Walmart Stores Sales Forcasting

One challenge of modeling retail data is the need to make decisions based on limited history. If Christmas comes but once a year, so does the chance to see how strategic decisions impacted the bottom line.

In this use case, provided with historical sales data for 45 Walmart stores located in different regions. Each store contains many departments, and participants must project the sales for each department in each store.

In addition, Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of which are the Super Bowl, Labor Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. Part of the challenge presented by this competition is modeling the effects of markdowns on these holiday weeks in the absence of complete/ideal historical data.

ML Problem Formulation

Time-series forecasting and Regression

- *To forcast sales for 45 stores, given sales data for 45 Walmart stores located in different regions and time.*

To solve the above we would be using data collected in Jan - Mar 2015 to predict the pickups in Jan - Mar 2016.

2010-02-05 to 2012-11-01

Performance Metric

This competition is evaluated on the weighted mean absolute error (WMAE):

WMAE=1∑wi∑i=1nwi|yi−y^i| where

n is the number of rows y^i is the predicted sales yi is the actual sales wi are weights. w = 5 if the week is a holiday week, 1 otherwise

Data

stores.csv

This file contains anonymized information about the 45 stores, indicating the type and size of store.

train.csv

This is the historical training data, which covers to 2010-02-05 to 2012-11-01. Within this file you will find the following fields:

Store - the store number Dept - the department number Date - the week Weekly_Sales - sales for the given department in the given store IsHoliday - whether the week is a special holiday week test.csv

This file is identical to train.csv, except we have withheld the weekly sales. You must predict the sales for each triplet of store, department, and date in this file.

features.csv

This file contains additional data related to the store, department, and regional activity for the given dates. It contains the following fields:

Store - the store number Date - the week Temperature - average temperature in the region Fuel_Price - cost of fuel in the region MarkDown1-5 - anonymized data related to promotional markdowns that Walmart is running. MarkDown data is only available after Nov 2011, and is not available for all stores all the time. Any missing value is marked with an NA. CPI - the consumer price index Unemployment - the unemployment rate IsHoliday - whether the week is a special holiday week For convenience, the four holidays fall within the following weeks in the dataset (not all holidays are in the data):

Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13 Labor Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13 Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13 Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
```

In [2]:
```
train_data = pd.read_csv('train.csv')
train_data.head()
```

Out[2]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

In [3]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
Store           421570 non-null int64
Dept            421570 non-null int64
Date            421570 non-null object
Weekly_Sales    421570 non-null float64
IsHoliday       421570 non-null bool
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
```

In [4]:
```
store_data = pd.read_csv('stores.csv')
store_data.head()
```

Out[4]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

In [5]:
```
features_data = pd.read_csv('features.csv')
features_data.head()
```

Out[5]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkI
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	

In [6]:
```
dataset = train_data.merge(features_data, how='left').merge(store_data, how='left'
dataset.head()
```

Out[6]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	NaN
1	1	1	2010-02-12	46039.49	True	38.51	2.548	NaN	NaN
2	1	1	2010-02-19	41595.55	False	39.93	2.514	NaN	NaN
3	1	1	2010-02-26	19403.54	False	46.63	2.561	NaN	NaN
4	1	1	2010-03-05	21827.90	False	46.50	2.625	NaN	NaN

In [7]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 0 to 421569
Data columns (total 16 columns):
Store           421570 non-null int64
Dept            421570 non-null int64
Date            421570 non-null object
Weekly_Sales    421570 non-null float64
IsHoliday       421570 non-null bool
Temperature     421570 non-null float64
Fuel_Price      421570 non-null float64
MarkDown1       150681 non-null float64
MarkDown2       111248 non-null float64
MarkDown3       137091 non-null float64
MarkDown4       134967 non-null float64
MarkDown5       151432 non-null float64
CPI             421570 non-null float64
Unemployment    421570 non-null float64
Type            421570 non-null object
Size            421570 non-null int64
dtypes: bool(1), float64(10), int64(3), object(2)
memory usage: 51.9+ MB
```

Data Cleaning/ Preprocessing

In [8]:
```
# Handling missing values

dataset['MarkDown1'].fillna(0, inplace=True)
dataset['MarkDown2'].fillna(0, inplace=True)
dataset['MarkDown3'].fillna(0, inplace=True)
dataset['MarkDown4'].fillna(0, inplace=True)
dataset['MarkDown5'].fillna(0, inplace=True)
```

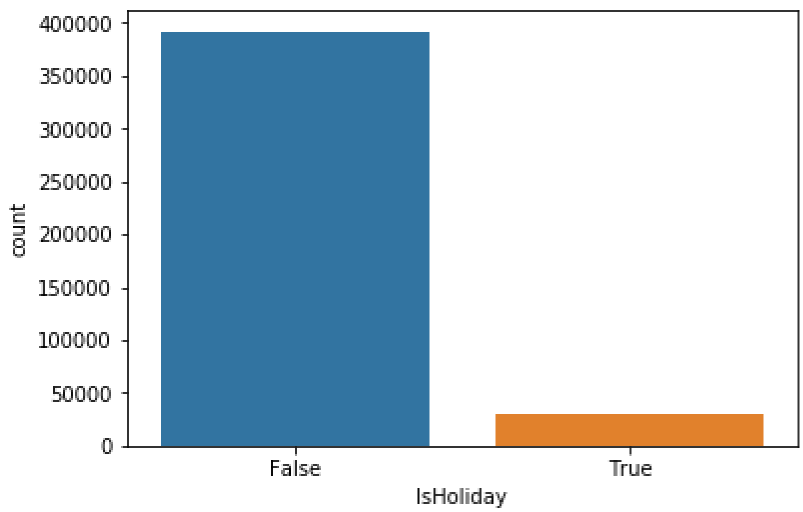
Exploratory Data Analysis

In [9]:
```
# There are 1285 records with negative Weekly_Sales which means refund.

dataset['Weekly_Sales'][dataset['Weekly_Sales']<0].count()

# Kept negative values as is, as its giving bad result after replacing those
```

Out[9]: 1285

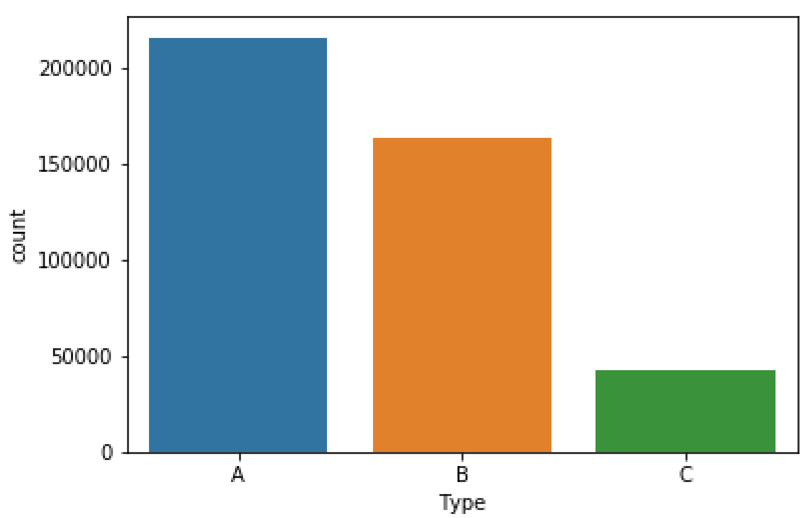
In [10]:
```
# We have very few records with Holidays

sns.countplot(dataset['IsHoliday'])
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x5683924da0>



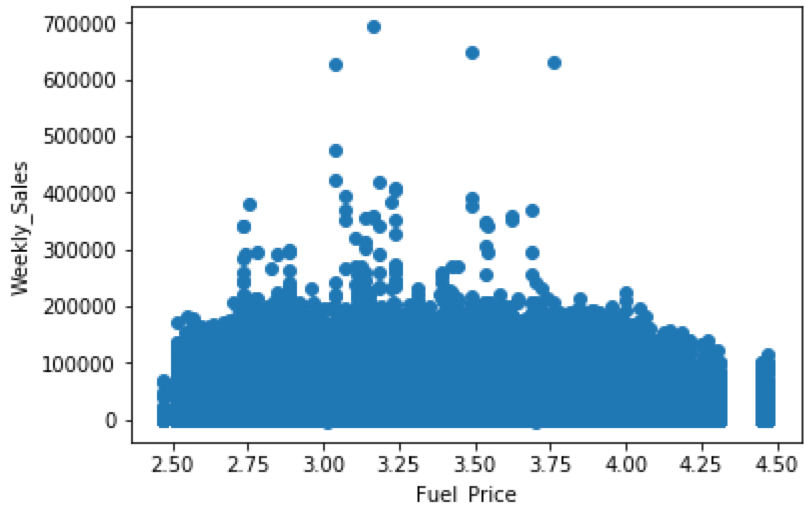
In [11]:
```
# In data most of the store are of Type 'A' and very few of type 'C'

sns.countplot(dataset['Type'])
```

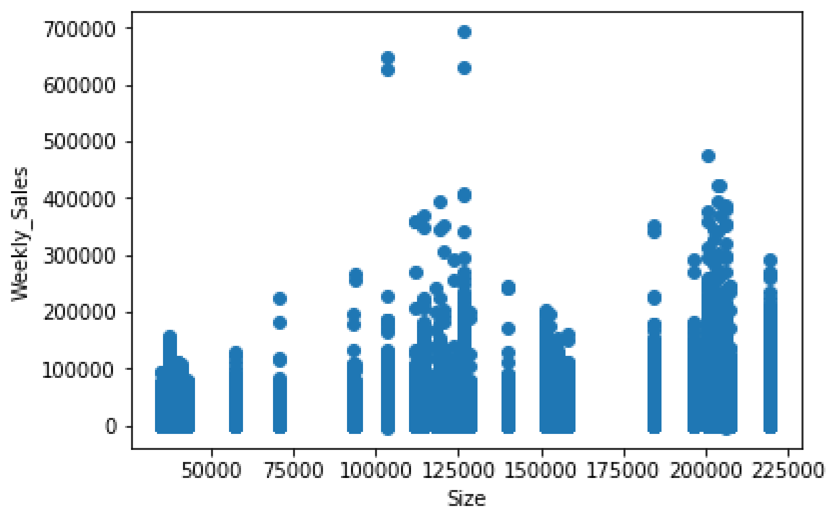
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x56838ab2b0>

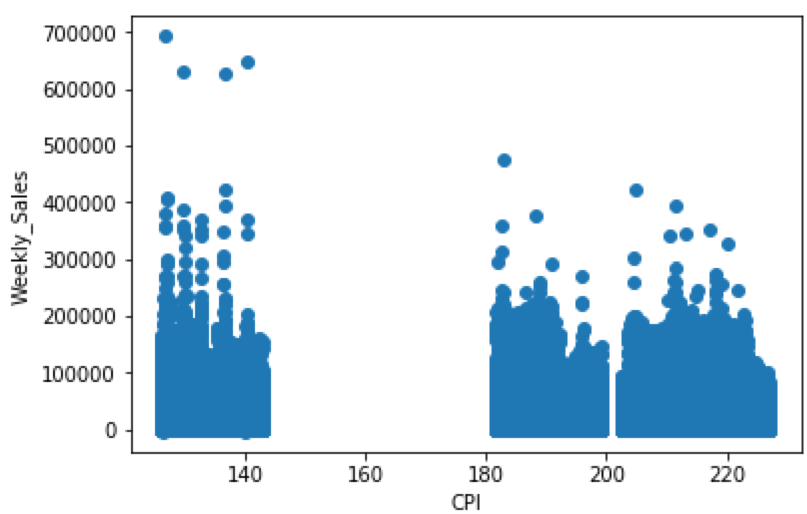


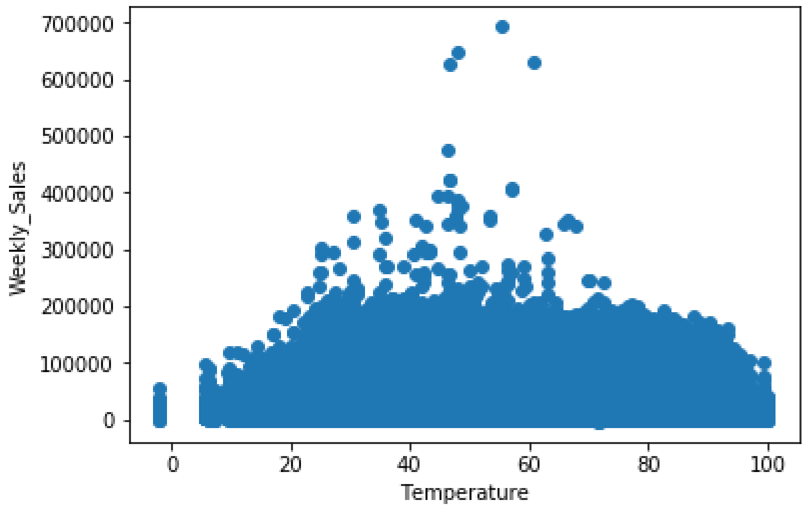
In [12]:
```
# Function to plot scatter plot betten selected feature and Weekly_Sales
def scatter(dataset, column):
    plt.figure()
    plt.scatter(dataset[column], dataset['Weekly_Sales'])
    plt.ylabel('Weekly_Sales')
    plt.xlabel(column)
```

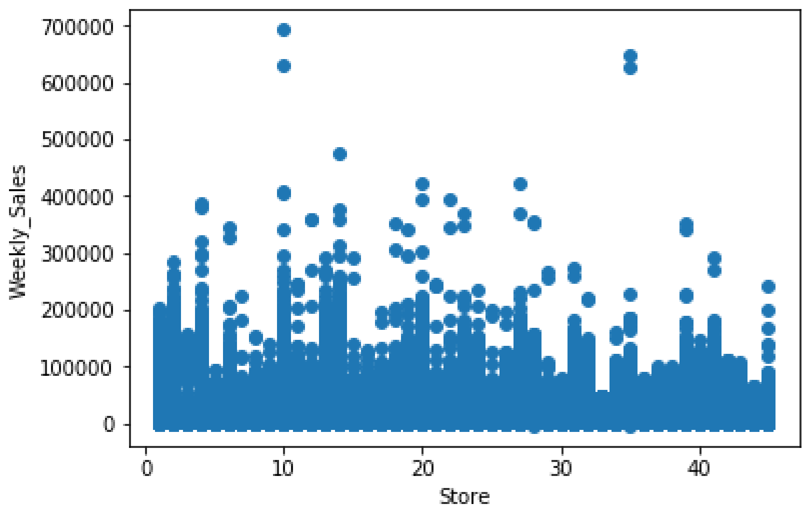
In [13]:
```
scatter(dataset, 'Fuel_Price')
scatter(dataset, 'Size')
scatter(dataset, 'CPI')
scatter(dataset, 'Temperature')
scatter(dataset, 'Store')
scatter(dataset, 'Dept')
```

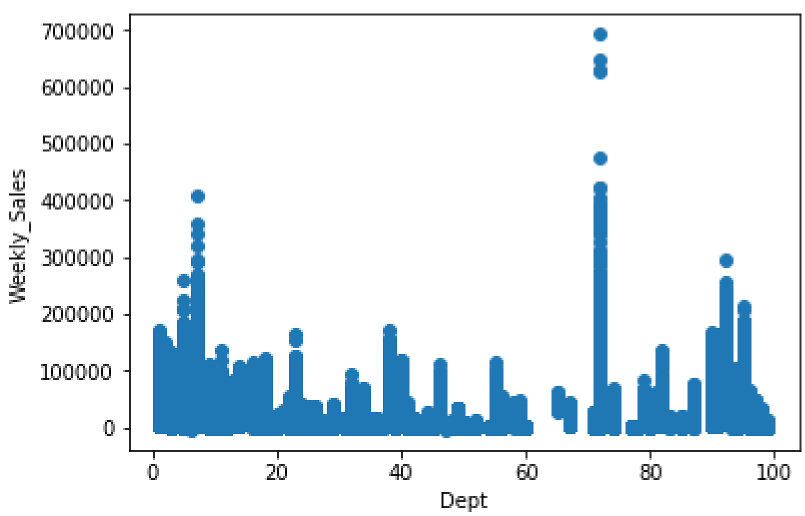








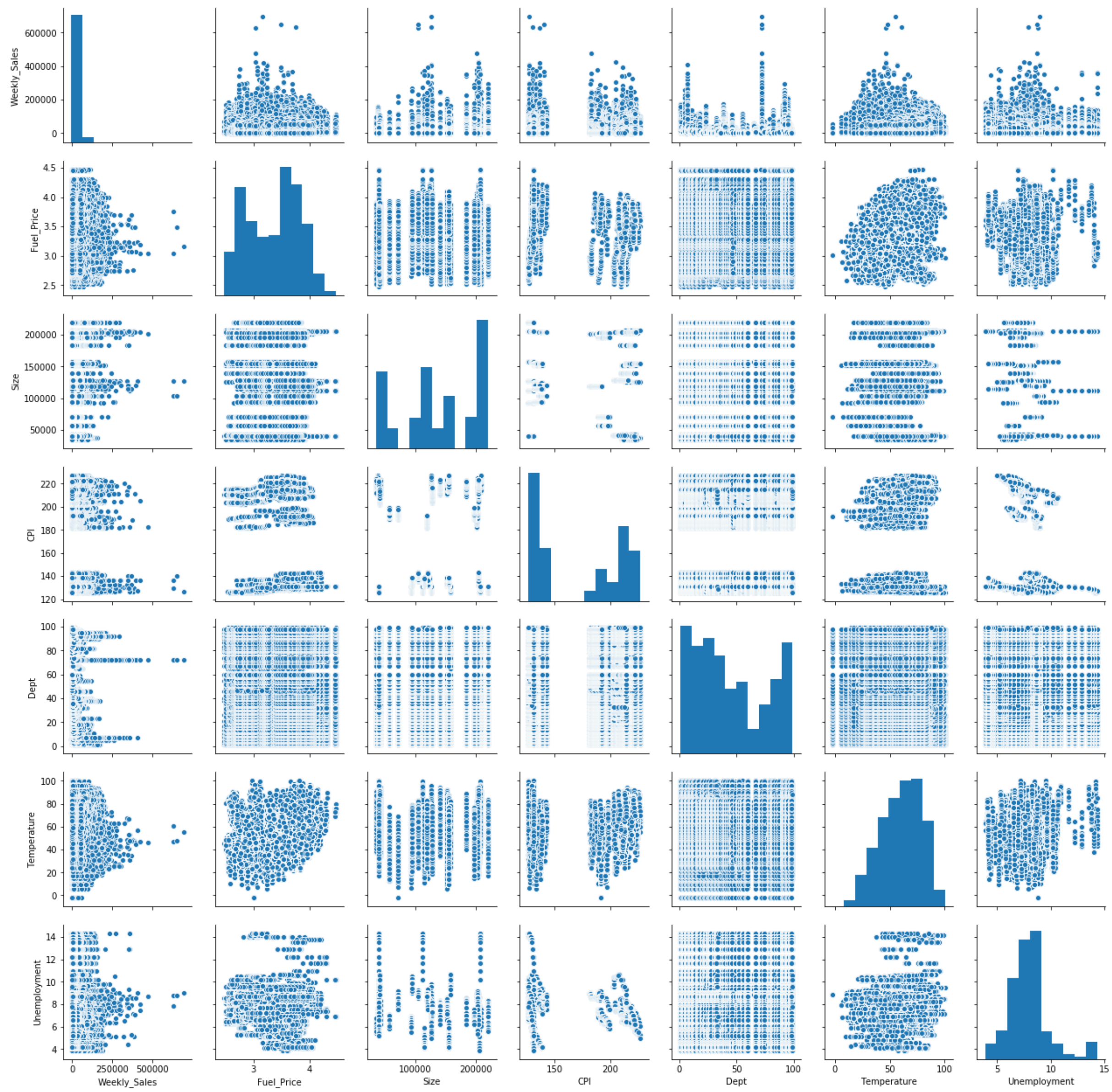




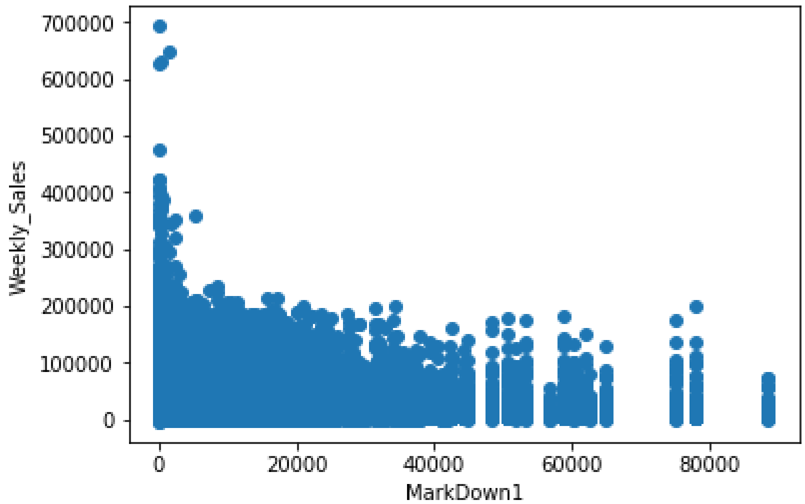
- Fuel_Price, Temperature and Size feature seems to be interesting

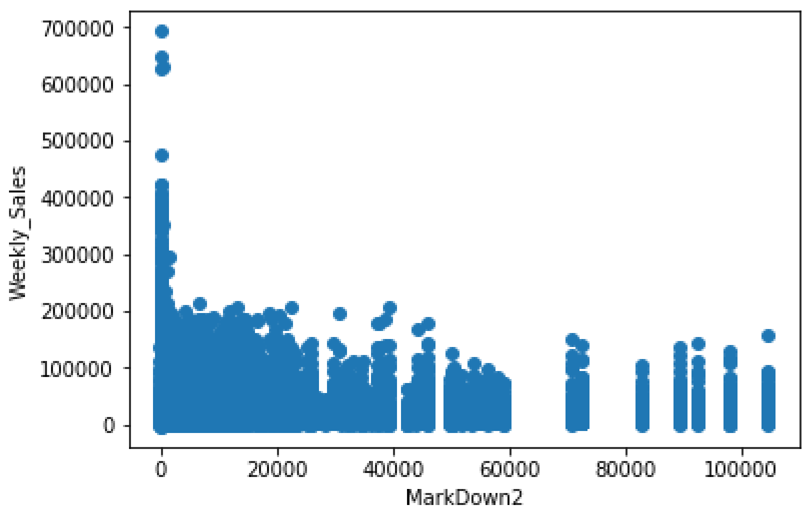
In [14]:
```
sns.pairplot(dataset, vars=['Weekly_Sales', 'Fuel_Price', 'Size', 'CPI', 'Dept', '
```

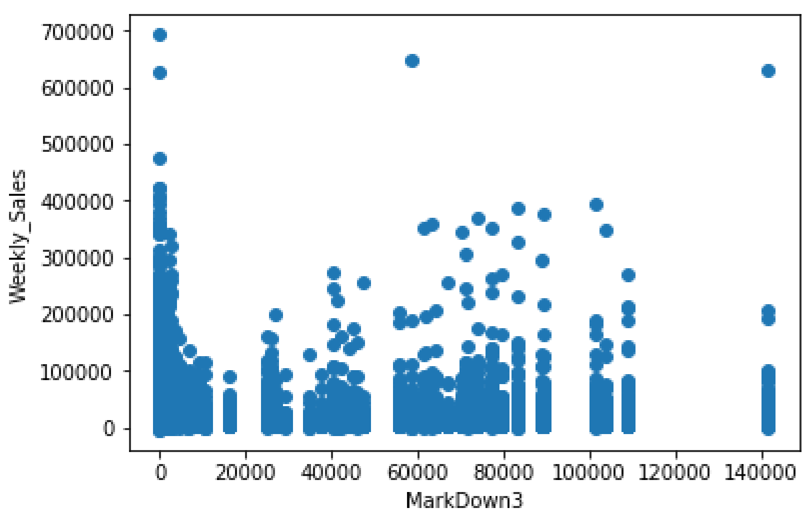
Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x5686389550>
```

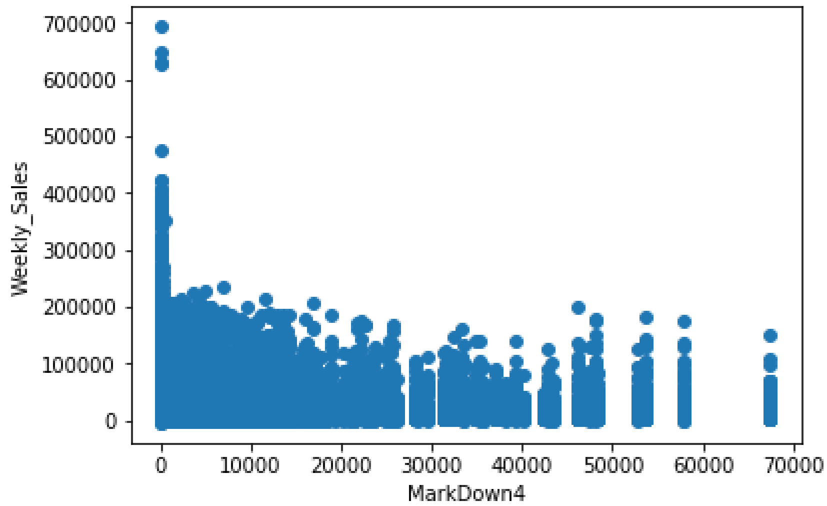


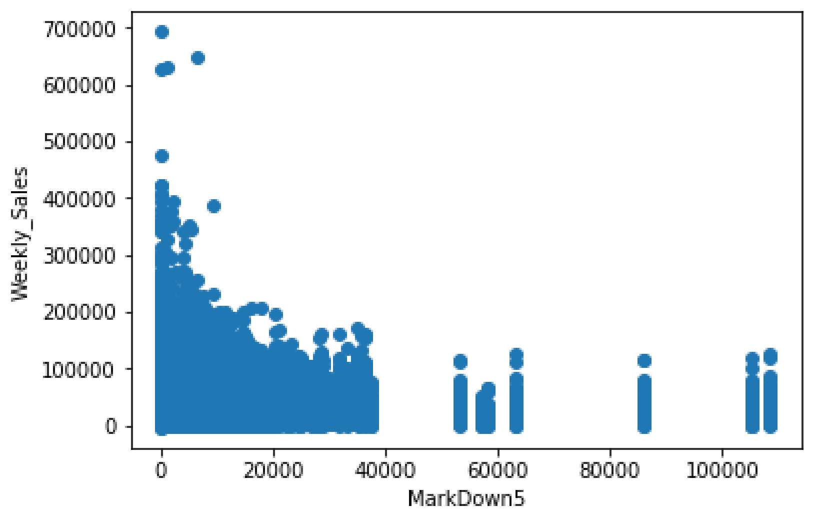
In [15]:
```
scatter(dataset, 'MarkDown1')
scatter(dataset, 'MarkDown2')
scatter(dataset, 'MarkDown3')
scatter(dataset, 'MarkDown4')
scatter(dataset, 'MarkDown5')
```











- MarkDown3 seems interesting, others Markdown doesn't seems to be important feature

Creating new features

In [16]:
```
# Formating the date
dataset['Date'] =  pd.to_datetime(dataset['Date'], format='%Y-%m-%d')

# Creating new feature from date i.e. Year and week
dataset['Year'] = dataset['Date'].dt.year
dataset['Week'] = dataset['Date'].dt.week

# dataset['Month'] = dataset['Date'].dt.month

# Map values of IsHoliday with 0 and 5 as per weights given to them
dataset['IsHoliday'] = dataset['IsHoliday'].map({False:0, True:5})
```

In [17]:

```
# Droping not useful features

dataset = dataset.drop(['Date', 'MarkDown1','MarkDown2','MarkDown4','MarkDown5', '
                        'Fuel_Price'], axis=1)
dataset.head()
```

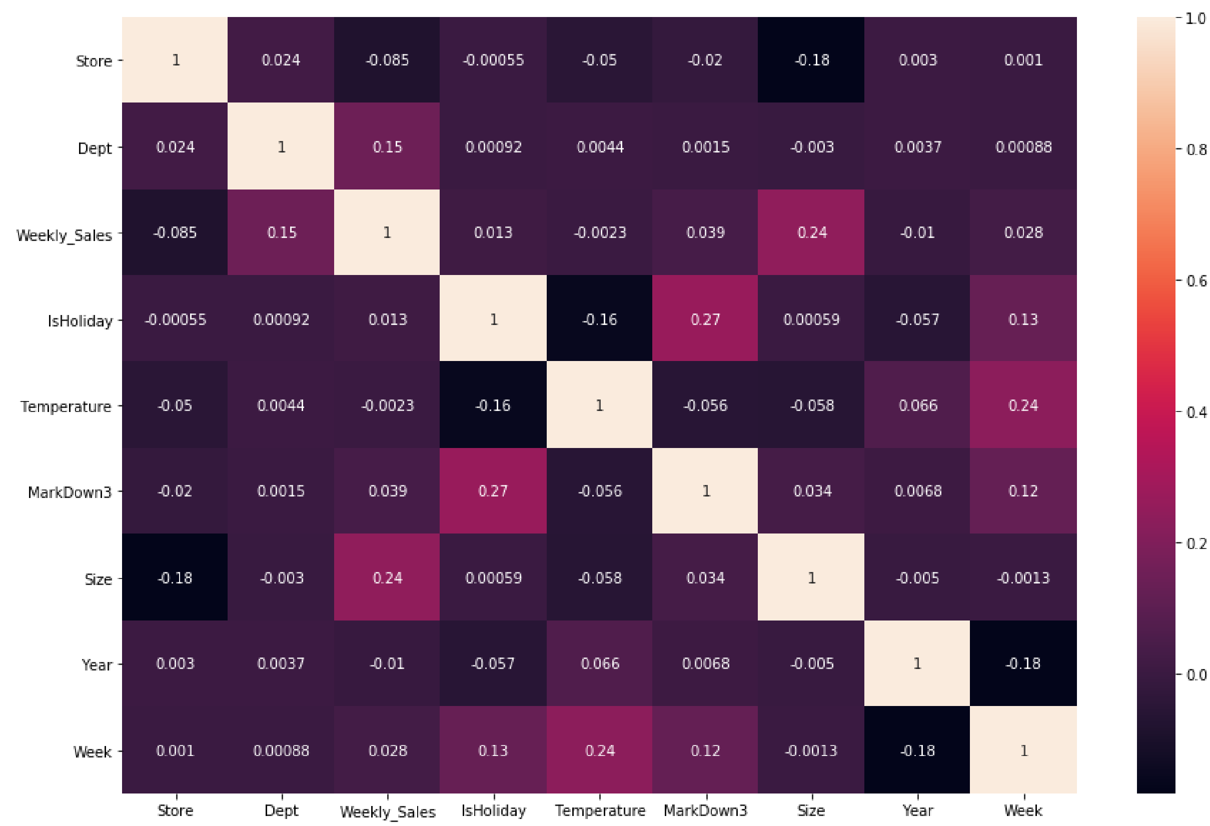
Out[17]:

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	MarkDown3	Size	Year	Week
0	1	1	24924.50	0	42.31	0.0	151315	2010	5
1	1	1	46039.49	5	38.51	0.0	151315	2010	6
2	1	1	41595.55	0	39.93	0.0	151315	2010	7
3	1	1	19403.54	0	46.63	0.0	151315	2010	8
4	1	1	21827.90	0	46.50	0.0	151315	2010	9

In [20]:

```
# Plotting correlation between all important features
corr = dataset.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr, annot=True)
plt.plot()
```

Out[20]: []



Split of data for train and test

In [21]:
```
# Used 2010 and 2011 data for training
# Used 2012 data for test

train_x = dataset.loc[dataset['Year']<2012]
train_y = dataset['Weekly_Sales'].loc[dataset['Year']<2012]
train_x = train_x.drop('Weekly_Sales', axis=1)

print("Records in training data :", train_x.shape)
print("Records in training target label :", train_y.shape)

test_x = dataset.loc[dataset['Year']>2011]
test_y = dataset['Weekly_Sales'].loc[dataset['Year']>2011]
test_x = test_x.drop('Weekly_Sales', axis=1)

print("Records in test data :", test_x.shape)
print("Records in test target label :", test_y.shape)
```

```
Records in training data : (294132, 8)
Records in training target label : (294132,)
Records in test data : (127438, 8)
Records in test target label : (127438,)
```

Modeling

In [22]:
```
def calculate_error(test_y, predicted, weights):
    return mean_absolute_error(test_y, predicted, sample_weight=weights)
```

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import RandomizedSearchCV
```

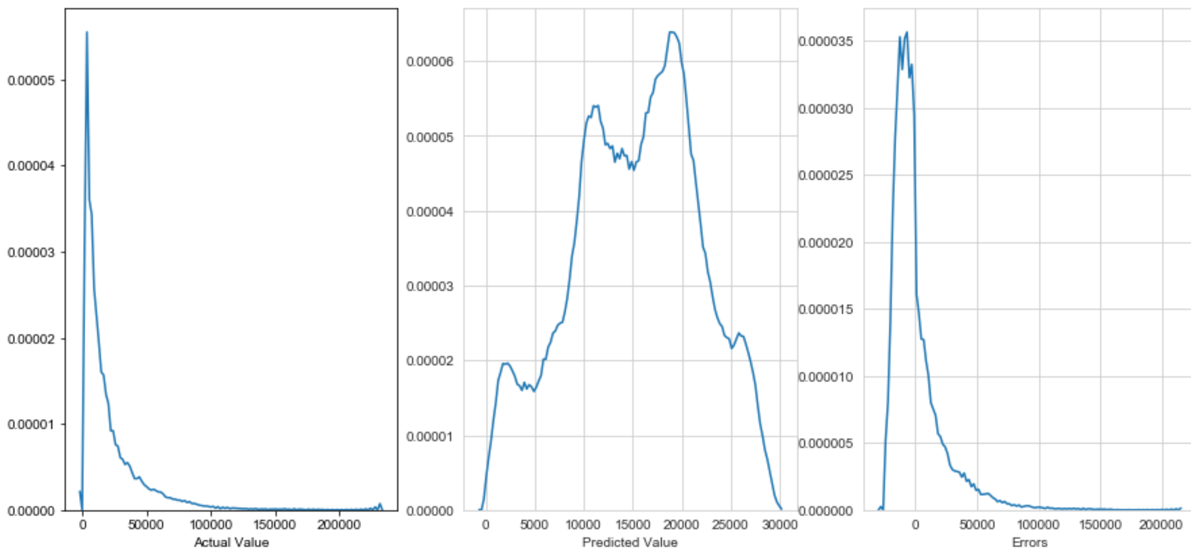
Linear Regression

In [24]:
```
def plots(test_y, pred_y):
    plt.figure(figsize=(15,15))

    plt.subplot(231)
    sns.set_style('whitegrid')
    sns.kdeplot(np.array(test_y), bw=0.5)
    plt.xlabel('Actual Value')

    plt.subplot(232)
    sns.kdeplot(np.array(pred_y), bw=0.5)
    plt.xlabel('Predicted Value')

    plt.subplot(233)
    delta_y = test_y - pred_y;
    sns.kdeplot(np.array(delta_y), bw=0.5)
    plt.xlabel('Errors')
    plt.show()
```

In [25]:
```
model = LinearRegression()
model.fit(train_x, train_y)

pred_y = model.predict(test_x)

weights = test_x['IsHoliday']
LR_error = calculate_error(test_y, pred_y, weights)
print("Error: ", LR_error)
```

```
Error:  14744.732702185725
```

In [26]:
```
# Plot between Actual Value vs Predicted Value
plots(test_y, pred_y)
```



- Actual values and predicted values are very different. Hence error is very high

RandomForestRegressor

In [27]:
```
from sklearn.ensemble import RandomForestRegressor
```

Hyperparamter tuning

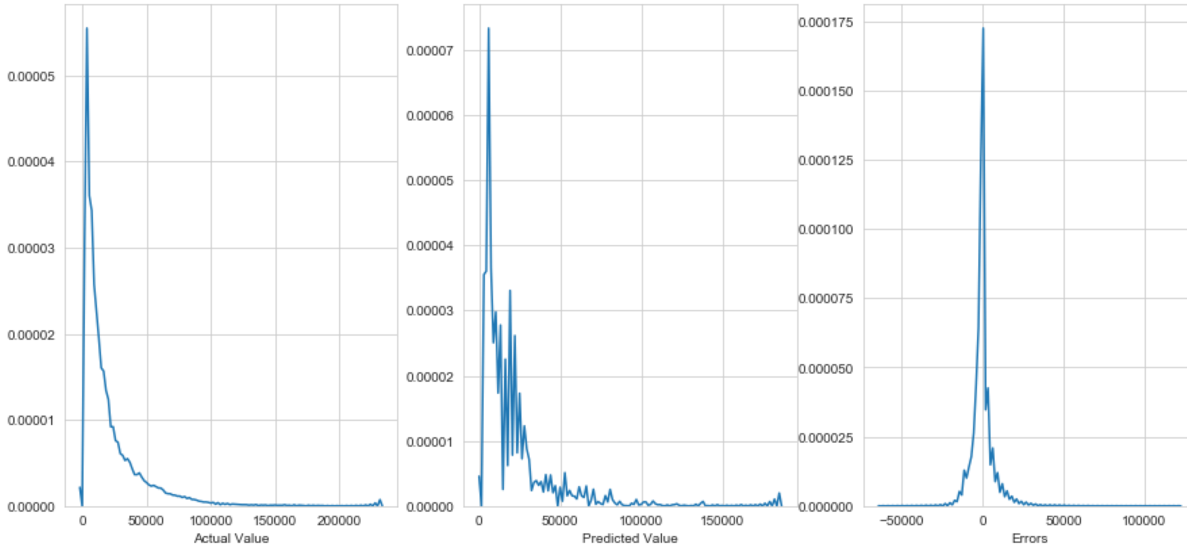
In [28]:
```
n_estimators = [50, 100, 150, 200]
max_depth=[5, 10, 15, 20]
min_samples_split = [2, 4 ,6, 8]
params = {'n_estimators' : n_estimators, 'max_depth': max_depth, 'min_samples_spli

randomForest = RandomForestRegressor()
randomCV = RandomizedSearchCV(randomForest, params , cv=5)
randomCV.fit(train_x, train_y)

optimal_n_estimators = randomCV.best_params_['n_estimators']
optimal_max_depth = randomCV.best_params_['max_depth']
optimal_min_samples_split = randomCV.best_params_['min_samples_split']

print("Best parameters: ", randomCV.best_params_)
print("Best estimators: ", randomCV.best_estimator_)
```

```
Best parameters:  {'n_estimators': 200, 'min_samples_split': 8, 'max_depth': 1
0}
Best estimators:  RandomForestRegressor(bootstrap=True, criterion='mse', max_d
epth=10,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=8,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [29]:
```
rfModel = RandomForestRegressor(n_estimators=optimal_n_estimators, max_depth=optim
                                min_samples_split=optimal_min_samples_split)
rfModel.fit(train_x, train_y)

pred_y = rfModel.predict(test_x)

weights = test_x['IsHoliday']
RF_error = calculate_error(test_y, pred_y, weights)
print("Error: ", RF_error)
```

