# TNSDC-NAAN MUDHALVAN(IBM)

## COLLEGE CODE:7240

**COLLEGE NAME: Dhaanish Ahmed Institute of Technology**

**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**

## 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:

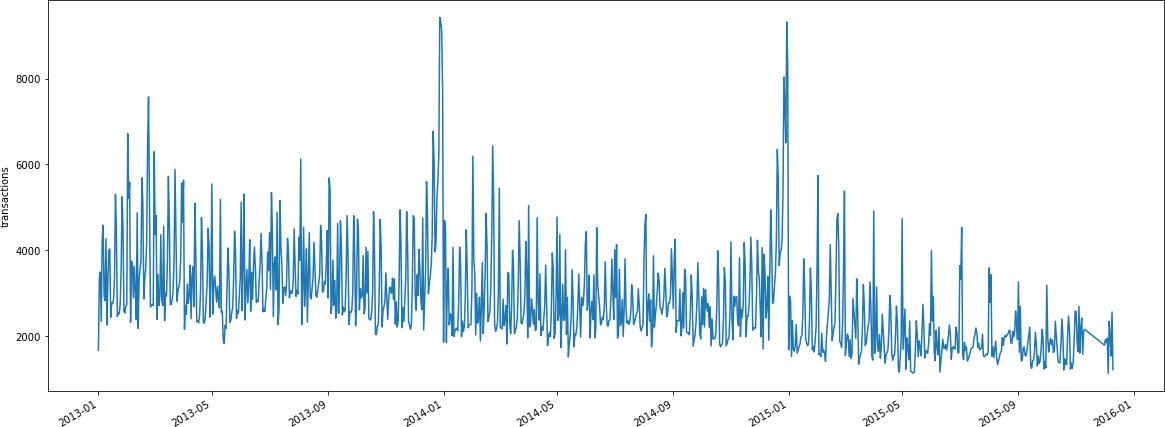
pd.to\_datetime(train.date).value\_counts().sort\_index(ascending=False).p lot(kind='line')

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()

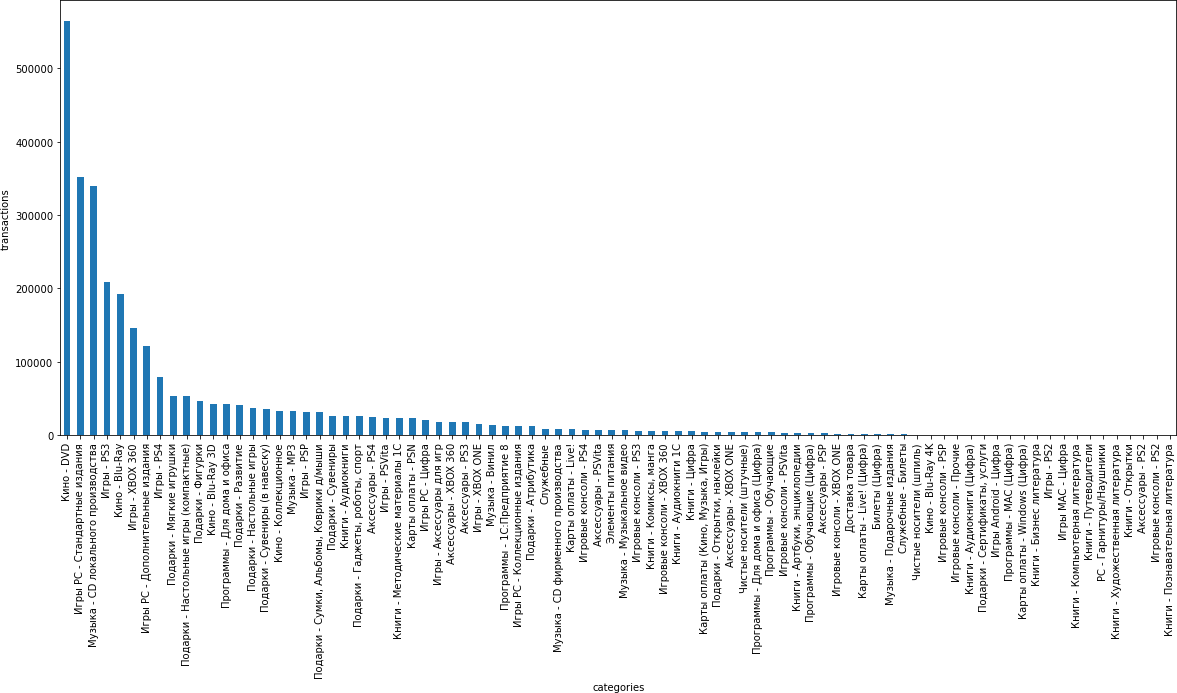
fig.set\_size\_inches(20, 8) plt.ylabel("transactions") plt.xlabel("categories") plt.show()

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"] = 'Се ргиевПосад 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) e lse "other" )

shops['category\_code'] = LabelEncoder().fit\_transform(shops['category']

).astype(np.int8) *# Applying label encoding*

*# POPULATION*

population = {'Якутск': 269601, 'Адыгея':144249, 'Балашиха': 215494, 'В олжский': 314255, 'Вологда': 301755, 'Воронеж': 889680,

'Жуковский': 104736, 'Казань': 1143535, 'Калуга':324698,

'Коломна': 144589, 'Красноярск': 973826, 'Курск': 415159, 'Москв

а':11503501, 'Мытищи': 173160, 'Н.Новгород': 1250619,

'Новосибирск': 1473754, 'Омск': 1154116, 'РостовНаДону': 1089261

, 'СПб': 4879566, 'Самара': 1164685,

'СергиевПосад': 111179, 'Сургут': 306675, 'Томск': 524669, 'Тюме

нь': 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]).a stype(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()

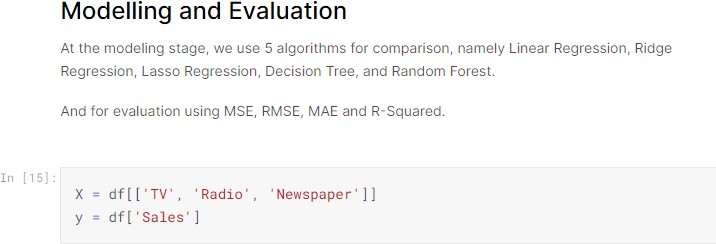
print(f"Preprocessing took **{**end\_time - start\_time**}**s")

## Preprocessing took 649.7621595859528s

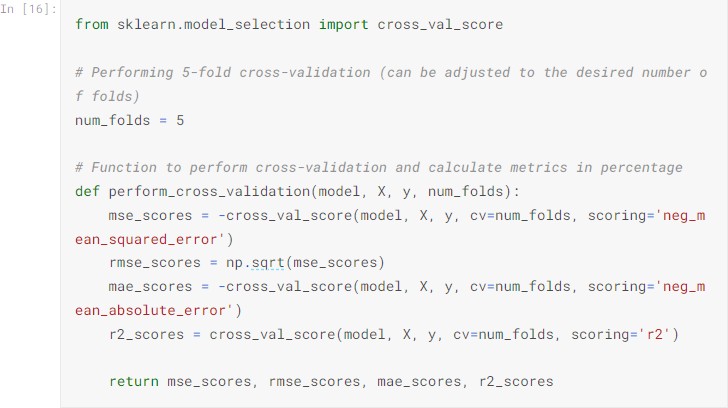
**Algorithm for the feature Engineering:**

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

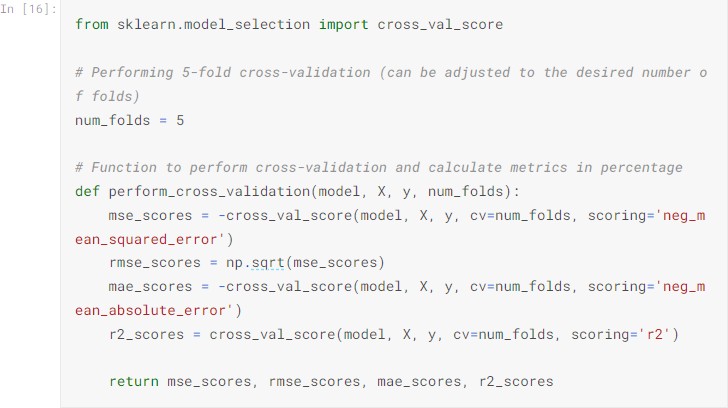
1. **Model Training:**



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





# 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.