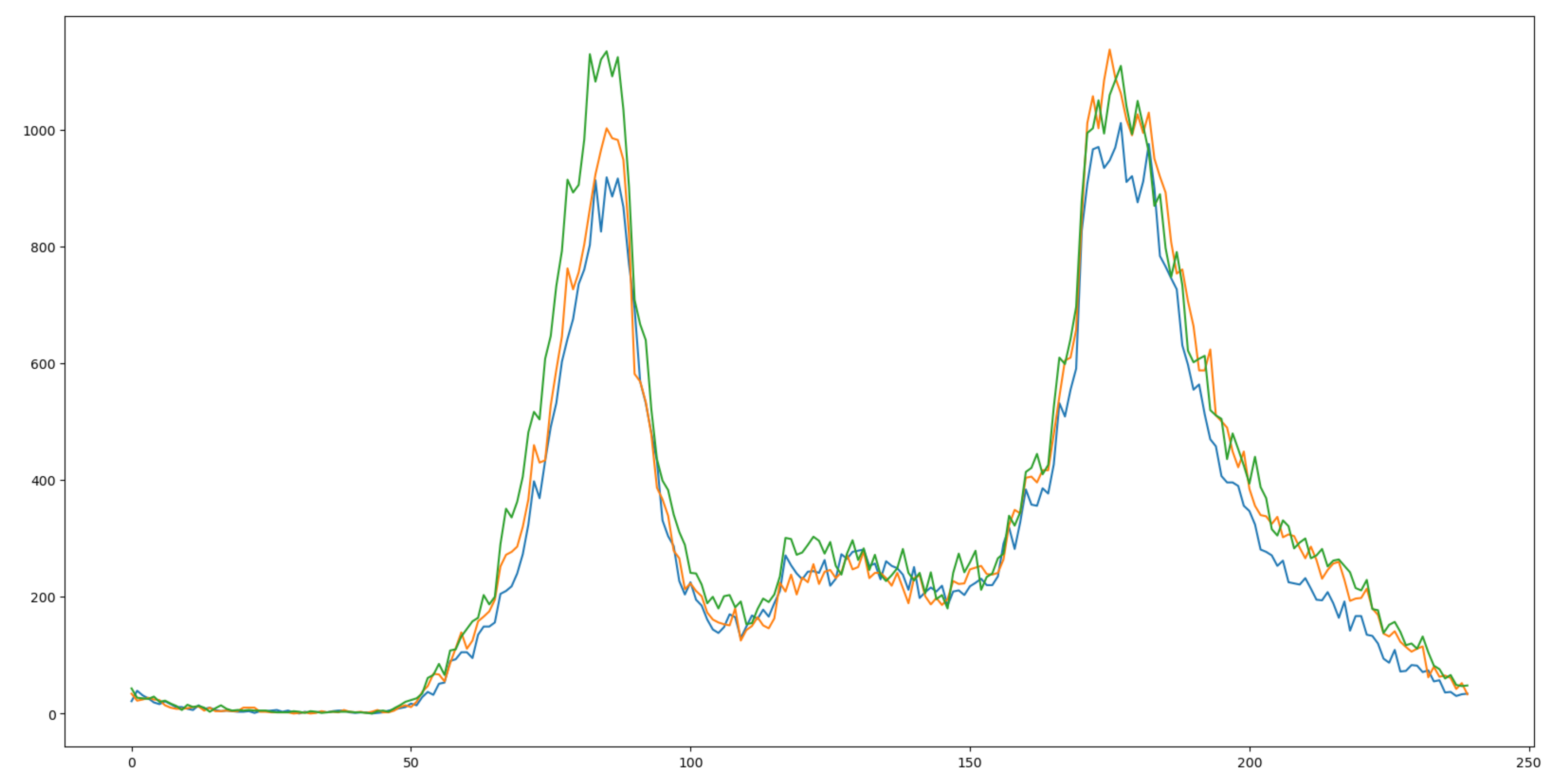
We read in the data

In [1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = 20, 10
import pandas as pd
import numpy as np


day_hour_count = pd.read_csv("../data/bikeshare_hour_count.csv")
day_hour_count = day_hour_count.fillna(0)
```

In [2]:
```
plt.figure(figsize=(20,10))
plt.plot(day_hour_count.index, day_hour_count["monday"])
plt.plot(day_hour_count.index, day_hour_count["tuesday"])
plt.plot(day_hour_count.index, day_hour_count["wednesday"])
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x11898f4f0>]
```



Assignment 4

Explain the results in a **paragraph + charts** of to describe which model you'd recommend. This means show the data and the model's line on the same chart. The paragraph is a simple justification and comparison of the several models you tried.

1. Using the `day_hour_count` dataframe create 4 dataframes `monday`, `tuesday`, `saturday` and `sunday` that represent the data for those days. (hint: Monday is day=0)

In [3]:
```
monday = day_hour_count[["hour","monday"]].copy()
```

In [4]:
```
monday = day_hour_count[["monday"]].copy()
tuesday = day_hour_count[["tuesday"]].copy()
saturday = day_hour_count[["saturday"]].copy()
sunday = day_hour_count[["sunday"]].copy()
```

In [5]:
```
monday
```

Out[5]:

	monday
0	21.0
1	39.0
2	31.0
3	26.0
4	19.0
...	...
235	36.0
236	37.0
237	30.0
238	33.0
239	34.0

240 rows × 1 columns

2a. Create 3 models fit to (`x=hour`, `y=monday`) with varying polynomial degrees (choose from `n=5,15,20`). (Repeat for `saturday` below)

Plot all the results for each polynomial.

In [6]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model, metrics
```

In [7]:
```
hour = day_hour_count[["hour"]]

poly5 = PolynomialFeatures(degree=5)
poly15 = PolynomialFeatures(degree=15)
poly20 = PolynomialFeatures(degree=20)
```

In [8]:
```
hour5 = poly5.fit_transform(hour)
```

In [9]:
```
linear5 = linear_model.LinearRegression()
linear5.fit(hour5, monday)
```

Out[9]:
```
▾ LinearRegression
LinearRegression()
```

In [10]:
```
hour15 = poly15.fit_transform(hour)
linear15 = linear_model.LinearRegression()
linear15.fit(hour15, monday)
```

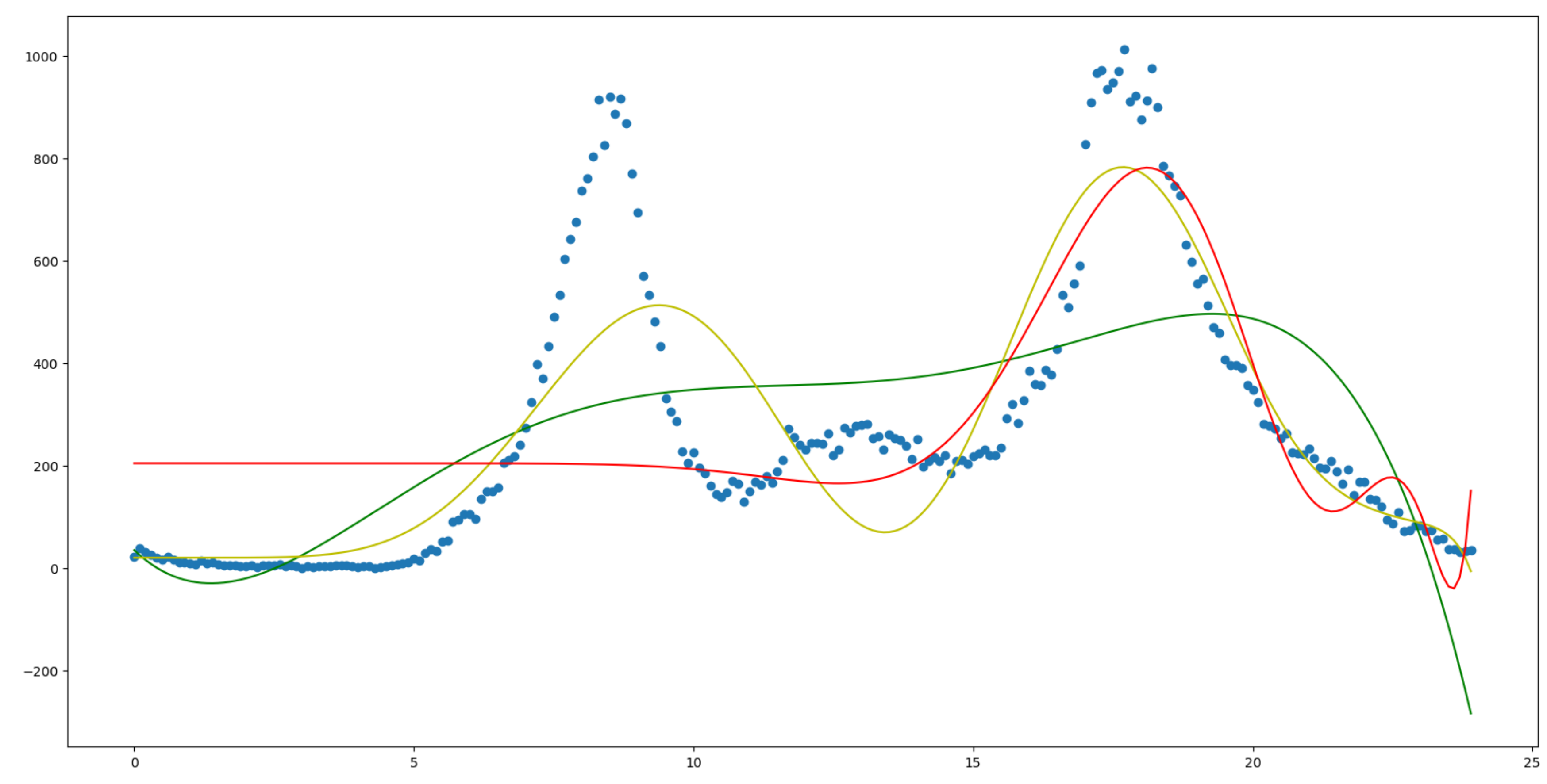
Out[10]:
```
▾ LinearRegression
LinearRegression()
```

In [11]:
```
hour20 = poly20.fit_transform(hour)
linear20 = linear_model.LinearRegression()
linear20.fit(hour20, monday)
```

Out[11]:
```
▾ LinearRegression
LinearRegression()
```

In [12]:
```
plt.scatter(hour, monday)
plt.plot(hour, linear5.predict(hour5), c='g')
plt.plot(hour, linear15.predict(hour15), c='y')
plt.plot(hour, linear20.predict(hour20), c='r')
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x12fb017b0>]
```



For monday, the model I recommend is the model with 15 polynomial degrees. This model is charted above in yellow. The 15-degree model seemed to fit the data the best, out of the three models. The 5 polynomial degree model did not capture the spikes as well as the 15-degree model. The 20-degree model was over-fit to the data and was too sensitive to changes in time.

2b. Repeat `2a` for `saturday`

In [13]:
```
linear5sat = linear_model.LinearRegression()
linear5sat.fit(hour5, saturday)
```

Out[13]:
```
▾ LinearRegression
LinearRegression()
```

In [14]:
```
linear15sat = linear_model.LinearRegression()
linear15sat.fit(hour15, saturday)
```

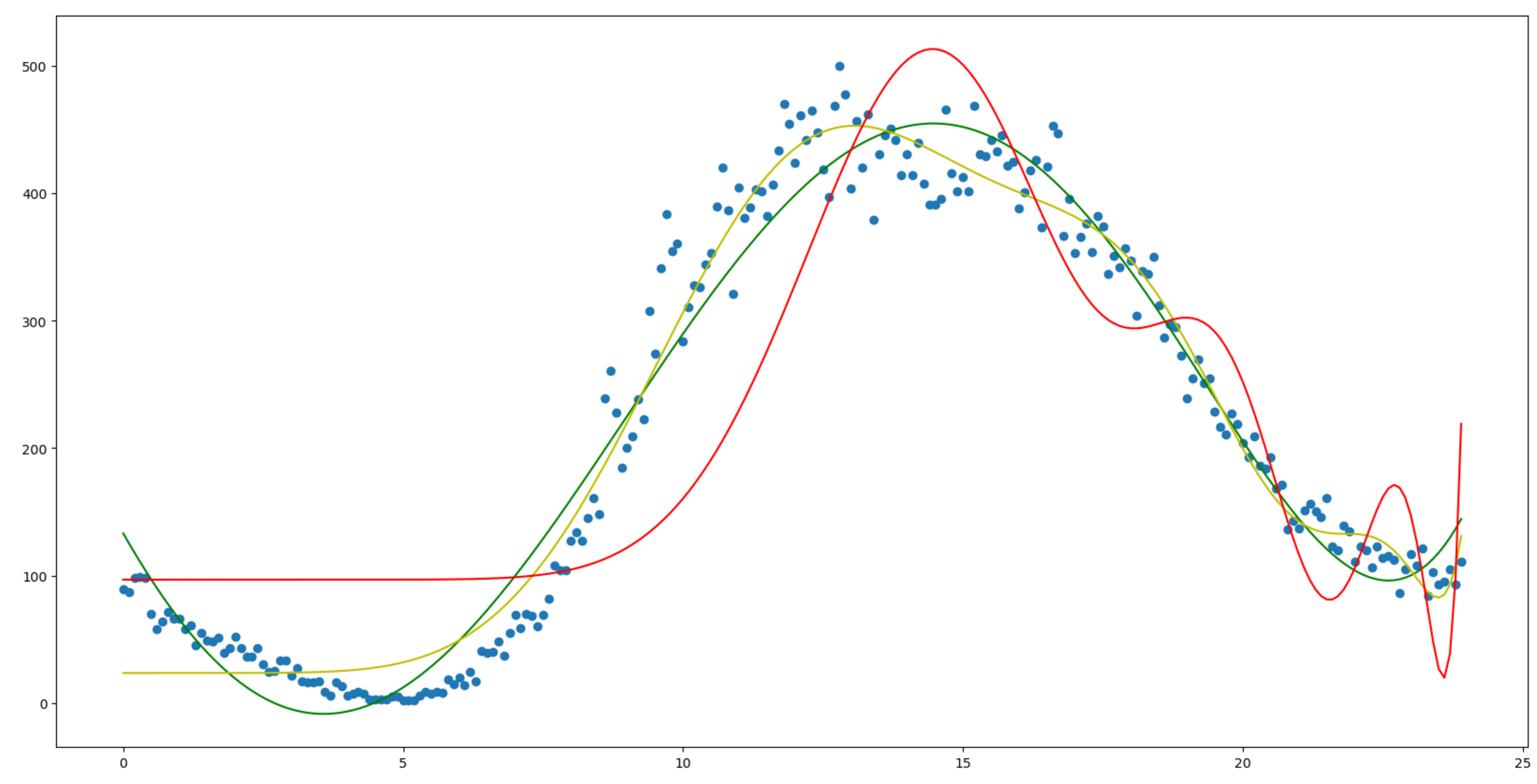
Out[14]:
```
▾ LinearRegression
LinearRegression()
```

In [15]:
```
linear20sat = linear_model.LinearRegression()
linear20sat.fit(hour20, saturday)
```

Out[15]:
```
▾ LinearRegression
LinearRegression()
```

In [16]:
```
plt.scatter(hour, saturday)
plt.plot(hour, linear5sat.predict(hour5), c='g')
plt.plot(hour, linear15sat.predict(hour15), c='y')
plt.plot(hour, linear20sat.predict(hour20), c='r')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x12fe67c10>]
```



For Saturday, the model I recommend is the one with 5 polynomia degrees, charted in green, above. This model seemed to fit the data the best, without over fitting to it. The other two models had more severe slope changes and did appear to match the data as well.