```
Error:  4377.498592401463
```

In [30]:
```
plots(test_y, pred_y)
```



- Actual values and predicted values are better than Linear Regression model and error is also less

ExtraTreesRegressor

Hyper parameter tuning

In [31]:
```
from sklearn.ensemble import ExtraTreesRegressor
```

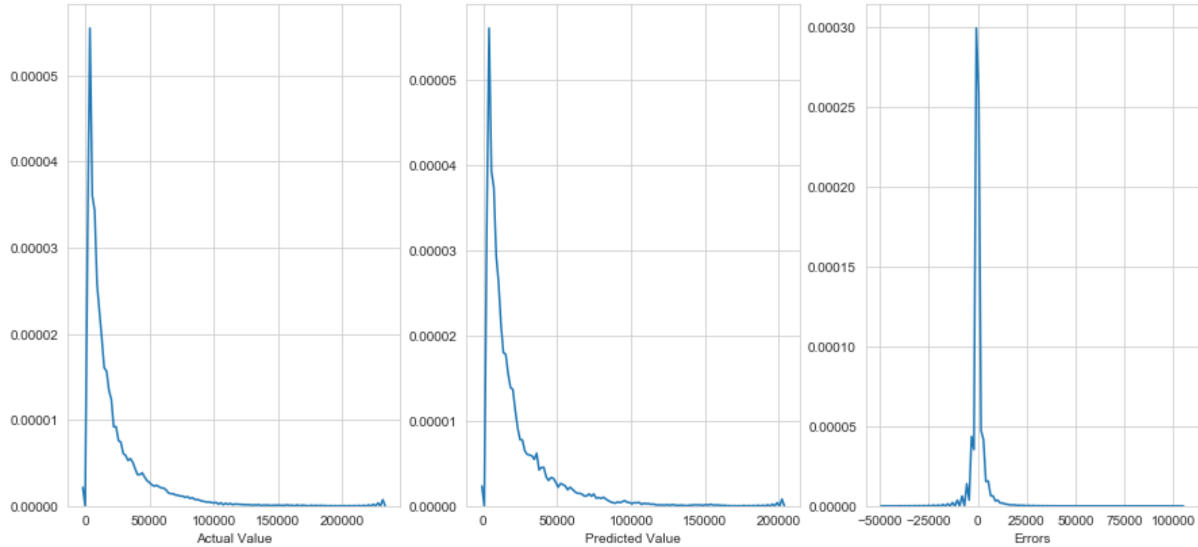
In [32]:
```
n_estimators = [50, 100, 150, 200]
max_depth=[5, 10, 15, 20]
min_samples_split = [2,4 ,8, 10]
params = {'n_estimators' : n_estimators, 'max_depth': max_depth, 'min_samples_spli

etree = ExtraTreesRegressor()
randomCV = RandomizedSearchCV(etree, params , cv=5)
randomCV.fit(train_x, train_y)

optimal_n_estimators = randomCV.best_params_['n_estimators']
optimal_max_depth = randomCV.best_params_['max_depth']
optimal_min_samples_split = randomCV.best_params_['min_samples_split']

print("Best parameters: ", randomCV.best_params_)
print("Best estimators: ", randomCV.best_estimator_)
```

```
Best parameters:  {'n_estimators': 150, 'min_samples_split': 8, 'max_depth': 2
0}
Best estimators:  ExtraTreesRegressor(bootstrap=False, criterion='mse', max_de
pth=20,
          max_features='auto', max_leaf_nodes=None,
          min_impurity_decrease=0.0, min_impurity_split=None,
          min_samples_leaf=1, min_samples_split=8,
          min_weight_fraction_leaf=0.0, n_estimators=150, n_jobs=None,
          oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [33]:
```
extraTree = ExtraTreesRegressor(n_estimators=optimal_n_estimators, max_depth = opt
                                min_samples_split = optimal_min_samples_split, n_j
extraTree.fit(train_x, train_y)

pred_y = extraTree.predict(test_x)

weights = test_x['IsHoliday']
ET_error = calculate_error(test_y, pred_y, weights)
print("Error: ", ET_error)
```

```
Error:  1981.3401446754774
```

In [34]:
```
plots(test_y, pred_y)
```



- Actaul value and predicted value seems very close now and results are much better using ExtraTrees Model

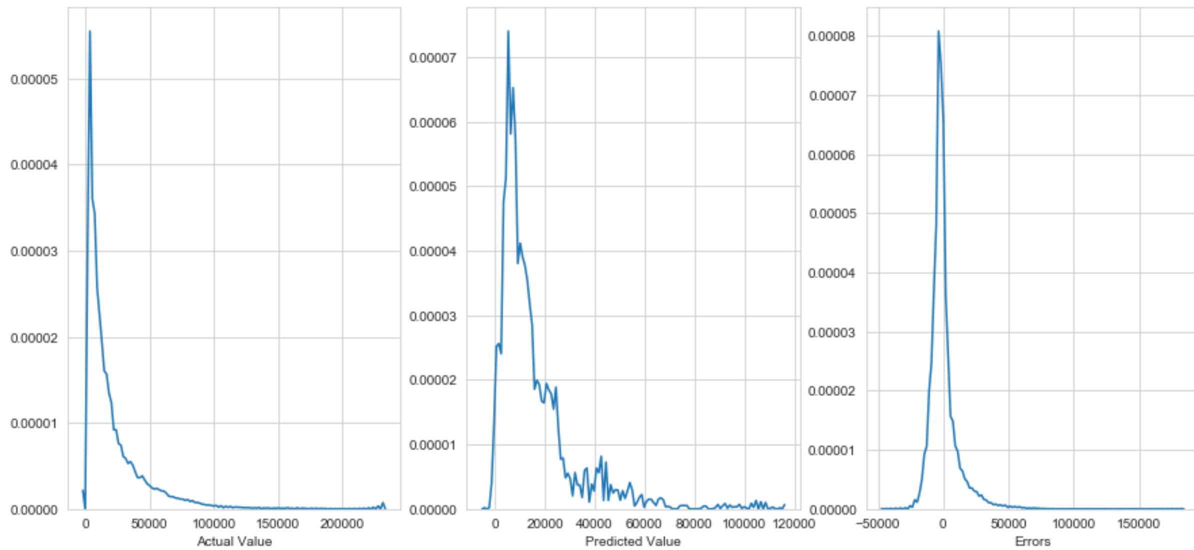
XGBRegressor

In [35]:
```
from xgboost import XGBRegressor
```

In [36]:
```
xgRegressor = XGBRegressor(learning_rate =0.1, n_estimators=100, max_depth=3, min_
                  gamma=0, subsample=0.8, reg_alpha=200, reg_lambda=200, colsample_byt

xgRegressor.fit(train_x, train_y)

pred_y = xgRegressor.predict(test_x)

weights = test_x['IsHoliday']
XG_error = calculate_error(test_y, pred_y, weights)
print("Error: ", XG_error)
```

```
Error:  7310.941025042182
```

In [37]:
```
plots(test_y, pred_y)
```



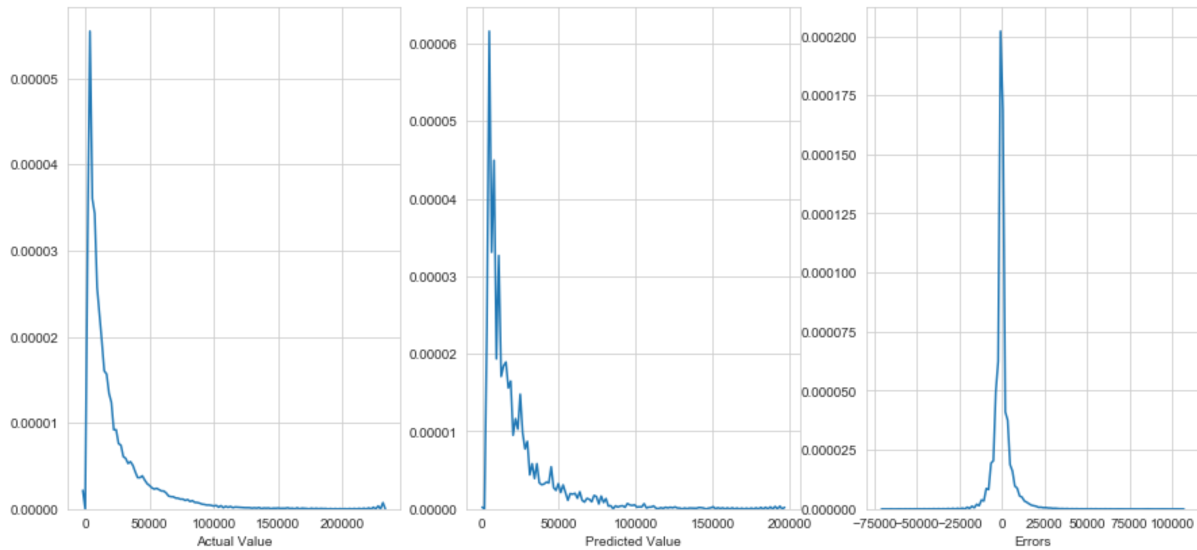
Stacking

In [38]:
```
from mlxtend.regressor import StackingRegressor
```

In [39]:
```
lrReg = LinearRegression()

rfReg = RandomForestRegressor(n_estimators=100, max_depth=10,
                                min_samples_split=2)
xgbReg = XGBRegressor(learning_rate =0.1, n_estimators=100, max_depth=3, min_child
              gamma=0, subsample=0.8, reg_alpha=200, reg_lambda=200, colsample_byt
xtraTreesReg = ExtraTreesRegressor(n_estimators=150, max_depth = 15, min_samples_s

stackRgr = StackingRegressor(regressors=[lrReg, rfReg, xgbReg], meta_regressor=xtr

# Training the stacking classifier
stackRgr.fit(train_x, train_y)
stack_pred_y = stackRgr.predict(test_x)


weights = test_x['IsHoliday']
Stack_error = calculate_error(test_y, stack_pred_y, weights)
print("Error: ", Stack_error)
```

```
Error:  3189.9629174081642
```

In [40]:
```
plots(test_y, stack_pred_y)
```



Test Data for submission

In [41]:
```
test_data = pd.read_csv('test.csv')
test_data.head()
```

Out[41]:

	Store	Dept	Date	IsHoliday
0	1	1	2012-11-02	False
1	1	1	2012-11-09	False
2	1	1	2012-11-16	False
3	1	1	2012-11-23	True
4	1	1	2012-11-30	False

In [42]:
```
# Merge test data with features data
test_data = test_data.merge(features_data, how='left').merge(store_data, how='left
```

In [43]:
```
# handle missing values
test_data['MarkDown3'] = test_data['MarkDown3'].fillna(0)
```

In [44]:
```
# Formating the date
test_data['Date'] =  pd.to_datetime(test_data['Date'], format='%Y-%m-%d')

# Creating new features year and week from date
test_data['Year'] = test_data['Date'].dt.year
test_data['Week'] = test_data['Date'].dt.week

# test_data['Month'] = test_data['Date'].dt.month

test_data['IsHoliday'] = test_data['IsHoliday'].map({False:0, True:5})
```

In [45]:
```
# Droping not important features
final_testdata = test_data.drop(['Date', 'MarkDown1','MarkDown2','MarkDown4','Mark
                                 'CPI', 'Fuel_Price'], axis=1)
final_testdata.head()
```

Out[45]:

	Store	Dept	IsHoliday	Temperature	MarkDown3	Size	Year	Week
0	1	1	0	55.32	50.82	151315	2012	44
1	1	1	0	61.24	40.28	151315	2012	45
2	1	1	0	52.92	103.78	151315	2012	46
3	1	1	5	56.23	74910.32	151315	2012	47
4	1	1	0	52.34	3838.35	151315	2012	48

In [46]:
```
# Using ExtraTress Model to predit result for test.csv
pred_test_y = extraTree.predict(final_testdata)

final_testdata['Weekly_Sales'] = pred_test_y
final_testdata['Id'] = test_data['Store'].astype(str) + '_'+ test_data['Dept'].ast

test_result = final_testdata[['Id', 'Weekly_Sales']]
test_result.to_csv('Test_results.csv', index=False)
```

In [47]:
```
from prettytable import PrettyTable

table = PrettyTable()
table.add_column('S.No.', [1, 2, 3, 4, 5])
table.add_column('Model', ['Linear Regression','Random Forest Regression','Extra T
                           'XgBoost Regression', 'Stacking'])
table.add_column('Train Error', [LR_error, RF_error, ET_error, XG_error, Stack_err
print(table)
```

```
+-------+--------------------------+--------------------+
| S.No. |          Model           |    Train Error     |
+-------+--------------------------+--------------------+
|   1   |    Linear Regression     | 14744.732702185725 |
|   2   | Random Forest Regression | 4377.498592401463  |
|   3   |  Extra Trees Regression  | 1981.3401446754774 |
|   4   |    XgBoost Regression    | 7310.941025042182  |
|   5   |         Stacking         | 3189.9629174081642 |
+-------+--------------------------+--------------------+
```

Conclusion

Steps Followed:

- Data Cleaned - Replaced NaN values
- Performed EDA and found out important features
- Created new featues from Date like year and week
- Created various models using LinearRegression, RandomForestRegressor, ExtraTressRegressor and XGBoostRegressor

- Also created model using stacking
- Plotted Actual values VS predicted values

Among 5 models, ExtraTressRegressor gives the best result