

We read in the data

```
In [178]: import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = 20, 10
import pandas as pd
import numpy as np
import math

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

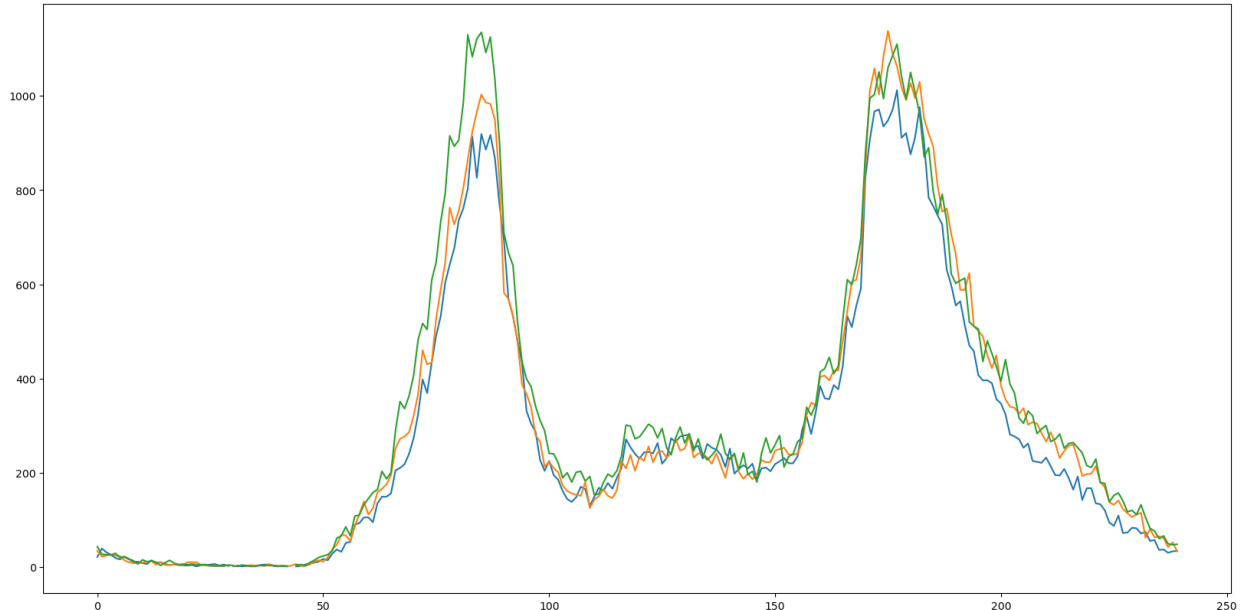
Out[178]:

	hour	monday	tuesday	wednesday	thursday	friday	saturday	sunday
0	0.0	21.0	34.0	43.0	47.0	51.0	89.0	106.0
1	0.1	39.0	22.0	27.0	37.0	56.0	87.0	100.0
2	0.2	31.0	24.0	26.0	42.0	50.0	98.0	77.0
3	0.3	26.0	27.0	25.0	29.0	52.0	99.0	87.0
4	0.4	19.0	24.0	29.0	29.0	50.0	98.0	69.0
...
235	23.5	36.0	65.0	60.0	94.0	80.0	93.0	28.0
236	23.6	37.0	61.0	66.0	100.0	81.0	95.0	28.0
237	23.7	30.0	42.0	49.0	80.0	101.0	105.0	27.0
238	23.8	33.0	52.0	47.0	79.0	91.0	93.0	24.0
239	23.9	34.0	33.0	48.0	65.0	105.0	111.0	23.0

240 rows × 8 columns

```
In [179]: 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[179]: [<matplotlib.lines.Line2D at 0x7feaa6802ec0>]
```



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 [180]: monday = day_hour_count[["hour", "monday"]].copy()
tuesday = day_hour_count[["hour", "tuesday"]].copy()
saturday = day_hour_count[["hour", "saturday"]].copy()
sunday = day_hour_count[["hour", "sunday"]].copy()
```

```
In [181]:
```

monday, tuesday, saturday, sunday

```

Out[181]: (
    hour  monday
0      0.0    21.0
1      0.1    39.0
2      0.2    31.0
3      0.3    26.0
4      0.4    19.0
..      ...    ...
235    23.5    36.0
236    23.6    37.0
237    23.7    30.0
238    23.8    33.0
239    23.9    34.0

[240 rows x 2 columns],
    hour  tuesday
0      0.0    34.0
1      0.1    22.0
2      0.2    24.0
3      0.3    27.0
4      0.4    24.0
..      ...    ...
235    23.5    65.0
236    23.6    61.0
237    23.7    42.0
238    23.8    52.0
239    23.9    33.0

[240 rows x 2 columns],
    hour  saturday
0      0.0    89.0
1      0.1    87.0
2      0.2    98.0
3      0.3    99.0
4      0.4    98.0
..      ...    ...
235    23.5    93.0
236    23.6    95.0
237    23.7   105.0
238    23.8    93.0
239    23.9   111.0

[240 rows x 2 columns],
    hour  sunday
0      0.0   106.0
1      0.1   100.0
2      0.2    77.0
3      0.3    87.0
4      0.4    69.0

```

```

..      ...      ...
235  23.5      28.0
236  23.6      28.0
237  23.7      27.0
238  23.8      24.0
239  23.9      23.0

[240 rows x 2 columns])

```

```

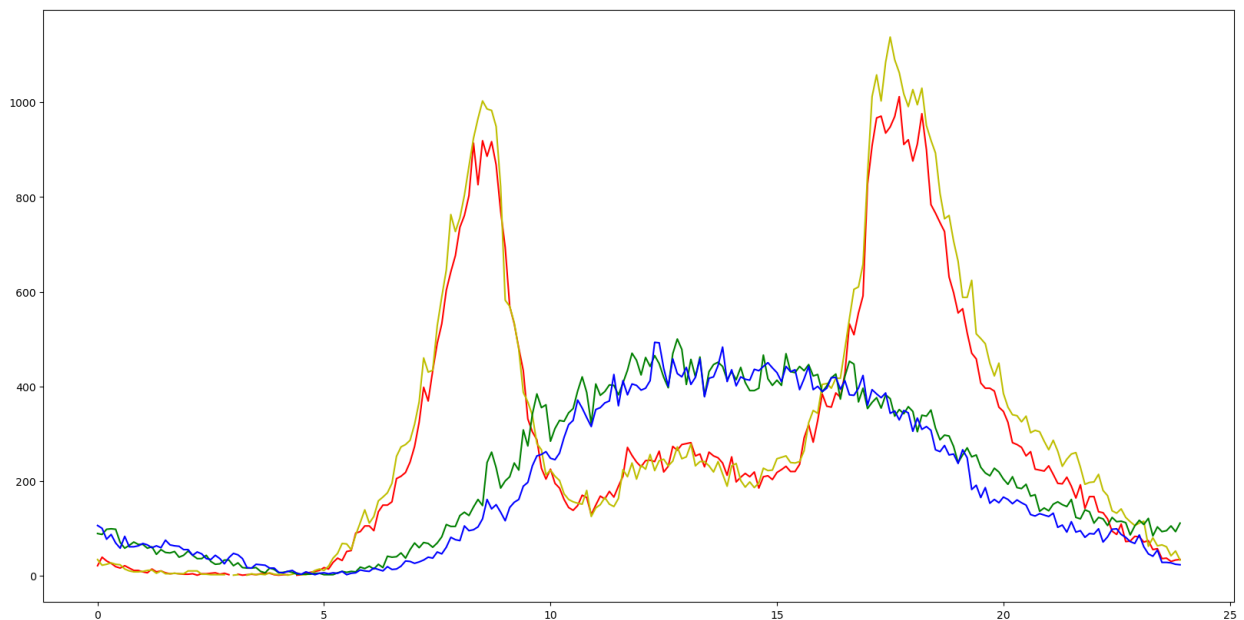
In [182]: plt.figure(figsize=(20,10))
plt.plot(monday.hour, monday.monday, c = "r")
plt.plot(tuesday.hour, tuesday.tuesday, c = "y")
plt.plot(saturday.hour, saturday.saturday, c = "g")
plt.plot(sunday.hour, sunday.sunday, c = "b")

```

```

Out[182]: [<matplotlib.lines.Line2D at 0x7fea8af69210>]

```



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 [183]: from sklearn import linear_model, metrics
          from sklearn.preprocessing import PolynomialFeatures
          from sklearn.model_selection import train_test_split

          linear = linear_model.LinearRegression()

          ridge = linear_model.Ridge()

          n_5 = PolynomialFeatures(degree = 5)
          n_15 = PolynomialFeatures(degree = 15)
          n_20 = PolynomialFeatures(degree = 20)
```

```
In [184]: # Monday

          monday_clean = monday.dropna()

          mon_x = monday_clean.hour

          mon_y = monday_clean.monday
```

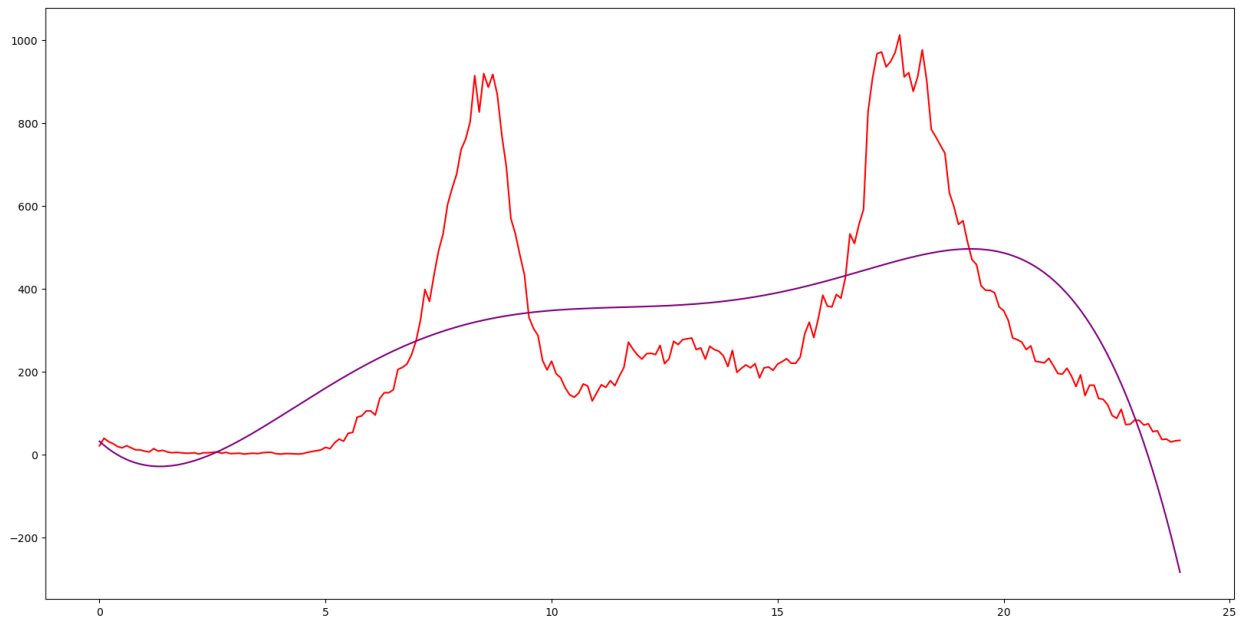
```
In [185]: monday_5 = n_5.fit_transform(mon_x.values.reshape(-1,1))

linear.fit(monday_5, mon_y)

#ridge.fit(monday_5, mon_y) I tested ridge and it is extremely similar

plt.figure(figsize=(20,10))
plt.plot(mon_x, mon_y, c = "r")
plt.plot(mon_x, np.dot(monday_5, linear.coef_) + linear.intercept_, c = "p")
#plt.plot(mon_x, np.dot(monday_5, ridge.coef_) + ridge.intercept_, c = "b")
```

Out[185]: [

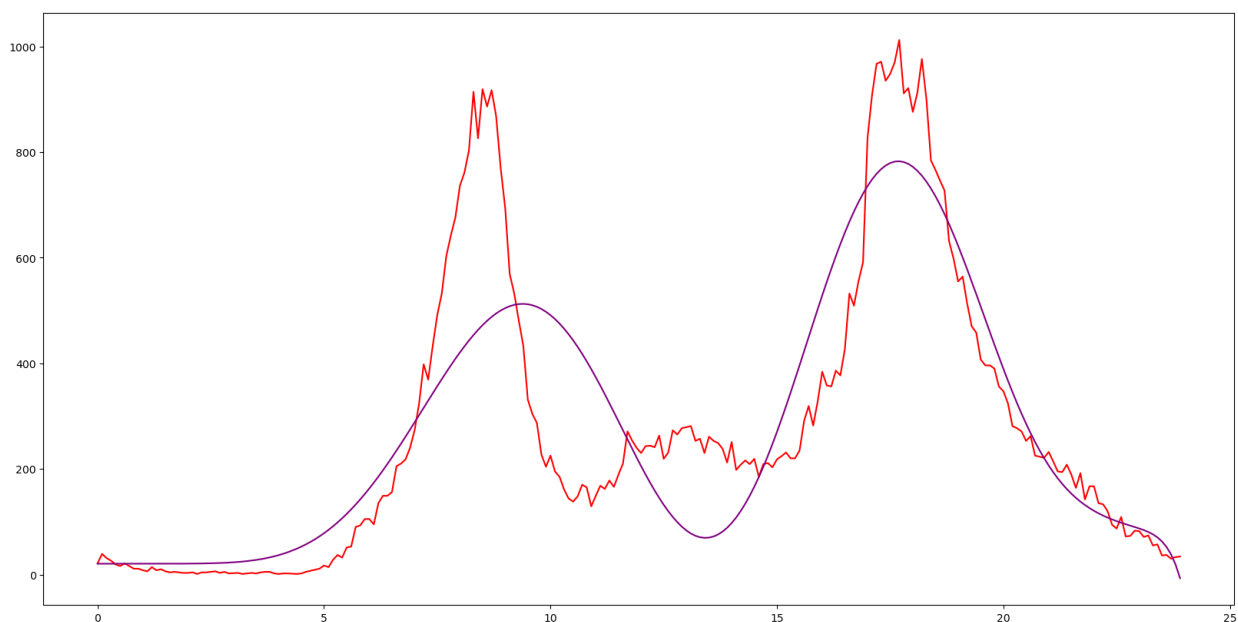


```
In [186]: monday_15 = n_15.fit_transform(mon_x.values.reshape(-1,1)) #Probably b
linear.fit(monday_15, mon_y)

#ridge.fit(monday_15, mon_y) I tested ridge and its significantly worse

plt.figure(figsize=(20,10))
plt.plot(mon_x, mon_y, c = "r")
plt.plot(mon_x, np.dot(monday_15, linear.coef_) + linear.intercept_, c = "b")
#plt.plot(mon_x, np.dot(monday_15, ridge.coef_) + ridge.intercept_, c = "g")
```

Out[186]: [



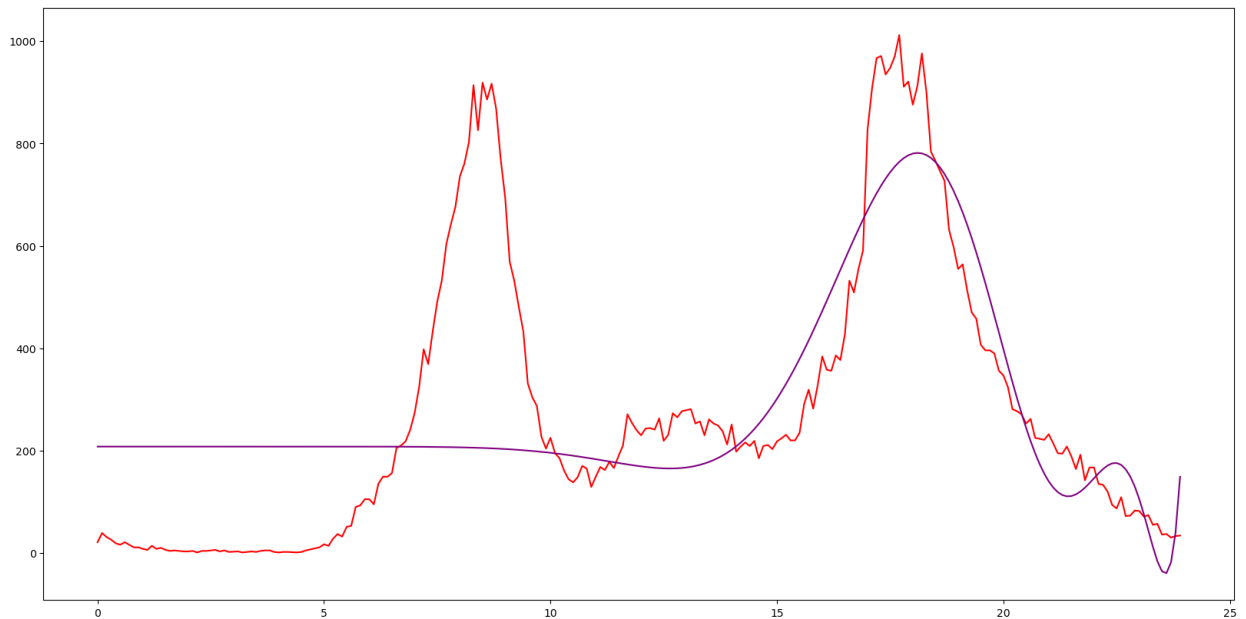
```
In [187]: monday_20 = n_20.fit_transform(mon_x.values.reshape(-1,1))

linear.fit(monday_20, mon_y)

#ridge.fit(monday_20, mon_y) I tested ridge and its significantly worse

plt.figure(figsize=(20,10))
plt.plot(mon_x, mon_y, c = "r")
plt.plot(mon_x, np.dot(monday_20, linear.coef_) + linear.intercept_, c = "b")
#plt.plot(mon_x, np.dot(monday_20, ridge.coef_) + ridge.intercept_, c = "g")
```

Out[187]: [matplotlib.lines.Line2D at 0x7fea86d69cf0]



2b. Repeat 2a for saturday

```
In [188]: # Saturday

saturday_clean = saturday.dropna()

sat_x = saturday_clean.hour

sat_y = saturday_clean.saturday
```



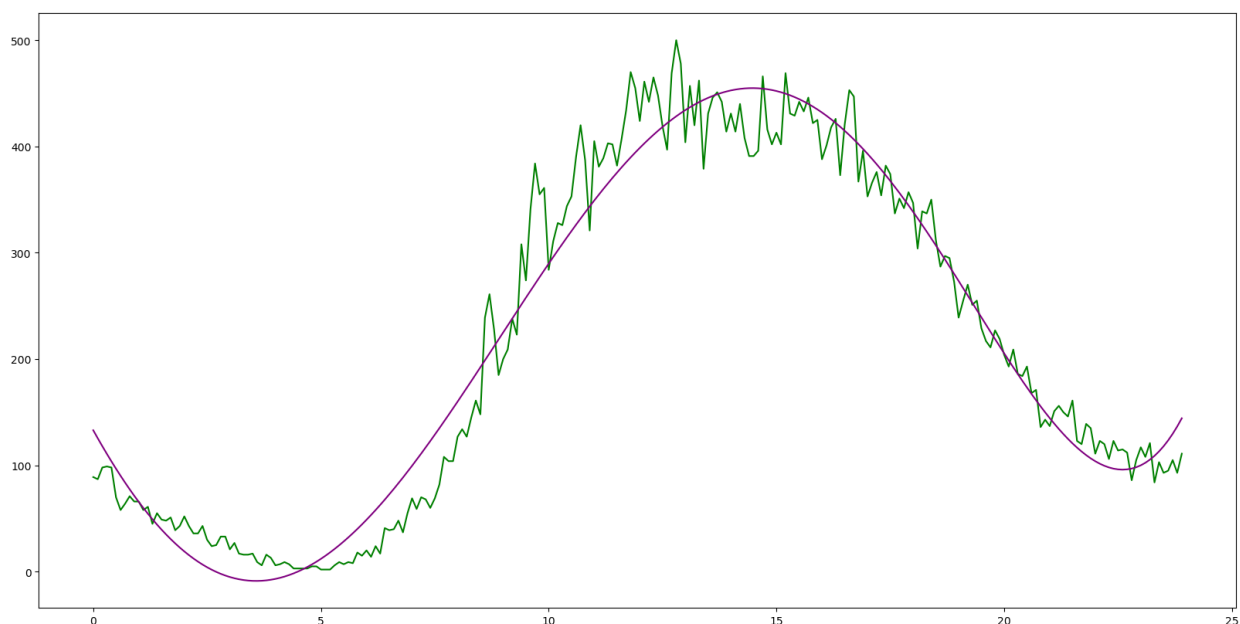
```
In [189]: saturday_5 = n_5.fit_transform(sat_x.values.reshape(-1,1))

linear.fit(saturday_5, sat_y)

#ridge.fit(saturday_5, sat_y) I tested ridge and it is extremely simil

plt.figure(figsize=(20,10))
plt.plot(sat_x, sat_y, c = "g")
plt.plot(sat_x, np.dot(saturday_5, linear.coef_) + linear.intercept_,
#plt.plot(sat_x, np.dot(saturday_5, ridge.coef_) + ridge.intercept_, c
```

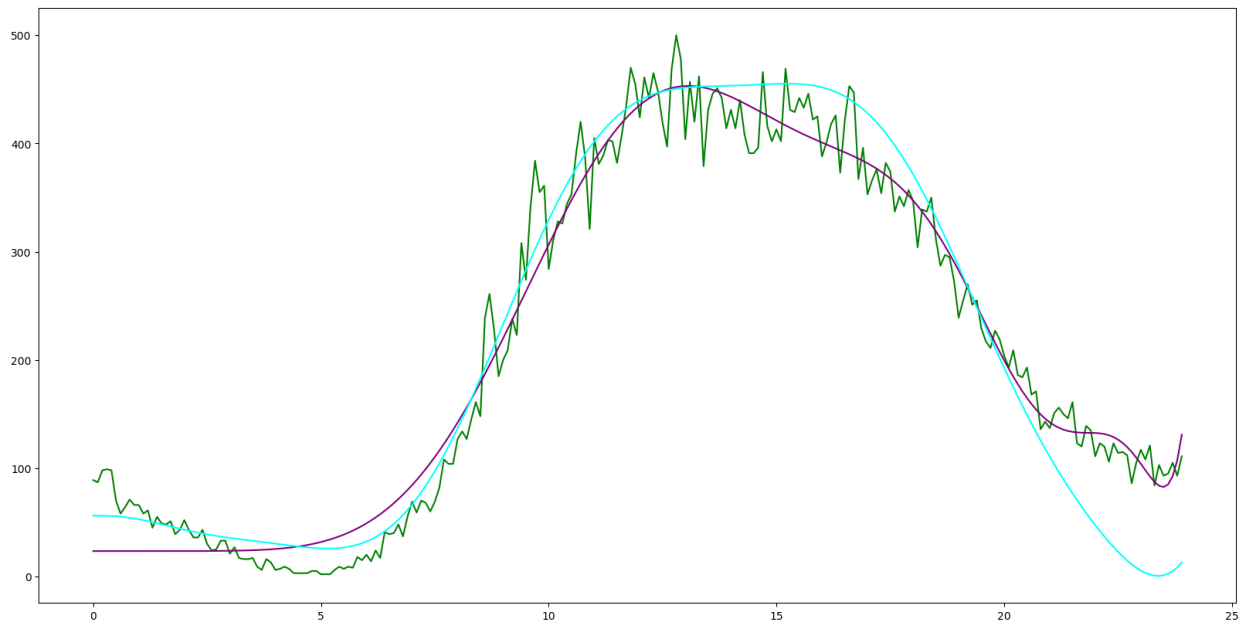
Out[189]: [



```
In [190]: saturday_15 = n_15.fit_transform(sat_x.values.reshape(-1,1)) #Probably
linear.fit(saturday_15, sat_y)
ridge.fit(saturday_15, sat_y)

plt.figure(figsize=(20,10))
plt.plot(sat_x, sat_y, c = "g")
plt.plot(sat_x, np.dot(saturday_15, linear.coef_) + linear.intercept_,
plt.plot(sat_x, np.dot(saturday_15, ridge.coef_) + ridge.intercept_, c
```

Out[190]: [



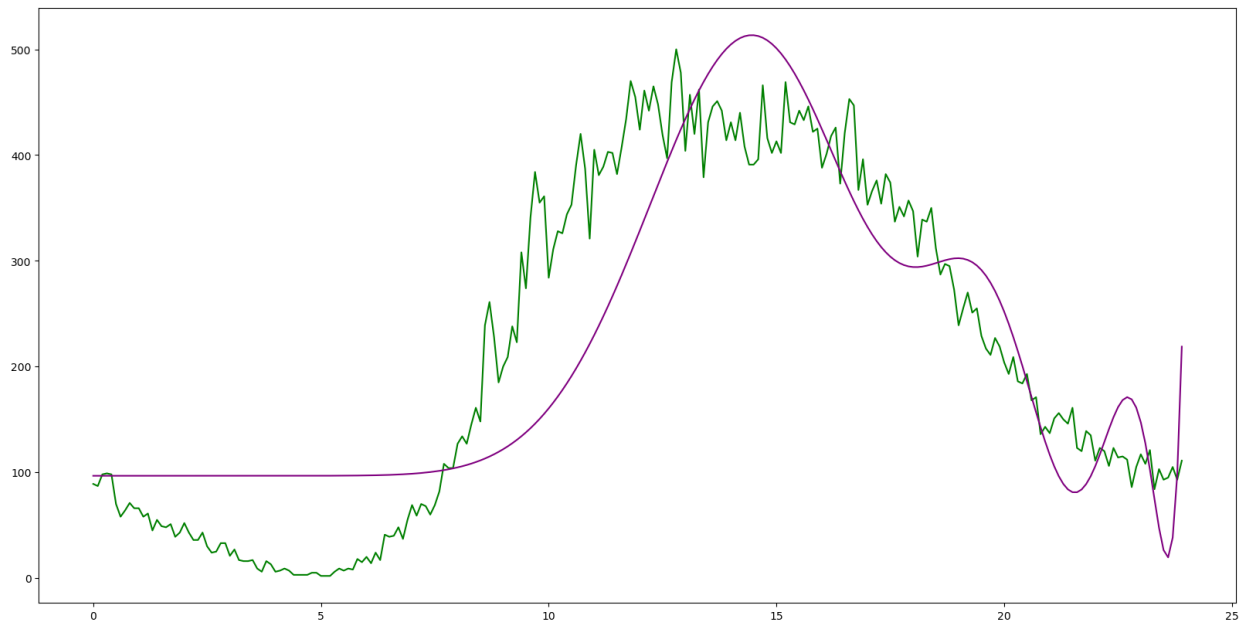
```
In [191]: saturday_20 = n_20.fit_transform(sat_x.values.reshape(-1,1))

linear.fit(saturday_20, sat_y)

#ridge.fit(saturday_20, sat_y) I tested ridge and it is significantly

plt.figure(figsize=(20,10))
plt.plot(sat_x, sat_y, c = "g")
plt.plot(sat_x, np.dot(saturday_20, linear.coef_) + linear.intercept_,
#plt.plot(sat_x, np.dot(saturday_20, ridge.coef_) + ridge.intercept_,
```

Out[191]: [



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

Of the three models for Monday, n equal to 5, 15, and 20, I assessed n equal to 15 to be the best predictor of y. This is visually apparent when plotted and backed up mathematically when comparing the MSE, MAE, and MAPE. The model n equal to 5 does not account for the peaks or troughs. The model n equal to 20 heavily biases towards the later peak. The model n equal to 15 is reasonably effective at predicting values of y but seems to under estimate peaks and over estimate troughs. This is a linear model as the ridge models had notable deviations from the values.

```
In [192]: #Monday

linear_mon = linear_model.LinearRegression()

linear_mon.fit(monday_15, mon_y)

(
    metrics.mean_squared_error(mon_y, linear_mon.predict(monday_15)),
    metrics.mean_absolute_error(mon_y, linear_mon.predict(monday_15)),
    metrics.mean_absolute_percentage_error(mon_y, linear_mon.predict(monday_15))
)
```

Out[192]: (19403.939668165112, 98.159188823613, 1.9402673082566422)

The Monday model n equal to 15 is a reasonable predictor of Tuesday values. This is apparent when comparing Monday and Tuesday values plotted. It is also backed up mathematically when comparing the MSE, MAE, and MAPE.

```
In [193]: #Tuesday

tuesday_clean = tuesday.dropna()

tues_y = tuesday_clean.tuesday

linear_tues = linear_model.LinearRegression()

linear_tues.fit(monday_15, tues_y)

(
    metrics.mean_squared_error(tues_y, linear_tues.predict(monday_15)),
    metrics.mean_absolute_error(tues_y, linear_tues.predict(monday_15)),
    metrics.mean_absolute_percentage_error(tues_y, linear_tues.predict(monday_15))
)
```

Out[193]: (22184.492963002634, 103.74837647509011, 2.180992352984371)

Of the three models for Saturday, n equal to 5, 15, and 20, I assessed n equal to 15 to be the best predictor of y. Between n equal to 5 and n equal to 15 it is very close. Ultimately n equal to 15 has slightly better values of MSE, MAE, and MAPE and accounts for some of the peak values for Saturday. It does struggle with the early hour values for Saturday. The model n equal to 20 seems to overfit for the later times and significantly under fit for the earlier times. The model n equal to 15 is a linear model as the ridge model, while the best of the ridge model so far, has a significant drop off later in the day.

```
In [194]: #Saturday

linear_sat = linear_model.LinearRegression()

linear_sat.fit(saturday_15, sat_y)

(
    metrics.mean_squared_error(sat_y, linear_sat.predict(saturday_15))
    metrics.mean_absolute_error(sat_y, linear_sat.predict(saturday_15))
    metrics.mean_absolute_percentage_error(sat_y, linear_sat.predict(saturday_15))
)
```

Out[194]: (809.0490983038588, 22.623668377460458, 0.7329285426915966)

The model for Saturday n equal to 15 is a reasonable predictor for Sunday. Infact it is actually slightly better at predicting values for Sunday than it is for Saturday. This is visually apparent when plotted and is mathematically supported by MSE, MAE, and MAPE.

```
In [195]: #Sunday

sunday_clean = sunday.dropna()

sun_y = sunday_clean.sunday

linear_sun = linear_model.LinearRegression()

linear_sun.fit(saturday_15, sun_y)

(
    metrics.mean_squared_error(sun_y, linear_sun.predict(saturday_15))
    metrics.mean_absolute_error(sun_y, linear_sun.predict(saturday_15))
    metrics.mean_absolute_percentage_error(sun_y, linear_sun.predict(saturday_15))
)
```

Out[195]: (625.9507592509589, 20.660683731665312, 0.9311980840848827)

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

Based on the MSE, MAE, and MAPE values linear15, in blue, appears to have the least overall errors and seems to have reasonable estimates of y for Saturday. The points are also fairly close to the actual data values for Saturday.

In [196]: *#Saturday*

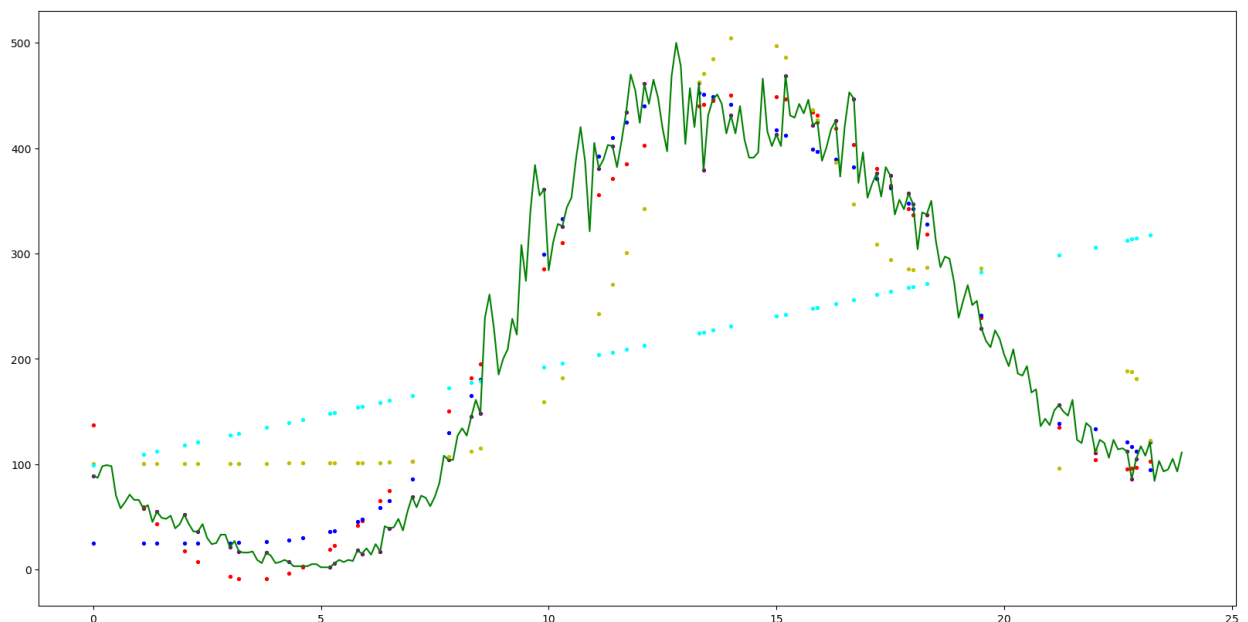
```

sat_xtrain, sat_xtest, sat_ytrain, sat_ytest = train_test_split(sat_x,
linear = linear_model.LinearRegression().fit(sat_xtrain.values.reshape
sat_xtrain5 = PolynomialFeatures(degree=5).fit_transform(sat_xtrain.va
sat_xtest5 = PolynomialFeatures(degree=5).fit_transform(sat_xtest.valu
linear5 = linear_model.LinearRegression().fit(sat_xtrain5, sat_ytrain)
sat_xtrain15 = PolynomialFeatures(degree=15).fit_transform(sat_xtrain.
sat_xtest15 = PolynomialFeatures(degree=15).fit_transform(sat_xtest.va
linear15 = linear_model.LinearRegression().fit(sat_xtrain15, sat_ytrain
sat_xtrain20 = PolynomialFeatures(degree=20).fit_transform(sat_xtrain.
sat_xtest20 = PolynomialFeatures(degree=20).fit_transform(sat_xtest.va
linear20 = linear_model.LinearRegression().fit(sat_xtrain20, sat_ytrain

size = 8
plt.scatter(sat_xtest, sat_ytest, c = 'purple', s=size)
plt.plot(sat_x, sat_y, c = "g")
plt.scatter(sat_xtest, linear5.predict(sat_xtest5), c='r', s=size)
plt.scatter(sat_xtest, linear15.predict(sat_xtest15), c='b', s=size)
plt.scatter(sat_xtest, linear20.predict(sat_xtest20), c = 'y', s=size)
plt.scatter(sat_xtest, linear.predict(sat_xtest.values.reshape(-1, 1))

```

Out[196]: <matplotlib.collections.PathCollection at 0x7fea85bda3e0>



In [197]: *#Saturday MSE*

```
(
    metrics.mean_squared_error(sat_ytest, linear.predict(sat_xtest.val
    metrics.mean_squared_error(sat_ytest, linear5.predict(sat_xtest5))
    metrics.mean_squared_error(sat_ytest, linear15.predict(sat_xtest15
    metrics.mean_squared_error(sat_ytest, linear20.predict(sat_xtest20
)
```

Out[197]: (23065.2703051803, 905.7097558268573, 839.1941372782458, 6509.813745670064)

In [198]: *#Saturday MAE*

```
(
    metrics.mean_absolute_error(sat_ytest, linear.predict(sat_xtest.val
    metrics.mean_absolute_error(sat_ytest, linear5.predict(sat_xtest5))
    metrics.mean_absolute_error(sat_ytest, linear15.predict(sat_xtest15
    metrics.mean_absolute_error(sat_ytest, linear20.predict(sat_xtest20
)
```

Out[198]: (139.32564971520176, 24.728582492281507, 23.257874920289463, 68.79431006780094)

In [199]: *#Saturday MAPE*

```
(
    metrics.mean_absolute_percentage_error(sat_ytest, linear.predict(s
    metrics.mean_absolute_percentage_error(sat_ytest, linear5.predict(
    metrics.mean_absolute_percentage_error(sat_ytest, linear15.predict
    metrics.mean_absolute_percentage_error(sat_ytest, linear20.predict
)
```

Out[199]: (4.944195277429863, 0.632339983209748, 0.9955009244667496, 3.2326811561478443)

Based on the MSE, MAE, and MAPE values linear15, in blue, appears to have the least overall errors and may have semi-reasonable estimates of y for Monday. The points are the closet of the models to estimating the two peaks in the Monday but seems to underestimate both peaks and the central trough as well.

In [200]:


```

#Monday
mon_xtrain, mon_xtest, mon_ytrain, mon_ytest = train_test_split(mon_x,

linear = linear_model.LinearRegression().fit(mon_xtrain.values.reshape

mon_xtrain5 = PolynomialFeatures(degree=5).fit_transform(mon_xtrain.va
mon_xtest5 = PolynomialFeatures(degree=5).fit_transform(mon_xtest.valu

linear5 = linear_model.LinearRegression().fit(mon_xtrain5, mon_ytrain)

mon_xtrain15 = PolynomialFeatures(degree=15).fit_transform(mon_xtrain.
mon_xtest15 = PolynomialFeatures(degree=15).fit_transform(mon_xtest.va

linear15 = linear_model.LinearRegression().fit(mon_xtrain15, mon_ytrai

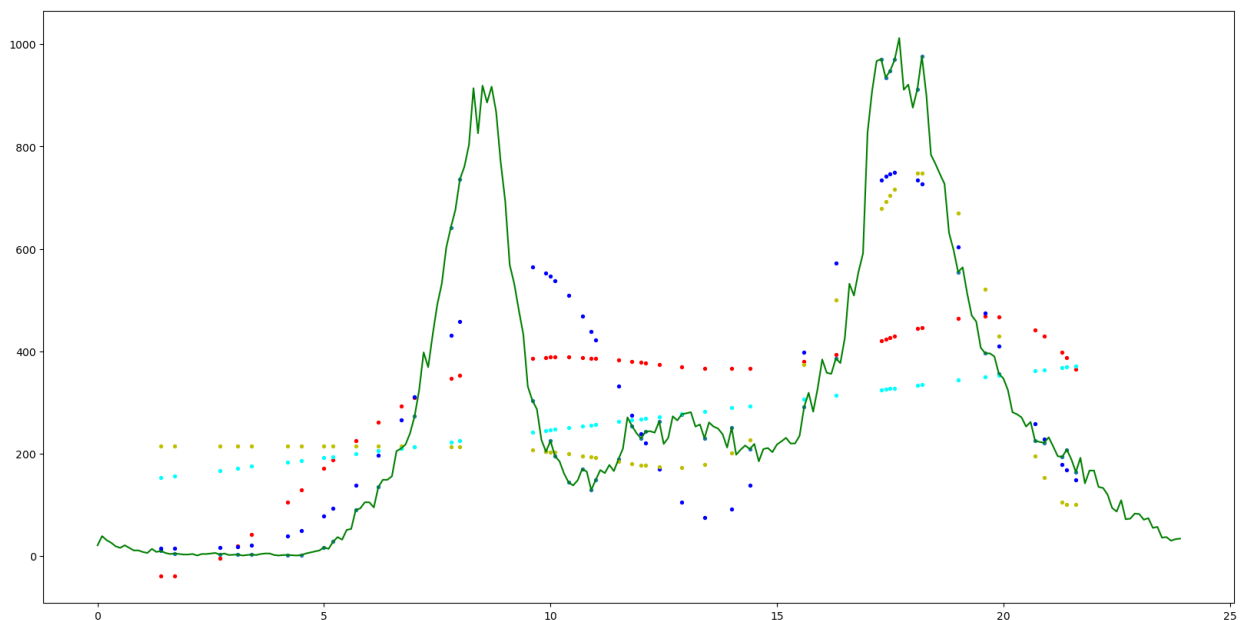
mon_xtrain20 = PolynomialFeatures(degree=20).fit_transform(mon_xtrain.
mon_xtest20 = PolynomialFeatures(degree=20).fit_transform(mon_xtest.va

linear20 = linear_model.LinearRegression().fit(mon_xtrain20, mon_ytrai

size = 8
plt.scatter(mon_xtest, mon_ytest, s=size)
plt.plot(mon_x, mon_y, c = "g")
plt.scatter(mon_xtest, linear5.predict(mon_xtest5), c='r', s=size)
plt.scatter(mon_xtest, linear15.predict(mon_xtest15), c='b', s=size)
plt.scatter(mon_xtest, linear20.predict(mon_xtest20), c='y', s=size)
plt.scatter(mon_xtest, linear.predict(mon_xtest.values.reshape(-1, 1))

```

Out[200]: <matplotlib.collections.PathCollection at 0x7fea85c5aaa0>



In [201]: *#Monday MSE*

```
(  
    metrics.mean_squared_error(mon_ytest, linear.predict(mon_xtest.val  
    metrics.mean_squared_error(mon_ytest, linear5.predict(mon_xtest5))  
    metrics.mean_squared_error(mon_ytest, linear15.predict(mon_xtest15  
    metrics.mean_squared_error(mon_ytest, linear20.predict(mon_xtest20  
)
```

Out[201]: (68925.67191776562, 56851.792551018436, 29129.16352815497, 28180.299288658687)

In [202]: *#Monday MAE*

```
(  
    metrics.mean_absolute_error(mon_ytest, linear.predict(mon_xtest.va  
    metrics.mean_absolute_error(mon_ytest, linear5.predict(mon_xtest5)  
    metrics.mean_absolute_error(mon_ytest, linear15.predict(mon_xtest1  
    metrics.mean_absolute_error(mon_ytest, linear20.predict(mon_xtest2  
)
```

Out[202]: (177.61003750483667, 188.67505600636116, 128.9651071108673, 128.98099545130987)

In [203]: *#Monday MAPE*

```
(  
    metrics.mean_absolute_percentage_error(mon_ytest, linear.predict(m  
    metrics.mean_absolute_percentage_error(mon_ytest, linear5.predict(  
    metrics.mean_absolute_percentage_error(mon_ytest, linear15.predict  
    metrics.mean_absolute_percentage_error(mon_ytest, linear20.predict  
)
```

Out[203]: (8.964683576327841, 4.023080080443859, 1.8679937801413182, 10.760666489302018)