1 Executive Summary

When we have continuous-by-continuous interactions in a linear regression model, it is impossible to directly interpret the coefficients on the interactions. Generalized Additive Models(GAMs) extend generalized linear models by allowing non-linear functions of features while maintaining additivity. Since the model is additive, it is easy to examine the effect of each X_i on Y individually while holding all other predictors constant. The result is a very flexible model, where it is easy to incorporate prior knowledge and control overfitting. In this notebook, I will test and compare the resuls of two python pachages which are **statsmodels.api** and **pyGAM** which allow us to fit interactions in our model.

2 Introduction

I plan to build a linear regression model in python to explore how the weight of a car, the number of cylinders and horepower are related to the mileage of the car. Additionally, I am interested to know how the weight affects the relationship between the number of cylinders and the mileage.

In [371]:
```
# importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
```

In [372]:
```
mtcars = sm.datasets.get_rdataset("mtcars", "datasets", cache=True).data
mtcars.head()
```

Out[372]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [373]:
```
data.shape
```

Out[373]: (32, 4)

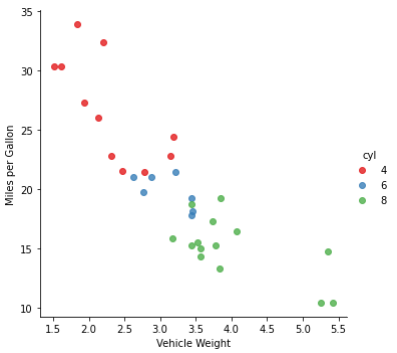
In [374]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, Mazda RX4 to Volvo 142E
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   wt      32 non-null     float64
 1   cyl     32 non-null     float64
 2   hp      32 non-null     float64
 3   mpg     32 non-null     float64
dtypes: float64(4)
memory usage: 2.5+ KB
```

The response variable is "mpg", "cyl" and "hp" represents the number of Cylinder and Horsepower Of a car. Let us visualize the relationship between the Weight, horsepower and Miles per Gallon which is also conditioned on a third variable "cyl" and plot the levels of the Culinder in different colors.

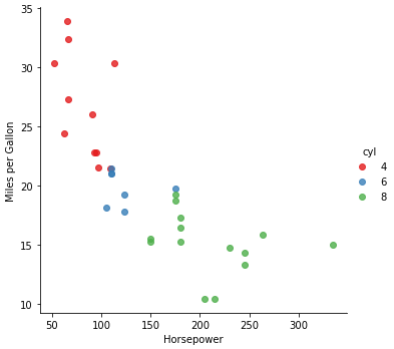
In [402]:
```
sns.lmplot(x='wt', y='mpg', hue='cyl', data=mtcars, fit_reg=False, palette='Set1', size=5, aspect=1)
plt.ylabel("Miles per Gallon")
plt.xlabel("Vehicle Weight")
```

Out[402]: Text(0.5, 20.80000000000002, 'Vehicle Weight')



In [403]:
```
sns.lmplot(x='hp', y='mpg', hue='cyl', data=mtcars, fit_reg=False, palette='Set1', size=5, aspect=1)
plt.ylabel("Miles per Gallon")
plt.xlabel("Horsepower")
```

Out[403]: Text(0.5, 20.80000000000002, 'Horsepower')



What we see here is that, heavy 8 cylinder vehicles have lower Miles per Gallon. on the other hand lighter 4 cylinder cars have higher Miles per Gallon. This confirms that the more cylinders in an engine, the more combustion that occurs, creating more movement to turn the crankshaft and power to move the car. However, more cylinders also require more gasoline to make the combustion necessary to drive the car.

3 Building the Regression Model

3.1 statsmodels.api

3.1.1 Scaling our Data

Scaling is importat for Algorithms which fit a model that uses a weighted sum of input variables, such as linear regression, logistic regression, and artificial neural networks (deep learning). Algorithms that use distance measures between examples or exemplars are also affected, such as k-nearest neighbors and support vector machines. There are also algorithms that are unaffected by the scale of numerical input variables, most notably decision trees and ensembles of trees, like random forest.

In [379]:
```
# let us transforms the features which we are interested in by scaling them to a given range (1,2)
# I will keep the "Cyl" values as they are becuase although it is numerical it consits of 3 values
# and makes the interpretation of plots easier
from sklearn.preprocessing import MinMaxScaler
trans = MinMaxScaler(feature_range=(1, 2))
names=mtcars.columns[2:]
feature_nresc=mtcars.columns[2:]
scaled_df = trans.fit_transform(mtcars[feature_nresc])
scaled_df = pd.DataFrame(scaled_df, columns=names, index=mtcars.index)
scaled_df['cyl']=np.float64(mtcars['cyl'])
```

3.1.2 Direct Effects (Main)

In [380]:
```
# Now we build our model here.
X=scaled_df[['wt', 'cyl', 'hp']]
y=mtcars.mpg
data=X.copy()
data['mpg']=y
model = sm.OLS(y,X).fit()
summary = model.summary()
summary
```

Out[380]:

OLS Regression Results

Dep. Variable:	mpg	**R-squared (uncentered):**	0.890
Model:	OLS	**Adj. R-squared (uncentered):**	0.879
Method:	Least Squares	**F-statistic:**	78.58
Date:	Sun, 15 Nov 2020	**Prob (F-statistic):**	4.95e-14
Time:	17:12:33	**Log-Likelihood:**	-107.37
No. Observations:	32	**AIC:**	220.7
Df Residuals:	29	**BIC:**	225.1
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
wt	14.2060	6.557	2.166	0.039	0.795	27.617
cyl	-5.5599	1.312	-4.238	0.000	-8.243	-2.877
hp	24.9170	7.738	3.220	0.003	9.091	40.743

Omnibus:	0.178	**Durbin-Watson:**	1.186
Prob(Omnibus):	0.915	**Jarque-Bera (JB):**	0.350
Skew:	-0.138	**Prob(JB):**	0.840
Kurtosis:	2.569	**Cond. No.**	48.3

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

As we can see, our model has been able to explain 89% variation in y with the help of X1, X2 and X3. The influence of all the 3 independant variables are significant in our model.

The coefficients of the model tell us:

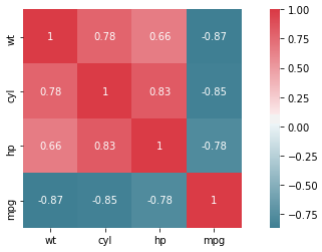
For every 1 unit increase in weight, mpg increases by 14.2 (holding cylinders constant). This is weird because our expectation is that heavier cars have a lower mpg. This indicates that the model should be corrected.
For every 1 unit increase in cylinders, mpg decreases by 5.5 (holding weight constant)
For every 1 unit increase in horsepower, mpg increases by 24.9 (holding weight constant)

3.1.3 Interaction Effects

3.1.3.1 Correlation Heatmaps

In [404]:
```
import seaborn as sns
f, ax = plt.subplots(figsize=(10, 4))
corr = data.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, annot=True, ax=ax)
```

Out[404]: <matplotlib.axes._subplots.AxesSubplot at 0x1e4160a24c8>



The correlation plot shows that all our predictors are negatively correlated with our target variable mpg. We can also observe some degree of coorelation between our predictors.

In [382]:
```
model_interaction = sm.OLS.from_formula('mpg ~ wt + cyl + hp + wt:cyl + hp:wt + cyl:hp', data=data).fit()
summary = model_interaction.summary()
summary
#summary.tables[1]
```

Out[382]:

OLS Regression Results

Dep. Variable:	mpg	**R-squared:**	0.895
Model:	OLS	**Adj. R-squared:**	0.870
Method:	Least Squares	**F-statistic:**	35.50
Date:	Sun, 15 Nov 2020	**Prob (F-statistic):**	4.66e-11
Time:	17:13:49	**Log-Likelihood:**	-66.323
No. Observations:	32	**AIC:**	146.6
Df Residuals:	25	**BIC:**	156.9
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	109.7458	21.799	5.035	0.000	64.851	154.641
wt	-33.8994	20.609	-1.645	0.113	-76.345	8.546
cyl	-6.9887	6.262	-1.116	0.275	-19.886	5.908
hp	-42.6031	31.121	-1.369	0.183	-106.697	21.491
wt:cyl	1.7138	3.052	0.562	0.579	-4.571	7.999
hp:wt	7.0502	26.002	0.271	0.789	-46.502	60.602
cyl:hp	3.4309	2.481	1.383	0.179	-1.679	8.541

Omnibus:	1.602	**Durbin-Watson:**	2.303
Prob(Omnibus):	0.449	**Jarque-Bera (JB):**	1.292
Skew:	0.301	**Prob(JB):**	0.524
Kurtosis:	2.222	**Cond. No.**	1.87e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.87e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

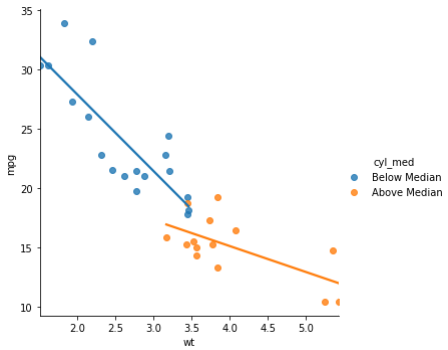
What we see here is that adding all the interactions terms does not significatly improve the R-squared they just added more complexity to our model. It also impacted the importamce of our original predictors. The model shows that the contributions of our predictited variables are not significant anymore. So the conclusion here is that sometimes it's best to start simple, and add complexity only as needed. For interactions, We should consider adding those that we think might be important based on our domain knowledge.

3.1.4 Visualizing Interactions

We can see the interactions by cutting one of the terms in the interaction along it's median, and then plotting the response variable against the other variable in the interacting pair. Even when we look at the visualization of the interaction terms, it doesnot seem that the "hp & wt" follows a good fit. So I will remove it from the model.

In [405]:
```
mtcars['cyl_med'] = mtcars.cyl > mtcars.cyl.median()
mtcars['cyl_med'] = np.where(mtcars.cyl_med == False, "Below Median", "Above Median")
sns.lmplot(x='wt', y='mpg', hue='cyl_med', data=mtcars, ci=None, size=5, aspect=1)
```

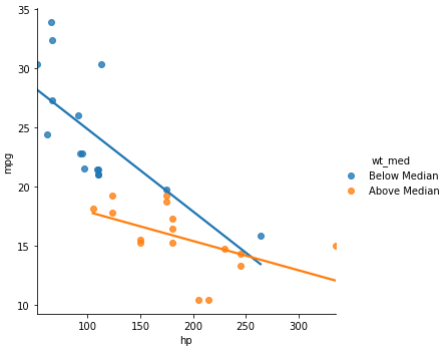
Out[405]: <seaborn.axisgrid.FacetGrid at 0x1e412c10408>



What this plot shows us, is that when the cylinder value is small (i.e. below the median value), the relationship between mpg and wt is strongly negative. Conversely, at higher cylinder values, there is a much weaker relationship between mpg and wt.

In [406]:
```
mtcars['wt_med'] = mtcars.wt > mtcars.wt.median()
mtcars['wt_med'] = np.where(mtcars.wt_med == False, "Below Median", "Above Median")
sns.lmplot(x='hp', y='mpg', hue='wt_med', data=mtcars, ci=None, size=5, aspect=1)
```

Out[406]: <seaborn.axisgrid.FacetGrid at 0x1e40c8aa0c8>



Similarly, with lighter cars(i.e. below the median value), the relationship between hp and mpg is strongly negative. However, with heavier cars, there is a much weaker relationship between mpg and hp!

In [385]:
```
# let us simplify our model
model_interaction = sm.OLS.from_formula('mpg ~ wt + cyl + hp + wt:cyl', data=data).fit()
summary = model_interaction.summary()
summary
#summary.tables[1]
```

Out[385]:

OLS Regression Results

Dep. Variable:	mpg	**R-squared:**	0.879
Model:	OLS	**Adj. R-squared:**	0.861
Method:	Least Squares	**F-statistic:**	49.11
Date:	Sun, 15 Nov 2020	**Prob (F-statistic):**	5.24e-12
Time:	17:14:30	**Log-Likelihood:**	-68.566
No. Observations:	32	**AIC:**	147.1
Df Residuals:	27	**BIC:**	154.5
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	81.5918	11.381	7.169	0.000	58.240	104.943
wt	-35.8162	8.657	-4.137	0.000	-53.580	-18.053
cyl	-5.4696	1.671	-3.274	0.003	-8.898	-2.042
hp	-6.1644	3.027	-2.036	0.052	-12.376	0.047
wt:cyl	3.4690	1.223	2.836	0.009	0.959	5.979

Omnibus:	1.882	**Durbin-Watson:**	2.067
Prob(Omnibus):	0.390	**Jarque-Bera (JB):**	1.502
Skew:	0.522	**Prob(JB):**	0.472
Kurtosis:	2.811	**Cond. No.**	435.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Well ....the AIC of the model is better than the first model that we built (lower values are better). The reduction in the R-squared is negligible and this model still signifies the importance of our predictors in a correct way.

Similarly the coefficients tell us:

For every 1 unit increase in weight, mpg decreases by 35.8 (holding cylinders and hp at 0) or it is better to say for every 1 unit increase in weight, mpg changes by −35.8 + cyl*13.8)

For every 1 unit increase in cylinders, mpg decreases by 21.8 (holding weight and hp at 0), or for every 1 unit increase in cylinders, mpg changes by -21.8+wt*13.8)

For every 1 unit increase in horsepower, mpg decreases by 6.1 (holding weight and cylinder at 0)

At 0 weight, 0 cylinders and 0 horsepower, we expect mpg to be 81.59. which doesnot make any sense for this problem because such cars do not exist

3.2 PyGAM

In [386]:
```
import pandas as pd
from pygam import LinearGAM, s, f, te, l
target_df = pd.Series(mtcars.mpg)
```

In [387]:
```
X = data[['cyl','wt', 'hp']]
y = data.mpg
feature_name= ['cyl','wt', 'hp']
```

3.2.1 Direct Effect

In [388]:
```
gam = LinearGAM().fit(X, y)
```

When we fit a LinearGAM without without providing the features it automatically creates a spline with 20 functions for each predictor. They are represented as S(feature index) in the model summary. In pyGAM, we specify the functional form using terms:
• l() linear terms: for terms like Xi
• s() spline terms
• f() factor terms
• te() tensor products
• intercept.
Note: GAM(..., intercept=True) so models include an intercept by default.

We can specify the number of splines for each feature by ourselves. pyGAM can also fit interactions using tensor products via te().We can treat a predictor also as a factor fpr example if it is approximately continuous, but only takes on specific number of discrete values like 10 or 3.

Model Tuning:
Find the best model requires the tuning of several key parameters including n_splines, lam, and constraints. Among them, lam is of great importance to the performance of GAMs. It controls the strength of the regularization penalty on each term. pyGAM built a grid search function that build a grid to search over multiple lam values so that the model with the lowest generalized cross-validation (GCV) score.

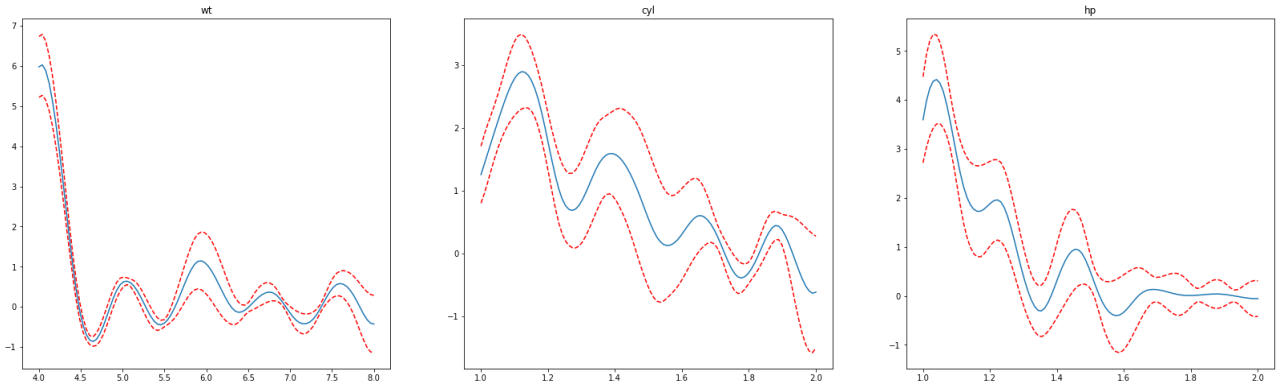
In [389]:
```
gam.summary()
```

```
LinearGAM
=============================================== ==========================================================
Distribution:                        NormalDist Effective DoF:                                     5.5759
Link Function:                     IdentityLink Log Likelihood:                                  -94.6029
Number of Samples:                           32 AIC:                                             202.3577
                                                AICc:                                            206.4372
                                                GCV:                                              10.4672
                                                Scale:                                             7.2458
                                                Pseudo R-Squared:                                    0.83
=========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [0.6]                20                        1.11e-16     ***
s(1)                              [0.6]                20                        1.11e-16     ***
s(2)                              [0.6]                20                        1.11e-16     ***
intercept                                              1                         1.11e-16     ***
=========================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

According to this model, the impact of our 3 predictors (wt, cyl and hp) are significant.

In [390]:
```
## plotting
import matplotlib.pyplot as plt
plt.figure();
plt.rcParams['figure.figsize'] = (28, 8)
fig, axs = plt.subplots(1,3)
titles = data.columns
for i, ax in enumerate(axs):
    XX = gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX, width=.95)[1], c='r', ls='--')
    ax.set_title(titles[i])
```

```
<Figure size 432x288 with 0 Axes>
```

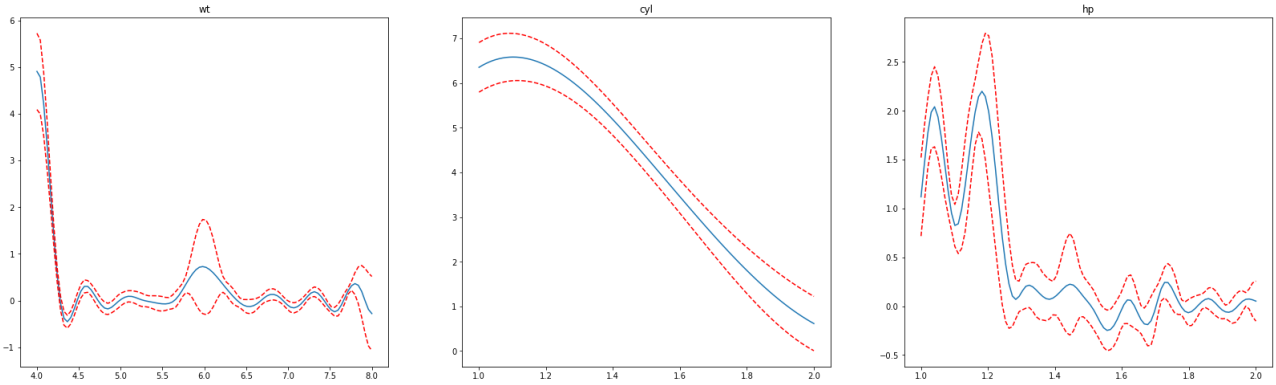


The plots shows that the relationships between our predictors and target variables obviously are not linear. from the domain knowledge we know that the unique values of the Cylender are only 3. So we dont need a huge number of splines for this feature we will decrease the spline values for this feature and we will increase the number of splines for the other two predictors. Knoing that we have 32 features I will set the default value to 32.

In [391]:
```
gam =LinearGAM(s(0, n_splines=32) + s(1, n_splines=5) + s(2, n_splines=32)).fit(X, y)
```

In [392]:
```
import matplotlib.pyplot as plt
plt.figure();
plt.rcParams['figure.figsize'] = (28, 8)
fig, axs = plt.subplots(1,3)
titles = data.columns
for i, ax in enumerate(axs):
    XX = gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX, width=.95)[1], c='r', ls='--')
    ax.set_title(titles[i])
```

```
<Figure size 2016x576 with 0 Axes>
```



As we can see, it seems as the number of cylenders increases the mpg decrease, the relationship is not purly linear though. The other two plots show higher complexity.

3.2.2 Interaction Effect

In [393]:
```
gam =LinearGAM(s(0, n_splines=32) + s(1, n_splines=4) + s(2, n_splines=32) + te(0,1) + te(2,0) + te(1,2)).fit(X, y)
```

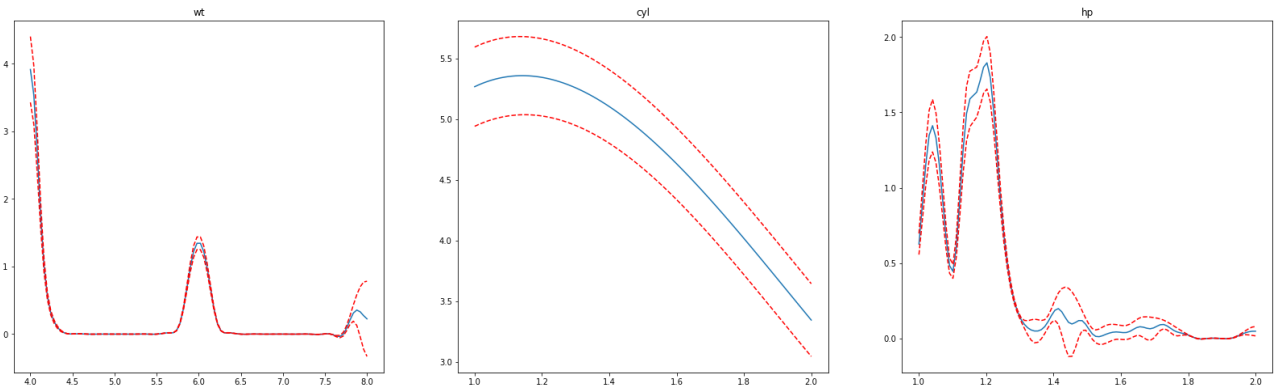
Our gam model specifies that we want a:
• spline function on feature 0, with 5 basis functions, spline function on feature 1 and 2 with the default basis finctions = 20
• tensor spline interaction on features (0 and 1) , (2 and 0), (1 and 2)

In [394]:
```
gam.summary()
```

```
LinearGAM
=============================================== ==========================================================
Distribution:                        NormalDist Effective DoF:                                     1.8252
Link Function:                     IdentityLink Log Likelihood:                                -108.7368
Number of Samples:                           32 AIC:                                            223.1239
                                                AICc:                                           223.8911
                                                GCV:                                             12.7511
                                                Scale:                                           11.449
                                                Pseudo R-Squared:                                 0.6932
==========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [0.6]                32                        1.11e-16     ***
s(1)                              [0.6]                4                         1.11e-16     ***
s(2)                              [0.6]                32                        1.11e-16     ***
te(0, 1)                          [0.6 0.6]            100                       1.11e-16     ***
te(2, 0)                          [0.6 0.6]            100                       1.11e-16     ***
te(1, 2)                          [0.6 0.6]            100                       1.11e-16     ***
intercept                                              1                         1.11e-16     ***
==========================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

In [395]:
```
## plotting
import matplotlib.pyplot as plt
plt.figure();
plt.rcParams['figure.figsize'] = (28, 8)
fig, axs = plt.subplots(1,3)
titles = data.columns
for i, ax in enumerate(axs):
    XX = gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX, width=.95)[1], c='r', ls='--')
    ax.set_title(titles[i])
```

```
<Figure size 2016x576 with 0 Axes>
```



Obviously adding the interaction terms, made the model more complex. The impact of wt is still non linear but it seems it has more predictive intervals. Let us remove the unneccerry interaction terms and check

the results one more time.

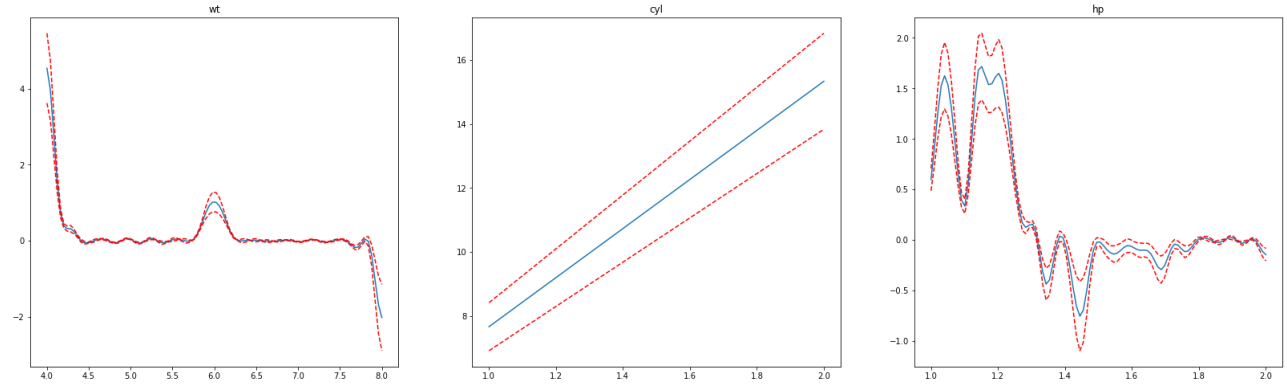
In [396]:
```
gam =LinearGAM(s(0, n_splines=32) + l(1) + s(2, n_splines=32) + te(0,1)).fit(X, y)
```

In [397]:
```
gam.summary()
```

```
LinearGAM
============================================= ==========================================================
Distribution:                        NormalDist Effective DoF:                                     1.9275
Link Function:                     IdentityLink Log Likelihood:                                -129.8894
Number of Samples:                         32 AIC:                                             265.6339
                                              AICc:                                            266.453
                                              GCV:                                             25.3661
                                              Scale:                                           22.6315
                                              Pseudo R-Squared:                                 0.3956
=================================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [0.6]                32                        1.11e-16     ***
l(1)                              [0.6]                1                         1.11e-16     ***
s(2)                              [0.6]                32                        1.11e-16     ***
te(0, 1)                          [0.6 0.6]            100                       1.11e-16     ***
intercept                                              1                         1.11e-16     ***
=================================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

In [398]:
```
## plotting
import matplotlib.pyplot as plt
plt.figure();
plt.rcParams['figure.figsize'] = (28, 8)
fig, axs = plt.subplots(1,3)
titles = data.columns
for i, ax in enumerate(axs):
    XX = gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX, width=.95)[1], c='r', ls='--')
    ax.set_title(titles[i])
```

```
<Figure size 2016x576 with 0 Axes>
```



If we are sure that there is a linear relationship between one of our predictors and our target variables we can model it as a linear predictor through l(). Here, we tested the type of relationship between cyl and pmpg and we wated to see the effect of wrongly assuming it to be a linear relationship. Obviously we got the wrong result here.

3.2.3 Build the Model via Gridsearch

In [399]:
```
from sklearn.metrics import accuracy_score
lambda_ = [0.6, 0.6, 0.6, 0.6, 0.6]
#n_splines = [4, 14, 4, 6, 12, 12]
constraints = [None, None, None, None, None]
gam =LinearGAM(s(0, n_splines=32) + s(1, n_splines=4) + s(2, n_splines=32) + te(0,1), constraints=constraints,
                  lam=lambda_).fit(X, y)
```

We will test this part in a bigger dataset where we can perform train test split.

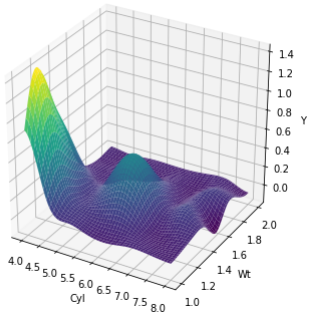
3.2.4 Visualizing the Interactions

3.2.5 partial dependence plot

pyGAM supports partial dependence plot with matplotlib. The partial dependence for each term in a GAM can be visualized with a 95% confidence interval for the estimation function.

In [408]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.ion()
plt.rcParams['figure.figsize'] = (6, 6)
for i in range(3, 4): # we can increase so that we plot more interaction terms currently we have only 1.
    fig = plt.figure()

    XX = gam.generate_X_grid(term=i, meshgrid=True)
    Z = gam.partial_dependence(term=i, X=XX, meshgrid=True)
    ax=plt.axes(projection='3d')
    xLabel = ax.set_xlabel('Cyl')
    yLabel = ax.set_ylabel('Wt')
    zLabel = ax.set_zlabel('Y')
    ax.plot_surface(XX[0], XX[1], Z, cmap='viridis')
```



The vertical height of the surface plot is the expected (predicted) value of y at each combination of Cyl and Wt. The slope of the surface on each edge of the plot is a marginal effect. In other words, the slope on the lefthand face of the plot is the marginal effect of Cyl when Wt==0 (in its lowest value, here it is scaled from 1 to 2). Similarly, the slope on the righthand face of the plot is the marginal effect of Wt when Cyl==1 (in its largest value, here it is 8 Cylinder).

4 Reference

[1]https://readthedocs.org/projects/pygam/downloads/pdf/latest/ (https://readthedocs.org/projects/pygam/downloads/pdf/latest/)
[2]https://pygam.readthedocs.io/en/latest/notebooks/tour_of_pygam.html (https://pygam.readthedocs.io/en/latest/notebooks/tour_of_pygam.html)
[3]https://www.statsmodels.org/dev/examples/notebooks/generated/ols.html (https://www.statsmodels.org/dev/examples/notebooks/generated/ols.html)
[4]https://thomasleeper.com/Rcourse/Tutorials/olsinteractionplots2.html (https://thomasleeper.com/Rcourse/Tutorials/olsinteractionplots2.html)