TNSDC-NAAN MUDHALVAN(IBM)

COLLEGE CODE:1108

COLLEGE NAME: JAYA ENGINEERING COLLEGE

DOMAIN: APPLIED DATA SCIENCE

PROJECT: FUTURE SALES PREDICTION

PROBLEM STATEMENT

- **Importing libraries**
- **Versions of packages**
- **Python version**
- **Importing the data**
- **Handling the missing data**
- **Encoding the Categorical Data**
- **Splitting the dataset**
- **Feature scaling**

TEAM MEMBERS,

1.ThiruKarthikeyan A(B.Tech[IT])

2. Praveen kumar (B.Tech[IT])

3. Veluprasath (B.Tech[IT])

4.Vignesh (B.Tech[IT])

5. Abinesh M (B.Tech[IT])

Importing Libraries:

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

import gc
from itertools import product
import pickle
import time

from sklearn.preprocessing import LabelEncoder

%matplotlib inline
```

Versions of packages:

```
import pkg_resources
import types
def get_imports():
    for name, val in globals().items():
        if isinstance(val, types.ModuleType):
            # Split ensures you get root package,
            # not just imported function
            name = val.__name__.split(".")[0]

        elif isinstance(val, type):
            name = val.__module__.split(".")[0]

        # Some packages are weird and have different
        # imported names vs. system names
        if name == "PIL":
            name = "Pillow"
        elif name == "sklearn":
            name = "scikit-learn"

        yield name
imports = list(set(get_imports()))

requirements = []
for m in pkg_resources.working_set:
    if m.project_name in imports and m.project_name!="pip":
        requirements.append((m.project_name, m.version))

for r in requirements:
    print("{}=={}".format(*r))
```

Output:

```
seaborn==0.10.0
scikit-learn==0.23.2
pandas==1.1.2
numpy==1.18.5
matplotlib==3.2.1
```

Python version:

```
import sys
print(sys.version)
```

```
Output:
   3.7.6 | packaged by conda-forge | (default, Mar 23 2020, 23:03:20)
[GCC 7.3.0]
```

Reading in the data:

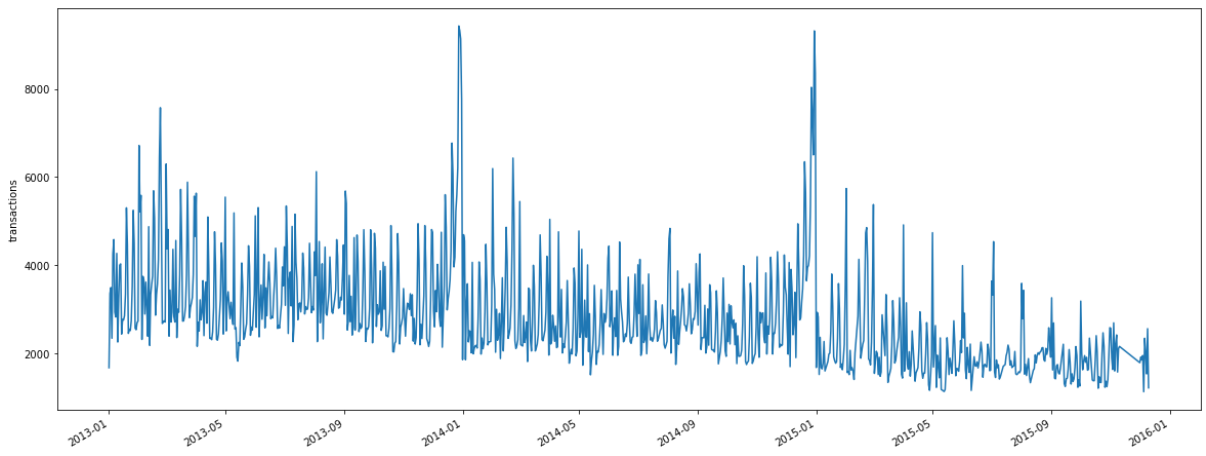
```
input_path = "../input/competitive-data-science-predict-future-sales/"
train = reduce_mem_usage(pd.read_csv(input_path + "sales_train.csv"))
```

Exploratory data analysis:

```
pd.to_datetime(train.date).value_counts().sort_index(ascending=False).p
lot(kind='line')
fig = plt.gcf()
fig.set_size_inches(20, 8)
plt.ylabel("transactions")
```

Output:

```
Text(0, 0.5, 'transactions')
```



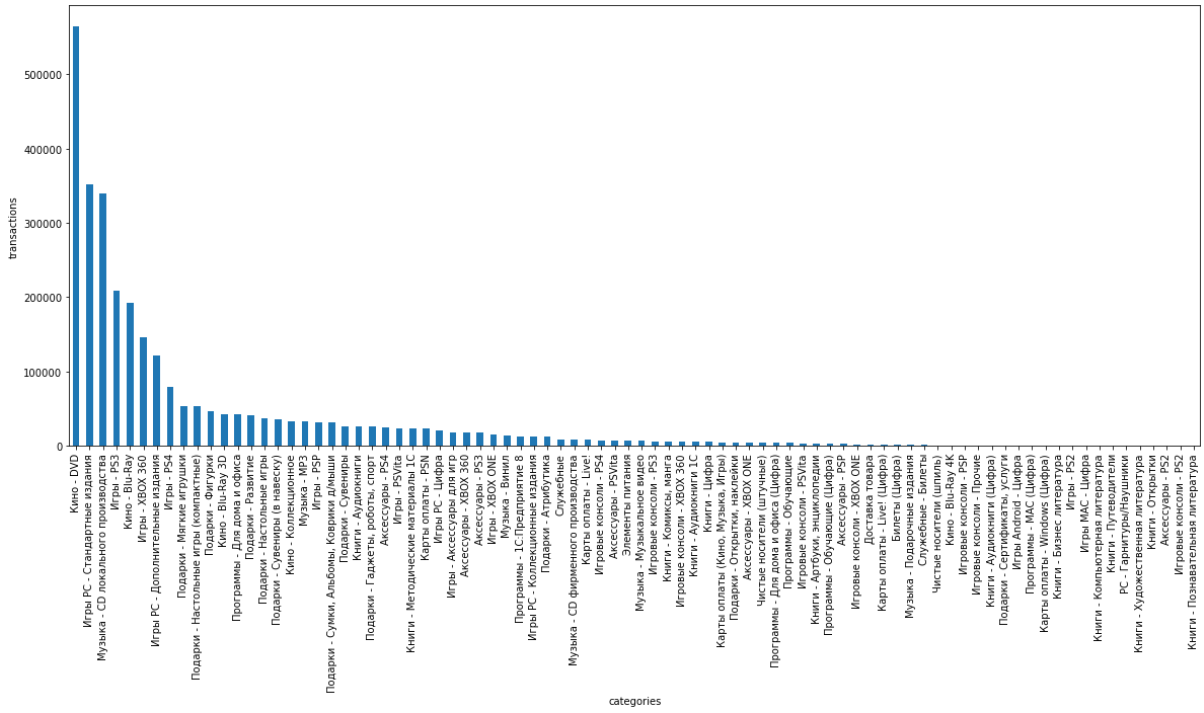
```
merged_train = pd.merge(train, items, on='item_id', how='inner')
merged_train = pd.merge(merged_train, cats, on='item_category_id', how=
'inner')
merged_train = pd.merge(merged_train, shops, on='shop_id', how='inner')
merged_train['revenue'] = merged_train['item_price'] *  merged_train['i
tem_cnt_day']

merged_train.item_category_name.value_counts().plot(kind='bar')
fig = plt.gcf()
fig.set_size_inches(20, 8)
plt.ylabel("transactions")
plt.xlabel("categories")
plt.show()

merged_train.groupby('shop_name')['revenue'].sum().sort_values(ascendin
g=False).plot(kind='bar')
fig = plt.gcf()
fig.set_size_inches(20, 8)
plt.ylabel("revenue")
plt.xlabel("shops")
plt.show()

del merged_train
gc.collect()
```

Output:



Output:
12392

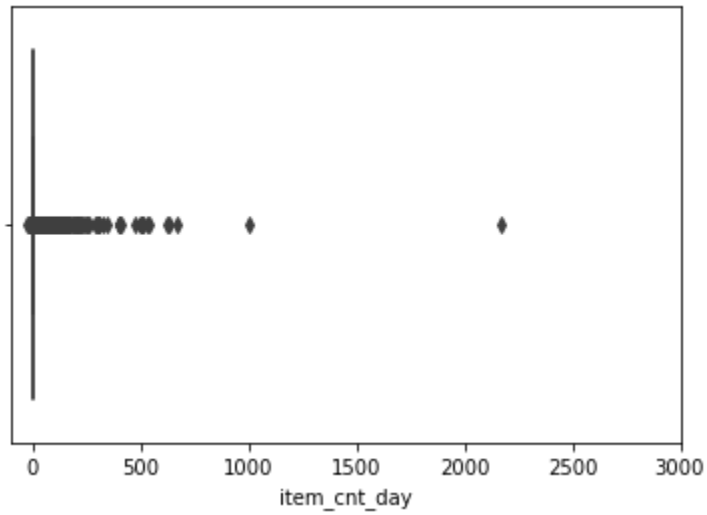
As we can see, there are a couple of outliers and negative item prices:

```
plt.xlim(-100, 3000)
sns.boxplot(x=train.item_cnt_day)

plt.figure(figsize=(10,4))
plt.xlim(train.item_price.min(), train.item_price.max()*1.1)
sns.boxplot(x=train.item_price)
```

Output:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d904d85d0>
```



Feature engineering:

```
# measure time it took for feature preprocessing

start_time = time.time()
```

Data Cleaning:

Removing the outliers and negative item prices.

```
train = train[(train.item_price < 100000) & (train.item_price > 0)]
train = train[train.item_cnt_day < 1001]
```

Some shops are duplicates. Merge them

```
train.loc[train.shop_id == 0, 'shop_id'] = 57
test.loc[test.shop_id == 0, 'shop_id'] = 57

train.loc[train.shop_id == 1, 'shop_id'] = 58
test.loc[test.shop_id == 1, 'shop_id'] = 58
```

```
train.loc[train.shop_id == 10, 'shop_id'] = 11
test.loc[test.shop_id == 10, 'shop_id'] = 11
```

Basic Feature Engineering + Extracting text features:

```
# REVENUE
train['revenue'] = train['item_price'] *  train['item_cnt_day']


# CITY NAME text feature
# city is before each shop
shops.loc[shops.shop_name == 'Сергиев Посад ТЦ "7Я"',"shop_name"] = 'СергиевПосад ТЦ "7Я"'
shops['city'] = shops['shop_name'].str.split(' ').map(lambda x: x[0])
shops.loc[shops.city == '!Якутск', 'city'] = 'Якутск'
shops['city_code'] = LabelEncoder().fit_transform(shops['city']).astype(np.int8) # Applying label encoding

# SHOP CATEGORY text feature
shops["category"] = shops.shop_name.str.split(" ").map( lambda x: x[1] )
category = []
for cat in shops.category.unique():
    if len(shops[shops.category == cat]) >= 5:
        category.append(cat)
shops.category = shops.category.apply( lambda x: x if (x in category) else "other" )
shops['category_code'] = LabelEncoder().fit_transform(shops['category']).astype(np.int8) # Applying label encoding


# POPULATION
population = {'Якутск': 269601, 'Адыгея':144249, 'Балашиха': 215494, 'Волжский': 314255, 'Вологда': 301755, 'Воронеж': 889680,
       'Жуковский': 104736, 'Казань': 1143535, 'Калуга':324698,
       'Коломна': 144589, 'Красноярск': 973826, 'Курск': 415159, 'Москва':11503501, 'Мытищи': 173160, 'Н.Новгород': 1250619,
       'Новосибирск': 1473754, 'Омск': 1154116, 'РостовНаДону': 1089261, 'СПб': 4879566, 'Самара': 1164685,
       'СергиевПосад': 111179, 'Сургут': 306675, 'Томск': 524669, 'Тюмень': 581907, 'Уфа': 1062319, 'Химки': 207425,
       'Чехов': 60720, 'Ярославль': 591486}
# filling the online store and other non-locations with mean
mean_population = int(np.mean([v for k, v in population.items()]))
population['Выездная'] = mean_population
population['Интернет-магазин'] = mean_population
population['Цифровой'] = mean_population
shops['city_population'] = shops['city'].map(lambda x: population[x]).astype(np.int32)
```

```
# COORDINATES
# lattitude and longitude of the cities
coords = dict()
coords['Якутск'] = (62.028098, 129.732555, 4)
coords['Адыгея'] = (44.609764, 40.100516, 3)
coords['Балашиха'] = (55.8094500, 37.9580600, 1)
coords['Волжский'] = (53.4305800, 50.1190000, 3)
coords['Вологда'] = (59.2239000, 39.8839800, 2)
coords['Воронеж'] = (51.6720400, 39.1843000, 3)
coords['Выездная'] = (0, 0, 0)
coords['Жуковский'] = (55.5952800, 38.1202800, 1)
coords['Интернет-магазин'] = (0, 0, 0)
coords['Казань'] = (55.7887400, 49.1221400, 4)
coords['Калуга'] = (54.5293000, 36.2754200, 4)
coords['Коломна'] = (55.0794400, 38.7783300, 4)
coords['Красноярск'] = (56.0183900, 92.8671700, 4)
coords['Курск'] = (51.7373300, 36.1873500, 3)
coords['Москва'] = (55.7522200, 37.6155600, 1)
coords['Мытищи'] = (55.9116300, 37.7307600, 1)
coords['Н.Новгород'] = (56.3286700, 44.0020500, 4)
coords['Новосибирск'] = (55.0415000, 82.9346000, 4)
coords['Омск'] = (54.9924400, 73.3685900, 4)
coords['РостовНаДону'] = (47.2313500, 39.7232800, 3)
coords['СПб'] = (59.9386300, 30.3141300, 2)
coords['Самара'] = (53.2000700, 50.1500000, 4)
coords['СергиевПосад'] = (56.3000000, 38.1333300, 4)
coords['Сургут'] = (61.2500000, 73.4166700, 4)
coords['Томск'] = (56.4977100, 84.9743700, 4)
coords['Тюмень'] = (57.1522200, 65.5272200, 4)
coords['Уфа'] = (54.7430600, 55.9677900, 4)
coords['Химки'] = (55.8970400, 37.4296900, 1)
coords['Цифровой'] = (0, 0, 0)
coords['Чехов'] = (55.1477000, 37.4772800, 4)
coords['Ярославль'] = (57.6298700, 39.8736800, 2)

shops['city_coord_1'] = shops['city'].apply(lambda x: coords[x][0]).ast
ype(np.float16)
shops['city_coord_2'] = shops['city'].apply(lambda x: coords[x][1]).ast
ype(np.float16)
shops['country_part'] = shops['city'].apply(lambda x: coords[x][2]).ast
ype(np.int8)


# SHOP TYPE
cats['split'] = cats['item_category_name'].str.split('-') # item catego
ry name is 'type-subtype'
cats['type'] = cats['split'].map(lambda x: x[0].strip())
cats['type_code'] = LabelEncoder().fit_transform(cats['type']).astype(n
p.int8) # Applying label encoding
```

```
cats['subtype'] = cats['split'].map(lambda x: x[1].strip() if len(x) > 
1 else x[0].strip())
cats['subtype_code'] = LabelEncoder().fit_transform(cats['subtype']).as
type(np.int8) # Applying label encoding
```

Selecting only the wanted columns:

```
# dropping unnecessary columns
cats = cats[['item_category_id','type_code', 'subtype_code']]
shops = shops[['shop_id','city_code', 'city_coord_1', 'city_coord_2', '
country_part', 'city_population', 'category_code']]
items.drop(['item_name'], axis=1, inplace=True)
```

Generate all combinations of items and shops for each month:

```
matrix = []
cols  = ["date_block_num", "shop_id", "item_id"]
for i in range(34):
    sales = train[train.date_block_num == i]
    matrix.append( np.array(list( product( [i], sales.shop_id.unique(), 
sales.item_id.unique() ) ), dtype = np.int16) )

matrix = pd.DataFrame( np.vstack(matrix), columns = cols )
matrix["date_block_num"] = matrix["date_block_num"].astype(np.int8)
matrix["shop_id"] = matrix["shop_id"].astype(np.int8)
matrix["item_id"] = matrix["item_id"].astype(np.int16)
matrix.sort_values( cols, inplace = True )
```

Calculating monthly items sold (target).

Filling missing values with zeros and clipping targets into [0,20] as suggested

```
group = train.groupby(['date_block_num','shop_id','item_id']).agg({'ite
m_cnt_day': ['sum']})
group.columns = ['item_cnt_month']
group.reset_index(inplace=True)

matrix = pd.merge(matrix, group, on=cols, how='left')
matrix['item_cnt_month'] = (matrix['item_cnt_month']
                                .fillna(0)
                                .clip(0,20)
                                .astype(np.float16))
```

Adding the correct block number to the test set :

```
test['date_block_num'] = 34
test['date_block_num'] = test['date_block_num'].astype(np.int8)

matrix = pd.concat([matrix, test], ignore_index=True, sort=False, keys=
cols)
matrix.fillna(0, inplace=True)
```

Merging the dataframes:

```
matrix = pd.merge(matrix, shops, on=['shop_id'], how='left')
matrix = pd.merge(matrix, items, on=['item_id'], how='left')
matrix = pd.merge(matrix, cats, on=['item_category_id'], how='left')
```

Lagging features:

```
# lagging features
def lag_feature(df, lags, col):
    tmp = df[['date_block_num','shop_id','item_id',col]]
    for i in lags:
        shifted = tmp.copy()
        shifted.columns = ['date_block_num','shop_id','item_id', col+'_
lag_'+str(i)]
        shifted['date_block_num'] += i
        df = pd.merge(df, shifted, on=['date_block_num','shop_id','item
_id'], how='left')
    return df

# lagged feature for item with id one less than current item
def lag_feature_adv(df, lags, col):
    tmp = df[['date_block_num','shop_id','item_id',col]]
    for i in lags:
        shifted = tmp.copy()
        shifted.columns = ['date_block_num','shop_id','item_id', col+'_
lag_'+str(i)+'_adv']
        shifted['date_block_num'] += i
        shifted['item_id'] -= 1
        df = pd.merge(df, shifted, on=['date_block_num','shop_id','item
_id'], how='left')
        df[col+'_lag_'+str(i)+'_adv'] = df[col+'_lag_'+str(i)+'_adv'].a
stype('float16')
    return df
```

Generated lagged features:

```
# generating lagged features
matrix = lag_feature(matrix, [1, 2, 3, 6, 12], 'item_cnt_month')
matrix = lag_feature_adv(matrix, [1, 2, 3, 6, 12], 'item_cnt_month')
```

Mean encoded lagged features

Generating mean encoded features

```
def mean_encoded_feature(df, group_by, feature, new_feature, lags=[1]):
    # mean encoding the feature
    group = df.groupby(group_by).agg({feature: ['mean']})
    group.columns = [new_feature]
    group.reset_index(inplace=True)

    df = pd.merge(df, group, on=group_by, how='left')
    df[new_feature] = df[new_feature].astype(np.float16)

    # Lagging the mean encoded feature
    df = lag_feature(df, lags, new_feature)

    # Removing for the current month
    df.drop([new_feature], axis=1, inplace=True)
    return df
# mean encoding various features
matrix = mean_encoded_feature(matrix, ['date_block_num'], 'item_cnt_mon
th', 'date_target_enc', [1])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'item_id'], 'i
tem_cnt_month', 'date_item_target_enc', [1,2,3,6,12])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'shop_id'], 'i
tem_cnt_month', 'date_shop_target_enc', [1,2,3,6,12])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'item_id', 'sh
op_id'], 'item_cnt_month', 'date_item_shop_target_enc', [1,2,3])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'item_category
_id'], 'item_cnt_month', 'date_cat_target_enc', [1])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'shop_id', 'it
em_category_id'], 'item_cnt_month', 'date_shop_cat_target_enc', [1])
matrix = mean_encoded_feature(matrix, ['date_block_num', 'shop_id', 'ty
pe_code'], 'item_cnt_month', 'date_shop_type_target_enc', [1])
```

Revenue features

```
# Total monthly revenue for each shop

group = train.groupby(['date_block_num','shop_id']).agg({'revenue': ['sum']
})
group.columns = ['date_shop_revenue']
group.reset_index(inplace=True)

matrix = pd.merge(matrix, group, on=['date_block_num','shop_id'], how='left
')
matrix['date_shop_revenue'] = matrix['date_shop_revenue'].astype(np.float32
)

# total average revenue for each shop
group = group.groupby(['shop_id']).agg({'date_shop_revenue': ['mean']})
```

```
group.columns = ['shop_avg_revenue']
group.reset_index(inplace=True)

matrix = pd.merge(matrix, group, on=['shop_id'], how='left')
matrix['shop_avg_revenue'] = matrix['shop_avg_revenue'].astype(np.float32)

# Monthly revenue difference from the average divided by the average revenue
for scaling
matrix['delta_revenue'] = (matrix['date_shop_revenue'] - matrix['shop_avg_r
evenue']) / matrix['shop_avg_revenue']
matrix['delta_revenue'] = matrix['delta_revenue'].astype(np.float16)

# Lagging the feature
matrix = lag_feature(matrix, [1], 'delta_revenue')

matrix.drop(['date_shop_revenue','shop_avg_revenue','delta_revenue'], axis=
1, inplace=True)
```

Date features:

```
import calendar

# month number from 0 to 11
matrix['month'] = matrix['date_block_num'] % 12
days = pd.Series([31,28,31,30,31,30,31,31,30,31,30,31])

# days in each month
matrix['days'] = matrix['month'].map(days).astype(np.int8)

# number of weekends each month
def weekends(date_block_num):
    month = date_block_num % 12 + 1
    year = 2013 + date_block_num // 12
    return len([1 for i in calendar.monthcalendar(year, month) if i[6]
!= 0])

matrix['weekends'] = matrix['date_block_num'].apply(lambda x: weekends(
x)).astype(np.int8)
```

Item date features (interactions)

Finding when item first appeared and when it was first bought.

```
# item first appeared
first_item_block = matrix.groupby(['item_id'])['date_block_num'].min().
reset_index()
first_item_block['item_first_interaction'] = 1

# item first bought
first_shop_item_buy_block = matrix[matrix['date_block_num'] > 0].groupb
y(['shop_id', 'item_id'])['date_block_num'].min().reset_index()
```

```
first_shop_item_buy_block['first_date_block_num'] = first_shop_item_buy
_block['date_block_num']

matrix = pd.merge(matrix, first_item_block[['item_id', 'date_block_num'
, 'item_first_interaction']], on=['item_id', 'date_block_num'], how='le
ft')
matrix = pd.merge(matrix, first_shop_item_buy_block[['item_id', 'shop_i
d', 'first_date_block_num']], on=['item_id', 'shop_id'], how='left')

matrix['first_date_block_num'].fillna(100, inplace=True)
matrix['shop_item_sold_before'] = (matrix['first_date_block_num'] < mat
rix['date_block_num']).astype('int8')
matrix.drop(['first_date_block_num'], axis=1, inplace=True)

matrix['item_first_interaction'].fillna(0, inplace=True)
matrix['shop_item_sold_before'].fillna(0, inplace=True)
```

```
# is it the first time the item appears
matrix['item_first_interaction'] = matrix['item_first_interaction'].ast
ype('int8')
```

```
# average category sales for the new item
item_id_target_mean = matrix[matrix['item_first_interaction'] == 1].gro
upby(['date_block_num','subtype_code'])['item_cnt_month'].mean().reset_
index().rename(columns={
    "item_cnt_month": "new_item_cat_avg"}, errors="raise")

matrix = pd.merge(matrix, item_id_target_mean, on=['date_block_num','su
btype_code'], how='left')

matrix['new_item_cat_avg'] = (matrix['new_item_cat_avg']
                              .fillna(0)
                              .astype(np.float16))

matrix = lag_feature(matrix, [1, 2, 3], 'new_item_cat_avg')
matrix.drop(['new_item_cat_avg'], axis=1, inplace=True)

# average category sales for the new item in each store
item_id_target_mean = matrix[matrix['item_first_interaction'] == 1].gro
upby(['date_block_num','subtype_code', 'shop_id'])['item_cnt_month'].me
an().reset_index().rename(columns={
    "item_cnt_month": "new_item_shop_cat_avg"}, errors="raise")

matrix = pd.merge(matrix, item_id_target_mean, on=['date_block_num','su
btype_code', 'shop_id'], how='left')

matrix['new_item_shop_cat_avg'] = (matrix['new_item_shop_cat_avg']
                              .fillna(0)
```

```
                                    .astype(np.float16))

matrix = lag_feature(matrix, [1, 2, 3], 'new_item_shop_cat_avg')
matrix.drop(['new_item_shop_cat_avg'], axis=1, inplace=True)
```

Time since first sale and since first sale at the shop

```
# months since first sale in the shop
matrix['item_shop_first_sale'] = matrix['date_block_num'] - matrix.grou
pby(['item_id','shop_id'])['date_block_num'].transform('min')
# months since first sale over all shops
matrix['item_first_sale'] = matrix['date_block_num'] - matrix.groupby('
item_id')['date_block_num'].transform('min')
```

Final Touches:

```
# fill the nan's caused by lagging the features

def fill_na(df):
    for col in df.columns:
        if ('_lag_' in col) & (df[col].isnull().any()):
            df[col].fillna(0, inplace=True)
    return df

matrix = fill_na(matrix)
```

Saving for quicker loading later:

```
# saving all the data
matrix.to_pickle('all_data.pkl')
# data for using 12 month lags (first 12 months already removed)
matrix = matrix[matrix.date_block_num > 11]
matrix.to_pickle('data.pkl')
```

```
# measuring the time the preprocessing and feature engineering took
end_time = time.time()
print(f"Preprocessing took {end_time - start_time}s")
```

Preprocessing took 649.7621595859528s

Algorithm for the feature Engineering:

- Imputation. Imputation deals with handling missing values in data. ...
- Discretization. ...
- Categorical Encoding. ...
- Feature Splitting. ...
- Handling Outliers. ...
- Variable Transformations. ...
- Scaling. ...
- Feature Creation in Machine Learning.

2.Model Training:

Once the features are prepared, the next step is to train a machine learning model. You can use various regression algorithms for IMDb score prediction. Some popular choices include:

- Linear Regression
- Decision Trees
- Random Forest
- Gradient Boosting
- Neural Networks

In [16]:

```
from sklearn.model_selection import cross_val_score

# Performing 5-fold cross-validation (can be adjusted to the desired number o
f folds)
num_folds = 5

# Function to perform cross-validation and calculate metrics in percentage
def perform_cross_validation(model, X, y, num_folds):
    mse_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_m
ean_squared_error')
    rmse_scores = np.sqrt(mse_scores)
    mae_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_m
ean_absolute_error')
    r2_scores = cross_val_score(model, X, y, cv=num_folds, scoring='r2')

    return mse_scores, rmse_scores, mae_scores, r2_scores
```

Modelling and Evaluation

At the modeling stage, we use 5 algorithms for comparison, namely Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Random Forest.

And for evaluation using MSE, RMSE, MAE and R-Squared.

In [15]:

```
X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']
```

In [16]:

```
from sklearn.model_selection import cross_val_score

# Performing 5-fold cross-validation (can be adjusted to the desired number o
f folds)
num_folds = 5

# Function to perform cross-validation and calculate metrics in percentage
def perform_cross_validation(model, X, y, num_folds):
    mse_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_m
ean_squared_error')
    rmse_scores = np.sqrt(mse_scores)
    mae_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_m
ean_absolute_error')
    r2_scores = cross_val_score(model, X, y, cv=num_folds, scoring='r2')

    return mse_scores, rmse_scores, mae_scores, r2_scores
```

In [18]:

```
# Ridge Regression
ridge_model = Ridge(alpha=1.0)  # You can adjust alpha as needed
ridge_mse, ridge_rmse, ridge_mae, ridge_r2 = perform_cross_validation(ridge
_model, X, y, num_folds)
print("Ridge Regression:")
print(f"Average MSE: {np.mean(ridge_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(ridge_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(ridge_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(ridge_r2) * 100:.2f}%")
print("\n")
```

```
Ridge Regression:
Average MSE: 19.67%
Average RMSE: 11.20%
Average MAE: 8.54%
Average R-squared: 89.19%
```

3.Evalution:

Evaluating the performance of your IMDb score prediction model is crucial to ensure it provides accurate and reliable results. Several metrics can be used for evaluation, such as:

- Mean Squared Error (MSE): This measures the average squared difference between predicted and actual IMDb scores.
- Root Mean Squared Error (RMSE): The square root of the MSE, which provides a more interpretable measure.
- Mean Absolute Error (MAE): This calculates the average absolute difference between predicted and actual IMDb scores.
- R-squared (R2): A measure of how well the model explains the variance in the data.