5 SEM PROJECT REPORT

PROJECT REPORT

PREPARED BY
SHIVANSH SINGH BHADOURIA

TOPIC
GENAPCT - FOOD DEMAND FORECASTING

OCTOBER 2020

ABOUT

We have been provided with historical data of Food demand on weekly basis of a food supplier (Genpact).
Our Goal is to Forecast the future demand of food (Number of orders) ,so that the company can maintain its stock at their respective centres and thus preventing the wastage of food and decreasing losses.

UNDERSTANDING THE DATA

We have been given 3 .CSV files:

- **Train.csv** - Contains the historical demand data for all centres. test.csv contains all the following features except the target variable

1. **id** - Unique ID
2. **week** - Week No
3. **center_id** - Unique ID for fulfillment center
4. **meal_id** - Unique ID for Meal
5. **checkout_price** - Final price including discount, taxes & delivery charges
6. **base_price** - Base price of the meal
7. **emailer_for_promotion** - Emailer sent for promotion of meal
8. **homepage_featured** - Meal featured at homepage
9. **num_orders** - Target VAR

- **fulfilment_center_info.csv**: Contains information for each fulfilment center. fulfilment_center_info.csv contains all the following features .

    1. **center_id** - Unique ID for fulfillment center
    2. **city_code** - Unique code for city
    3. **region_code** - Unique code for region
    4. **center_type** - Anonymized center type
    5. **op_area** - Area of operation (in km^2)

- **meal_info.csv**: Contains information for each meal being served. meal_info.csv contains all the following features.

    1. **meal_id** - Unique ID for the meal
    2. **category** - Type of meal (beverages/snacks/soups....)
    3. **cuisine** - Meal cuisine (Indian/Italian/...)

MERGING THE DATA SETS

As we have 3 diffrent Data sets , so in order to make use of all the data we ought to merge the three data sets togeother using pd.merge() :

```
def join(data_1 , data_2 , common_feature):
    train = pd.merge(left = data_1 , right = data_2 ,
    how = "left" , left_on = common_feature ,
    right_on = common_feature)
    return train
```

DATA PREPROCESSING

Now , we have a combined data set on which we can finally start working:

Checking Null Values :

First of all we have checked for Null values in our dataset by using isnull() function . We did not find any blank columns or Null values on Checking:

```
train.isnull().sum()
```

We did not find any blank columns or Null values on Checking.

Indexing :

Setting ' id ' as the Index usng set_index() function:

```
def set_index(train):
    train.set_index('id' ,inplace =True)
set_index(train)
```

Encoding String Values :

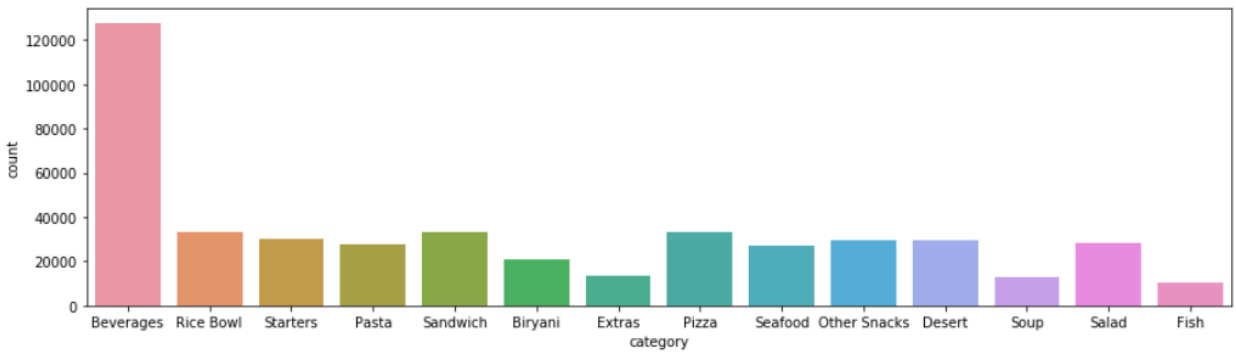
We have implemented the get_dummies function on three features on our train data set i.e: 'category' , 'cuisine' and 'center_type':

But why have we encoded these features and created dummies? Well lets have a look at their Values:

countplot for 'category'

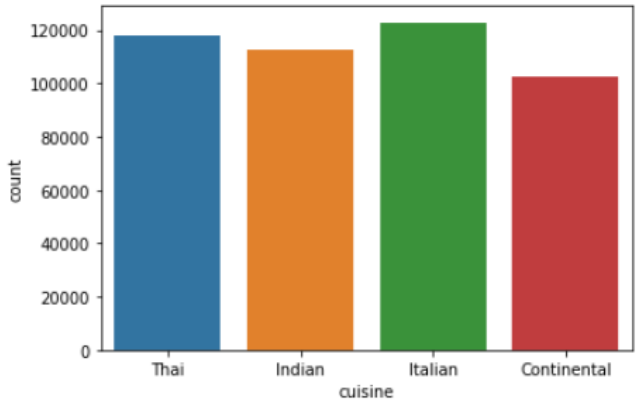
```
plt.figure(figsize=(15,4))
sns.countplot(train['category'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bdf5f5fa88>
```



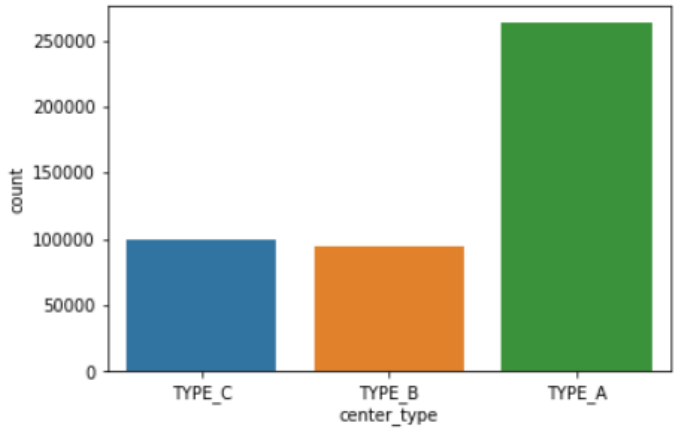
countplot for 'cuisine'

```
In [70]: sns.countplot(train['cuisine'])
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x150189e9b88>
```



countplot for 'center_type'

```
In [21]: sns.countplot(train['center_type'])
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x233cfce7408>
```



As we can see that these features contain categorical Values and one of them i.e 'category' has many categories ,so it was necessary to create dummies for it. Talking of the other two features i.e 'cuisine' and 'centre_type' those are also categorical features and implementing Label encoder instead of dummies would not have been a wise move as it would have affected the accuracy of our model and moreover the Algorithm would have treated the encoded values i.e 0,1,2,3.... with respect to their values and not as different type of cuisine or center_types.

```
def process_catagorical_vars(train):
    cat_dummies = pd.get_dummies(train['category'] , drop_first=True)
    cuisine_dummies = pd.get_dummies(train['cuisine'] , drop_first=True)
    center_type_dummies = pd.get_dummies(train['center_type'] , drop_first=True)

    train.drop(['category','cuisine','center_type'], axis =1 , inplace =True)

    return pd.concat([train , cat_dummies , cuisine_dummies ,center_type_dummies] , axis = 1 )
```

```
train = process_catagorical_vars(train)
```

Dropping Irrelevant Features :

For Dropping irrelevant features we will use pearson correlation to check the correlation of target variable with all the features. We will use train.corr() function for the same:

```
corr = train.corr()
```

```
corr
```

	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	city_code
week	1.000000	-0.003450	0.019814	0.026581	0.028614	-0.000841	-0.008263	-0.017210	0.000405
center_id	-0.003450	1.000000	0.009893	0.001348	0.000604	0.013658	-0.005043	-0.053035	0.061078
meal_id	0.019814	0.009893	1.000000	0.010748	0.002605	0.013402	0.016354	0.010597	-0.003198
checkout_price	0.026581	0.001348	0.010748	1.000000	0.953389	0.004818	-0.057184	-0.282108	-0.004805
base_price	0.028614	0.000604	0.002605	0.953389	1.000000	0.171173	0.057156	-0.222306	-0.002054
emailer_for_promotion	-0.000841	0.013658	0.013402	0.004818	0.171173	1.000000	0.390534	0.277147	-0.005234
homepage_featured	-0.008263	-0.005043	0.016354	-0.057184	0.057156	0.390534	1.000000	0.294490	0.008640
num_orders	-0.017210	-0.053035	0.010597	-0.282108	-0.222306	0.277147	0.294490	1.000000	0.041596
city_code	0.000405	0.061078	-0.003198	-0.004805	-0.002054	-0.005234	0.008640	0.041596	1.000000
region_code	0.004600	-0.003426	-0.001662	-0.003648	-0.001934	-0.007462	0.003605	0.029744	0.042686
op_area	0.001550	-0.111869	-0.001546	0.021569	0.018031	-0.019462	0.041498	0.176976	0.131476

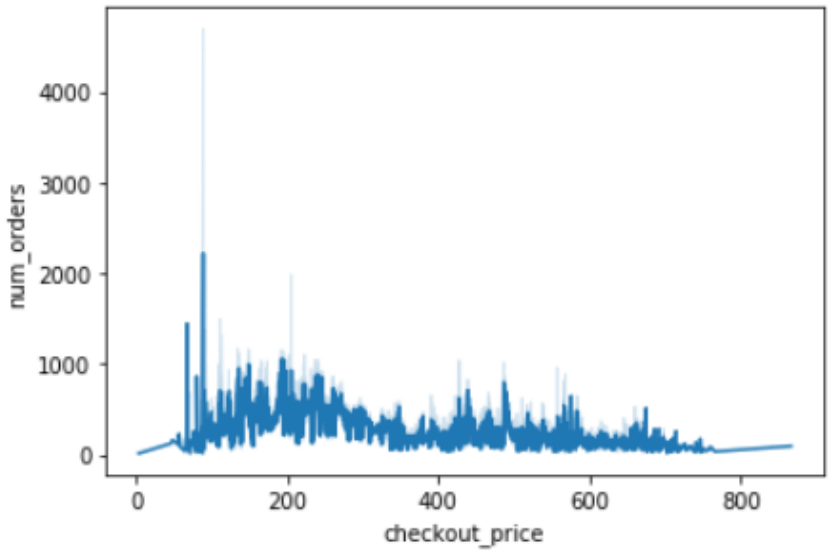
Here we can see that almost all the features have decent correlation with the target variable except one feature i.e 'base_price' , hence we will drop it , but we also see that some features also have negative correlation with the target Variable ,but why didn't we drop them? Well, as we know that correlation is just a mathematical computation it takes values as input and gives output accordingly but it cannot

understand the fundamental importance of the feature ,so we will not just blindly drop the features with negative or less Correlation. Secondly there are also some categorical features so it is obvious that the correlation will come out to be negative , hence we will also not drop them.

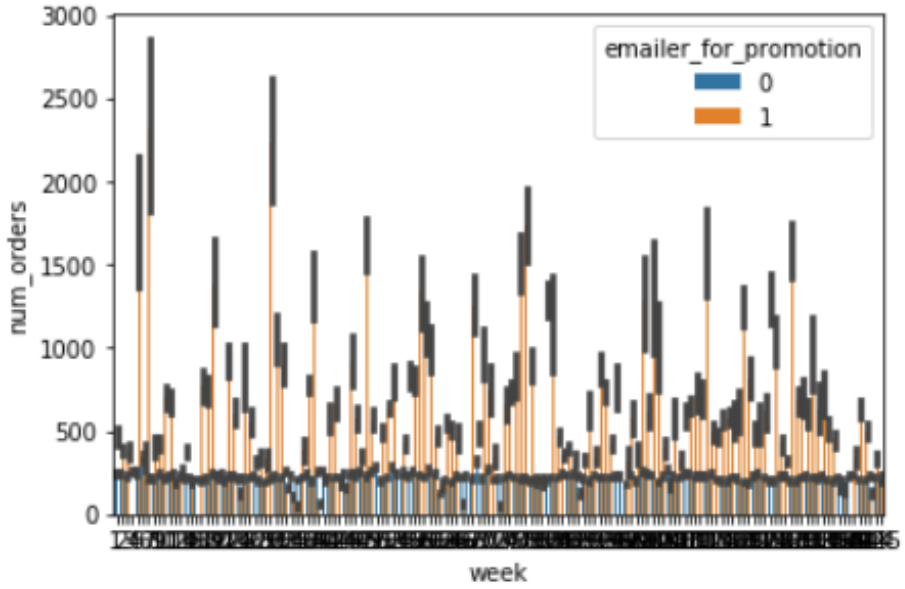
```
def dropping_col(train,col_to_drop):
    train.drop(col_to_drop,inplace = True, axis = 1 )

dropping_col(train,'base_price')
```

Exp: Relation of 'checkout_price' with tagret variable checkout price is highly correlated with num_orders , when the checkout price is low ,the num_orders are higher and vice-versa



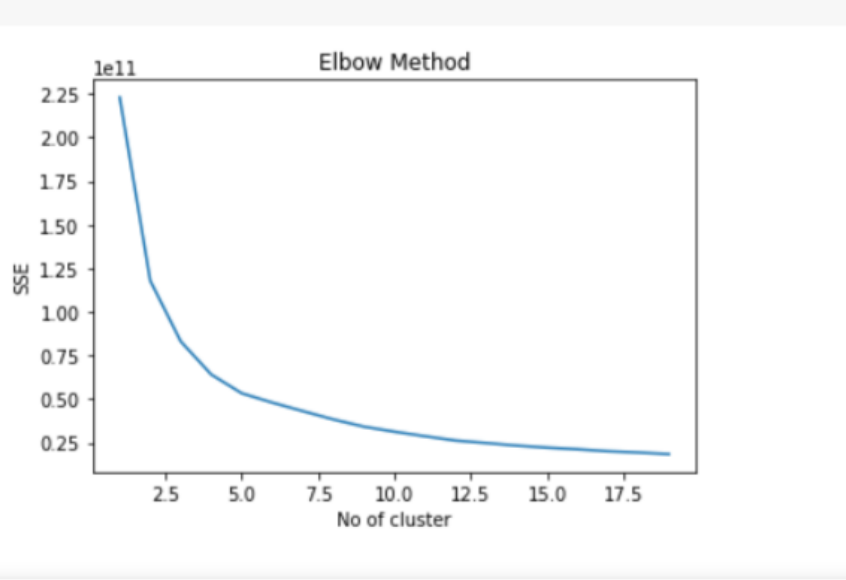
Exp: Relation of 'emailer_for_promotion' with tagret variable w.r.t "week : this plot shows that the no. of orders were significntly higher when an email was sent for promotion



Creating clusters (KMean) :

We have created clusters / Labels using the KMeans technique as it will help our machine Learning model to understand the data better . KMeans algorithm divides the data point into clusters and generates labels . The number of clusters is to be decided by the Data Scientist / Coder him/her-self.

We can use a test called the Elbow test to determine the no. of clusters it will be suitable to make



Elbow Test indicated we should go with 5 Clusters

```
In [55]: from sklearn.cluster import KMeans

In [56]: kmeans = KMeans(n_clusters=5)

In [57]: kmeans.fit(train)

Out[57]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
                n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=None, tol=0.0001, verbose=0)

In [58]: train['labels']=kmeans.labels_
```

Splitting Data into Training set and validation set:

We need to split our Data set into x_train , y_train and x_val and y_val:

x_train will contain 80% of Original Data and all the features except our target Variable.
y_train will contain only target Variable to check accuracy.
x_val will contain 20% of our Original Data set and all the features except our target Variable.
y_val will contain only target Variable to check accuracy

```
def train_val_split(data):
    train,val= train_test_split(data,test_size =0.2, random_state=112)
    return train,val

train,val = train_val_split(train)
```

```
def x_y_split(data,target_var):
    x = data.drop(target_var , axis = 1 )
    y = data[target_var]
    return x , y

x_train , y_train = x_y_split(train,'num_orders')
x_val , y_val = x_y_split(val,'num_orders')
```

TESTING ACCURACY OF THE MODEL

- **Which Machine Learning Model should i use?**

- Our Target variable i.e 'num_orders' is found to be continuous and hence we will use a regression model.

- **Ok , but what kind of regression model should i use?**

  -We will use a Random forest regressor Model as the datapoints are scattered in a large area and using simple regression that plots the best fit "Straight Line" will "***not***" be usefull .

- **How will we evaluate the accuracy of our model**

  We will evaluate our model by using RMSLE value.

- **RMSLE Value (Train Dataset)**

```
pred = reg.predict(x_train)
```

```
sqrt(mean_squared_log_error( y_train, pred ))
```

```
0.22644210667925158
```

- **Validating on Validation Data set**

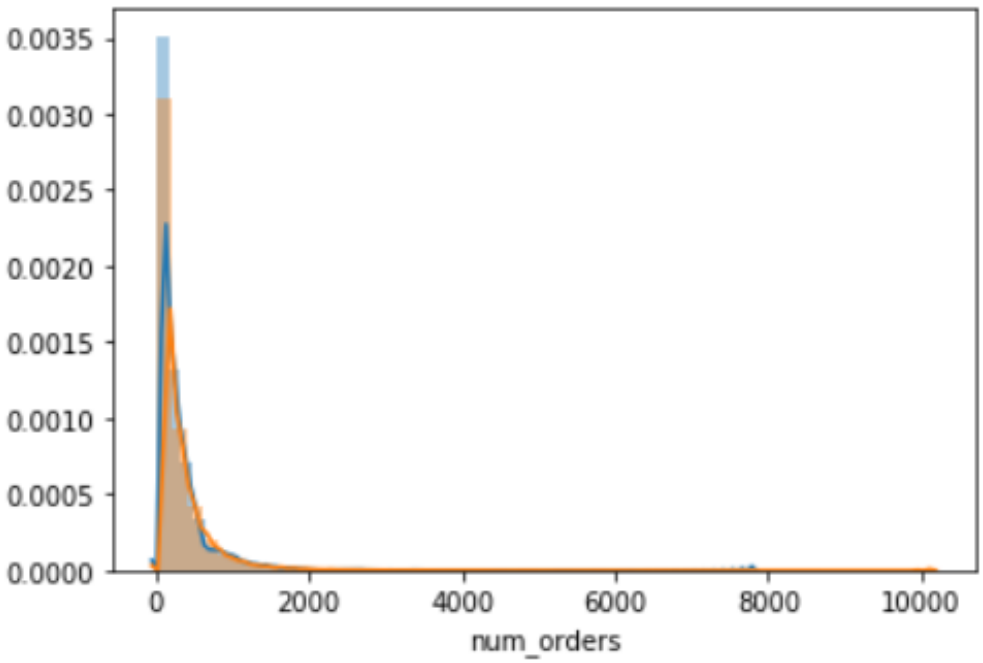
RMSLE for Validation dataset

```
sqrt(mean_squared_log_error( y_val, pred_2 ))
```

```
0.24643465736977352
```

```
sns.distplot(pred_2) , sns.distplot(val['num_orders'])
```

```
(<matplotlib.axes._subplots.AxesSubplot at 0x21a8238bd88>,
 <matplotlib.axes._subplots.AxesSubplot at 0x21a8238bd88>)
```



BLUE : Predicted num or orders
ORANGE : Original num of orders

5 SEM PROJECT REPORT

THANK YOU

PREPARED BY
SHIVANSH SINGH BHADOURIA

TOPIC
GENAPCT - FOOD DEMAND FORECASTING

OCTOBER 2020