3. Using the best `monday` model's prediction, determine the errors (MSE, MAE, MAPE) between the prediction with the `monday` and `tuesday` datasets

Repeat for `saturday` / `sunday`

In [17]:
```
#Monday/Tuesday

MSEmonday = metrics.mean_squared_error(monday, linear15.predict(hour15))
MSEtues = metrics.mean_squared_error(tuesday, linear15.predict(hour15))
MAEmonday = metrics.mean_absolute_error(monday, linear15.predict(hour15))
MAEtues = metrics.mean_absolute_error(tuesday, linear15.predict(hour15))
MAPEmonday = metrics.mean_absolute_percentage_error(monday, linear15.predict(hour15))
MAPEtues = metrics.mean_absolute_percentage_error(tuesday, linear15.predict(hour15))

print("MSE Monday:",MSEmonday)
print("MSE Tuesday:",MSEtues,"\n")
print("MAE Monday:",MAEmonday)
print("MAE Tuesday:",MAEtues,"\n")
print("MAPE Monday:",MAPEmonday)
print("MAPE Tuesday:",MAPEtues)
```

```
MSE Monday: 19252.738417377448
MSE Tuesday: 23673.89255545521

MAE Monday: 97.45584672616988
MAE Tuesday: 105.0807857982436

MAPE Monday: 1246970262014381.0
MAPE Tuesday: 844753895223760.4
```

In [18]:
```
#Saturday/Sunday

MSEsat = metrics.mean_squared_error(saturday, linear5.predict(hour5))
MSEsun = metrics.mean_squared_error(sunday, linear5.predict(hour5))
MAEsat = metrics.mean_absolute_error(saturday, linear5.predict(hour5))
MAEsun = metrics.mean_absolute_error(sunday, linear5.predict(hour5))
MAPEsat = metrics.mean_absolute_percentage_error(saturday, linear5.predict(hour5))
MAPEsun = metrics.mean_absolute_percentage_error(sunday, linear5.predict(hour5))

print("MSE Saturday:",MSEsat)
print("MSE Sunday:",MSEsun,"\n")
print("MAE Saturday:",MAEsat)
print("MAE Sunday:",MAEsun,"\n")
print("MAPE Saturday:",MAPEsat)
print("MAPE Sunday:",MAPEsun)
```

```
MSE Saturday: 21579.82661326697
MSE Sunday: 26201.963969261542

MAE Saturday: 118.84225835863452
MAE Sunday: 130.22382193115354

MAPE Saturday: 3.8944912842277457
MAPE Sunday: 4.312959816153958
```

4. With `saturday`, use `train_test_split` to create training and test sets and build a model. Create predictions using the xtest from and determine the errors between these predictions and the ytest (MSE, MAE, MAPE).

repeat for `monday`

In [19]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(hour, saturday, test_size=0.2)

xtrain5 = PolynomialFeatures(degree=5).fit_transform(xtrain)
xtest5 = PolynomialFeatures(degree=5).fit_transform(xtest)

train_linear5 = linear_model.LinearRegression().fit(xtrain5, ytrain)

MSEsat = metrics.mean_squared_error(ytest, train_linear5.predict(xtest5))
MAEsat = metrics.mean_absolute_error(ytest, train_linear5.predict(xtest5))
MAPEsat = metrics.mean_absolute_percentage_error(ytest, train_linear5.predict(xtest5))

print("MSE Saturday:",MSEsat)
print("MAE Saturday:",MAEsat)
print("MAPE Saturday:",MAPEsat)
```

```
MSE Saturday: 970.4829578942678
MAE Saturday: 25.130921478189777
MAPE Saturday: 0.5629278003088224
```

In [20]:
```
xtrain, xtest, ytrain, ytest = train_test_split(hour, monday, test_size=0.2)

xtrain15 = PolynomialFeatures(degree=15).fit_transform(xtrain)
xtest15 = PolynomialFeatures(degree=15).fit_transform(xtest)

train_linear15 = linear_model.LinearRegression().fit(xtrain15, ytrain)

MSEmonday = metrics.mean_squared_error(ytest, train_linear15.predict(xtest15))
MAEmonday = metrics.mean_absolute_error(ytest, train_linear15.predict(xtest15))
MAPEmonday = metrics.mean_absolute_percentage_error(ytest, train_linear15.predict(xtest15))

print("MSE Monday:",MSEmonday)
print("MAE Monday:",MAEmonday)
print("MAPE Monday:",MAPEmonday)
```

```
MSE Monday: 25128.092198510352
MAE Monday: 104.80686734709616
MAPE Monday: 1.7024496746096807
```

In [ ]: