Daily Volatility Analysis - Amazon.com Inc.

Project Luther

Maragatham K N

In [1]:
```
import requests
import pandas as pd
import numpy as np
import datetime
import time
import os
import statsmodels.api as sm
import statsmodels.formula.api as smf
import patsy
import matplotlib.pyplot as plt
import seaborn as sns
from bs4 import BeautifulSoup
from datetime import timedelta
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from sklearn.model_selection import train_test_split,KFold
from sklearn import linear_model
from sklearn.linear_model import LinearRegression, Ridge,Lasso,ElasticNet
from sklearn.preprocessing import StandardScaler, PolynomialFeatures,Norm
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_e
import scipy.stats as stats
from sklearn import metrics
import sklearn as sk
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
from sklearn.cross_validation import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegre
from sklearn.pipeline import make_pipeline
import warnings
warnings.filterwarnings(action='ignore')
%matplotlib inline
```

```
/Users/maragatham/anaconda3/lib/python3.6/site-packages/statsmodels/co
mpat/pandas.py:56: FutureWarning: The pandas.core.datetools module is
deprecated and will be removed in a future version. Please use the pan
```

```
das.tseries module instead.
  from pandas.core import datetools
/Users/maragatham/anaconda3/lib/python3.6/site-packages/sklearn/cross_
validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the r
efactored classes and functions are moved. Also note that the interfac
e of the new CV iterators are different from that of this module. This
module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/maragatham/anaconda3/lib/python3.6/site-packages/sklearn/grid_s
earch.py:42: DeprecationWarning: This module was deprecated in version
0.18 in favor of the model_selection module into which all the refacto
red classes and functions are moved. This module will be removed in 0.
20.
  DeprecationWarning)
```

Function to Get the data from Yahoo Finance

In [2]:
```
def get_data_from_web(url,company):
    chromedriver = "/Applications/chromedriver" # path to the chromedrive
    os.environ["webdriver.chrome.driver"] = chromedriver
    driver = webdriver.Chrome(chromedriver)
    driver.get(url)
    time.sleep(5)
    headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_1
    dataf=parse_data_from_page(driver,company)
    return dataf
```

Function to Parse the data from on the website

In [3]:
```
def parse_data_from_page(driver,company):
    i=500
    while i>0:
        elem = driver.find_element_by_tag_name("body")
        time.sleep(0.2)
        elem.send_keys(Keys.PAGE_DOWN)
        i-=1
    loc=driver.find_elements_by_xpath('//*[@id="Col1-1-HistoricalDataTabl
    data_list=[x.split() for x in loc[0].text.split('\n')]
    df=pd.DataFrame(data_list)
    df['Company']=company
    return df
```

Function to get url for Amazon and drive the parse functions

In [4]:
```
def get_data():
    columns=['Month','Day','Year','Open','High','Low','Close','Adj_close'
    stock_data=pd.DataFrame()
    company_dict={'Amazon.com Inc.':'https://finance.yahoo.com/quote/AMZN
                  }
    for key in company_dict.keys():
        data=get_data_from_web(company_dict[key],key)
        stock_data=stock_data.append(data)
    stock_data.rename(columns={0:columns[0],1:columns[1],2:columns[2],\
                            3:columns[3],4:columns[4],5:columns[5],\
                            6:columns[6],7:columns[7],
                              8:columns[8] },inplace=True)
    stock_data.to_csv('stock_data_2000-2018.csv')
    return(stock_data)
```

Function to create Residual and Q-Q plots

In [5]:
```
def create_plots(residual,y_predict,title):
    plt.figure(figsize=(20,5))
    #plt.scatter(pred, res)
    plt.subplot(1,2,1)

    plt.scatter(y_predict,residual)
    #sns.regplot(y_predict,residual)
    plt.title('Residual Plot '+title)
    plt.xlabel("prediction")
    plt.ylabel("residuals")

    ## Plot the Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(residual, dist="norm", plot=plt)
    plt.title('Q-Q plot '+title)
    plt.show()
```

In [6]:
```
stock_data=get_data()
```

In [7]:
```
stock_data.count()
```

Out[7]:
```
Month        4546
Day          4546
Year         4546
Open         4546
High         4546
Low          4546
Close        4546
Adj_close    4546
Volume       4546
Company      4546
dtype: int64
```

In [8]:
```
stock_data.head()
```

Out[8]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company
0	Jan	26,	2018	1,392.01	1,402.53	1,380.91	1,402.05	1,402.05	4,857,300	Amazon.com Inc.
1	Jan	25,	2018	1,368.00	1,378.34	1,357.62	1,377.95	1,377.95	4,753,000	Amazon.com Inc.
2	Jan	24,	2018	1,374.82	1,388.16	1,338.00	1,357.51	1,357.51	6,807,500	Amazon.com Inc.
3	Jan	23,	2018	1,338.09	1,364.90	1,337.34	1,362.54	1,362.54	5,169,300	Amazon.com Inc.
4	Jan	22,	2018	1,297.17	1,327.45	1,296.66	1,327.31	1,327.31	4,140,100	Amazon.com Inc.

Replacing the ',' from the Number fields

In [9]:
```
stock_data['Open']= stock_data['Open'].str.replace(',', '')
stock_data['High']= stock_data['High'].str.replace(',', '')
stock_data['Low']= stock_data['Low'].str.replace(',', '')
stock_data['Close']= stock_data['Close'].str.replace(',', '')
stock_data['Adj_close']= stock_data['Adj_close'].str.replace(',', '')
stock_data['Volume']= stock_data['Volume'].str.replace(',', '')
stock_data = stock_data.convert_objects(convert_numeric=True)
```

In [10]:
```
stock_data.head()
```

Out[10]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company
0	Jan	26,	2018	1392.01	1402.53	1380.91	1402.05	1402.05	4857300	Amazon.com Inc.
1	Jan	25,	2018	1368.00	1378.34	1357.62	1377.95	1377.95	4753000	Amazon.com Inc.
2	Jan	24,	2018	1374.82	1388.16	1338.00	1357.51	1357.51	6807500	Amazon.com Inc.
3	Jan	23,	2018	1338.09	1364.90	1337.34	1362.54	1362.54	5169300	Amazon.com Inc.
4	Jan	22,	2018	1297.17	1327.45	1296.66	1327.31	1327.31	4140100	Amazon.com Inc.

Calculating the Daily Volitility

In [11]:
```
stock_data['Diff']=stock_data['High']-stock_data.Low
```

In [12]:
```
stock_data.head()
```

Out[12]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Dif
0	Jan	26,	2018	1392.01	1402.53	1380.91	1402.05	1402.05	4857300	Amazon.com Inc.	21.6
1	Jan	25,	2018	1368.00	1378.34	1357.62	1377.95	1377.95	4753000	Amazon.com Inc.	20.7
2	Jan	24,	2018	1374.82	1388.16	1338.00	1357.51	1357.51	6807500	Amazon.com Inc.	50.1
3	Jan	23,	2018	1338.09	1364.90	1337.34	1362.54	1362.54	5169300	Amazon.com Inc.	27.5
4	Jan	22,	2018	1297.17	1327.45	1296.66	1327.31	1327.31	4140100	Amazon.com Inc.	30.7

In [13]:
```
stock_data['Date']=pd.to_datetime(stock_data['Year'].astype(str)+' '+stoc
```

In [14]:
```
stock_data.tail()
```

Out[14]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Diff	I
4541	Jan	07,	2000	67.00	70.50	66.19	69.56	69.56	10505400	Amazon.com Inc.	4.31	2 0
4542	Jan	06,	2000	71.31	72.69	64.00	65.56	65.56	18752000	Amazon.com Inc.	8.69	2 0
4543	Jan	05,	2000	70.50	75.13	68.00	69.75	69.75	38457400	Amazon.com Inc.	7.13	2 0
4544	Jan	04,	2000	85.38	91.50	81.75	81.94	81.94	17487400	Amazon.com Inc.	9.75	2 0
4545	Jan	03,	2000	81.50	89.56	79.05	89.38	89.38	16117600	Amazon.com Inc.	10.51	2 0

Getting the Quarter End Date for every record

In [15]:
```
stock_data['Quarter_end_date']=[date - pd.tseries.offsets.DateOffset(days
```

In [16]:
```
stock_data.tail()
```

Out[16]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Diff	I
4541	Jan	07,	2000	67.00	70.50	66.19	69.56	69.56	10505400	Amazon.com Inc.	4.31	2 0
4542	Jan	06,	2000	71.31	72.69	64.00	65.56	65.56	18752000	Amazon.com Inc.	8.69	2 0
4543	Jan	05,	2000	70.50	75.13	68.00	69.75	69.75	38457400	Amazon.com Inc.	7.13	2 0
4544	Jan	04,	2000	85.38	91.50	81.75	81.94	81.94	17487400	Amazon.com Inc.	9.75	2 0
4545	Jan	03,	2000	81.50	89.56	79.05	89.38	89.38	16117600	Amazon.com Inc.	10.51	2 0

In [17]:
```
stock_data.columns
```

Out[17]:
```
Index(['Month', 'Day', 'Year', 'Open', 'High', 'Low', 'Close', 'Adj_cl
ose',
       'Volume', 'Company', 'Diff', 'Date', 'Quarter_end_date'],
      dtype='object')
```

Getting the Previous day Volitility

In [18]:
```
stock_data['Prev_diff']=stock_data.groupby(['Company'])["Diff"]\
.transform(lambda grp: grp.shift(-1))
```

In [19]:
```
stock_data.tail()
```

Out[19]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Diff	I
4541	Jan	07,	2000	67.00	70.50	66.19	69.56	69.56	10505400	Amazon.com Inc.	4.31	2 0
4542	Jan	06,	2000	71.31	72.69	64.00	65.56	65.56	18752000	Amazon.com Inc.	8.69	2 0
4543	Jan	05,	2000	70.50	75.13	68.00	69.75	69.75	38457400	Amazon.com Inc.	7.13	2 0
4544	Jan	04,	2000	85.38	91.50	81.75	81.94	81.94	17487400	Amazon.com Inc.	9.75	2 0
4545	Jan	03,	2000	81.50	89.56	79.05	89.38	89.38	16117600	Amazon.com Inc.	10.51	2 0

In [20]:
```
stock_data['Quarter']=stock_data.Quarter_end_date.dt.quarter.astype(str)+
```

Adding a column for the Previous Quarter date

In [21]:
```
stock_data['PrevQuarter']=[date - pd.tseries.offsets.DateOffset(days=90)+
                           pd.tseries.offsets.QuarterEnd() for date in  s
```

In [22]:
```
stock_data.tail(20)
```

Out[22]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Diff	I
4526	Jan	31,	2000	60.38	64.75	58.44	64.56	64.56	10697900	Amazon.com Inc.	6.31	2 0
4527	Jan	28,	2000	65.00	66.44	60.00	61.69	61.69	13777900	Amazon.com Inc.	6.44	2 0
4528	Jan	27,	2000	65.19	67.75	64.63	66.94	66.94	6784000	Amazon.com Inc.	3.12	2 0
4529	Jan	26,	2000	68.63	70.00	64.75	64.81	64.81	6558000	Amazon.com Inc.	5.25	2 0
4530	Jan	25,	2000	70.00	71.25	66.00	69.25	69.25	9434100	Amazon.com Inc.	5.25	2 0

4531	Jan	24,	2000	67.56	73.38	67.50	70.13	70.13	29170200	Amazon.com Inc.	5.88	2 0
4532	Jan	21,	2000	64.63	64.63	60.00	62.06	62.06	11461900	Amazon.com Inc.	4.63	2 0
4533	Jan	20,	2000	66.94	67.00	63.94	64.75	64.75	5978000	Amazon.com Inc.	3.06	2 0
4534	Jan	19,	2000	64.13	67.50	63.00	66.81	66.81	8245500	Amazon.com Inc.	4.50	2 0
4535	Jan	18,	2000	63.44	65.19	63.00	64.13	64.13	5384900	Amazon.com Inc.	2.19	2 0
4536	Jan	14,	2000	66.75	68.63	64.00	64.25	64.25	6853600	Amazon.com Inc.	4.63	2 0
4537	Jan	13,	2000	64.94	67.19	63.13	65.94	65.94	10448100	Amazon.com Inc.	4.06	2 0
4538	Jan	12,	2000	67.88	68.00	63.00	63.56	63.56	10804500	Amazon.com Inc.	5.00	2 0
4539	Jan	11,	2000	66.88	70.00	65.00	66.75	66.75	10532700	Amazon.com Inc.	5.00	2 0
4540	Jan	10,	2000	72.56	72.63	65.56	69.19	69.19	14757900	Amazon.com Inc.	7.07	2 0
4541	Jan	07,	2000	67.00	70.50	66.19	69.56	69.56	10505400	Amazon.com Inc.	4.31	2 0
4542	Jan	06,	2000	71.31	72.69	64.00	65.56	65.56	18752000	Amazon.com Inc.	8.69	2 0
4543	Jan	05,	2000	70.50	75.13	68.00	69.75	69.75	38457400	Amazon.com Inc.	7.13	2 0
4544	Jan	04,	2000	85.38	91.50	81.75	81.94	81.94	17487400	Amazon.com Inc.	9.75	2 0
4545	Jan	03,	2000	81.50	89.56	79.05	89.38	89.38	16117600	Amazon.com Inc.	10.51	2 0

In [23]:
```
stock_data['PrevQuarter']=stock_data.PrevQuarter.dt.quarter.astype(str)+'
```

In [24]:
```
stock_data.tail()
```

Out[24]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Diff	
4541	Jan	07,	2000	67.00	70.50	66.19	69.56	69.56	10505400	Amazon.com Inc.	4.31	2 0
4542	Jan	06,	2000	71.31	72.69	64.00	65.56	65.56	18752000	Amazon.com Inc.	8.69	2 0
4543	Jan	05,	2000	70.50	75.13	68.00	69.75	69.75	38457400	Amazon.com Inc.	7.13	2 0
4544	Jan	04,	2000	85.38	91.50	81.75	81.94	81.94	17487400	Amazon.com Inc.	9.75	2 0
4545	Jan	03,	2000	81.50	89.56	79.05	89.38	89.38	16117600	Amazon.com Inc.	10.51	2 0

Assigning Dictionaries to match up the GDP ,EPS and Revenue for the previous Quarter of the record

In [25]:

```
# In millions
amz_revenue_dict={'1,2017':35714,'2,2017':37955,'3,2017':43744,'4,2017':6
                  '1,2016':29128,'2,2016':30404,'3,2016':32714,'4,2016':4
                  '1,2015':22717,'2,2015':23184,'3,2015':25358,'4,2015':3
                  '1,2014':19741,'2,2014':19340,'3,2014':20578,'4,2014':2
                  '1,2013':16070,'2,2013':15704,'3,2013':17091,'4,2013':2
                  '1,2012':13185,'2,2012':12834,'3,2012':13806,'4,2012':2
                  '1,2011':9857,'2,2011':9913,'3,2011':10874,'4,2011':174
                  '1,2010':7131,'2,2010':6566,'3,2010':7560,'4,2010':1294
                  '1,2009':4889,'2,2009':4652,'3,2009':5448,'4,2009':9520
                  '1,2008':4135,'2,2008':4063,'3,2008':4265,'4,2008':6703
                  '1,2007':3015,'2,2007':2886,'3,2007':3262,'4,2007':5672
                  '1,2006':2279,'2,2006':2139,'3,2006':2307,'4,2006':3986
                 }
# in
amz_eps_dict={'1,2017':1.48,'2,2017':.39 ,'3,2017':.52,'4,2017':3.75,\
                  '1,2016':1.07,'2,2016':1.77,'3,2016':.52 ,'4,2016':1.54
                  '1,2015':-.12,'2,2015':.19 ,'3,2015':.17 ,'4,2015':1.01
                  '1,2014':.23 ,'2,2014':-.27,'3,2014':-.95,'4,2014':.47,
                  '1,2013':.18 ,'2,2013':-.02,'3,2013':-.09,'4,2013':.52,
                  '1,2012':.28,'2,2012':.02,'3,2012':-.6,'4,2012':.21,\
                  '1,2011':.44,'2,2011':.41,'3,2011':.14,'4,2011':.38,\
                  '1,2010':.66,'2,2010':.45,'3,2010':.51,'4,2010':.91,\
                  '1,2009':.41,'2,2009':.32,'3,2009':.45,'4,2009':.86,\
                  '1,2008':.34,'2,2008':.36,'3,2008':.27,'4,2008':.52,\
                  '1,2007':.26,'2,2007':.19,'3,2007':.19,'4,2007':.48,\
                  '1,2006':.12,'2,2006':.05,'3,2006':.05,'4,2006':.23\
             }
#GDP in billions
amz_gdp_dict={'1,2017':19057.705,'2,2017':19250.009 ,'3,2017':19500.602,'
                  '1,2016':18325.187,'2,2016':18538.039,'3,2016':18729.13
                  '1,2015':17874.715,'2,2015':18093.224 ,'3,2015':18227.6
                  '1,2014':17031.324 ,'2,2014':17320.921,'3,2014':17622.2
                  '1,2013':16475.44 ,'2,2013':16541.39,'3,2013':16749.349
                  '1,2012':15973.881 ,'2,2012':16121.851,'3,2012':16227.9
                  '1,2011':15238.371 ,'2,2011':15460.926,'3,2011':15587.1
                  '1,2010':14681.063 ,'2,2010':14888.6,'3,2010':15057.66,
                  '1,2009':14383.885 ,'2,2009':14340.417,'3,2009':14384.1
                  '1,2008':14668.445 ,'2,2008':14812.974,'3,2008':14842.9
                  '1,2007':14233.226 ,'2,2007':14422.313,'3,2007':14569.6
                  '1,2006':13648.904 ,'2,2006':13799.794,'3,2006':13908.4
                  '1,2005':12813.729 ,'2,2005':12974.083,'3,2005':13205.4
                  '1,2004':11988.403 ,'2,2004':12181.398,'3,2004':12367.7
                  '1,2003':11230.078 ,'2,2003':11370.653,'3,2003':11625.1
                  '1,2002':10834.445 ,'2,2002':10934.752,'3,2002':11037.0
                  '1,2001':10508.121 ,'2,2001':10638.384,'3,2001':10639.4
                  '1,2000':10031.031 ,'2,2000':10278.34,'3,2000':10357.44
             }
```

In [26]:
```
stock_data.columns
```

Out[26]:
```
Index(['Month', 'Day', 'Year', 'Open', 'High', 'Low', 'Close', 'Adj_cl
ose',
       'Volume', 'Company', 'Diff', 'Date', 'Quarter_end_date', 'Prev_
diff',
       'Quarter', 'PrevQuarter'],
      dtype='object')
```

In [27]:
```
stock_data['Revenue']=stock_data['Quarter'].map(amz_revenue_dict)
stock_data['Revenue']=stock_data['Revenue']*1000000
stock_data['EPS']=stock_data['Quarter'].map(amz_eps_dict)

stock_data['GDP']=stock_data['Quarter'].map(amz_gdp_dict)
stock_data['GDP']=stock_data['GDP']*1000000000
```

In [28]:
```
stock_data.head()
```

Out[28]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	Di
0	Jan	26,	2018	1392.01	1402.53	1380.91	1402.05	1402.05	4857300	Amazon.com Inc.	21.6
1	Jan	25,	2018	1368.00	1378.34	1357.62	1377.95	1377.95	4753000	Amazon.com Inc.	20.7
2	Jan	24,	2018	1374.82	1388.16	1338.00	1357.51	1357.51	6807500	Amazon.com Inc.	50.1
3	Jan	23,	2018	1338.09	1364.90	1337.34	1362.54	1362.54	5169300	Amazon.com Inc.	27.5
4	Jan	22,	2018	1297.17	1327.45	1296.66	1327.31	1327.31	4140100	Amazon.com Inc.	30.7

In [29]:
```
stock_data['Prev_Revenue']=stock_data['PrevQuarter'].map(amz_revenue_dict
stock_data['Prev_Revenue']=stock_data['Prev_Revenue']*1000000
stock_data['Prev_EPS']=stock_data['PrevQuarter'].map(amz_eps_dict)

stock_data['Prev_GDP']=stock_data['PrevQuarter'].map(amz_gdp_dict)
stock_data['Prev_GDP']=stock_data['Prev_GDP']*1000000000
```

In [30]:
```
stock_data.head()
```

Out[30]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	...
0	Jan	26,	2018	1392.01	1402.53	1380.91	1402.05	1402.05	4857300	Amazon.com Inc.	...
1	Jan	25,	2018	1368.00	1378.34	1357.62	1377.95	1377.95	4753000	Amazon.com Inc.	...
2	Jan	24,	2018	1374.82	1388.16	1338.00	1357.51	1357.51	6807500	Amazon.com Inc.	...
3	Jan	23,	2018	1338.09	1364.90	1337.34	1362.54	1362.54	5169300	Amazon.com Inc.	...
4	Jan	22,	2018	1297.17	1327.45	1296.66	1327.31	1327.31	4140100	Amazon.com Inc.	...

5 rows × 22 columns

Adding Weekday column to the record

In [31]:
```
stock_data['WDay']=stock_data['Date'].dt.weekday_name
```

In [32]:
```
stock_data.head()
```

Out[32]:

	Month	Day	Year	Open	High	Low	Close	Adj_close	Volume	Company	...
0	Jan	26,	2018	1392.01	1402.53	1380.91	1402.05	1402.05	4857300	Amazon.com Inc.	...
1	Jan	25,	2018	1368.00	1378.34	1357.62	1377.95	1377.95	4753000	Amazon.com Inc.	...
2	Jan	24,	2018	1374.82	1388.16	1338.00	1357.51	1357.51	6807500	Amazon.com Inc.	...
3	Jan	23,	2018	1338.09	1364.90	1337.34	1362.54	1362.54	5169300	Amazon.com Inc.	...
4	Jan	22,	2018	1297.17	1327.45	1296.66	1327.31	1327.31	4140100	Amazon.com Inc.	...

5 rows × 23 columns

In [33]:
```
stock_data.to_csv('StockData_entire_2000-2018.csv')
```

In [34]:
```
stock_data=pd.read_csv('StockData_entire_2000-2018.csv')
```

In [35]:
```
master=stock_data[(stock_data.Year > 2005)]
```

In [36]:
```
master=master[['Diff','Open','Volume','Prev_diff','Prev_GDP','Prev_EPS','
```

In [74]:
```
master.count()
```

Out[74]:
```
Diff              3038
Open              3038
Volume            3038
Prev_diff         3038
Prev_GDP          3038
Prev_EPS          2977
Prev_Revenue      2977
WDay_Friday       3038
WDay_Monday       3038
WDay_Thursday     3038
WDay_Tuesday      3038
WDay_Wednesday    3038
dtype: int64
```

In [38]:
```
master.head()
```

Out[38]:

	Diff	Open	Volume	Prev_diff	Prev_GDP	Prev_EPS	Prev_Revenue	WDay
0	21.62	1392.01	4857300	20.72	1.973889e+13	3.75	6.050000e+10	Friday
1	20.72	1368.00	4753000	50.16	1.973889e+13	3.75	6.050000e+10	Thursday
2	50.16	1374.82	6807500	27.56	1.973889e+13	3.75	6.050000e+10	Wednesday
3	27.56	1338.09	5169300	30.79	1.973889e+13	3.75	6.050000e+10	Tuesday
4	30.79	1297.17	4140100	20.01	1.973889e+13	3.75	6.050000e+10	Monday

Performing EDA

Initialize Plot Size

In [39]:
```
# Get current size
fig_size = plt.rcParams["figure.figsize"]

# Prints: [8.0, 6.0]
print(fig_size)

# Set figure width to 12 and height to 9
fig_size[0] = 15
fig_size[1] = 9
plt.rcParams["figure.figsize"] = fig_size
```

```
[6.0, 4.0]
```

Create dummies for the Weekday column

In [40]:
```
master=pd.get_dummies(master)
```

Analyze the correlation

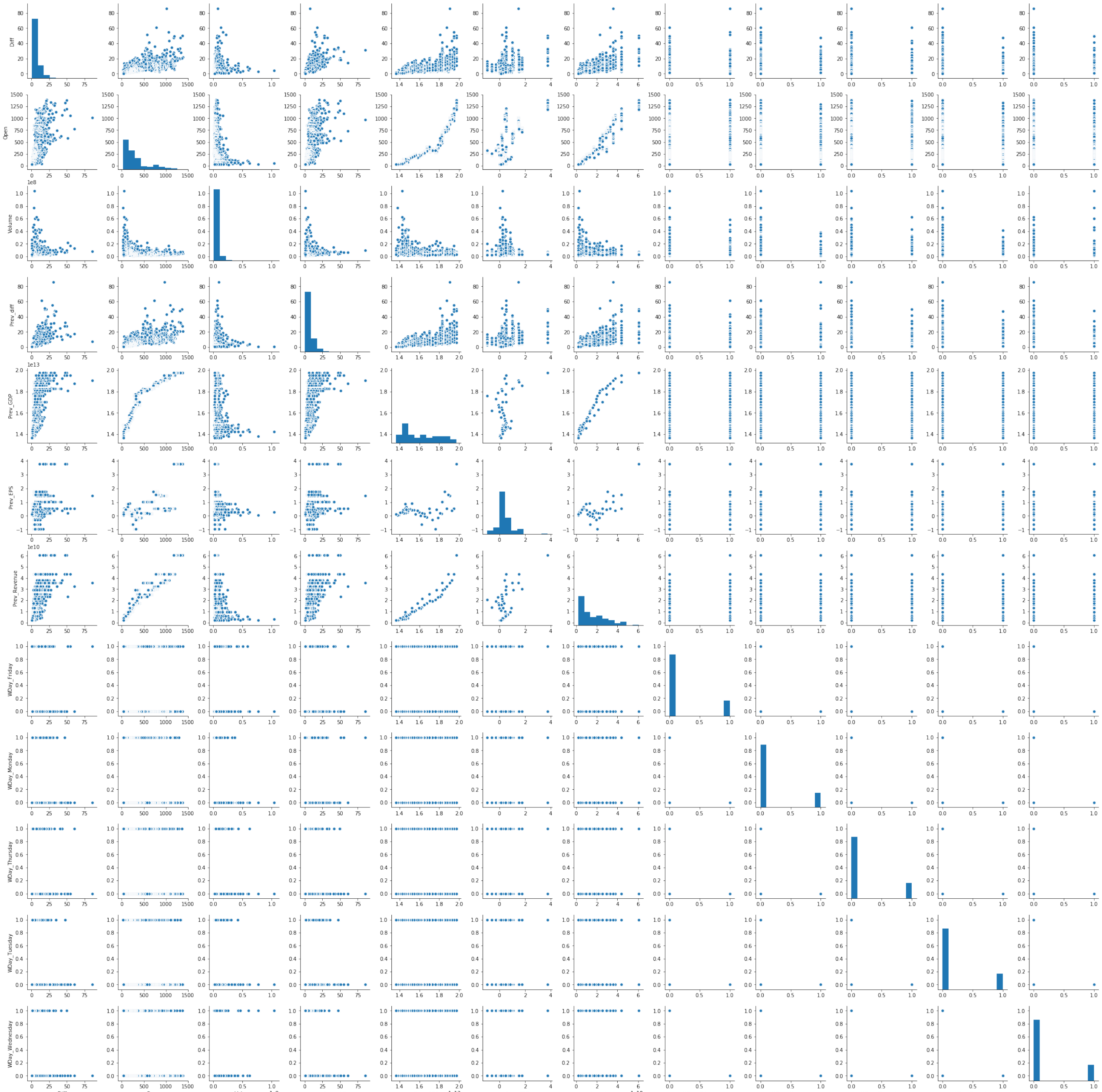
In [41]:
```
master.corr()
```

Out[41]:

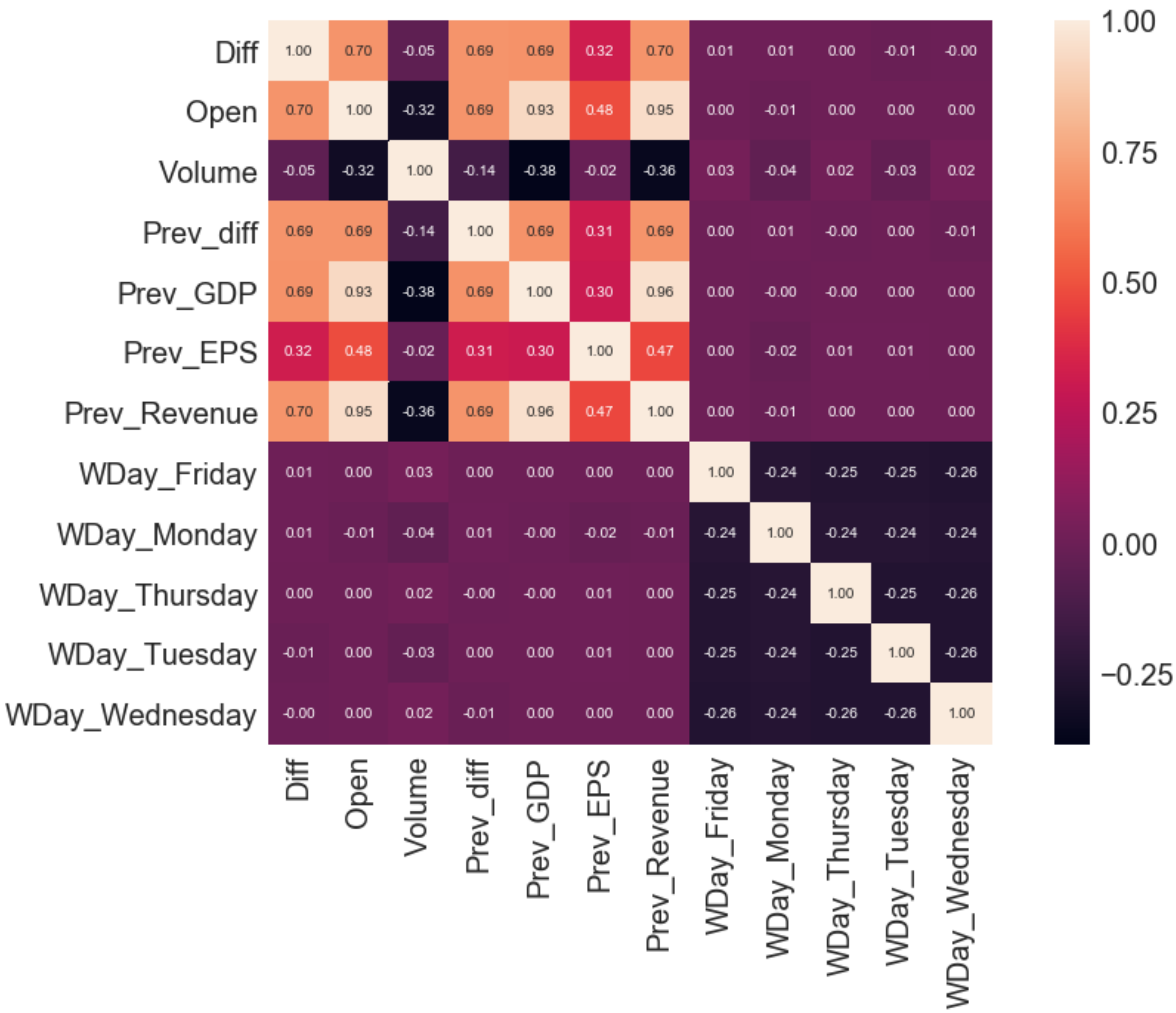
	Diff	Open	Volume	Prev_diff	Prev_GDP	Prev_EPS	Prev_Revenu
Diff	1.000000	0.701418	-0.047344	0.699566	0.697617	0.321343	0.69504
Open	0.701418	1.000000	-0.321671	0.698255	0.929573	0.483716	0.94676
Volume	-0.047344	-0.321671	1.000000	-0.137596	-0.376876	-0.015118	-0.35538
Prev_diff	0.699566	0.698255	-0.137596	1.000000	0.696948	0.314246	0.69327
Prev_GDP	0.697617	0.929573	-0.376876	0.696948	1.000000	0.302701	0.95837
Prev_EPS	0.321343	0.483716	-0.015118	0.314246	0.302701	1.000000	0.46603
Prev_Revenue	0.695049	0.946766	-0.355380	0.693279	0.958377	0.466037	1.00000
WDay_Friday	0.007197	0.003599	0.027867	0.002623	0.001921	0.002348	0.00330
WDay_Monday	0.007123	-0.005498	-0.041941	0.008349	-0.002231	-0.019593	-0.01045
WDay_Thursday	0.001000	0.000521	0.020590	-0.002498	-0.000962	0.008471	0.00297
WDay_Tuesday	-0.009929	0.000643	-0.029107	-0.000366	0.001173	0.005979	0.00259
WDay_Wednesday	-0.005077	0.000579	0.021455	-0.007800	0.000034	0.002217	0.00128

In [42]:
```
sns.pairplot(master.dropna())
```

Out[42]: <seaborn.axisgrid.PairGrid at 0x1c164596d8>



In [43]:
```
import seaborn as sns
cm = np.corrcoef(master.dropna().values.T)
sns.set(font_scale=2)
hm = sns.heatmap(cm,
                cbar=True,
                annot=True,
                square=True,
                fmt='.2f',
                annot_kws={'size':10},
                yticklabels=master.columns,
                xticklabels=master.columns)
plt.show()
```



Splitting the data into Test and Training Sets

In [ ]:

In [44]:
```
master_copy=master.dropna()

X, y = master_copy.drop('Diff',axis=1), master_copy['Diff']

X, X_test, y, y_test = train_test_split(X, y, test_size=.3, random_state=

X_mod = X.drop(303)
```

Feature Engineering

Stats Model for 1 feature - Prev_GDP

In [45]:
```
model = sm.OLS(y,sm.add_constant(X['Prev_GDP']))
results = model.fit()

results.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.479
Model:	OLS	**Adj. R-squared:**	0.479
Method:	Least Squares	**F-statistic:**	1914.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	4.99e-297
Time:	18:04:46	**Log-Likelihood:**	-6057.0
No. Observations:	2083	**AIC:**	1.212e+04
Df Residuals:	2081	**BIC:**	1.213e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-33.0596	0.912	-36.253	0.000	-34.848	-31.271
Prev_GDP	2.461e-12	5.62e-14	43.748	0.000	2.35e-12	2.57e-12

Omnibus:	2176.047	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	251509.376
Skew:	4.877	**Prob(JB):**	0.00
Kurtosis:	55.941	**Cond. No.**	1.52e+14

Stats Model for the feature - Open

In [46]:
```
model = sm.OLS(y,sm.add_constant(X['Open']))
results = model.fit()

results.summary()
```

Out[46]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.487
Model:	OLS	**Adj. R-squared:**	0.487
Method:	Least Squares	**F-statistic:**	1978.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	3.59e-304
Time:	18:04:46	**Log-Likelihood:**	-6040.6
No. Observations:	2083	**AIC:**	1.209e+04
Df Residuals:	2081	**BIC:**	1.210e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.0034	0.141	14.162	0.000	1.726	2.281
Open	0.0151	0.000	44.469	0.000	0.014	0.016

Omnibus:	2026.253	**Durbin-Watson:**	2.017
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	190239.424
Skew:	4.376	**Prob(JB):**	0.00
Kurtosis:	48.993	**Cond. No.**	613.

Stats Model for both Prev_GDP and Open

In [47]:
```
model = sm.OLS(y,sm.add_constant(X[['Open','Prev_GDP']]))
results = model.fit()

results.summary()
```

Out[47]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.501
Model:	OLS	**Adj. R-squared:**	0.500
Method:	Least Squares	**F-statistic:**	1042.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	2.78e-314
Time:	18:04:46	**Log-Likelihood:**	-6013.2
No. Observations:	2083	**AIC:**	1.203e+04
Df Residuals:	2080	**BIC:**	1.205e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-14.2145	2.185	-6.506	0.000	-18.499	-9.930
Open	0.0087	0.001	9.452	0.000	0.007	0.010
Prev_GDP	1.127e-12	1.51e-13	7.438	0.000	8.3e-13	1.42e-12

Omnibus:	2123.811	**Durbin-Watson:**	2.019
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	233482.891
Skew:	4.689	**Prob(JB):**	0.00
Kurtosis:	54.012	**Cond. No.**	3.72e+14

Stats Model For Prev_Revenue

In [48]:

```
model = sm.OLS(y,sm.add_constant(X['Prev_Revenue']))
results = model.fit()

results.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.483
Model:	OLS	**Adj. R-squared:**	0.483
Method:	Least Squares	**F-statistic:**	1948.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	7.96e-301
Time:	18:04:46	**Log-Likelihood:**	-6048.3
No. Observations:	2083	**AIC:**	1.210e+04
Df Residuals:	2081	**BIC:**	1.211e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	1.0909	0.158	6.901	0.000	0.781	1.401
Prev_Revenue	3.497e-10	7.92e-12	44.131	0.000	3.34e-10	3.65e-10

Omnibus:	2149.245	**Durbin-Watson:**	2.012
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	254699.047
Skew:	4.759	**Prob(JB):**	0.00
Kurtosis:	56.329	**Cond. No.**	3.26e+10

Stats Model For Prev Diff

In [49]:
```
model = sm.OLS(y,sm.add_constant(X['Prev_diff']))
results = model.fit()

results.summary()
```

Out[49]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.477
Model:	OLS	**Adj. R-squared:**	0.477
Method:	Least Squares	**F-statistic:**	1898.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	2.83e-295
Time:	18:04:46	**Log-Likelihood:**	-6061.0
No. Observations:	2083	**AIC:**	1.213e+04
Df Residuals:	2081	**BIC:**	1.214e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	1.9261	0.145	13.285	0.000	1.642	2.210
Prev_diff	0.7136	0.016	43.571	0.000	0.681	0.746

Omnibus:	2245.303	**Durbin-Watson:**	1.983
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	381334.594
Skew:	5.000	**Prob(JB):**	0.00
Kurtosis:	68.526	**Cond. No.**	13.3

Stats Model for Prev_diff and Prev_revenue

In [50]:

```
model = sm.OLS(y,sm.add_constant(np.array(X[['Prev_diff','Prev_Revenue']]
results = model.fit()

results.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.563
Model:	OLS	**Adj. R-squared:**	0.562
Method:	Least Squares	**F-statistic:**	1338.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5874.7
No. Observations:	2083	**AIC:**	1.176e+04
Df Residuals:	2080	**BIC:**	1.177e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6255	0.147	4.243	0.000	0.336	0.915
x1	0.4113	0.021	19.422	0.000	0.370	0.453
x2	2.082e-10	1.03e-11	20.188	0.000	1.88e-10	2.28e-10

Omnibus:	2430.087	**Durbin-Watson:**	2.003
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	521108.603
Skew:	5.695	**Prob(JB):**	0.00
Kurtosis:	79.645	**Cond. No.**	3.30e+10

Stats Model For Prev_diff,Prev_GDP, Open and Prev_revenue

In [51]:

```
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_Revenue','Prev_GDP'
results = model.fit()

results.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.571
Model:	OLS	**Adj. R-squared:**	0.571
Method:	Least Squares	**F-statistic:**	692.8
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5853.7
No. Observations:	2083	**AIC:**	1.172e+04
Df Residuals:	2078	**BIC:**	1.175e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-6.2581	2.624	-2.385	0.017	-11.404	-1.112
Prev_diff	0.3892	0.021	18.311	0.000	0.347	0.431
Prev_Revenue	3.583e-11	2.98e-11	1.204	0.229	-2.25e-11	9.42e-11
Prev_GDP	5.095e-13	1.87e-13	2.728	0.006	1.43e-13	8.76e-13
Open	0.0050	0.001	5.022	0.000	0.003	0.007

Omnibus:	2400.669	**Durbin-Watson:**	2.008
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	487688.118
Skew:	5.588	**Prob(JB):**	0.00
Kurtosis:	77.123	**Cond. No.**	4.82e+14

Removing Prev_GDP from the previous step

In [52]:

```
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_Revenue','Open']]))
results = model.fit()

results.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.570
Model:	OLS	**Adj. R-squared:**	0.569
Method:	Least Squares	**F-statistic:**	918.4
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5857.4
No. Observations:	2083	**AIC:**	1.172e+04
Df Residuals:	2079	**BIC:**	1.175e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.8872	0.153	5.805	0.000	0.587	1.187
Prev_diff	0.3936	0.021	18.541	0.000	0.352	0.435
Prev_Revenue	8.82e-11	2.28e-11	3.872	0.000	4.35e-11	1.33e-10
Open	0.0057	0.001	5.895	0.000	0.004	0.008

Omnibus:	2385.542	**Durbin-Watson:**	2.009
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	477451.550
Skew:	5.527	**Prob(JB):**	0.00
Kurtosis:	76.341	**Cond. No.**	3.45e+10

Removing Prev_Revenue and replace Prev_GDP from the previous step

In [53]:

```
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_GDP','Open']]))
results = model.fit()

results.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.571
Model:	OLS	**Adj. R-squared:**	0.571
Method:	Least Squares	**F-statistic:**	923.0
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5854.4
No. Observations:	2083	**AIC:**	1.172e+04
Df Residuals:	2079	**BIC:**	1.174e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-8.2305	2.051	-4.014	0.000	-12.252	-4.209
Prev_diff	0.3916	0.021	18.504	0.000	0.350	0.433
Prev_GDP	6.545e-13	1.43e-13	4.586	0.000	3.75e-13	9.34e-13
Open	0.0056	0.001	6.479	0.000	0.004	0.007

Omnibus:	2394.822	**Durbin-Watson:**	2.007
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	480867.221
Skew:	5.567	**Prob(JB):**	0.00
Kurtosis:	76.597	**Cond. No.**	3.77e+14

Adding EPS

In [54]:
```
#prev_eps=master.dropna()
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_GDP','Open','Prev_E
results = model.fit()

results.summary()
```

Out[54]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.573
Model:	OLS	**Adj. R-squared:**	0.572
Method:	Least Squares	**F-statistic:**	696.0
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5850.9
No. Observations:	2083	**AIC:**	1.171e+04
Df Residuals:	2078	**BIC:**	1.174e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-11.1345	2.323	-4.793	0.000	-15.690	-6.579
Prev_diff	0.3885	0.021	18.358	0.000	0.347	0.430
Prev_GDP	8.52e-13	1.61e-13	5.297	0.000	5.37e-13	1.17e-12
Open	0.0041	0.001	3.862	0.000	0.002	0.006
Prev_EPS	0.5660	0.214	2.647	0.008	0.147	0.985

Omnibus:	2405.123	**Durbin-Watson:**	2.003
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	488827.870
Skew:	5.607	**Prob(JB):**	0.00
Kurtosis:	77.205	**Cond. No.**	4.28e+14

Adding Volume and Prev_Revenue and removing Prev_EPS

In [55]:
```
#prev_vol=master.dropna()
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_GDP','Open','Volume
results = model.fit()

results.summary()
```

Out[55]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.602
Model:	OLS	**Adj. R-squared:**	0.601
Method:	Least Squares	**F-statistic:**	628.1
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5776.9
No. Observations:	2083	**AIC:**	1.157e+04
Df Residuals:	2077	**BIC:**	1.160e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-14.7071	2.617	-5.620	0.000	-19.839	-9.575
Prev_diff	0.3428	0.021	16.468	0.000	0.302	0.384
Prev_GDP	9.771e-13	1.84e-13	5.316	0.000	6.17e-13	1.34e-12
Open	0.0042	0.001	4.319	0.000	0.002	0.006
Volume	2.35e-07	1.86e-08	12.603	0.000	1.98e-07	2.72e-07
Prev_Revenue	4.07e-11	2.87e-11	1.419	0.156	-1.56e-11	9.7e-11

Omnibus:	2372.067	**Durbin-Watson:**	2.044
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	521331.023
Skew:	5.429	**Prob(JB):**	0.00
Kurtosis:	79.738	**Cond. No.**	4.99e+14

In [56]:
```
master.columns
```

Out[56]:
```
Index(['Diff', 'Open', 'Volume', 'Prev_diff', 'Prev_GDP', 'Prev_EPS',
       'Prev_Revenue', 'WDay_Friday', 'WDay_Monday', 'WDay_Thursday',
       'WDay_Tuesday', 'WDay_Wednesday'],
      dtype='object')
```

Adding Tuesday and Wednesday

In [57]:

```
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_GDP','Open','Volume
                                                                  ,'WDay_Wedne
results = model.fit()

results.summary()
```

Out[57]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.602
Model:	OLS	**Adj. R-squared:**	0.601
Method:	Least Squares	**F-statistic:**	448.4
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5776.7
No. Observations:	2083	**AIC:**	1.157e+04
Df Residuals:	2075	**BIC:**	1.161e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-14.7123	2.618	-5.620	0.000	-19.846	-9.578
Prev_diff	0.3425	0.021	16.442	0.000	0.302	0.383
Prev_GDP	9.81e-13	1.84e-13	5.333	0.000	6.2e-13	1.34e-12
Open	0.0042	0.001	4.316	0.000	0.002	0.006
Volume	2.352e-07	1.87e-08	12.603	0.000	1.99e-07	2.72e-07
Prev_Revenue	4.043e-11	2.87e-11	1.408	0.159	-1.59e-11	9.67e-11
WDay_Tuesday	-0.1115	0.220	-0.507	0.612	-0.543	0.320
WDay_Wednesday	-0.1456	0.217	-0.672	0.502	-0.571	0.280

Omnibus:	2369.167	**Durbin-Watson:**	2.044
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	518333.983
Skew:	5.419	**Prob(JB):**	0.00
Kurtosis:	79.516	**Cond. No.**	4.99e+14

Replacing the days with other 3 days

In [58]:

```
model = sm.OLS(y,sm.add_constant(X[['Prev_diff','Prev_GDP','Open','Volume
                                                                  ,'WDay_Thurs
results = model.fit()

results.summary()
```

Out[58]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.602
Model:	OLS	**Adj. R-squared:**	0.601
Method:	Least Squares	**F-statistic:**	392.6
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5776.0
No. Observations:	2083	**AIC:**	1.157e+04
Df Residuals:	2074	**BIC:**	1.162e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-14.8265	2.624	-5.651	0.000	-19.972	-9.681
Prev_diff	0.3419	0.021	16.412	0.000	0.301	0.383
Prev_GDP	9.798e-13	1.84e-13	5.326	0.000	6.19e-13	1.34e-12
Open	0.0042	0.001	4.320	0.000	0.002	0.006
Volume	2.353e-07	1.87e-08	12.604	0.000	1.99e-07	2.72e-07
Prev_Revenue	4.095e-11	2.87e-11	1.426	0.154	-1.54e-11	9.73e-11
WDay_Monday	0.1913	0.238	0.802	0.422	-0.276	0.659
WDay_Thursday	-0.0425	0.232	-0.184	0.854	-0.497	0.412
WDay_Friday	0.2411	0.230	1.046	0.295	-0.211	0.693

Omnibus:	2366.759	**Durbin-Watson:**	2.046
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	515189.643
Skew:	5.410	**Prob(JB):**	0.00
Kurtosis:	79.281	**Cond. No.**	5.00e+14

Using all the columns

In [59]:

```
model = sm.OLS(y,sm.add_constant(X))
results = model.fit()

results.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	Diff	**R-squared:**	0.603
Model:	OLS	**Adj. R-squared:**	0.601
Method:	Least Squares	**F-statistic:**	349.3
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	18:04:46	**Log-Likelihood:**	-5775.1
No. Observations:	2083	**AIC:**	1.157e+04
Df Residuals:	2073	**BIC:**	1.163e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-14.6140	2.780	-5.257	0.000	-20.066	-9.162
Open	0.0037	0.001	3.526	0.000	0.002	0.006
Volume	2.332e-07	1.87e-08	12.439	0.000	1.96e-07	2.7e-07
Prev_diff	0.3421	0.021	16.423	0.000	0.301	0.383
Prev_GDP	1.177e-12	2.35e-13	5.008	0.000	7.16e-13	1.64e-12
Prev_EPS	0.3214	0.238	1.348	0.178	-0.146	0.789
Prev_Revenue	1.882e-11	3.31e-11	0.569	0.569	-4.6e-11	8.37e-11
WDay_Friday	-2.7614	0.580	-4.761	0.000	-3.899	-1.624
WDay_Monday	-2.8063	0.581	-4.829	0.000	-3.946	-1.667
WDay_Thursday	-3.0451	0.580	-5.250	0.000	-4.183	-1.908
WDay_Tuesday	-2.9822	0.582	-5.122	0.000	-4.124	-1.840
WDay_Wednesday	-3.0190	0.584	-5.168	0.000	-4.165	-1.873

Omnibus:	2370.085	**Durbin-Watson:**	2.043

Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	514643.717
Skew:	5.426	**Prob(JB):**	0.00
Kurtosis:	79.236	**Cond. No.**	2.03e+29

In [60]:
```
X.head()
```

Out[60]:

	Open	Volume	Prev_diff	Prev_GDP	Prev_EPS	Prev_Revenue	WDay_Friday	WDay_M
2203	82.68	7924000	2.77	1.438388e+13	0.41	4.889000e+09	0	
2287	51.66	1645500	1.49	1.484298e+13	0.27	4.265000e+09	0	
745	371.23	2786500	5.79	1.773593e+13	0.47	2.932900e+10	0	
1705	178.38	4616500	2.76	1.523837e+13	0.44	9.857000e+09	0	
1676	195.94	3409000	2.62	1.523837e+13	0.44	9.857000e+09	0	

In [61]:
```
data=X[['Prev_diff','Prev_GDP','Open','Volume','Prev_Revenue']]
```

In [62]:
```
data.head()
```

Out[62]:

	Prev_diff	Prev_GDP	Open	Volume	Prev_Revenue
2203	2.77	1.438388e+13	82.68	7924000	4.889000e+09
2287	1.49	1.484298e+13	51.66	1645500	4.265000e+09
745	5.79	1.773593e+13	371.23	2786500	2.932900e+10
1705	2.76	1.523837e+13	178.38	4616500	9.857000e+09
1676	2.62	1.523837e+13	195.94	3409000	9.857000e+09

Cross Validation

In [63]:
```
data.head()
```

Out[63]:

	Prev_diff	Prev_GDP	Open	Volume	Prev_Revenue
2203	2.77	1.438388e+13	82.68	7924000	4.889000e+09
2287	1.49	1.484298e+13	51.66	1645500	4.265000e+09
745	5.79	1.773593e+13	371.23	2786500	2.932900e+10
1705	2.76	1.523837e+13	178.38	4616500	9.857000e+09
1676	2.62	1.523837e+13	195.94	3409000	9.857000e+09

In [64]:
```
data_outlier=data
```

In [65]:
```
data_outlier.columns
```

Out[65]:
```
Index(['Prev_diff', 'Prev_GDP', 'Open', 'Volume', 'Prev_Revenue'], dty
pe='object')
```

Removing an outlier from the Diff column

In [66]:
```
y.argmax()
```

Out[66]:
```
159
```

In [67]:
```
y=y.drop(303)
```

In [68]:
```
data=data.drop(303)
```

In [70]:
```
len(data)
```

Out[70]:
```
2082
```

In [71]:
```
data.tail()
```

Out[71]:

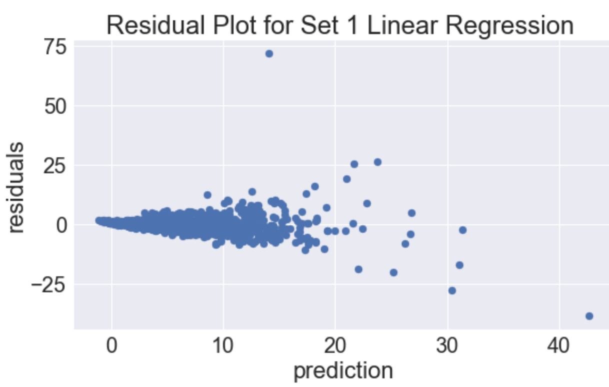
	Prev_diff	Prev_GDP	Open	Volume	Prev_Revenue
2009	4.58	1.456651e+13	117.12	12405900	9.520000e+09
1180	5.29	1.647544e+13	268.74	1741200	1.607000e+10
1344	2.87	1.612185e+13	261.74	6059300	1.283400e+10
527	10.97	1.822769e+13	666.83	2664000	2.535800e+10
1289	6.83	1.622794e+13	251.07	2628100	1.380600e+10

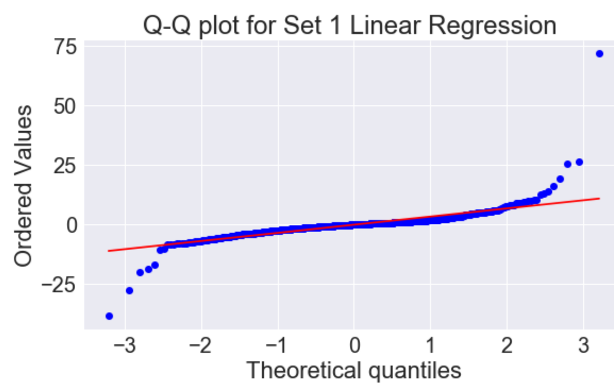
In [ ]:
```
```

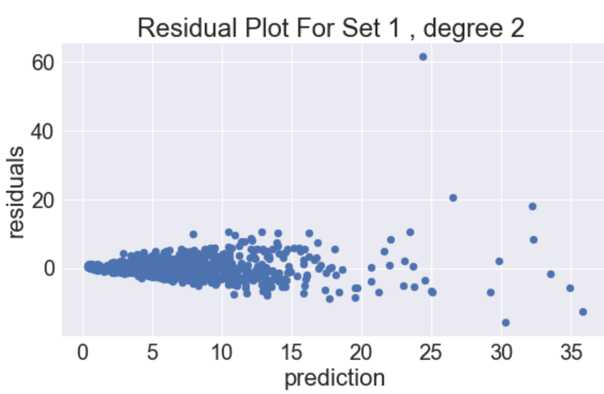
Cross Validation with Linear Regression and Polynomial Regression

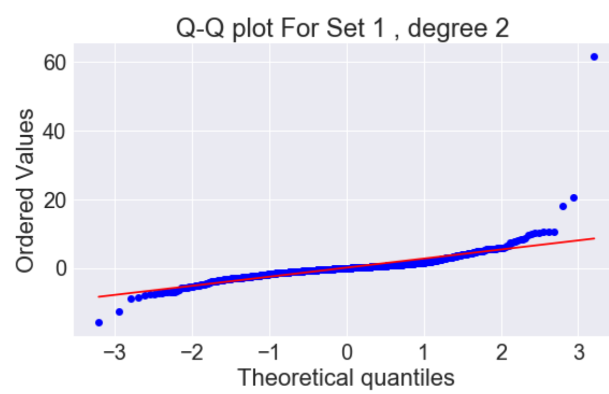
In [81]:
```
X_arr,y_arr=np.array(data),np.array(y)
kf = KFold(n_splits=2, shuffle=True, random_state = 20)
#collect the validation results for both models
linear_score= []
polynomial_2=[]
polynomial_3=[]
polynomial_4=[]

i=0
for train_ind, val_ind in kf.split(X_arr,y_arr):
    i+=1

    X_train, y_train = X_arr[train_ind], y_arr[train_ind]
    X_val, y_val = X_arr[val_ind], y_arr[val_ind]

    ## Linear Regression
    linear = LinearRegression()
    linear.fit(X_train,y_train)
    linear_predict = linear.predict(X_val)
    #print('Linear Regression Score for Validation Set {} :{}'.format(i,l
    linear_score.append(linear.score(X_val,y_val))

    ## Plot the residuals
    residual = y_val - linear_predict
    create_plots(residual,linear_predict,'for Set {} Linear Regression '.
    #create_YX_plot(y_val,X_val,linear.coef_)


    for degree in range(2,5):
```

```
        ## Polynomial Regression
        poly=PolynomialFeatures(degree,interaction_only=True)
        X_fit=poly.fit_transform(X_train)
        model=linear_model.LinearRegression(fit_intercept=True)
        model.fit(X_fit,y_train)
        poly_predict=model.predict(poly.fit_transform(X_val))
        #print(poly_predict.)
        #print('Polynomial degree {} score for set {} :'.format(degree,i)
        #print(model.score(poly.fit_transform(X_train),y_train))
        #print(model.score(poly.fit_transform(X_val),y_val))
        if degree ==2:
            polynomial_2.append(model.score(poly.fit_transform(X_val),y_v
        elif degree==3:
            polynomial_3.append(model.score(poly.fit_transform(X_val),y_v
        else:
            polynomial_4.append(model.score(poly.fit_transform(X_val),y_v

        ## Plot the residual Plots
        residual_poly = y_val-poly_predict
        create_plots(residual_poly,poly_predict,'For Set {} , degree {} '


print('Mean Linear Regression Score :{}'.format(np.mean(linear_score)))
print('Polynomial Regression Score for degree 2 : {}'.format(np.mean(poly
print('Polynomial Regression Score for degree 3 : {}'.format(np.mean(poly
print('Polynomial Regression Score for degree 4 : {}'.format(np.mean(poly
```



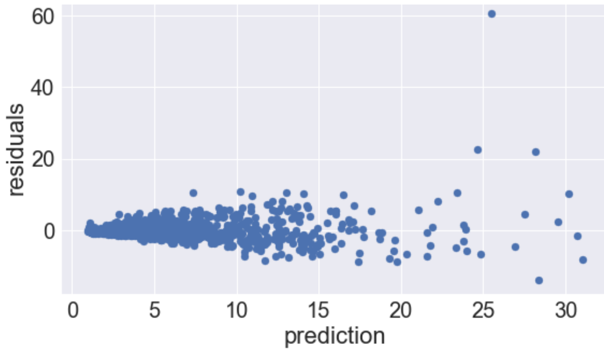


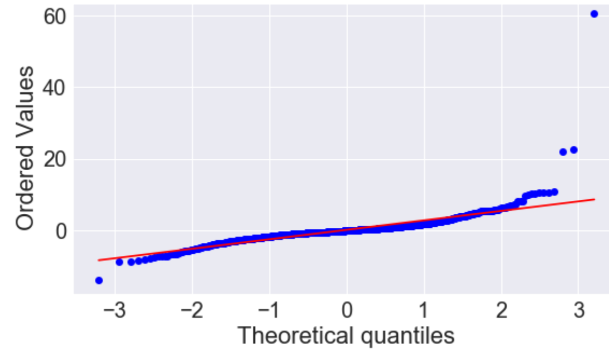




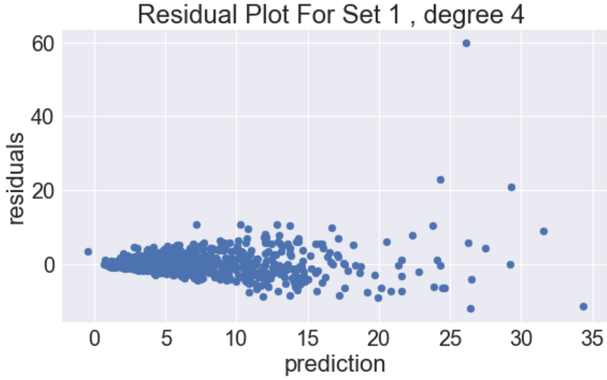
Residual Plot For Set 1 , degree 3

Q-Q plot For Set 1 , degree 3

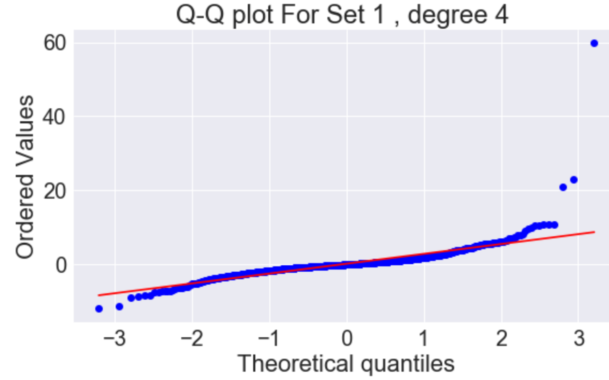




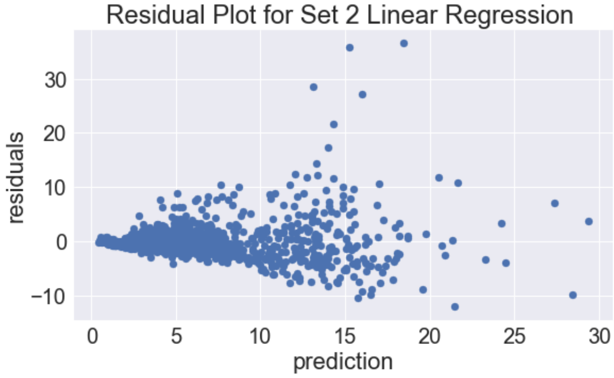
Residual Plot For Set 1 , degree 4



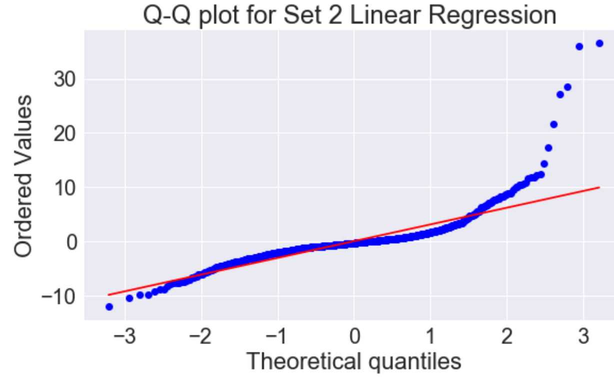
Q-Q plot For Set 1 , degree 4



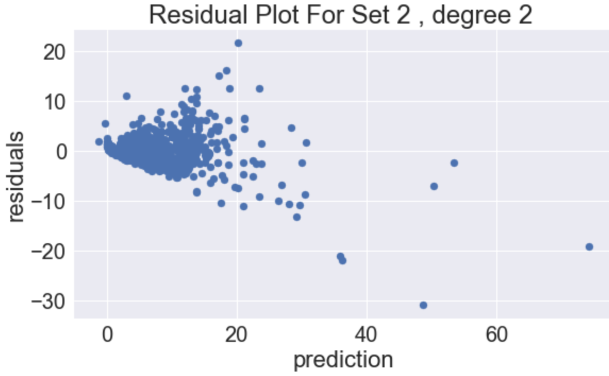
Residual Plot for Set 2 Linear Regression



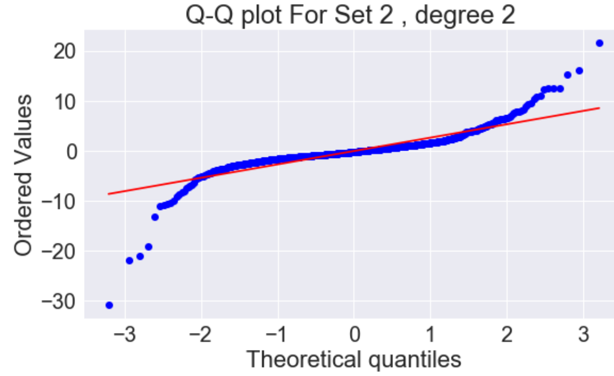
Q-Q plot for Set 2 Linear Regression



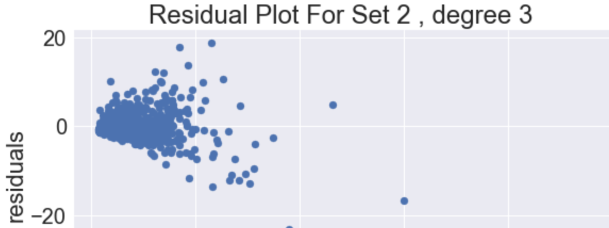
Residual Plot For Set 2 , degree 2



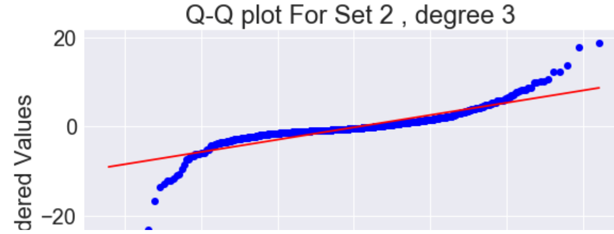
Q-Q plot For Set 2 , degree 2

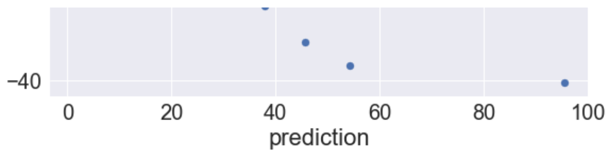


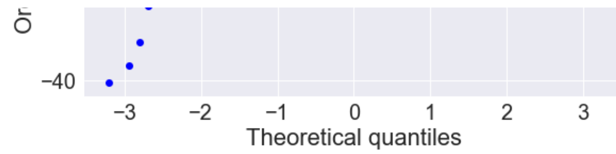
Residual Plot For Set 2 , degree 3

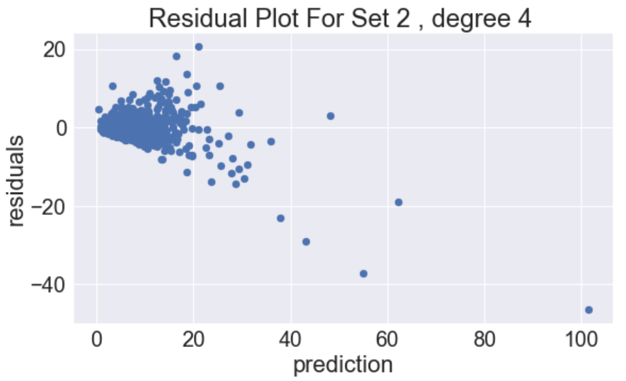


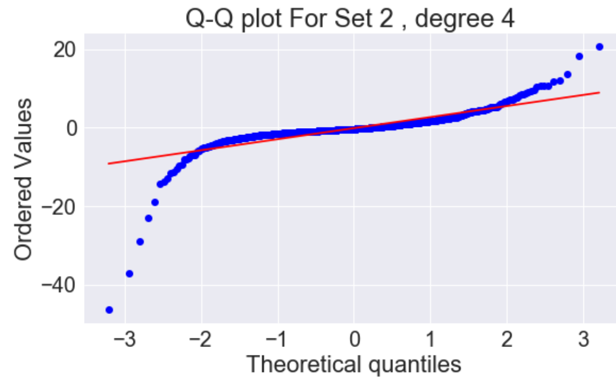
Q-Q plot For Set 2 , degree 3












```
Mean Linear Regression Score :0.5728391273811562
Polynomial Regression Score for degree 2 : 0.7257032643845746
Polynomial Regression Score for degree 3 : 0.6961484561558975
Polynomial Regression Score for degree 4 : 0.6862946448389207
```

In [82]:
```
data.head()
```

Out[82]:

	Prev_diff	Prev_GDP	Open	Volume	Prev_Revenue
2203	2.77	1.438388e+13	82.68	7924000	4.889000e+09
2287	1.49	1.484298e+13	51.66	1645500	4.265000e+09
745	5.79	1.773593e+13	371.23	2786500	2.932900e+10
1705	2.76	1.523837e+13	178.38	4616500	9.857000e+09
1676	2.62	1.523837e+13	195.94	3409000	9.857000e+09

In [ ]:

.752 for degree 2 with Interaction true

.735 for degree 2 with INteraction False

.742 for degree 4 with Interaction False

on the basis of high scores going ahead with degree 2

plotting Y_pred against y_actual

In [79]:
```
def create_YX_plot(y_val,X_val,coef):
    y_pred=np.array([x[0]*coef[0] + x[1]*coef[1] + x[2]*coef[2] + x[3]*co
    plt.scatter(y_pred,y_val)
    plt.xlabel('Target Predicted')
    plt.ylabel('Target Actuals')
    plt.show()
```

In [108]:
```
len(poly_predict)
```

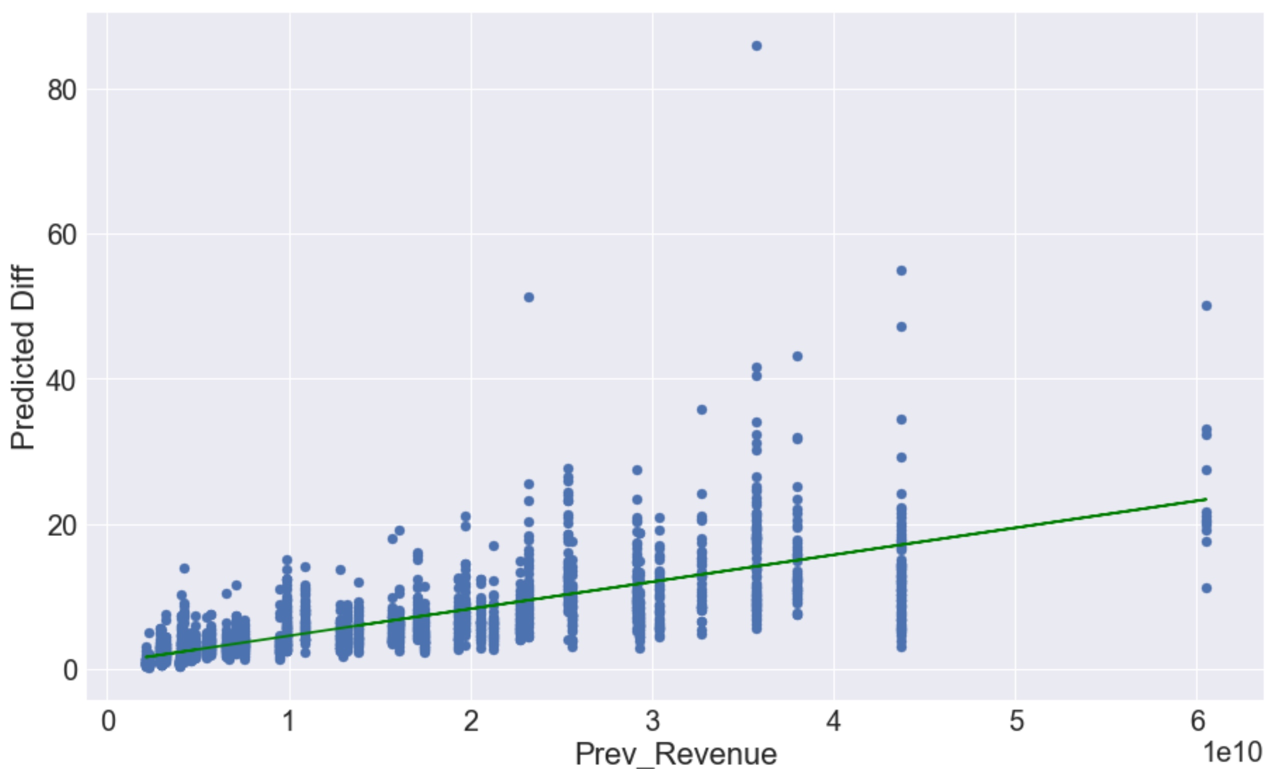
Out[108]: 1041

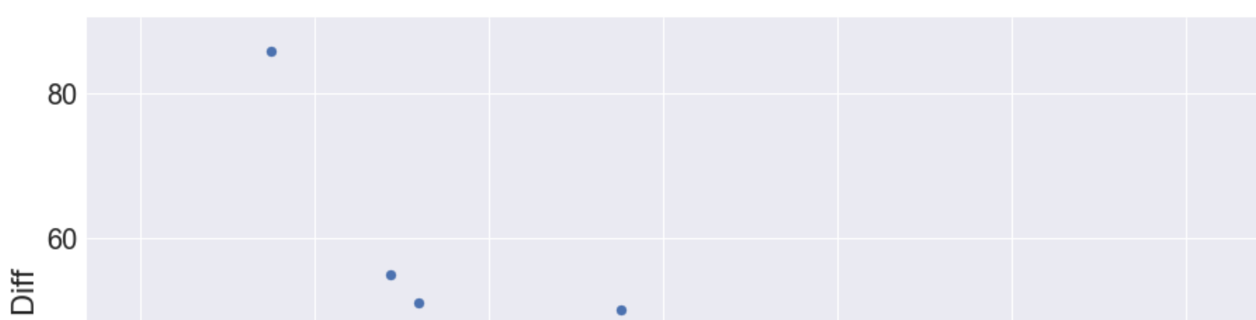
In [ ]:

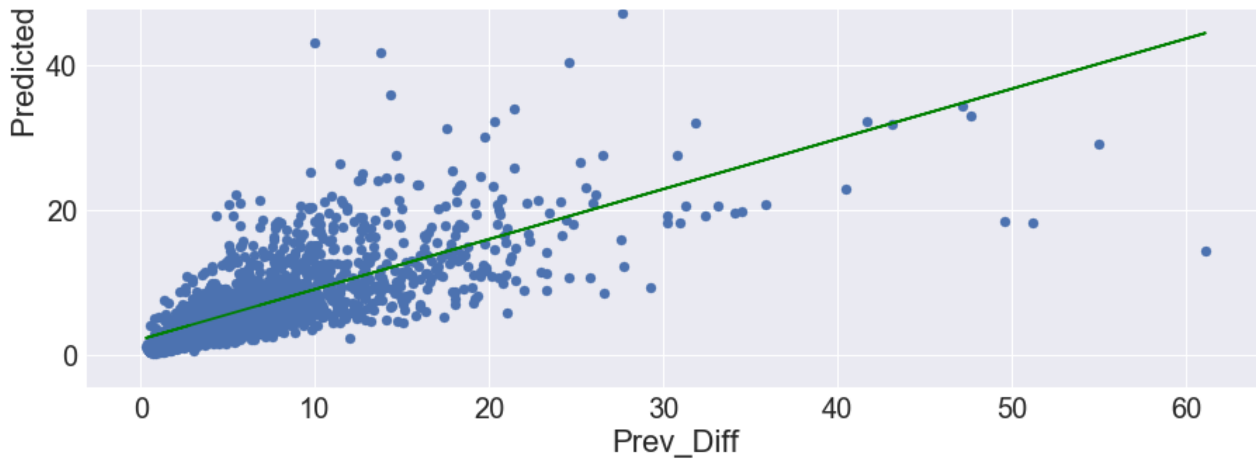
In [110]:
```
poly_predict=model.predict(poly.fit_transform(X_val))
plt.figure(figsize=(10,5))

fig,ax=plt.subplots()
fit=np.polyfit(X_val[:,4],poly_predict,deg=1)
ax.plot(X_arr[:,4],fit[0]*X_arr[:,4] + fit[1],color='green')
ax.scatter(X_arr[:,4],y_arr)
plt.xlabel('Prev_Revenue')
plt.ylabel('Predicted Diff')

fig,ax=plt.subplots()
fit=np.polyfit(X_val[:,0],poly_predict,deg=1)
ax.plot(X_arr[:,0],fit[0]*X_arr[:,0] + fit[1],color='green')
ax.scatter(X_arr[:,0],y_arr)
plt.xlabel('Prev_Diff')
plt.ylabel('Predicted Diff')


fig,ax=plt.subplots()#Prev_diff Prev_GDP    Open    Volume  Prev_Revenue
fit=np.polyfit(X_val[:,1],poly_predict,deg=1)
ax.plot(X_arr[:,1],fit[0]*X_arr[:,1] + fit[1],color='green')
ax.scatter(X_arr[:,1],y_arr)
plt.xlabel('Prev_GDP')
plt.ylabel('Predicted Diff')


fig,ax=plt.subplots()
```

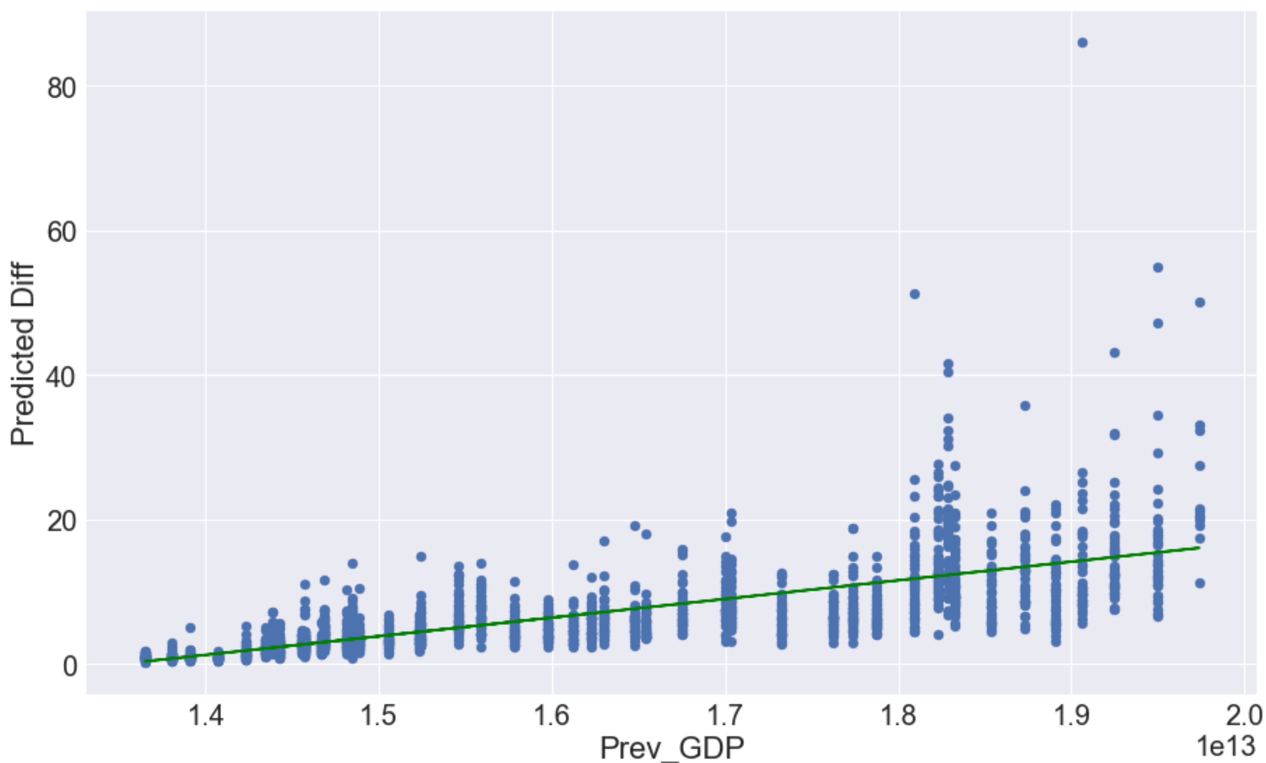
```
fig,ax=plt.subplots()
fit=np.polyfit(X_val[:,2],poly_predict,deg=1)
ax.plot(X_arr[:,2],fit[0]*X_arr[:,2] + fit[1],color='green')
ax.scatter(X_arr[:,2],y_arr)
plt.xlabel('Open')
plt.ylabel(' Predicted Diff')


fig,ax=plt.subplots()
fit=np.polyfit(X_val[:,3],poly_predict,deg=1)
ax.plot(X_arr[:,3],fit[0]*X_arr[:,3] + fit[1],color='green')
ax.scatter(X_arr[:,3],y_arr)
plt.xlabel('Volume')
plt.ylabel('Predicted Diff')
```

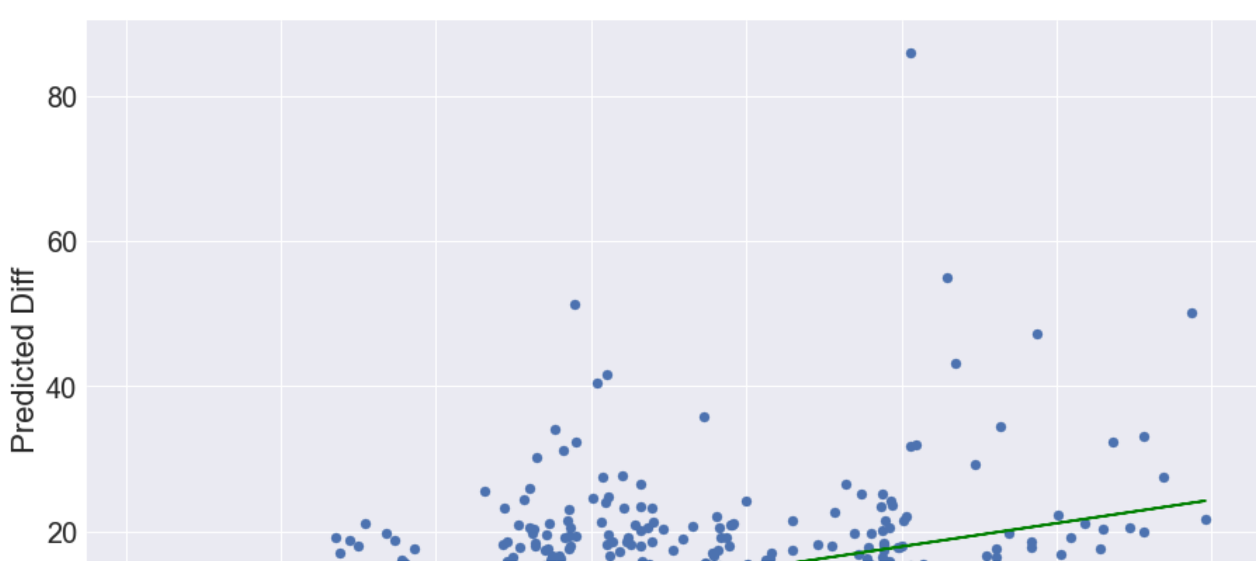
```
Out[110]: Text(0,0.5,'Predicted Diff')

<matplotlib.figure.Figure at 0x1c16748390>
```

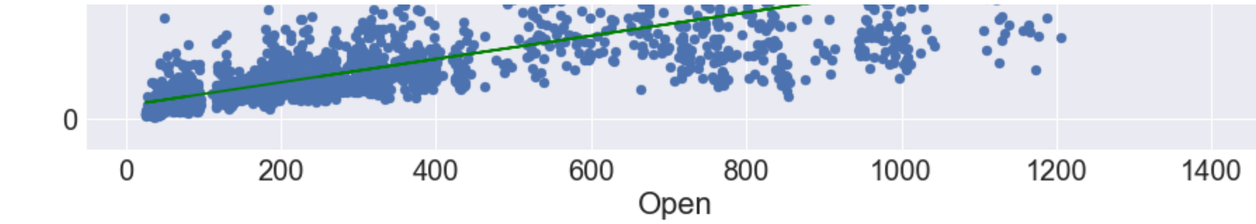


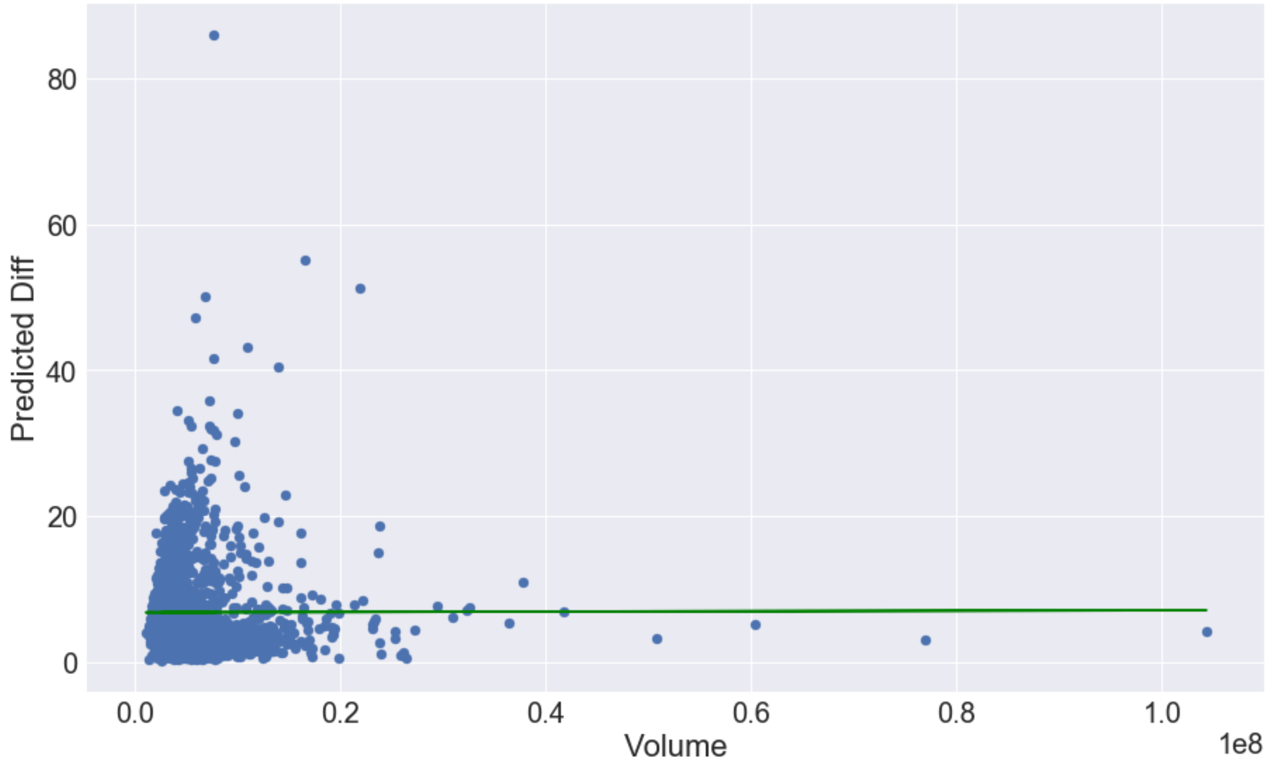



Predicted
0
20
40
0
10
20
30
40
50
60
Prev_Diff


Predicted Diff
0
20
40
60
80
1.4
1.5
1.6
1.7
1.8
1.9
2.0
1e13
Prev_GDP


Predicted Diff
20
40
60
80

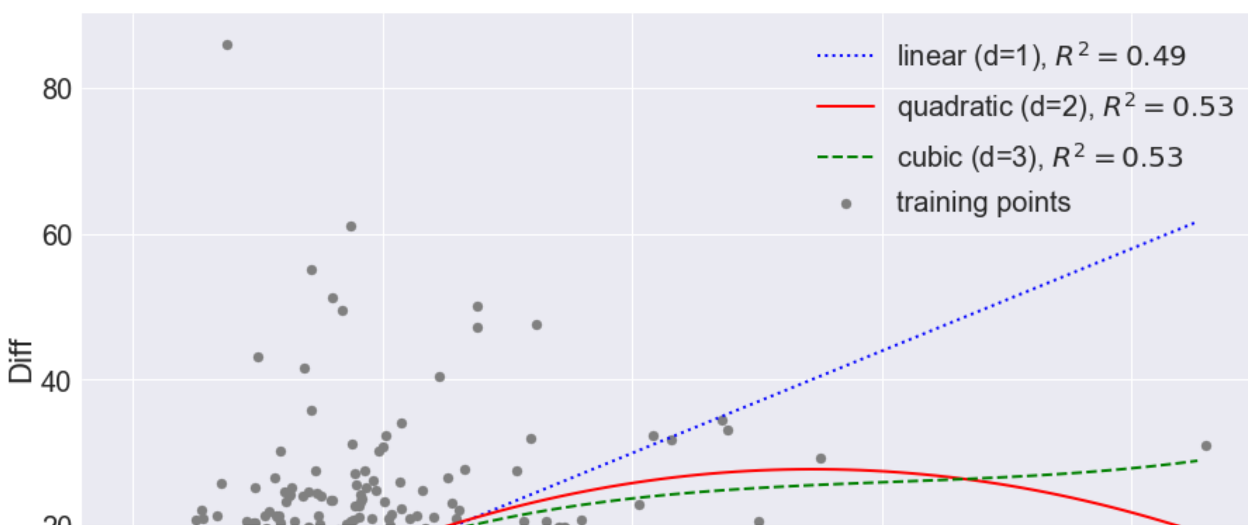


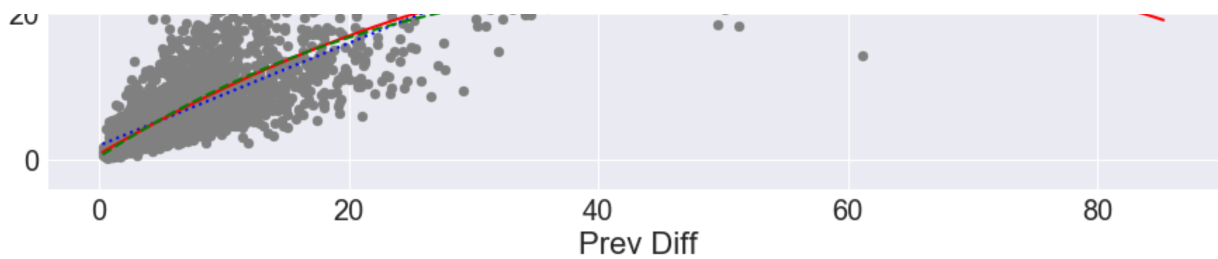


In [134]:

```
L = master[['Prev_diff']].values
y = master['Diff'].values
regr = LinearRegression()

quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
four = PolynomialFeatures(degree=4)
five = PolynomialFeatures(degree=5)
L_quad = quadratic.fit_transform(L)
L_cubic = cubic.fit_transform(L)
L_four = four.fit_transform(L)
L_five = five.fit_transform(L)

# linear fit
L_fit = np.arange(L.min(), L.max(), 1)[:, np.newaxis]
regr = regr.fit(L,y)
y_lin_fit = regr.predict(L_fit)
linear_r2 = r2_score(y, regr.predict(L))

# quadratic fit
regr = regr.fit(L_quad, y)
```

```
y_quad_fit = regr.predict(quadratic.fit_transform(L_fit))
quadratic_r2 = r2_score(y, regr.predict(L_quad))

# cubic fit
regr = regr.fit(L_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(L_fit))
cubic_r2 = r2_score(y, regr.predict(L_cubic))

#Plot results
plt.scatter(L, y,
           label='training points',
           color='gray')
plt.plot(L_fit, y_lin_fit,
        label='linear (d=1), $R^2=%.2f$'
        % linear_r2,
        color='blue',
        lw=2,
        linestyle=':')
plt.plot(L_fit, y_quad_fit,
        label='quadratic (d=2), $R^2=%.2f$'
        % quadratic_r2,
        color='red',
        lw=2,
        linestyle='-')
plt.plot(L_fit, y_cubic_fit,
        label='cubic (d=3), $R^2=%.2f$'
        % cubic_r2,
        color='green',
        lw=2,
        linestyle='--')

plt.xlabel('Prev Diff')
plt.ylabel('Diff')
plt.legend(loc='upper right')
plt.show()
```

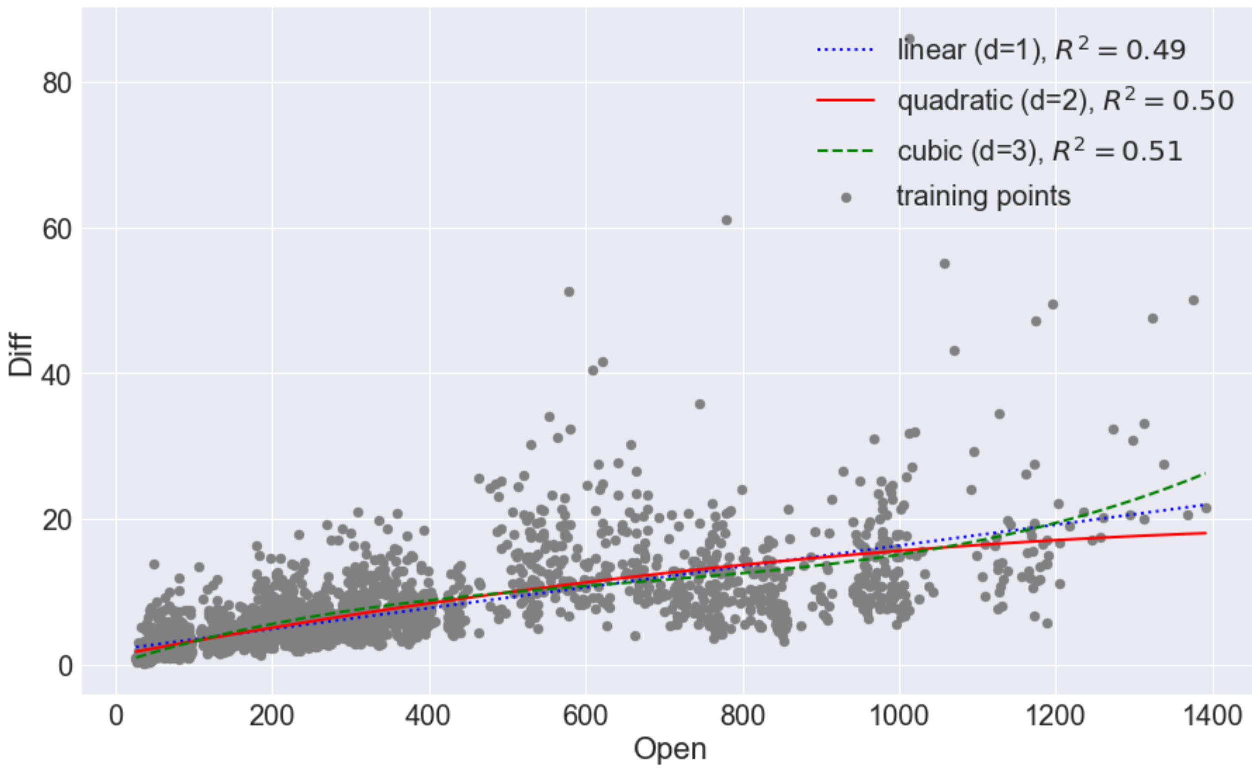




In [135]:

```
L = master[['Open']].values
y = master['Diff'].values
regr = LinearRegression()

quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
four = PolynomialFeatures(degree=4)
five = PolynomialFeatures(degree=5)
L_quad = quadratic.fit_transform(L)
L_cubic = cubic.fit_transform(L)
L_four = four.fit_transform(L)
L_five = five.fit_transform(L)

# linear fit
L_fit = np.arange(L.min(), L.max(), 1)[:, np.newaxis]
regr = regr.fit(L,y)
y_lin_fit = regr.predict(L_fit)
linear_r2 = r2_score(y, regr.predict(L))

# quadratic fit
regr = regr.fit(L_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(L_fit))
quadratic_r2 = r2_score(y, regr.predict(L_quad))

# cubic fit
regr = regr.fit(L_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(L_fit))
cubic_r2 = r2_score(y, regr.predict(L_cubic))

#Plot results
plt.scatter(L, y,
            label='training points',
            color='gray')
plt.plot(L_fit, y_lin_fit,
        label='linear (d=1), $R^2=%.2f$'
        % linear_r2,
        color='blue',
        lw=2,
        linestyle=':')
plt.plot(L_fit, y_quad_fit,
        label='quadratic (d=2), $R^2=%.2f$'
        % quadratic_r2,
```

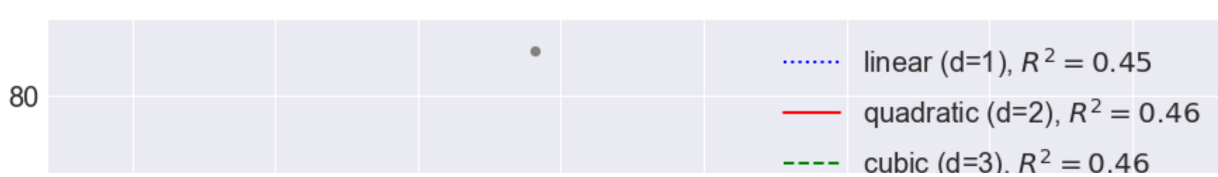
```
        color='red',
        lw=2,
        linestyle='-')
plt.plot(L_fit, y_cubic_fit,
        label='cubic (d=3), $R^2=%.2f$'
        % cubic_r2,
        color='green',
        lw=2,
        linestyle='--')

plt.xlabel('Open')
plt.ylabel('Diff')
plt.legend(loc='upper right')
plt.show()
```

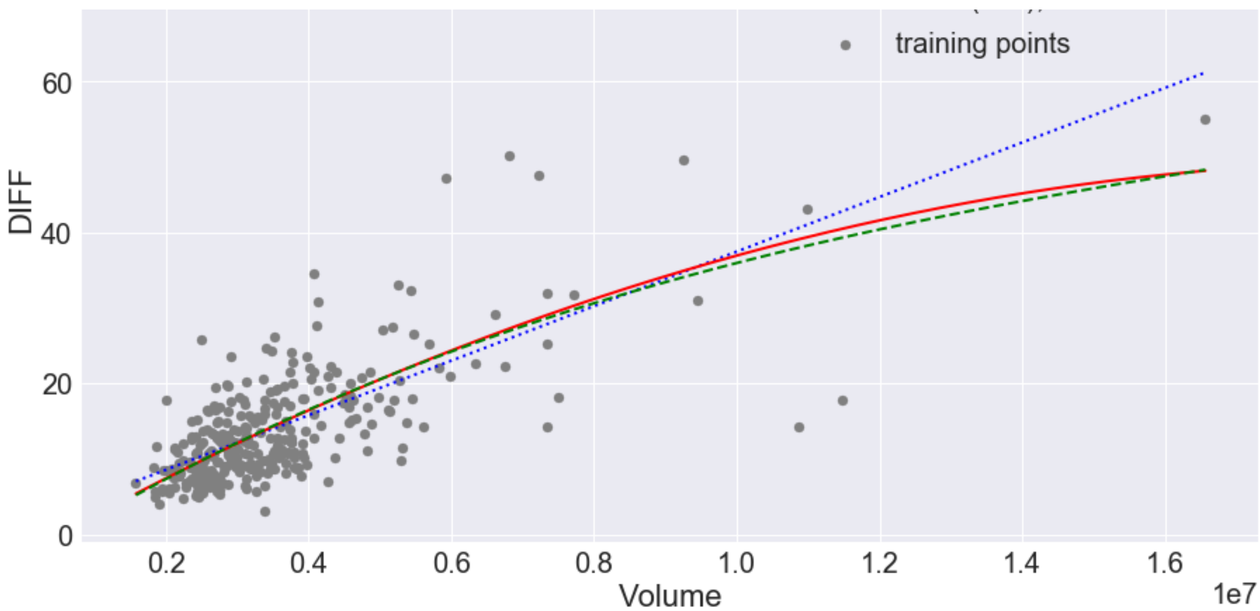


In [140]:

```
df=master.dropna()
L = df[['Volume']].head(300).values
y = df['Diff'].head(300).values
regr = LinearRegression()

quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
four = PolynomialFeatures(degree=4)
five = PolynomialFeatures(degree=5)
L_quad = quadratic.fit_transform(L)
L_cubic = cubic.fit_transform(L)
##L_four = four.fit_transform(L)
#L_five = five.fit_transform(L)
```

```
# linear fit
L_fit = np.arange(L.min(), L.max(), 1)[:, np.newaxis]
regr = regr.fit(L,y)
y_lin_fit = regr.predict(L_fit)
linear_r2 = r2_score(y, regr.predict(L))

# quadratic fit
regr = regr.fit(L_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(L_fit))
quadratic_r2 = r2_score(y, regr.predict(L_quad))

# cubic fit
regr = regr.fit(L_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(L_fit))
cubic_r2 = r2_score(y, regr.predict(L_cubic))

#
#Plot results
plt.scatter(L, y,
           label='training points',
           color='gray')
plt.plot(L_fit, y_lin_fit,
        label='linear (d=1), $R^2=%.2f$'
        % linear_r2,
        color='blue',
        lw=2,
        linestyle=':')
plt.plot(L_fit, y_quad_fit,
        label='quadratic (d=2), $R^2=%.2f$'
        % quadratic_r2,
        color='red',
        lw=2,
        linestyle='-')
plt.plot(L_fit, y_cubic_fit,
        label='cubic (d=3), $R^2=%.2f$'
        % cubic_r2,
        color='green',
        lw=2,
        linestyle='--')

plt.xlabel('Volume')
plt.ylabel('DIFF')
plt.legend(loc='upper right')
plt.show()
```





In [ ]:

```
df=master.dropna()
L = df[['Prev_GDP']].head(300).values
y = df['Diff'].head(300).values
regr = LinearRegression()

quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
four = PolynomialFeatures(degree=4)
five = PolynomialFeatures(degree=5)
L_quad = quadratic.fit_transform(L)
L_cubic = cubic.fit_transform(L)
#L_four = four.fit_transform(L)
#L_five = five.fit_transform(L)

# linear fit
L_fit = np.arange(L.min(), L.max(), 1)[:, np.newaxis]
regr = regr.fit(L,y)
y_lin_fit = regr.predict(L_fit)
linear_r2 = r2_score(y, regr.predict(L))

# quadratic fit
regr = regr.fit(L_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(L_fit))
quadratic_r2 = r2_score(y, regr.predict(L_quad))

# cubic fit
regr = regr.fit(L_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(L_fit))
cubic_r2 = r2_score(y, regr.predict(L_cubic))

#
#Plot results
```

```
plt.scatter(L, y,
            label='training points',
            color='gray')
plt.plot(L_fit, y_lin_fit,
        label='linear (d=1), $R^2=%.2f$'
        % linear_r2,
        color='blue',
        lw=2,
        linestyle=':')
plt.plot(L_fit, y_quad_fit,
        label='quadratic (d=2), $R^2=%.2f$'
        % quadratic_r2,
        color='red',
        lw=2,
        linestyle='-')
plt.plot(L_fit, y_cubic_fit,
        label='cubic (d=3), $R^2=%.2f$'
        % cubic_r2,
        color='green',
        lw=2,
        linestyle='--')

plt.xlabel('Volume')
plt.ylabel('DIFF')
plt.legend(loc='upper right')
plt.show()
```

In [ ]:

```
df=master.dropna()
L = df[['Prev_GDP']].head(300).values
y = df['Diff'].head(300).values
regr = LinearRegression()

quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
four = PolynomialFeatures(degree=4)
five = PolynomialFeatures(degree=5)
L_quad = quadratic.fit_transform(L)
L_cubic = cubic.fit_transform(L)
#L_four = four.fit_transform(L)
#L_five = five.fit_transform(L)

# linear fit
L_fit = np.arange(L.min(), L.max(), 1)[:, np.newaxis]
regr = regr.fit(L,y)
y_lin_fit = regr.predict(L_fit)
linear_r2 = r2_score(y, regr.predict(L))

# quadratic fit
regr = regr.fit(L_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(L_fit))
```

```
quadratic_r2 = r2_score(y, regr.predict(L_quad))

# cubic fit
regr = regr.fit(L_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(L_fit))
cubic_r2 = r2_score(y, regr.predict(L_cubic))

#
#Plot results
plt.scatter(L, y,
           label='training points',
           color='gray')
plt.plot(L_fit, y_lin_fit,
        label='linear (d=1), $R^2=%.2f$'
        % linear_r2,
        color='blue',
        lw=2,
        linestyle=':')
plt.plot(L_fit, y_quad_fit,
        label='quadratic (d=2), $R^2=%.2f$'
        % quadratic_r2,
        color='red',
        lw=2,
        linestyle='-')
plt.plot(L_fit, y_cubic_fit,
        label='cubic (d=3), $R^2=%.2f$'
        % cubic_r2,
        color='green',
        lw=2,
        linestyle='--')

plt.xlabel('Volume')
plt.ylabel('DIFF')
plt.legend(loc='upper right')
plt.show()
```

In [ ]:

In [ ]:

In [84]:
```
y_arr.shape
```

Out[84]: (2082,)

Creating an numpy array for the required columns from the Test Data set

In [85]:
```
X_test_arr=np.array(X_test[['Open', 'Volume', 'Prev_diff', 'Prev_GDP',
       'Prev_Revenue']])

y_test_arr=np.array(y_test)
```

Calculating Alpha for the Polynomial Features degree 2 with Lasso

In [86]:
```
alphalist = 10**(np.linspace(3,9,1000))

err_vec_test = np.zeros(len(alphalist))
err_vec_train = np.zeros(len(alphalist))

for i,curr_alpha in enumerate(alphalist):

    est = make_pipeline(PolynomialFeatures(2,interaction_only=True), Lass
    est.fit(X_train, y_train)

    ## Predicting the score for the test set
    y_predict=est.score(X_test_arr,y_test)
    err_vec_test[i] = np.sqrt(np.mean((y_predict - y_test)**2))
    ## Predicting the score for the validation test
    y_val_predict = est.score(X_val,y_val)
    err_vec_train[i] = np.sqrt(np.mean((y_val_predict - y_val)**2))
#print('Minimum Error for Test Set:{} and alpha :{}'.format(err_vec_test.
print('Minimum Error for Validation Set:{} and alpha :{}'.format(err_vec_
```

```
Minimum Error for Validation Set:8.441623676355176 and alpha :17752080
1.1717636
```

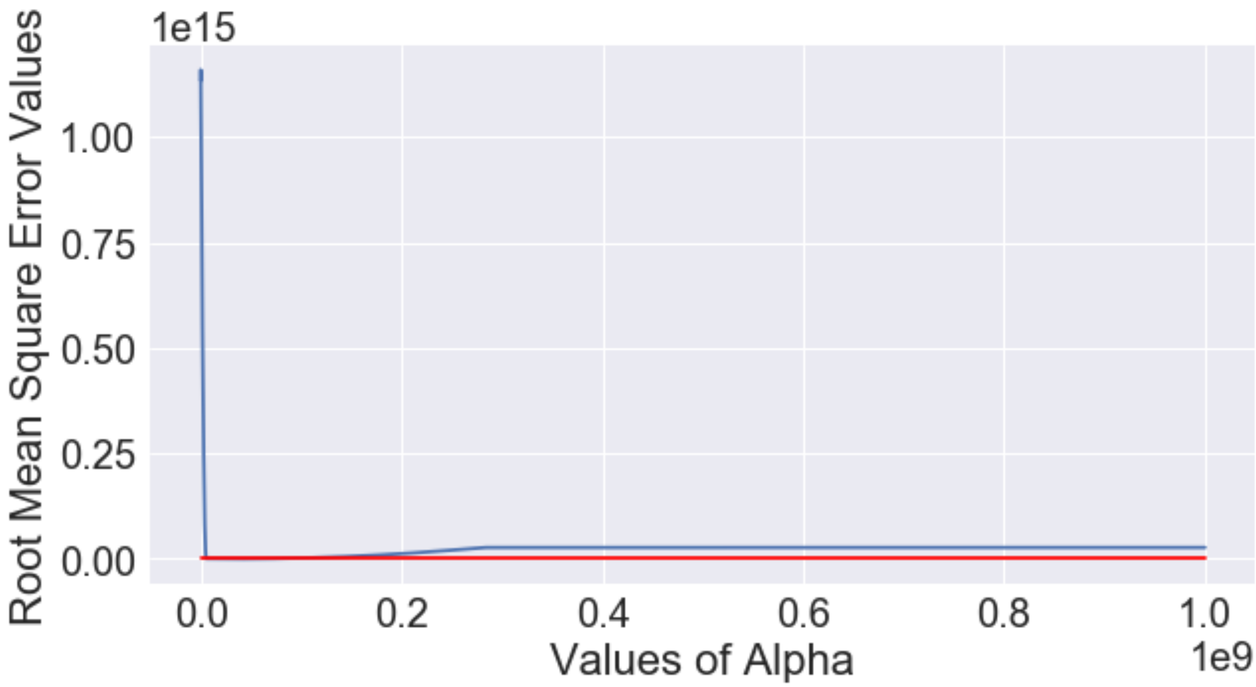
In [ ]:

In [87]:
```
est = make_pipeline(PolynomialFeatures(2,interaction_only=True), Lasso(al
est.fit(X_train, y_train)
print('Training Data Score: ',est.score(X_train,y_train))
y_predict=est.score(X_val,y_val)
print('Validation Data Score: ',est.score(X_val,y_val))
y_predict=est.score(X_test_arr,y_test)
print('Test Data Score :',est.score(X_test_arr,y_test_arr))
```

```
Training Data Score:  0.747877312696
Validation Data Score:  0.735935677468
Test Data Score : -1.15591914746e+15
```

In [88]:
```
plt.figure(figsize=(10,5))
plt.plot((alphalist),err_vec_test)
plt.plot((alphalist),err_vec_train,c='r')
plt.xlabel('Values of Alpha')
plt.ylabel('Root Mean Square Error Values')
```

Out[88]: `Text(0,0.5,'Root Mean Square Error Values')`



Test Error is huge wrt Validation and Training..

Trying Linear Regression with Lasso

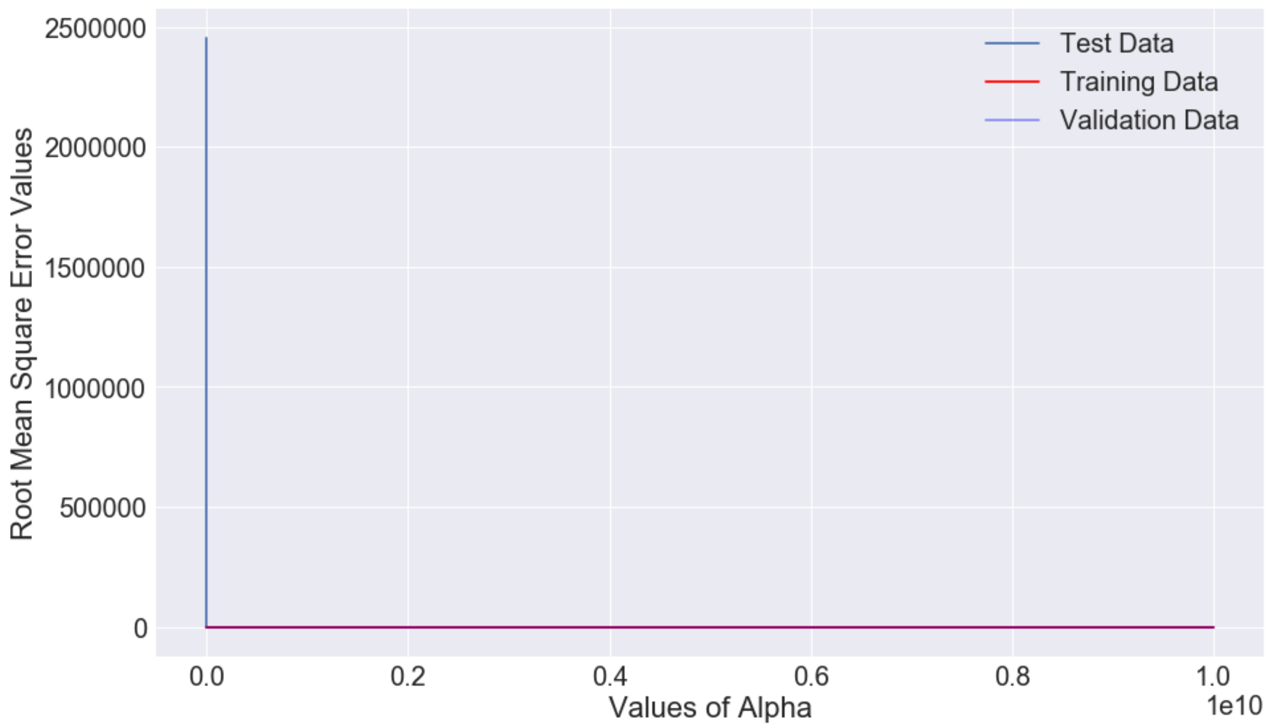
In [89]:
```
## Test forLInear Regression Lasso
alphalist = 10**(np.linspace(-10,10,1000))
err_lr_test = np.zeros(len(alphalist))
err_lr_train = np.zeros(len(alphalist))
err_lr_val = np.zeros(len(alphalist))

for i,curr_alpha in enumerate(alphalist):

    steps = [('standardize', StandardScaler()), ('lasso', Lasso(alpha = c

    pipe = Pipeline(steps)
    pipe.fit(X_train, y_train)
    test_set_pred = pipe.predict(X_test_arr)
    err_lr_test[i] = np.sqrt(np.mean((test_set_pred - y_test)**2))

    val_set_pred = pipe.predict(X_val)
    err_lr_val[i] = np.sqrt(np.mean((val_set_pred - y_val)**2))

    train_set_pred = pipe.predict(X_train)
    err_lr_train[i] = np.sqrt(np.mean((train_set_pred - y_train)**2))

#print('Minimum Error for Test Set:{} and alpha :{}'.format(err_lr_test.m
print('Minimum Error for Validation Set:{} and alpha :{}'.format(err_lr_v
print('Minimum Error for Validation Set:{} and alpha :{}'.format(err_lr_t
```

```
Minimum Error for Validation Set:3.6022002430808127 and alpha :1.32777
08293554292e-07
Minimum Error for Validation Set:3.9413376543638803 and alpha :1.04717
68194855202e-10
```

PLotting the Training , Test and Validation errors

In [90]:
```
plt.plot((alphalist),err_lr_test)
plt.plot((alphalist),err_lr_train,c='r')
plt.plot((alphalist),err_lr_val,c='b',alpha=.4)
plt.legend(['Test Data','Training Data','Validation Data'],loc=1)
plt.xlabel('Values of Alpha')
plt.ylabel('Root Mean Square Error Values')
```

Out[90]:
```
Text(0,0.5,'Root Mean Square Error Values')
```



Error for Test is huge

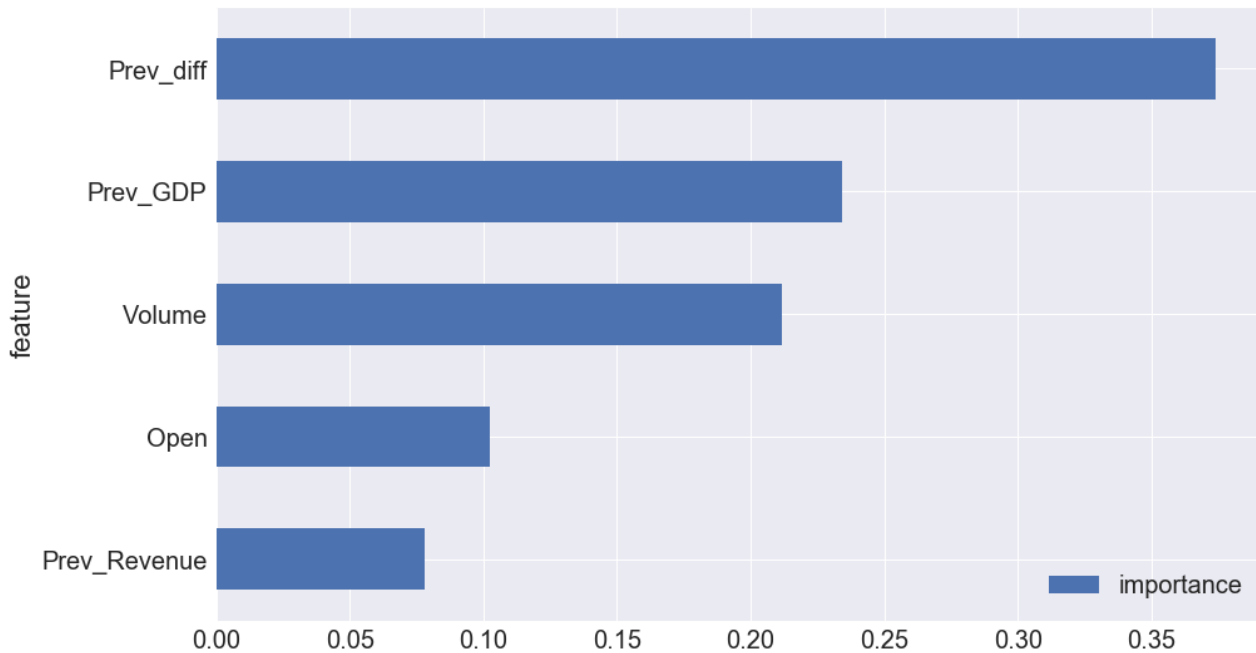
Testing Random Forest Regressor to understand which all features are important

In [92]:
```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre

rf = RandomForestRegressor(n_estimators=1800, max_features=3)
rf.fit(X_train, y_train)
print('Score for Validation Data :',rf.score(X_val, y_val))
print('Score for Test Data :',rf.score(X_test_arr, y_test_arr))
```

```
Score for Validation Data : 0.779524928367
Score for Test Data : 0.13684149301
```

In [93]:
```
feat_imps = list(zip(['Open', 'Volume', 'Prev_diff', 'Prev_GDP',
       'Prev_Revenue'],rf.feature_importances_))
feat_imps = sorted(feat_imps, key = lambda x: x[1], reverse=False)
feat_imps = pd.DataFrame(feat_imps, columns=['feature','importance'])

feat_imps.plot(x='feature',y='importance',kind='barh')
```

Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x1c27f4a588>



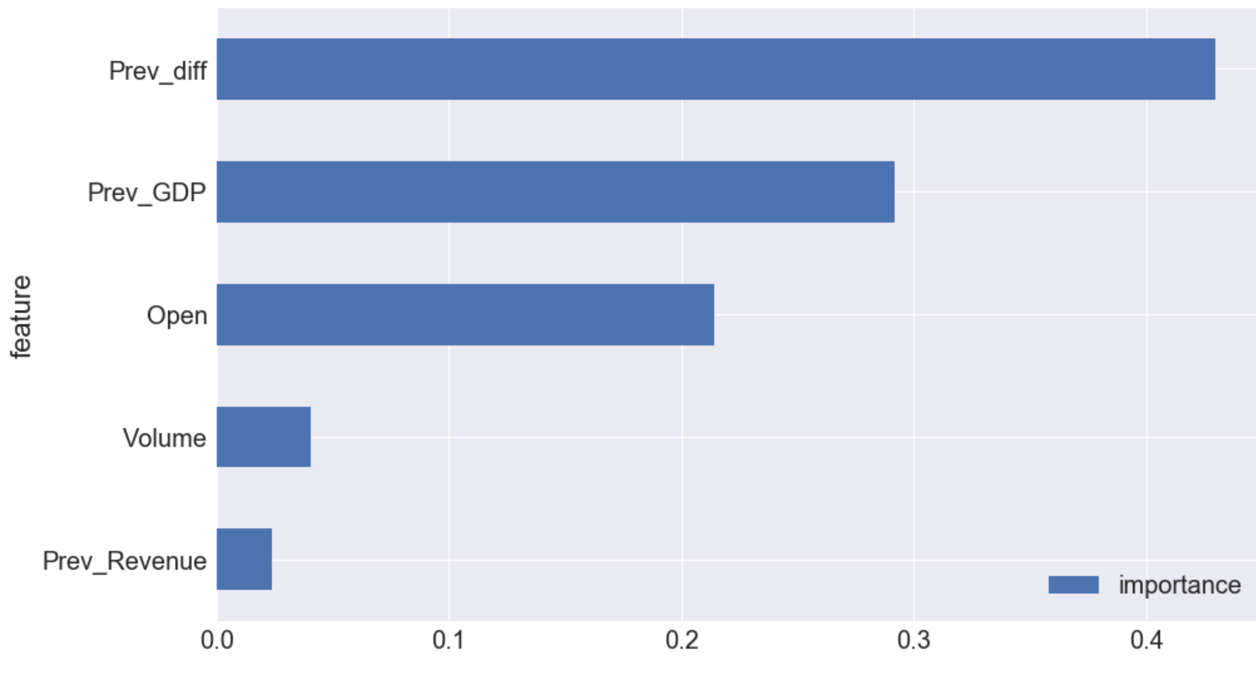
Testing Gradient Boosting Regressor to understand which all features are important

In [94]:
```
gbm = GradientBoostingRegressor(n_estimators=1600, max_depth=3, learning_
gbm.fit(X_train, y_train)
print('Score for Validation Data :',gbm.score(X_val, y_val))
print('Score for Test Data :',gbm.score(X_test_arr, y_test_arr))
```

```
Score for Validation Data : 0.776272236596
Score for Test Data : -0.0950690457442
```

In [95]:
```
feat_imps = list(zip(['Open', 'Volume', 'Prev_diff', 'Prev_GDP',
       'Prev_Revenue'],gbm.feature_importances_))
feat_imps = sorted(feat_imps, key = lambda x: x[1], reverse=False)
feat_imps = pd.DataFrame(feat_imps, columns=['feature','importance'])

feat_imps.plot(x='feature',y='importance',kind='barh')
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x1c276338d0>



As Open and Previous Revenue column are not that important as indicated by RandomForest implementing that in Polynomial

Removing Open and Prev Revenue to try and bring the score of Test Up.

