ds_2

March 29, 2022

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

0.1 Load data

[ ]:
```
train_data = pd.read_csv(r'../datasets/train_data.csv')
validation_data = pd.read_csv(r'../datasets/validation_data.csv')
test_data = pd.read_csv(r'../datasets/test_data.csv')
```

[ ]:
```
train_data.head()
```

[ ]:

	CategoryCode	ItemCode	DateID	DailySales
0	category_2	117610	11/6/2021	7
1	category_4	836584	11/18/2021	16
2	category_1	370195	1/24/2022	6
3	category_2	172582	10/30/2021	5
4	category_2	1006009	10/30/2021	5

[ ]:
```
validation_data.head()
```

[ ]:

	CategoryCode	ItemCode	Week	WeeklySales
0	category_2	1044502	w1	11
1	category_2	1105009	w1	11
2	category_2	913561	w4	5
3	category_1	1048975	w4	30
4	category_1	17287	w2	60

[ ]:
```
test_data.head()
```

[ ]:

	CategoryCode	ItemCode	Week	PredictedSales
0	category_1	43738	w4	NaN
1	category_2	1006090	w1	NaN
2	category_2	1076929	w4	NaN

```
3   category_1   1081321   w3             NaN
4   category_2    216151   w4             NaN
```

0.2 Preprocess

[ ]:
```
category_codes = np.unique([*train_data['CategoryCode'].values,␣
↪*validation_data['CategoryCode'].values, *test_data['CategoryCode'].values])
category_map = {}

for i in range(len(category_codes)):
    category_map[category_codes[i]] = i
```

[ ]:
```
item_codes = np.unique([*train_data['ItemCode'].values, *test_data['ItemCode'].
↪values, *validation_data['ItemCode'].values])
item_map = {}

for i in range(len(item_codes)):
    item_map[item_codes[i]] = i
```

[ ]:
```
def string_to_date(d):
    return datetime(int(d.split('/')[2]), int(d.split('/')[0]), int(d.split('/
↪')[1]))
```

[ ]:
```
def get_year(date):
    return date.year

def get_month(date):
    return date.month

def get_annual_week_id(date):
    return pd.Period(date).week

def get_monthly_week_id(date):
    first_day = date.replace(day=1)

    dom = date.day
    adjusted_dom = dom + first_day.weekday()

    return int(ceil(adjusted_dom/7.0))

def get_category_id(id):
    return category_map[id]

def get_item_code_id(id):
    return item_map[id]
```

[ ]:
```
def week_to_weekid(week):
    if (week == "w1"):
        return get_annual_week_id(('02/14/2022'))
    if (week == "w2"):
        return get_annual_week_id(('02/21/2022'))
    if (week == "w3"):
        return get_annual_week_id(('02/28/2022'))
    if (week == "w4"):
        return get_annual_week_id(('03/07/2022'))

weeks = np.unique(test_data['Week'].values)
week_id_map = {}

for i in range(len(weeks)):
    week_id_map[weeks[i]] = week_to_weekid(weeks[i])
```

[ ]:
```
def get_week_id_from_map(week):
    return week_id_map[week]
```

Preprocess train data

[ ]:
```
train_data['WeekID'] = train_data['DateID'].apply(get_annual_week_id)
train_data['DateID'] = train_data['DateID'].apply(string_to_date)
train_data['DailySales'] = train_data['DailySales']
train_data['Year'] = train_data['DateID'].apply(get_year)
train_data['ItemCode'] = train_data['ItemCode'].apply(get_item_code_id)
train_data['CategoryCode'] = train_data['CategoryCode'].apply(get_category_id)
```

[ ]:
```
train_grp_by_week = train_data.groupby(['WeekID', 'CategoryCode', 'ItemCode',␣
↪'Year'])['DailySales'].sum().reset_index()
train_grp_by_week = train_grp_by_week.rename(columns = {'DailySales':␣
↪'WeeklySales'}, inplace = False)
train_grp_by_week.head()
```

[ ]:

	WeekID	CategoryCode	ItemCode	Year	WeeklySales
0	1	0	0	2022	83
1	1	0	1	2022	66
2	1	0	5	2022	21
3	1	0	6	2022	621
4	1	0	10	2022	31

[ ]:
```
train_grp_by_week.describe()
```

[ ]:

	WeekID	CategoryCode	ItemCode	Year	WeeklySales
count	3952.000000	3952.000000	3952.000000	3952.000000	3952.000000
mean	33.918775	0.919787	95.639676	2021.329200	37.058957
std	19.510868	0.869572	55.659723	0.469982	72.419014

```
min       1.000000      0.000000      0.000000  2021.000000     1.000000
25%       6.000000      0.000000     47.750000  2021.000000     9.000000
50%      43.000000      1.000000     96.000000  2021.000000    17.000000
75%      48.000000      1.000000    143.000000  2022.000000    36.250000
max      52.000000      3.000000    193.000000  2022.000000   909.000000
```

Preprocess validation data

[ ]:
```
validation_data['ItemCode'] = validation_data['ItemCode'].
 ↪apply(get_item_code_id)
validation_data['CategoryCode'] = validation_data['CategoryCode'].
 ↪apply(get_category_id)

validation_data['WeekID'] = validation_data['Week'].apply(get_week_id_from_map)
```

[ ]:
```
val_data_grp_by_week = validation_data.groupby(['CategoryCode', 'ItemCode',␣
 ↪'WeekID'])['WeeklySales'].sum().reset_index()
val_data_grp_by_week.head()
```

[ ]:
```
   CategoryCode  ItemCode  WeekID  WeeklySales
0             0         0       7           25
1             0         0       8           69
2             0         0       9          120
3             0         0      10           69
4             0         1       7            7
```

[ ]:
```
val_data_grp_by_week.describe()
```

[ ]:
```
       CategoryCode    ItemCode      WeekID  WeeklySales
count    370.000000  370.000000  370.000000   370.000000
mean       0.848649   94.956757    8.516216    42.759459
std        0.774814   55.496364    1.121848    72.832726
min        0.000000    0.000000    7.000000     1.000000
25%        0.000000   50.250000    8.000000    11.000000
50%        1.000000   94.500000    9.000000    22.000000
75%        1.000000  142.000000   10.000000    48.000000
max        3.000000  191.000000   10.000000   771.000000
```

Preprocess test data

[ ]:
```
test_data['ItemCode'] = test_data['ItemCode'].apply(get_item_code_id)
test_data['CategoryCode'] = test_data['CategoryCode'].apply(get_category_id)

test_data['WeekID'] = test_data['Week'].apply(get_week_id_from_map)
```

[ ]:
```
test_data_grp_by_week = test_data.drop(['Week', 'PredictedSales'],␣
 ↪axis='columns')
```

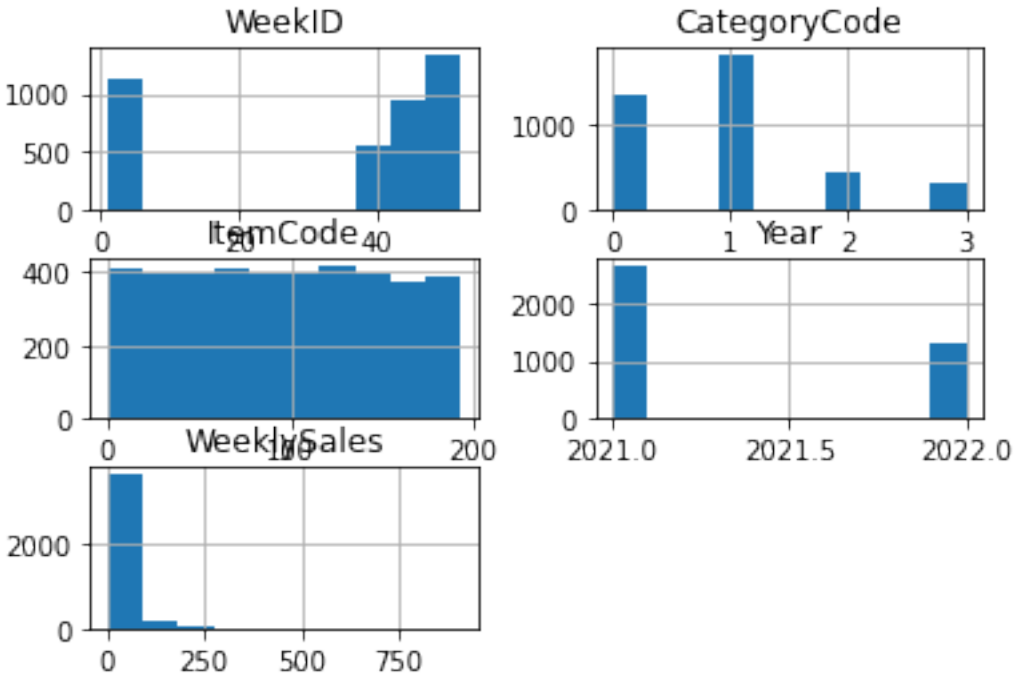
```
test_data_grp_by_week.head()
```

[ ]:

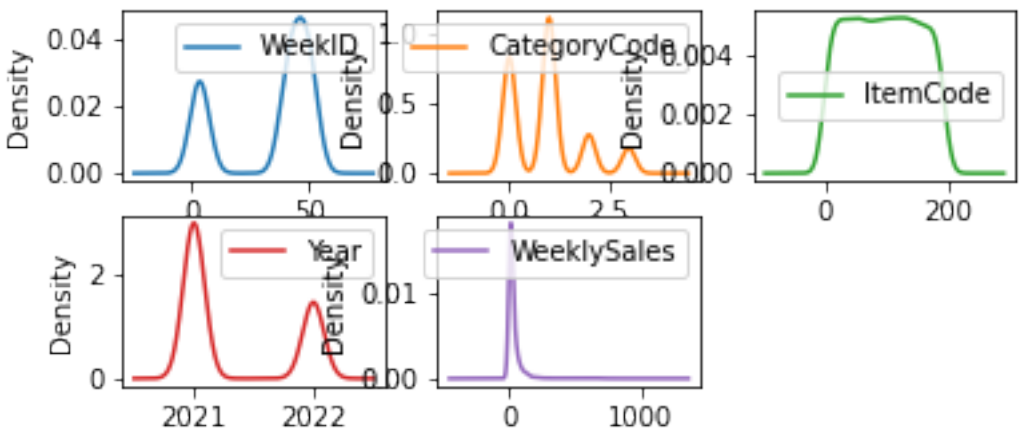
	CategoryCode	ItemCode	WeekID
0	0	27	10
1	1	110	7
2	1	160	10
3	0	167	9
4	1	68	10

0.3 Visualizations

[ ]:
```
train_grp_by_week.hist()
plt.show()
```

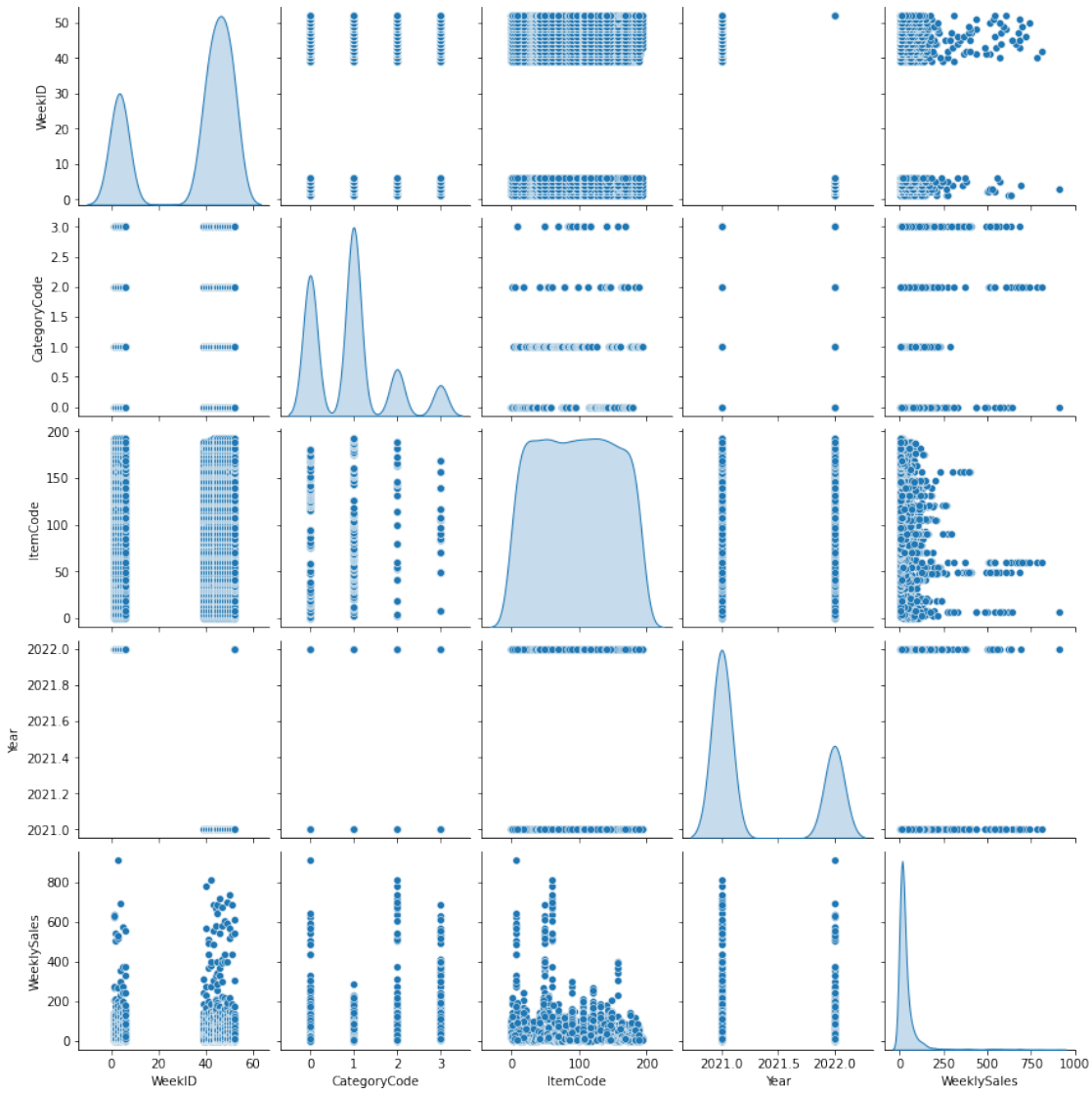


[ ]:
```
train_grp_by_week.plot(kind='density', subplots=True, sharex=False, layout=(3,␣
↪3))
plt.show()
```



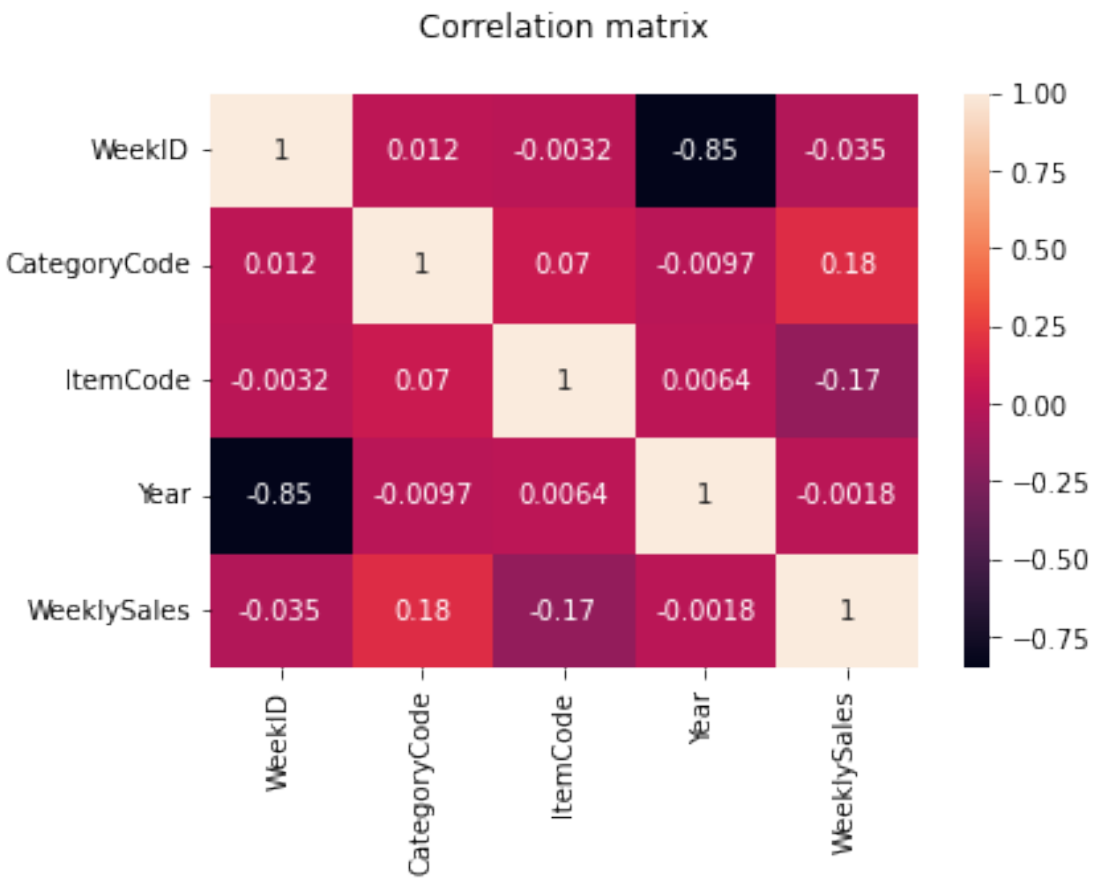
[ ]:
```
sns.pairplot(data=train_grp_by_week, diag_kind='kde')
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f9558723460>
```

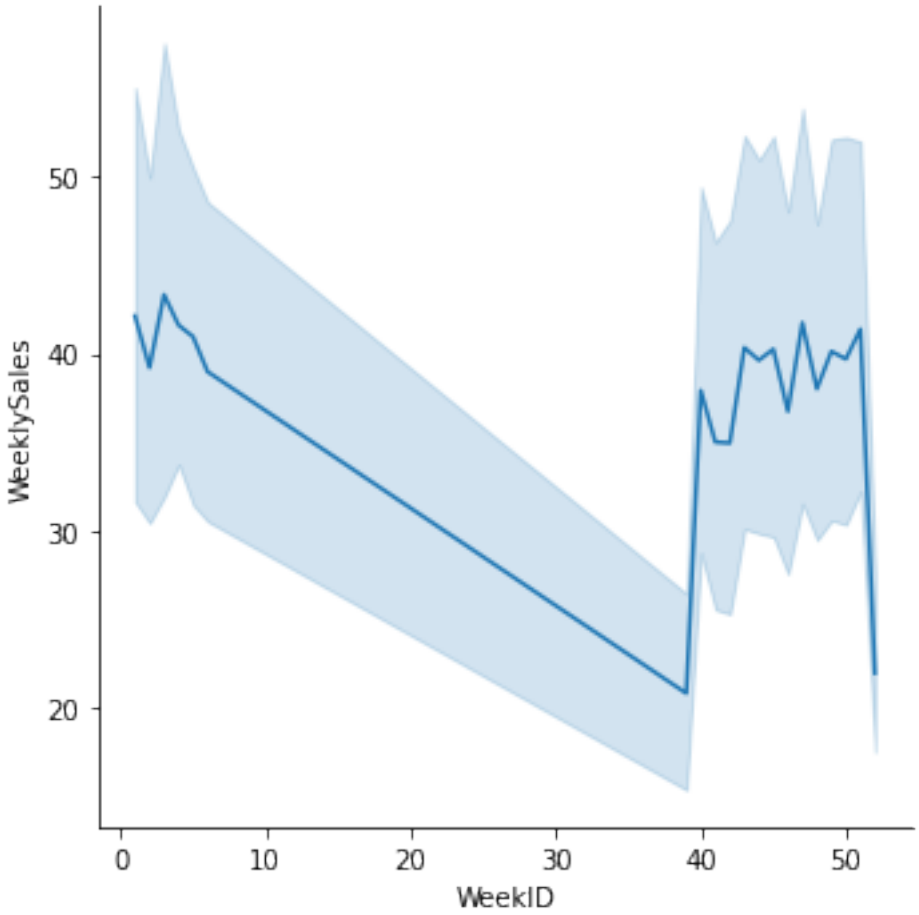


[ ]:
```
hm = sns.heatmap(train_grp_by_week.corr(), annot = True)
hm.set(title = "Correlation matrix\n")
```

[ ]:
```
[Text(0.5, 1.0, 'Correlation matrix\n')]
```



[ ]:
```
sns.relplot(x=train_grp_by_week['WeekID'], y=train_grp_by_week['WeeklySales'],␣
↪kind='line')
plt.show()
```



0.4 Normalize data

[ ]:
```
from sklearn.preprocessing import Normalizer
```

[ ]:
```
def normalize(arr):
    scalar = Normalizer().fit([arr])
    return scalar.transform([arr])[0]
```

[ ]:
```
week_values = [*train_grp_by_week['WeekID'].values,␣
↪*test_data_grp_by_week['WeekID'].values, *val_data_grp_by_week['WeekID'].
↪values]
category_codes = [*train_grp_by_week['CategoryCode'].values,␣
↪*test_data_grp_by_week['CategoryCode'].values,␣
↪*val_data_grp_by_week['CategoryCode'].values]
item_codes = [*train_grp_by_week['ItemCode'].values,␣
↪*test_data_grp_by_week['ItemCode'].values, *val_data_grp_by_week['ItemCode'].
↪values]
```

[ ]:
```
normalized_week_ids = normalize(week_values)
normalized_category_codes = normalize(category_codes)
normalized_item_codes = normalize(item_codes)
```

[ ]:
```
train_len = len(train_grp_by_week['WeekID'].values)
test_len = len(test_data_grp_by_week['WeekID'].values)
val_len = len(val_data_grp_by_week['CategoryCode'].values)

print(train_len, test_len, val_len)
```

```
3952 377 370
```

[ ]:
```
train_x = pd.DataFrame()
train_x['week_id_of_year'] = normalized_week_ids[:train_len]
train_x['category_code'] = normalized_category_codes[:train_len]
train_x['item_code'] = normalized_item_codes[:train_len]

train_x.shape
```

[ ]:
```
(3952, 3)
```

[ ]:
```
test_x = pd.DataFrame()
test_x['week_id_of_year'] = normalized_week_ids[train_len:(test_len +␣
↪train_len)]
test_x['category_code'] = normalized_category_codes[train_len:(test_len +␣
↪train_len)]
test_x['item_code'] = normalized_item_codes[train_len:(test_len + train_len)]

test_x.shape
```

[ ]:
```
(377, 3)
```

[ ]:
```
val_x = pd.DataFrame()
val_x['week_id_of_year'] = normalized_week_ids[(test_len + train_len): ]
val_x['category_code'] = normalized_category_codes[(test_len + train_len): ]
val_x['item_code'] = normalized_item_codes[(test_len + train_len): ]

val_x.shape
```

[ ]:
```
(370, 3)
```

[ ]:
```
train_y = train_grp_by_week['WeeklySales']
val_y = val_data_grp_by_week['WeeklySales']
```

[ ]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(train_x, train_y,␣
↪test_size=0.33, shuffle=True)
print(X_train.shape, Y_train.shape)
```

```
(2647, 3) (2647,)
```

[ ]:
```
# from imblearn.over_sampling import RandomOverSampler

# def getBalancedData(X, Y):
#     sm = RandomOverSampler(random_state=42)
#     return sm.fit_resample(X, Y)

# balanced_X, balanced_Y = getBalancedData(X_train, Y_train)
# print(balanced_X.shape, balanced_Y.shape)
```

0.5 Metrics (Explained variance score)

[ ]:
```
from sklearn.metrics import explained_variance_score

def get_ev_score(true_values, predicted_values):
    return explained_variance_score(true_values, predicted_values)
```

0.6 Random Forest Regressor

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
random_forest_reg = RandomForestRegressor()
random_forest_reg.fit(X_train, Y_train)

random_forest_reg.score(X_test, Y_test)
random_forest_predictions = random_forest_reg.predict(X_test)
```

[ ]:
```
# uniform avarage score
random_forest_reg.score(val_x, val_y)
```

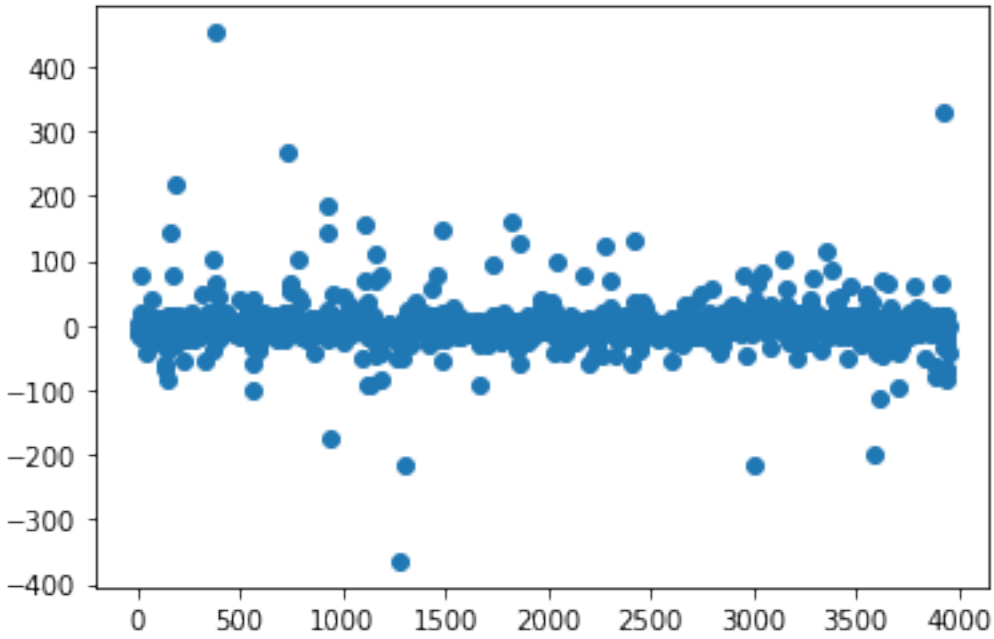
[ ]:
```
0.756871980523901
```

[ ]:
```
# explained variance score
get_ev_score(Y_test, random_forest_predictions)
```

[ ]:
```
0.7795390487797285
```

[ ]:
```
plt.plot(Y_test - random_forest_predictions, marker='o', linestyle='')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f95550166d0>]
```



0.7 Extra Trees Regressor

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

extra_trees_reg = ExtraTreesRegressor()
extra_trees_reg.fit(X_train, Y_train)

extra_trees_reg.score(X_test, Y_test)
extra_trees_predictions = extra_trees_reg.predict(X_test)
```

[ ]:
```
# uniform avarage score
extra_trees_reg.score(val_x, val_y)
```

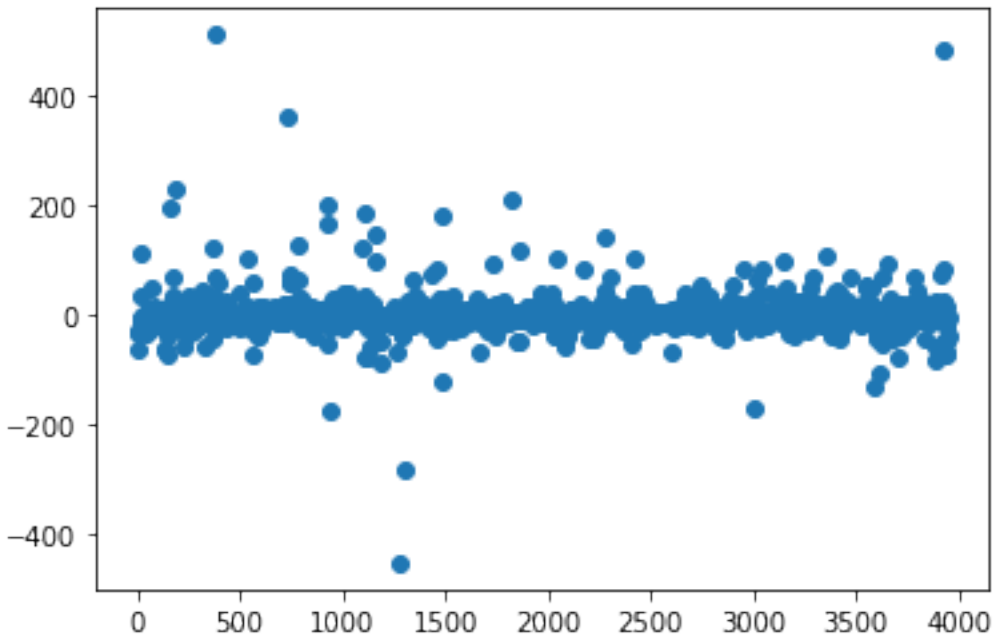
[ ]:
```
0.7000322220704596
```

[ ]:
```
# explained variance score
get_ev_score(Y_test, extra_trees_predictions)
```

[ ]:
```
0.7078502351293492
```

[ ]:
```
plt.plot(Y_test - extra_trees_predictions, marker='o', linestyle='')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f9554f94820>]
```



0.8 Voting Regressor

[ ]:
```
from sklearn.ensemble import VotingRegressor

r1 = RandomForestRegressor(n_estimators=100, random_state=10,␣
↪criterion='absolute_error')
r2 = RandomForestRegressor(n_estimators=10, random_state=1)

voting_reg = VotingRegressor([('rf', r1), ('et', r2)]).fit(X_train, Y_train)
voting_reg_predictions = voting_reg.predict(X_test)
```

[ ]:
```
# uniform avarage score
voting_reg.score(X_test, Y_test)
```

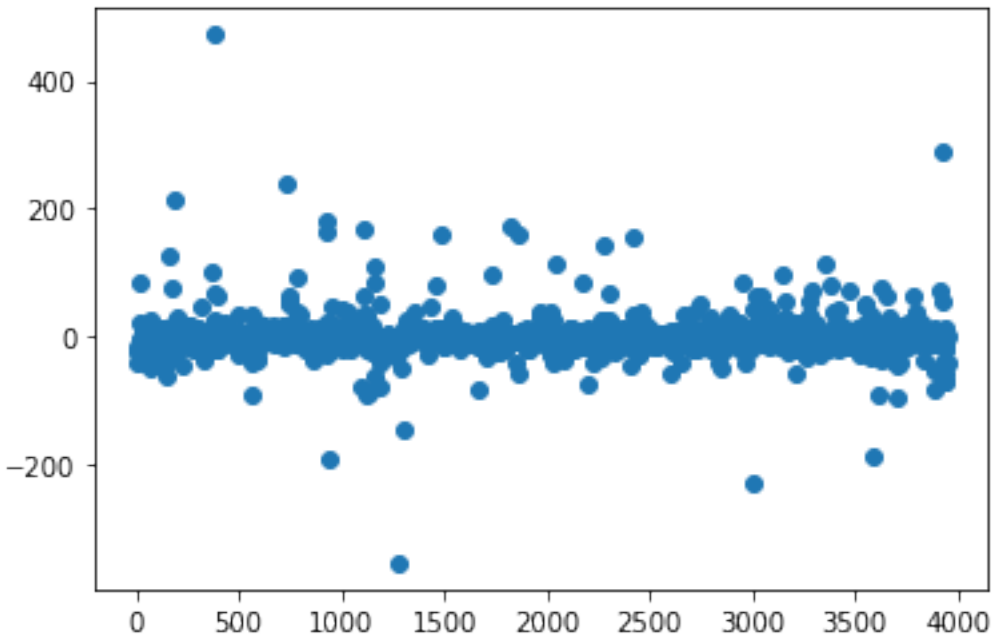
[ ]:
```
0.7851608081629777
```

[ ]:
```
# explained variance score
get_ev_score(Y_test, voting_reg_predictions)
```

[ ]:
```
0.7851756554190874
```

[ ]:
```
plt.plot(Y_test - voting_reg_predictions, marker='o', linestyle='')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f9554f88fa0>]
```



[ ]:
```
from sklearn.ensemble import GradientBoostingRegressor,␣
↪HistGradientBoostingRegressor, AdaBoostRegressor, BaggingRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.preprocessing import QuantileTransformer

# generic function to fit model and return metrics for every algorithm


def boost_models(x):
    # transforming target variable through quantile transformer
    regr_trans = TransformedTargetRegressor(
        regressor=x,␣
↪transformer=QuantileTransformer(output_distribution='normal'))
    regr_trans.fit(X_train, Y_train)
    yhat = regr_trans.predict(X_test)
    algoname = x.__class__.__name__
    return algoname, get_ev_score(Y_test, yhat)


r1 = RandomForestRegressor(
    n_estimators=100, random_state=10, criterion='absolute_error')
r2 = RandomForestRegressor(n_estimators=10, random_state=1)

vr = VotingRegressor([('rf', r1), ('et', r2)])
```

```
algo = [GradientBoostingRegressor(), vr, RandomForestRegressor(),␣
 ↪ExtraTreesRegressor(), HistGradientBoostingRegressor(), AdaBoostRegressor(),␣
 ↪BaggingRegressor()]
score = []
for a in algo:
    score.append(boost_models(a))

# Collate all scores in a table
pd.DataFrame(score, columns=['Model', 'Score'])
```

[ ]:

	Model	Score
0	GradientBoostingRegressor	0.429291
1	VotingRegressor	0.755375
2	RandomForestRegressor	0.765854
3	ExtraTreesRegressor	0.623977
4	HistGradientBoostingRegressor	0.745807
5	AdaBoostRegressor	0.106199
6	BaggingRegressor	0.736430

[ ]:
```
# from sklearn.model_selection import GridSearchCV

# param_grid = {'loss': ['squared_error', 'absolute_error', 'poisson'],
#               'max_depth': [7, 8],
#               'max_itr': [100, 80, 60, 55, 51, 45],
#               'learning_rate' :[0.26, 0.25, 0.2, 0.1, 0.15, 0.3, 0.16, 0.13]
#              }

r1 = RandomForestRegressor(
    n_estimators=100, random_state=10, criterion='absolute_error')
r2 = RandomForestRegressor(n_estimators=10, random_state=1)

vr = VotingRegressor([('rf', r1), ('et', r2)])

hist_regresor = RandomForestRegressor()
regr_trans = TransformedTargetRegressor(regressor=vr,␣
 ↪transformer=QuantileTransformer(output_distribution='normal'))

regr_trans.fit(X_train, Y_train)
hist_reg_predictions = regr_trans.predict(val_x)
```

[ ]:
```
get_ev_score(val_y, hist_reg_predictions)
```

[ ]:
```
0.7689616896121296
```

0.9 Most effective regressior identified (Voting regressor with 2 random forests)

[ ]:
```
test_data = pd.read_csv(r'../datasets/test_data.csv')
test_data['PredictedSales'] = regr_trans.predict(test_x)
test_data['PredictedSales'] = test_data['PredictedSales'].apply(lambda val:␣
↪round(val))
test_data['ItemCode'] = test_data['ItemCode'].apply(lambda val: str(val))

submission_df = pd.DataFrame()
submission_df['ID'] = test_data[['CategoryCode', 'ItemCode', 'Week']].
↪agg(lambda x: '_'.join(x.values), axis=1)
submission_df['WeeklySales'] = test_data['PredictedSales']
submission_df.head()
```

[ ]:

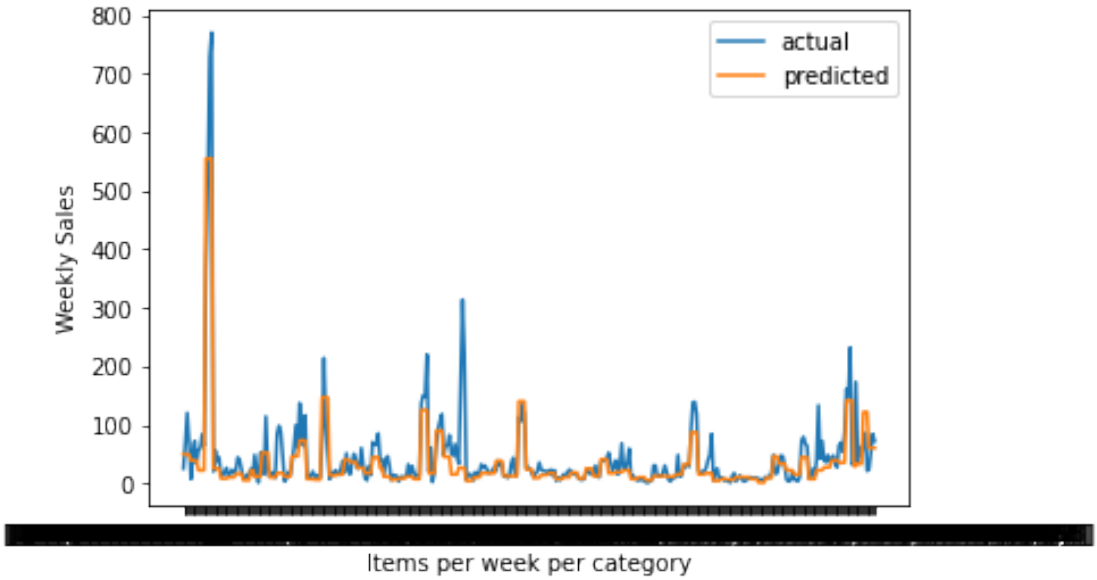
	ID	WeeklySales
0	category_1_43738_w4	24
1	category_2_1006090_w1	17
2	category_2_1076929_w4	6
3	category_1_1081321_w3	7
4	category_2_216151_w4	14

[ ]:
```
submission_df.to_csv('../results/test_predictions.csv', index=False)
```

1 Actual values vs predicted values (using voting regressor with two random forests) graph

[ ]:
```
plt.plot(val_x[['week_id_of_year', 'category_code', 'item_code']].agg(lambda x:␣
↪str(x.values), axis=1), val_y, label='actual')
plt.plot(val_x[['week_id_of_year', 'category_code', 'item_code']].agg(lambda x:␣
↪str(x.values), axis=1), hist_reg_predictions, label='predicted')
plt.legend()
plt.xlabel('Items per week per category')
plt.ylabel('Weekly Sales')
```

[ ]:
```
Text(0, 0.5, 'Weekly Sales')
```


800
700
600
500
400
300
200
100
0
Weekly Sales
actual
predicted
Items per week per category