In [481]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
import statsmodels.formula.api as smf
from patsy import dmatrix

%matplotlib inline
plt.style.use('seaborn-white')

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, L
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error

%matplotlib inline
plt.style.use('seaborn-white')
```

In [2]:
```
df = pd.read_csv("desktop/College.csv")
```

In [3]:
```
df
```

Out[3]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	537
1	Adelphi University	Yes	2186	1924	512	16	29	2683	1227
2	Adrian College	Yes	1428	1097	336	22	50	1036	99
3	Agnes Scott College	Yes	417	349	137	60	89	510	63
4	Alaska Pacific University	Yes	193	146	55	16	44	249	869
...	...	...	...	...	...	...	...	...	...
772	Worcester State College	No	2197	1515	543	4	26	3089	2029
773	Xavier University	Yes	1959	1805	695	24	47	2849	1107
774	Xavier University of Louisiana	Yes	2097	1915	695	34	61	2793	166
775	Yale University	Yes	10705	2453	1317	95	99	5217	83
776	York College of Pennsylvania	Yes	2989	1855	691	28	63	2988	1726

777 rows × 19 columns

In [4]:
```
df1 = df.iloc[:,1:]
```

In [5]:
```
df1['Private'] = df['Private'].map({'No':0, 'Yes':1})
```

In [6]:
```
df1
```

Out[6]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Ro
0	1	1660	1232	721	23	52	2885	537	7440	
1	1	2186	1924	512	16	29	2683	1227	12280	
2	1	1428	1097	336	22	50	1036	99	11250	
3	1	417	349	137	60	89	510	63	12960	
4	1	193	146	55	16	44	249	869	7560	
...	...	...	...	...	...	...	...	...	...	
772	0	2197	1515	543	4	26	3089	2029	6797	
773	1	1959	1805	695	24	47	2849	1107	11520	
774	1	2097	1915	695	34	61	2793	166	6900	
775	1	10705	2453	1317	95	99	5217	83	19840	
776	1	2989	1855	691	28	63	2988	1726	4990	

777 rows × 18 columns

In [7]:
```
df1.corr()
```

Out[7]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Unde
Private	1.000000	-0.432095	-0.475252	-0.567908	0.164132	0.095752	-0.615561	-0.45
Apps	-0.432095	1.000000	0.943451	0.846822	0.338834	0.351640	0.814491	0.39
Accept	-0.475252	0.943451	1.000000	0.911637	0.192447	0.247476	0.874223	0.4
Enroll	-0.567908	0.846822	0.911637	1.000000	0.181294	0.226745	0.964640	0.5
Top10perc	0.164132	0.338834	0.192447	0.181294	1.000000	0.891995	0.141289	-0.1
Top25perc	0.095752	0.351640	0.247476	0.226745	0.891995	1.000000	0.199445	-0.0
F.Undergrad	-0.615561	0.814491	0.874223	0.964640	0.141289	0.199445	1.000000	0.5
P.Undergrad	-0.452088	0.398264	0.441271	0.513069	-0.105356	-0.053577	0.570512	1.0
Outstate	0.552650	0.050159	-0.025755	-0.155477	0.562331	0.489394	-0.215742	-0.2
Room.Board	0.340532	0.164939	0.090899	-0.040232	0.371480	0.331490	-0.068890	-0.0
Books	-0.018549	0.132559	0.113525	0.112711	0.118858	0.115527	0.115550	0.0
Personal	-0.304485	0.178731	0.200989	0.280929	-0.093316	-0.080810	0.317200	0.3
PhD	-0.156714	0.390697	0.355758	0.331469	0.531828	0.545862	0.318337	0.1
Terminal	-0.129620	0.369491	0.337583	0.308274	0.491135	0.524749	0.300019	0.1
S.F.Ratio	-0.472205	0.095633	0.176229	0.237271	-0.384875	-0.294629	0.279703	0.2
perc.alumni	0.414775	-0.090226	-0.159990	-0.180794	0.455485	0.417864	-0.229462	-0.2
Expend	0.258461	0.259592	0.124717	0.064169	0.660913	0.527447	0.018652	-0.0
Grad.Rate	0.336162	0.146755	0.067313	-0.022341	0.494989	0.477281	-0.078773	-0.2

In [8]:
```
X = df1.drop(['Apps'], axis = 1)
Y = df1["Apps"]
```

In [9]:
```
X_train,X_test,Y_train,Y_test = sklearn.model_selection.train_test_split(
    X, Y, test_size=0.3, random_state = 10)
```

In [10]:
```
lr = LinearRegression()
lr.fit(X_train, Y_train)
pred0 = lr.predict(X_test)
print('MSE = ', mean_squared_error(Y_test, pred0))
```

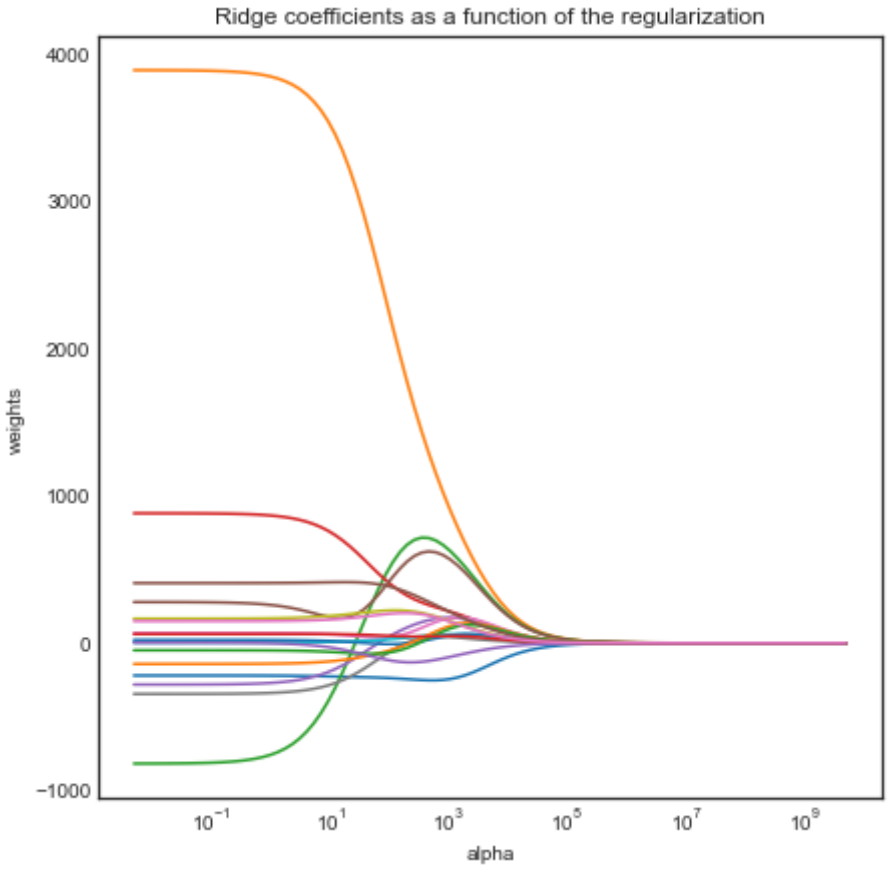
```
MSE =  743341.4398347524
```

In [11]:
```
round(lr.score(X_test, Y_test)*100, 2)
```

Out[11]: 93.37

In [12]:
```
scaler = StandardScaler().fit(X_train)
```

In [13]:
```
#Generate different values of alpha to fit different Ridge models
alphas = 10**np.linspace(10,-2,100)*0.5

ridge = Ridge()
coefs = []

for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(scale(X), Y)
    coefs.append(ridge.coef_)

plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
#ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization');
```



In [14]:
```
# Perform CV and figure out the best alpha
ridgecv = RidgeCV(alphas=alphas, scoring='neg_mean_squared_error')
ridgecv.fit(scale(X_train), Y_train)
print('Ridge alpha =', ridgecv.alpha_)
```

```
Ridge alpha = 0.005
```

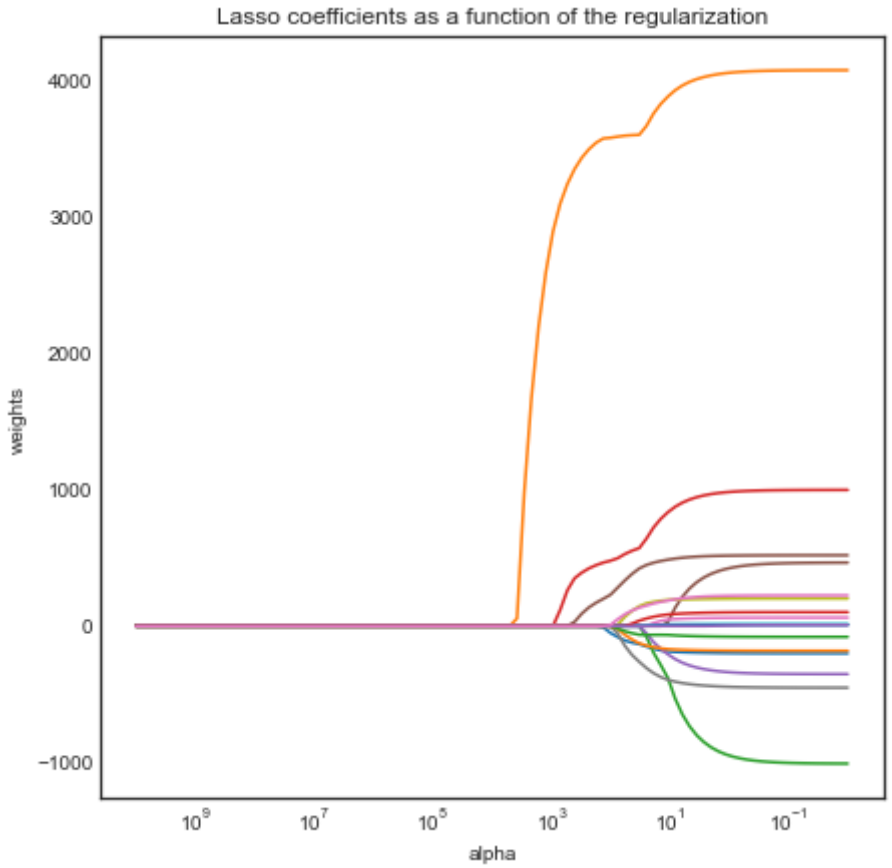
In [15]:
```
ridge1 = Ridge(alpha=0.005)
ridge1.fit(scale(X_train), Y_train)
pred1 = ridge1.predict(scaler.transform(X_test))
print('MSE = ', mean_squared_error(Y_test, pred1))
```

```
MSE =  743280.5282586691
```

In [16]:
```
round(ridge1.score(scale(X_test), Y_test)*100, 2)
```

Out[16]:
```
90.65
```

In [17]:
```
#LASSO
# Generate different values of alpha to fit different Ridge models
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas*2:
    lasso.set_params(alpha=a)
    lasso.fit(scale(X_train), Y_train)
    coefs.append(lasso.coef_)
plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization');
```



In [18]:
```
lassocv = LassoCV(alphas=None, cv=10, max_iter=10000)
lassocv.fit(scale(X_train), Y_train.values.ravel())
```

Out[18]:
```
LassoCV(cv=10, max_iter=10000)
```

In [19]:
```
lassocv.alpha_
```

Out[19]:
```
16.540428244909734
```

In [20]:
```
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(scale(X_train), Y_train)
pred2 = lasso.predict(scaler.transform(X_test))
print('MSE = ', mean_squared_error(Y_test, pred2))
```

```
MSE =  689601.1948271467
```

In [21]:
```
round(lasso.score(scale(X_test), Y_test)*100, 2)
```

Out[21]:
```
91.46
```

In [22]:
```
pd.Series(lasso.coef_, index=X.columns)
```

Out[22]:
```
Private        -172.065590
Accept         3787.763267
Enroll         -239.428559
Top10perc       752.753636
Top25perc      -141.926723
F.Undergrad       0.000000
P.Undergrad      29.801080
Outstate       -358.951178
Room.Board      181.069763
Books             6.664191
Personal          5.752569
PhD            -158.677325
Terminal        -64.515531
S.F.Ratio        72.771253
perc.alumni      -0.000000
Expend          470.057478
Grad.Rate       171.094097
dtype: float64
```

14 of the coefficients are non-zero, only 2 coefficients was shrunken to 0.

In [23]:
```
#PCA

pca = PCA()
X_reduced = pca.fit_transform(scale(X))

print(pca.components_.shape)  #   Loadings
pd.DataFrame(pca.components_.T).loc[:4,:5]
```

```
(17, 17)
```

Out[23]:

	0	1	2	3	4	5
0	0.202812	-0.319559	0.149372	0.207191	-0.031502	-0.036004
1	0.013150	0.419033	-0.027107	0.362324	-0.112086	0.002294
2	-0.028705	0.442952	-0.028317	0.250415	-0.175489	-0.053479
3	0.344736	0.130412	0.005602	-0.221019	-0.331967	-0.059989
4	0.318675	0.161423	-0.057655	-0.252482	-0.344360	0.020636

In [24]:
```
print(X_reduced.shape) #   Principal Components
pd.DataFrame(X_reduced).loc[:4,:5]
```

```
(777, 17)
```

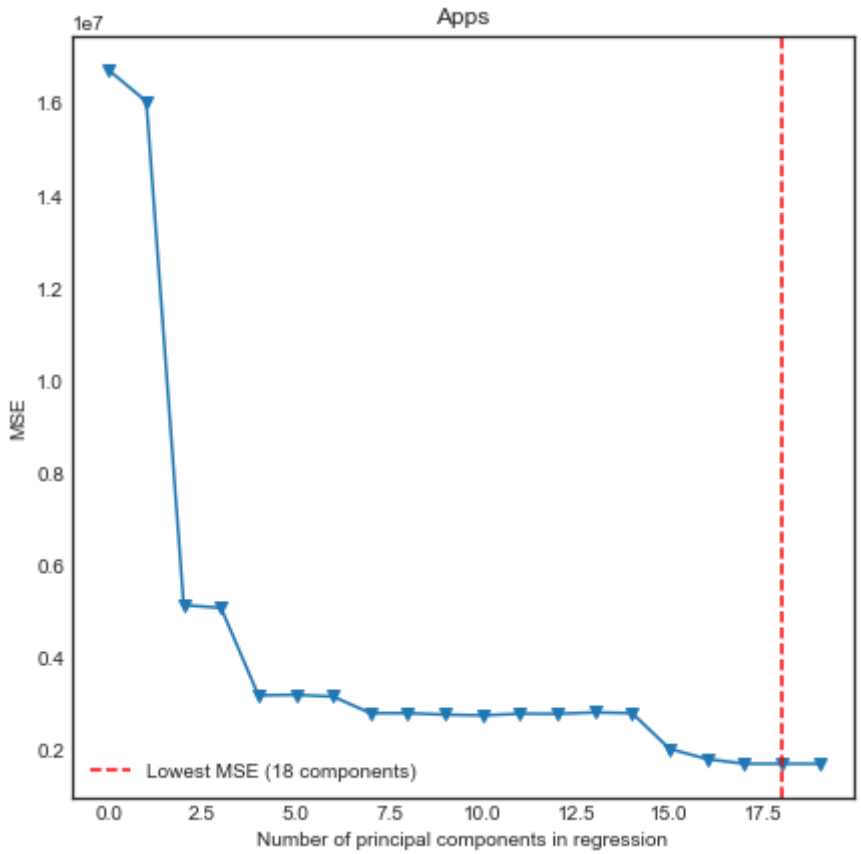
Out[24]:

	0	1	2	3	4	5
0	-1.593338	-0.230948	-0.101716	-0.849744	-0.275266	-0.369145
1	-1.469387	-2.027180	2.897925	3.060121	-0.555634	-0.084863
2	-0.597659	-1.813161	-0.293113	0.404538	-0.592998	-0.964864
3	3.836386	-0.543832	-0.194869	-1.293558	0.448208	-1.048643
4	-1.759793	-1.062573	2.237288	-1.141496	1.300800	0.043282

In [25]:
```
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

Out[25]:
```
array([31.67, 57.3 , 64.3 , 69.91, 75.39, 80.38, 83.99, 87.4 , 90.5 ,
       92.9 , 95.  , 96.8 , 97.89, 98.74, 99.35, 99.83, 99.99])
```

In [27]:
```
pca = PCA()
X_reduced_train = pca.fit_transform(scale(X_train))
n = len(X_reduced_train)
regr = LinearRegression()

# 10-fold CV, with shuffle
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)

mse = []

# Calculate MSE with only the intercept (no principal components in regress
score = -1*cross_val_score(regr, np.ones((n,1)), Y_train, cv=kf_10, scoring
mse.append(score)

# Calculate MSE using CV for the 19 PCs, adding one component at the time.
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X_reduced_train[:,:i], Y_train, cv=kf_
    mse.append(score)
plt.figure(figsize=(7, 7))
plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Apps')
plt.axvline(18, linestyle="--", color="r", label="Lowest MSE (18 components
plt.legend(loc='best')
plt.xlim(xmin=-1);
```



In [28]:
```
X_reduced_test = pca.transform(scale(X_test))
```

In [29]:
```
# Train regression model on training data
regr = LinearRegression()
regr.fit(X_reduced_train, Y_train)

# Prediction with test data
pred3 = regr.predict(X_reduced_test)
print('MSE = ', mean_squared_error(Y_test, pred3))
```

```
MSE =  1048191.1913548651
```

In [30]:
```
round(regr.score(X_reduced_test, Y_test)*100, 2)
```

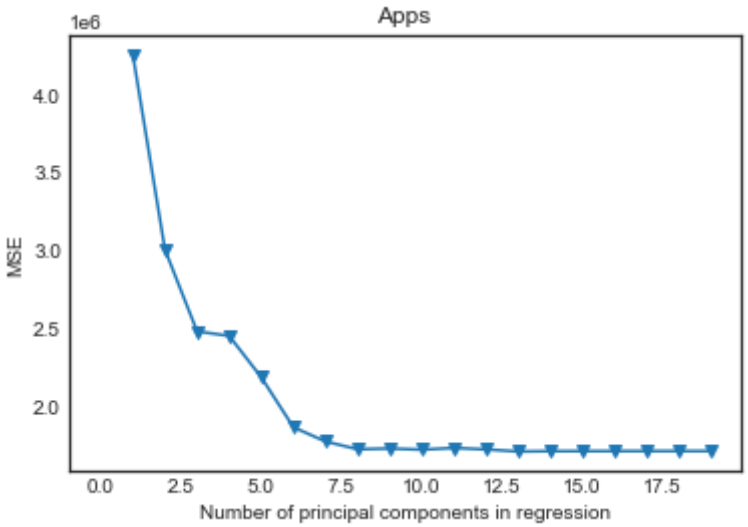
Out[30]:
```
90.65
```

In [31]:
```
n = len(X_train)

# 10-fold CV, with shuffle
kf_10 = KFold(n_splits=10, shuffle=True, random_state=0)

mse = []

for i in np.arange(1, 20):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(pls, scale(X_train), Y_train, cv=kf_10, scoring
    mse.append(-score)

plt.plot(np.arange(1, 20), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Apps')
plt.xlim(xmin=-1);
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
```

```
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(18) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/c
```

```
ross_decomposition/_pls.py:185: FutureWarning: As of version 0.24, n_comp
onents(19) should be in [1, n_features].n_components=17 will be used inst
ead. In version 1.1 (renaming of 0.26), an error will be raised.
  warnings.warn(
```



In [32]:
```
pls = PLSRegression(n_components=2)
pls.fit(scale(X_train), Y_train)

mean_squared_error(Y_test, pls.predict(scale(X_test)))
```

Out[32]:
```
1501477.6248559698
```

In [33]:
```
round(pls.score(scale(X_test), Y_test)*100, 2)
```

Out[33]:
```
86.6
```

The Linear regression had the highest R squared of 93.46%, which COULD imply that it best explains our model, however, considering that the linear regression does not shrink coefficients or reduce dimensionality, the high R squared could be due to higher number of variables. My conlusion is that the LASSO regression performed best since it achieved the second highest R squared of 91.46% and has the lowest MSE. The difference in MSEs among the linear, ridge and lasso regressions is not big, and the difference in MSEs among PCR and PLS is not big, however, PCR and PLS had a much bigger MSE whe compared to the rest of the models.

Question 6.11

In [34]:
```
df2 = pd.read_csv("Desktop/Boston.csv")
```

In [35]:
```
df2
```

Out[35]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat	me
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	2
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	2
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	3
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	3
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
501	502	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	9.67	2
502	503	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	9.08	2
503	504	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	2
504	505	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	2
505	506	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	1

506 rows × 14 columns

In [36]:
```
df3 = df2.iloc[:,1:]
```

In [37]:
```
df3
```

Out[37]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	11.9

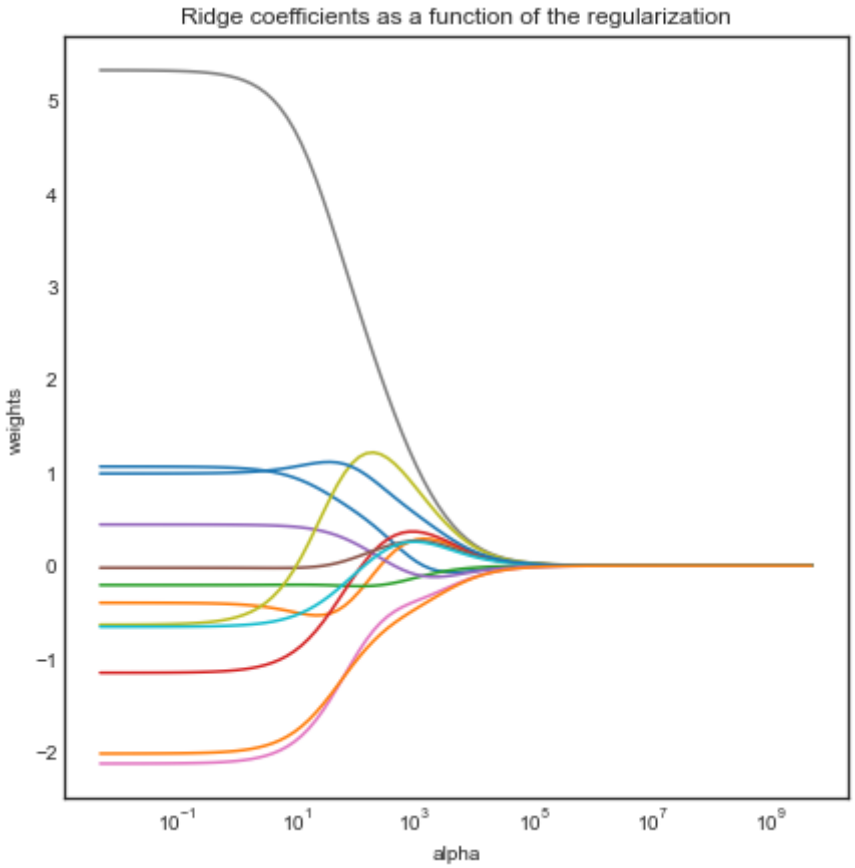
506 rows × 13 columns

In [38]:
```
X1 = df3.drop(['crim'], axis = 1)
Y1 = df3["crim"]
```

In [39]:
```
X1_train,X1_test,Y1_train,Y1_test = sklearn.model_selection.train_test_spli
    X1, Y1, test_size=0.3, random_state = 10)
```

In [40]:
```
scaler1 = StandardScaler().fit(X1_train)
```

In [41]:
```
#Generate different values of alpha to fit different Ridge models
alphas = 10**np.linspace(10,-2,100)*0.5

ridge = Ridge()
coefs = []

for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(scale(X1), Y1)
    coefs.append(ridge.coef_)

plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
#ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization');
```



In [42]:
```
# Perform CV and figure out the best alpha
ridgecv = RidgeCV(alphas=alphas, scoring='neg_mean_squared_error')
ridgecv.fit(scale(X1_train), Y1_train)
print('Ridge alpha =', ridgecv.alpha_)
```

```
Ridge alpha = 7.087370814634009
```

In [44]:
```
# Estimate the Ridge Model using the alpha above
ridge2 = Ridge(alpha=7.087370814634009)
ridge2.set_params(alpha=7.087370814634009)
ridge2.fit(scale(X1_train), Y1_train)
mean_squared_error(Y1_test, ridge2.predict(scale(X1_test)))
print('MSE = ', mean_squared_error(Y1_test, ridge2.predict(scale(X1_test))))
pd.Series(ridge2.coef_.flatten(), index=X1.columns)
```

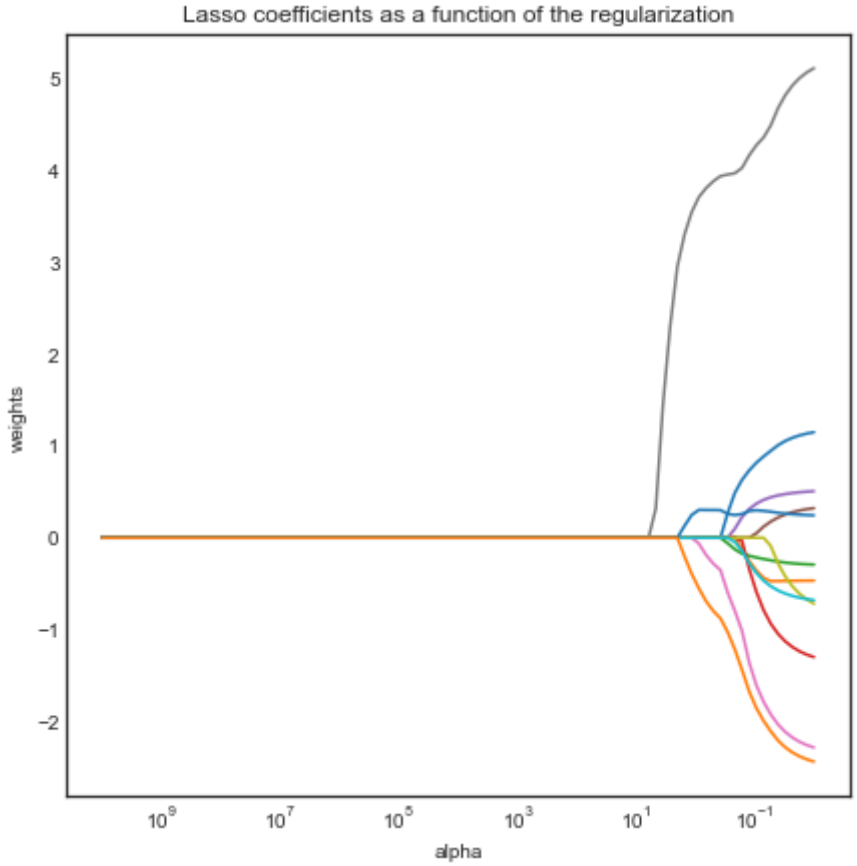
```
MSE =  47.78157933705228
```

Out[44]:
```
zn          1.031919
indus      -0.530225
chas       -0.296624
nox        -1.094437
rm          0.527493
age         0.307772
dis        -2.051020
rad         4.553490
tax        -0.209772
ptratio    -0.581560
lstat       0.377994
medv       -2.200858
dtype: float64
```

In [45]:
```
round(ridge2.score(scale(X1_test), Y1_test)*100, 2)
```

Out[45]:
```
47.58
```

In [46]:
```
#LASSO
# Generate different values of alpha to fit different Ridge models
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas*2:
    lasso.set_params(alpha=a)
    lasso.fit(scale(X1_train), Y1_train)
    coefs.append(lasso.coef_)
plt.figure(figsize=(7, 7))
ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization');
```



In [47]:
```
lassocv1 = LassoCV(alphas=None, cv=10, max_iter=10000)
lassocv1.fit(scale(X1_train), Y1_train.values.ravel())
```

Out[47]:
```
LassoCV(cv=10, max_iter=10000)
```

In [48]:
```
lassocv1.alpha_
```

Out[48]:
```
0.04625883390498626
```

In [49]:
```
pd.Series(lasso.coef_, index=X1.columns)
```

Out[49]:
```
zn          1.148639
indus      -0.466316
chas       -0.292333
nox        -1.299125
rm          0.506616
age         0.321591
dis        -2.283316
rad         5.110412
tax        -0.717959
ptratio    -0.678782
lstat       0.244385
medv       -2.436469
dtype: float64
```

In [52]:
```
lasso.set_params(alpha=0.04625883390498626)
lasso.fit(scale(X1_train), Y1_train)
print('MSE = ',mean_squared_error(Y1_test, lasso.predict(scale(X1_test))))
```

```
MSE =  47.62754303473997
```

In [53]:
```
round(lasso.score(scale(X1_test), Y1_test)*100, 2)
```

Out[53]:
```
47.75
```

In [54]:
```
#PCA

pca = PCA()
X1_reduced = pca.fit_transform(scale(X1))

print(pca.components_.shape)  #   Loadings
pd.DataFrame(pca.components_.T).loc[:4,:5]
```

```
(12, 12)
```

Out[54]:

	0	1	2	3	4	5
0	0.269817	-0.119615	-0.344498	0.317929	0.487210	-0.274547
1	-0.352246	0.116128	-0.022672	0.025875	0.053987	0.302961
2	0.000655	0.404474	0.234328	0.834655	-0.271635	-0.057294
3	-0.343119	0.245898	0.027899	-0.025013	0.271594	0.099482
4	0.220433	0.457233	-0.347586	-0.226620	-0.110094	-0.461634

In [55]:
```
print(X1_reduced.shape) #   Principal Components
pd.DataFrame(X1_reduced).loc[:4,:5]
```

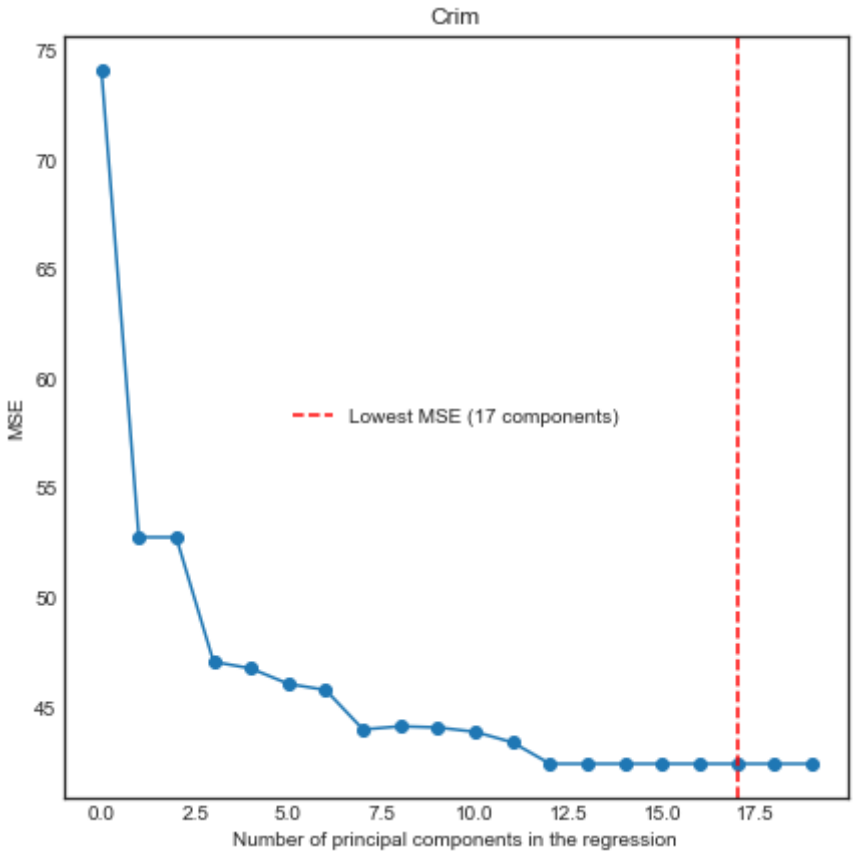
```
(506, 12)
```

Out[55]:

	0	1	2	3	4	5
0	1.990085	0.484733	0.646415	-0.581191	0.742116	0.050567
1	1.224254	-0.229886	0.972351	-0.651369	-0.369496	-0.398419
2	2.296435	0.918381	0.127189	-1.065436	-0.838976	-0.010040
3	2.783216	0.215875	-0.049407	-0.813701	-1.234901	0.072348
4	2.710595	0.453556	-0.049125	-0.907683	-1.174403	-0.276861

In [56]:
```
# 10-fold CV, with shuffle
n = len(X1_reduced)
kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)

regr = LinearRegression()
mse = []

# Calculate MSE with only the intercept (no principal components in regress
score = -1*cross_val_score(regr, np.ones((n,1)), Y1.ravel(), cv=kf_10, scor
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one compon
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X1_reduced[:,:i], Y1.ravel(), cv=kf_10
    mse.append(score)
plt.figure(figsize=(7, 7))
plt.plot(mse, '-o')
plt.xlabel('Number of principal components in the regression')
plt.ylabel('MSE')
plt.title('Crim')
plt.xlim(xmin=-1);
plt.axvline(17, linestyle="--", color="r", label="Lowest MSE (17 components
plt.legend(loc='center')
```

Out[56]: <matplotlib.legend.Legend at 0x7f9cb0a51b80>



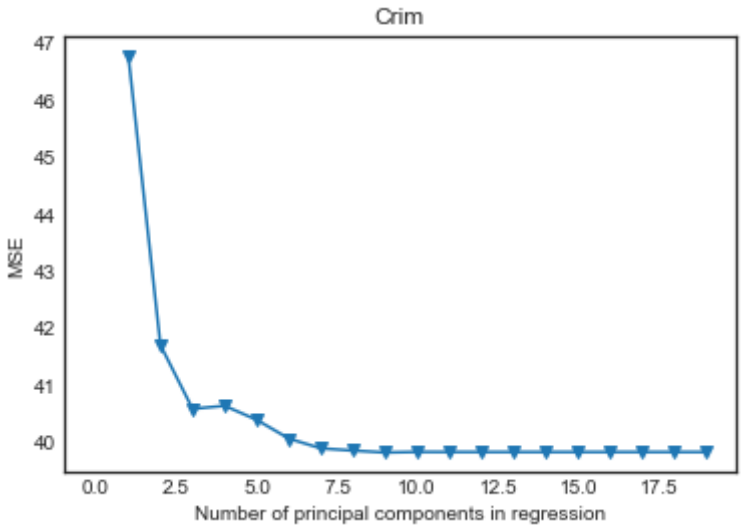
In [57]:
```
# Train regression model on training data
regr = LinearRegression()
regr.fit(X1_reduced, Y1)

# Prediction with test data
pred99 = regr.predict(X1_reduced)
print('MSE = ', mean_squared_error(Y1, pred99))
```

```
MSE =  40.66109406395015
```

In [58]:
```
round(regr.score(X1_reduced, Y1)*100, 2)
```

Out[58]:
```
44.93
```

In [59]:
```
n = len(X1_train)

# 10-fold CV, with shuffle
kf_10 = KFold(n_splits=10, shuffle=True, random_state=0)

mse = []

for i in np.arange(1, 20):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(pls, scale(X1_train), Y1_train, cv=kf_10, scori
    mse.append(-score)

plt.plot(np.arange(1, 20), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Crim')
plt.xlim(xmin=-1);
```



In [60]:
```
pls = PLSRegression(n_components=2)
pls.fit(scale(X1_train), Y1_train)

mean_squared_error(Y1_test, pls.predict(scale(X1_test)))
```

Out[60]: 49.46914314129166

In [61]:
```
round(pls.score(scale(X1_test), Y1_test)*100, 2)
```

Out[61]: 45.73

ccording to all the results obtained from the four models ran, LASSO seems to have highest R-squared. All the MSEs are very close to each other except for the PCR, which had the lowest MSE of approximately 40. Based on the results, I would consider PCR at 7 components which has an MSE just below 45 and I would also consider LASSO since it has the highest R-Squared and its MSE is not too far from the PCR model.

In [62]:
```
# Step 1: Use CV to get the best alpha
enetcv = ElasticNetCV(cv=10, max_iter=10000)
enetcv.fit(scale(X1_train), Y1_train.values.ravel())

# Step 2: Estimate the model w/ best alpha
enet_best = ElasticNet(alpha=enetcv.alpha_)
enet_best.fit(scale(X1_train), Y1_train)

# Step 3: Print model estimates
print(list(zip(enet_best.coef_, X1)))

# Step 4: Print Error Metrics
print('R^2 Test Set', round(enet_best.score(scale(X1_test), Y1_test)*100, 2
```

```
[(0.946480930126323, 'zn'), (-0.5060977386580986, 'indus'), (-0.276050124
2893158, 'chas'), (-0.923994646775656, 'nox'), (0.4926047895881885, 'r
m'), (0.2425961852813953, 'age'), (-1.883123640154613, 'dis'), (4.3195748
60194322, 'rad'), (-0.0, 'tax'), (-0.5085866124691384, 'ptratio'), (0.389
0931594901945, 'lstat'), (-2.053358993157858, 'medv')]
R^2 Test Set 47.63
```

In [63]:
```
mean_squared_error(Y1_test, enet_best.predict(scale(X1_test)))
```

Out[63]: 47.73916958991018

After running an elastic net, I will choose LASSO as the best performing model since the bias/ MSE trade off seems best to me. Note: I would choose elastic net as second best performing model since the R-squared and MSE are very slightly lower than those of LASSO and elastic net was able to shrink one variable to 0.

7.7

In [64]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
import statsmodels.formula.api as smf
from patsy import dmatrix
```

In [136]:
```
df11 = pd.read_csv("desktop/Wage.csv")
```

In [137]:
```
df11
```

Out[137]:

	year	age	maritl	race	education	region	jobclass	health	health_ins	logwage	
0	2006	18	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	
1	2004	24	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	
2	2003	45	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	1
3	2003	43	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	1
4	2005	50	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	
...	...	...	...	...	...	...	...	...	...	...	
2995	2008	44	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	5.041393	1
2996	2007	30	2. Married	1. White	2. HS Grad	2. Middle Atlantic	1. Industrial	2. >=Very Good	2. No	4.602060	
2997	2005	27	2. Married	2. Black	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.193125	
2998	2005	27	1. Never Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	4.477121	
2999	2009	55	5. Separated	1. White	2. HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.505150	

3000 rows × 11 columns

In [149]:
```
df11['education'].describe()
```

Out[149]:
```
count            3000
unique              5
top       2. HS Grad
freq              971
Name: education, dtype: object
```

In [142]:
```
df11['maritl'] = df11['maritl'].map({'1. Never Married':1, '2. Married':2,
```

In [147]:
```
df11['race'] = df11['race'].map({'1. White':1, '2. Black':2, '3. Asian':3,
```

In [155]:
```
df11['education'] = df11['education'].map({'1. < HS Grad':1, '2. HS Grad':2
                                           '4. College Grad':4,'5. Advanced
```

In [157]:
```
df11['jobclass'] = df11['jobclass'].map({'1. Industrial':1, '2. Information
```

In [159]:
```
df11['health'] = df11['health'].map({'1. <=Good':1, '2. >=Very Good':2})
```

In [161]:
```
df11['health_ins'] = df11['health_ins'].map({'1. Yes':1, '2. No':2})
```

In [162]:
```
df11
```

Out[162]:

	year	age	maritl	race	education	region	jobclass	health	health_ins	logwage	wage
0	2006	18	1	1	1	2. Middle Atlantic	1	1	2	4.318063	75.04315
1	2004	24	1	1	4	2. Middle Atlantic	2	2	2	4.255273	70.47602
2	2003	45	2	1	3	2. Middle Atlantic	1	1	1	4.875061	130.98217
3	2003	43	2	3	4	2. Middle Atlantic	2	2	1	5.041393	154.68529
4	2005	50	4	1	2	2. Middle Atlantic	2	1	1	4.318063	75.04315
...	...	...	...	...	...	...	...	...	...	...	...
2995	2008	44	2	1	3	2. Middle Atlantic	1	2	1	5.041393	154.68529
2996	2007	30	2	1	2	2. Middle Atlantic	1	2	2	4.602060	99.68946
2997	2005	27	2	2	1	2. Middle Atlantic	1	1	2	4.193125	66.22940
2998	2005	27	1	1	3	2. Middle Atlantic	1	2	1	4.477121	87.98103
2999	2009	55	5	1	2	2. Middle Atlantic	1	1	1	4.505150	90.48191

3000 rows × 11 columns

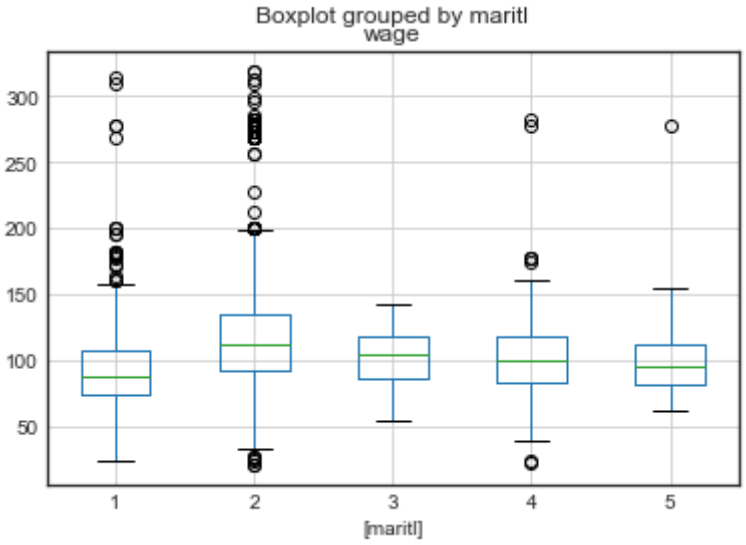
In [163]:
```
df11.corr()
```

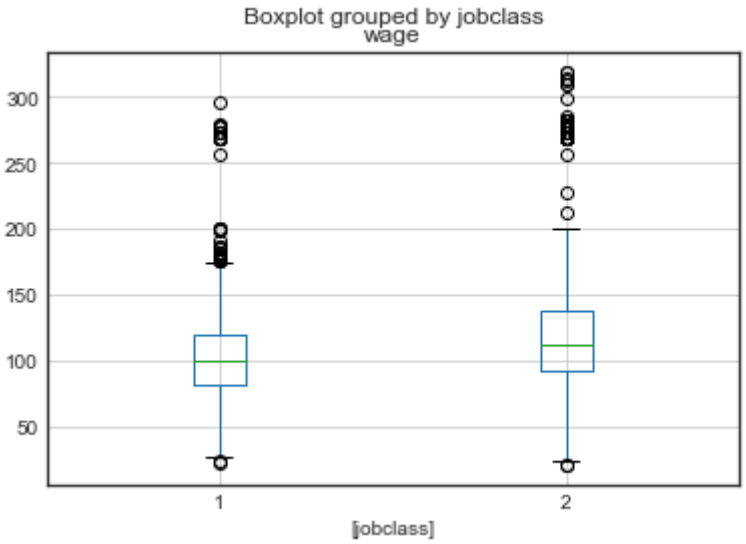
Out[163]:

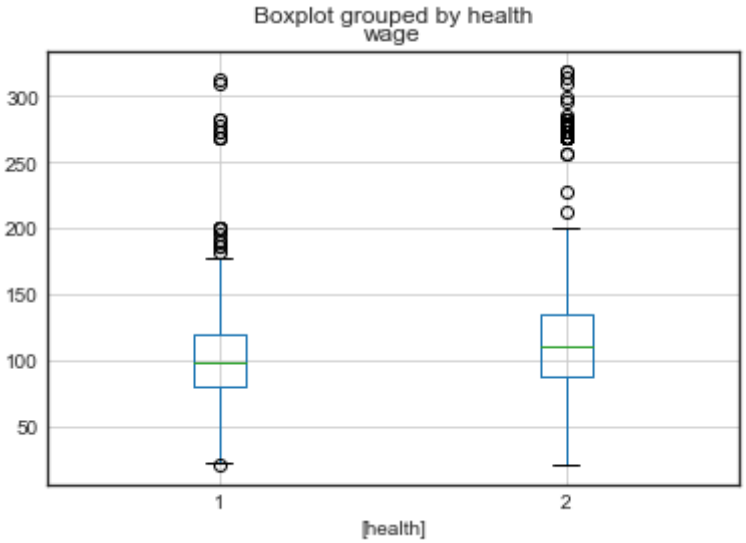
	year	age	maritl	race	education	jobclass	health	health_ins
year	1.000000	0.038425	-0.011493	0.033528	0.014116	-0.006155	-0.001938	0.008091
age	0.038425	1.000000	0.315966	-0.018970	0.070784	0.090691	-0.138907	-0.142589
maritl	-0.011493	0.315966	1.000000	-0.048012	-0.024811	0.026438	-0.044540	-0.057831
race	0.033528	-0.018970	-0.048012	1.000000	0.035474	0.057929	-0.022073	0.040772
education	0.014116	0.070784	-0.024811	0.035474	1.000000	0.303698	0.174534	-0.208461
jobclass	-0.006155	0.090691	0.026438	0.057929	0.303698	1.000000	0.067030	-0.149190
health	-0.001938	-0.138907	-0.044540	-0.022073	0.174534	0.067030	1.000000	-0.076437
health_ins	0.008091	-0.142589	-0.057831	0.040772	-0.208461	-0.149190	-0.076437	1.000000
logwage	0.076239	0.217889	0.085448	-0.032701	0.472849	0.205408	0.158280	-0.369733
wage	0.065544	0.195637	0.067360	-0.026865	0.475775	0.206897	0.152337	-0.308310

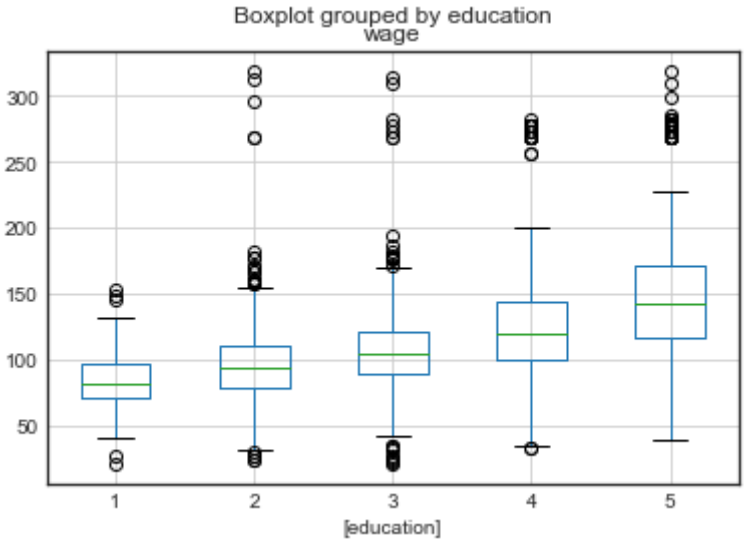
In [164]:
```
df11.boxplot(column=['wage'], by = ['maritl'])
df11.boxplot(column=['wage'], by = ['jobclass'] )
df11.boxplot(column=['wage'], by = ['race'] )
df11.boxplot(column=['wage'], by = ['health'])
df11.boxplot(column=['wage'], by = ['education'])
```

Out[164]: <AxesSubplot:title={'center':'wage'}, xlabel='[education]'>









1) Wage vs marital status, It seems that around a wage of 100, the marital status doesnt affect the wage much. However when we look at wages above 200 we clearly see that married and never married earn most. 2) Information job class earns higher wages than industrial job class. 3) Wage vs Race, it seems that around wage of 100, there not much difference across races, except for other races seems to earn a little lower. However, looking at wages of 200+, we clearly observe that the white race earns higher, possibly due to wage discrimination. 4) Individuals with very good health earn higher wages compated to those with good health, however, several individuals with good health are able to earn as much as individuals with very good health 5) The more education a person achieves, the higher their wage would be, however, there are some outliers who do not have college or advanced degrees but were able to earn as much as indiviuals with advanced degrees.

Based on correlation and boxplots, I will choose maritl, job class and education as predictors.

In [180]:
```
X10 = PolynomialFeatures(1).fit_transform(df11.jobclass.values.reshape(-1,1
X20 = PolynomialFeatures(2).fit_transform(df11.jobclass.values.reshape(-1,1
X30 = PolynomialFeatures(3).fit_transform(df11.jobclass.values.reshape(-1,1
X40 = PolynomialFeatures(4).fit_transform(df11.jobclass.values.reshape(-1,1
X50 = PolynomialFeatures(5).fit_transform(df11.jobclass.values.reshape(-1,1
Y1 = df11['wage']
```

In [166]:
```
fit1 = sm.GLS(df11.wage, X10).fit()
fit1.summary().tables[1]
```

Out[166]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	86.0495	2.338	36.811	0.000	81.466	90.633
x1	17.2716	1.492	11.579	0.000	14.347	20.196

In [168]:
```
fit2 = sm.GLS(df11.wage, X20).fit()
fit2.summary().tables[1]
```

Out[168]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	68.8661	1.083	63.576	0.000	66.742	70.990
x1	43.0468	0.525	81.967	0.000	42.017	44.077
x2	-8.5917	0.645	-13.321	0.000	-9.856	-7.327

In [169]:
```
fit3 = sm.GLS(df11.wage, X30).fit()
fit3.summary().tables[1]
```

Out[169]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	51.3562	0.641	80.149	0.000	50.100	52.613
x1	42.0740	0.501	83.940	0.000	41.091	43.057
x2	23.5097	0.226	103.981	0.000	23.066	23.953
x3	-13.6188	0.354	-38.512	0.000	-14.312	-12.925

In [170]:
```
fit4 = sm.GLS(df11.wage, X40).fit()
fit4.summary().tables[1]
```

Out[170]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	38.8362	0.435	89.362	0.000	37.984	39.688
x1	35.3416	0.391	90.422	0.000	34.575	36.108
x2	28.3524	0.304	93.418	0.000	27.757	28.947
x3	14.3739	0.131	110.113	0.000	14.118	14.630
x4	-13.5830	0.228	-59.660	0.000	-14.029	-13.137

In [171]:
```
fit5 = sm.GLS(df11.wage, X50).fit()
fit5.summary().tables[1]
```

Out[171]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	30.2417	0.321	94.251	0.000	29.613	30.871
x1	28.8710	0.305	94.566	0.000	28.272	29.470
x2	26.1297	0.274	95.299	0.000	25.592	26.667
x3	20.6469	0.212	97.390	0.000	20.231	21.063
x4	9.6814	0.088	109.659	0.000	9.508	9.855
x5	-12.2496	0.164	-74.889	0.000	-12.570	-11.929

In [172]:
```
X11 = PolynomialFeatures(1).fit_transform(df11.maritl.values.reshape(-1,1))
X21 = PolynomialFeatures(2).fit_transform(df11.maritl.values.reshape(-1,1))
X31 = PolynomialFeatures(3).fit_transform(df11.maritl.values.reshape(-1,1))
X41 = PolynomialFeatures(4).fit_transform(df11.maritl.values.reshape(-1,1))
X51 = PolynomialFeatures(5).fit_transform(df11.maritl.values.reshape(-1,1))
Y1 = df11['wage']
```

In [173]:
```
fit6 = sm.GLS(df11.wage, X11).fit()
fit6.summary().tables[1]
```

Out[173]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	104.8443	2.005	52.284	0.000	100.912	108.776
x1	3.4620	0.937	3.697	0.000	1.626	5.298

In [174]:
```
fit7 = sm.GLS(df11.wage, X21).fit()
fit7.summary().tables[1]
```

Out[174]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	53.9607	4.294	12.567	0.000	45.542	62.380
x1	49.1975	3.557	13.830	0.000	42.223	56.172
x2	-8.6667	0.652	-13.300	0.000	-9.944	-7.389

In [175]:
```
fit8 = sm.GLS(df11.wage, X31).fit()
fit8.summary().tables[1]
```

Out[175]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	15.7602	8.418	1.872	0.061	-0.745	32.265
x1	109.6620	12.012	9.129	0.000	86.109	133.215
x2	-36.1462	5.257	-6.876	0.000	-46.453	-25.839
x3	3.5335	0.671	5.268	0.000	2.218	4.849

In [176]:
```
fit9 = sm.GLS(df11.wage, X41).fit()
fit9.summary().tables[1]
```

Out[176]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-144.1233	94.381	-1.527	0.127	-329.182	40.935
x1	421.5431	183.767	2.294	0.022	61.221	781.865
x2	-232.4193	115.521	-2.012	0.044	-458.927	-5.912
x3	51.7715	28.370	1.825	0.068	-3.855	107.398
x4	-4.0373	2.374	-1.701	0.089	-8.692	0.617

In [177]:
```
fit10 = sm.GLS(df11.wage, X51).fit()
fit10.summary().tables[1]
```

Out[177]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.0350	21.287	0.471	0.637	-31.703	51.773
x1	69.5483	16.466	4.224	0.000	37.262	101.835
x2	56.6276	21.997	2.574	0.010	13.496	99.759
x3	-57.4240	23.583	-2.435	0.015	-103.665	-11.183
x4	15.2325	6.792	2.243	0.025	1.915	28.550
x5	-1.2847	0.611	-2.104	0.035	-2.482	-0.087

In [179]:
```
X12 = PolynomialFeatures(1).fit_transform(df11.education.values.reshape(-1,
X22 = PolynomialFeatures(2).fit_transform(df11.education.values.reshape(-1,
X32 = PolynomialFeatures(3).fit_transform(df11.education.values.reshape(-1,
X42 = PolynomialFeatures(4).fit_transform(df11.education.values.reshape(-1,
X52 = PolynomialFeatures(5).fit_transform(df11.education.values.reshape(-1,
Y1 = df11['wage']
```

In [181]:
```
fit11 = sm.GLS(df11.wage, X12).fit()
fit11.summary().tables[1]
```

Out[181]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	62.5444	1.790	34.941	0.000	59.035	66.054
x1	16.3320	0.551	29.617	0.000	15.251	17.413

In [182]:
```
fit12 = sm.GLS(df11.wage, X22).fit()
fit12.summary().tables[1]
```

Out[182]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	83.2130	4.236	19.646	0.000	74.908	91.518
x1	0.5947	2.977	0.200	0.842	-5.242	6.432
x2	2.5339	0.471	5.379	0.000	1.610	3.458

In [183]:
```
fit13 = sm.GLS(df11.wage, X32).fit()
fit13.summary().tables[1]
```

Out[183]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	67.7339	8.628	7.850	0.000	50.816	84.651
x1	20.2345	9.992	2.025	0.043	0.642	39.827
x2	-4.6826	3.536	-1.324	0.186	-11.617	2.252
x3	0.7925	0.385	2.059	0.040	0.038	1.547

After running fits to test the statistical signficance of the polynomial degrees of our predictors, Job class is statistically significant up to 5 degrees, Marital status is statistically significant up to a cubic degree, and education is statistically significant up to 1 degree.

In [196]:
```
mod1 = smf.gls(formula='wage ~ X50 + X12 +X31', data=df11).fit()
print(mod1.summary())
```

```
                            GLS Regression Results
==============================================================================
Dep. Variable:                   wage   R-squared:                       0.281
Model:                            GLS   Adj. R-squared:                  0.280
Method:                 Least Squares   F-statistic:                     234.4
Date:                Wed, 04 May 2022   Prob (F-statistic):          1.06e-211
Time:                        15:54:51   Log-Likelihood:                -14954.
No. Observations:                3000   AIC:                         2.992e+04
Df Residuals:                    2994   BIC:                         2.996e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -3.9078      1.177     -3.319      0.001      -6.216      -1.599
X50[0]        -3.9078      1.177     -3.319      0.001      -6.216      -1.599
X50[1]        -3.7245      1.123     -3.315      0.001      -5.927      -1.522
X50[2]        -3.3579      1.016     -3.306      0.001      -5.349      -1.366
X50[3]        -2.6247      0.800     -3.281      0.001      -4.193      -1.056
X50[4]        -1.1584      0.369     -3.138      0.002      -1.882      -0.435
X50[5]         1.7744      0.495      3.585      0.000       0.804       2.745
X12[0]        -3.9078      1.177     -3.319      0.001      -6.216      -1.599
X12[1]        15.0889      0.561     26.876      0.000      13.988      16.190
X31[0]        -3.9078      1.177     -3.319      0.001      -6.216      -1.599
X31[1]       103.4963     10.559      9.802      0.000      82.792     124.200
X31[2]       -35.5135      4.620     -7.687      0.000     -44.573     -26.454
X31[3]         3.6713      0.590      6.227      0.000       2.515       4.827
==============================================================================
Omnibus:                     1040.541   Durbin-Watson:
```

```
1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             550
5.823
Skew:                           1.562   Prob(JB):
0.00
Kurtosis:                       8.856   Cond. No.                     2.2
1e+33
==============================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 5.7e-61. This might indicate that there ar
e
strong multicollinearity problems or that the design matrix is singular.
```

In [641]:
```
mod1.mse_model
```

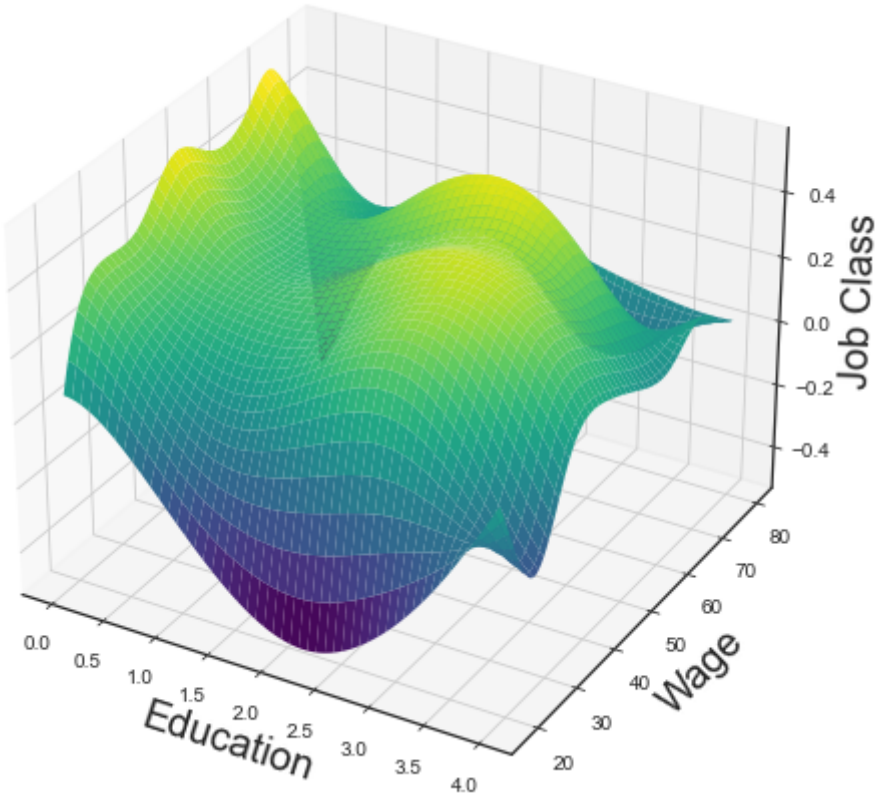
Out[641]:
```
293840.67251988157
```

After running a GLS, using the chosen predictors with their optimal polynomial degree, all of them are statistically significant.

In [497]:
```
X0 = [["X50","X12","X31"]]
```

In [524]:
```
# Fit a more comlpex GAM and display the 3D surface
from pygam import PoissonGAM, s, te



X, y = wage(return_X_y=True)
gam = PoissonGAM(s(0, n_splines=200) + te(2, 1) + s(2)).fit(X0, Y1)

import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

plt.ion()
plt.rcParams['figure.figsize'] = (12, 8)
XX = gam.generate_X_grid(term=1, meshgrid=True)
Z = gam.partial_dependence(term=1, X=XX, meshgrid=True)

ax = plt.axes(projection='3d')
ax.set_xlabel('Education', fontsize=20, rotation=150)
ax.set_ylabel('Wage', fontsize=20, rotation=45)
ax.set_zlabel('Job Class', fontsize=20, rotation=90)
surf =ax.plot_surface(XX[0], XX[1], Z, cmap='viridis')
fig.colorbar(surf)
```

Out[524]:
```
<matplotlib.colorbar.Colorbar at 0x7f9c77670c10>
```



In [512]:
```
gam.summary()
```

```
PoissonGAM
============================================== ========================
================================
Distribution:                          PoissonDist Effective DoF:
46.6282
Link Function:                             LogLink Log Likelihood:
-23668.7835
Number of Samples:                            3000 AIC:
47430.8233
                                                   AICc:
47432.3928
                                                   UBRE:
11.3237
                                                   Scale:
1.0
                                                   Pseudo R-Squared:
0.34
==========================================================================
================================
Feature Function                  Lambda               Rank         EDoF
P > x        Sig. Code
================================= ==================== ============ =====
======= ============ ============
s(0)                              [0.6]                200          10.2
0.00e+00     ***
te(2, 1)                          [0.6 0.6]            100          36.5
0.00e+00     ***
s(2)                              [0.6]                20           0.0
0.00e+00     ***
intercept                                              1            0.0
0.00e+00     ***
==========================================================================
================================
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 no
t.

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


<ipython-input-512-dec6a6acdaaa>:1: UserWarning: KNOWN BUG: p-values comp
uted in this summary are likely much smaller than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163
```

```
gam.summary()
```

Question 7.9

In [198]:
```
df4 = pd.read_csv('desktop/Boston.csv')
```

In [199]:
```
df4
```

Out[199]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat	me
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	2
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	2
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	3
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	3
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
501	502	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	9.67	2
502	503	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	9.08	2
503	504	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	2
504	505	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	2
505	506	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	1

506 rows × 14 columns

In [277]:
```
X1 = PolynomialFeatures(1).fit_transform(df4.dis.values.reshape(-1,1))
X2 = PolynomialFeatures(2).fit_transform(df4.dis.values.reshape(-1,1))
X3 = PolynomialFeatures(3).fit_transform(df4.dis.values.reshape(-1,1))
X4 = PolynomialFeatures(4).fit_transform(df4.dis.values.reshape(-1,1))
X5 = PolynomialFeatures(5).fit_transform(df4.dis.values.reshape(-1,1))
X6 = PolynomialFeatures(6).fit_transform(df4.dis.values.reshape(-1,1))
X7 = PolynomialFeatures(7).fit_transform(df4.dis.values.reshape(-1,1))
X8 = PolynomialFeatures(8).fit_transform(df4.dis.values.reshape(-1,1))
X9 = PolynomialFeatures(9).fit_transform(df4.dis.values.reshape(-1,1))
X10 = PolynomialFeatures(10).fit_transform(df4.dis.values.reshape(-1,1))
Y2 = df4.nox
```

In [289]:
```
fit15 = sm.GLS(Y2, X1).fit()
fit15.summary().tables[1]
```

Out[289]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.7153	0.007	105.257	0.000	0.702	0.729
x1	-0.0423	0.002	-27.027	0.000	-0.045	-0.039

In [290]:
```
fit16 = sm.GLS(Y2, X2).fit()
fit16.summary().tables[1]
```

Out[290]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.8440	0.011	75.385	0.000	0.822	0.866
x1	-0.1116	0.005	-20.982	0.000	-0.122	-0.101
x2	0.0071	0.001	13.462	0.000	0.006	0.008

In [291]:
```
fit17 = sm.GLS(Y2, X3).fit()
fit17.summary().tables[1]
```

Out[291]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.9341	0.021	45.110	0.000	0.893	0.975
x1	-0.1821	0.015	-12.389	0.000	-0.211	-0.153
x2	0.0219	0.003	7.476	0.000	0.016	0.028
x3	-0.0009	0.000	-5.124	0.000	-0.001	-0.001

In [292]:
```
fit18 = sm.GLS(Y2, X4).fit()
fit18.summary().tables[1]
```

Out[292]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.9522	0.039	24.172	0.000	0.875	1.030
x1	-0.2008	0.038	-5.333	0.000	-0.275	-0.127
x2	0.0280	0.012	2.395	0.017	0.005	0.051
x3	-0.0017	0.001	-1.151	0.250	-0.004	0.001
x4	3.244e-05	6.01e-05	0.540	0.589	-8.56e-05	0.000

In [293]:
```
fit19 = sm.GLS(Y2, X5).fit()
fit19.summary().tables[1]
```

Out[293]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.8214	0.072	11.338	0.000	0.679	0.964
x1	-0.0312	0.087	-0.357	0.721	-0.203	0.141
x2	-0.0484	0.037	-1.293	0.197	-0.122	0.025
x3	0.0135	0.007	1.876	0.061	-0.001	0.028
x4	-0.0013	0.001	-2.088	0.037	-0.003	-7.85e-05
x5	4.441e-05	2.07e-05	2.149	0.032	3.81e-06	8.5e-05

In [294]:
```
fit20 = sm.GLS(Y2, X6).fit()
fit20.summary().tables[1]
```

Out[294]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.4741	0.132	3.593	0.000	0.215	0.733
x1	0.5132	0.194	2.646	0.008	0.132	0.894
x2	-0.3634	0.107	-3.394	0.001	-0.574	-0.153
x3	0.1004	0.029	3.510	0.000	0.044	0.157
x4	-0.0136	0.004	-3.432	0.001	-0.021	-0.006
x5	0.0009	0.000	3.292	0.001	0.000	0.001
x6	-2.258e-05	7.2e-06	-3.137	0.002	-3.67e-05	-8.44e-06

In [295]:
```
fit21 = sm.GLS(Y2, X7).fit()
fit21.summary().tables[1]
```

Out[295]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.1421	0.257	-0.552	0.581	-0.648	0.363
x1	1.6549	0.453	3.652	0.000	0.765	2.545
x2	-1.1782	0.311	-3.783	0.000	-1.790	-0.566
x3	0.3926	0.109	3.610	0.000	0.179	0.606
x4	-0.0710	0.021	-3.380	0.001	-0.112	-0.030
x5	0.0071	0.002	3.156	0.002	0.003	0.012
x6	-0.0004	0.000	-2.957	0.003	-0.001	-0.000
x7	8.039e-06	2.89e-06	2.783	0.006	2.36e-06	1.37e-05

In [296]:
```
fit22 = sm.GLS(Y2, X8).fit()
fit22.summary().tables[1]
```

Out[296]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.9868	0.506	-1.950	0.052	-1.981	0.007
x1	3.4706	1.041	3.335	0.001	1.426	5.516
x2	-2.7341	0.861	-3.174	0.002	-4.426	-1.042
x3	1.0898	0.376	2.899	0.004	0.351	1.828
x4	-0.2510	0.095	-2.634	0.009	-0.438	-0.064
x5	0.0348	0.014	2.407	0.016	0.006	0.063
x6	-0.0029	0.001	-2.219	0.027	-0.005	-0.000
x7	0.0001	6.22e-05	2.064	0.040	6.17e-06	0.000
x8	-2.423e-06	1.25e-06	-1.937	0.053	-4.88e-06	3.5e-08

In [297]:
```
fit23 = sm.GLS(Y2, X9).fit()
fit23.summary().tables[1]
```

Out[297]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.6777	1.012	-1.658	0.098	-3.666	0.310
x1	5.1775	2.402	2.156	0.032	0.459	9.896
x2	-4.4603	2.352	-1.896	0.059	-9.082	0.162
x3	2.0308	1.251	1.623	0.105	-0.427	4.489
x4	-0.5572	0.400	-1.394	0.164	-1.343	0.228
x5	0.0968	0.080	1.211	0.227	-0.060	0.254
x6	-0.0107	0.010	-1.066	0.287	-0.031	0.009
x7	0.0007	0.001	0.952	0.341	-0.001	0.002
x8	-2.843e-05	3.3e-05	-0.861	0.389	-9.33e-05	3.64e-05
x9	4.724e-07	5.99e-07	0.789	0.431	-7.04e-07	1.65e-06

In [298]:
```
fit24 = sm.GLS(Y2, X10).fit()
fit24.summary().tables[1]
```

Out[298]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.6407	1.996	-1.323	0.187	-6.563	1.281
x1	7.8580	5.358	1.467	0.143	-2.669	18.385
x2	-7.5793	6.049	-1.253	0.211	-19.464	4.306
x3	4.0333	3.790	1.064	0.288	-3.414	11.480
x4	-1.3458	1.465	-0.919	0.359	-4.224	1.532
x5	0.2969	0.366	0.811	0.418	-0.423	1.016
x6	-0.0440	0.060	-0.730	0.466	-0.162	0.074
x7	0.0043	0.006	0.669	0.504	-0.008	0.017
x8	-0.0003	0.000	-0.623	0.533	-0.001	0.001
x9	9.799e-06	1.67e-05	0.588	0.557	-2.3e-05	4.26e-05
x10	-1.545e-07	2.76e-07	-0.560	0.576	-6.97e-07	3.88e-07

In [322]:
```
fit15.rsquared
```

Out[322]: 0.5917149670934201

In [323]:
```
resids15 = (1 - fit15.rsquared)**2
resids15
```

Out[323]: 0.16669666809552705

In [324]:
```
fit16.rsquared
```

Out[324]: 0.6998562064647851

In [325]:
```
resids16 = (1 - fit16.rsquared)**2
resids16
```

Out[325]: 0.09008629679770973

In [326]:
```
fit17.rsquared
```

Out[326]: 0.7147737433422647

In [329]:
```
resids17 = (1 - fit17.rsquared)**2
resids17
```

Out[329]: 0.08135401748698427

In [330]:
```
fit18.rsquared
```

Out[330]: 0.714939705173344

In [331]:
```
resids18 = (1 - fit18.rsquared)**2
resids18
```

Out[331]: 0.08125937168666002

In [332]:
```
fit19.rsquared
```

Out[332]: 0.7175486833345454

In [333]:
```
resids19 = (1 - fit19.rsquared)**2
resids19
```

Out[333]: 0.07977874628604893

In [334]:
```
fit20.rsquared
```

Out[334]: 0.7230099578761483

In [335]:
```
resids20 = (1 - fit20.rsquared)**2
resids20
```

Out[335]: 0.07672348343577313

In [336]:
```
fit21.rsquared
```

Out[336]: 0.7272532657173183

In [338]:
```
resids21 = (1 - fit21.rsquared)**2
resids21
```

Out[338]: 0.07439078106186778

In [339]:
```
fit22.rsquared
```

Out[339]: 0.7292963294738428

In [340]:
```
resids22 = (1 - fit22.rsquared)**2
resids22
```

Out[340]: 0.07328047723633425

In [341]:
```
fit23.rsquared
```

Out[341]: 0.7296353501892514

In [342]:
```
resids23 = (1 - fit23.rsquared)**2
resids23
```

Out[342]: 0.07309704386728871

In [343]:
```
fit24.rsquared
```

Out[343]: 0.7298063703293011

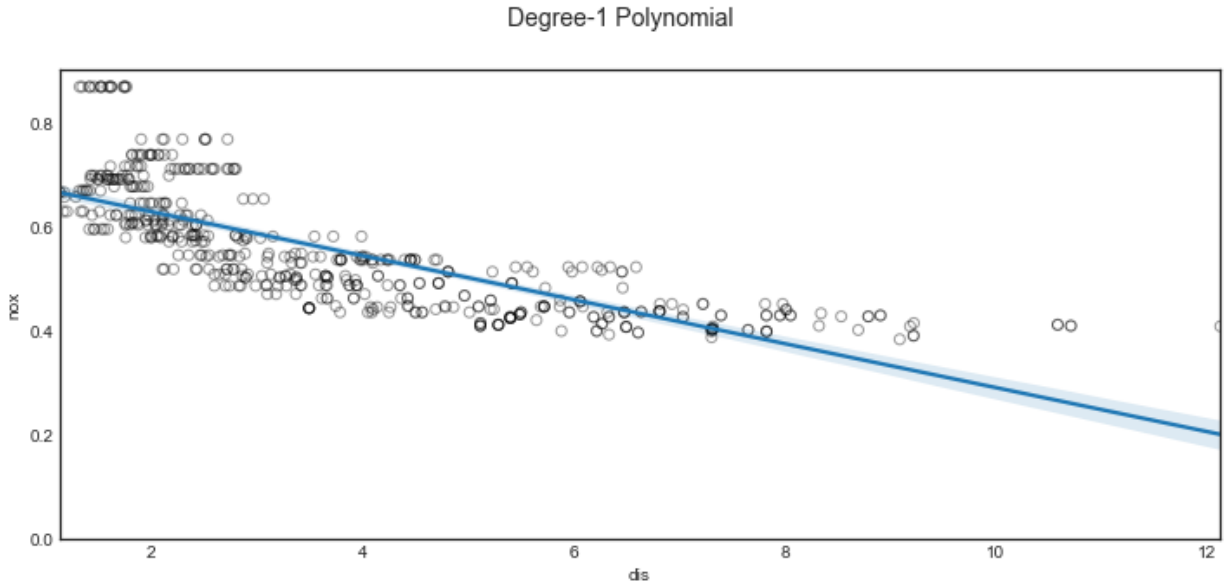
In [344]:
```
resids24 = (1 - fit24.rsquared)**2
resids24
```

Out[344]: 0.0730045975146268

In [250]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-1 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly1 = sns.regplot(df4.dis, df4.nox, order = 1, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

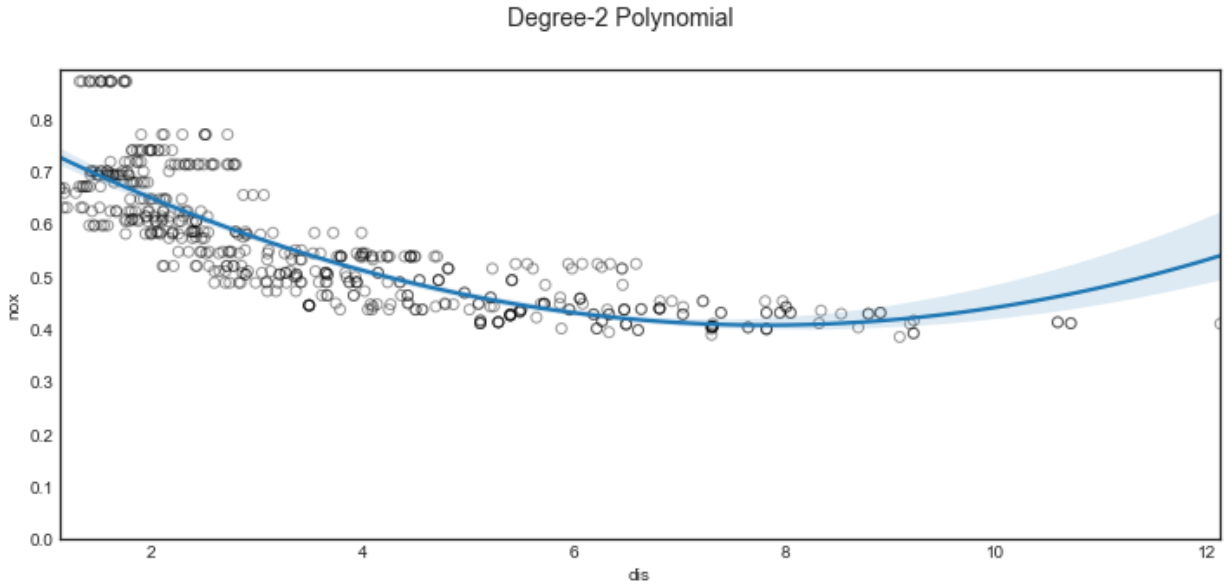
Out[250]:
```
(0.0, 0.9059436133836061)
```



In [251]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-2 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly2 = sns.regplot(df4.dis, df4.nox, order = 2, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

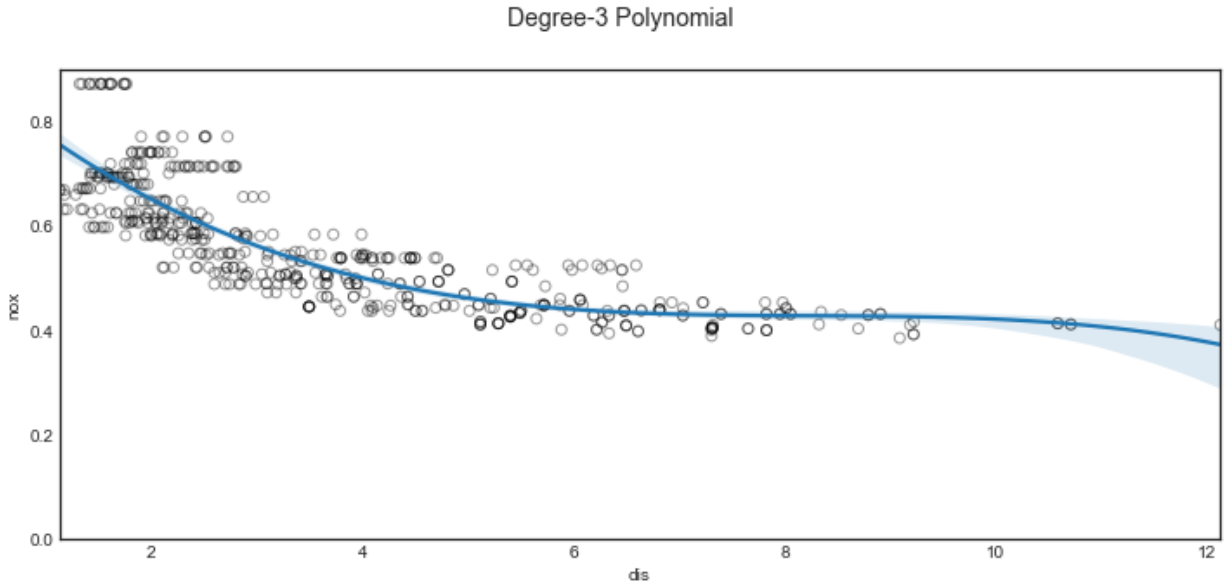
Out[251]:
```
(0.0, 0.8953)
```



In [252]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-3 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly3 = sns.regplot(df4.dis, df4.nox, order = 3, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

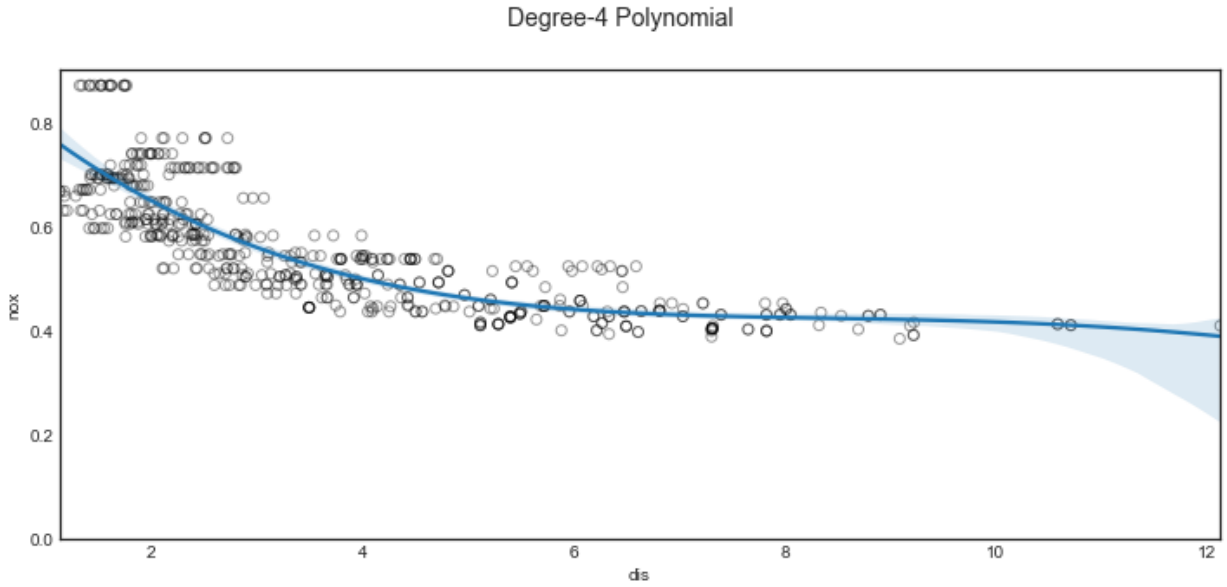
Out[252]: (0.0, 0.9001432402213405)



In [253]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-4 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly4 = sns.regplot(df4.dis, df4.nox, order = 4, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

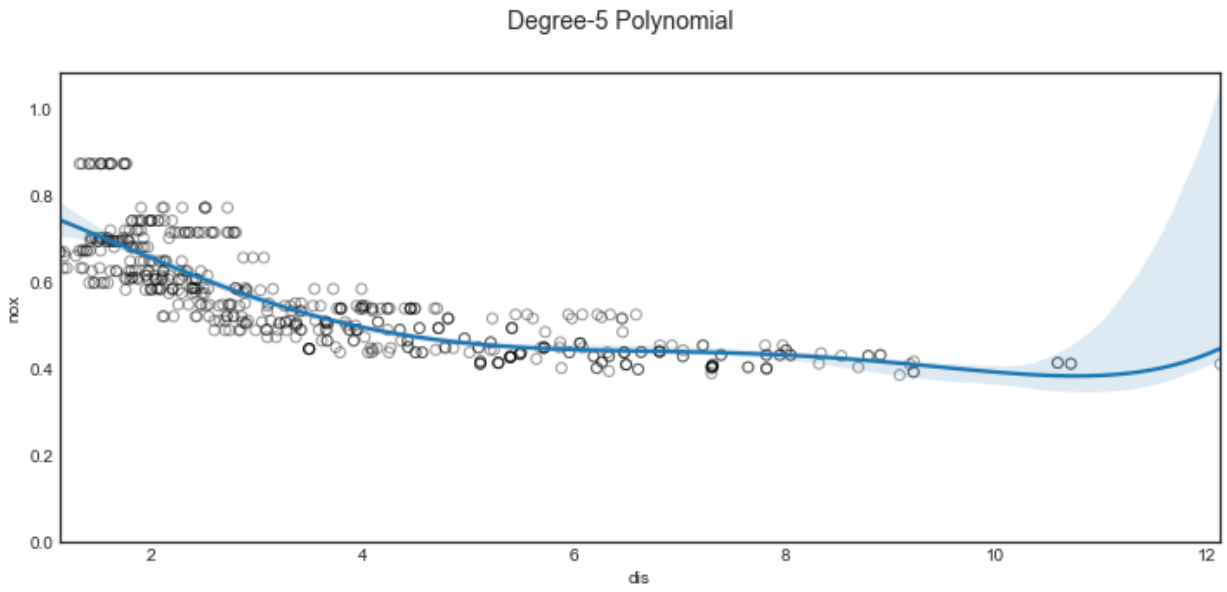
Out[253]:
```
(0.0, 0.9033327186737848)
```



In [254]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-5 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly5 = sns.regplot(df4.dis, df4.nox, order = 5, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

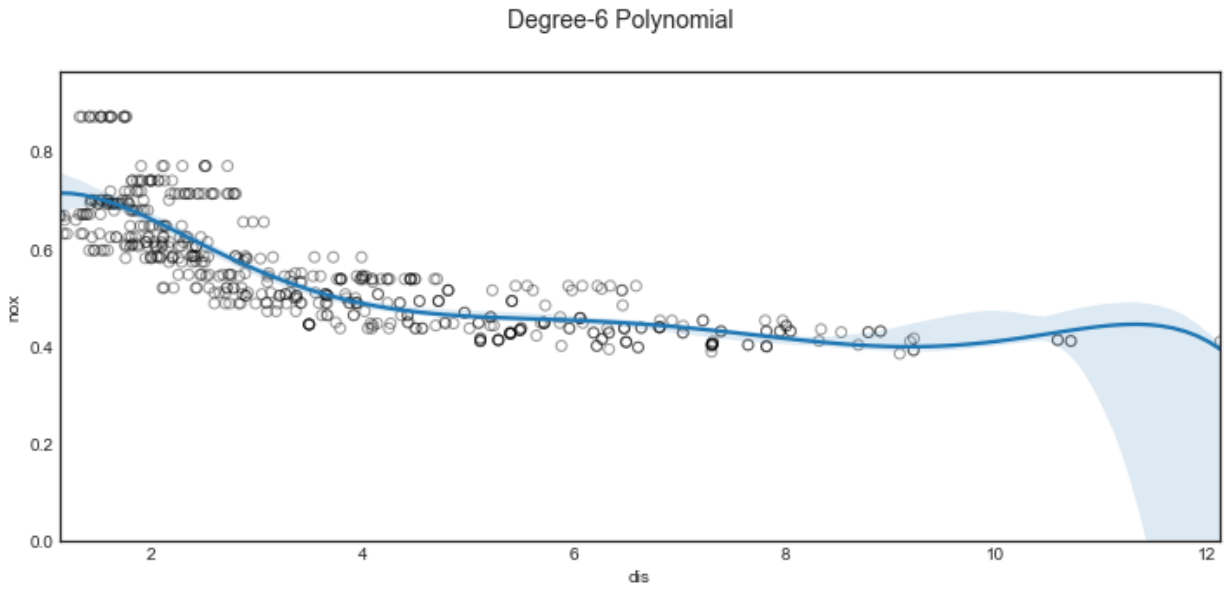
Out[254]: (0.0, 1.0829222201622202)



In [255]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-6 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly6 = sns.regplot(df4.dis, df4.nox, order = 6, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

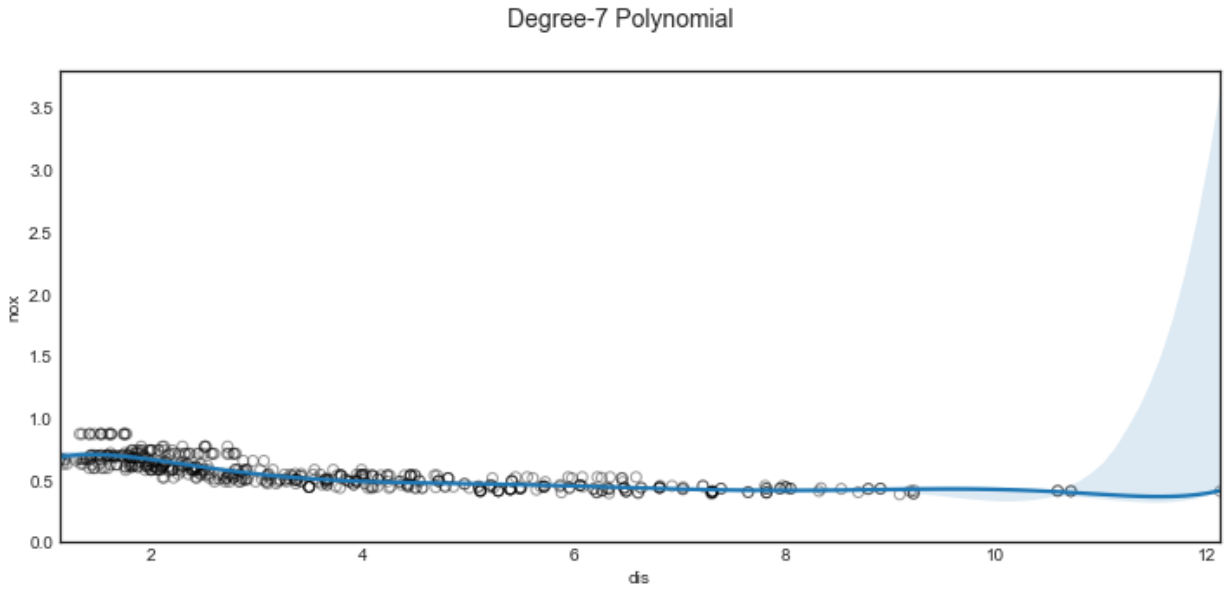
Out[255]: (0.0, 0.9656021264843891)



In [256]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-7 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly7 = sns.regplot(df4.dis, df4.nox, order = 7, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

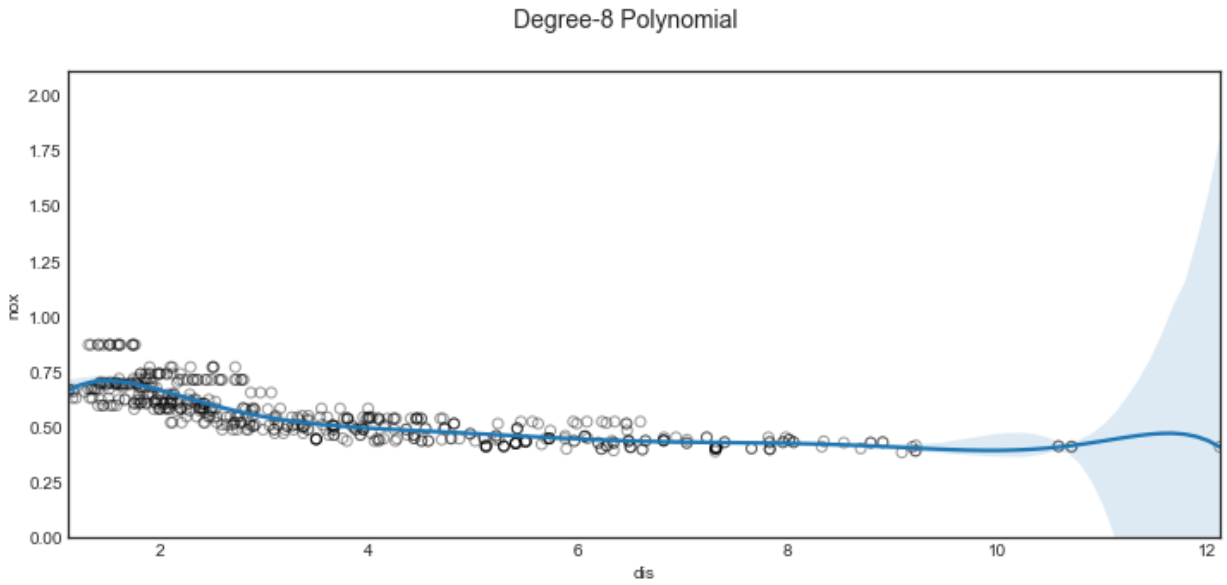
Out[256]: (0.0, 3.8027234470957074)



In [257]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-8 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly8 = sns.regplot(df4.dis, df4.nox, order = 8, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

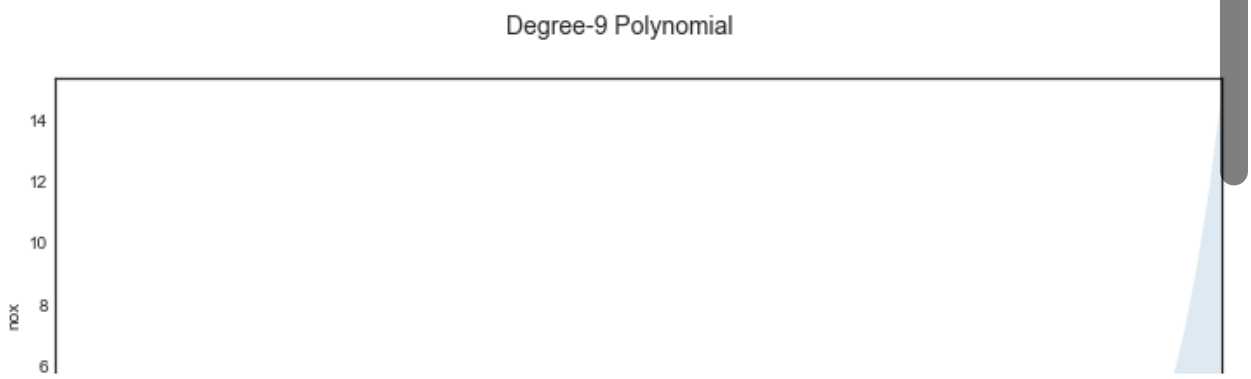
Out[257]:
```
(0.0, 2.1104134568991135)
```



In [258]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-9 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly9 = sns.regplot(df4.dis, df4.nox, order = 9, truncate=True, scatter=Fal
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

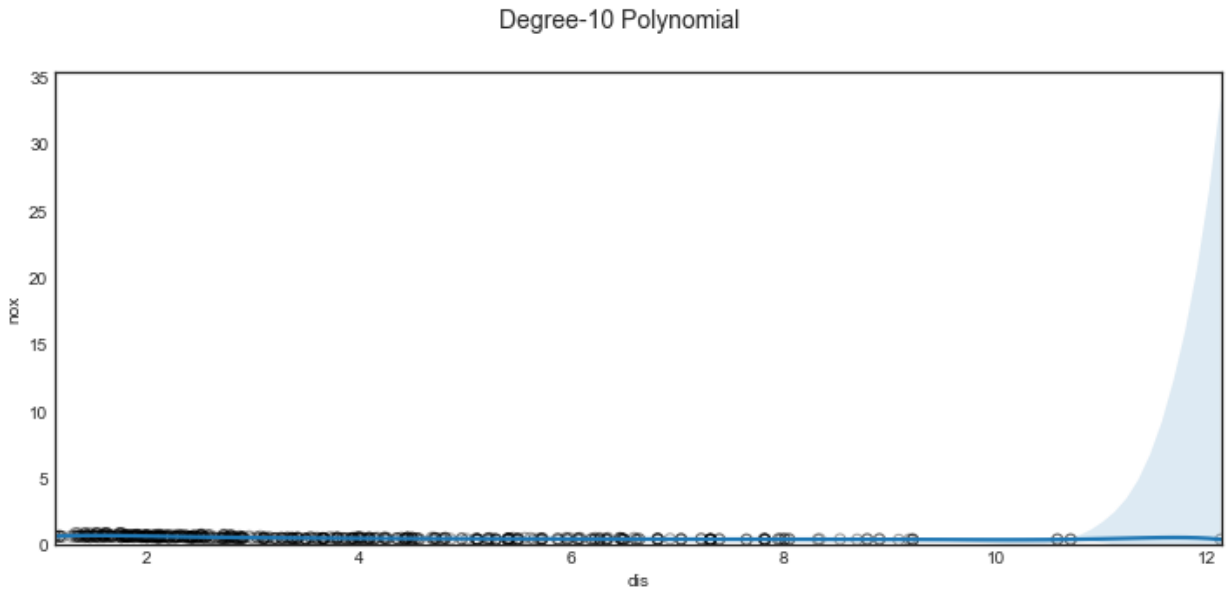
Out[258]: (0.0, 15.354373884144188)



In [259]:
```
fig, (ax1) = plt.subplots(1,1, figsize=(12,5))
fig.suptitle('Degree-10 Polynomial', fontsize=14)


ax1.scatter(df4.dis, df4.nox, facecolor='None', edgecolor='k', alpha=0.4)


poly10 = sns.regplot(df4.dis, df4.nox, order = 10, truncate=True, scatter=F
ax1.set_ylim(ymin=0)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/seaborn/_
decorators.py:36: FutureWarning: Pass the following variables as keyword
args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will resu
lt in an error or misinterpretation.
  warnings.warn(
```

Out[259]:
```
(0.0, 35.433873820091904)
```



After calculating the Sum squared residuals, I got what I expected, as we increase the number of variables R-squared will go up, which means that sum of squared errors would go down. Therefore, as the degree of polynomial increases, the sum of squared resdiuals decreases.

In [385]:
```
sm.stats.anova.anova_lm(fit15, fit16, fit17, fit18, fit19,fit20,fit21,fit22
```

Out[385]:

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	504.0	2.768563	0.0	NaN	NaN	NaN
1	503.0	2.035262	1.0	0.733301	198.116860	3.558589e-38
2	502.0	1.934107	1.0	0.101155	27.329219	2.522104e-07
3	501.0	1.932981	1.0	0.001125	0.304045	5.816033e-01
4	500.0	1.915290	1.0	0.017691	4.779699	2.926006e-02
5	499.0	1.878257	1.0	0.037033	10.005161	1.656138e-03
6	498.0	1.849484	1.0	0.028774	7.773823	5.503632e-03
7	497.0	1.835630	1.0	0.013854	3.742933	5.359844e-02
8	496.0	1.833331	1.0	0.002299	0.621093	4.310180e-01
9	495.0	1.832171	1.0	0.001160	0.313312	5.759079e-01

The quadratic fit has the second lowest p-values and its p-value decreases from linear fit, therefore, I believe its the best fit.

In [408]:
```
df4.dis.min()
```

Out[408]:
```
1.1296
```

In [407]:
```
df4.dis.max()
```

Out[407]:
```
12.1265
```

In [391]:
```
dis_grid = np.arange(df4.dis.min(), df4.dis.max()).reshape(-1,1)
```

In [450]:
```
# We could also use the df option to produce a spline with knots at uniform
# Specifying df = 4 degrees of freedom
transformed_x2 = dmatrix("bs(df4.dis, df=4, degree=3, include_intercept=Fal
                         {"df4.dis": df4.dis}, return_type='dataframe')
spline_fit2 = sm.GLM(df4.nox, transformed_x2).fit()
spline_pred2 = spline_fit2.predict(dmatrix("bs(dis_grid, df=4, degree=3, in
                              {"dis_grid": dis_grid}, return_type='dataframe
spline_fit2.params
```

Out[450]:
```
Intercept                                                    0.734474
bs(df4.dis, df=4, degree=3, include_intercept=False)[0]     -0.058098
bs(df4.dis, df=4, degree=3, include_intercept=False)[1]     -0.463563
bs(df4.dis, df=4, degree=3, include_intercept=False)[2]     -0.199788
bs(df4.dis, df=4, degree=3, include_intercept=False)[3]     -0.388809
dtype: float64
```

In [451]:
```
# We could also use the df option to produce a spline with knots at uniform
# Specifying df = 4 degrees of freedom
transformed_x3 = dmatrix("bs(df4.dis, df=5, degree=3, include_intercept=Fal
                         {"df4.dis": df4.dis}, return_type='dataframe')
spline_fit3 = sm.GLM(df4.nox, transformed_x3).fit()
spline_pred3 = spline_fit3.predict(dmatrix("bs(dis_grid, df=5, degree=3, in
                              {"dis_grid": dis_grid}, return_type='dataframe
spline_fit3.params
```

Out[451]:
```
Intercept                                                    0.672482
bs(df4.dis, df=5, degree=3, include_intercept=False)[0]      0.083105
bs(df4.dis, df=5, degree=3, include_intercept=False)[1]     -0.134604
bs(df4.dis, df=5, degree=3, include_intercept=False)[2]     -0.255052
bs(df4.dis, df=5, degree=3, include_intercept=False)[3]     -0.267850
bs(df4.dis, df=5, degree=3, include_intercept=False)[4]     -0.261032
dtype: float64
```

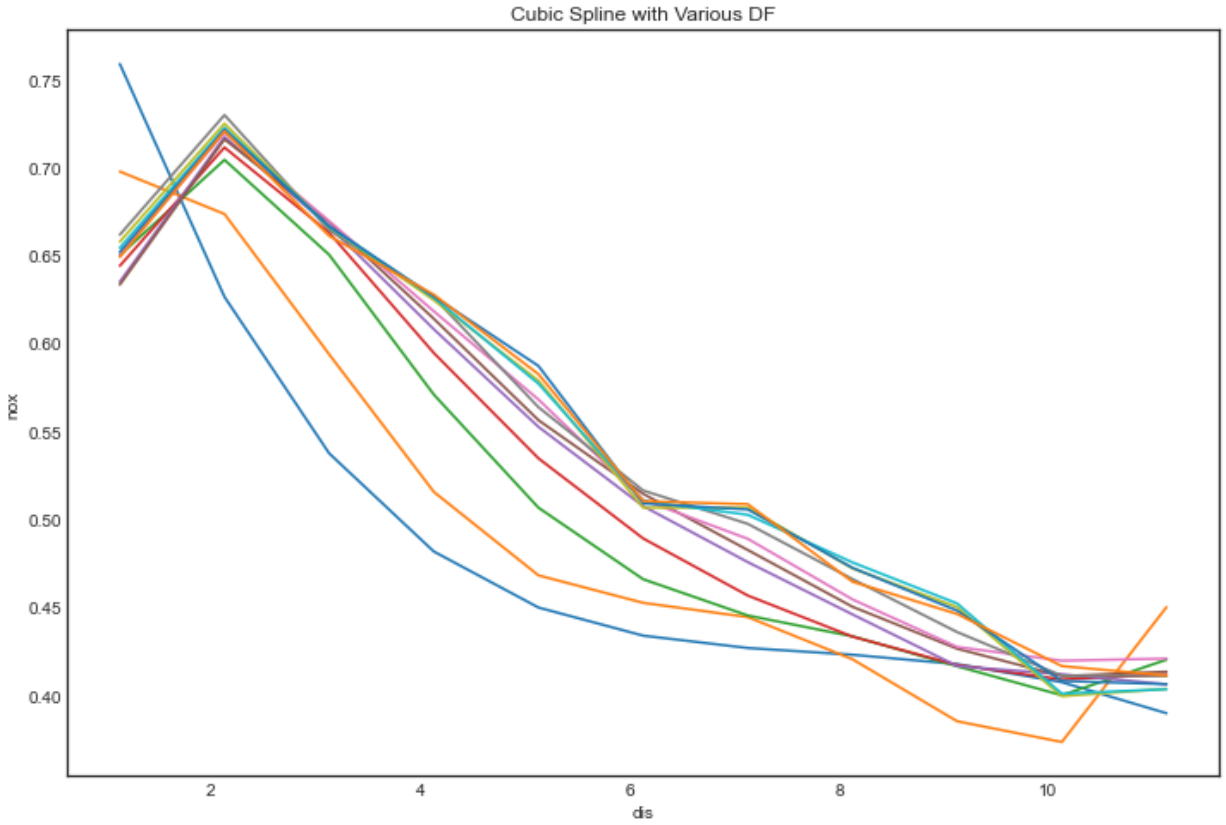
In [452]:
```
# We could also use the df option to produce a spline with knots at uniform
# Specifying df = 4 degrees of freedom
transformed_x4 = dmatrix("bs(df4.dis, df=6, degree=3, include_intercept=Fal
                         {"df4.dis": df4.dis}, return_type='dataframe')
spline_fit4 = sm.GLM(df4.nox, transformed_x4).fit()
spline_pred4 = spline_fit4.predict(dmatrix("bs(dis_grid, df=6, degree=3, in
                              {"dis_grid": dis_grid}, return_type='dataframe
spline_fit4.params
```

Out[452]:
```
Intercept                                                    0.656223
bs(df4.dis, df=6, degree=3, include_intercept=False)[0]      0.102221
bs(df4.dis, df=6, degree=3, include_intercept=False)[1]     -0.029629
bs(df4.dis, df=6, degree=3, include_intercept=False)[2]     -0.159590
bs(df4.dis, df=6, degree=3, include_intercept=False)[3]     -0.228147
bs(df4.dis, df=6, degree=3, include_intercept=False)[4]     -0.262716
bs(df4.dis, df=6, degree=3, include_intercept=False)[5]     -0.240025
dtype: float64
```

In [454]:
```
# We could also use the df option to produce a spline with knots at uniform
# Specifying df = 4 degrees of freedom
transformed_x5 = dmatrix("bs(df4.dis, df=7, degree=3, include_intercept=Fal
                         {"df4.dis": df4.dis}, return_type='dataframe')
spline_fit5 = sm.GLM(df4.nox, transformed_x5).fit()
spline_pred5 = spline_fit5.predict(dmatrix("bs(dis_grid, df=7, degree=3, in
                              {"dis_grid": dis_grid}, return_type='dataframe
spline_fit5.params
```

Out[454]:
```
Intercept                                                    0.645577
bs(df4.dis, df=7, degree=3, include_intercept=False)[0]      0.112384
bs(df4.dis, df=7, degree=3, include_intercept=False)[1]      0.024605
bs(df4.dis, df=7, degree=3, include_intercept=False)[2]     -0.092162
bs(df4.dis, df=7, degree=3, include_intercept=False)[3]     -0.162117
bs(df4.dis, df=7, degree=3, include_intercept=False)[4]     -0.222239
bs(df4.dis, df=7, degree=3, include_intercept=False)[5]     -0.248845
bs(df4.dis, df=7, degree=3, include_intercept=False)[6]     -0.230906
dtype: float64
```

In [639]:
```
RSS = []
fig = plt.figure(figsize=(12,8))
for i in range(4,16,1):
    transformed_x4 = dmatrix("bs(df4.dis, df=i, degree=4, include_intercept=
    spline_fit4 = sm.GLS(df4.nox, transformed_x4).fit()
    spline_pred4 = spline_fit4.predict(dmatrix("bs(dis_grid, df=i, degree=4,
    plt.plot(dis_grid, spline_pred4,label= i)
    RSS.append(sum((spline_fit4.fittedvalues -df4.nox) ** 2))
    plt.xlabel('dis')
    plt.ylabel('nox')
    plt.title('Cubic Spline with Various DF')
```



In [640]:
```
RSS
```

Out[640]:
```
[1.9329813272985943,
 1.8634055091330497,
 1.8342264008000653,
 1.8292649951993032,
 1.8286135870342595,
 1.827735072380106,
 1.810353020280363,
 1.7999719571216124,
 1.7869391581484588,
 1.7887720060878298,
 1.7883657041449736,
 1.7816662056347528]
```

In [457]:
```
spline_fit2.summary2()
```

Out[457]:

Model:	GLM	AIC:	-1373.8544
Link Function:	identity	BIC:	-3117.5721
Dependent Variable:	nox	Log-Likelihood:	691.93
Date:	2022-05-04 22:22	LL-Null:	58.989
No. Observations:	506	Deviance:	1.9228
Df Model:	4	Pearson chi2:	1.92
Df Residuals:	501	Scale:	0.0038379
Method:	IRLS		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.7345	0.0146	50.3055	0.0000	0.7059	0.7631
bs(df4.dis, df=4, degree=3, include_intercept=False)[0]	-0.0581	0.0219	-2.6578	0.0079	-0.1009	-0.0153
bs(df4.dis, df=4, degree=3, include_intercept=False)[1]	-0.4636	0.0237	-19.5961	0.0000	-0.5099	-0.4172
bs(df4.dis, df=4, degree=3, include_intercept=False)[2]	-0.1998	0.0431	-4.6339	0.0000	-0.2843	-0.1153
bs(df4.dis, df=4, degree=3, include_intercept=False)[3]	-0.3888	0.0455	-8.5439	0.0000	-0.4780	-0.2996

In [466]:
```
spline_fit3.summary2()
```

Out[466]:

Model:	GLM	AIC:	-1394.0729
Link Function:	identity	BIC:	-3111.4282
Dependent Variable:	nox	Log-Likelihood:	703.04
Date:	2022-05-04 22:27	LL-Null:	31.779
No. Observations:	506	Deviance:	1.8402
Df Model:	5	Pearson chi2:	1.84
Df Residuals:	500	Scale:	0.0036803
Method:	IRLS		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.6725	0.0200	33.6990	0.0000	0.6334	0.7116
bs(df4.dis, df=5, degree=3, include_intercept=False)[0]	0.0831	0.0288	2.8905	0.0038	0.0268	0.1395
bs(df4.dis, df=5, degree=3, include_intercept=False)[1]	-0.1346	0.0198	-6.7821	0.0000	-0.1735	-0.0957
bs(df4.dis, df=5, degree=3, include_intercept=False)[2]	-0.2551	0.0348	-7.3359	0.0000	-0.3232	-0.1869
bs(df4.dis, df=5, degree=3, include_intercept=False)[3]	-0.2678	0.0406	-6.5990	0.0000	-0.3474	-0.1883
bs(df4.dis, df=5, degree=3, include_intercept=False)[4]	-0.2610	0.0521	-5.0066	0.0000	-0.3632	-0.1588

In [467]:
```
spline_fit4.summary2()
```

Out[467]:

Model:	GLM	AIC:	-1393.7825
Link Function:	identity	BIC:	-3105.2078
Dependent Variable:	nox	Log-Likelihood:	703.89
Date:	2022-05-04 22:27	LL-Null:	30.859
No. Observations:	506	Deviance:	1.8340
Df Model:	6	Pearson chi2:	1.83
Df Residuals:	499	Scale:	0.0036753
Method:	IRLS		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.6562	0.0237	27.6887	0.0000	0.6098	0.7027
bs(df4.dis, df=6, degree=3, include_intercept=False)[0]	0.1022	0.0352	2.9072	0.0036	0.0333	0.1711
bs(df4.dis, df=6, degree=3, include_intercept=False)[1]	-0.0296	0.0234	-1.2671	0.2051	-0.0755	0.0162
bs(df4.dis, df=6, degree=3, include_intercept=False)[2]	-0.1596	0.0279	-5.7181	0.0000	-0.2143	-0.1049
bs(df4.dis, df=6, degree=3, include_intercept=False)[3]	-0.2281	0.0332	-6.8640	0.0000	-0.2933	-0.1630
bs(df4.dis, df=6, degree=3, include_intercept=False)[4]	-0.2627	0.0493	-5.3288	0.0000	-0.3593	-0.1661
bs(df4.dis, df=6, degree=3, include_intercept=False)[5]	-0.2400	0.0543	-4.4167	0.0000	-0.3465	-0.1335

In [468]:
```
spline_fit5.summary2()
```

Out[468]:

Model:	GLM	AIC:	-1392.9098
Link Function:	identity	BIC:	-3098.9854
Dependent Variable:	nox	Log-Likelihood:	704.45
Date:	2022-05-04 22:27	LL-Null:	30.710
No. Observations:	506	Deviance:	1.8299
Df Model:	7	Pearson chi2:	1.83
Df Residuals:	498	Scale:	0.0036745
Method:	IRLS		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	0.6456	0.0263	24.5162	0.0000	0.5940	0.6972
bs(df4.dis, df=7, degree=3, include_intercept=False)[0]	0.1124	0.0410	2.7424	0.0061	0.0321	0.1927
bs(df4.dis, df=7, degree=3, include_intercept=False)[1]	0.0246	0.0264	0.9328	0.3509	-0.0271	0.0763
bs(df4.dis, df=7, degree=3, include_intercept=False)[2]	-0.0922	0.0312	-2.9550	0.0031	-0.1533	-0.0310
bs(df4.dis, df=7, degree=3, include_intercept=False)[3]	-0.1621	0.0283	-5.7308	0.0000	-0.2176	-0.1067
bs(df4.dis, df=7, degree=3, include_intercept=False)[4]	-0.2222	0.0387	-5.7382	0.0000	-0.2981	-0.1463
bs(df4.dis, df=7, degree=3, include_intercept=False)[5]	-0.2488	0.0515	-4.8344	0.0000	-0.3497	-0.1480
bs(df4.dis, df=7, degree=3, include_intercept=False)[6]	-0.2309	0.0578	-3.9954	0.0001	-0.3442	-0.1176

The model with df = 4 performs the best since it has the lowest BIC.

In [472]:
```
spline_fit2.resid_response.mean()
```

Out[472]: -9.654113257610057e-17

In [473]:
```
spline_fit3.resid_response.mean()
```

Out[473]: 1.0070995421006855e-16

In [474]:
```
spline_fit4.resid_response.mean()
```

Out[474]: 6.2312912844574e-17

In [475]:
```
spline_fit5.resid_response.mean()
```

Out[475]: 9.829642589566603e-17

The conclusion from the resdiuals is consistent with the conclusion from the BIC, the fit with df = 4 performs better.

7.10

In [581]:
```
df9 = pd.read_csv("desktop/College.csv")
```

In [585]:
```
df9
```

Out[585]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Ro
0	1	1660	1232	721	23	52	2885	537	7440	
1	1	2186	1924	512	16	29	2683	1227	12280	
2	1	1428	1097	336	22	50	1036	99	11250	
3	1	417	349	137	60	89	510	63	12960	
4	1	193	146	55	16	44	249	869	7560	
...	...	...	...	...	...	...	...	...	...	
772	0	2197	1515	543	4	26	3089	2029	6797	
773	1	1959	1805	695	24	47	2849	1107	11520	
774	1	2097	1915	695	34	61	2793	166	6900	
775	1	10705	2453	1317	95	99	5217	83	19840	
776	1	2989	1855	691	28	63	2988	1726	4990	

777 rows × 18 columns

In [583]:
```
df9 = df9.iloc[: , 1:]
```

In [584]:
```
df9['Private'] = df9['Private'].map({'No':0, 'Yes':1})
```

In [586]:
```
X2 = df9[['Private','Apps','Accept','Enroll','Top10perc','Top25perc','F.Und
Y3 = df9[["Outstate"]]
```

In [592]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X2, Y3, test_size=0.3,
```

In [593]:
```
Fit1 = sm.GLM(Y_train, X_train['Private']).fit()
Fit1.summary().tables[1]
```

Out[593]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.184e+04	248.153	47.699	0.000	1.14e+04	1.23e+04

In [597]:
```
Fit2 = sm.GLM(Y_train, X_train[['Private','Apps']]).fit()
Fit2.summary().tables[1]
```

Out[597]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.048e+04	194.709	53.848	0.000	1.01e+04	1.09e+04
Apps	0.6947	0.033	21.120	0.000	0.630	0.759

In [596]:
```
Fit3 = sm.GLM(Y_train, X_train[['Private','Apps','Accept']]).fit()
Fit3.summary().tables[1]
```

Out[596]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.045e+04	194.251	53.772	0.000	1.01e+04	1.08e+04
Apps	0.4157	0.111	3.730	0.000	0.197	0.634
Accept	0.4614	0.176	2.619	0.009	0.116	0.807

In [598]:
```
Fit4 = sm.GLM(Y_train, X_train[['Private','Apps','Accept','Enroll']]).fit()
Fit4.summary().tables[1]
```

Out[598]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.044e+04	191.647	54.487	0.000	1.01e+04	1.08e+04
Apps	0.4474	0.110	4.059	0.000	0.231	0.663
Accept	-0.1477	0.232	-0.637	0.524	-0.602	0.307
Enroll	1.5960	0.402	3.972	0.000	0.808	2.384

In [599]:
```
Fit5 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll']]).fit()
Fit5.summary().tables[1]
```

Out[599]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.044e+04	191.243	54.567	0.000	1.01e+04	1.08e+04
Apps	0.3938	0.071	5.528	0.000	0.254	0.533
Enroll	1.4266	0.301	4.738	0.000	0.836	2.017

In [603]:
```
Fit6 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Top10perc']]).fi
Fit6.summary().tables[1]
```

Out[603]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	7597.9650	252.353	30.108	0.000	7103.362	8092.568
Apps	0.1096	0.063	1.731	0.083	-0.014	0.234
Enroll	1.1038	0.256	4.315	0.000	0.602	1.605
Top10perc	118.6506	8.095	14.657	0.000	102.784	134.517

In [602]:
```
Fit7 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Top25perc']]).fi
Fit7.summary().tables[1]
```

Out[602]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	5892.5331	299.800	19.655	0.000	5304.936	6480.130
Apps	0.2324	0.058	4.038	0.000	0.120	0.345
Enroll	-0.1213	0.256	-0.474	0.635	-0.623	0.380
Top25perc	96.5619	5.486	17.602	0.000	85.810	107.314

In [604]:
```
Fit8 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','F.Undergrad']]).
Fit8.summary().tables[1]
```

Out[604]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.043e+04	200.331	52.046	0.000	1e+04	1.08e+04
Apps	0.3927	0.072	5.481	0.000	0.252	0.533
Enroll	1.5347	0.768	1.999	0.046	0.030	3.040
F.Undergrad	-0.0204	0.133	-0.153	0.878	-0.282	0.241

In [608]:
```
Fit9 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','P.Undergrad']]).
Fit9.summary().tables[1]
```

Out[608]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	1.043e+04	191.210	54.561	0.000	1.01e+04	1.08e+04
Apps	0.4063	0.072	5.638	0.000	0.265	0.548
Enroll	1.2457	0.341	3.655	0.000	0.578	1.914
P.Undergrad	0.1396	0.123	1.132	0.258	-0.102	0.381

In [607]:
```
Fit10 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Room.Board']]).
Fit10.summary().tables[1]
```

Out[607]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	3866.3391	330.744	11.690	0.000	3218.092	4514.586
Apps	0.2436	0.052	4.657	0.000	0.141	0.346
Enroll	-0.3899	0.234	-1.663	0.096	-0.849	0.070
Room.Board	1.6713	0.076	21.897	0.000	1.522	1.821

In [612]:
```
Fit11 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Books']]).fit()
Fit11.summary().tables[1]
```

Out[612]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	7812.8211	343.326	22.756	0.000	7139.914	8485.728
Apps	0.4249	0.067	6.377	0.000	0.294	0.555
Enroll	0.1469	0.316	0.466	0.641	-0.471	0.765
Books	5.7297	0.641	8.945	0.000	4.474	6.985

In [613]:
```
Fit12 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal']]).fi
Fit12.summary().tables[1]
```

Out[613]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	9785.1329	256.510	38.147	0.000	9282.382	1.03e+04
Apps	0.4332	0.071	6.086	0.000	0.294	0.573
Enroll	0.8447	0.336	2.517	0.012	0.187	1.502
Personal	0.6810	0.182	3.750	0.000	0.325	1.037

In [614]:
```
Fit13 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal','PhD'
Fit13.summary().tables[1]
```

Out[614]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	5283.3549	275.555	19.174	0.000	4743.277	5823.433
Apps	0.3542	0.052	6.855	0.000	0.253	0.455
Enroll	-0.9523	0.256	-3.716	0.000	-1.455	-0.450
Personal	-0.7931	0.147	-5.378	0.000	-1.082	-0.504
PhD	101.7382	4.599	22.122	0.000	92.724	110.752

In [616]:
```
Fit14 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal','PhD'
Fit14.summary().tables[1]
```

Out[616]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	5001.0723	283.103	17.665	0.000	4446.200	5555.945
Apps	0.3622	0.051	7.085	0.000	0.262	0.462
Enroll	-1.0377	0.254	-4.079	0.000	-1.536	-0.539
Personal	-0.8882	0.148	-5.998	0.000	-1.178	-0.598
PhD	53.5925	13.901	3.855	0.000	26.347	80.838
Terminal	49.1359	13.407	3.665	0.000	22.860	75.412

In [617]:
```
Fit15 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal','PhD'
Fit15.summary().tables[1]
```

Out[617]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	4283.6142	291.982	14.671	0.000	3711.340	4855.888
Apps	0.3517	0.049	7.155	0.000	0.255	0.448
Enroll	-0.9693	0.245	-3.961	0.000	-1.449	-0.490
Personal	-0.6839	0.145	-4.702	0.000	-0.969	-0.399
PhD	44.0421	13.431	3.279	0.001	17.718	70.366
Terminal	41.2499	12.934	3.189	0.001	15.900	66.600
perc.alumni	69.0964	10.221	6.760	0.000	49.063	89.130

In [618]:
```
Fit16 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal','PhD'
Fit16.summary().tables[1]
```

Out[618]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	3623.1002	266.800	13.580	0.000	3100.183	4146.018
Apps	0.1748	0.046	3.765	0.000	0.084	0.266
Enroll	-0.5309	0.222	-2.395	0.017	-0.965	-0.096
Personal	-0.6487	0.130	-4.993	0.000	-0.903	-0.394
PhD	22.7514	12.132	1.875	0.061	-1.027	46.529
Terminal	41.1391	11.551	3.561	0.000	18.499	63.779
perc.alumni	40.7582	9.444	4.316	0.000	22.248	59.268
Expend	0.2893	0.025	11.705	0.000	0.241	0.338

In [619]:
```
Fit17 = sm.GLM(Y_train, X_train[['Private','Apps','Enroll','Personal','PhD'
Fit17.summary().tables[1]
```

Out[619]:

	coef	std err	z	P>|z|	[0.025	0.975]
Private	3919.7350	315.292	12.432	0.000	3301.774	4537.696
Apps	0.3257	0.050	6.566	0.000	0.228	0.423
Enroll	-0.9428	0.243	-3.878	0.000	-1.419	-0.466
Personal	-0.6881	0.144	-4.764	0.000	-0.971	-0.405
PhD	41.7195	13.360	3.123	0.002	15.535	67.904
Terminal	32.6198	13.175	2.476	0.013	6.798	58.442
perc.alumni	61.1134	10.507	5.816	0.000	40.520	81.707
Grad.Rate	21.1152	7.190	2.937	0.003	7.024	35.207

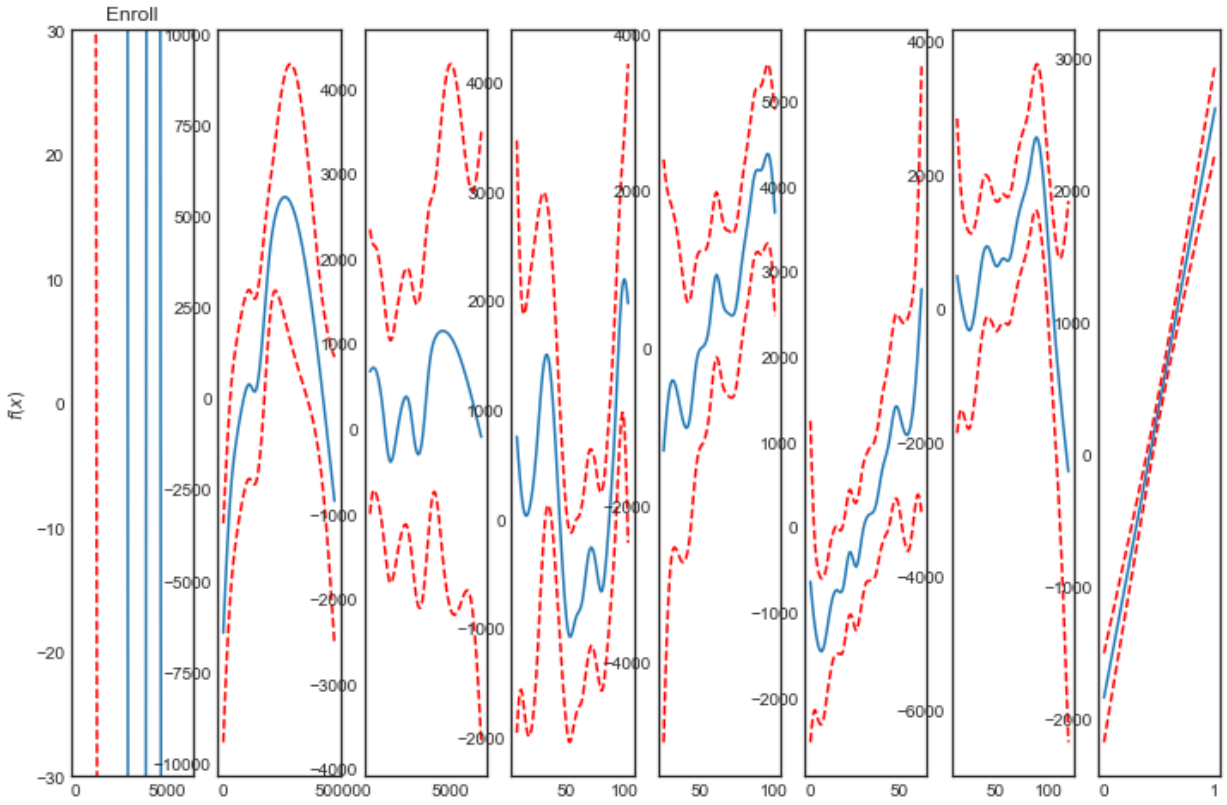
After running different 17 fits and removing variables that negatively impact our p-values, I will be moving forward with the predictors in fit 17. I did not make any decisions based on R-squared

because R-squared keeps increasing as we add more predictos, regardless of the predictor's signficance.

In [627]:
```
df10 = df9

X10 = df10[['Enroll','Apps','Personal','PhD', 'Terminal','perc.alumni','Gra
Y10 = df10[['Outstate']]
#part a
X10_train, X10_test, Y10_train, Y10_test = train_test_split(X10, Y10,
                                                    test_size = 0.3,
                                                    random_state = 414)

gam10 = GAM().fit(X10_train, Y10_train)

gam10.summary()
```

```
GAM
============================================== ============================
================================
Distribution:                          NormalDist Effective DoF:
67.8333
Link Function:                     IdentityLink Log Likelihood:
-8846.6136
Number of Samples:                          543 AIC:
17830.8938
                                                AICc:
17851.2116
                                                GCV:
6102487.0739
                                                Scale:
4747683.9215
                                                Pseudo R-Squared:
0.7509
================================================================================
================================
```

In [658]:
```
## plotting
fig, axs = plt.subplots(1,8);
titles = ['Enroll', 'Apps', 'Personal','PhD','Terminal','perc.alumni','Grad
for i, ax in enumerate(axs):
    XX = gam10.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam10.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam10.partial_dependence(term=i, X=XX, width=.95)[1],

    if i == 0:
        ax.set_ylabel('$f(x)$')
        ax.set_ylim(-30,30)
        ax.set_title(titles[i]);
```



In [628]:
```
pred = gam10.predict(X10_test)
print(mean_squared_error(Y10_test, pred))
```

```
4863667.151544344
```

The Pdeuso R-squared is 75% which is a relatively high percentage of variation explained by the model.

Based on the plots, all the predictors seem to have non-linear relationship with the response variable except for Enroll and Private.