#Importing libraries

In [239]:

```
#### importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import numpy as np
from datetime import datetime
pd.set_option('max_rows',80000,'max_columns',200)
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
import xgboost as xgb
import pickle as pkl
import os
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score,mean_absolute_percentage_error
#### importing libraries
import os
import statsmodels.api as sm
#!pip install statsmodels==0.11.1
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
#from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

#Loading dataset

In [4]:

```
#### Dataset
trte_featured = pd.read_csv("trte_featured.csv", parse_dates=[2])
```

#Train,CV and Test split -> Ratio (60:20:20)

In [33]:

```
### Let's do a 80:20 split for train as (train and test)
x_train =  trte_featured.loc[(trte_featured.set_type == 'Train')]
x_train = x_train.drop(['set_type'],axis=1)

x_test =  trte_featured.loc[(trte_featured.set_type == 'Test')]
x_test =  x_test.drop(['set_type'],axis=1)

y_train = x_train['Weekly_Sales']
```

In [6]:

```
## Fill na's with -1
x_train = x_train.fillna(-1)
x_test = x_test.fillna(-1)
```

In [7]:

```
x_train.dtypes
```

Out[7]:

```
Store                      int64
Dept                       int64
Date              datetime64[ns]
Weekly_Sales             float64
IsHoliday                  int64
Temperature              float64
MarkDown1                float64
MarkDown2                float64
MarkDown3                float64
MarkDown5                float64
CPI                      float64
Unemployment             float64
Type                       int64
Size                       int64
week                       int64
year                       int64
month                      int64
day                        int64
major_holiday            float64
rolling_mean             float64
expanding_mean           float64
EWM_0.1                  float64
EWM_0.4                  float64
EWM_0.5                  float64
EWM_0.7                  float64
lag_52                   float64
holt_avg                 float64
dtype: object
```

In [34]:

```
## creating our validation set
udates = x_train.Date.unique()
len_ = int(80*len(udates) /100)
tr_dates =  sorted(udates)[:len_]
te_dates = sorted(udates)[len_:]

X_train = x_train.loc[(x_train.Date.isin(tr_dates))]
Y_train = X_train['Weekly_Sales']
X_cv = x_train.loc[(x_train.Date.isin(te_dates))]
Y_cv = X_cv['Weekly_Sales']
```

In [9]:

```
## Reset indexes to avoid to avoid bad merges between dataframe
##### x_train -  train data which will be used to predict test data
#### X_train - x_train data which is divided into further train and cv in (80:20) and be used to predict cv for tuning
#### x_test -   test set
#### X_cv - cv set formed from x_train
#### y_train - target variable for x_train
### Y_train,Y_cv - taget variable for X_train,X_cv

x_train = x_train.reset_index(drop=True)
x_test = x_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

X_train = X_train.reset_index(drop=True)
X_cv = X_cv.reset_index(drop=True)
Y_train = Y_train.reset_index(drop=True)
Y_cv = Y_cv.reset_index(drop=True)
```

In [35]:

```
def cal_wmae(holiday,true,predict):
    '''Calculate weighted mean absolute error with weightage 5 for holiday week and 1 for non-holiday week'''
    err = []
    sum_ = 0
    for i in range(len(true)):
        if holiday[i]:
            err.append(abs(true[i] - predict[i])*5)
            sum_+=5
        else:
            err.append(abs(true[i] - predict[i]))
            sum_+=1

    return sum(err)/sum_

def calculate_r2_mae(true,predict):
    mae=mean_absolute_error(true,predict)
    r2=r2_score(true,predict)
    return r2,mae
```

7.Model Building

7.1 Baseline Models

In [11]:

```
### create a model which is taking mean or median of past values
class baseline_model:

        def fit_baseline(self,tr):
            'Calculates median sales values of all the store dept combo per week'
            global average_df
            average_df  = pd.DataFrame(columns=['Store','Dept','week','mean'])
            average_df = tr.groupby(['Store','Dept','week'],as_index=False).median()[['Store','Dept','week','Weekly_Sales']]
            average_df =  average_df.rename(columns={'Weekly_Sales':'mean'})
            average_df['mean'] = np.round(average_df['mean'],2) ### taking average of past year weeks sales
            print('Model is fit with training data')
            return average_df

        def predict_baseline(self,cv):
            ''' Predict sales values of given date based on the average sales values of the past on same date '''
            compute_ = pd.merge(cv[['Store','Dept','week']],average_df,how='inner',on=['Store','Dept','week'])
            predict_array = np.array(compute_.sort_values(by = ['Store','Dept','week'])['mean'])
            return predict_array

```

In [177]:

```
## predict train
bs = baseline_model()
bs.fit_baseline(x_train)
baseline_predict_tr = bs.predict_baseline(x_train)

## predict cv
bs = baseline_model()
bs.fit_baseline(X_train)
baseline_predict_cv = bs.predict_baseline(X_cv)

## predict test
bs = baseline_model()
bs.fit_baseline(x_train)
baseline_predict_te =  bs.predict_baseline(x_test)
```

```
Model is fit with training data
Model is fit with training data
Model is fit with training data
```

In [13]:

```
performance_cv,performance_train = dict(),dict()
## performance cv
mae,r2 = calculate_r2_mae(Y_cv,baseline_predict_cv)
wmae = cal_wmae(X_cv['IsHoliday'].values,Y_cv,baseline_predict_cv)
performance_cv['baseline_model'] = [mae,r2,wmae]
## performance train
mae,r2 = calculate_r2_mae(Y_train,baseline_predict_tr)
wmae = cal_wmae(X_train['IsHoliday'].values,Y_train,baseline_predict_tr)
performance_train['baseline_model'] = [mae,r2,wmae]

```

7.2 Random-Forest & XGBoost & ExtraTrees Regressors

7.2.1 step1: Determine hyperparameters for tree-models

In [53]:

```python
##Hyper parameter tuning random forest

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 500, num = 6)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [20,27,30]
# Minimum number of samples required to split a node
min_samples_split = [5, 12]
# Minimum number of samples required at each leaf node
min_samples_leaf = [2, 6]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
randomforest_params = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
# create xgboost params
xg_boost_params = {'n_estimators': n_estimators,
                   'max_depth' : max_depth,
                   'subsamples': [0.4,0.5,0.7],
                   'colsample_bytree':[0.4,0.5,0.7],
                   'learning_rate':[0.01, 0.1],
                   'min_child_weight': [5, 10,20],
                   'objective': ['reg:squarederror']}

## create extratrees params
extra_tree_params = {'n_estimators':n_estimators,
                    'max_features': max_features,
                    'min_samples_leaf':min_samples_leaf,
                    'min_samples_split': min_samples_leaf}
```

7.2.2: let's take the best selected features from our forward feature selection from EDA

In [54]:

```python
### our train dataset
#### let's take the best selected features from eda
### month and day are highly correlated to year  hence we remove one month
top_cols_basic_feats = ['Store','Dept','IsHoliday','major_holiday','Size','week','Type','day','year']
```

7.2.3 : Hyperparameter tuning Ranndom forest,XGB and Extra-trees regressors using RandomizedSearchCV

7.2.3 : Predict sales using the optimal parameters.

In [55]:

```
def run_tree_models(x_train,x_test,y_train,regressor,model_params,cols):
    '''Function takes model estimator as input and performs Hyperparameter tuning on trainset.
       Best parameters are taken as input to fit the best model.
       Returns test predictions.'''

    ## tuning the parameters
    clf = RandomizedSearchCV(regressor,model_params,verbose=10,cv=2,scoring='neg_mean_absolute_error')
    clf.fit(x_train[cols],y_train)
    ### fitting the best model on train data
    best_fit =  clf.best_estimator_.fit(x_train[cols],y_train)
    ### predict train_values
    train_predict = best_fit.predict(x_train[cols])
    ### predict test_values
    test_predict = best_fit.predict(x_test[cols])

    return train_predict,test_predict
```

In [57]:

```
if not os.path.isfile('pickle/rf_predictions.pkl'):
    rf_train_predict,rf_test_predict  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),RandomForestRegressor()
                                                        ,randomforest_params,top_cols_basic_feats)
    with open('pickle/rf_predictions.pkl','wb') as f:
        pkl.dump([rf_train_predict,rf_test_predict] ,f)
else:
    print('RandomForest predictions aready present in file')
    rf_train_predict,rf_test_predict = pkl.load(open('pickle/rf_predictions.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 1/2; 1/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 2.5min
[CV 2/2; 1/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 2/2; 1/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 2.5min
[CV 1/2; 2/10] START bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=500
[CV 1/2; 2/10] END bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=500; total time= 6.0min
[CV 2/2; 2/10] START bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=500
[CV 2/2; 2/10] END bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=500; total time= 5.9min
[CV 1/2; 3/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=500
[CV 1/2; 3/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=500; total time= 3.8min
[CV 2/2; 3/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=500
[CV 2/2; 3/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=500; total time= 3.7min
[CV 1/2; 4/10] START bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=12, n_estimators=100
[CV 1/2; 4/10] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=12, n_estimators=100; total time= 1.2min
[CV 2/2; 4/10] START bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=12, n_estimators=100
[CV 2/2; 4/10] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=12, n_estimators=100; total time= 1.1min
[CV 1/2; 5/10] START bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=100
[CV 1/2; 5/10] END bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=  46.0s
[CV 2/2; 5/10] START bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=100
[CV 2/2; 5/10] END bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=  44.9s
[CV 1/2; 6/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=500
[CV 1/2; 6/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=500; total time= 1.9min
[CV 2/2; 6/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=500
[CV 2/2; 6/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=500; total time= 1.9min
[CV 1/2; 7/10] START bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 1/2; 7/10] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 1.0min
[CV 2/2; 7/10] START bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 2/2; 7/10] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 1.0min
[CV 1/2; 8/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 1/2; 8/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time=222.4min
[CV 2/2; 8/10] START bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 2/2; 8/10] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time= 3.2min
[CV 1/2; 9/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100
[CV 1/2; 9/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=  23.2s
[CV 2/2; 9/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100
[CV 2/2; 9/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=  23.6s
[CV 1/2; 10/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420
[CV 1/2; 10/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420; total time= 1.6min
[CV 2/2; 10/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420
[CV 2/2; 10/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420; total time= 1.7min
```

In [18]:

```
if not os.path.isfile('pickle/et_predictions.pkl'):
    et_train_predict,et_test_predict  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),
                                                         ExtraTreesRegressor(),extra_tree_params,top_cols_basic_feats)
    with open('pickle/et_predictions.pkl','wb') as f:
        pkl.dump([et_train_predict,et_test_predict] ,f)
else:
    print('RandomForest predictions aready present in file')
    et_train_predict,et_test_predict = pkl.load(open('pickle/et_predictions.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100
[CV 1/2; 1/10] END max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=  36.0s
[CV 2/2; 1/10] START max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100
[CV 2/2; 1/10] END max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=  35.3s
[CV 1/2; 2/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=420
[CV 1/2; 2/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=420; total time= 1.3min
[CV 2/2; 2/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=420
[CV 2/2; 2/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=420; total time= 1.3min
[CV 1/2; 3/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=340
[CV 1/2; 3/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=340; total time=  43.7s
[CV 2/2; 3/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=340
[CV 2/2; 3/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=340; total time=  44.4s
[CV 1/2; 4/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=100
[CV 1/2; 4/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=100; total time=  12.9s
[CV 2/2; 4/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=100
[CV 2/2; 4/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=100; total time=  13.1s
[CV 1/2; 5/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=420
[CV 1/2; 5/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=420; total time=  54.3s
[CV 2/2; 5/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=420
[CV 2/2; 5/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=420; total time=  54.8s
[CV 1/2; 6/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500
[CV 1/2; 6/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500; total time= 1.1min
[CV 2/2; 6/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500
[CV 2/2; 6/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500; total time= 1.1min
[CV 1/2; 7/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420
[CV 1/2; 7/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420; total time= 1.9min
[CV 2/2; 7/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420
[CV 2/2; 7/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420; total time= 1.8min
[CV 1/2; 8/10] START max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=100
[CV 1/2; 8/10] END max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=100; total time=  27.5s
[CV 2/2; 8/10] START max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=100
[CV 2/2; 8/10] END max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=100; total time=  26.4s
[CV 1/2; 9/10] START max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=180
[CV 1/2; 9/10] END max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=180; total time= 1.1min
[CV 2/2; 9/10] START max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=180
[CV 2/2; 9/10] END max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=180; total time= 1.1min
[CV 1/2; 10/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=180
[CV 1/2; 10/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=180; total time=  49.3s
[CV 2/2; 10/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=180
[CV 2/2; 10/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=180; total time=  47.5s
```

In [19]:

```
if not os.path.isfile('pickle/xg_predictions.pkl'):
    xg_train_predict,xg_test_predict  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),xgb.XGBRegressor(),
                                                        xg_boost_params,top_cols_basic_feats)
    with open('pickle/xg_predictions.pkl','wb') as f:
        pkl.dump([xg_train_predict,xg_test_predict] ,f)
else:
    print('RandomForest predictions aready present in file')
    xg_train_predict,xg_test_predict = pkl.load(open('pickle/xg_predictions.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START colsample_bytree=0.7, learning_rate=0.1, max_depth=20, min_child_weight=10, n_estimators=180, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 1/10] END colsample_bytree=0.7, learning_rate=0.1, max_depth=20, min_child_weight=10, n_estimators=180, objective=reg:squarederror, subsamples=0.7; total ti
me= 1.1min
[CV 2/2; 1/10] START colsample_bytree=0.7, learning_rate=0.1, max_depth=20, min_child_weight=10, n_estimators=180, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 1/10] END colsample_bytree=0.7, learning_rate=0.1, max_depth=20, min_child_weight=10, n_estimators=180, objective=reg:squarederror, subsamples=0.7; total ti
me= 1.1min
[CV 1/2; 2/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 2/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total t
ime= 1.3min
[CV 2/2; 2/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 2/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total t
ime= 1.3min
[CV 1/2; 3/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=180, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 3/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=180, objective=reg:squarederror, subsamples=0.7; total ti
me=  38.7s
[CV 2/2; 3/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=180, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 3/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=180, objective=reg:squarederror, subsamples=0.7; total ti
me=  39.4s
[CV 1/2; 4/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 4/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.7; total t
ime= 1.5min
[CV 2/2; 4/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 4/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.7; total t
ime= 1.5min
[CV 1/2; 5/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 5/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total ti
me= 2.8min
[CV 2/2; 5/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 5/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total ti
me= 2.9min
[CV 1/2; 6/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=5, n_estimators=420, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 6/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=5, n_estimators=420, objective=reg:squarederror, subsamples=0.4; total ti
me= 2.7min
[CV 2/2; 6/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=5, n_estimators=420, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 6/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=5, n_estimators=420, objective=reg:squarederror, subsamples=0.4; total ti
me= 2.7min
[CV 1/2; 7/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=27, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.5
[CV 1/2; 7/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=27, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.5; total t
ime= 3.0min
[CV 2/2; 7/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=27, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.5
[CV 2/2; 7/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=27, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.5; total t
ime= 3.0min
[CV 1/2; 8/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.5
[CV 1/2; 8/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.5; total t
ime=  50.2s
[CV 2/2; 8/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.5
[CV 2/2; 8/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.5; total t
```

```
ime=  50.4s
[CV 1/2; 9/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=27, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 9/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=27, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4; total tim
e= 2.6min
[CV 2/2; 9/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=27, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 9/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=27, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4; total tim
e= 2.7min
[CV 1/2; 10/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.5
[CV 1/2; 10/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.5; total
time=  57.4s
[CV 2/2; 10/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.5
[CV 2/2; 10/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=260, objective=reg:squarederror, subsamples=0.5; total
time=  58.3s
```

7.3 Random-Forest & XGBoost & ExtraTrees Regressors using Advanced Features(rolling features)

7.3.1: let's take the best selected features from our forward feature selection from EDA along with rolling the features

In [88]:

```
rolling_features = ['rolling_mean', 'expanding_mean', 'EWM_0.1', 'EWM_0.4',
       'EWM_0.5', 'EWM_0.7', 'lag_52', 'holt_avg']
top_cols_with_rolling_features = top_cols_basic_feats + rolling_features
```

7.3.2 : Hyperparameter tuning Ranndom forest,XGB and Extra-trees regressors using RandomizedSearchCV

7.3.2 : Predict sales using the optimal parameters.

In [74]:
```
if not os.path.isfile('pickle/rf_predictions_with_advanced_feats.pkl'):
    rf_train_predict2,rf_test_predict2  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),
                                                       RandomForestRegressor(),randomforest_params,top_cols_with_rolling_features)
    with open('pickle/rf_predictions_with_advanced_feats.pkl','wb') as f:
        pkl.dump([rf_train_predict2,rf_test_predict2] ,f)
else:
    print('RandomForest predictions with advanced features already present in file')
    rf_train_predict2,rf_test_predict2 = pkl.load(open('pickle/rf_predictions_with_advanced_feats.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 1/2; 1/10] END bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time= 2.9min
[CV 2/2; 1/10] START bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 2/2; 1/10] END bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time= 3.0min
[CV 1/2; 2/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420
[CV 1/2; 2/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420; total time= 2.2min
[CV 2/2; 2/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420
[CV 2/2; 2/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=420; total time= 2.2min
[CV 1/2; 3/10] START bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=12, n_estimators=500
[CV 1/2; 3/10] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=12, n_estimators=500; total time= 3.1min
[CV 2/2; 3/10] START bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=12, n_estimators=500
[CV 2/2; 3/10] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=12, n_estimators=500; total time= 3.1min
[CV 1/2; 4/10] START bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=100
[CV 1/2; 4/10] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=  25.3s
[CV 2/2; 4/10] START bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=100
[CV 2/2; 4/10] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=  25.8s
[CV 1/2; 5/10] START bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 1/2; 5/10] END bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 3.3min
[CV 2/2; 5/10] START bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 2/2; 5/10] END bootstrap=False, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 3.3min
[CV 1/2; 6/10] START bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 1/2; 6/10] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time= 2.9min
[CV 2/2; 6/10] START bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420
[CV 2/2; 6/10] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=420; total time= 3.0min
[CV 1/2; 7/10] START bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 1/2; 7/10] END bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 2.2min
[CV 2/2; 7/10] START bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 2/2; 7/10] END bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 2.2min
[CV 1/2; 8/10] START bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=500
[CV 1/2; 8/10] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=500; total time= 8.6min
[CV 2/2; 8/10] START bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=500
[CV 2/2; 8/10] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=12, n_estimators=500; total time= 3.2min
[CV 1/2; 9/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 1/2; 9/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 2.2min
[CV 2/2; 9/10] START bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500
[CV 2/2; 9/10] END bootstrap=True, max_depth=27, max_features=sqrt, min_samples_leaf=6, min_samples_split=5, n_estimators=500; total time= 2.3min
[CV 1/2; 10/10] START bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 1/2; 10/10] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 8.2min
[CV 2/2; 10/10] START bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340
[CV 2/2; 10/10] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=6, min_samples_split=5, n_estimators=340; total time= 8.3min
```

In [80]:

```
if not os.path.isfile('pickle/et_predictions_with_advanced_feats.pkl'):
    et_train_predict2,et_test_predict2  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),
                                                          ExtraTreesRegressor(),extra_tree_params,top_cols_with_rolling_features)
    with open('pickle/et_predictions_with_advanced_feats.pkl','wb') as f:
        pkl.dump([et_train_predict2,et_test_predict2] ,f)
else:
    print('Extra trees predictions with advanced features already present in file')
    et_train_predict2,et_test_predict2 = pkl.load(open('pickle/et_predictions_with_advanced_feats.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420
[CV 1/2; 1/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420; total time= 3.0min
[CV 2/2; 1/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420
[CV 2/2; 1/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=420; total time= 3.0min
[CV 1/2; 2/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500
[CV 1/2; 2/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500; total time= 1.1min
[CV 2/2; 2/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500
[CV 2/2; 2/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=6, n_estimators=500; total time= 1.1min
[CV 1/2; 3/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100
[CV 1/2; 3/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; total time=  17.8s
[CV 2/2; 3/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100
[CV 2/2; 3/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; total time=  18.1s
[CV 1/2; 4/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=180
[CV 1/2; 4/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=180; total time=  24.5s
[CV 2/2; 4/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=180
[CV 2/2; 4/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=180; total time=  25.0s
[CV 1/2; 5/10] START max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=260
[CV 1/2; 5/10] END max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=260; total time= 1.9min
[CV 2/2; 5/10] START max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=260
[CV 2/2; 5/10] END max_features=auto, min_samples_leaf=6, min_samples_split=2, n_estimators=260; total time= 1.9min
[CV 1/2; 6/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=260
[CV 1/2; 6/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=260; total time= 1.9min
[CV 2/2; 6/10] START max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=260
[CV 2/2; 6/10] END max_features=auto, min_samples_leaf=6, min_samples_split=6, n_estimators=260; total time= 1.8min
[CV 1/2; 7/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=340
[CV 1/2; 7/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=340; total time= 1.1min
[CV 2/2; 7/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=340
[CV 2/2; 7/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=340; total time= 1.1min
[CV 1/2; 8/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=340
[CV 1/2; 8/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=340; total time= 1.0min
[CV 2/2; 8/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=340
[CV 2/2; 8/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=340; total time= 1.0min
[CV 1/2; 9/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=260
[CV 1/2; 9/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=260; total time=  47.0s
[CV 2/2; 9/10] START max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=260
[CV 2/2; 9/10] END max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=260; total time=  47.3s
[CV 1/2; 10/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=500
[CV 1/2; 10/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=500; total time= 1.1min
[CV 2/2; 10/10] START max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=500
[CV 2/2; 10/10] END max_features=sqrt, min_samples_leaf=6, min_samples_split=2, n_estimators=500; total time= 1.1min
```

In [90]:

```
if not os.path.isfile('pickle/xg_predictions_with_advanced_feats.pkl'):
    xg_train_predict2,xg_test_predict2  =  run_tree_models(x_train.copy(),x_test.copy(),y_train.copy(),
                                                         xgb.XGBRFRegressor(),xg_boost_params,top_cols_with_rolling_features)
    with open('pickle/xg_predictions_with_advanced_feats.pkl','wb') as f:
        pkl.dump([xg_train_predict2,xg_test_predict2] ,f)
else:
    print('XGB predictions with advanced features already present in file')
    xg_train_predict2,xg_test_predict2 = pkl.load(open('pickle/xg_predictions_with_advanced_feats.pkl','wb'))
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV 1/2; 1/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 1/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4; total time=
3.4min
[CV 2/2; 1/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 1/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.4; total time=
3.2min
[CV 1/2; 2/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 2/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.7; total time
= 1.9min
[CV 2/2; 2/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 2/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.7; total time
= 1.8min
[CV 1/2; 3/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=260, objective=reg:squarederror, subsamples=0.5
[CV 1/2; 3/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=260, objective=reg:squarederror, subsamples=0.5; total time=
1.9min
[CV 2/2; 3/10] START colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=260, objective=reg:squarederror, subsamples=0.5
[CV 2/2; 3/10] END colsample_bytree=0.5, learning_rate=0.01, max_depth=27, min_child_weight=5, n_estimators=260, objective=reg:squarederror, subsamples=0.5; total time=
1.9min
[CV 1/2; 4/10] START colsample_bytree=0.7, learning_rate=0.1, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 4/10] END colsample_bytree=0.7, learning_rate=0.1, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.4; total time=
2.3min
[CV 2/2; 4/10] START colsample_bytree=0.7, learning_rate=0.1, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 4/10] END colsample_bytree=0.7, learning_rate=0.1, max_depth=30, min_child_weight=20, n_estimators=260, objective=reg:squarederror, subsamples=0.4; total time=
2.2min
[CV 1/2; 5/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 5/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total time
= 1.8min
[CV 2/2; 5/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 5/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=27, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total time
= 1.8min
[CV 1/2; 6/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 6/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total time=
3.0min
[CV 2/2; 6/10] START colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 6/10] END colsample_bytree=0.5, learning_rate=0.1, max_depth=20, min_child_weight=20, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total time=
3.0min
[CV 1/2; 7/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 7/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total time=
2.4min
[CV 2/2; 7/10] START colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 7/10] END colsample_bytree=0.4, learning_rate=0.01, max_depth=20, min_child_weight=5, n_estimators=500, objective=reg:squarederror, subsamples=0.7; total time=
2.4min
[CV 1/2; 8/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 8/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total time
= 2.3min
[CV 2/2; 8/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.4
```

```
[CV 2/2; 8/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=20, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total time
= 2.3min
[CV 1/2; 9/10] START colsample_bytree=0.4, learning_rate=0.1, max_depth=27, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.7
[CV 1/2; 9/10] END colsample_bytree=0.4, learning_rate=0.1, max_depth=27, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.7; total time=
1.8min
[CV 2/2; 9/10] START colsample_bytree=0.4, learning_rate=0.1, max_depth=27, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.7
[CV 2/2; 9/10] END colsample_bytree=0.4, learning_rate=0.1, max_depth=27, min_child_weight=20, n_estimators=340, objective=reg:squarederror, subsamples=0.7; total time=
1.8min
[CV 1/2; 10/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4
[CV 1/2; 10/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total tim
e= 3.2min
[CV 2/2; 10/10] START colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4
[CV 2/2; 10/10] END colsample_bytree=0.7, learning_rate=0.01, max_depth=30, min_child_weight=10, n_estimators=340, objective=reg:squarederror, subsamples=0.4; total tim
e= 3.1min
```

7.4 SARIMA(Seasonal Auto Regression Integrated Moving Average)(p, d, q)(P, D, Q, S):

Introduction :

SARIMA is a powerful tool in predicting time-series such as weekly-sales.As the name suggests it's useful in factoring seasonality unlike ARMIA.We checked in our EDA part that our sales data has additive yearly seasonality.SARIMA is determined by combination of non-seasonal ARIMA (p,q,d) and seasonal ARIMA(P,Q,D,S)

Quick walk through of terms :

S(AR){P} -- P is the number of lags to predict the future values
S(I){D} ---- D is the seasonal differencing part refers to order of difference,also ensures if the series is stationary. Eg. if d=1 then yhat = yt - y(t-1), if d=2 then yhat=(yt-y(t-1)) + (y(t-1)-y(t-2)) here yhat in d=2 shows change in change
S(MA){Q} ---- Q is quantity of lagged forecasting errors.
S{S} ----- S is the season's length.(yearly=12,quaterly=4)

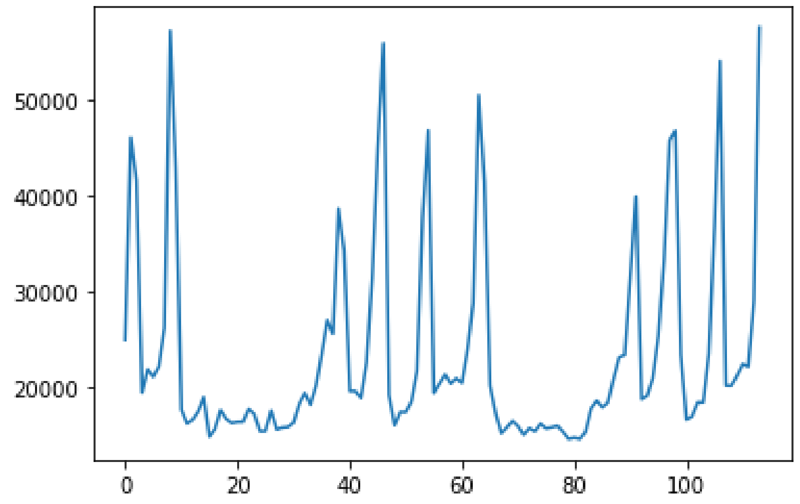
D should not be more than 1 and d+D not be more than 2.if d+D=2 then constant should be supressed.

In [36]:
```
### import libraries
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
import itertools
```

In [53]:
```
### consider date and sales columns only
sarima_train = x_train[['Date','Weekly_Sales']].set_index('Date')
```

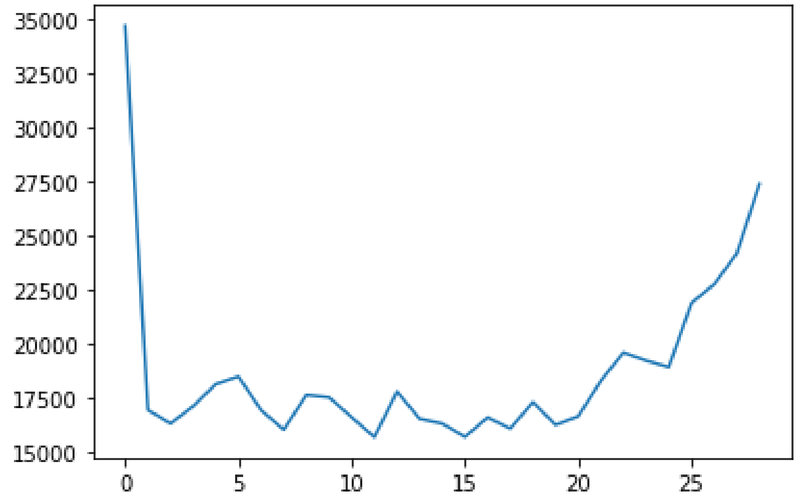
In [54]:
```
Y_train[:len_].plot()
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x17d4253e320>



In [55]:
```
Y_cv[:143-len_].plot()
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x17d42580b70>



7.4.1 ADFuller Test - Check stationarity of data

In [56]:

```
data = sarima_train.resample('W-FRI').mean()  ## data is weekly and minimum date '2010-02-05' was on friday
```

In [57]:

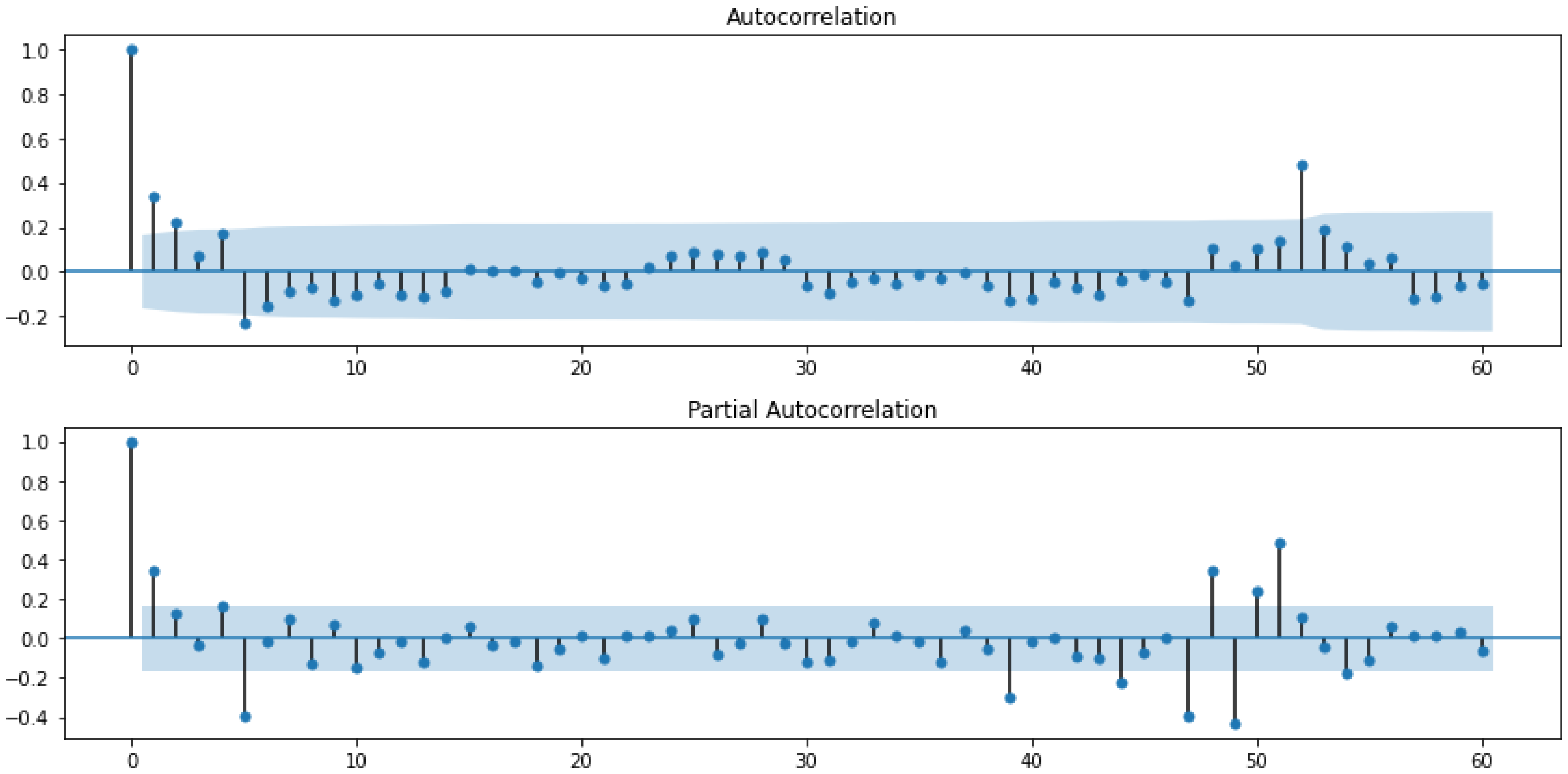
```
print('Dickey-Fuller Test:')
result = adfuller(data)
adftest = pd.Series(result[0:4], index=['Test Statistic','p-value','Lags Used','Observations'])
adftest = np.round(adftest,10)

for key, value in result[4].items():
    adftest["Critical Value (%s)"%key] = value.round(4)

print(adftest)

if adftest[1] < 0.05:
    print('\np-value Value lower than 0.05%.\nWe accept H0(null hypothesis)!!The series seems to be stationary')
else:
    print("\np-value Value higher than 0.05%.\nWe reject H0(null hypothesis)!The series isn't stationary")
```

```
Dickey-Fuller Test:
Test Statistic         -5.908398e+00
p-value                 2.675000e-07
Lags Used               4.000000e+00
Observations            1.380000e+02
Critical Value (1%)    -3.478600e+00
Critical Value (5%)    -2.882700e+00
Critical Value (10%)   -2.578100e+00
dtype: float64

p-value Value lower than 0.05%.
We accept H0(null hypothesis)!!The series seems to be stationary
```

7.4.2 PLOT ACF(Auto-Correlation factor plot) AND PCF(Partial-Correlation Factor plot)

In [58]:

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
fig,ax = plt.subplots(2,figsize=(12,6))
ax[0] = plot_acf(data,ax=ax[0],lags=60)
ax[1] = plot_pacf(data,ax=ax[1],lags=60)
plt.tight_layout()
```



Observations :

```
* ACF plot : Lag 52 shows high corelation with predictor.
* PACF plot : Lag 48 and 51 individually promote high corelations,similarly lag1 is also useful.
* PACF plot : Lag 47 is highly negetively correlated.
```

Theory:

```
* ACF plot shows 52 weeks past data cummulatively to predict future sales.
* PACF plot shows that to predict a future value we can lag back 51 weeks value.
* The seasonal component (s) should be 52.
```

7.4.3 Building SARIMA

In [59]:

```python
p  = np.arange(0, 2)
d  = np.arange(0,2)
q = np.arange(0,2)
pdq = list(itertools.product(p,q,d))
seasonal_pdq = [(x[0],x[1],x[2],52) for x in list(itertools.product(p,q,d))]
###
print('Example combinations of SARIMA...')
print('SARIMA----{} x {}'.format(pdq[1],seasonal_pdq[1]))
```

```
Example combinations of SARIMA...
SARIMA----(0, 0, 1) x (0, 0, 1, 52)
```

In [60]:

```python
time_data = x_train[['Store','Dept','Date','Weekly_Sales']].set_index('Date')
```

7.4.4 : Hyperparameter tuning Sarima to find optimal (p,q,d,s)

7.4.4 : Predict sales using the optimal parameters.

In [37]:

```
def tuning_sarima(time_data,y_cv,store,dept):
    ''''Returns best parameter for each store dept combination.
        Returns cv predictions after tuning for (p,q,d,s)'''
    params = []
    i = 0
    sarima_predict_cv = []
    MAE = []

    for pm in pdq:
        for pm_seasonal in seasonal_pdq:
            #tdf = tdf.resample('MS').mean()
            model = sm.tsa.statespace.SARIMAX(tdf,
                                              order=pm,simple_differencing=True,
                                              seasonal_order=pm_seasonal,
                                              enforce_invertibility=False,
                                              enforce_stationarity=False)
            model_aic = model.fit()
            ## store  aic and bic values for each itration
            forecast = model_aic.get_forecast(steps=29)
            future_forecast = forecast.predicted_mean
            mae_err = mean_absolute_error(y_cv[29*i:29*i + 29],future_forecast)
            MAE.append((pm,pm_seasonal,mae_err))

    ## choose model with least aic
    best_params = MAE[np.argmin(list(map(itemgetter(2),MAE)))]

    ## predict cv with best model
    model = sm.tsa.statespace.SARIMAX(tdf, order=best_params[0],seasonal_order=best_params[1],
                                      simple_differencing=True,
                                      enforce_invertibility=False, enforce_stationarity=False)
    model_aic = model.fit()
    forecast = model_aic.get_forecast(steps=29)
    future_forecast = forecast.predicted_mean
    return (store,dept,best_params),future_forecast

```

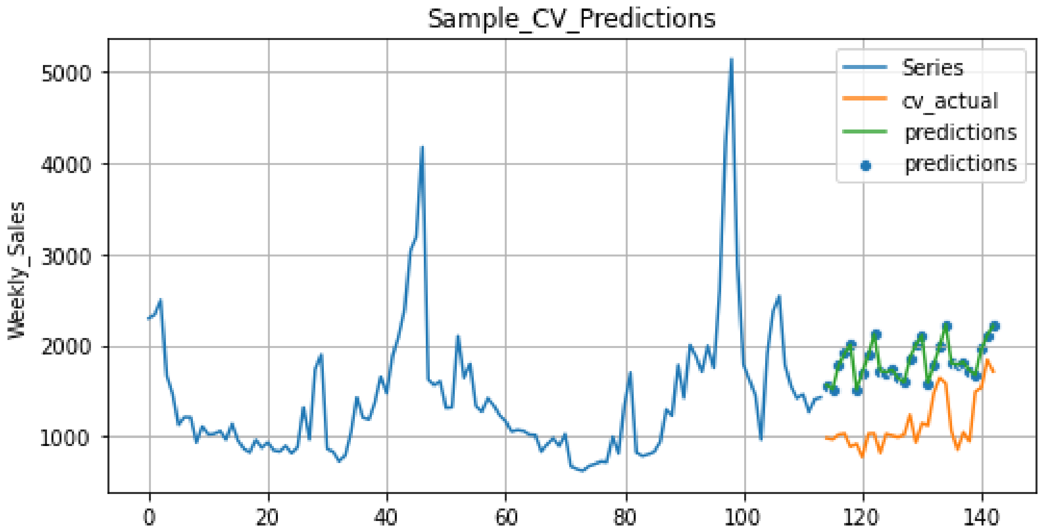
In [126]:

```
ustore  = sorted(X_train.Store.unique()) ## unique stores
udept  = sorted(X_train.Dept.unique()) ## unique departments
sarima_cv_pred = []
best_params_multiple = []
i=0
start =  datetime.now()
if not os.path.isfile('pickle/sarima_tune_with_sl_12.pkl'):
    print('Tunee series for {} stores and {} depts'.format(len(ustore),len(udept))," Combination:",len(ustore)*len(udept) )
    print('Start ---',datetime.now())
    for store in ustore[:1]:
        for dept in udept:
            tdf = time_data.loc[(time_data.Store == store)&(time_data.Dept == dept)]['Weekly_Sales']
            if len(tdf)!=0:
                params,pred = tuning_sarima(X_train.set_index('Date'),Y_cv,store,dept)
                best_params_multiple.append(params)
                sarima_cv_pred.extend(pred)
                i+=1
                if i%20 == 0 :
                    print('Predictions done for {}/3645 combos!!'.format(i),'----------Elapsed:',datetime.now()-start)
                    print('Best-params for store:',store,' dept:',dept,' ',' pdq=',params[2][0],' pdqs=',params[2][1],' MAE=',params[2][2])
    with open('pickle/sarima_tune_with_sl_12.pkl','wb') as f:
        pkl.dump([best_params_multiple,sarima_cv_pred],f)
else:
    print('Tuned series already present in disk!!!!')
    best_params_multiple,sarima_cv_pred = pkl.load(open('pickle/sarima_tune_with_sl_12.pkl','rb'))
print('Total time taken to run the cell:',datetime.now()-start)
```

```
Tuned series already present in disk!!!!
Total time taken to run the cell: 0:00:00.003991
```

In [150]:

```
def plot_predictions(index_lst,train_lst,pred_list,pred_len,train_len,nrows,ncol,title):
    cnt=0
    fig = plt.figure(figsize=(8,4))
    for i in index_lst:

        series_range = np.arange(0,train_len)

        cv_actual_range = np.arange(114,114+29)

        pred_range = np.arange(train_len,train_len+pred_len)

        sns.lineplot(x = series_range , y = train_lst[train_len*i:train_len*(i+1)],label='Series')


        sns.lineplot(x = cv_actual_range, y = Y_cv[29*(i):29*(i)+29],label='cv_actual',palette=['orange'])
        #sns.scatterplot(x = cv_actual_range, y = Y_cv[29*(i):29*(i)+29],markers='b')

        sns.lineplot(x = pred_range , y = pred_list[pred_len*i: pred_len*(i+1)],label='predictions',palette=['green'])
        sns.scatterplot(x = pred_range , y = pred_list[pred_len*i: pred_len*(i+1)],label='predictions',markers='r')

        plt.title(title)
        plt.grid()
        plt.show()

plot_predictions([25],Y_train,sarima_cv_pred,29,114,2,2,'Sample_CV_Predictions')
```



In [16]:

```
print('WMAE for CV for SARIMA is calculated to be:',cal_wmae(X_cv['IsHoliday'].values,Y_cv,sarima_cv_pred))
print('MAE for CV for SARIMA is calculated to be:',mean_absolute_error(Y_cv,sarima_cv_pred))
print('R2 for CV for SARIMA is calculated to be:',(r2_score(Y_cv,sarima_cv_pred)))
```

```
WMAE for CV for SARIMA is calculated to be: 7853.771383629283
MAE for CV for SARIMA is calculated to be: 7597.727465692465
R2 for CV for SARIMA is calculated to be: 0.04511809110799647
```

7.4.5: Refit the model to predict test data using best params

In [239]:

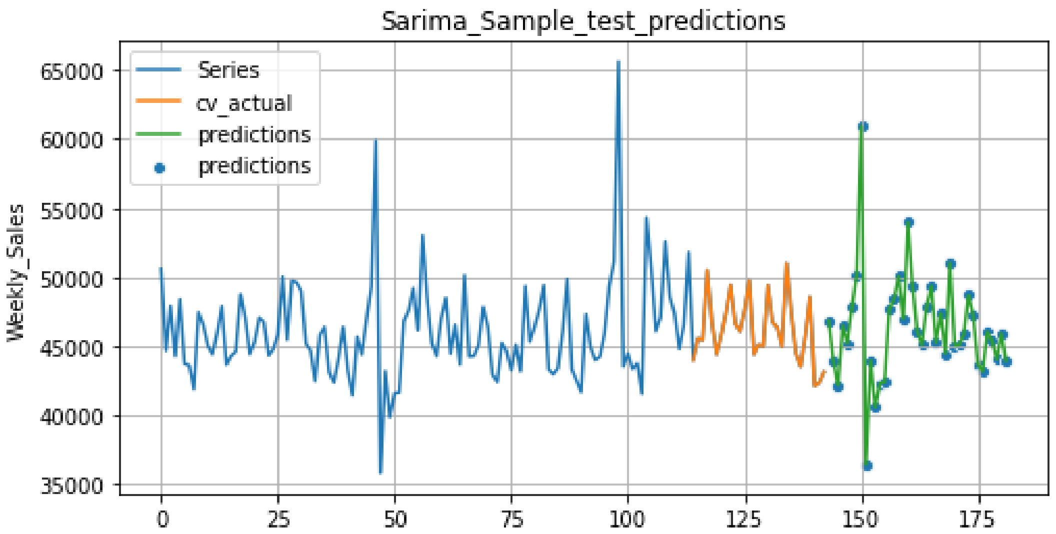
```
cnt=-1
sarima_test_pred = []
start =  datetime.now()
i=0
# pkl.dump(sarima_test_pred,open('pickle/sarima_predictions_52x2.pkl','wb'))
if not os.path.isfile('pickle/sarima_predictions_52x2.pkl'):
    for store in ustore:
            for dept in udept:
                tdf = x_train.loc[(x_train.Store == store)&(x_train.Dept == dept)]['Weekly_Sales']
                if len(tdf)!=0:
                    pm=best_params_multiple[i][2][0]
                    pm_seasonal = best_params_multiple[i][2][1]
                    pm_seasonal = list(pm_seasonal)
                    pm_seasonal[3] = 52*2
                    pm_seasonal = tuple(pm_seasonal)

                    model = sm.tsa.statespace.SARIMAX(tdf,order=pm,
                                                      seasonal_order=pm_seasonal,
                                                      simple_differencing=True,
                                                      enforce_invertibility=False,
                                                      enforce_stationarity=False)
                    model_aic = model.fit()
                    forecast = model_aic.get_forecast(steps=39)
                    future_forecast = forecast.predicted_mean
                    sarima_test_pred.extend(future_forecast)
                    i+=1
            print('Predictions completd till store :',store,'Elapsed--------',datetime.now()-start)
    with open('pickle/sarima_predictions_52x2.pkl','rb') as f:
         pkl.dump(sarima_test_pred,open('pickle/sarima_predictions_52x2.pkl','wb'))
else:
    print('Sarima test predictions present in disk!!')
    sarima_test_pred =  pkl.load(open('pickle/sarima_predictions_52x2.pkl','rb'))

```

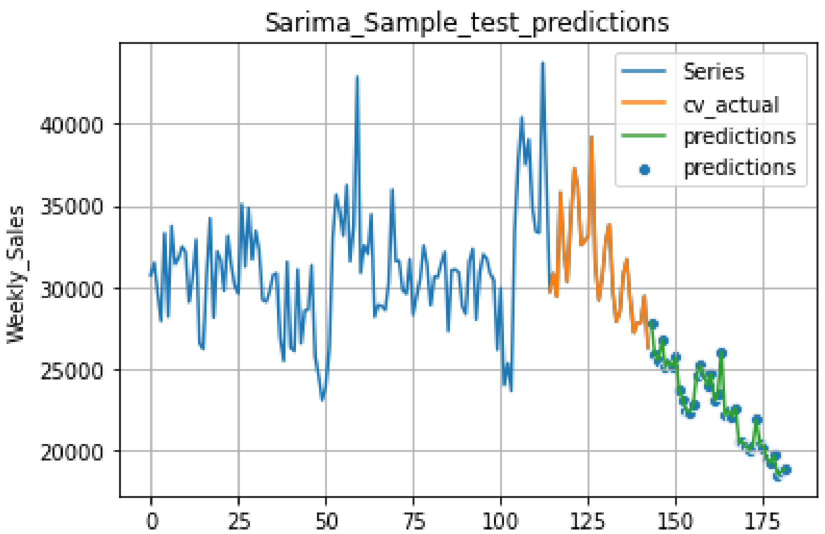
```
Predictions completd till store : 1 Elapsed-------- 0:07:04.812003
Predictions completd till store : 2 Elapsed-------- 0:13:53.519485
Predictions completd till store : 3 Elapsed-------- 0:21:15.051378
Predictions completd till store : 4 Elapsed-------- 0:25:46.190389
Predictions completd till store : 5 Elapsed-------- 0:32:09.859393
Predictions completd till store : 6 Elapsed-------- 0:37:50.993468
Predictions completd till store : 7 Elapsed-------- 0:47:28.817425
Predictions completd till store : 8 Elapsed-------- 0:55:09.193705
Predictions completd till store : 9 Elapsed-------- 1:02:14.581647
Predictions completd till store : 10 Elapsed-------- 1:08:32.199262
Predictions completd till store : 11 Elapsed-------- 1:15:51.753876
Predictions completd till store : 12 Elapsed-------- 1:23:59.174623
Predictions completd till store : 13 Elapsed-------- 1:31:29.697864
Predictions completd till store : 14 Elapsed-------- 1:37:47.562660
Predictions completd till store : 15 Elapsed-------- 1:46:00.849340
Predictions completd till store : 16 Elapsed-------- 1:53:56.592884
Predictions completd till store : 17 Elapsed-------- 2:04:21.607009
Predictions completd till store : 18 Elapsed-------- 2:12:34.089105
Predictions completd till store : 19 Elapsed-------- 2:20:27.532379
Predictions completd till store : 20 Elapsed-------- 2:29:01.863203
Predictions completd till store : 21 Elapsed-------- 2:40:36.097532
Predictions completd till store : 22 Elapsed-------- 2:47:40.487439
Predictions completd till store : 23 Elapsed-------- 2:53:17.631815
```

```
Predictions completd till store : 24 Elapsed-------- 3:00:19.094662
Predictions completd till store : 25 Elapsed-------- 3:06:35.723387
Predictions completd till store : 26 Elapsed-------- 3:13:16.269972
Predictions completd till store : 27 Elapsed-------- 3:20:30.442107
Predictions completd till store : 28 Elapsed-------- 3:28:56.916515
Predictions completd till store : 29 Elapsed-------- 3:35:49.222004
Predictions completd till store : 30 Elapsed-------- 3:40:04.082086
Predictions completd till store : 31 Elapsed-------- 3:47:04.659556
Predictions completd till store : 32 Elapsed-------- 3:53:10.486787
Predictions completd till store : 33 Elapsed-------- 3:57:51.869713
Predictions completd till store : 34 Elapsed-------- 4:05:52.638631
Predictions completd till store : 35 Elapsed-------- 5:44:29.299254
Predictions completd till store : 36 Elapsed-------- 5:48:31.431989
Predictions completd till store : 37 Elapsed-------- 5:53:54.087153
Predictions completd till store : 38 Elapsed-------- 6:00:09.347443
Predictions completd till store : 39 Elapsed-------- 6:07:30.372114
Predictions completd till store : 40 Elapsed-------- 6:15:50.117874
Predictions completd till store : 41 Elapsed-------- 6:22:12.365433
Predictions completd till store : 42 Elapsed-------- 6:28:11.591011
Predictions completd till store : 43 Elapsed-------- 6:32:08.734174
Predictions completd till store : 44 Elapsed-------- 6:37:01.160461
Predictions completd till store : 45 Elapsed-------- 6:43:43.979831
```

In [151]:

```
plot_predictions([1,9],y_train,sarima_test_pred,39,143,2,2,'Sarima_Sample_test_predictions')
```





7.5 FBProphet

[https://facebook.github.io/prophet/ (https://facebook.github.io/prophet/)](https://facebook.github.io/prophet/)

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

In [21]:
```
from fbprophet import Prophet
### using prophet we can also include the holiday effects to impact our predictions
### let's create our holdiay dataframe with date and holiday name
## https://facebook.github.io/prophet/docs/seasonality,_holiday_effects,_and_regressors.html
```

In [22]:
```
# References : 0:Not Holiday ,1:Thanksgiving ,2:Labour day, 3:Christmas, 4: Superbowl
Thanskgiving = pd.DataFrame({'holiday':'Thanksgiving','ds':pd.to_datetime(trte_featured.loc[(trte_featured.major_holiday==1)].Date.unique())})
LabourDay = pd.DataFrame({'holiday':'LabourDay','ds':pd.to_datetime(trte_featured.loc[(trte_featured.major_holiday==2)].Date.unique())})
Christmas = pd.DataFrame({'holiday':'Christmas','ds':pd.to_datetime(trte_featured.loc[(trte_featured.major_holiday==3)].Date.unique())})
SuperBowl = pd.DataFrame({'holiday':'SuperBowl','ds':pd.to_datetime(trte_featured.loc[(trte_featured.major_holiday==4)].Date.unique())})
holidays = pd.concat([Thanskgiving,LabourDay,Christmas,SuperBowl])
```

In [23]:
```
df = pd.DataFrame({'holiday':['Christmas','Thanksgiving','LabourDay'],
                  'ds':[pd.to_datetime('2013-12-27'),pd.to_datetime('2013-11-29'),pd.to_datetime('2013-11-29')]})
holidays = holidays.append(df,ignore_index=True)
holidays = holidays.sort_values(by='holiday').reset_index(drop=True)
holidays
```

Out[23]:

	holiday	ds
0	Christmas	2010-12-24
1	Christmas	2011-12-23
2	Christmas	2012-12-21
3	Christmas	2013-12-27
4	LabourDay	2010-09-10
5	LabourDay	2011-09-09
6	LabourDay	2012-09-07
7	LabourDay	2013-11-29
8	SuperBowl	2010-02-12
9	SuperBowl	2011-02-11
10	SuperBowl	2012-02-10
11	SuperBowl	2013-02-08
12	Thanksgiving	2010-11-26
13	Thanksgiving	2011-11-25
14	Thanksgiving	2012-11-23
15	Thanksgiving	2013-11-29

7.5.1 For each store-dept combination run prophet and predict for test and train

7.5.1 For each store-dept combination run prophet and predict for test and train

In [155]:

```
prophet_test_pred = []
start = datetime.now()
i=0
if not os.path.isfile('pickle/prophet_predictions.pkl'):
    for store in ustore:
        for dept in udept:
            tdf = x_train.loc[(x_train.Store==store)&((x_train.Dept==dept))][['Date','Weekly_Sales']].reset_index(drop=True)
            if len(tdf)!=0:
            ### fbprophet takes input with columns 'ds' of type datetime and y of type float
                tdf = tdf.rename(columns={'Date':'ds','Weekly_Sales':'y'})

                ## let's include our holiday dataframe
                prophet = Prophet(holidays=holidays)

                ## let's fit our prophet model
                prophet.fit(tdf)
                ### predicting using prophet model
                forecast = prophet.make_future_dataframe(periods=39, freq='W-FRI')
                forecast = prophet.predict(forecast)

                ## storing forecasted value in list
                prophet_train_pred.extend(forecast['yhat'][:143])
                prophet_test_pred.extend(forecast['yhat'][143:])

                if i%20 == 0:
                    print('Predictions done till Store,Dept','(',store,',',dept,')')
                    print('time_elapsed:',datetime.now()-start)
                i+=1
        with open('pickle/prophet_predictions.pkl','wb') as f:
             pkl.dump([prophet_train_pred,prophet_test_pred],f)
else:
    print('Prophet test predictions present in disk !!')
    prophet_train_pred,prophet_test_pred = pkl.load(open('pickle/prophet_predictions.pkl','rb'))
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

7.5.2 Plot Prophet prediction for random store

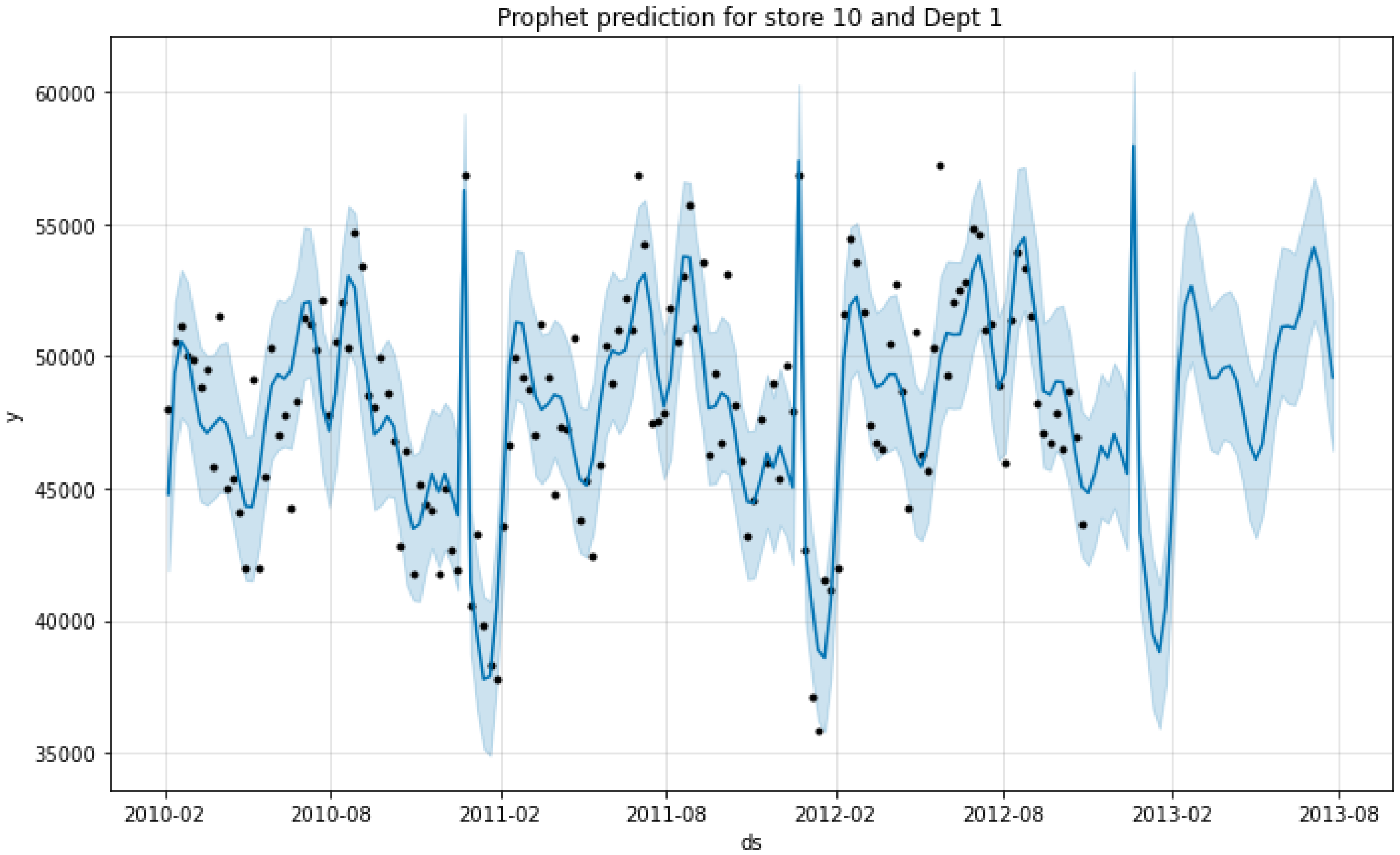
In [295]:

```
tdf = x_train.loc[(x_train.Store==10)&((x_train.Dept==10))][['Date','Weekly_Sales']].reset_index(drop=True)
tdf = tdf.rename(columns={'Date':'ds','Weekly_Sales':'y'})
prophet = Prophet(holidays=holidays)
prophet.fit(tdf)
forecast = prophet.make_future_dataframe(periods=39, freq='W-FRI')
forecast = prophet.predict(forecast)
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

In [296]:

```
prophet.plot(forecast)
plt.title('Prophet prediction for store 10 and Dept 1')
plt.show()
```



- The light blue band shows the predicted weekly lower and upper sales range.
- the dark blue lineplot shows our predicted yhat
- the scatter plot is the actual values

Observations: We see that the predicted values overlaps well with our actual values.As we included the holiday dates of christmas,superbowl,labour day and thanksgiving in our prophet model we are able to see good spike in predicted sales during these weeks.

10. Error Table

In [215]:

```
model_names = [('Baseline',baseline_predict_tr,baseline_predict_cv,baseline_predict_te),
               ('Holt_winters',holts_train_predict,'-',holts_test_predict),
               ('RandomForest',rf_train_predict,'-',rf_test_predict),
               ('ExtraTrees',et_train_predict,'-',et_test_predict),
               ('XGBoost',xg_train_predict,'-',xg_test_predict),
              ('RandomForest_with_rollingfeatures',rf_train_predict2,'-',rf_test_predict2),
               ('ExtraTrees_with_rollingfeatures',et_train_predict2,'-',et_test_predict2),
               ('XGBoost_with_rollingfeatures',xg_train_predict2,'-',xg_test_predict2),
               ('Sarima','-',sarima_cv_pred,sarima_test_pred),
               ('Prophet',prophet_train_pred,'-',prophet_test_pred)]
```

In [272]:

```
err_dict = dict()
kaggle_test_scores = [6237.80,3945.51,2734.01,3030.65,4276.90,3941.69,3891.69,3229.916,6551.97,2851.96]
i=0
for name in model_names:
    err_dict[name[0]] = {}
    if name[1] != '-':
        wmae = cal_wmae(x_train['IsHoliday'].values,y_train.values,name[1])
        r2,mae = calculate_r2_mae(y_train.values,name[1])
        err_dict[name[0]]['Train'] = {'WMAE':wmae,'R2':r2,'MAE':mae}
    if name[2] != '-':
        wmae = cal_wmae(X_cv['IsHoliday'].values,Y_cv.values,name[2])
        r2,mae = calculate_r2_mae(Y_cv.values,name[2])
        err_dict[name[0]]['CV'] = {'WMAE':wmae,'R2':r2,'MAE':mae}
    err_dict[name[0]]['Test'] = {'WMAE':kaggle_test_scores[i]}
    i+=1
```

In [283]:

```
#https://www.geeksforgeeks.org/nested-dictionary-to-multiindex-dataframe/
reformed_dict = {}
for outerKey, innerDict in err_dict.items():
    for innerKey, values in innerDict.items():
        reformed_dict[(outerKey, innerKey)] = values

# Display multiindex dataframe
multiIndex_df = pd.DataFrame(reformed_dict)


# making a green border
multiIndex_df.T.fillna('-').style.set_table_styles([{'selector' : '',
                            'props' : [('border',
                                        '2px solid blue')]}])
```

Out[283]:

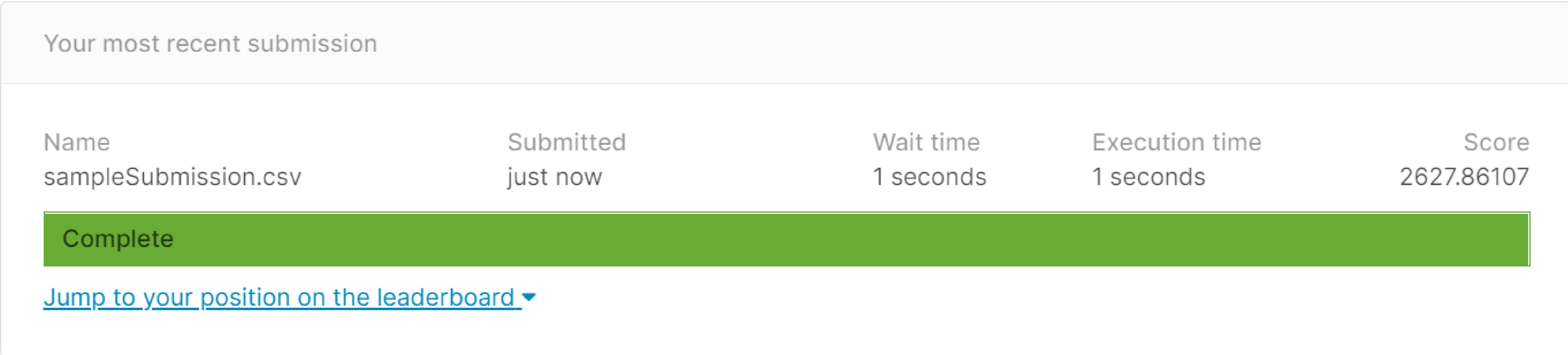
		WMAE	R2	MAE
Baseline	**Train**	3492.988596	0.815249	3140.736298
	CV	1686.249023	0.971261	1661.493771
	Test	6237.800000	-	-
Holt_winters	**Train**	620.350820	0.995928	560.516662
	Test	3945.510000	-	-
RandomForest	**Train**	1192.151524	0.983502	1033.879459
	Test	2734.010000	-	-
ExtraTrees	**Train**	1289.080282	0.980988	1096.952667
	Test	3030.650000	-	-
XGBoost	**Train**	2432.774696	0.950219	2337.912480
	Test	4276.900000	-	-
RandomForest_with_rollingfeatures	**Train**	509.742171	0.997217	459.447750
	Test	3941.690000	-	-
ExtraTrees_with_rollingfeatures	**Train**	490.185697	0.996645	437.359134
	Test	3891.690000	-	-
XGBoost_with_rollingfeatures	**Train**	1256.732012	0.984532	1183.544362
	Test	3229.916000	-	-
Sarima	**CV**	7853.771384	0.045118	7597.727466
	Test	6551.970000	-	-
Prophet	**Train**	1060.361034	0.987964	1037.049374
	Test	2851.960000	-	-

- Observations :
- We see that the model which best performs on test data is Randomforest with WMAE of 2734.01 followed by prophet model with WMAE of 2851 on test data.
- Theory : Given top two models with least WMAE ,let's try stacking models for RandomForest and Prophet and see if the WMAE reduces further

Stacking Models:

- In order to make a robust predictive models when model ambiguity is tall it in turn diminishes the quality of prediction!! One effective way is to form agreement between many models. By averaging out between models we can even out overestimation and underestimation.
- let's consider our top two models RandomForest and Prophet
- As we already have our prophet and randomforest predictions trained with best hyperparameters we dont need to train again.

In [289]:
```
if not os.path.isfile('pickle/stacking_predictions.pkl'):
    rf_prophet_pred_test = np.column_stack([prophet_test_pred,rf_test_predict])
    stacking_predictions_test = np.mean(rf_prophet_pred_test,axis=1)
    ##
    rf_prophet_pred_train = np.column_stack([prophet_train_pred,rf_train_predict])
    stacking_predictions_train = np.mean(rf_prophet_pred_train,axis=1)
    with open('pickle/stacking_predictions.pkl','wb') as f:
        pkl.dump([stacking_predictions_train,stacking_predictions_test] ,f)
else:
    print('Stacked predictions present in disk!!')
    stacking_predictions_train,stacking_predictions_test = pkl.load(open('pickle/stacking_predictions.pkl','rb'))
```



- As we can see our stacked model further reduced our Weighted-mean-absolute-error to 2627.861 with Rank 20.

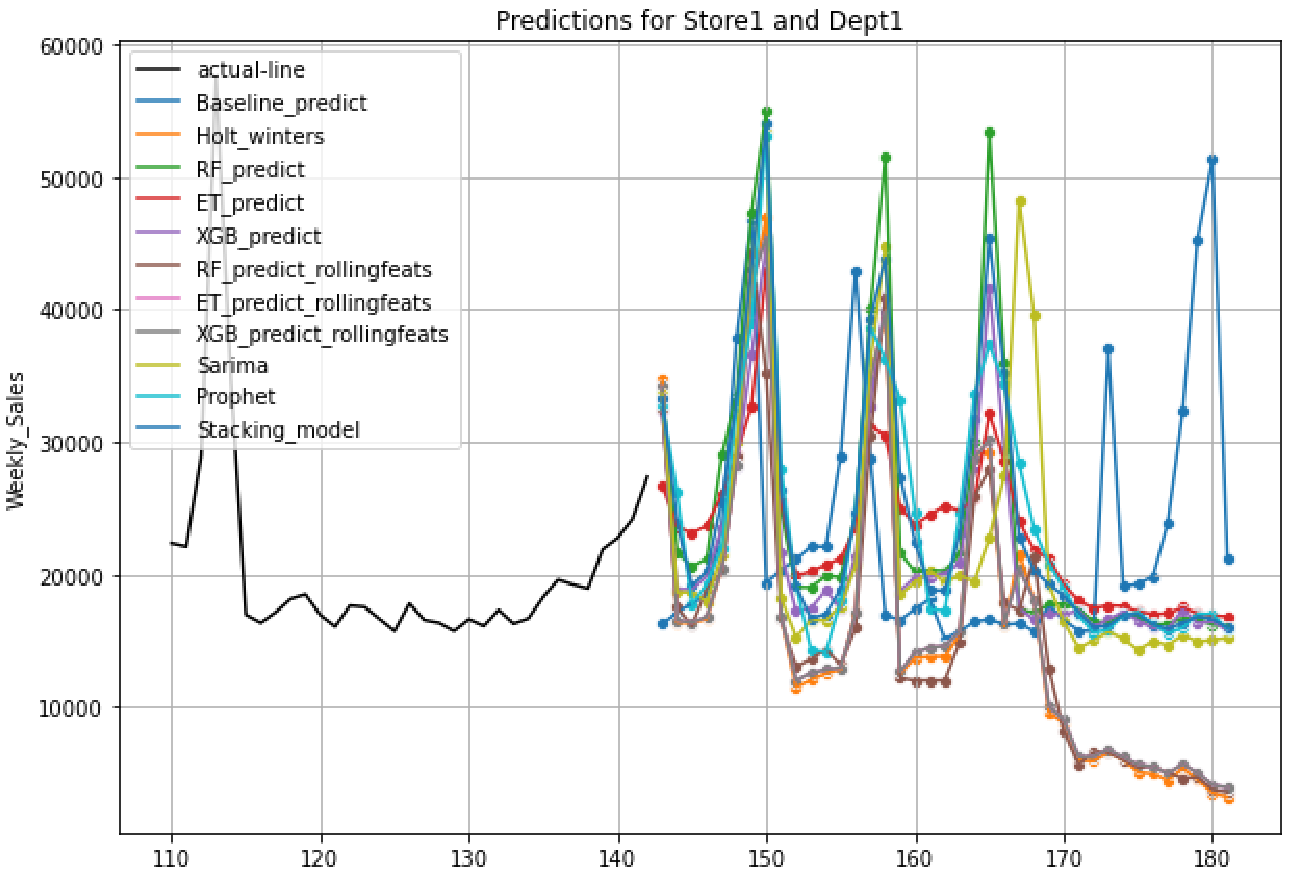
9.Plot model predictions :

In [300]:

```
fig = plt.figure(figsize=(10,7))
sns.lineplot(x=np.arange(110,143),y=y_train[110:143],label='actual-line',color='black')

sns.lineplot(x=np.arange(143,143+39),y=baseline_predict_te[:39],label='Baseline_predict')
sns.scatterplot(x=np.arange(143,143+39),y=baseline_predict_te[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=holts_test_predict[:39],label='Holt_winters')
sns.scatterplot(x=np.arange(143,143+39),y=holts_test_predict[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=rf_test_predict[:39],label='RF_predict')
sns.scatterplot(x=np.arange(143,143+39),y=rf_test_predict[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=et_test_predict[:39],label='ET_predict')
sns.scatterplot(x=np.arange(143,143+39),y=et_test_predict[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=xg_test_predict[:39],label='XGB_predict')
sns.scatterplot(x=np.arange(143,143+39),y=xg_test_predict[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=rf_test_predict2[:39],label='RF_predict_rollingfeats')
sns.scatterplot(x=np.arange(143,143+39),y=rf_test_predict2[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=et_test_predict2[:39],label='ET_predict_rollingfeats')
sns.scatterplot(x=np.arange(143,143+39),y=et_test_predict2[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=et_test_predict2[:39],label='XGB_predict_rollingfeats')
sns.scatterplot(x=np.arange(143,143+39),y=et_test_predict2[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=sarima_test_pred[:39],label='Sarima')
sns.scatterplot(x=np.arange(143,143+39),y=sarima_test_pred[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=prophet_test_pred[:39],label='Prophet')
sns.scatterplot(x=np.arange(143,143+39),y=prophet_test_pred[:39],marker='o')

sns.lineplot(x=np.arange(143,143+39),y=stacking_predictions_test[:39],label='Stacking_model')
sns.scatterplot(x=np.arange(143,143+39),y=stacking_predictions_test[:39],marker='o')

plt.grid()
plt.title('Predictions for Store1 and Dept1')
```

Out[300]:

```
Text(0.5, 1.0, 'Predictions for Store1 and Dept1')
```



- Observations :
  - We can see that all model predictions seems overlapping,and it is able to capture the spike in sales well during the holiday weeks.
  - models like Holt winters,RF_predict_rollingefeats,et_predict_rollingefeats xg_predict_rollingefeats shows a downward prediction as we go further in week.And does not follow a steady trend like it's past weeks.
  - However other good predictor models like (rf,prophet,extrtrees,xgb) follws a constant path.

#Submission

In [290]:

```
def apply(store,dept,date):
    return '_'.join([str(store),str(dept),str(date)[:-9]])

kaggle_df = pd.DataFrame()
kaggle_df['kaggle_ids'] = x_test.apply(lambda x : apply(x['Store'],x['Dept'],x['Date']),axis=1)
kaggle_df['baseline_predict'] = baseline_predict_te
kaggle_df['holts_predict'] = holts_test_predict
kaggle_df['rf_predict'] = rf_test_predict
kaggle_df['et_predict'] = et_test_predict
kaggle_df['xg_predict'] = xg_test_predict
kaggle_df['rf_predict2'] = rf_test_predict2
kaggle_df['et_predict2'] = et_test_predict2
kaggle_df['xg_predict2'] = xg_test_predict2
kaggle_df['sarima_predict'] = sarima_test_pred
kaggle_df['prophet_predict'] = prophet_test_pred
kaggle_df['stacking_predict'] = stacking_predictions_test
kaggle_df.to_csv('kaggle_submission.csv',index=False)
```

In [291]:

```
kaggle_df.to_csv('kaggle_submission.csv',index=False)
```

#some-references

https://www.google.com/url?q=https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/&usg=AOvVaw1tL-6yVl41xRCj0oFtwn_I (https://www.google.com/url?q=https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/&usg=AOvVaw1tL-6yVl41xRCj0oFtwn_I) https://www.google.com/url?q=https://towardsdatascience.com/time-series-forecasting-with-a-sarima-model-db051b7ae459&usg=AOvVaw0z333DvF58iNjHFxGsuSdk (https://www.google.com/url?q=https://towardsdatascience.com/time-series-forecasting-with-a-sarima-model-db051b7ae459&usg=AOvVaw0z333DvF58iNjHFxGsuSdk) https://www.youtube.com/watch?v=sCl6CXZ2xBg (https://www.youtube.com/watch?v=sCl6CXZ2xBg) https://www.google.com/url?q=https://otexts.com/fpp2/seasonal-arima.html&usg=AOvVaw3FfB8gnRJVKrc54po_hDPJ (https://www.google.com/url?q=https://otexts.com/fpp2/seasonal-arima.html&usg=AOvVaw3FfB8gnRJVKrc54po_hDPJ) https://www.youtube.com/watch?v=pmZNQoUfp3Y (https://www.youtube.com/watch?v=pmZNQoUfp3Y) https://www.youtube.com/watch?v=qSz9xfnXSwg (https://www.youtube.com/watch?v=qSz9xfnXSwg) https://www.google.com/url?q=https://www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/&usg=AOvVaw2pebFcLpxVLyY0ude6Su7Z (https://www.google.com/url?q=https://www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/&usg=AOvVaw2pebFcLpxVLyY0ude6Su7Z)

In [ ]:

```
1
```