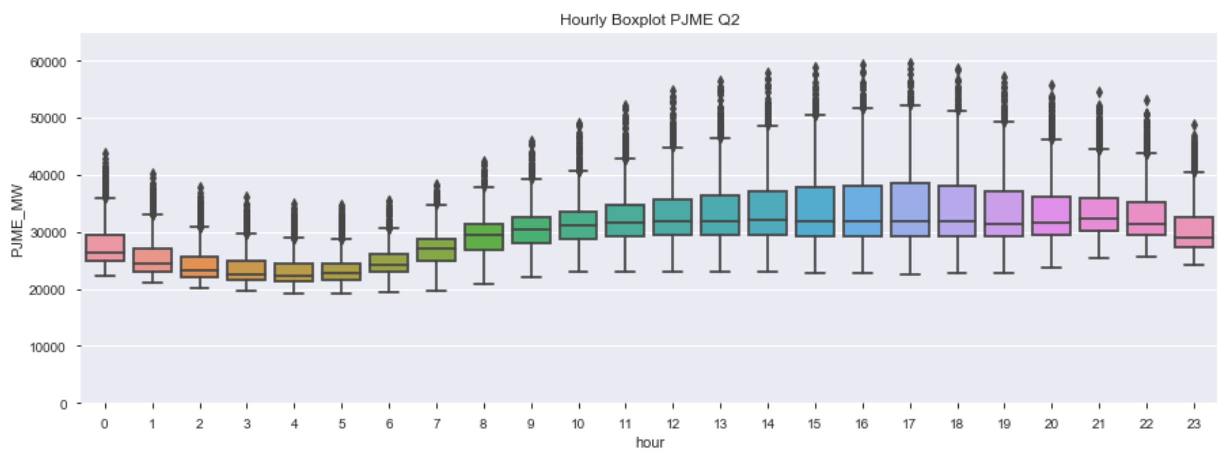
`date` has a high cardinality: 6059 distinct values **Warning**

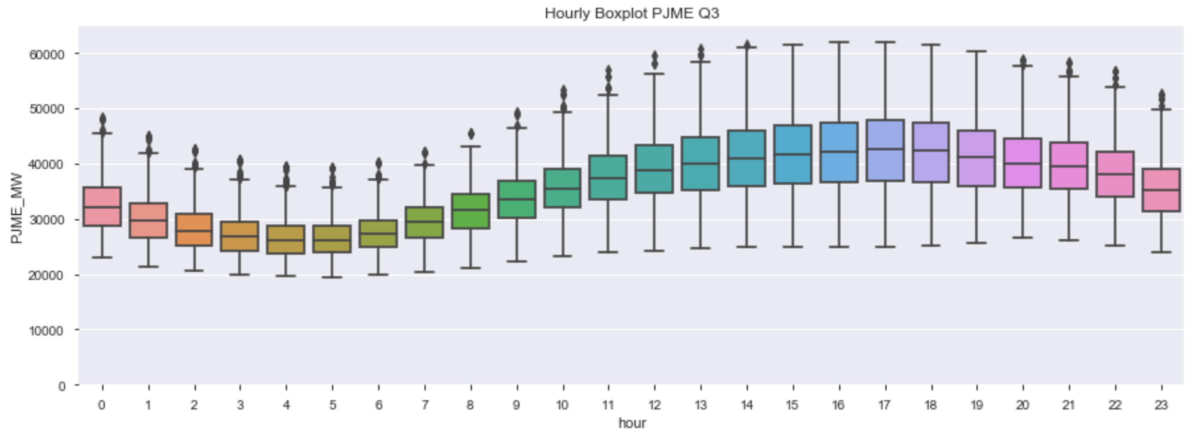
Out[33]:

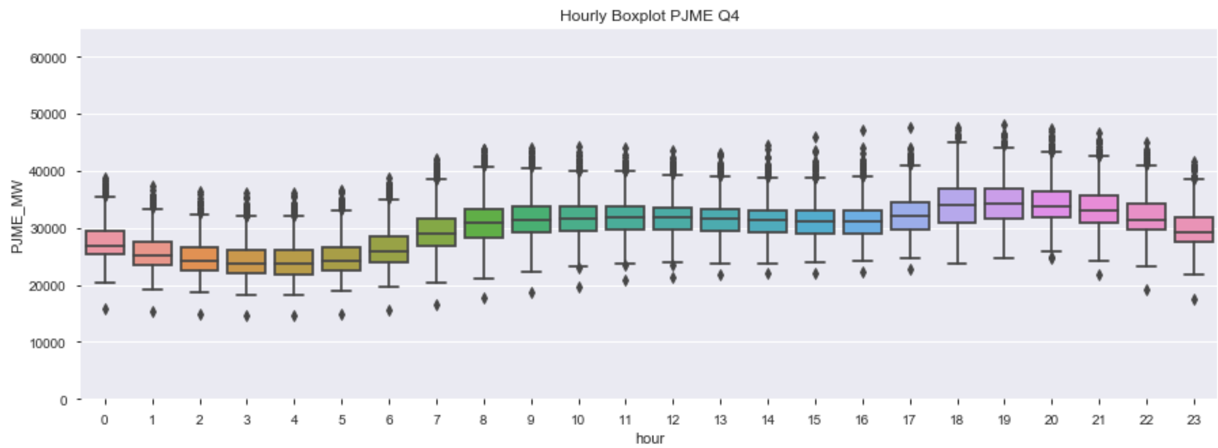
Quarterly Trends

In [34]:
```
fig, ax = plt.subplots(figsize=(15,5))
sns.boxplot(df.loc[df['quarter']==1].hour, df.loc[df['quarter']==1].PJME_MW)
ax.set_title('Hourly Boxplot PJME Q1')
ax.set_ylim(0,65000)
fig, ax = plt.subplots(figsize=(15,5))
sns.boxplot(df.loc[df['quarter']==2].hour, df.loc[df['quarter']==2].PJME_MW)
ax.set_title('Hourly Boxplot PJME Q2')
ax.set_ylim(0,65000)
fig, ax = plt.subplots(figsize=(15,5))
sns.boxplot(df.loc[df['quarter']==3].hour, df.loc[df['quarter']==3].PJME_MW)
ax.set_title('Hourly Boxplot PJME Q3')
ax.set_ylim(0,65000)
fig, ax = plt.subplots(figsize=(15,5))
sns.boxplot(df.loc[df['quarter']==4].hour, df.loc[df['quarter']==4].PJME_MW)
ax.set_title('Hourly Boxplot PJME Q4')
_ = ax.set_ylim(0,65000)
```





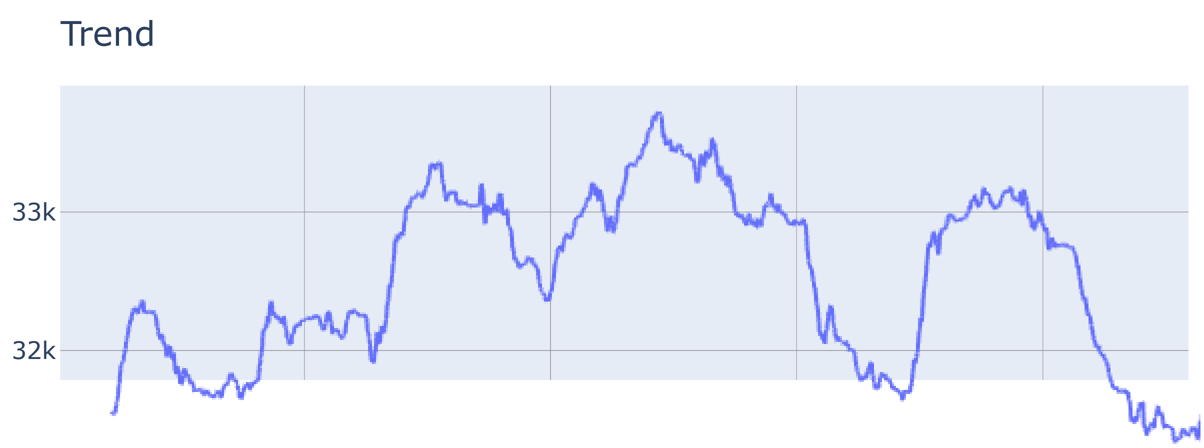


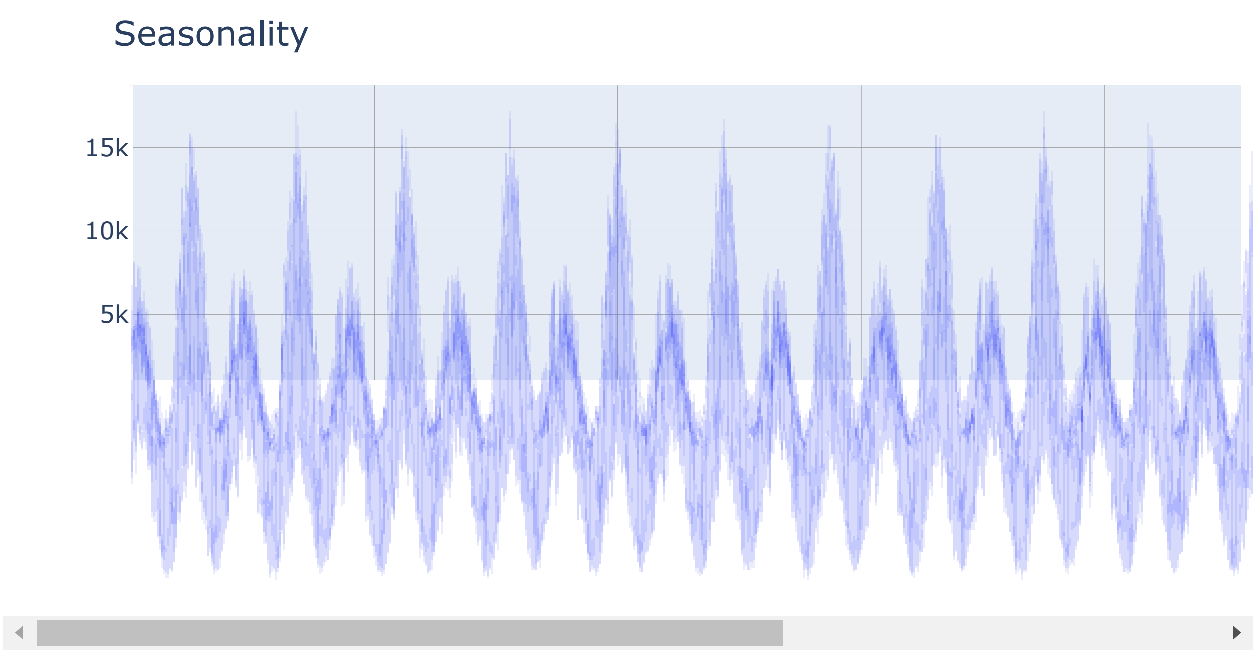


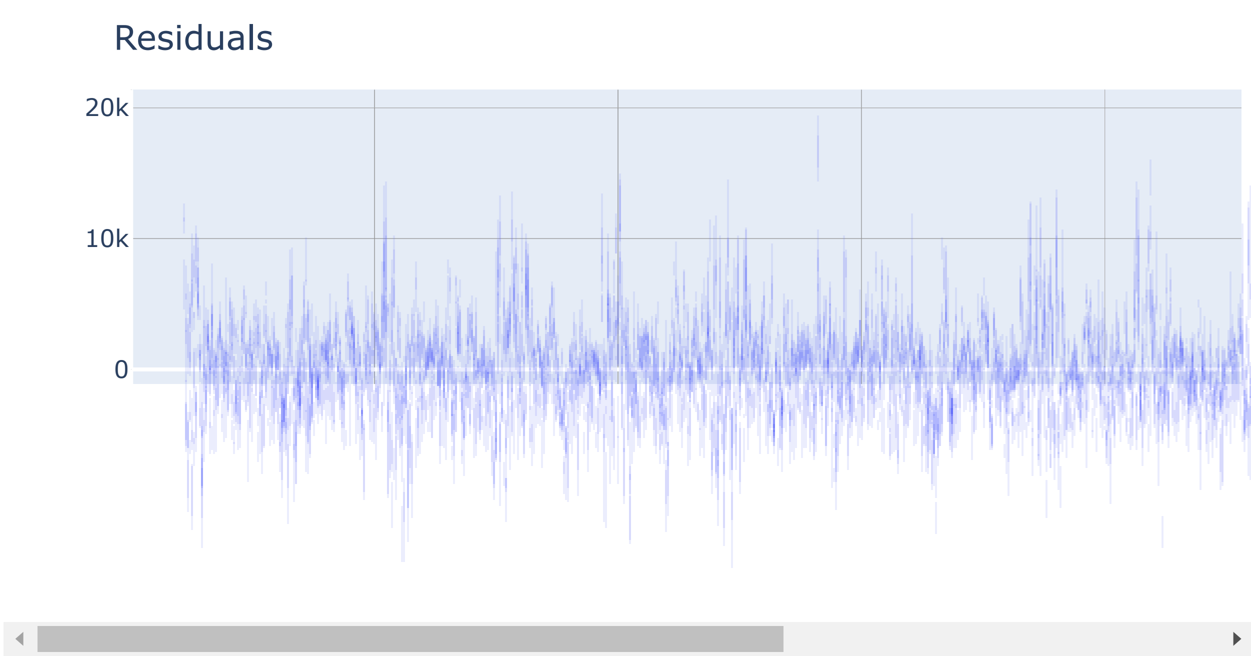
Time Series Decomposition

In [35]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# seasonal_decompose needs a dataframe with a datetime index
series = df.PJME_MW
frequency = 24*365

# decomposing the time-series, with the frequency being 24 hours per 365 days
decomposed = seasonal_decompose(series, model='additive', freq=frequency)
```

In [36]:
```python
# plotting the different elements constituting our time-series
def plot_decompositions(decompositions, titles, line_widths):
    for d, t, lw in zip(decompositions, titles, line_widths):

        # draw a line plot of the data
        fig = px.line(d,
              y='PJME_MW',
              title=t,
              height=300)

        # adjust line width
        fig.update_traces(line=dict(width=lw))

        # change layout of axes and the figure's margins
        # to emulate tight_layout
        fig.update_layout(
            xaxis=dict(
                showticklabels=False,
                linewidth=1
            ),
            yaxis=dict(title=''),
            margin=go.layout.Margin(
                l=40, r=40, b=0, t=40, pad=0
            ),
        )

        # display
        fig.show()

# calling the function
plot_decompositions(decompositions=[decomposed.trend,
                                    decomposed.seasonal,
                                    decomposed.resid],
                    titles=['Trend',
                            'Seasonality',
                            'Residuals'],
                    line_widths=[2, 0.025, 0.05])
```







Feature selection

In [37]:
```
from sklearn.feature_selection import SelectKBest, f_regression
X = df[['dow', 'doy', 'year', 'month', 'quarter', 'hour',
        'woy', 'dom', 'season']]
y = df[['PJME_MW']]
```

In [38]:
```
selector = SelectKBest(f_regression, k=2)
selector.fit(X, y)
```

Out[38]:
```
SelectKBest(k=2, score_func=<function f_regression at 0x0000027045736598>)
```

In [39]:
```
print(X.columns[selector.get_support()])
```

```
Index(['dow', 'hour'], dtype='object')
```

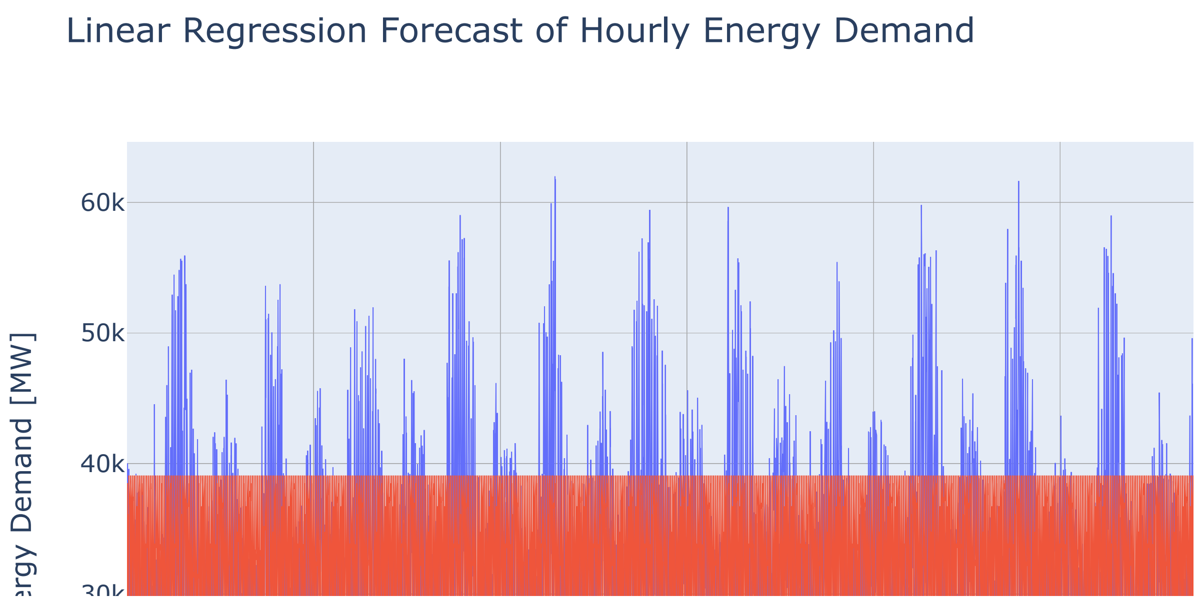
Prediction using Linear Regression

In [40]:
```
column_names = ['hour','dow'] # selecting hour and dow as our input features
X = df[column_names]
y = df['PJME_MW']
```

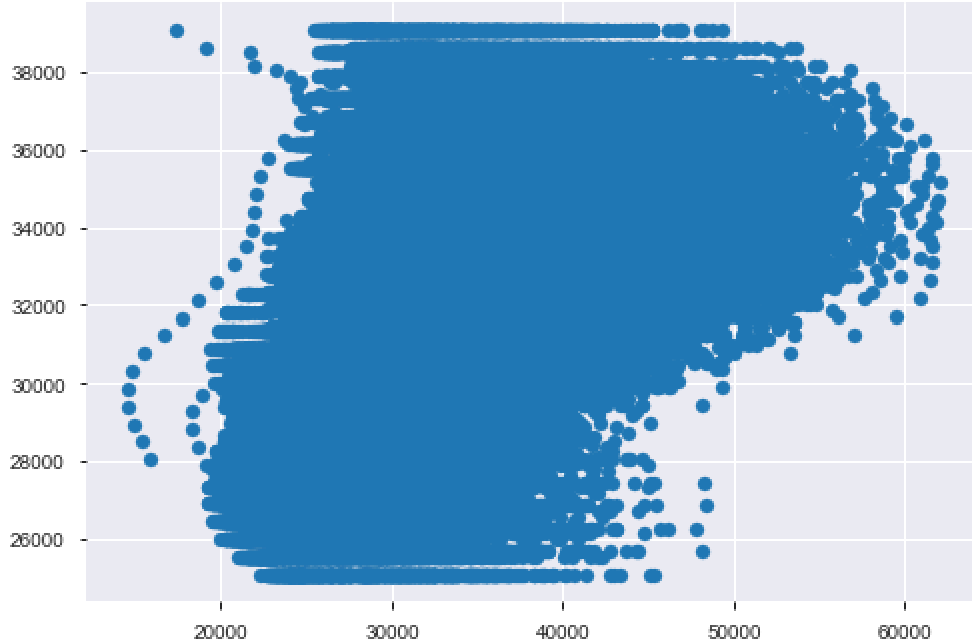
using entire dataset to predict

In [41]:
```
model = LinearRegression()
model.fit(X, y)
df['predicted'] = model.predict(X)
```

In [42]:
```
# create figure
fig = go.Figure()
fig.add_trace(go.Scatter(x=df.index, y=df.PJME_MW,
                         mode='lines',
                         name='Actual'))
fig.add_trace(go.Scatter(x=df.index, y=df.predicted,
                         mode='lines',
                         name='Predicted'))

# adjust layout
fig.update_traces(line=dict(width=0.5))
fig.update_layout(title='Linear Regression Forecast of Hourly Energy Demand',
                  xaxis_title='Date & Time (yyyy/mm/dd hh:MM)',
                  yaxis_title='Energy Demand [MW]')
```



In [43]:
```
plt.scatter(df.PJME_MW, df.predicted);
```



In [44]:
```
from sklearn import metrics
np.sqrt(metrics.mean_squared_error(y,df.predicted))
```

Out[44]: 5522.286462980462

In [45]:
```
df = df.drop(columns = 'predicted')
```

- looks like linear regression wont work in our case

lets try Holt-winter model for forecasting

In [46]:
```
df.head()
```

Out[46]:

	PJME_MW	dow	doy	year	month	quarter	hour	weekday	woy	dom	date	season
2002-01-01 01:00:00	30393.0	1	1	2002	1	1	1	Tuesday	1	1	2002-01-01	1
2002-01-01 02:00:00	29265.0	1	1	2002	1	1	2	Tuesday	1	1	2002-01-01	1
2002-01-01 03:00:00	28357.0	1	1	2002	1	1	3	Tuesday	1	1	2002-01-01	1
2002-01-01 04:00:00	27899.0	1	1	2002	1	1	4	Tuesday	1	1	2002-01-01	1
2002-01-01 05:00:00	28057.0	1	1	2002	1	1	5	Tuesday	1	1	2002-01-01	1

In [47]:
```
# set manually
CUTOFF_DATE = pd.to_datetime('2017-08-01')
TIME_DELTA = pd.DateOffset(years=8)

# splitting
train = df.loc[(df.index < CUTOFF_DATE) & (df.index >= CUTOFF_DATE-TIME_DELTA) ]
test = df.loc[df.index >= CUTOFF_DATE].copy()
```

In [48]:
```
train.head()
```

Out[48]:

	PJME_MW	dow	doy	year	month	quarter	hour	weekday	woy	dom	date	season
2009-08-01 00:00:00	32182.0	5	213	2009	8	3	0	Saturday	31	1	2009-08-01	3
2009-08-01 01:00:00	29524.0	5	213	2009	8	3	1	Saturday	31	1	2009-08-01	3
2009-08-01 02:00:00	27739.0	5	213	2009	8	3	2	Saturday	31	1	2009-08-01	3
2009-08-01 03:00:00	26386.0	5	213	2009	8	3	3	Saturday	31	1	2009-08-01	3
2009-08-01 04:00:00	25549.0	5	213	2009	8	3	4	Saturday	31	1	2009-08-01	3

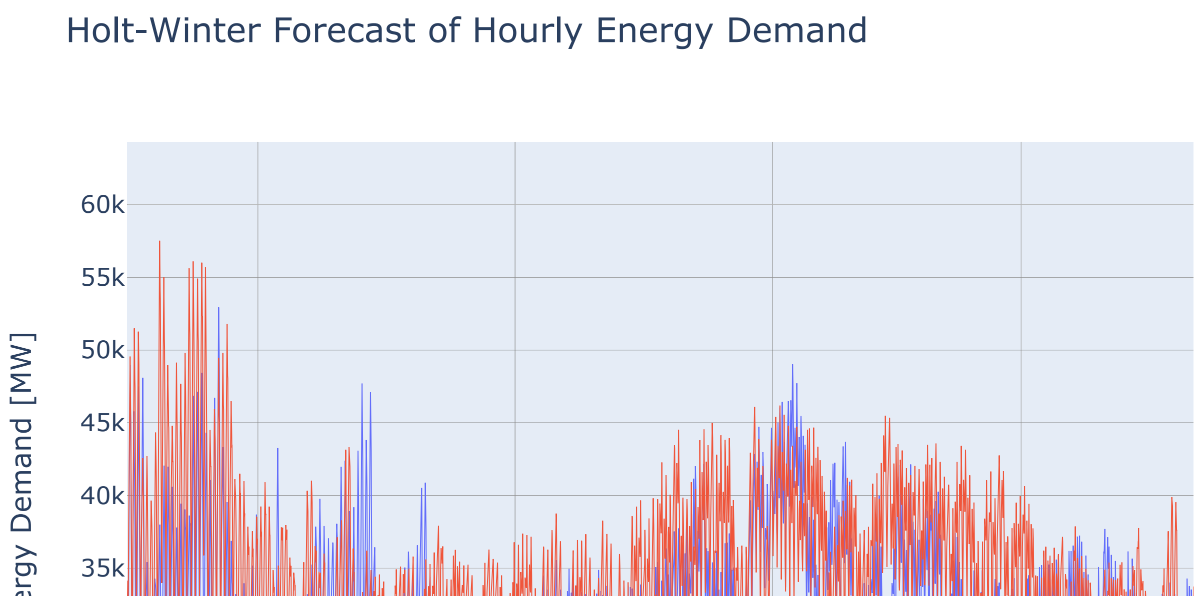
In [49]:
```
test.head()
```

Out[49]:

	PJME_MW	dow	doy	year	month	quarter	hour	weekday	woy	dom	date	season
2017-08-01 00:00:00	33342.0	1	213	2017	8	3	0	Tuesday	31	1	2017-08-01	3
2017-08-01 01:00:00	30396.0	1	213	2017	8	3	1	Tuesday	31	1	2017-08-01	3
2017-08-01 02:00:00	28443.0	1	213	2017	8	3	2	Tuesday	31	1	2017-08-01	3
2017-08-01 03:00:00	27128.0	1	213	2017	8	3	3	Tuesday	31	1	2017-08-01	3
2017-08-01 04:00:00	26409.0	1	213	2017	8	3	4	Tuesday	31	1	2017-08-01	3

In [50]:
```
import statsmodels.api as sm
exp_smooth_train, exp_smooth_test = train['PJME_MW'], test['PJME_MW']
# fit & predict
holt_winter = sm.tsa.ExponentialSmoothing(exp_smooth_train,
                                          seasonal_periods=24*365,
                                          seasonal='add').fit()
y_hat_holt_winter = holt_winter.forecast(len(exp_smooth_test))
```

In [51]:
```python
# create figure
fig = go.Figure()
fig.add_trace(go.Scatter(x=exp_smooth_test.index, y=exp_smooth_test,
                         mode='lines',
                         name='Actual'))
fig.add_trace(go.Scatter(x=y_hat_holt_winter.index, y=y_hat_holt_winter,
                         mode='lines',
                         name='Predicted'))

# adjust layout
fig.update_traces(line=dict(width=0.5))
fig.update_layout(title='Holt-Winter Forecast of Hourly Energy Demand',
                  xaxis_title='Date & Time',
                  yaxis_title='Energy Demand [MW]')
```



Classification

About Dataset

This dataset contains daily weather observations from numerous Australian weather stations.

The target variable RainTomorrow means: Did it rain the next day? Yes or No.

Note: You should exclude the variable Risk-MM when training a binary classification model. Not excluding it will leak the answers to your model and reduce its predictability.

In [52]:
```
df = pd.read_csv('E:/Data Science with Python/Project/weather-dataset-rattle-pacl
```

In [53]:
```
df = df.drop(columns = 'RISK_MM') #the dataset description asked me to do so
```

In [54]:
```
df.head()
```

Out[54]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustS
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	

5 rows × 23 columns

In [55]:
```
df.describe().T
```

Out[55]:

	count	mean	std	min	25%	50%	75%	max
MinTemp	141556.0	12.186400	6.403283	-8.5	7.6	12.0	16.8	33.9
MaxTemp	141871.0	23.226784	7.117618	-4.8	17.9	22.6	28.2	48.1
Rainfall	140787.0	2.349974	8.465173	0.0	0.0	0.0	0.8	371.0
Evaporation	81350.0	5.469824	4.188537	0.0	2.6	4.8	7.4	145.0
Sunshine	74377.0	7.624853	3.781525	0.0	4.9	8.5	10.6	14.5
WindGustSpeed	132923.0	39.984292	13.588801	6.0	31.0	39.0	48.0	135.0
WindSpeed9am	140845.0	14.001988	8.893337	0.0	7.0	13.0	19.0	130.0
WindSpeed3pm	139563.0	18.637576	8.803345	0.0	13.0	19.0	24.0	87.0
Humidity9am	140419.0	68.843810	19.051293	0.0	57.0	70.0	83.0	100.0
Humidity3pm	138583.0	51.482606	20.797772	0.0	37.0	52.0	66.0	100.0
Pressure9am	128179.0	1017.653758	7.105476	980.5	1012.9	1017.6	1022.4	1041.0
Pressure3pm	128212.0	1015.258204	7.036677	977.1	1010.4	1015.2	1020.0	1039.6
Cloud9am	88536.0	4.437189	2.887016	0.0	1.0	5.0	7.0	9.0
Cloud3pm	85099.0	4.503167	2.720633	0.0	2.0	5.0	7.0	9.0
Temp9am	141289.0	16.987509	6.492838	-7.2	12.3	16.7	21.6	40.2
Temp3pm	139467.0	21.687235	6.937594	-5.4	16.6	21.1	26.4	46.7

In [56]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 23 columns):
Date             142193 non-null object
Location         142193 non-null object
MinTemp          141556 non-null float64
MaxTemp          141871 non-null float64
Rainfall         140787 non-null float64
Evaporation      81350 non-null float64
Sunshine         74377 non-null float64
WindGustDir      132863 non-null object
WindGustSpeed    132923 non-null float64
WindDir9am       132180 non-null object
WindDir3pm       138415 non-null object
WindSpeed9am     140845 non-null float64
WindSpeed3pm     139563 non-null float64
Humidity9am      140419 non-null float64
Humidity3pm      138583 non-null float64
Pressure9am      128179 non-null float64
Pressure3pm      128212 non-null float64
Cloud9am         88536 non-null float64
Cloud3pm         85099 non-null float64
Temp9am          141289 non-null float64
Temp3pm          139467 non-null float64
RainToday        140787 non-null object
RainTomorrow     142193 non-null object
dtypes: float64(16), object(7)
memory usage: 25.0+ MB
```

dealing with null values

In [57]:
```
(df.isnull().sum()/df.count())*100 #calculating the percentage of null values
```

Out[57]:
```
Date              0.000000
Location          0.000000
MinTemp           0.449999
MaxTemp           0.226967
Rainfall          0.998672
Evaporation      74.791641
Sunshine         91.178725
WindGustDir       7.022271
WindGustSpeed     6.973962
WindDir9am        7.575276
WindDir3pm        2.729473
WindSpeed9am      0.957080
WindSpeed3pm      1.884454
Humidity9am       1.263362
Humidity3pm       2.604937
Pressure9am      10.933148
Pressure3pm      10.904596
Cloud9am         60.604726
Cloud3pm         67.091270
Temp9am           0.639823
Temp3pm           1.954584
RainToday         0.998672
RainTomorrow      0.000000
dtype: float64
```

- We have 4 columns where more than 60% of the data is missing....hence i am going to drop those columns

In [58]:
```
df = df.drop(columns=['Sunshine','Evaporation','Cloud3pm','Cloud9am'],axis=1)
```

In [59]:
```
df = df.dropna(how='any') #dropping all null values since their percentage is ve
```

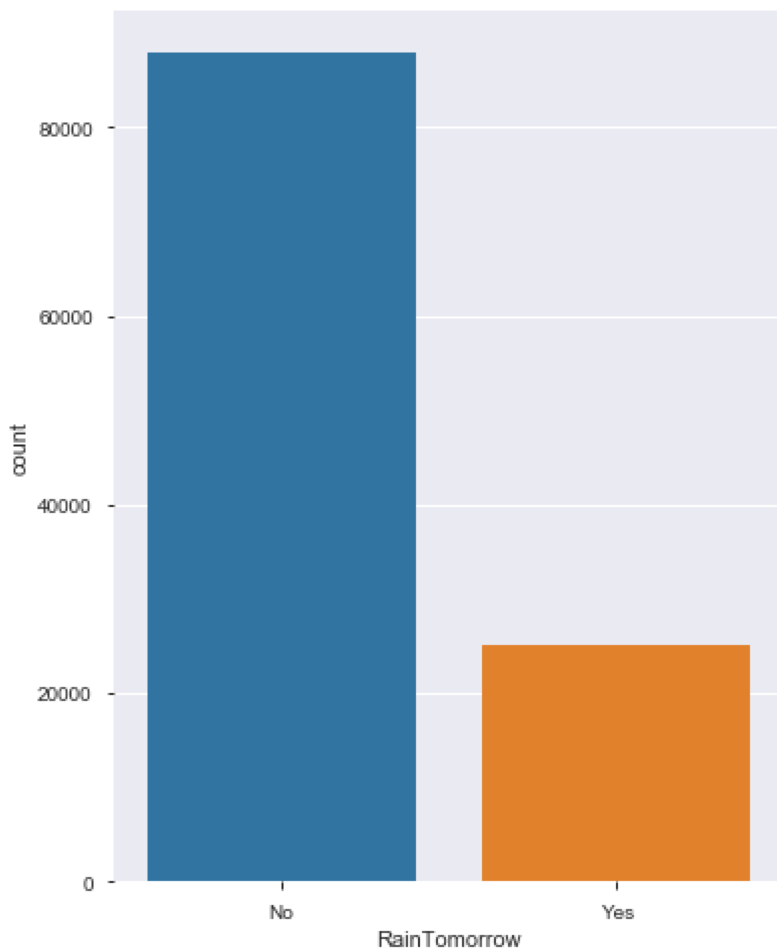
In [60]:
```
df.shape
```

Out[60]:
```
(112925, 19)
```

In [61]:
```
df.isnull().sum() #checking if there are anymore null values
```

Out[61]:
```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

In [62]:
```
f, ax = plt.subplots(figsize=(6, 8))
ax = sns.countplot(x="RainTomorrow", data=df)
plt.show()
```



using pandas profiling

In [63]:
```
pandas_profiling.ProfileReport(df)
```

Overview

Dataset info

Number of variables	20
Number of observations	112925
Missing cells	0 (0.0%)
Duplicate rows	0 (0.0%)
Total size in memory	17.2 MiB
Average record size in memory	160.0 B

Variables types

Numeric	11
Categorical	5
Boolean	2
Date	0
URL	0
Text (Unique)	0
Rejected	2
Unsupported	0

Warnings

`Date` only contains datetime values, but is categorical. Consider applying `pd.to_datetime()` **Type**

`Date` has a high cardinality: 3417 distinct values **Warning**

Out[63]:

creating features from the date variable

In [64]:
```
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
```

In [65]:
```
df.drop('Date', axis=1, inplace = True) #Dropping the original date
```

In [66]:
```
df.head()
```

Out[66]:

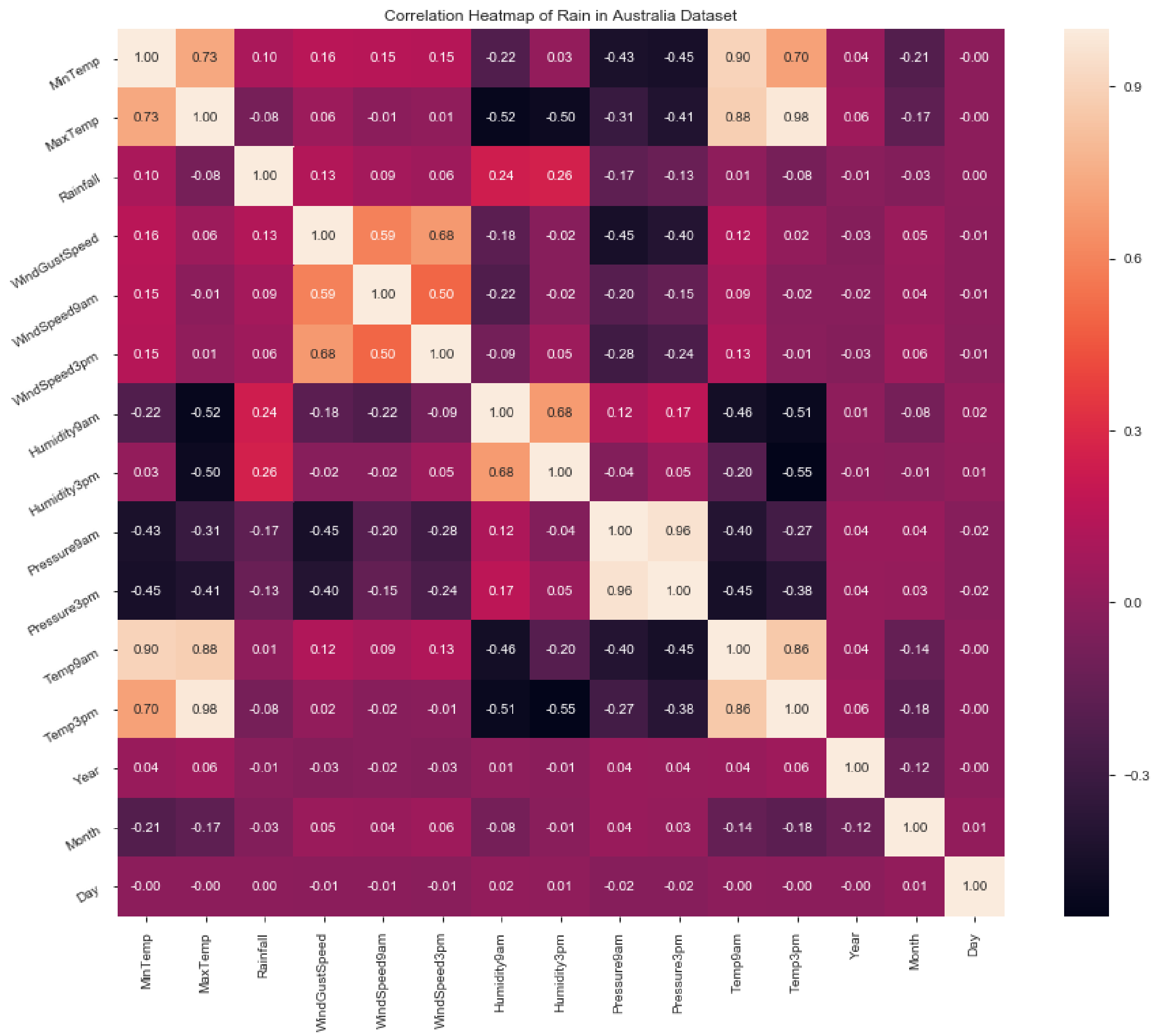
	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm
0	Albury	13.4	22.9	0.6	W	44.0	W	WNW
1	Albury	7.4	25.1	0.0	WNW	44.0	NNW	WSW
2	Albury	12.9	25.7	0.0	WSW	46.0	W	WSW
3	Albury	9.2	28.0	0.0	NE	24.0	SE	E
4	Albury	17.5	32.3	1.0	W	41.0	ENE	NW

5 rows × 21 columns

Correlation Plot

In [67]:
```
correlation = df.corr()
```

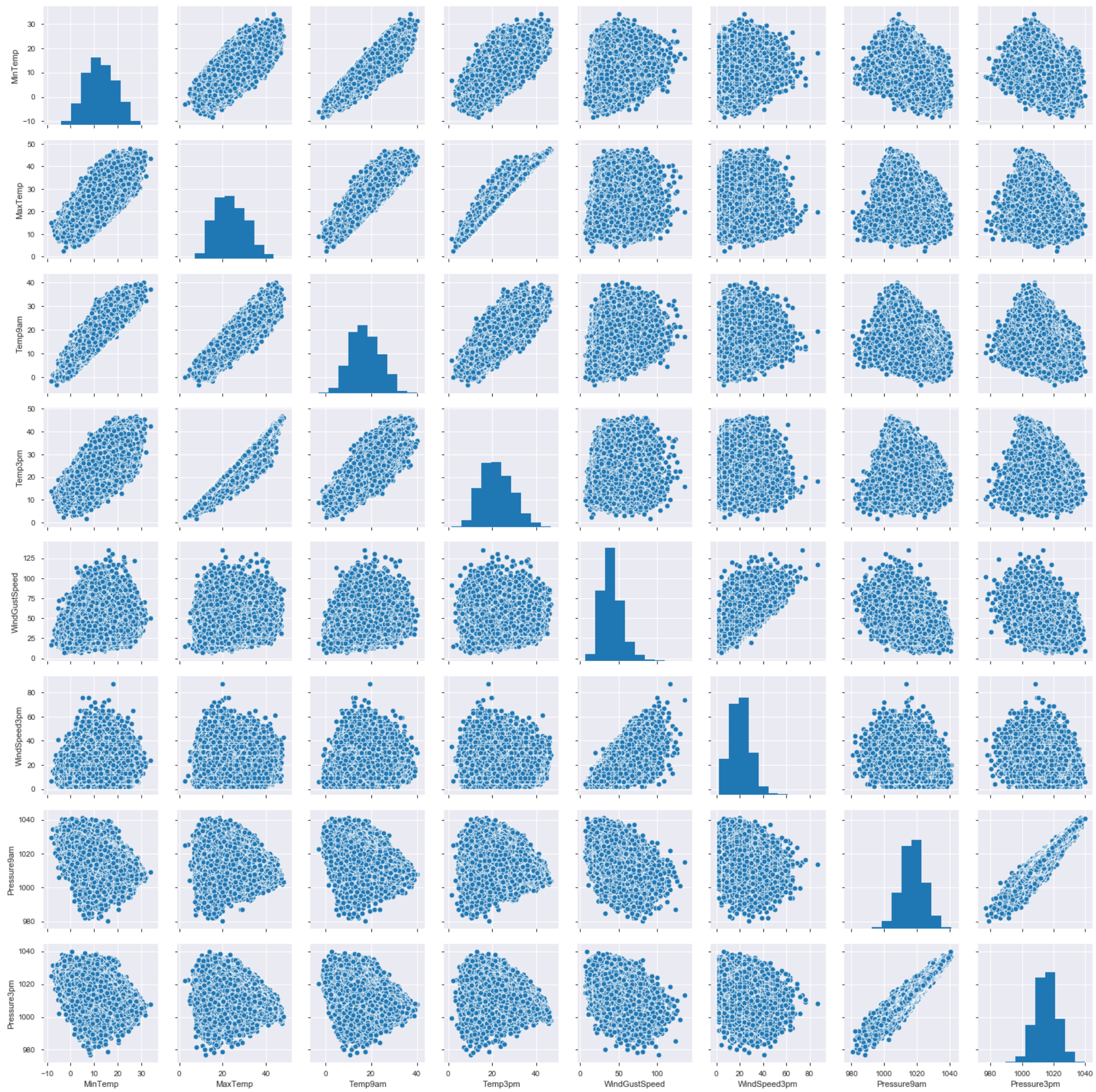
In [68]:
```
plt.figure(figsize=(16,12))
plt.title('Correlation Heatmap of Rain in Australia Dataset')
ax = sns.heatmap(correlation, square=True, annot=True, fmt='.2f', linecolor='whi
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
ax.set_yticklabels(ax.get_yticklabels(), rotation=30)
plt.show()
```



- MinTemp and MaxTemp are highly correlated
- MinTemp and Temp9am are highly correlated
- MinTemp and Temp3pm are highly correlated
- MaxTemp and Temppam are highly correlated
- MaxTemp and Temp3pm are highly correlated
- WindGustSpeed and WindSpeed3pm are highly correlated
- Pressure9am and Pressure3pm are highly correlated
- Temp9am and Temp3pm are highly correlated

Lets see a pairplot to know more about the correlated variables

In [69]:
```
var = ['MinTemp', 'MaxTemp', 'Temp9am', 'Temp3pm', 'WindGustSpeed', 'WindSpeed3pm
sns.pairplot(df[var], kind='scatter', diag_kind='hist', palette='Rainbow')
plt.show()
```



Label Encoding the categorical data

In [70]:
```
categorical = [var for var in df.columns if df[var].dtype=='O']
```

In [71]:
```
categorical
```

Out[71]:
```
['Location',
 'WindGustDir',
 'WindDir9am',
 'WindDir3pm',
 'RainToday',
 'RainTomorrow']
```

In [72]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df[categorical]=df[categorical].apply(le.fit_transform)
```

In [73]:
```
df.head()
```

Out[73]:

	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm
0	1	13.4	22.9	0.6	13	44.0	13	14
1	1	7.4	25.1	0.0	14	44.0	6	15
2	1	12.9	25.7	0.0	15	46.0	13	15
3	1	9.2	28.0	0.0	4	24.0	9	0
4	1	17.5	32.3	1.0	13	41.0	1	7

5 rows × 21 columns

In [74]:
```
df[categorical].head()
```

Out[74]:

	Location	WindGustDir	WindDir9am	WindDir3pm	RainToday	RainTomorrow
0	1	13	13	14	0	0
1	1	14	6	15	0	0
2	1	15	13	15	0	0
3	1	4	9	0	0	0
4	1	13	1	7	0	0

Perform PCA to determine no of features

In [75]:
```
from sklearn.decomposition import PCA
X = df.drop(columns='RainTomorrow')
y = df.RainTomorrow
```

In [76]:
```
pca = PCA(n_components=2)

X_r = pca.fit(X).transform(X)

X_r.shape
```

Out[76]:
```
(112925, 2)
```

In [77]:
```
X_r[0:5]
```

Out[77]:
```
array([[ 19.40581282,  -1.17630785],
       [ 34.39886721,  -4.0504934 ],
       [ 35.97590611,   7.72260151],
       [ 39.3513555 , -26.31717343],
       [  7.15350488,  -7.22581408]])
```

In [78]:
```
pca.inverse_transform(X_r)[0:5]
```

Out[78]:
```
array([[ 2.07938251e+01,  1.33851240e+01,  2.68187320e+01,
         5.27205317e-01,  7.39406981e+00,  4.14752913e+01,
         6.56305100e+00,  7.64860094e+00,  1.57235474e+01,
         1.94263745e+01,  5.51879867e+01,  3.65809822e+01,
         1.01716206e+03,  1.01436423e+03,  1.93884176e+01,
         2.53531131e+01,  9.37700145e-02,  2.01276868e+03,
         6.49573575e+00,  1.56079147e+01],
       [ 2.01371640e+01,  1.36792054e+01,  2.91895144e+01,
        -1.13827043e+00,  7.07767848e+00,  4.06315966e+01,
         6.01054386e+00,  7.50695282e+00,  1.55124585e+01,
         1.86346897e+01,  4.60279143e+01,  2.51527748e+01,
         1.01743829e+03,  1.01426157e+03,  2.06684977e+01,
         2.78177751e+01, -1.89100744e-02,  2.01278818e+03,
         6.53494202e+00,  1.55274925e+01],
       [ 2.16214723e+01,  1.53230099e+01,  2.98875258e+01,
         1.34242886e-01,  7.43804813e+00,  4.89957662e+01,
         6.14493896e+00,  7.83602677e+00,  1.93713927e+01,
         2.30635352e+01,  4.33549853e+01,  2.72910929e+01,
         1.01451796e+03,  1.01163112e+03,  2.20828180e+01,
         2.82467031e+01,  4.02554036e-02,  2.01274307e+03,
         6.60310985e+00,  1.55063532e+01],
       [ 1.71780056e+01,  1.09290699e+01,  2.91806209e+01,
        -4.24409199e+00,  6.27824161e+00,  2.54305938e+01,
         5.48819299e+00,  6.85293223e+00,  8.59118215e+00,
         1.04077320e+01,  4.60199051e+01,  1.54808797e+01,
         1.02273623e+03,  1.01885224e+03,  1.88322625e+01,
         2.83323493e+01, -1.81501209e-01,  2.01287787e+03,
         6.43483292e+00,  1.55233124e+01],
       [ 2.02500938e+01,  1.20230707e+01,  2.45693071e+01,
         8.81377515e-01,  7.37884680e+00,  3.62858710e+01,
         6.88067090e+00,  7.52512696e+00,  1.31967705e+01,
         1.69352211e+01,  6.38624736e+01,  4.35969051e+01,
         1.01898758e+03,  1.01627005e+03,  1.74523405e+01,
         2.32209813e+01,  1.36460220e-01,  2.01278555e+03,
         6.41915475e+00,  1.56824544e+01]])
```

In [79]:
```
print(pca.explained_variance_ratio_)
```

```
[0.40449216 0.16439351]
```

Feature Scaling

In [80]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df)
scaled_df = pd.DataFrame(scaler.transform(df), index=df.index, columns=df.column
scaled_df.head()
```

Out[80]:

	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pr
0	0.023256	0.513064	0.446154	0.001632	0.866667	0.289062	0.866667	0.93333
1	0.023256	0.370546	0.494505	0.000000	0.933333	0.289062	0.400000	1.00000
2	0.023256	0.501188	0.507692	0.000000	1.000000	0.304688	0.866667	1.00000
3	0.023256	0.413302	0.558242	0.000000	0.266667	0.132812	0.600000	0.00000
4	0.023256	0.610451	0.652747	0.002720	0.866667	0.265625	0.066667	0.46666

5 rows × 21 columns

Feature extraction

In [81]:
```
from sklearn.feature_selection import SelectKBest, chi2
X = scaled_df.loc[:,scaled_df.columns!='RainTomorrow']
y = scaled_df['RainTomorrow']
selector = SelectKBest(chi2, k=5)
selector.fit(X, y)
X_new = selector.transform(X)
print(X.columns[selector.get_support(indices=True)])
```

```
Index(['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday'],
dtype='object')
```

Apply PCA on these new features with scaled vs unscaled data

In [82]:
```
#scaled data
X = scaled_df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainT
y = scaled_df.RainTomorrow
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
print(pca.explained_variance_ratio_)
```

```
[0.72127177 0.19247338]
```

In [83]:
```
# unscaled data
X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df.RainTomorrow
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
print(pca.explained_variance_ratio_)
```

```
[0.6478445  0.18459905]
```

- we see that scaled data has higher variance than unscaled data and first 2 components contribute 91% of total variance

Logistic Regression

In [84]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df.RainTomorrow

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_s
```

In [85]:
```
clf = make_pipeline(MinMaxScaler(),
                        PCA(n_components=2),LogisticRegression())
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [86]:
```
from sklearn.metrics import accuracy_score
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.8222
```

In [87]:
```
result = []
result.append(('Logistic Regression',accuracy_score(y_test, y_pred)))
```

In [88]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

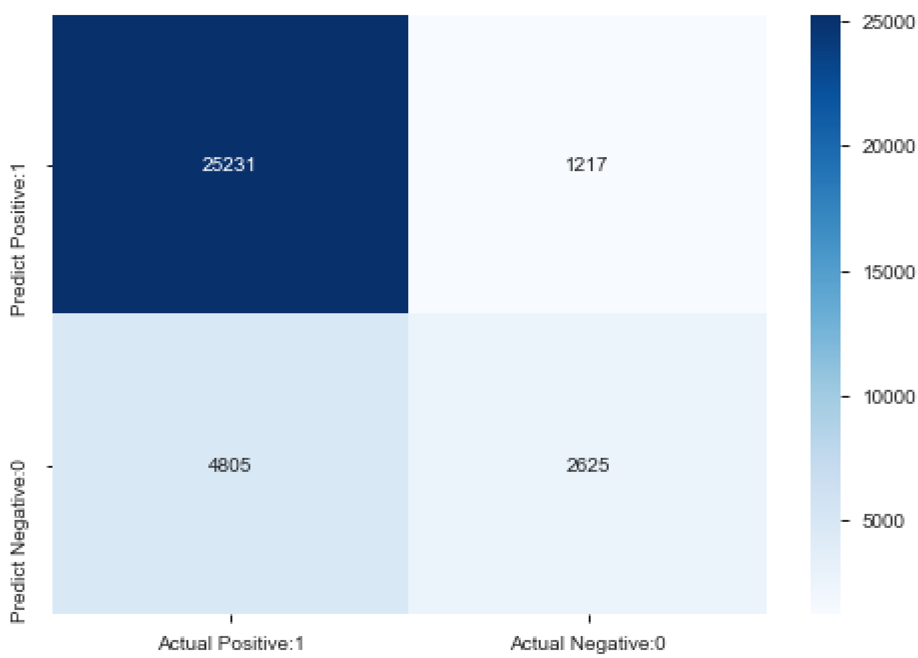
```
Training set score: 0.8150
Test set score: 0.8222
```

In [89]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
True Positives(TP) =  25231

True Negatives(TN) =  2625

False Positives(FP) =  1217

False Negatives(FN) =  4805
```

In [90]:
```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                 index=['Predict Positive:1', 'Predict Negative:(

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Blues')
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x27046cdd9b0>



Classification Analysis

Naive Bayes

In [91]:
```
# NaiveBayes Classifier
from sklearn.naive_bayes import GaussianNB
X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df.RainTomorrow
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_st
clf = make_pipeline(MinMaxScaler(),PCA(n_components=2), GaussianNB())
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [92]:
```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.8030
```

In [93]:
```
result.append(('Naive Bayes',accuracy_score(y_test, y_pred)))
```

In [94]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.8001
Test set score: 0.8030
```

In [95]:
```
cm = confusion_matrix(y_test, y_pred)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```

True Positives(TP) =  24081

True Negatives(TN) =  3123

False Positives(FP) =  2367

False Negatives(FN) =  4307
```

In [96]:
```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                 index=['Predict Positive:1', 'Predict Negative:(

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Reds')
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x27044c925f8>

Decision Tree

In [97]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [98]:
```
X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df.RainTomorrow
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_s
clf = make_pipeline(MinMaxScaler(),PCA(n_components=2), DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [99]:
```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.7729
```

In [100]:
```
result.append(('Decision Tree',accuracy_score(y_test, y_pred)))
```

In [101]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

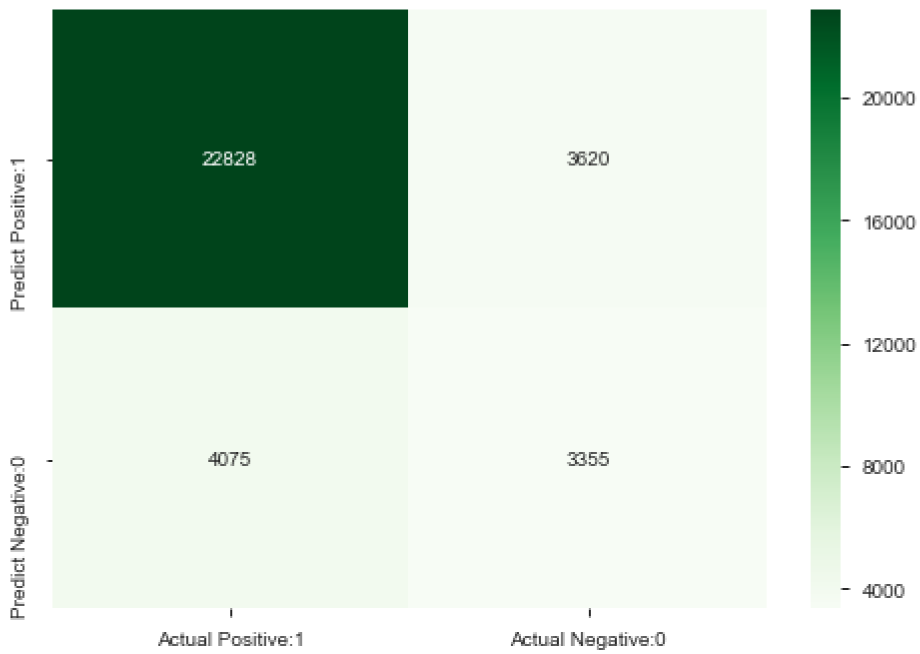
```
Training set score: 0.9749
Test set score: 0.7729
```

In [102]:
```
cm = confusion_matrix(y_test, y_pred)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```

True Positives(TP) =  22828

True Negatives(TN) =  3355

False Positives(FP) =  3620

False Negatives(FN) =  4075
```

In [103]:
```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                  index=['Predict Positive:1', 'Predict Negative:(

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Greens')
```

Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x27045531278>



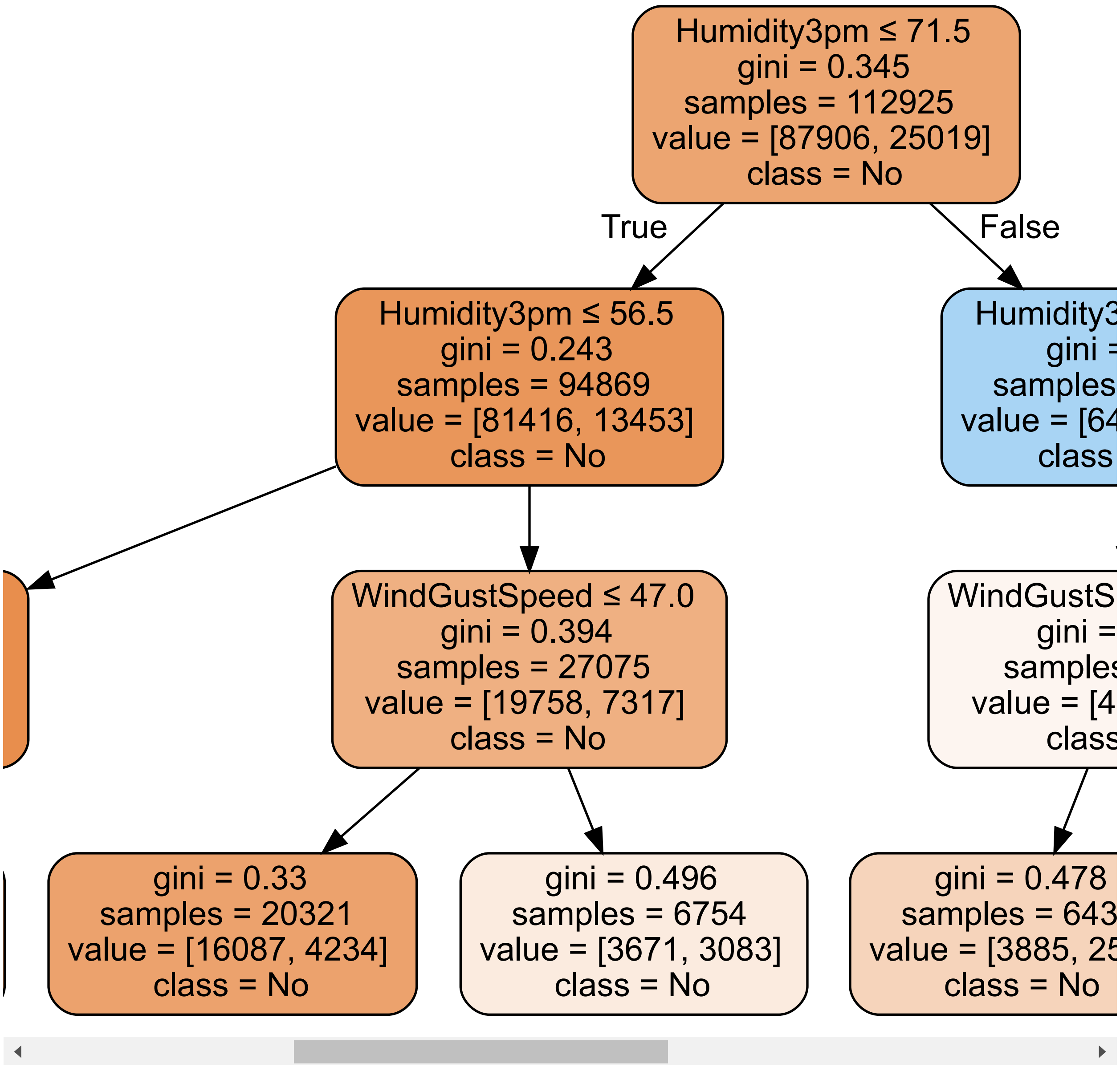
Visualizing Decision Tree

In [104]:
```
from sklearn.tree import export_graphviz, plot_tree
import graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin'
X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df[['RainTomorrow']]
clf1 = DecisionTreeClassifier(max_depth=3)
clf1 = clf1.fit(X, y)
```

In [105]:
```
df1 = pd.read_csv('E:/Data Science with Python/Project/weather-dataset-rattle-pa(
```

In [106]:
```
data = export_graphviz(clf1,out_file=None,
                       feature_names=X.columns,
                        class_names=df1.RainTomorrow.unique(),
                        filled=True, rounded=True,
                        special_characters=True)
graph = graphviz.Source(data)

graph
```

Out[106]:



SVM

In [107]:
```
from sklearn.svm import SVC
```

In [108]:
```
X = df[['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'RainToday']]
y = df.RainTomorrow
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_s
clf = make_pipeline(MinMaxScaler(),PCA(n_components=2), SVC())
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [109]:
```
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.8197
```

In [110]:
```
result.append(('SVM',accuracy_score(y_test, y_pred)))
```

In [111]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.8138
Test set score: 0.8197
```

In [112]:
```
cm = confusion_matrix(y_test, y_pred)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```

True Positives(TP) =  25650

True Negatives(TN) =  2119

False Positives(FP) =  798

False Negatives(FN) =  5311
```

In [113]:
```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                 index=['Predict Positive:1', 'Predict Negative:(

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='coolwarm')
```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x27044c965f8>

Clustering

In [114]:
```
scaled_df.head()
```

Out[114]:

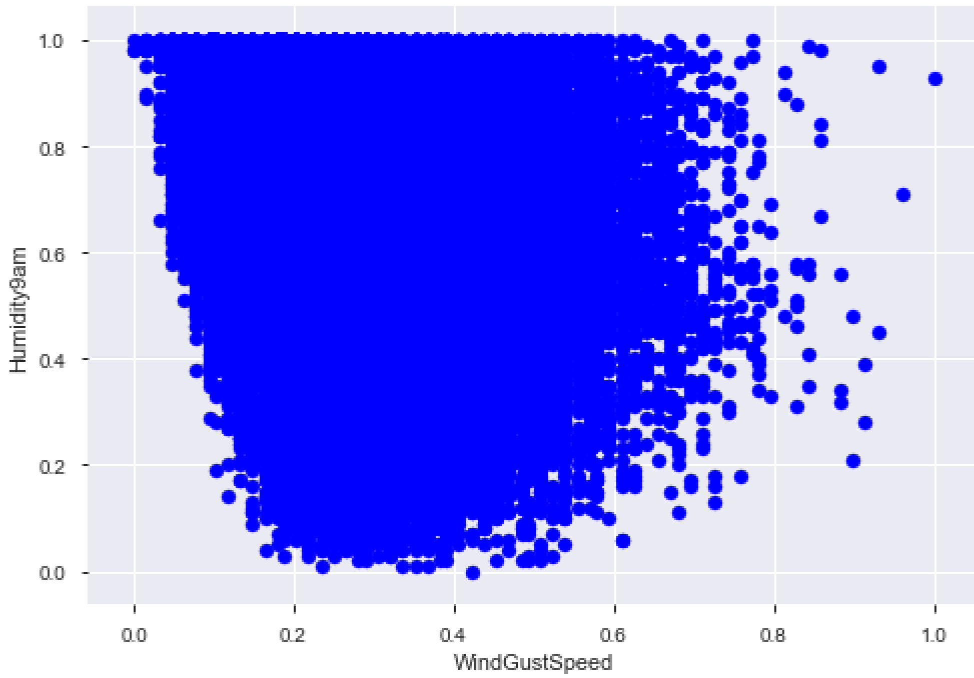
	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pr
0	0.023256	0.513064	0.446154	0.001632	0.866667	0.289062	0.866667	0.93333
1	0.023256	0.370546	0.494505	0.000000	0.933333	0.289062	0.400000	1.00000
2	0.023256	0.501188	0.507692	0.000000	1.000000	0.304688	0.866667	1.00000
3	0.023256	0.413302	0.558242	0.000000	0.266667	0.132812	0.600000	0.00000
4	0.023256	0.610451	0.652747	0.002720	0.866667	0.265625	0.066667	0.46666

5 rows × 21 columns

In [115]:
```
X = scaled_df[['WindGustSpeed','Humidity9am']]
```

In [116]:
```
plt.scatter(X["WindGustSpeed"],X["Humidity9am"],c='blue')
plt.xlabel('WindGustSpeed')
plt.ylabel('Humidity9am')
```

Out[116]:
```
Text(0, 0.5, 'Humidity9am')
```



In [117]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
```

Out[117]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

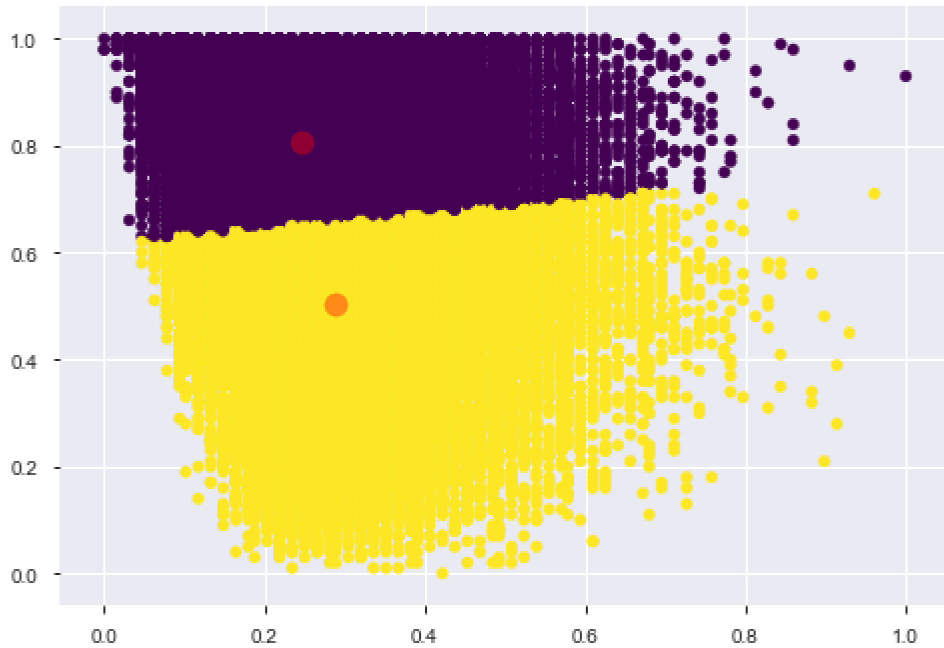
In [118]:
```
print(kmeans.labels_)
```

```
[0 1 1 ... 1 1 1]
```

In [119]:
```
centers = kmeans.cluster_centers_
print(centers)
```

```
[[0.24582544 0.80677708]
 [0.2873439  0.50284845]]
```

In [120]:
```
plt.scatter(X['WindGustSpeed'], X['Humidity9am'], c=kmeans.labels_, s=32, cmap="
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=128, alpha=0.4);
```



- using kmeans we are able to seperate the data into two clusters

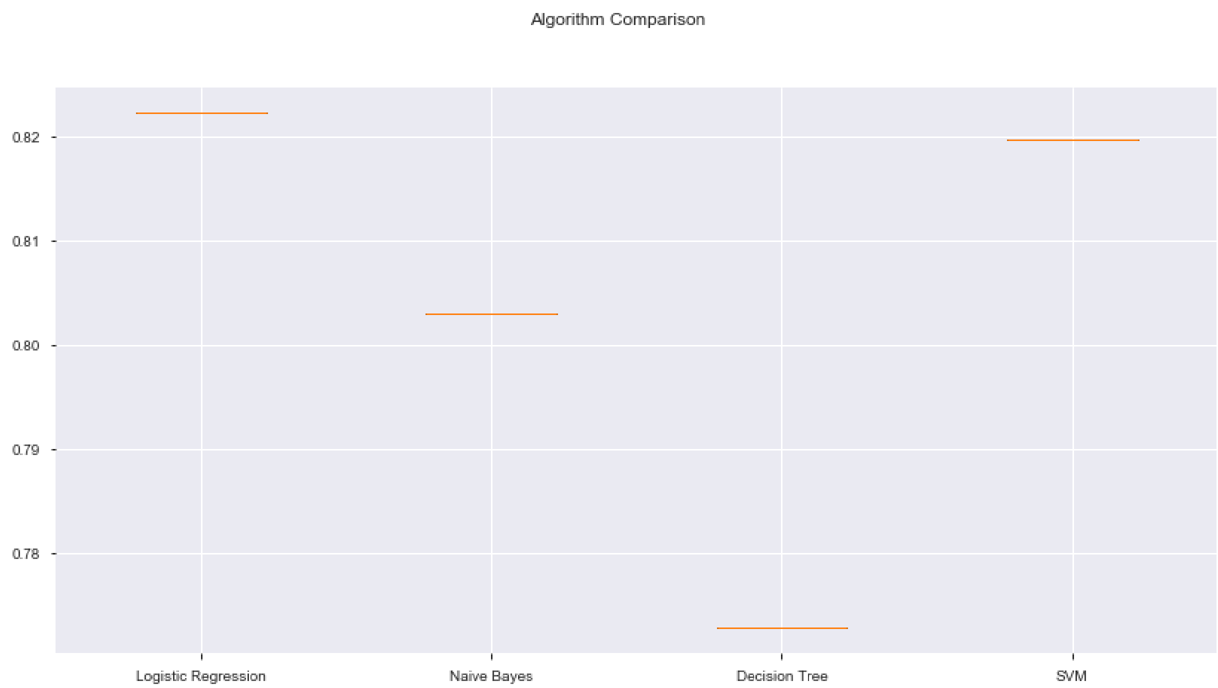
In [121]:
```
result
```

Out[121]:
```
[('Logistic Regression', 0.8222445244701576),
 ('Naive Bayes', 0.8029989963988429),
 ('Decision Tree', 0.7728614440049589),
 ('SVM', 0.8196764862152429)]
```

In [122]:
```
Names, accuracy = zip(*result)
```

In [123]:
```
def extractDigits(lst):
    return [[el] for el in lst]
accuracy1 = extractDigits(accuracy)
```

In [124]:
```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
fig.set_figheight(7)
fig.set_figwidth(14)
plt.boxplot(accuracy1)
ax.set_xticklabels(Names)
#ax.set_ylim([ymin,ymax])
plt.show()
```



Logistic regression has the best accuracy

In [ ]:

In [ ]: