SALES FORECASTING

Dependent variable: **Item\_Outlet\_Sales**

**Import Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import os

import warnings

warnings.filterwarnings('ignore')

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 15, 6

**Check the current working directory**

display (os.getcwd())

**Change the current working directory**

os.chdir ('C:\\Noble\\Training\\Acmegrade\\Data Science\\Projects\\PRJ Sales Forecasting\\')

display (os.getcwd())

**Read and display the data set**

dt = pd.read\_csv('Train.csv')

display (dt.head())

**Display the shape**

print (dt.shape)

**Display the column names**

display (dt.columns)

**Describe the column**

display (dt.describe())

**Display Info**

display (dt.info())

**Display the Unique Values for each column**

display (dt.apply(lambda x: len(x.unique())))

**Check for Null Values**

display (dt.isnull().sum())

**Store the Categorical columns in a list**

cat\_col = []

for x in dt.dtypes.index:

if dt.dtypes[x] == 'object':

cat\_col.append(x)

display (cat\_col)

**Delete the columns**

cat\_col.remove('Item\_Identifier')

cat\_col.remove('Outlet\_Identifier')

display (cat\_col)

**Display the Unique Values in category columns – Count**

for col in cat\_col:

print(col , len(dt[col].unique()))

**Unique values in each category**

for col in cat\_col:

print(col)

print(dt[col].value\_counts())

print()

print ('\*' \*50)

**Display the missing values - missing values will be True**

miss\_bool = dt['Item\_Weight'].isnull()

display (miss\_bool)

**Missing value count - column - Item\_Weight**

display (dt['Item\_Weight'].isnull().sum())

**Display all NULL Records**

Item\_Weight\_null = dt[dt['Item\_Weight'].isna()]

display (Item\_Weight\_null)

**NULL Records by Item Identifier column**

Item\_Weight\_null['Item\_Identifier'].value\_counts()

**Find the mean for the column – Item Weight group by Item Identifier**

item\_weight\_mean = dt.pivot\_table(values = "Item\_Weight", index = 'Item\_Identifier')

display (item\_weight\_mean)

**Display Item Identifier column**

display (dt['Item\_Identifier'])

**Fill the missing values (Item Weight) with mean, the mean calculated by group by Item identifier**

for i, item in enumerate(dt['Item\_Identifier']):

if miss\_bool[i]:

if item in item\_weight\_mean.index:

dt['Item\_Weight'][i] = item\_weight\_mean.loc[item]['Item\_Weight']

else:

dt['Item\_Weight'][i] = np.mean(dt['Item\_Weight'])

**Check the Null values again – Same column**

result = dt['Item\_Weight'].isnull().sum()

display (result)

**Record count based on 'Outlet\_Size'**

result = dt.groupby('Outlet\_Size').agg({'Outlet\_Size': np.size})

display (result)

**NULL Record based on 'Outlet\_Size'**

result= dt['Outlet\_Size'].isnull().sum()

display (result)

**Display all NULL Records**

Outlet\_Size\_null= dt[dt['Outlet\_Size'].isna()]

display (Outlet\_Size\_null)

**Null Record count based on -Outlet Type**

result = Outlet\_Size\_null['Outlet\_Type'].value\_counts()

display (result)

**Group by based on Outlet\_Type and Outlet\_Size to find the most repeated value, this is to fill missing value by Outlet Type**

result= dt.groupby (['Outlet\_Type','Outlet\_Size'] ).agg({'Outlet\_Type':[np.size]})

display (result)

**Alternate way to identify most repeated value – Mode**

outlet\_size\_mode = dt.pivot\_table(values='Outlet\_Size', columns='Outlet\_Type', aggfunc=(lambda x: x.mode()[0]))

display (outlet\_size\_mode)

**Use Mode to fill missing values**

miss\_bool = dt['Outlet\_Size'].isnull()

dt.loc[miss\_bool, 'Outlet\_Size'] = dt.loc[miss\_bool, 'Outlet\_Type'].apply(lambda x: outlet\_size\_mode[x])

**Option -1 Equivalent For loop for the above statement**

miss\_bool = dt['Outlet\_Size'].isnull()

for i, item in enumerate (dt['Outlet\_Size']):

if miss\_bool[i]:

dt['Outlet\_Size'][i] = outlet\_size\_mode.loc['Outlet\_Size',dt['Outlet\_Type'][i] ]

**Option -2 Equivalent For loop for the above statement using simple imputer**

import pandas as pd

import numpy as np

import os

import warnings

warnings.filterwarnings('ignore')

os.chdir ('C:\\Noble\\Training\\Acmegrade\\Data Science\\Projects\\PRJ Sales Forecasting\\')

dt = pd.read\_csv('Train.csv')

outlet = dt['Outlet\_Type'].unique()

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most\_frequent')

newdt = pd.DataFrame()

for i in outlet:

dt\_unique = dt[dt['Outlet\_Type']==i]

dt\_unique.iloc[:,8:9]=imputer.fit\_transform (dt\_unique.iloc[:,8:9])

newdt=newdt.append (dt\_unique )

result = dt.groupby (['Outlet\_Type','Outlet\_Size'] ).agg({'Outlet\_Type':[np.size]})

display (result)

result = newdt.groupby (['Outlet\_Type','Outlet\_Size'] ).agg({'Outlet\_Type':[np.size]})

display (result)

**Check the Null values**

display (dt['Outlet\_Size'].isnull().sum())

**Check the group by count to see if the count increased**

result = dt.groupby (['Outlet\_Type','Outlet\_Size'] ).agg({'Outlet\_Type':[np.size]})

display (result)

**Check Item visibility column with value - 0**

display (sum(dt['Item\_Visibility']==0))

**Replace zeros with mean**

dt.loc[:, 'Item\_Visibility'].replace([0], [dt['Item\_Visibility'].mean()], inplace=True)

**Check any value with 0 again**

sum(dt['Item\_Visibility']==0)

**Check distinct values - Item\_Fat\_Content**

display (sum(dt['Item\_Visibility']==0))

**Consolidate similar Column Values**

dt['Item\_Fat\_Content'] = dt['Item\_Fat\_Content'].replace({'LF':'Low Fat', 'reg':'Regular', 'low fat':'Low Fat'})

result = dt['Item\_Fat\_Content'].value\_counts()

display (result)

Creating New Attributes

**Create new attributes with first two characters of item identifier column**

dt['New\_Item\_Type'] = dt['Item\_Identifier'].apply(lambda x: x[:2])

display (dt['New\_Item\_Type'])

**Display Number of records in each category**

display (dt['New\_Item\_Type'].value\_counts())

**Map the values**

dt['New\_Item\_Type'] = dt['New\_Item\_Type'].map({'FD':'Food', 'NC':'Non-Consumable', 'DR':'Drinks'})

display (dt['New\_Item\_Type'].value\_counts())

**Display distinct values in Item\_Fat\_Content**

display (dt['Item\_Fat\_Content'].value\_counts())

**Display the count based on New\_Item\_Type and Item\_Fat\_Content**

result = dt.groupby (['New\_Item\_Type','Item\_Fat\_Content'] ).agg({'Outlet\_Type':[np.size]})

display (result)

**Update Item\_Fat\_Content to ‘Non Edible’ where New\_Item\_Type = Non-Consumable**

dt.loc[dt['New\_Item\_Type']=='Non-Consumable', 'Item\_Fat\_Content'] = 'Non-Edible'

result = (dt['Item\_Fat\_Content'].value\_counts())

display (result)

**Display the count based on New\_Item\_Type and Item\_Fat\_Content**

result = dt.groupby (['New\_Item\_Type','Item\_Fat\_Content'] ).agg({'Outlet\_Type':[np.size]})

display (result)

**Display how many years the outlet is present**

2022 (Current year) - 'Outlet\_Establishment\_Year'

dt['Outlet\_Years'] = 2022 - dt['Outlet\_Establishment\_Year']

print (dt['Outlet\_Years'])

**Display Top 5 Records**

display (dt.head())

Exploratory Data Analysis

**Create Dist Plot – Item Weight**

sns.distplot(dt['Item\_Weight'])

plt.show()

**Create Dist Plot – Item Visibility**

sns.distplot(dt['Item\_Visibility'])

plt.show()

**Create Dist Plot – Item MRP**

sns.distplot(dt['Item\_MRP'])

plt.show()

**Create Dist Plot – Item Outlet Sales**

sns.distplot(dt['Item\_Outlet\_Sales'])

plt.show()

**Log Transformation to reduce Outliers**

# The above dist plot is right skewed, there might be outliers in the right side. To reduce the outliers, implement log transformation

dt['Item\_Outlet\_Sales'] = np.log(1+dt['Item\_Outlet\_Sales'])

display (dt['Item\_Outlet\_Sales'])

**Create Dist Plot – again**

sns.distplot(dt['Item\_Outlet\_Sales'])

plt.show()

**Create Count Plot – Number of records in each category**

sns.countplot(x = dt["Item\_Fat\_Content"])

plt.show()

**Create Count Plot – Item Type**

# l is the list of unique Item Types - This is used to display X-Label

l = list(dt['Item\_Type'].unique())

chart = sns.countplot(x =dt["Item\_Type"])

chart.set\_xticklabels(labels=l, rotation=90)

plt.show()

**Create Count Plot – Establishment year**

Number of stores started per year

sns.countplot(x= dt['Outlet\_Establishment\_Year'])

plt.show()

**Count Plot Outlet Size**

sns.countplot(x=dt['Outlet\_Size'])

plt.show()

**Count Plot Outlet Location Type**

sns.countplot(x=dt['Outlet\_Location\_Type'])

plt.show()

**Count Plot Outlet Type**

sns.countplot(x= dt['Outlet\_Type'])

plt.show()

Co-relation Matrix

**Print Co relation**

corr = dt.corr()

display (corr)

**Print Co Relation Matrix**

sns.heatmap(corr, annot=True, cmap='coolwarm')

plt.show()

**Display Top 5 Records**

display (dt.head())

Label Encoding

**Label Encoding – Column Outlet Identifier**

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

dt['Outlet'] = le.fit\_transform(dt['Outlet\_Identifier'])

display (dt['Outlet'])

**Label Encoding – Remaining columns with For loop**

cat\_col = ['Item\_Fat\_Content', 'Item\_Type', 'Outlet\_Size', 'Outlet\_Location\_Type', 'Outlet\_Type', 'New\_Item\_Type']

for col in cat\_col:

dt[col] = le.fit\_transform(dt[col])

display (dt.head())

One hot Encoding

dt = pd.get\_dummies(dt, columns=['Item\_Fat\_Content', 'Outlet\_Size', 'Outlet\_Location\_Type', 'Outlet\_Type', 'New\_Item\_Type'])

display (dt.head())

**Create X – Remove un used columns**

X = dt.drop(columns=['Outlet\_Establishment\_Year', 'Item\_Identifier', 'Outlet\_Identifier', 'Item\_Outlet\_Sales'])

X.head()

**Create y**

y = dt['Item\_Outlet\_Sales']

y.head()

**Train Test Split**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

print (X.shape, y.shape)

print (X\_train.shape, X\_test.shape , y\_train.shape, y\_test.shape)

**Function to create Model**

Display all Scoring options

from sklearn.metrics import SCORERS

SCORERS.keys()

Two scoring options used to check model performance

Neg\_mean\_square and R2\_score, Default options is R2 Score

Displaying the absolute of Neg\_mean\_square as mean

A smaller value in abs (Neg\_mean\_square) is better , R2\_Score high value is better

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

def train(model, X, y):

# training the model

model.fit(X, y)

pred = model.predict(X)

# perform cross-validation

cv\_score = cross\_val\_score(model, X, y, scoring='neg\_mean\_squared\_error', cv=5)

print("Model Report")

print ('Scoring - neg\_mean\_squared\_error')

print ( cv\_score )

cv\_score = np.abs(np.mean(cv\_score))

print ('ABS Average of - neg\_mean\_squared\_error',cv\_score )

cv\_score = cross\_val\_score(model, X, y, cv=5)

print ()

print ('R2 Score ')

print ( cv\_score )

cv\_score = np.mean(cv\_score)

print ('Average R2 Score ',cv\_score)

print ()

print ('Accuracy for full Data')

print('R2\_Score:', r2\_score(y,pred))

print ()

**Create Linear Regression Model**

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

model = LinearRegression()

train(model, X\_train, y\_train)

coef = pd.Series(model.coef\_, X.columns).sort\_values()

print (coef)

coef.plot(kind='bar', title="Model Coefficients")

plt.show()

**Create Ridge Regression**

model = Ridge()

train(model, X\_train, y\_train)

coef = pd.Series(model.coef\_, X.columns).sort\_values()

coef.plot(kind='bar', title="Model Coefficients")

plt.show()

**Create Lasso Regression**

model = Lasso()

train(model, X\_train, y\_train)

coef = pd.Series(model.coef\_, X.columns).sort\_values()

coef.plot(kind='bar', title="Model Coefficients")

plt.show()

**Decision Tree Regression**

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

train(model,X\_train, y\_train)

coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False)

coef.plot(kind='bar', title="Feature Importance")

plt.show()

**Random Forest Regression**

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()

train(model, X\_train, y\_train)

coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False)

coef.plot(kind='bar', title="Feature Importance")

plt.show()

**Extra Tree Regression**

from sklearn.ensemble import ExtraTreesRegressor

model = ExtraTreesRegressor()

train(model, X\_train, y\_train)

coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False)

coef.plot(kind='bar', title="Feature Importance")

plt.show()

**LGBMRegressor**

from lightgbm import LGBMRegressor

model = LGBMRegressor()

train(model, X\_train, y\_train)

coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False)

coef.plot(kind='bar', title="Feature Importance")

plt.show()

**XG Boost Regressor**

from xgboost import XGBRegressor

model = XGBRegressor()

train(model, X\_train, y\_train)

coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False)

coef.plot(kind='bar', title="Feature Importance")

plt.show()

**Random Search CV**

from sklearn.model\_selection import RandomizedSearchCV

**Parameters**

max\_features = ['auto', 'sqrt']

max\_depth = [int(x) for x in np.linspace(5, 30, num = 6)]

min\_samples\_split = [2, 5, 10, 15, 100]

min\_samples\_leaf = [1, 2, 5, 10]

**Param Grid**

random\_grid = {

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf}

print(random\_grid)

**Random Forest Regression**

rf = RandomForestRegressor()

rf=RandomizedSearchCV(estimator = rf, param\_distributions = random\_grid,scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, random\_state=42, n\_jobs = 1)

display (rf.fit(X\_train, y\_train))

**Best Parameters**

print(rf.best\_params\_)

print(rf.best\_score\_)

predictions=rf.predict(X\_test)

display (r2\_score (y\_test,predictions))

display (predictions)

**Create the Dist plot**

sns.distplot(y\_test-predictions)

plt.show()

**Parameter for LGBM Regressor**

from scipy.stats import uniform, randint

params = {

"gamma": uniform(0, 0.5),

"learning\_rate": uniform(0.03, 0.3), # default 0.1

"max\_depth": randint(2, 6), # default 3

"n\_estimators": randint(100, 150), # default 100

"subsample": uniform(0.6, 0.4)

}

**Model LGBM Regressor**

lgb=LGBMRegressor()

lgb = RandomizedSearchCV(estimator = lgb, param\_distributions = params,scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, random\_state=42, n\_jobs = 1)

lgb.fit(X,y)

**Best Parameter**

print(lgb.best\_params\_)

print(lgb.best\_score\_)

predictions=lgb.predict(X\_test)

display (r2\_score (y\_test,predictions))

display (predictions)

**Create Dist Plot**

sns.distplot(y\_test-predictions)

plt.show()

**Model XG Boost**

params = {

"gamma": uniform(0, 0.5),

"learning\_rate": uniform(0.03, 0.3), # default 0.1

"max\_depth": randint(2, 6), # default 3

"n\_estimators": randint(100, 150), # default 100

"subsample": uniform(0.6, 0.4)

}

**XG Boost Regressor**

xgb = RandomizedSearchCV(estimator = model, param\_distributions = params,scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, random\_state=42, n\_jobs = 1)

xgb.fit(X,y)

**Print Best Parameter**

print(xgb.best\_params\_)

print(xgb.best\_score\_)

predictions=xgb.predict(X\_test)

display (r2\_score (y\_test,predictions))

display (predictions)

**Create Dist plot**

sns.distplot(y\_test-predictions)

plt.show()