Sales Forecasting WOMart Store

https://www.kaggle.com/shelvigarg/sales-forecasting-womart-store?select=TRAIN.csv

WOMart is a leading nutrition and supplement retail chain that offers a comprehensive range of products for all your wellness and fitness needs.

WOMart follows a multi-channel distribution strategy with 350+ retail stores spread across 100+ cities.

Problem: to predict the store sales for each store in the test set for the next two months.

About Train Data

> - ID: Unique Identifier for a row
> - Store_id: Unique id for each Store
> - Store_Type: Type of the Store
> - Location_Type: Type of the location where Store is located
> - Region_Code: Code of the Region where Store is located
> - Date: Information about the Date
> - Holiday: If there is holiday on the given Date, 1 : Yes, 0 : No
> - Discount: If discount is offered by store on the given Date, Yes/ No
> - Orders: Number of Orders received by the Store on the given Day
> - Sales: Total Sale for the Store on the given Day

```
jovian.commit(project='real-world-machine-learning-model-sales-forecasting')
```

```
[jovian] Detected Colab notebook...
[jovian] Please enter your API key ( from https://jovian.ai/ ):
API KEY: ··········
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting
```

```
'https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting'
```

Required Library

```
!pip install panda numpy matplotlib seaborn plotly opendatasets --upgrade --quiet
```

```
     |████████████████████████████████| 15.7 MB 250 kB/s
     |████████████████████████████████| 11.2 MB 40.4 MB/s
     |████████████████████████████████| 895 kB 39.3 MB/s
```

```
  Building wheel for panda (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
yellowbrick 1.3.post1 requires numpy<1.20,>=1.16.0, but you have numpy 1.21.5 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
```

Loading Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
import plotly.express as px
import os
import opendatasets as od
```

```
pd.set_option("display.max_columns", 120)
pd.set_option("display.max_rows", 120)
{"username":"shawket","key":"af2f5a5dc1deb79d9206f22e40689220"}
```

```
{'key': 'af2f5a5dc1deb79d9206f22e40689220', 'username': 'shawket'}
```

```
od.download('https://www.kaggle.com/shelvigarg/sales-forecasting-womart-store')
```

```
Downloading sales-forecasting-womart-store.zip to ./sales-forecasting-womart-store

100%|██████████| 2.16M/2.16M [00:00<00:00, 41.3MB/s]
```

```
os.listdir('./sales-forecasting-womart-store')
```

```
['TEST_FINAL.csv', 'TRAIN.csv']
```

```
raw_train_df =pd.read_csv('./sales-forecasting-womart-store/TRAIN.csv')
y_test_df = pd.read_csv('./sales-forecasting-womart-store/TEST_FINAL.csv')
```

```
raw_train_df.shape, y_test_df.shape
```

```
((188340, 10), (22265, 8))
```

```
raw_train_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
92348	T1092349	329	S4	L2	R3	2018-09-11	0	Yes	83	57459.0
118143	T1118144	277	S1	L3	R1	2018-11-20	0	No	59	30048.0
65457	T1065458	140	S4	L1	R1	2018-06-29	0	No	63	39579.0
91481	T1091482	230	S2	L4	R4	2018-09-08	0	No	30	19101.0
22989	T1022990	74	S3	L1	R2	2018-03-04	0	Yes	93	58242.0

```
y_test_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount
14698	T1203039	198	S4	L2	R3	2019-07-11	0	Yes
18573	T1206914	50	S3	L2	R4	2019-07-21	0	Yes
4761	T1193102	63	S1	L1	R2	2019-06-14	0	Yes
5358	T1193699	121	S1	L1	R4	2019-06-15	0	No
2966	T1191307	31	S1	L5	R2	2019-06-09	0	Yes

```
raw_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188340 entries, 0 to 188339
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   ID             188340 non-null  object 
 1   Store_id       188340 non-null  int64  
 2   Store_Type     188340 non-null  object 
 3   Location_Type  188340 non-null  object 
 4   Region_Code    188340 non-null  object 
 5   Date           188340 non-null  object 
 6   Holiday        188340 non-null  int64  
 7   Discount       188340 non-null  object 
 8   #Order         188340 non-null  int64  
 9   Sales          188340 non-null  float64
```

```
dtypes: float64(1), int64(3), object(6)
memory usage: 14.4+ MB
```

```
y_test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22265 entries, 0 to 22264
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   ID             22265 non-null  object
 1   Store_id       22265 non-null  int64 
 2   Store_Type     22265 non-null  object
 3   Location_Type  22265 non-null  object
 4   Region_Code    22265 non-null  object
 5   Date           22265 non-null  object
 6   Holiday        22265 non-null  int64 
 7   Discount       22265 non-null  object
dtypes: int64(2), object(6)
memory usage: 1.4+ MB
```

Feature Enginearing

Date

as time serise data date column is important and aslo spliting is also important

```
def dateConvertion(df):
  df.Date = pd.to_datetime(df.Date)
  df['Year'] = df.Date.dt.year
  df['Month'] = df.Date.dt.month
  df['Day']  = df.Date.dt.day
  df['Day_name'] = df.Date.dt.day_name().str[:3]
  df['WeekOfYear'] = df.Date.dt.isocalendar().week
```

```
dateConvertion(raw_train_df)
dateConvertion(y_test_df)
```

```
raw_train_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	Y
114697	T1114698	357	S4	L1	R1	2018-11-11	0	Yes	163	93732.0	2

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	\
47598	T1047599	19	S1	L1	R3	2018-05-11	0	No	42	32388.0	2
16899	T1016900	43	S1	L1	R2	2018-02-16	0	No	47	29793.0	2
135253	T1135254	270	S1	L1	R2	2019-01-06	0	Yes	67	55905.0	2
157275	T1157276	292	S1	L1	R1	2019-03-07	0	No	36	23109.0	2

Month-Year column for EDA

```
raw_train_df['MonYr'] = raw_train_df['Month'].map(str) +'-'+raw_train_df['Year'].map(st
raw_train_df['MonYr'] =pd.to_datetime(raw_train_df['MonYr'], format ='%m-%Y').dt.strfti
```

```
raw_train_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
184050	T1184051	307	S1	L5	R2	2019-05-20	0	No	53	30345.00
95564	T1095565	309	S1	L5	R1	2018-09-19	0	Yes	69	33525.00
178008	T1178009	62	S2	L5	R1	2019-05-03	0	No	35	23487.00
82517	T1082518	350	S1	L3	R2	2018-08-15	1	No	27	14781.78
175411	T1175412	281	S1	L4	R1	2019-04-26	0	No	76	40506.00

```
raw_train_df.describe(include='all')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.
  """Entry point for launching an IPython kernel.
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	
count	188340	188340.000000	188340	188340	188340	188340	188340.000000	188340	18
unique	188340	NaN	4	5	4	516	NaN	2	
top	T1123744	NaN	S1	L1	R1	2018-07-17 00:00:00	NaN	No	
freq	1	NaN	88752	85140	63984	365	NaN	104051	
first	NaN	NaN	NaN	NaN	NaN	2018-01-01 00:00:00	NaN	NaN	

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount
last	NaN	NaN	NaN	NaN	NaN	2019-05-31 00:00:00	NaN	NaN
mean	NaN	183.000000	NaN	NaN	NaN	NaN	0.131783	NaN
std	NaN	105.366308	NaN	NaN	NaN	NaN	0.338256	NaN
min	NaN	1.000000	NaN	NaN	NaN	NaN	0.000000	NaN
25%	NaN	92.000000	NaN	NaN	NaN	NaN	0.000000	NaN
50%	NaN	183.000000	NaN	NaN	NaN	NaN	0.000000	NaN
75%	NaN	274.000000	NaN	NaN	NaN	NaN	0.000000	NaN
max	NaN	365.000000	NaN	NaN	NaN	NaN	1.000000	NaN

Note about Data

> - No. of Store_Type 4,
> - No. of Location_Type 5,
> - No. of Region_Code 4,
> - Data Duration from 01 Jan 2018 to 31 May 2019

Data cleaning and Processing

> - Cleaning NAN values.
> - Holidays sales
> - Outlier

Null Value

```
raw_train_df.isnull().sum()
```

```
ID              0
Store_id        0
Store_Type      0
Location_Type   0
Region_Code     0
Date            0
Holiday         0
Discount        0
#Order          0
Sales           0
Year            0
Month           0
Day             0
Day_name        0
```

```
WeekOfYear       0
MonYr            0
dtype: int64
```

```
y_test_df.isnull().sum()
```

```
ID               0
Store_id         0
Store_Type       0
Location_Type    0
Region_Code      0
Date             0
Holiday          0
Discount         0
Year             0
Month            0
Day              0
Day_name         0
WeekOfYear       0
dtype: int64
```

There is no null values

Holidays Sales

```
print ('No. of Non holidays : ',raw_train_df[raw_train_df.Holiday==0].shape[0])
print ('No. of holidays : ',raw_train_df[raw_train_df.Holiday==1].shape[0])
print('no of Sales not 0 with Holiday : ', raw_train_df[(raw_train_df.Holiday==1) &(raw
```

```
No. of Non holidays :  163520
No. of holidays :  24820
no of Sales not 0 with Holiday :  24820
```

```
print('Holidays with no sales:',
      raw_train_df[raw_train_df.Holiday==1].shape[0] - raw_train_df[(raw_train_df.Holic
```

```
Holidays with no sales: 5
```

Mentioned Holidays, actually their is no holidays at all exceptional for 5 Days.

because all holidays has non 0 (zero) sales. That means store are open in holidays too.

Sales 0 Replacement

> - replaceing 0 sales with matiching Store type, region, location, that year and that week of year mean value

```
print('No. of records with Sales = 0 is ', raw_train_df[raw_train_df.Sales==0].Sales.co
raw_train_df[raw_train_df.Sales==0].sample(5)
```

```
No. of records with Sales = 0 is  19
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	Yea
90458	T1090459	233	S1	L3	R3	2018-09-05	0	No	0	0.0	2018
38143	T1038144	222	S4	L2	R2	2018-04-15	1	No	0	0.0	2018
100215	T1100216	12	S1	L3	R2	2018-10-02	1	No	0	0.0	2018
166738	T1166739	204	S4	L1	R3	2019-04-02	0	No	0	0.0	2019
97301	T1097302	271	S1	L3	R2	2018-09-24	0	No	0	0.0	2018

```
raw_train_df['Sales'] = raw_train_df.apply(lambda x : x['Sales'] if x['Sales'] !=0
                                           else raw_train_df[(raw_train_df['Store_Type
                                               (raw_train_df['Location_Type']== x['Loc
                                               (raw_train_df['Region_Code']== x['Regio
                                               (raw_train_df['WeekOfYear'] == x['WeekO
                                               (raw_train_df['Year'] == x['Year'])].Sa

#raw_train_df.sample(5)
```

```
raw_train_df[raw_train_df.Date ==0]
```

ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	Year	Month	Day

Outlier

```
fig =px.histogram(raw_train_df, x= 'Sales',
                  color='Store_Type',
                  marginal ='violin',
                 # histnorm ='density'
                  title = 'Histogram of Sales with Store Type'
                  )
fig.show()
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Histogram chart shows there is many outlier. this may be inturupt the Forecasting. Lets set Max Sales range for the Store Type with help of above histogram.

- Store Type S1 sales over 125K contain outlier.
- Store Type S2 sales over 76K contain outlier.

- Store Type S3 sales over 126K contain outlier.
- Store Type S4 sales over 189K contain outlier.

these outlier will be replace with their max range sales excluding outlier

```
store= {'sto':['S1','S2','S3','S4'],
        'val' :[125000, 76000,126000,189000]}
store_df = pd.DataFrame(store)
```

```
# correcting outlier
for index, row in store_df.iterrows():
    raw_train_df['Sales'] = raw_train_df.apply(lambda x : row['val'] if (x['Sales'] >=
    & (x['Store_Type']==row['sto']) else x['Sales'], axis =1)
```

```
fig =px.histogram(raw_train_df, x= 'Sales',
                  color='Store_Type',
                  marginal ='violin',
                  title = 'Histogram of Sales with Store Type'
                  )
fig.show()
```

Output hidden; open in https://colab.research.google.com to view.

Explitory Data Analysis

```
raw_train_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
42683	T1042684	246	S3	L1	R3	2018-04-27	0	No	56	33384.00
107014	T1107015	320	S2	L3	R2	2018-10-21	0	No	45	25563.00
173412	T1173413	99	S1	L1	R3	2019-04-21	1	Yes	120	74463.99
56602	T1056603	325	S3	L1	R2	2018-06-05	0	No	61	42729.00
64362	T1064363	170	S1	L1	R2	2018-06-26	0	No	55	41193.00

```
y_test_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	Year	Month	Day
10848	T1199189	16	S1	L3	R1	2019-06-30	0	Yes	2019	6	30
11843	T1200184	219	S4	L2	R1	2019-07-03	0	Yes	2019	7	3
9303	T1197644	75	S3	L2	R3	2019-06-26	0	No	2019	6	26

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	Year	Month	Day
11329	T1199670	179	S1	L1	R2	2019-07-02	0	No	2019	7	2
3399	T1191740	82	S4	L2	R1	2019-06-10	0	Yes	2019	6	10

Monthly Sales Trend by Store type

```
fig = px.line(raw_train_df.groupby(['MonYr', 'Store_Type' ], as_index=False, sort=False
        x='MonYr', y ='Sales', color = 'Store_Type',
        )
fig.update_xaxes(
        tickangle = 90,
        title_text = "Month-Year",
        title_font = dict(size=18, family='Courier', color='crimson'),
        title_standoff = 25,   # Distance from base line
          tickfont=dict(family='Rockwell', color='crimson', size=14),
        )
fig.update_yaxes(
        title_text = "Monthly Sales",
        title_font = dict(size=18, family='Courier', color='crimson'),
        tickfont=dict(family='Rockwell', color='crimson', size=14),
        )
fig.update_layout(
     title={
        'text': "Monthly Sales Trend by Store type",
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'},
        title_font =dict(family='Rockwell', color='crimson', size=18),
        )
```

- Every Alternative month Sales has flauctuation, excluding Aug to Nov
- From Aug to Nov Sales fall down. its about half of Jan Sales. It may by the seasonality. May be This sales comes from some loyal customers of that Store.
- S1 and S4 store soled higher then s2 and S3. S1 store type sales highest and S2 type soled lowest.
- All store Types sales trend are almost simillar pattern through Jan to Dec with their own level.

Monthly Total Sales by Location Type

```
px.bar(raw_train_df.groupby(['MonYr','Location_Type'], as_index=False, sort=False)['Sal
        x='MonYr', y ='Sales', color = 'Location_Type',
        barmode='group',
        title ='Monthly Total Sales by Location Type')
```

Location wise monthly sales can be sorted in Decending order L1>L2>l3>L5>L4

Monthly Total Sales by Region

```
px.bar(raw_train_df.groupby(['MonYr','Region_Code'], as_index=False, sort=False)['Sales
        x='MonYr', y ='Sales', color = 'Region_Code',
        barmode='group',
        title ='Monthly Total Sales by Region')
```

Region wise monthly sales can be sorted in Decending order R1>R2>R3>R4

Total Sales Distribution among Store Type, Location and Region

```
px.sunburst(raw_train_df.groupby(['Store_Type', 'Location_Type','Region_Code' ], as_inc
            path=['Location_Type', 'Region_Code', 'Store_Type' ],
            values = 'Sales',
            color = 'Sales',
            title= 'Total Sales Distribution among Store Type, Location and Region'
             )
```

Week Day Sales

```
px.bar(raw_train_df[(raw_train_df.Month) ==1 & (raw_train_df.Day<7)& (raw_train_df.Year
        x= 'Store_Type', y ='Sales', color = 'Day_name',
        barmode='group',
        title ='Week Day wise Sales Jan 2019')
```

> - Above graph shows Sales are alse effected by Week day
> - Friday sales highest and Monday sales lowest except S2 type store.

Forecasting Sales

Required Class

```
!pip install jovian  xgboost graphviz lightgbm scikit-learn xgboost lightgbm --upgrade
```

```
     |████████████████████████████████| 173.6 MB 8.6 kB/s
     |████████████████████████████████| 46 kB 2.3 MB/s
     |████████████████████████████████| 2.0 MB 45.5 MB/s
```

Input and Target Columns

Let's select the columns that we'll use for training.

```
print ('Numaric Columns : ' ,raw_train_df.select_dtypes(include=['int', 'float']).colum
print ('Catagorical Columns : ' ,raw_train_df.select_dtypes(include=['object']).columns
print ('Date and Time Columns : ' ,raw_train_df.select_dtypes(include=['datetime']).col
```

```
Numaric Columns :  Index(['Store_id', 'Holiday', '#Order', 'Sales', 'Year', 'Month',
'Day'], dtype='object')
Catagorical Columns :  Index(['ID', 'Store_Type', 'Location_Type', 'Region_Code',
'Discount',
       'Day_name', 'MonYr'],
      dtype='object')
Date and Time Columns :  Index(['Date'], dtype='object')
```

Numarical and Catagorical Columns

```
num_cols = ['Holiday', 'Day','Year', 'Month', 'WeekOfYear']
cat_cols =['Store_Type', 'Location_Type', 'Region_Code', 'Discount']
input_cols = num_cols + cat_cols
Target_cols = 'Sales'
```

Scaling Numeric Features

Let's scale numeric values to the 0 to 1 range.

```
from sklearn.preprocessing import MinMaxScaler
```

```
train_df = raw_train_df[input_cols].copy()
```

```
numScal = MinMaxScaler().fit(train_df[num_cols])
train_df[num_cols] = numScal.transform(train_df[num_cols])
y_test_df[num_cols] = numScal.transform(y_test_df[num_cols])
```

```
train_df[num_cols].sample(5)
```

	Holiday	Day	Year	Month	WeekOfYear
64431	0.0	0.833333	0.0	0.454545	0.490196
141163	0.0	0.700000	1.0	0.000000	0.058824
108091	1.0	0.766667	0.0	0.818182	0.823529
68742	0.0	0.233333	0.0	0.545455	0.509804
167357	0.0	0.100000	1.0	0.272727	0.254902

Scaling Catagorical Features

> - Sacling catagorical column with OneHonEncoding

```
from sklearn.preprocessing import OneHotEncoder
```

```
catScal = OneHotEncoder(sparse=False, handle_unknown='ignore').fit(train_df[cat_cols])
```

```
encod_cols = list(catScal.get_feature_names_out(cat_cols))
```

```
encod_cols
```

```
['Store_Type_S1',
 'Store_Type_S2',
 'Store_Type_S3',
 'Store_Type_S4',
 'Location_Type_L1',
 'Location_Type_L2',
 'Location_Type_L3',
 'Location_Type_L4',
 'Location_Type_L5',
 'Region_Code_R1',
 'Region_Code_R2',
 'Region_Code_R3',
 'Region_Code_R4',
 'Discount_No',
 'Discount_Yes']
```

```
train_df[encod_cols] = catScal.transform(train_df[cat_cols])
y_test_df[encod_cols] = catScal.transform(y_test_df[cat_cols])
```

```
train_df.head(5)
```

	Holiday	Day	Year	Month	WeekOfYear	Store_Type	Location_Type	Region_Code	Discount	Store_Type_S1	Store
0	1.0	0.0	0.0	0.0	0.0	S1	L3	R1	Yes	1.0	
1	1.0	0.0	0.0	0.0	0.0	S4	L2	R1	Yes	0.0	
2	1.0	0.0	0.0	0.0	0.0	S3	L2	R1	Yes	0.0	
3	1.0	0.0	0.0	0.0	0.0	S2	L3	R1	Yes	0.0	
4	1.0	0.0	0.0	0.0	0.0	S2	L3	R4	Yes	0.0	

```
y_test_df.sample(5)
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	Year	Month	
13850	T1202191	173	S4	L2	R1	2019-07-08	0.0	Yes	1.0	0.545455	0.2
9757	T1198098	239	S3	L1	R3	2019-06-27	0.0	No	1.0	0.454545	0.8
13352	T1201693	50	S3	L2	R4	2019-07-07	0.0	No	1.0	0.545455	0.2
15973	T1204314	115	S4	L2	R1	2019-07-14	0.0	Yes	1.0	0.545455	0.4
11960	T1200301	162	S4	L1	R2	2019-07-03	0.0	No	1.0	0.545455	0.0

Split train and validation set

```
from sklearn.model_selection import train_test_split
```

```
x_train, val_train,x_target, val_target = train_test_split(train_df[num_cols+encod_cols
                                                                    raw_train_df[Targe
                                                                    train_size = 0.8,
```

Shape of input ant Validation Set

```
print('x_train_df shape: ', x_train.shape)
print('val_train_df shape: ', val_train.shape)
print('x_targer shape: ', x_target.shape)
print('val_target shape: ', val_target.shape)
```

```
x_train_df shape:  (150672, 20)
val_train_df shape:  (37668, 20)
x_targer shape:  (150672,)
val_target shape:  (37668,)
```

Model Selection

Our problem is supervised regrassional problem. So selecting best model for my problem, I will trysome of regrassional models (class) like

> - XGBRegressor,
> - RandomForestRegressor and
> - ElasticNet,
> - Ridge,
> - Lasso

Let look on which model is good fit for our problem. then good 2 model will be hyper parameter for tunning.

```
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNet, Ridge, Lasso
from sklearn.metrics import mean_squared_error
```

Helping Function

I can use this as a benchmark for hyperparmeter tuning.

```
def train_and_evaluate(mod_name, X_train, train_targets, X_val, val_targets, **params):
    model =''
    if mod_name =='xgb_model':
      model = XGBRegressor(random_state=10, n_jobs=-1, **params)
    if mod_name =='rfReg_model':
      model = RandomForestRegressor(random_state=10, n_jobs=-1, **params)
    if mod_name == 'enet':
      model =ElasticNet(random_state=42, alpha=2, **params)
    if mod_name == 'ridReg':
      model =Ridge(random_state=5, alpha=2, **params)
    if mod_name == 'lasReg':
      model =Lasso(random_state=5, alpha=2, **params)

    model.fit(X_train, train_targets)
    train_rmse = mean_squared_error(model.predict(X_train), train_targets, squared=Fals
    val_rmse = mean_squared_error(model.predict(X_val), val_targets, squared=False)
    return model, train_rmse, val_rmse
```

RandomForest Model Test with Default parameter

```
%%time
rfReg_model, train_rmse_rf, val_rmse_rf = train_and_evaluate('rfReg_model',x_train, x_t
```

```
CPU times: user 1min 4s, sys: 519 ms, total: 1min 5s
Wall time: 55.2 s
```

```
rfReg_model, train_rmse_rf, val_rmse_rf
#8400.004896201814, 9822.018646859702 # 8407.522837129078, 9768.465631793864
```

```
(RandomForestRegressor(n_jobs=-1, random_state=10),
 8407.494169354944,
 9764.723969618239)
```

```
rfReg_model.score(x_train, x_target), rfReg_model.score(val_train, val_target)
#(0.7927081155612595, 0.7158267319763129)
```

```
(0.7928751755621931, 0.7146090246988761)
```

XGB Model Test with Default parameter

```
%%time
xgb_model, train_rmse_xgb, val_rmse_xgb =  train_and_evaluate('xgb_model',x_train, x_ta
```

```
CPU times: user 34.1 s, sys: 116 ms, total: 34.2 s
Wall time: 17.6 s
```

```
xgb_model, train_rmse_xgb, val_rmse_xgb
# 9170.274347952873, 9341.678222834895 # 9175.973016737245, 9235.370780922176(42 inside
```

```
(XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=-1,
              num_parallel_tree=1, predictor='auto', random_state=10,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None),
 9175.973016737245,
 9235.370780922176)
```

```
xgb_model.score(x_train, x_target), xgb_model.score(val_train, val_target)
# (0.7529482818430573, 0.7429417653219261)
```

```
(0.7532806232436868, 0.7447128436664092)
```

```

```

ElasticNet Linear Regrassion Model Test with Default parameter

```
#'enet','ridReg', 'lasReg'
%%time
enet_model, train_rmse_en, val_rmse_en =  train_and_evaluate('enet',x_train, x_target,
```

```
CPU times: user 148 ms, sys: 26.9 ms, total: 174 ms
Wall time: 110 ms
```

```
enet_model, train_rmse_en, val_rmse_en
```

```
(ElasticNet(alpha=2, random_state=42), 15920.064791934556, 15759.367553345694)
```

```
enet_model.score(x_train, x_target), enet_model.score(val_train, val_target)
```

```
(0.25734233885093194, 0.2566422687053903)
```

Ridge Linear Regrassion Model Test with Default parameter

```
#'enet','ridReg', 'lasReg'
%%time
ridReg_model, train_rmse_rig, val_rmse_rig =  train_and_evaluate('ridReg',x_train, x_ta
```

```
CPU times: user 78.8 ms, sys: 69.8 ms, total: 149 ms
Wall time: 88.9 ms
```

```
ridReg_model, train_rmse_rig, val_rmse_rig
```

```
(Ridge(alpha=2, random_state=5), 12367.31997859511, 12246.831644730897)
```

```
ridReg_model.score(x_train, x_target), ridReg_model.score(val_train, val_target)
```

```
(0.551822346458865, 0.551081204662095)
```

Lasso Linear Regrassion Model Test with Default parameter

```
#'enet','ridReg', 'lasReg'
%%time
lasReg_model, train_rmse_las, val_rmse_las =  train_and_evaluate('lasReg',x_train, x_ta
```

```
CPU times: user 9.11 s, sys: 674 ms, total: 9.79 s
Wall time: 5.07 s
```

```
lasReg_model, train_rmse_las, val_rmse_las
```

```
(Lasso(alpha=2, random_state=5), 12367.627725557646, 12247.651630342087)
```

```
lasReg_model.score(x_train, x_target), lasReg_model.score(val_train, val_target)
```

```
(0.5518000413793728, 0.5510210880071538)
```

Comparing Model Score Test with Default parameter

```
df_errScore = pd.DataFrame({
    'Model':['rfReg_model','xgb_model','enet_model','ridReg_model', 'lasReg_model'],
    'Tarin_Score': [rfReg_model.score(x_train, x_target), xgb_model.score(x_train, x_ta
    'val_Score': [rfReg_model.score(val_train, val_target), xgb_model.score(val_train,
})
```

```
df_errScore.sort_values('val_Score', ascending=False)
```

	Model	Tarin_Score	val_Score
1	xgb_model	0.753281	0.744713
0	rfReg_model	0.792875	0.714609
3	ridReg_model	0.551822	0.551081
4	lasReg_model	0.551800	0.551021
2	enet_model	0.257342	0.256642

Comparing Model rmse with Default parameter

```
df_rmse = pd.DataFrame({
    'Model':['rfReg_model','xgb_model','enet_model','ridReg_model', 'lasReg_model'],
    'Tarin_rmse': [train_rmse_rf, train_rmse_xgb, train_rmse_en, train_rmse_rig, train_
    'val_rmse': [val_rmse_rf, val_rmse_xgb, val_rmse_en, val_rmse_rig, val_rmse_las]
})
```

```
df_rmse.sort_values('val_rmse')
```

	Model	Tarin_rmse	val_rmse
1	xgb_model	9175.973017	9235.370781
0	rfReg_model	8407.494169	9764.723970
3	ridReg_model	12367.319979	12246.831645
4	lasReg_model	12367.627726	12247.651630
2	enet_model	15920.064792	15759.367553

```

```

Hyperparameter Tuning and Regularization

Just like other machine learning models, there are several hyperparameters we can to adjust the capacity of model and reduce overfitting. The two good scoring models are -

> - XGBRegressor,
> - RandomForestRegressor

Helping Function

I will use this as a benchmark for hyperparmeter tuning

```
pd.options.plotting.backend = "plotly"
def test_param_and_plot(mod_name, param_name, param_values):
    train_errors, val_errors = [], []
    for value in param_values:
        params = {param_name: value}
```

```
        if mod_name =='xgb_model':
          model_nm, train_rmse, val_rmse = train_and_evaluate('xgb_model',x_train, x_ta
        if mod_name =='rfReg_model':
          model_nm, train_rmse, val_rmse = train_and_evaluate('rfReg_model',x_train, x_
        train_errors.append(train_rmse)
        val_errors.append(val_rmse)

    df_error = pd.DataFrame({
          'param_values': param_values,
          'train_errors': train_errors,
          'val_errors': val_errors
                            }).set_index('param_values')
    fig = df_error.plot()
    fig.update_layout(title='Overfitting curve: ' + param_name)
    fig.show();
```

Random Forests

Some of the hyperparameters values I will use in Random Forests for hyperparmeter tuning.

n_estimators

```
test_param_and_plot('rfReg_model','n_estimators', [600,700,800, 900])
```

The optimal values of n_estimators lies somewhere between 0 and unbounded.

Max_depth

```
test_param_and_plot('rfReg_model', 'max_depth', [10,15,20,25,30])
```

The optimal values of max_depth =20

min_samples_leaf

```
test_param_and_plot('rfReg_model','min_samples_split', [10,15,20,25,30])
```

The optimal values of min_samples_split_nodes=20.

min_samples_leaf

```
test_param_and_plot('rfReg_model','min_samples_leaf', [3,5,7,9,11])
```

The optimal values of minsamples_leaf=5.

ccp_alpha

```
test_param_and_plot('rfReg_model','ccp_alpha', [0.1,0.12,0.14, 0.16, 0.18])
```

The optimal values of cc_alpha lies somewhere between 0 and unbounded.

min_impurity_decrease

```
test_param_and_plot('rfReg_model','min_impurity_decrease', [1e-3,1e-4,1e-5,1e-6])
```

The optimal values of min_impurity_decrease lies somewhere between 0 and unbounded.

Putting obtained optimal Value

Random forest with customized hyperparameters based on our obtained optimal hyperpraams. Others parameters remian default.

```
rfReg_model_h, train_rmse_h, val_rmse_h =  train_and_evaluate('rfReg_model',x_train, x_
                                                              min_samples_split=20,
                                                              max_depth = 20,
                                                              min_samples_leaf=5
                                                              )
```

```
rfReg_model_h, train_rmse_h, val_rmse_h
```

```
(RandomForestRegressor(max_depth=20, min_samples_leaf=5, min_samples_split=20,
                       n_jobs=-1, random_state=10),
 8955.843148282844,
 9484.502009831787)
```

```
rf_targer_score_change =rfReg_model_h.score(val_train, val_target)/(df_errScore[df_errS
print('rf_val score impovement %.2f%%:' %(rf_targer_score_change*100))
```

```
rf_val score impovement 2.26%:
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting
```

```
'https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting'
```

XGBRegressor Model

Some of the hyperparameters values I will use in XGBooster for hyperparmeter tuning.

n_estimators

```
test_param_and_plot('xgb_model','n_estimators', [200,300,400,500,600,700])
```

The optimal values of n_estimators= 300.

max_depth

```
test_param_and_plot('xgb_model', 'max_depth', [5,6,7,8,9])
```

The optimal values of max_depth =7

colsample_bytree

```
test_param_and_plot('xgb_model','colsample_bytree', [  0.55, 0.6, 0.65, 0.7,0.75])
```

The optimal values of colsample_bytree =0.7

learning_rate

```
test_param_and_plot('xgb_model','learning_rate', [0.6, 0.65, 0.7, 0.75, 0.8 ])
```

The optimal values of learning_rate =0.7

gamma

```
test_param_and_plot('xgb_model','gamma', [0.005, 0.01, 0.015, 0.02, 0.025, 0.3 ])
```

The optimal values of gamma lies somewhere between 0 and unbounded.

Putting obtained optimal Value

XGBRegressor with customized hyperparameters based on our obtained optimal hyperpraams. Others parameters remian default.

```
xgb_model_h, train_rmse_h, val_rmse_h =  train_and_evaluate('xgb_model',x_train, x_targ
                                                            n_estimators=300,
                                                            max_depth=7,
                                                            learning_rate =0.7,
                                                            colsample_bytree =0.7
```

```
                                                                    )
```

```
xgb_model_h, train_rmse_h, val_rmse_h
```

```
(XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.7, max_delta_step=0,
              max_depth=7, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=300, n_jobs=-1,
              num_parallel_tree=1, predictor='auto', random_state=10,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None),
 8686.545056626579,
 9366.31307929354)
```

```
train_rmse_h, val_rmse_h
```

```
(8686.545056626579, 9366.31307929354)
```

Improvement

```
xgb_val_score_change = xgb_model_h.score(val_train, val_target)/df_errScore[df_errScore
print('rf_val score impovement %.2f%%:' %(xgb_val_score_change*100))
```

```
rf_val score impovement -0.98%:
```

XGBRegressor Default parameter shows good result

> - Or it my be required more tunning with others hyperparameters.

Feature Importance

```
pd.DataFrame({
    'Feature' :num_cols+encod_cols,
    'Value' :xgb_model_h.feature_importances_
}).sort_values('Value' ,ascending= False).head(10)
```

	Feature	Value
10	Location_Type_L2	0.349729
19	Discount_Yes	0.267717
9	Location_Type_L1	0.081921
8	Store_Type_S4	0.062060
5	Store_Type_S1	0.045749
0	Holiday	0.041856

	Feature	Value
4	WeekOfYear	0.024866
14	Region_Code_R1	0.019009
3	Month	0.017204
11	Location_Type_L3	0.015694

Test Model with single dataset

Using a helping Function

```
def test_single_data(ind, mod):
    model=mod
    data = val_train[num_cols+encod_cols].iloc[[ind]]
    actual_sales = val_target.iloc[ind]
    pred_sales = mod.predict(data)
    print('Actual Sales:', actual_sales, ', Predicted Sales:', pred_sales , ', Differen
```

```
#test 01
id =2021
print('XGB Result')
test_single_data(id, xgb_model_h)
print('RandomForests Result')
test_single_data(id, rfReg_model_h)
```

```
XGB Result
Actual Sales: 44757.0 , Predicted Sales: [44983.426] , Difference : [226.42578]
RandomForests Result
Actual Sales: 44757.0 , Predicted Sales: [46131.89714541] , Difference :
[1374.89714541]
```

```
#test 02
id =1000
print('XGB Result')
test_single_data(id, xgb_model_h)
print('RandomForests Result')
test_single_data(id, rfReg_model_h)
```

```
XGB Result
Actual Sales: 25608.0 , Predicted Sales: [26701.242] , Difference : [1093.2422]
RandomForests Result
Actual Sales: 25608.0 , Predicted Sales: [27308.394296] , Difference : [1700.394296]
```

Saving Models

```
import joblib
```

```
sales_forecasting_WOMart = {
    'rfReg_model_h':rfReg_model_h,
    'xgb_model_h': xgb_model_h,
    'encoder': encod_cols,
    'numeric_cols': num_cols,
    'categorical_cols': cat_cols,
    'input_cols': input_cols,
    'imputer': numScal,
    'scaler': catScal,
}
```

```
joblib.dump(sales_forecasting_WOMart, 'sales_forecasting_WOMart.joblib')
```

```
['sales_forecasting_WOMart.joblib']
```

```
# Calling variables
#sales_forecasting_WOMart = joblib.load('sales_forecasting_WOMart.joblib')
```

Summary

This project is covering following topics.

- Selecting real-world dataset from a Kaggle competition.
- Download the data
- Exploratory data analysis and visualisation
- Preprocessing and Feature Engineering
- Scaling numerical data
- Encoding categorical data
- Spliting Traing and Validation Data set
- Training and evaluating XGBRegressor, RandomForestRegression, ElasticNet, Ridge and Lasso Regression Model
- Tunning with hyperparameter and evaluate the RandomForestRegression Model
- Tunning with hyperparameter and evaluate the XGBRegressor Model
- Making Predictions and evaluate with the Single Inputs
- Saving and Loading Trained Model

References

- https://jovian.ai/learn/machine-learning-with-python-zero-to-gbms
- https://jovian.ai/aakashns/interactive-visualization-plotly#C83
- https://stackoverflow.com/questions

Special thanks to Mr. Aakash and Jovian Team

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting
```

'https://jovian.ai/salirenata/real-world-machine-learning-model-sales-forecasting'

```

```