Submission for:

Daniel Diamont (dd28977)

John Sigmon (js85773)

In [145]:

```
import sklearn as sk
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

EE 461P: Data Science Principles

Assignment 2

Total points: 85

Due: Tuesday, October 2nd, submitted via Canvas by 11:59 pm

Your homework should be written in a **Jupyter notebook**. You may work in groups of two if you wish. Only one student per team needs to submit the assignment on Canvas. **Please include the name and UTEID for both students on all submitted files (including this notebook).**

Also, please make sure your code runs and the graphics (and anything else) are displayed in your notebook before submitting. (%matplotlib inline)

Question 0. Bias-variance (15pts)

Use the following code to read in a small set of data and divide it into training and testing sets. Inputs are x; outputs are y.

In [289]:

```
import numpy as np

data_train = np.genfromtxt('data_q0_train.csv', delimiter=',')
x_train = data_train[:,0].reshape(-1, 1)
y_train = data_train[:,1].reshape(-1, 1)

data_test = np.genfromtxt('data_q0_test.csv', delimiter=',')
x_test = data_test[:,0].reshape(-1, 1)
y_test = data_test[:,1].reshape(-1, 1)
```

We want to build a model that can predict y for unknown inputs x.

(a) (5pts)

Fit a linear model to the training data, and report mean squared error on the test data. Plot the data, fitted model, and predictions, clearly denoting the training, testing, and predicted points.

In [290]:

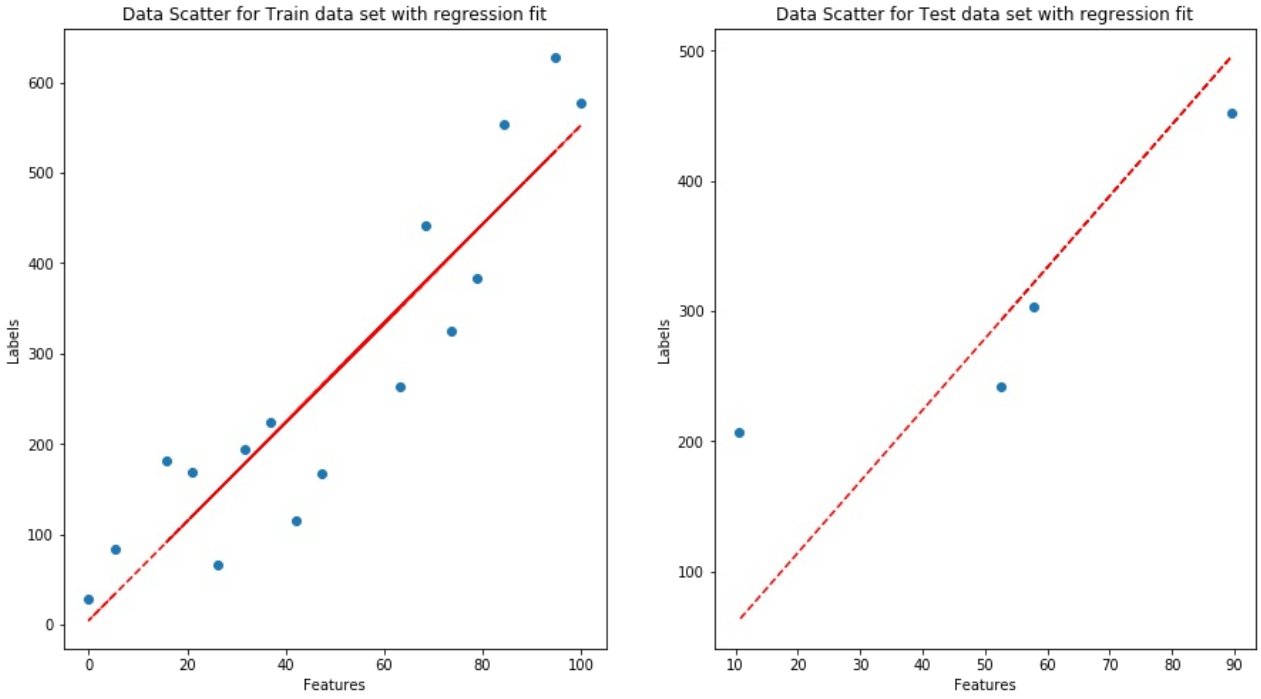
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
_ = model.fit(x_train,y_train)
```

In [291]:

```
predict = model.predict(x_test)
```

In [313]:

```
plt.subplot(121)
plt.scatter(x_train,y_train)
plt.title("Data Scatter for Train data set with regression fit")
plt.xlabel("Features")
_ = plt.ylabel("Labels")
_ = plt.plot(x_train,  model.predict(x_train), "r--")

plt.subplot(122)
plt.scatter(x_test,y_test)
plt.title("Data Scatter for Test data set with regression fit")
plt.xlabel("Features")
_ = plt.ylabel("Labels")
_ = plt.plot(x_test,  model.predict(x_test), "r--")
```



In [314]:

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, predict)
print("Test MSE is : {}".format(mse))
```

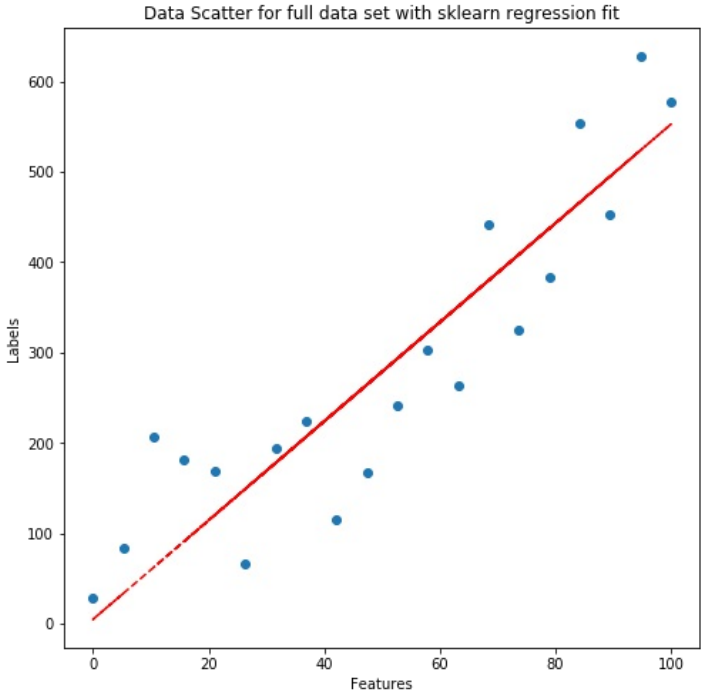
```
Test MSE is : 6455.708453830028
```

In [315]:

```
full_x = np.concatenate((x_train,x_test))
full_y = np.concatenate((y_train,y_test))
```

In [296]:

```
plt.scatter(full_x,full_y)
plt.title("Data Scatter for full data set with sklearn regression fit")
plt.xlabel("Features")
_ = plt.ylabel("Labels")
_ = plt.plot(full_x,  model.predict(full_x), "r--")
```



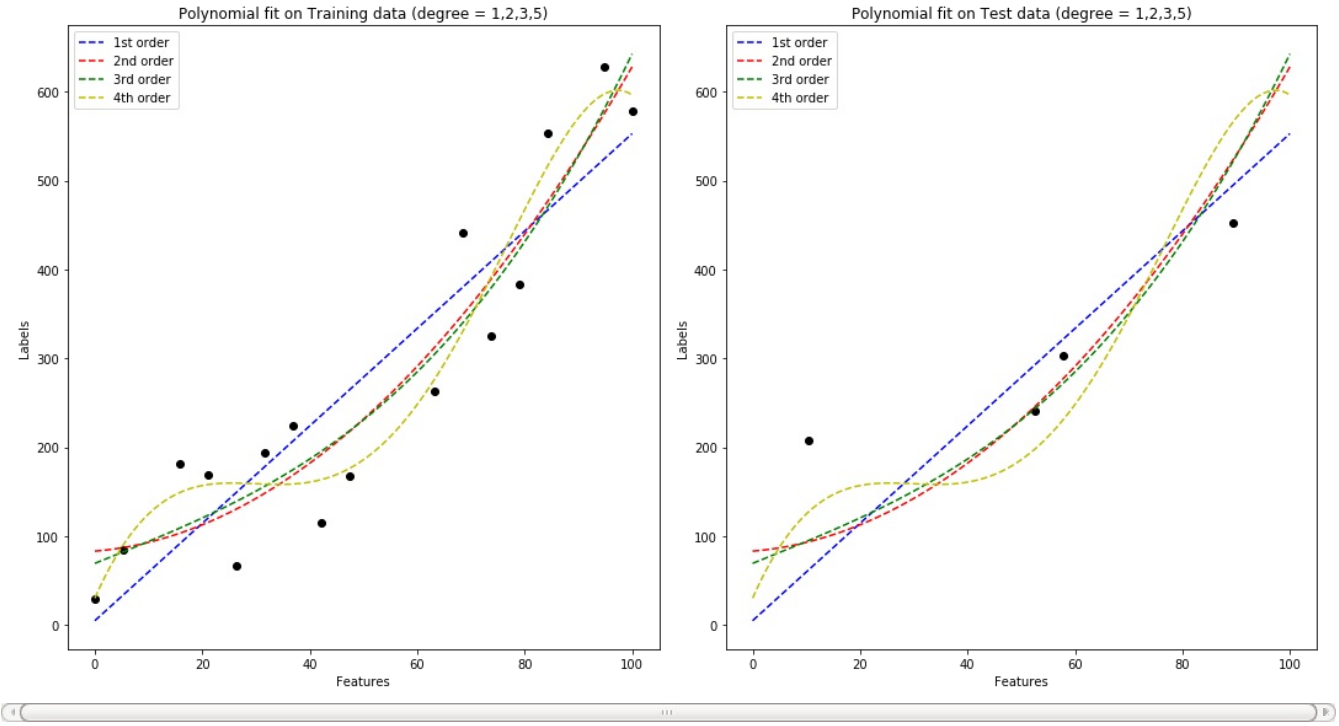
(b) (5pts)

Fit polynomial models of degrees 1, 2, 3, and 5 to the training data, and report mean squared error for both models. Plot the data, the fitted models, and the predicted outputs.

In [329]:

```
import warnings
warnings.filterwarnings('ignore')

#reshape x and y sets
x_values = np.array(x_train.reshape(-1))
# x_values = np.sort(x_values)
y_values = np.array(y_train.reshape(-1))

# get coefficients for p2,p3,5
c1 = poly.polyfit(x_values,y_values,1)
c2 = poly.polyfit(x_values,y_values,2)
c3 = poly.polyfit(x_values,y_values,3)
c5 = poly.polyfit(x_values,y_values,5)

# make predictions on test set for degrees 1,2,3, and 5

y1 = c1[1]*x_test + c1[0]
y2 = c2[2]*(x_test**2) + c2[1]*x_test + c2[0]
y3 = c3[3]*(x_test**3) + c3[2]*(x_test**2) + c3[1]*x_test + c3[0]
y5 = c5[5]*(x_test**5) + c5[4]*(x_test**4) + c5[3]*(x_test**3) + c5[2]*(x_test**2) + c5[1]*x_test + c5[0]

# mean squared error calculations
mse1 = mean_squared_error(y_test, y1)
mse2 = mean_squared_error(y_test, y2)
mse3 = mean_squared_error(y_test, y3)
mse5 = mean_squared_error(y_test, y5)

print("MSE on test set (deg: 1) = " + str(mse1))
print("MSE on test set (deg: 2) = " + str(mse2))
print("MSE on test set (deg: 3) = " + str(mse3))
print("MSE on test set (deg: 5) = " + str(mse5))

from numpy.polynomial import Polynomial
fig_size = [15,8]

plt.subplot(121)
_ = plt.rcParams["figure.figsize"] = fig_size
plt.scatter(x_train,y_train, color="black")
plt.title("Polynomial fit on Training data (degree = 1,2,3,5)")
plt.xlabel("Features")
_ = plt.ylabel("Labels")

#reshape x and y sets
x_values = np.array(x_train.reshape(-1))
# x_values = np.sort(x_values)
y_values = np.array(y_train.reshape(-1))

# fit polynomials
p1 = Polynomial.fit(x_values,y_values,1)
p2 = Polynomial.fit(x_values,y_values,2)
p3 = Polynomial.fit(x_values,y_values,3)
p5 = Polynomial.fit(x_values,y_values,5)

# create plots
_ = plt.plot(*p1.linspace(), "b--", label=r'1st order')
_ = plt.plot(*p2.linspace(), "r--", label=r'2nd order')
_ = plt.plot(*p3.linspace(), "g--", label=r'3rd order')
_ = plt.plot(*p5.linspace(), "y--", label=r'4th order')
_ = plt.legend()

# add legend

# plot predictions
plt.subplot(122)
fig_size = [15,8]
_ = plt.rcParams["figure.figsize"] = fig_size
plt.scatter(x_test,y_test, color="black")
plt.title("Polynomial fit on Test data (degree = 1,2,3,5)")
plt.xlabel("Features")
_ = plt.ylabel("Labels")

# create plots
_ = plt.plot(*p1.linspace(), "b--", label=r'1st order')
_ = plt.plot(*p2.linspace(), "r--", label=r'2nd order')
_ = plt.plot(*p3.linspace(), "g--", label=r'3rd order')
_ = plt.plot(*p5.linspace(), "y--", label=r'4th order')
_ = plt.legend()

_ = plt.tight_layout()
# add legend
```

```
MSE on test set (deg: 1) = 6455.708453830018
MSE on test set (deg: 2) = 4683.309906329018
MSE on test set (deg: 3) = 4602.202716060094
MSE on test set (deg: 5) = 6599.1329527980215
```



(c) (5pts)

Which model performed the best? Explain using the bias-variance tradeoff.

Answer:

The model that performed the best on the test set was the 3rd degree polynomial model. In this case, as the complexity of our model increased, the bias of our estimator increased while the variance decreased. We can see that in this case, the 3rd degree polynomial struck the best balance between bias and variance as it was able to best explain the variability in the *test set* while providing the lowest mean squared error out of the four different models.

Question 1. Data Exploration (23pts)

Use the following code to import the dataset.

In [330]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#read the data
data = pd.read_csv('data_q1.csv', index_col=0)
```

The columns are:

- TV: advertising dollars spent on TV for a single product (in thousands of dollars)
- Radio: advertising dollars spent on Radio
- Newspaper: advertising dollars spent on Newspaper
- Sales (dependent variable): sales of a single product in a given market (in thousands of widgets)

(a)

(2pt) Print the shape (number of rows and columns) of the data matrix and show the first 5 rows.

In [335]:

```
print(data.shape)
data.head()
```

(200, 4)

Out[335]:

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

(b)

(4pts) Generate box-plots for each of the four columns and identify the cutoff values for outliers.

In [402]:

```
data.describe()
```

Out[402]:

	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

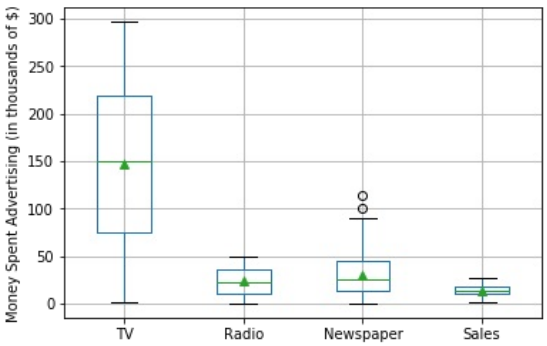
In [419]:

```
summary = data.describe()
summary = summary.iloc[[4,6]] # remove everything except the 1st and 3rd quartile information

# compute interquartile
IQR_tv = summary.iloc[1][0] - summary.iloc[0][0]
IQR_radio = summary.iloc[1][1] - summary.iloc[0][1]
IQR_news = summary.iloc[1][2] - summary.iloc[0][2]
IQR_sales = summary.iloc[1][3] - summary.iloc[0][3]


# compute cutoffs for outliers
cutoff_tv = summary.iloc[1][0] + 1.5*IQR_tv
cutoff_radio = summary.iloc[1][1] + 1.5*IQR_radio
cutoff_news = summary.iloc[1][2] + 1.5*IQR_news
cutoff_sales = summary.iloc[1][3] + 1.5*IQR_sales
```

In [445]:

```
ax = data.boxplot(showmeans=True)
_ = ax.set_ylabel("Money Spent Advertising (in thousands of $)")

# determine cutoff values for outliers
summary = data.describe()

print("Cutoffs for outliers: ")
print("TV cutoff: \t${:,.2f}".format(cutoff_tv*1000))
print("Radio cutoff: \t${:,.2f}".format(cutoff_radio*1000))
print("News cutoff: \t${:,.2f}".format(cutoff_news*1000))
print("Sales cutoff: \t{:,.2f} widgets".format(cutoff_sales*1000))
```

```
Cutoffs for outliers:
TV cutoff:      $435,500.00
Radio cutoff:   $76,350.00
News cutoff:    $93,625.00
Sales cutoff:   27,937.50 widgets
```

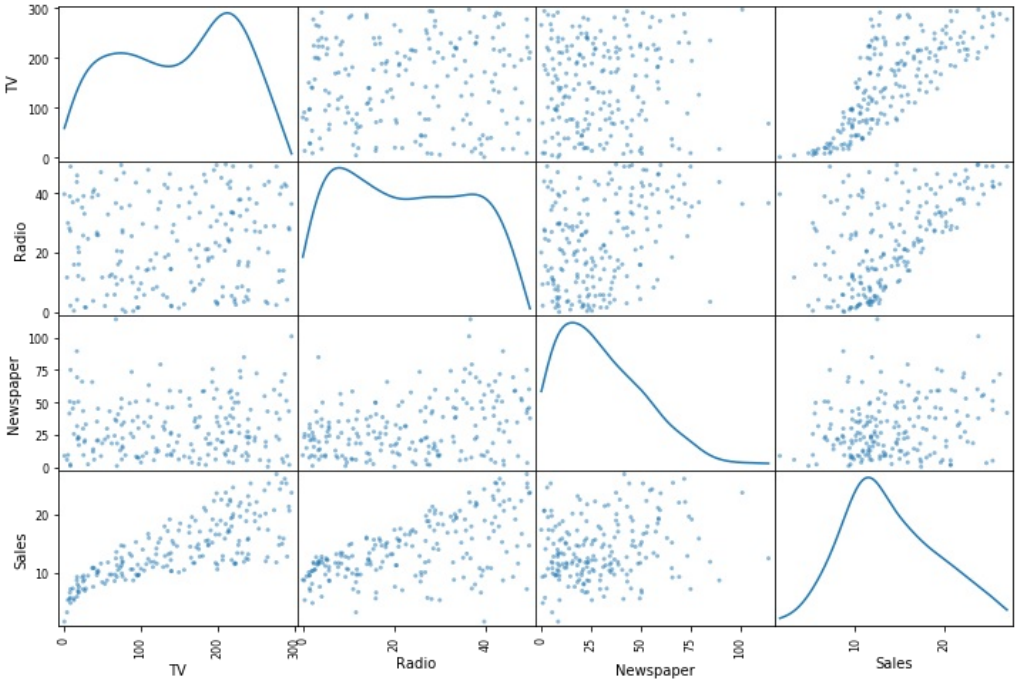


(c)

(4pts) Visualize the relationship between the features and the response variable (Sales) using scatterplots. Comment on the fits.

In [441]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(data, figsize=(12,8), diagonal='kde')
_ = plt.show()
```



By looking at the fourth row, we can visualize the relationship between TV-sales, Radio-sales, and Newspaper-sales.

We see that money spent on TV advertising appears to have a positive linear correlation with sales. Out of the three features, TV advertising vs. Sales also has the least variability in the data, suggesting that TV advertising may be a strong predictor of sales.

Radio advertising also appears to have a positive correlation with Sales, though the variability is much higher than that of TV advertising versus Sales. This suggests that Radio advertising is a weaker predictor of Sales in comparison to TV advertising.

Newspaper vs. sales shows tremendous variance in the data and a positive or negative correlation is difficult to find, suggesting that this is a weak predictor of Sales.

(d)

(4pts) Fit a simple linear regression of 'Sales' on 'TV'. What is the regression coefficient for 'TV'? What is its interpretation?

In [495]:

```
# split data into features and labels

X = data.iloc[:,:-1] # features
y = data.iloc[:,-1] # labels

X_tv = X.iloc[:,0]
X_tv = (np.array(X_tv))
X_tv = X_tv.reshape(-1,1)

model = LinearRegression()
model.fit(X_tv,y)

print("Regression coefficient: " + str(model.coef_))
```

```
Regression coefficient: [0.04753664]
```

The interpretation of the coefficient is that there is a positive linear relationship between the amount of money spent of TV advertising to the amount of widgets sold. The more ads, the more sales.

Now split the data randomly into a training and test set (keep one third of the data for test).

In [559]:

```
# split date into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state=20)
```

(e)

(4pts) Fit an MLR on all the feature variables using the training data and evaluate the trained model on the test data using root mean squared error.

In [560]:

```
model = LinearRegression()
model.fit(X_train,y_train)
y_pred_test = model.predict(X_test)
mse_test = mean_squared_error(y_test,y_pred_test)
print("MSE test = {:.5f}".format(mse_test))
```

```
MSE test = 4.55178
```

(f)

(3pts) Report the MSE obtained on train data. How much does this increase when you score your model on test data?

In [507]:

```
y_pred_train = model.predict(X_train)
mse_train = mean_squared_error(y_train,y_pred_train)
print("MSE train = {:.5f}".format(mse_train))

print("MSE increases by {:.5f} when scoring model on test data versus the training data.".format(mse_test - mse_train))
```

```
MSE train = 2.07046
MSE increases by 2.48132 when scoring model on test data versus the training data.
```

(g)

(2pts) Report the coefficients obtained by your model.

In [508]:

```
print("Coefficients: " + str(model.coef_))
```

```
Coefficients: [0.04226803 0.19051052 0.00132449]
```

Question 2. Regression with Feature Selection and Outlier Detection (20pts)

In this problem, we will build a predictive model to estimate the market price using the Dow-Jones index (https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average (https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average)). To simplify the problem, we will use previous weeks' price and volume.

In [78]:

```
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

df = pd.read_csv('dow_jones_index.data')
df = df.dropna()
X_df = df[[
        u'open', u'high', u'low', u'close',
        u'percent_change_price',
        u'volume',
        u'previous_weeks_volume',
        u'percent_change_volume_over_last_wk',
        'next_weeks_open']]

feature_name = list(X_df.columns.values)

# check the first 5 rows of data loaded
pd.DataFrame.head(X_df)
```

Out[78]:

	open	high	low	close	percent_change_price	volume	previous_weeks_volume	percent_change_volume_over_last_wk	r
1	$16.71	$16.71	$15.64	$15.97	-4.428490	242963398	239655616.0	1.380223	
2	$16.19	$16.38	$15.60	$15.79	-2.470660	138428495	242963398.0	-43.024959	
3	$15.87	$16.63	$15.82	$16.13	1.638310	151379173	138428495.0	9.355500	
4	$16.18	$17.39	$16.18	$17.14	5.933250	154387761	151379173.0	1.987452	
5	$17.33	$17.48	$16.97	$17.37	0.230814	114691279	154387761.0	-25.712195	

Feature exploration

In [79]:

```
X_df.shape
```

Out[79]:

```
(720, 9)
```

In [80]:

```
X_df = X_df[X_df.columns[0:]].replace('[\$,]', '', regex=True).astype(float) #replace text $ with float
X_df.head()
```

Out[80]:

	open	high	low	close	percent_change_price	volume	previous_weeks_volume	percent_change_volume_over_last_wk	nex
1	16.71	16.71	15.64	15.97	-4.428490	242963398.0	239655616.0	1.380223	
2	16.19	16.38	15.60	15.79	-2.470660	138428495.0	242963398.0	-43.024959	
3	15.87	16.63	15.82	16.13	1.638310	151379173.0	138428495.0	9.355500	
4	16.18	17.39	16.18	17.14	5.933250	154387761.0	151379173.0	1.987452	
5	17.33	17.48	16.97	17.37	0.230814	114691279.0	154387761.0	-25.712195	

In [81]:

```
X = X_df.iloc[:,:-1] # features
y = X_df.iloc[:,-1] # labels

# split date into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state=20)
```

Forward/ backward feature selection.

(a)

(3pts) Calculate the MSE of the training set using linear regression.

In [82]:

```
model = LinearRegression()
model.fit(X_train,y_train)
y_train_pred = model.predict(X_train)
mse_train = mean_squared_error(y_train,y_train_pred)
print("MSE of training set: {:.5f}".format(mse_train))
```

```
MSE of training set: 0.17062
```

(b)

(3pts) Find the three most significant factors using (1) forward feature selection and (2) backward feature selection. Please install and use MLXTEND package for above feature selections (https://anaconda.org/conda-forge/mlxtend (https://anaconda.org/conda-forge/mlxtend)). You may find the following link helpful: https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#example-5-sequential-feature-selection-for-regression (https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#example-5-sequential-feature-selection-for-regression). Please set the cross-validation parameter to 3 (cv=3).

In [83]:

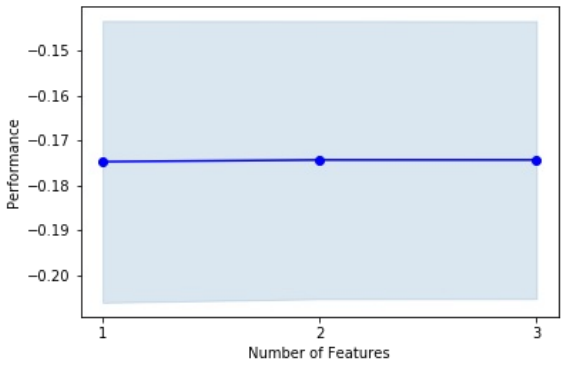
```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
from sklearn.linear_model import LinearRegression
```

(1) Using forward feature selection

In [84]:

```
lr = LinearRegression()

sfs = SFS(lr,
          k_features=3,
          forward=True,
          floating=False,
          scoring='neg_mean_squared_error',
          cv=3)

sfs = sfs.fit(X_train.values, y_train.values)
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')
print("Selected Features: ", sfs.k_feature_idx_)
```

```
Selected Features:  (3, 6, 7)
```

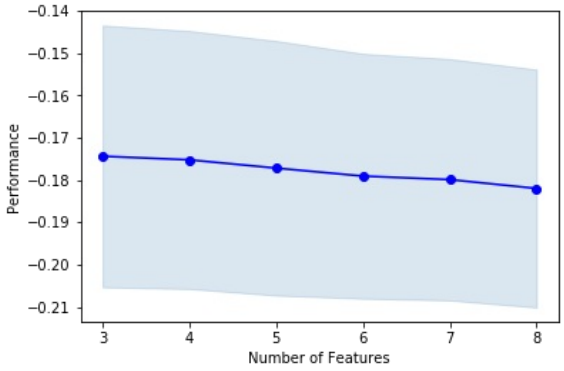


(2) Using backward feature selection

In [85]:

```
sfs1 = SFS(lr,
           k_features=3,
           forward=False,
           floating=False,
           scoring='neg_mean_squared_error',
           cv=3)

sfs1 = sfs1.fit(X_train.values, y_train.values)
fig = plot_sfs(sfs1.get_metric_dict(), kind='std_err')
print("Selected Features: ", sfs1.k_feature_idx_)
```

```
Selected Features:  (3, 6, 7)
```



(c) (2pt) Are the three most significant features found using forward and backward feature selection the same?

Yes, the three most significant features are the same.

Huber Loss function and outlier

In the second part, we will fit a linear model to the data, using a Huber loss function rather than the L2 norm usually used in OLS. sklearn has a nice API you can use: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn-linear-model-huberregressor (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn-linear-model-huberregressor). For this problem, we will consider only one feature: "percent_change_price".

In [99]:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

df = pd.read_csv('dow_jones_index.data')
stock = df[["percent_change_price", "next_weeks_open"]]
stock = stock.dropna()
stock = stock.values[1:]
stock[:, 1] = [i.split("$")[1] for i in stock[:, 1]]
stock = stock.astype('float')
stock[:, 0] = (stock[:, 0] - np.mean(stock[:, 0], axis=0))/np.std(stock[:, 0], axis=0)

X = stock[:,:1]
y = stock[:, 1]

X_train = X[:400,]
y_train = y[:400]

X_test = X[400:,]
y_test = y[400:,]
```

(d)

(2pts) Calculate the MSE using linear regression (linear_model.LinearRegression) using this single feature. Please do not use any regularization coefficient (set $\lambda = 0$).

In [100]:

```
lr = LinearRegression()
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
mse_test = mean_squared_error(y_test,y_pred)
print("MSE of test set: {:.5f}".format(mse_test))
```

```
MSE of test set: 1192.17337
```

(e)

(2pts) Calculate the MSE using Huber regression (linear_model.HuberRegressor) and compare with the result obtained in (d).

In [101]:

```
from sklearn.linear_model import HuberRegressor
hr = HuberRegressor()
hr.fit(X_train,y_train)
y_pred = hr.predict(X_test)
mse_test = mean_squared_error(y_test,y_pred)
print("MSE of test set: {:.5f}".format(mse_test))
```

```
MSE of test set: 1235.96981
```

Now we purposefully insert an outlier into the training set. Please note that this is simply for academic purpose.

In [103]:

```
y_train_outliers = np.copy(y_train)
y_train_outliers[0] = 10000.0
```

(f) and (g)

(3pts) Explain the difference in predictive performance of the two models in these two scenarios: with and without an outlier.

In [105]:

```
lr = LinearRegression()
lr.fit(X_train,y_train_outliers)
y_pred = lr.predict(X_test)
mse_test = mean_squared_error(y_test,y_pred)
print("(Linear) MSE of test set: {:.5f}".format(mse_test))

hr = HuberRegressor()
hr.fit(X_train,y_train_outliers)
y_pred = hr.predict(X_test)
mse_test = mean_squared_error(y_test,y_pred)
print("(Huber)  MSE of test set: {:.5f}".format(mse_test))
```

```
(Linear) MSE of test set: 3570.05064
(Huber)  MSE of test set: 1232.54691
```

Without an outlier:

The difference is that the Huber Regression produces an estimator which is more biased but has reduced variance in comparison to Ordinary Least Squares. Because of this, the OLS estimator produces a smaller MSE on the test set.

With an outlier:

The Linear Regressor model will be significantly impacted by the outlier and will try to account for the variability given by the outlier by producing a more biased estimator. However, the Huber Regressor is impacted very little by the outlier since it imposes a penalty on models of increased complexity that would try to explain the variability given by the single outlier.

** Note: Huber Regression is a hybrid of Lasso and Ridge regression. It forces some coefficients down to 0 (like Lasso) as well as shrinks all the coefficients (Like Ridge). It works by making small coefficients contribute their L1 norm to a penalty and large coefficients contribute their L2 norm.

Question 3. Sampling (6pts)

A recent survey estimated that 30% of all Europeans aged 20 to 22 have driven under the influence of drugs or alcohol, based on a simple "Yes or No" question. A similar survey is being planned for Americans. The survey designers want the 90% confidence interval to have a margin of error of at most ± 0.09.

$n \geq \hat{p}(1-\hat{p}) * (\frac{z}{\epsilon})^2$,where n is the sample size, z is the z-score, ϵ is the margin of error, and $\hat{p}$ is the percentage.

In [126]:

```
import math

p_hat = .3
p_hat_uninformed = .5
z_90 = 1.645
z_95 = 1.96
MOE = 0.09
```

(a)

(2pts) Find the necessary sample size needed to conduct this survey assuming that the expected percentage of "yes" answers will be very close to that obtained from the European survey?

In [127]:

```
n = p_hat*(1-p_hat)*(z_90/MOE)**2
print("Min sample size: " + str(math.ceil(n)) + " people.")
```

```
Min sample size: 71 people.
```

(b)

(2pts) Suppose the tolerance level was kept the same but the confidence level needs to increase to 95%. What is the required sample size for this new specification?

In [128]:

```
n = p_hat*(1-p_hat)*(z_95/MOE)**2
print("Min sample size: " + str(math.ceil(n)) + " people.")
```

```
Min sample size: 100 people.
```

(c)

(2pts) If one does not know where the true "p" may lie, one can conservatively conduct a survey assuming the worst case (in terms of required minimum sample size) scenario of $p = 0.5$. Redo part (b) for this "worst case" scenario.

In [129]:

```
p_hat = 0.5
n = p_hat_uninformed*(1-p_hat_uninformed)*(z_95/MOE)**2
print("Min sample size: " + str(math.ceil(n)) + " people.")
```

```
Min sample size: 119 people.
```

Question 4. Principal Component Analysis (11pts)

Use the following code to read in data of US Imports.

In [197]:

```
import pandas as pd

df = pd.read_csv('data_q4.csv', index_col=0)
df.iloc[:5,:5]
```

Out[197]:

Country	Agricultural machinery, equipment	Alcoholic beverages, excluding wine	Apparel, household goods - cotton	Apparel, household goods - wool	Apparel, textiles, nonwool or cotton
Afghanistan	3105.0	0.0	10739.0	7314.0	11942.0
Albania	0.0	34741.0	2752171.0	50838.0	1298224.0
Algeria	0.0	0.0	0.0	0.0	0.0
Andorra	0.0	0.0	351.0	0.0	0.0
Angola	0.0	24505.0	0.0	0.0	0.0

In [198]:

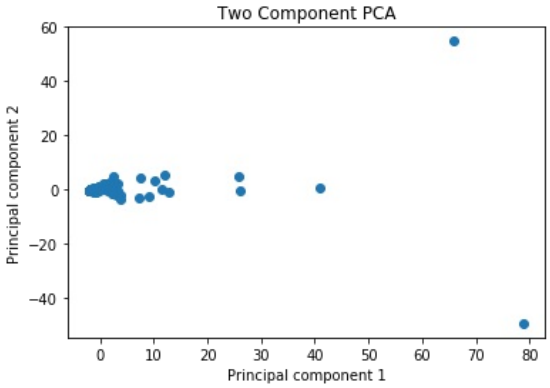
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

df_scaled = StandardScaler().fit_transform(df)
```

(a)

(3pts) Find the top two principal components from this dataset, and make a scatter plot with the first component as the x-axis and the second as the y-axis. You may find the sklearn PCA package to be useful.

In [201]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df_scaled)
pc_df = pd.DataFrame(data = principalComponents,
                     columns = ['principal component 1','principalcomponent 2'],index=df.index.values)
pc_1 = pc_df.iloc[:,0]
pc_2 = pc_df.iloc[:,1]
plt.scatter(pc_1.values,pc_2.values)
plt.xlabel("Principal component 1")
plt.ylabel("Principal component 2")
_ = plt.title("Two Component PCA")
```



(b)

(2pts) Find the names of the six countries with the highest first component (these should be clear outliers).

In [202]:

```
outliers = pc_df.nlargest(columns='principal component 1', n=6)
outliers_df = pd.DataFrame(outliers.iloc[:,0])
outliers_df
```

Out[202]:

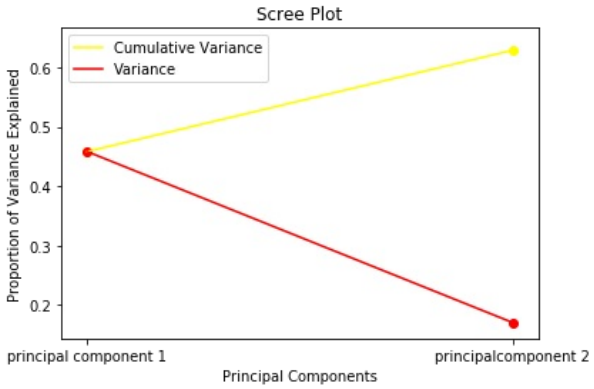
	principal component 1
China	78.808512
Canada	65.904733
Mexico	40.812633
Japan	26.098581
Germany	25.743293
Italy	12.843229

(c)

(3pts) Create i) a scree plot depicting the proportion of variance and ii) a cumulative proportion of variance explained by the principal components of the data (X matrix). Refer to Figure 10.4 of JW for an example. If you are using sklearn's PCA implementation, you may use the output attribute *explained variance ratio*.

In [226]:

```
explained_var = pca.explained_variance_ratio_
cum_var = []
cum_var.append(explained_var[0])
cum_var.append(explained_var[0] + explained_var[1])

plt.title("Scree Plot")
plt.ylabel("Proportion of Variance Explained")
plt.xlabel("Principal Components")
plt.scatter(pc_df.columns.values,cum_var,color="yellow")
plt.plot(pc_df.columns.values,cum_var, label="Cumulative Variance", color="yellow")
plt.scatter(pc_df.columns.values,explained_var,color="red")
plt.plot(pc_df.columns.values,explained_var, label="Variance",color="red")
plt.legend()
plt.show()
```



(d)

(3pts) How many principal components are required to explain cumulative variance of 30%, 60%, and 90%, respectively?

In [227]:

```
explained_var
```

Out[227]:

```
array([0.45836493, 0.17055199])
```

In [240]:

```
pca_1 = PCA()
pc = pca_1.fit_transform(df_scaled)
arr = pca_1.explained_variance_ratio_

t = .9
summer = 0
found = False
num_components = 0
for i in range(len(arr)):
    summer += arr[i]
    if(summer > t and not found):
        found = True
        num_components = i

print("Explain 30% --\t1\tprincipal compoment.")
print("Explain 60% --\t2\tprincipal components.")
print("Explain 90% --\t" + str(num_components) + "\tprincipal components.")
```

```
Explain 30% --  1       principal compoment.
Explain 60% --  2       principal components.
Explain 90% --  12      principal components.
```

Question 5. PCA (conceptual) (10pts)

(a)

(5pts) Give two reasons why we might want to use PCA.

Two reasons we wight want to use PCA are:

1. We wish to reduce the dimensionality of our data to a lower-dimensional manifold that manages to explain the variance of the original data. In this manner, we can speed up machine learning algorithms by reducing the input size of our data.
2. We can use PCA for visualization purposes. For instance, we can determine 2 or 3 principal components of some n-dimensional data set and we can plot those in a 2D or 3D graph that makes it easier for us to visualize the variance of the data.

(b)

(5pts) If we approach PCA using eigenvalue decomposition on the covariance matrix, explain what the eigenvectors and eigenvalues represent.

The eigenvectors form a basis for the covariance data of the distribution. These are the principal components. The eigenvalues can be interpreted as the magnitude of the corresponding eigenvector. In other words, the eigenvalue provides a measure of how informative its respective eigenvector is. The larger the eigenvalue, the more variance in the dataset is accounted for by its corresponding eigenvector. This is why, to choose the best principal components, we sort the eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues (where k is less than or equal to the number of dimensions of the new feature subspace).