In [96]:
```
data_modified=data[['Prev_diff','Prev_GDP','Volume']]## training Data
```

In [97]:
```
X_train_mod_arr=np.array(data_modified)
```

In [98]:
```
y_train_mod_arr=np.array(y)
```

In [99]:
```
X_test[['Prev_diff','Prev_GDP','Volume']].describe()
```

Out[99]:

	Prev_diff	Prev_GDP	Volume
count	894.000000	8.940000e+02	8.940000e+02
mean	6.502796	1.620826e+13	5.823059e+06
std	5.802910	1.747113e+12	5.288041e+06
min	0.320000	1.364890e+13	9.844000e+05
25%	2.920000	1.466844e+13	3.047550e+06
50%	4.910000	1.597388e+13	4.549100e+06
75%	8.270000	1.773593e+13	6.980175e+06
max	85.990000	1.973889e+13	6.217950e+07

In [128]:
```
model = sm.OLS(y_train_mod_arr,sm.add_constant(X_train_mod_arr))
results = model.fit()

results.summary()
```

Out[128]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.605
Model:	OLS	**Adj. R-squared:**	0.604
Method:	Least Squares	**F-statistic:**	1061.
Date:	Thu, 01 Feb 2018	**Prob (F-statistic):**	0.00
Time:	20:40:34	**Log-Likelihood:**	-5726.4
No. Observations:	2082	**AIC:**	1.146e+04
Df Residuals:	2078	**BIC:**	1.148e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-26.6032	1.146	-23.212	0.000	-28.851	-24.356
x1	0.3630	0.020	18.211	0.000	0.324	0.402
x2	1.825e-12	7.34e-14	24.864	0.000	1.68e-12	1.97e-12
x3	2.336e-07	1.82e-08	12.850	0.000	1.98e-07	2.69e-07

Omnibus:	2389.025	**Durbin-Watson:**	2.036
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	589761.028
Skew:	5.460	**Prob(JB):**	0.00
Kurtosis:	84.726	**Cond. No.**	2.24e+14

In [100]:
```
X_test_mod_arr=np.array(X_test[['Prev_diff','Prev_GDP','Volume']])
```

In [101]:
```
y_test_mod_arr=np.array(y_test)
```

Cross Validation with Linear Regression and Polynomial Regression for 3 features

In [102]:
```
def CrossValidation(X_train,y_train):
```

```
kf = KFold(n_splits=3, shuffle=True, random_state = 20)
linear_score= [] #collect the validation results for both models
polynomial_2=[]
polynomial_3=[]
polynomial_4=[]

i=0
for train_ind, val_ind in kf.split(X_train,y_train):
    i+=1

    X_train_split, y_train_split = X_train[train_ind], y_train[train_
    X_val_split, y_val_split = X_train[val_ind], y_train[val_ind]

    ## Linear Regression
    linear = LinearRegression()
    linear.fit(X_train_split,y_train_split)
    linear_predict_mod = linear.predict(X_val_split)
    #print('Linear Regression Score for Validation Set {} :{}'.format
    linear_score.append(linear.score(X_val_split,y_val_split))

    ## Plot the residuals
    #plt.subplot(1, 2, 1)
    residual_mod = y_val_split - linear_predict_mod
    create_plots(residual_mod,linear_predict_mod,'for Set {} Linear R
    #create_YX_plot(y_val_split,X_val_split,linear.coef_)


    for degree in range(2,5):
        ## Polynomial Regression
        poly=PolynomialFeatures(degree,interaction_only=True)
        X_fit=poly.fit_transform(X_train_split)
        model=linear_model.LinearRegression(fit_intercept=True)
        model.fit(X_fit,y_train_split)
        poly_predict_mod=model.predict(poly.fit_transform(X_val_split
        #print(poly_predict.)
        #print('Polynomial degree {} score for set {} :'.format(degre
        #print(model.score(poly.fit_transform(X_train),y_train))
        #print(model.score(poly.fit_transform(X_val),y_val))
        if degree ==2:
            polynomial_2.append(model.score(poly.fit_transform(X_val_
        elif degree==3:
            polynomial_3.append(model.score(poly.fit_transform(X_val_
        else:
            polynomial_4.append(model.score(poly.fit_transform(X_val_

        ## Plot the residual Plots
        residual_poly_mod = y_val_split-poly_predict_mod
        create_plots(residual_poly_mod,poly_predict_mod,'For Set {} ,
        #plt.subplot(2,2,degree-1)
        #sns.regplot(poly_predict,residual)
```

```
    print('Mean Linear Regression Score :{}'.format(np.mean(linear_score)
    print('Polynomial Regression Score for degree 2 : {}'.format(np.mean(
    print('Polynomial Regression Score for degree 3 : {}'.format(np.mean(
    print('Polynomial Regression Score for degree 4 : {}'.format(np.mean(
```

In [103]:
```
len(y_train_mod_arr)
```

Out[103]: 2082

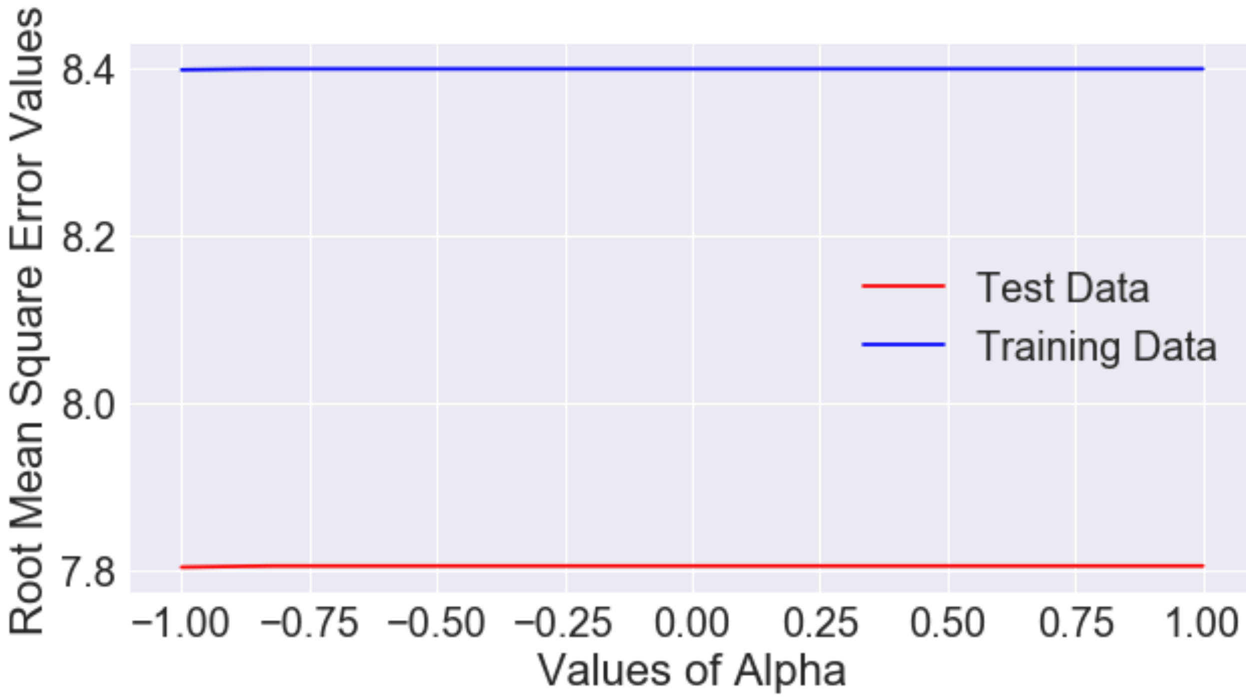
Trying out different alpha for polynomial regression of degree 2

In [132]:
```
def polynomial_features(X_test_mod_arr,y_test_mod_arr,X_train_mod_arr,y_t
    alphalist = 10**(np.linspace(-1,1,1000))
    err_vec_test = np.zeros(len(alphalist))
    err_vec_train = np.zeros(len(alphalist))

    for i,curr_alpha in enumerate(alphalist):

        est = make_pipeline(PolynomialFeatures(2,interaction_only=True),L
        est.fit(X_train_mod_arr, y_train_mod_arr)

        y_predict = est.predict(X_test_mod_arr)
        y_predict_score=est.score(X_test_mod_arr,y_test_mod_arr)
        err_vec_test[i] = np.sqrt(np.mean((y_predict_score - y_test_mod_a

        y_train_predict = est.predict(X_train_mod_arr)
        y_train_predict_score = est.score(X_train_mod_arr,y_train_mod_arr
        err_vec_train[i] = np.sqrt(np.mean((y_train_predict_score - y_tra

    ## Plotting Residual Plots

    #create_plots((y_predict - y_test_mod_arr),y_predict,'Test Data Resid
    #
    #create_plots((y_train_predict - y_train_mod_arr),y_train_predict,'Tr

    ## Plotting Alpha vs Error
    plt.figure(figsize=(10,5))
    plt.plot(np.log10(alphalist),err_vec_test,c='r')
    plt.plot(np.log10(alphalist),err_vec_train,c='b')
    plt.legend(['Test Data','Training Data'],loc=5)
    plt.xlabel('Values of Alpha')
    plt.ylabel('Root Mean Square Error Values')

    ## Printing Mean of errors
    print('Mean of Error for Test Data :{}'.format(np.mean(err_vec_test))
    print('Mean of Error for Training Data:{}'.format(np.mean(err_vec_tra
    print('Alpha:',alphalist[err_vec_test.argmin()])
    print('Alpha :',alphalist[err_vec_train.argmin()])
    plt.show()
```

In [ ]:

In [133]:
```
polynomial_features(X_test_mod_arr,y_test_mod_arr,X_train_mod_arr,y_train
```

```
Mean of Error for Test Data :7.805509267545955
Mean of Error for Training Data:8.398808077540318
Alpha: 0.1
Alpha : 0.1
```



In [117]:
```
est = make_pipeline(PolynomialFeatures(2,interaction_only=True),Lasso(alp
est.fit(X_train_mod_arr, y_train_mod_arr)

y_predict = est.predict(X_test_mod_arr)
y_predict_score=est.score(X_test_mod_arr,y_test_mod_arr)
print('RMSE for Test Data: {}' .format(np.sqrt(np.mean((y_predict_score -
print('MAPE for Test Data: {}%' .format(np.mean(abs(y_predict_score - y_t


y_train_predict = est.predict(X_train_mod_arr)
y_train_predict_score = est.score(X_train_mod_arr,y_train_mod_arr)
print('RMSE for Training Data: {}'.format(np.mean((y_train_predict_score
print('MAPE for Training Data: {}%'.format(np.mean(abs(y_train_predict_sc
```

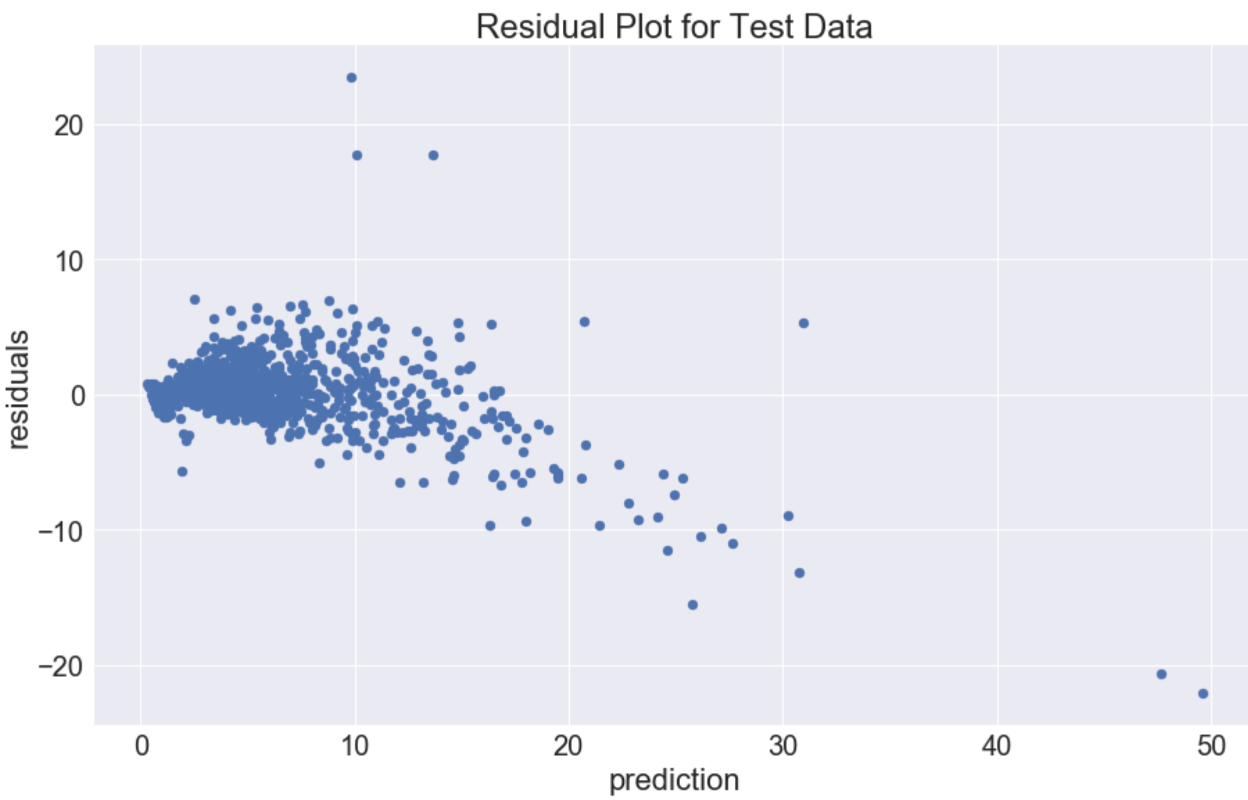
```
RMSE for Test Data: 7.805578627545259
MAPE for Test Data: 79.74023213312142%
RMSE for Training Data: 70.54074572199728
MAPE for Training Data: 78.68526490738141%
```

In [130]:
```
plt.scatter(y_train_mod_arr,y_train_predict-y_train_mod_arr)#(y_train_pre
plt.title('Residual Plot for Training Data')
plt.xlabel("prediction")
plt.ylabel("residuals")
```

Out[130]: Text(0,0.5,'residuals')

In [129]:
```
plt.scatter(y_test_mod_arr,y_predict-y_test_mod_arr)
plt.title('Residual Plot for Test Data')
plt.xlabel("prediction")
plt.ylabel("residuals")
```

Out[129]: Text(0,0.5,'residuals')



In [ ]: