In [145]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler as sc
from sklearn.model_selection import cross_validate as cv
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
file= r"C:\Users\Madhujit\Desktop\train.csv"
df=pd.read_csv(file)
```

In [3]:

```
x=df.drop(["num_orders"],axis=1)
y=df["num_orders"]
```

In [4]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)
```

In [5]:

```
reg=RandomForestRegressor()
reg.fit(x_train,y_train)
```

Out[5]:

```
array([0.05587578, 0.07525422, 0.14967843, 0.22338599, 0.22431454,
       0.12980547, 0.0538618 , 0.08782376])
```

In [8]:

```
dict(zip(reg.feature_importances_,x_train.columns))
```

Out[8]:

```
{0.055875783328167686: 'id',
 0.07525422058954744: 'week',
 0.14967843018650095: 'center_id',
 0.22338598887924882: 'meal_id',
 0.22431453969438275: 'checkout_price',
 0.12980547075887186: 'base_price',
 0.05386180196490727: 'emailer_for_promotion',
 0.08782376459837314: 'homepage_featured'}
```

In [140]:

```
x1=df.drop(["num_orders","id","week","emailer_for_promotion","homepage_featured"],axis=1)
y1=df["num_orders"]
```

In [141]:

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=0.33,random_state=42)
```

In [147]:

```
standard_scaler=StandardScaler()
```

In [148]:

```
scale_x=standard_scaler.fit_transform(x_train)
scale_x_test=standard_scaler.transform(x_test)
```

In [149]:

```
reg=RandomForestRegressor(max_depth=10, min_samples_leaf=4, min_samples_split=4,
                      n_estimators=80)
```

```
In [150]: model=reg.fit(scale_x,y_train)
```

```
In [151]: pred=model.predict(scale_x_test)
```

```
In [152]: y_pred=pd.DataFrame(pred,columns=["Forecast"])
```

```
In [153]: y_test_data=pd.DataFrame(y_test)
          y_test_data=y_test_data.reset_index()
```

```
In [154]: pred_data=pd.concat([y_test_data,y_pred],axis=1)
```

```
In [155]: pred_data
```

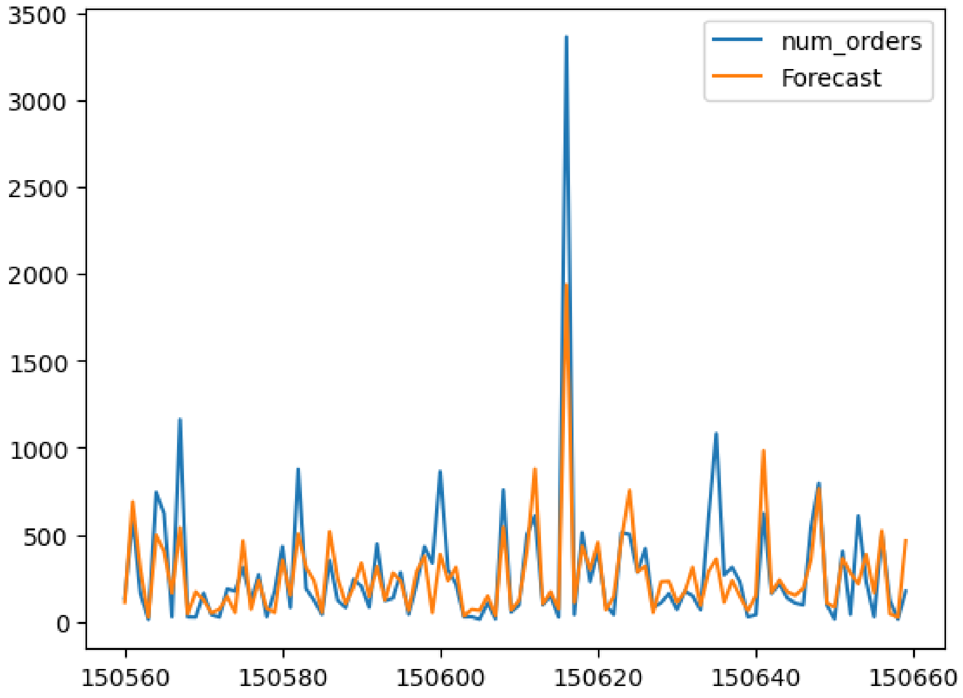
Out[155]:

	index	num_orders	Forecast
0	203536	28	133.056643
1	301801	176	298.633444
2	254032	391	151.316024
3	339158	14	52.996730
4	3203	405	335.674607
...	...	...	...
150656	418537	500	522.204712
150657	453334	122	45.757286
150658	126678	14	25.793546
150659	335876	175	465.969535
150660	350694	109	111.133084

150661 rows × 3 columns

```
In [156]: pred_data[["num_orders","Forecast"]][150560:150660].plot()
```

```
Out[156]: <Axes: >
```



In [157]:
```
mean_absolute_error(y_test,pred)
```

Out[157]:
```
124.11255495392192
```

In [158]:
```
errors = np.abs(pred - y_test)
std_dev = np.std(errors)
```

Out[158]:
```
214.51008362035526
```

In [159]:
```
std_dev
```

Out[159]:
```
214.51008362035526
```

In [68]:
```
param={"n_estimators":[10,20,50,70,80,90,100],"criterion":["squared_error"],"max_depth":[2,4
       "min_samples_split":[2,3,4,5],"min_samples_leaf":[1,2,3,4,5]}
```

In [70]:
```
reg_model=RandomizedSearchCV(reg,param,n_iter=5,scoring="neg_mean_absolute_error",cv=5)
```

In [71]:
```
reg_model.fit(scale_x,y_train)
```

Out[71]:
```
▸        RandomizedSearchCV
▸ estimator: RandomForestRegressor
     ▸ RandomForestRegressor
```

In [72]:
```
reg_model.best_estimator_
```

Out[72]:
```
▾                       RandomForestRegressor
RandomForestRegressor(max_depth=10, min_samples_leaf=4, min_samples_split=4,
                      n_estimators=80)
```

In [163]:
```
file= r"C:\Users\Madhujit\Desktop\test.csv"
test=pd.read_csv(file)
```

In [165]:
```
test_df=test.drop(["id","week","emailer_for_promotion","homepage_featured"],axis=1)
```

In [166]:
```
test_df
```

Out[166]:

	center_id	meal_id	checkout_price	base_price
0	55	1885	158.11	159.11
1	55	1993	160.11	159.11
2	55	2539	157.14	159.14
3	55	2631	162.02	162.02
4	55	1248	163.93	163.93
...	...	...	...	...
32568	61	1543	482.09	484.09
32569	61	2304	483.09	483.09
32570	61	2664	322.07	323.07
32571	61	2569	322.07	323.07
32572	61	2490	276.45	276.45

32573 rows × 4 columns

In [167]:
```
sc=StandardScaler()
test_scaler=sc.fit_transform(test_df)
```

In [168]:
```
prediction=pd.DataFrame(model.predict(test_scaler),columns=["forecast"])
```

In [170]:
```
prediction[100:]
```

Out[170]:
```
          forecast
  100   907.915911
  101    66.041850
  102   125.752494
  103   688.022376
  104   889.139563
  ...          ...
32568    62.290860
32569    52.996730
32570   316.194832
32571   271.654174
32572   267.113057

32473 rows × 1 columns
```

In [171]:
```
from lightgbm import LGBMRegressor
```

In [178]:
```
lgb=LGBMRegressor()
```

In [173]:
```
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
```

In [174]:
```
param_dist = {
    'num_leaves': sp_randint(6, 50),  # Number of leaves in each tree
    'max_depth': sp_randint(3, 15),  # Maximum depth of each tree
    'learning_rate': sp_uniform(loc=0.01, scale=0.2),  # Learning rate
    'n_estimators': sp_randint(50,100),  # Number of boosting rounds
    'min_child_samples': sp_randint(10, 200),  # Minimum number of samples per leaf
    'subsample': sp_uniform(loc=0.5, scale=0.5),  # Subsample ratio of the training instance
    'colsample_bytree': sp_uniform(loc=0.5, scale=0.5)}
```

In [179]:
```
reg_model2=RandomizedSearchCV(lgb,param,n_iter=5,scoring="neg_mean_absolute_error",cv=5)
```

In [180]:

```
reg_model2.fit(scale_x,y_train)
```

```
[LightGBM] [Warning] Unknown parameter: min_samples_split
[LightGBM] [Warning] min_data_in_leaf is set with min_child_samples=20, will be overridde
n by min_samples_leaf=4. Current value: min_data_in_leaf=4
[LightGBM] [Warning] Unknown parameter: criterion
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^
max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] Unknown parameter: min_samples_split
[LightGBM] [Warning] min_data_in_leaf is set with min_child_samples=20, will be overridde
n by min_samples_leaf=4. Current value: min_data_in_leaf=4
[LightGBM] [Warning] Unknown parameter: criterion
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^
max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] Unknown parameter: min_samples_split
[LightGBM] [Warning] min_data_in_leaf is set with min_child_samples=20, will be overridde
n by min_samples_leaf=4. Current value: min_data_in_leaf=4
[LightGBM] [Warning] Unknown parameter: criterion
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^
max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.0
07248 seconds
```

In [182]:

```
model2=lgb.fit(scale_x,y_train)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006
359 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 637
[LightGBM] [Info] Number of data points in the train set: 305887, number of used features:
4
[LightGBM] [Info] Start training from score 262.125193
```

In [183]:

```
lgb_pred=model2.predict(scale_x_test)
```

In [185]:

```
mean_absolute_error(y_test,lgb_pred)
```

Out[185]:

```
112.87893559651542
```

In [190]:

```
Final_pred=pd.DataFrame(model2.predict(test_scaler),columns=["Forecast"])
```

In [191]:

```
Final_pred
```

Out[191]:

	Forecast
0	109.771203
1	504.967116
2	131.310140
3	6.613130
4	-14.722792
...	...
32568	66.720441
32569	180.686709
32570	345.146461
32571	302.608089
32572	275.182451

32573 rows × 1 columns

In [ ]: