RESTAURANT REVENUE PREDICTION

Introduction

Project Overview

This project is one project in Kaggle which help TFI deciding when and where to open new restaurants. We choose this project is beacuse it is more pratical in a real business. We want to know how the different models we learn perform in this real case and practice what we learned from the course. We choose 5 models for it: Random Forest Regressor, KNN Regressor, Lasso, Ridge and XGB Regressor. Four of them we learned from the class, and one is new to us. We will use them to compare the data results and try to figure out which variables are the most important for forecasting revenue and provide an overview of our best performing models by root mean sqaure error.

Data Description

We'll train our model with a data set which has 43 variables and 137 transactions record, which is small size. As describe in the data set, revenue is the response variable and it's indicates revenue of the restaurant in a given year. The values are transformed so they don't mean real dollar values. In the remaining variables,P1-P37 are anonymized data and we have 4 categorical vairables and one date vairable which we need convert later.

Starter Code

Loading Package

```
# General
import numpy as np
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder


# Modeling
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Ridge, Lasso
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

Loading Data

```
df= pd.read_csv('/content/RestaurantRevenue.csv')
```

Data Pre-processing

Data Overview

```
df.shape
```

```
(137, 43)
```

```python
df.head()
```

	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	...	P29	P30	P31	P32	P33	P34	P35	P36	P37	revenue
0	0	07/17/1999	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	...	3.0	5	3	4	5	5	4	3	4	5653753.0
1	1	02/14/2008	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	...	3.0	0	0	0	0	0	0	0	0	6923131.0
2	2	03/09/2013	Diyarbakır	Other	IL	2	4.0	2.0	5.0	2	...	3.0	0	0	0	0	0	0	0	0	2055379.0
3	3	02/02/2012	Tokat	Other	IL	6	4.5	6.0	6.0	4	...	7.5	25	12	10	6	18	12	12	6	2675511.0
4	4	05/09/2009	Gaziantep	Other	IL	3	4.0	3.0	4.0	2	...	3.0	5	1	3	2	3	4	3	3	4316715.0

5 rows × 43 columns

```python
df.describe()
```

	Id	P1	P2	P3	P4	P5	P6	P7	P8	P9
count	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000

```
df.columns
```

```
Index(['Id', 'Open Date', 'City', 'City Group', 'Type', 'P1', 'P2', 'P3', 'P4',
       'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15',
       'P16', 'P17', 'P18', 'P19', 'P20', 'P21', 'P22', 'P23', 'P24', 'P25',
       'P26', 'P27', 'P28', 'P29', 'P30', 'P31', 'P32', 'P33', 'P34', 'P35',
       'P36', 'P37', 'revenue'],
      dtype='object')
```

75%	102.000000	4.000000	5.000000	5.000000	5.000000	2.000000	4.000000	5.000000	5.000000	5.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137 entries, 0 to 136
Data columns (total 43 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Id          137 non-null    int64
 1   Open Date   137 non-null    object
 2   City        137 non-null    object
 3   City Group  137 non-null    object
 4   Type        137 non-null    object
 5   P1          137 non-null    int64
 6   P2          137 non-null    float64
 7   P3          137 non-null    float64
 8   P4          137 non-null    float64
 9   P5          137 non-null    int64
 10  P6          137 non-null    int64
 11  P7          137 non-null    int64
 12  P8          137 non-null    int64
 13  P9          137 non-null    int64
 14  P10         137 non-null    int64
 15  P11         137 non-null    int64
 16  P12         137 non-null    int64
 17  P13         137 non-null    float64
 18  P14         137 non-null    int64
```

```
 19  P15         137 non-null    int64  
 20  P16         137 non-null    int64  
 21  P17         137 non-null    int64  
 22  P18         137 non-null    int64  
 23  P19         137 non-null    int64  
 24  P20         137 non-null    int64  
 25  P21         137 non-null    int64  
 26  P22         137 non-null    int64  
 27  P23         137 non-null    int64  
 28  P24         137 non-null    int64  
 29  P25         137 non-null    int64  
 30  P26         137 non-null    float64
 31  P27         137 non-null    float64
 32  P28         137 non-null    float64
 33  P29         137 non-null    float64
 34  P30         137 non-null    int64  
 35  P31         137 non-null    int64  
 36  P32         137 non-null    int64  
 37  P33         137 non-null    int64  
 38  P34         137 non-null    int64  
 39  P35         137 non-null    int64  
 40  P36         137 non-null    int64  
 41  P37         137 non-null    int64  
 42  revenue     137 non-null    float64
dtypes: float64(9), int64(30), object(4)
memory usage: 46.1+ KB
```

As we can see, there is no missing value in the dataset. We don't need to use the feature 'ID' as it's reduant and not going to give any insight of the revenue. So we will simply drop it later. Features 'Open date','City', 'City Group' and 'Type' are object type which need to be converted into machine-readable type later.

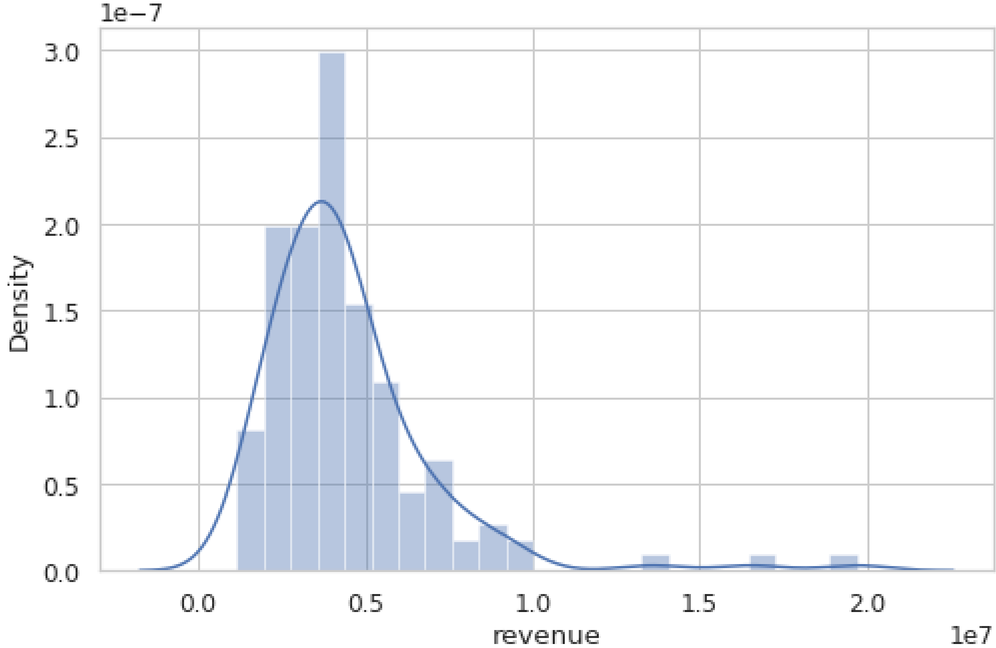
Data Analysis & Preprocessing

Target Column - 'revenue'

Check the target column distribution ['revenue']

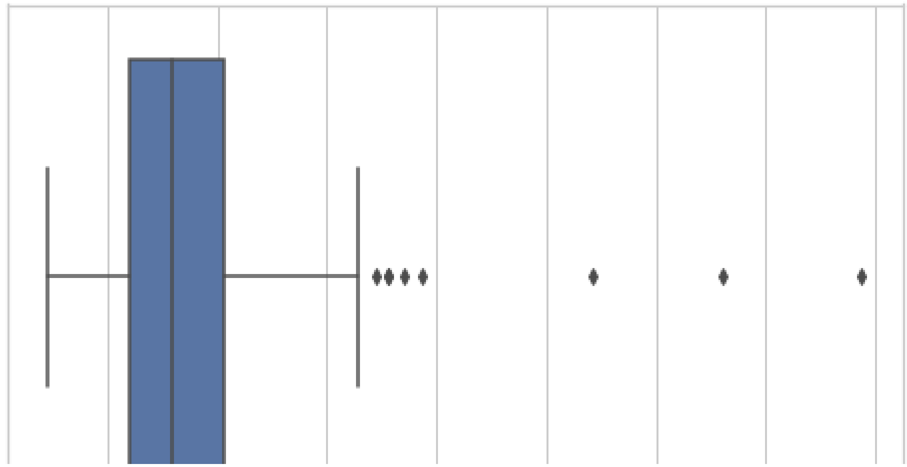
```
sns.distplot(df.revenue)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f842c54cfa0>
```




```
sns.boxplot(df.revenue)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From versi
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f842c6e5fd0>
```



From the above plots, we can see the revenue is drawn from a normal distribution(with a little bit skew) and there are some outliers in revenue. So we want to drop some outliers here.

revenue 1e7

```
df = df[df['revenue']<8e+06].copy()
df.shape
```

```
(128, 43)
```

Remove unnecessery column - 'ID'

```
df=df.drop('Id',axis=1)
```

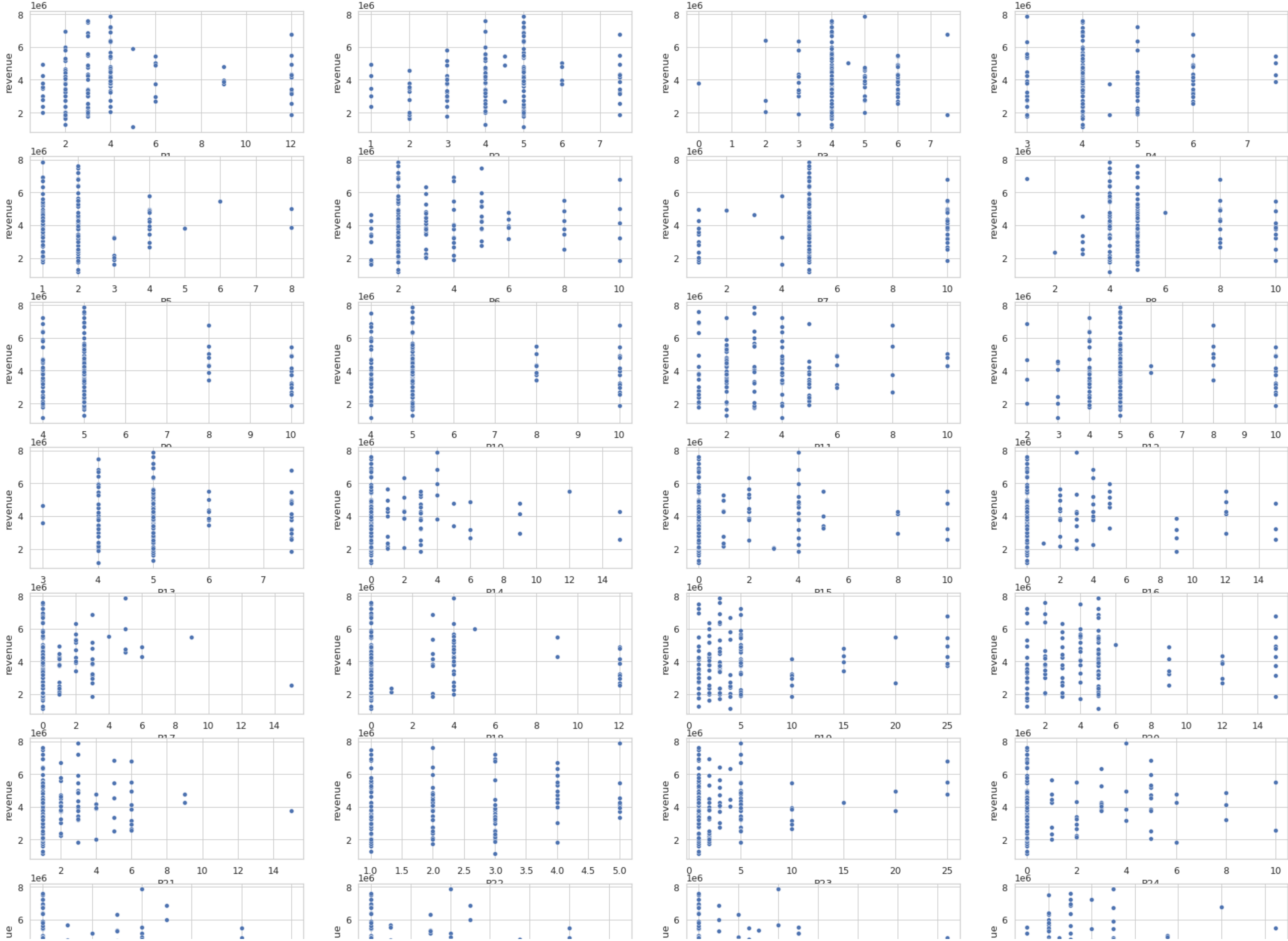
Numerical Features

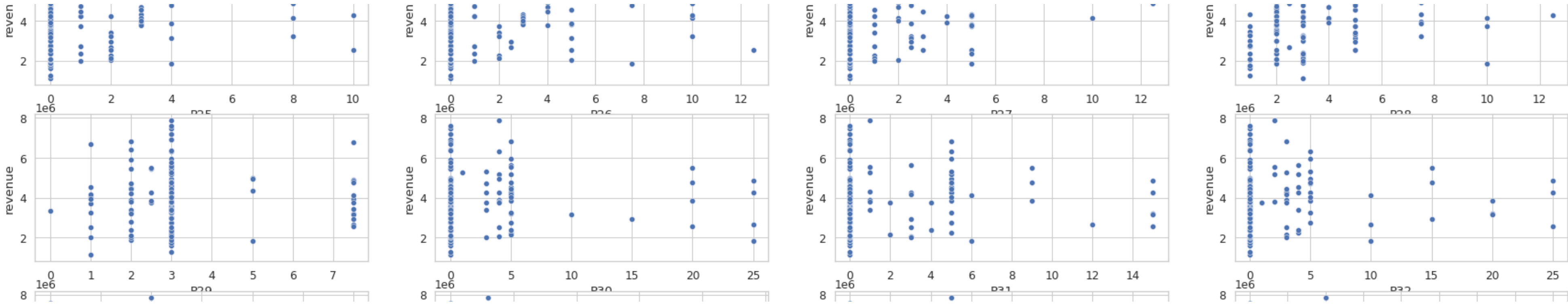
```
numerical_features = df.select_dtypes([np.number]).columns.tolist()
```

```
numerical_features

     ['P1',
      'P2',
      'P3',
      'P4',
      'P5',
      'P6',
      'P7',
      'P8',
      'P9',
      'P10',
      'P11',
      'P12',
      'P13',
      'P14',
      'P15',
      'P16',
      'P17',
      'P18',
      'P19',
      'P20',
      'P21',
      'P22',
      'P23',
      'P24',
      'P25',
      'P26',
      'P27',
      'P28',
      'P29',
      'P30',
      'P31',
      'P32',
      'P33',
      'P34',
      'P35',
      'P36',
      'P37',
      'revenue']
```

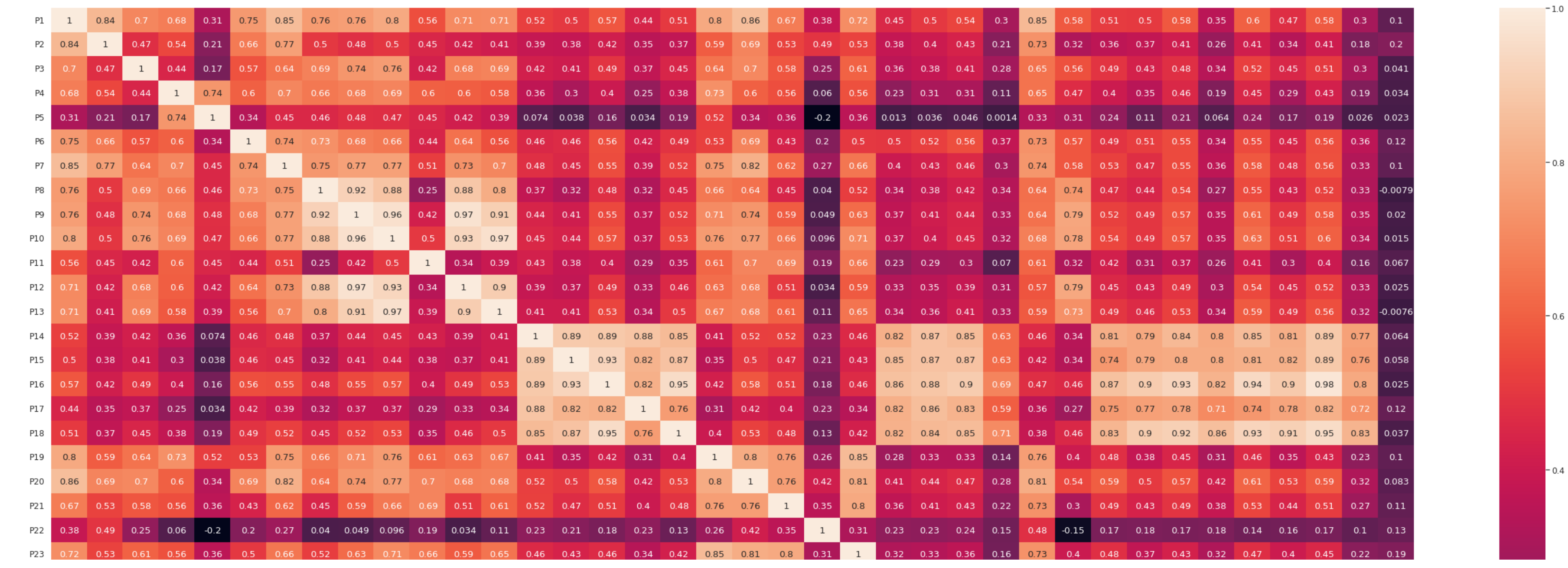
```
fig, ax = plt.subplots(10, 4, figsize=(30, 35))
for variable, subplot in zip(numerical_features, ax.flatten()):
    sns.scatterplot(x=df[variable], y=df['revenue'], ax=subplot)
```


1e6
revenue




```
plt.figure(figsize=(45,25))
sns.heatmap(df.corr(),annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f842ba45520>



As the sctter plot and corrleation matrix above, there is no obvious corrleation in numerical features

Date Features

Date does not give us any insight of the revenue. However we can extract month and year from date to see if there is relationship between them with revenue. First we convert the 'Open Date' feature in datetime format and then we extract the month and year from it to see how is the distribution of them.

```
df['Open  Date']=pd.to_datetime(df['Open  Date'])
df['Month']=[x.month  for  x  in  df['Open  Date']]
df['Year']=[x.year  for  x  in  df['Open  Date']]
```

```
df=df.drop(['Open  Date'],axis=1)
df
```

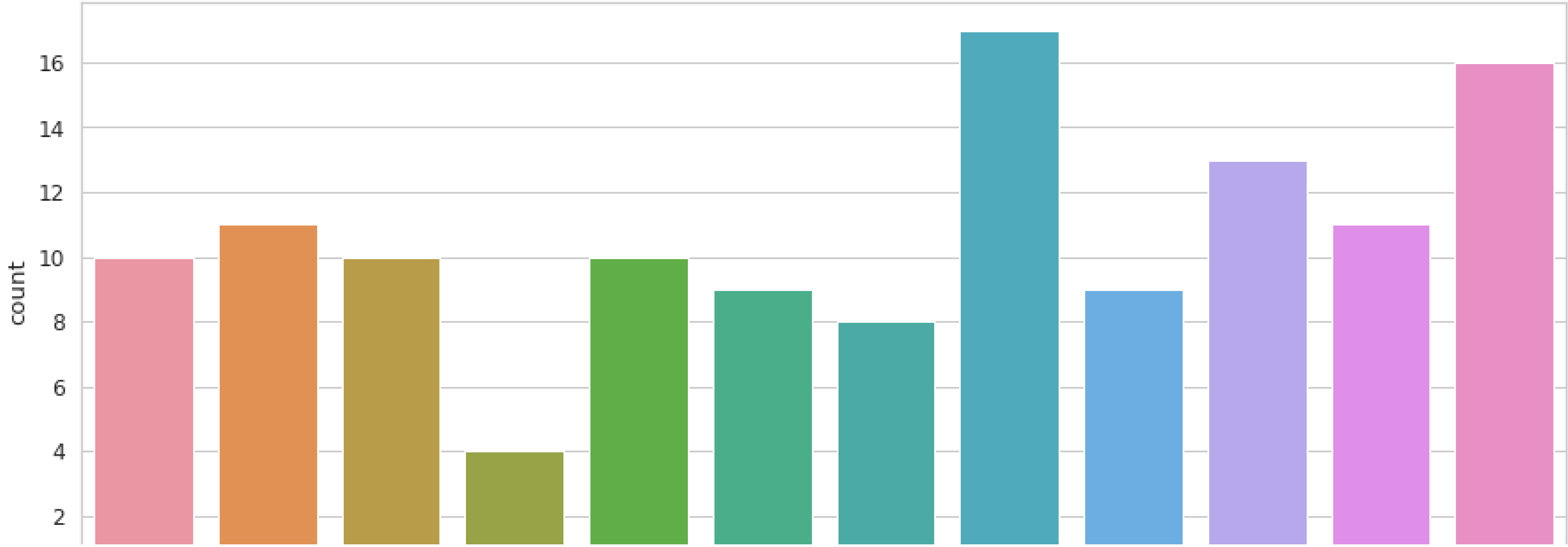
	City	City Group	Type	P1	P2	P3	P4	P5	P6	P7	...	P31	P32	P33	P34	P35	P36	P37	revenue	Month	Year
0	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	2	5	...	3	4	5	5	4	3	4	5653753.0	7	1999
1	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	2	5	...	0	0	0	0	0	0	0	6923131.0	2	2008
2	Diyarbakır	Other	IL	2	4.0	2.0	5.0	2	3	5	...	0	0	0	0	0	0	0	2055379.0	3	2013
3	Tokat	Other	IL	6	4.5	6.0	6.0	4	4	10	...	12	10	6	18	12	12	6	2675511.0	2	2012
4	Gaziantep	Other	IL	3	4.0	3.0	4.0	2	2	5	...	1	3	2	3	4	3	3	4316715.0	5	2009
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
131	Ankara	Big Cities	FC	3	4.0	4.0	5.0	3	4	5	...	0	0	0	0	0	0	0	3199619.0	11	2002
132	Trabzon	Other	FC	2	3.0	3.0	5.0	4	2	4	...	0	0	0	0	0	0	0	5787594.0	6	2008
134	Kayseri	Other	FC	3	4.0	4.0	4.0	2	3	5	...	0	0	0	0	0	0	0	2544857.0	7	2006
135	İstanbul	Big Cities	FC	4	5.0	4.0	5.0	2	2	5	...	0	0	0	0	0	0	0	7217634.0	10	2010
136	İstanbul	Big Cities	FC	4	5.0	3.0	5.0	2	2	5	...	0	0	0	0	0	0	0	6363241.0	9	2009

128 rows × 43 columns

Now let's try to visualize the trends in month and year to understand how they affect the revenue

```
plt.figure(figsize=(15,6))
sns.countplot(df['Month'])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From versi
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f842b79e070>
```



From the above plot we can look at the occurence of various months in the dataset. We have the most data for the last 5 months. The highest of them is from August and December. Now let's see in which month did we have the most revenue.

```
df.groupby('Month')['revenue'].mean()
```

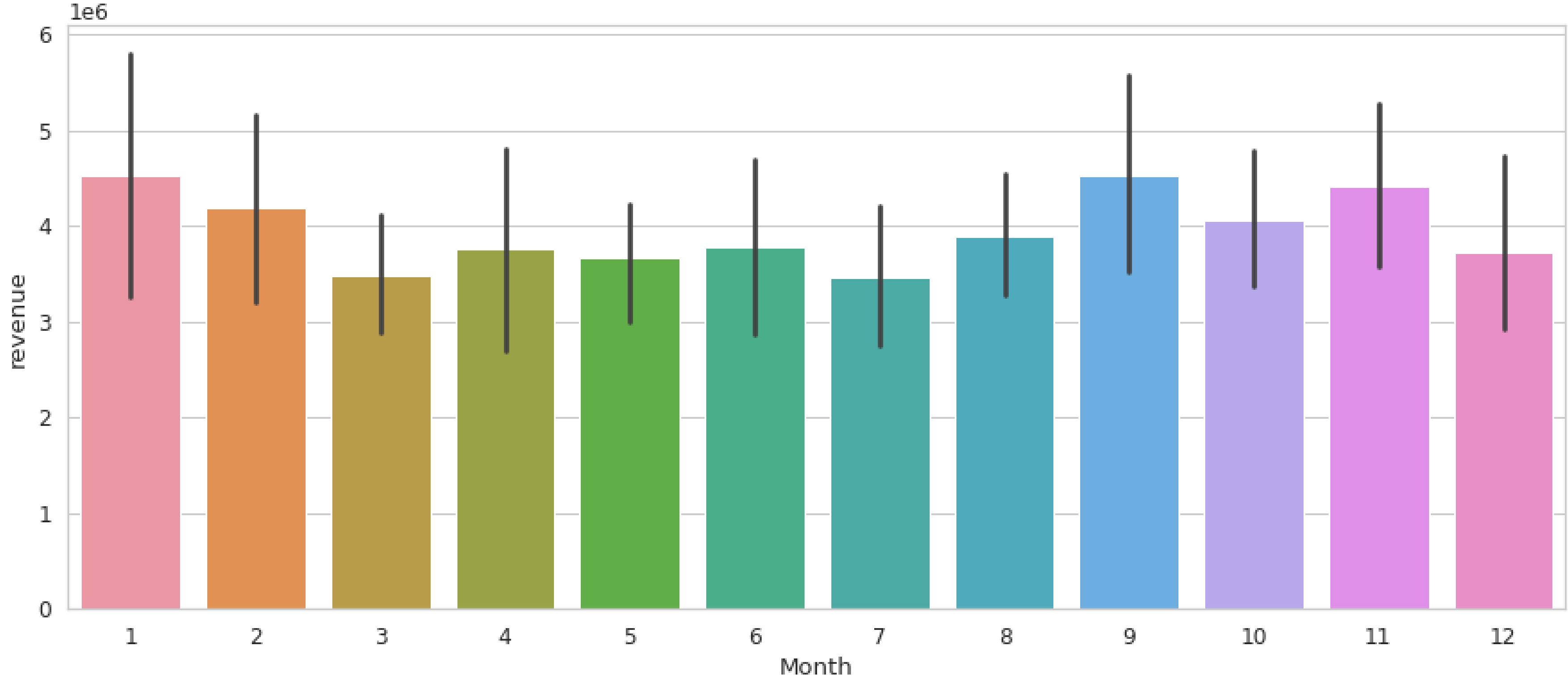
```
Month
1      4.521243e+06
2      4.189109e+06
3      3.477052e+06
4      3.749950e+06
5      3.657800e+06
6      3.776214e+06
7      3.458596e+06
8      3.883020e+06
9      4.526998e+06
10     4.056980e+06
11     4.403934e+06
```

```
12    3.720047e+06
Name: revenue, dtype: float64
```

From here we can see that the month January gave the most revenue to the restraunts. September and October followed January.

```
plt.figure(figsize=(15,6))
sns.barplot('Month','revenue',data=df)
```

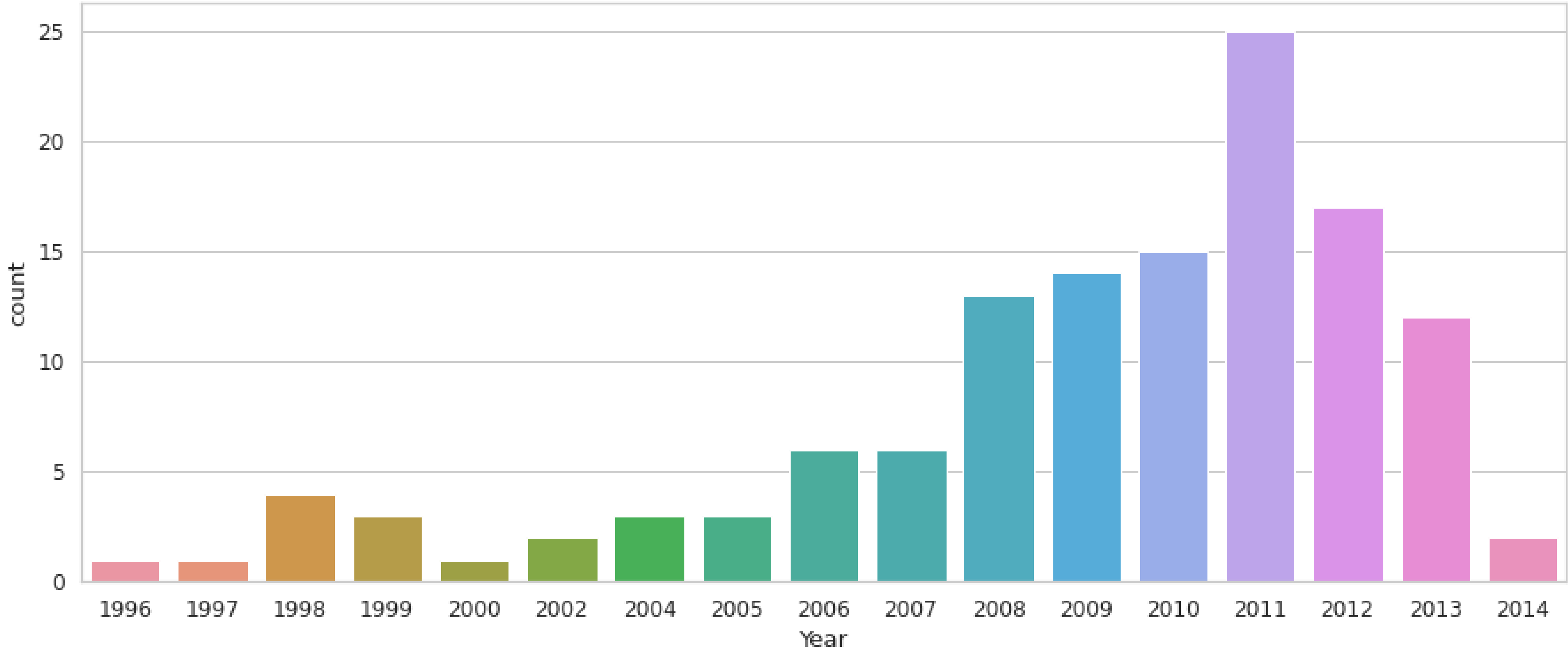
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From ve
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f842b7eb5b0>
```




```
plt.figure(figsize=(15,6))
```

```
sns.countplot(df['Year'])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From versi
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f842bbb3d60>
```



From here we can see that the most of the data is from the years 2008-2013. Out of them the most of the data is from the year 2011. The other years are contruibuting really less on the basis of number of data. This is also going to affect the results as well.

```
df.groupby('Year')['revenue'].mean()
```

```
Year
1996     3.903884e+06
1997     4.286645e+06
1998     4.251905e+06
```

```
1999    5.246965e+06
2000    7.495092e+06
2002    4.991022e+06
2004    3.482435e+06
2005    3.298470e+06
2006    3.360841e+06
2007    4.317164e+06
2008    4.588214e+06
2009    4.094408e+06
2010    4.383878e+06
2011    4.147879e+06
2012    3.540404e+06
2013    2.532287e+06
2014    2.464944e+06
Name: revenue, dtype: float64
```

```
plt.figure(figsize=(15,6))
sns.barplot('Year','revenue',data=df)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From ve
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f842b97b940>
```

Out of all the years, the highest revenue was generated in the year 2000.

Encode 'Year'

```
le=LabelEncoder()
df['Year']=le.fit_transform(df['Year'])
df
```

	City	City Group	Type	P1	P2	P3	P4	P5	P6	P7	...	P31	P32	P33	P34	P35	P36	P37	revenue	Month	Year
1	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	2	3	...	0	0	0	0	0	0	0	6923131.0	2	10
3	Tokat	Other	IL	6	4.5	6.0	6.0	4	4	10	...	12	10	6	18	12	12	6	2675511.0	2	14
132	Trabzon	Other	FC	2	3.0	3.0	5.0	4	2	4	...	0	0	0	0	0	0	0	5787594.0	6	10

Categorical Features

using categorical features to visualization and get insights of data

```
df['City Group'].value_counts()
```

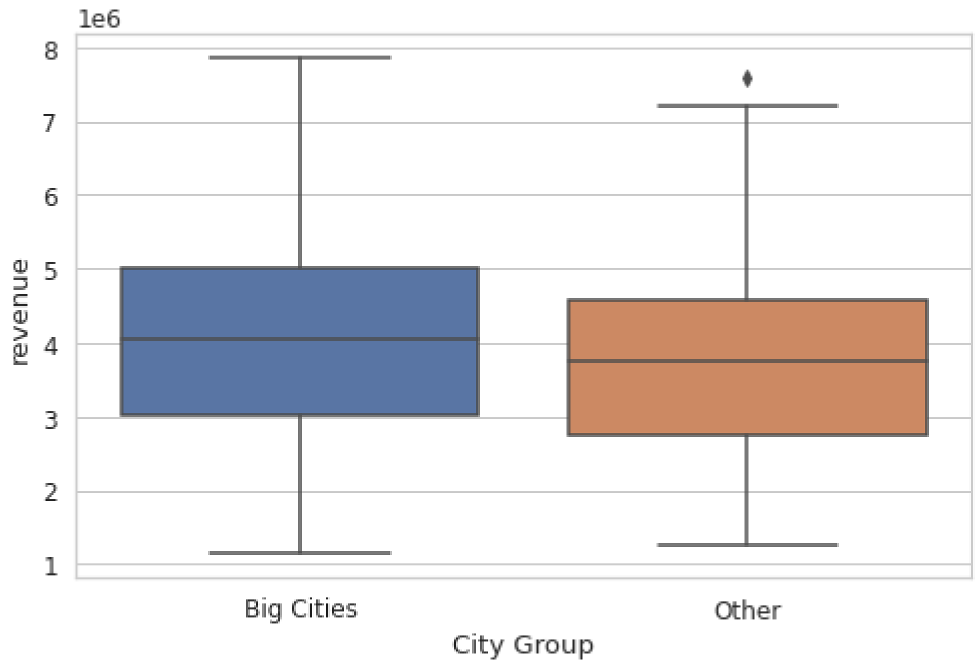
```
Big Cities    69
Other         59
Name: City Group, dtype: int64
```

```
sns.countplot(data=df, x="City Group", palette="Blues")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f842acb1040>
```


```
sns.boxplot(x="City Group", y='revenue', data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f842d923310>
```

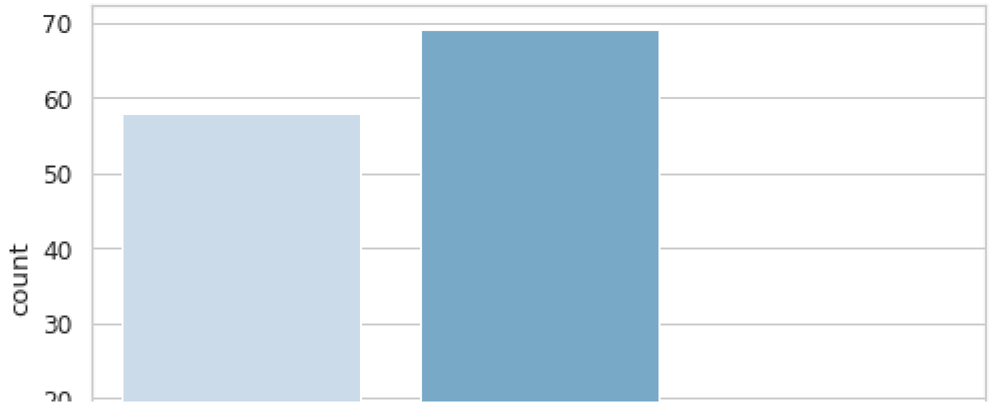



```
df['Type'].value_counts()
```

```
FC    69
IL    58
DT     1
Name: Type, dtype: int64
```

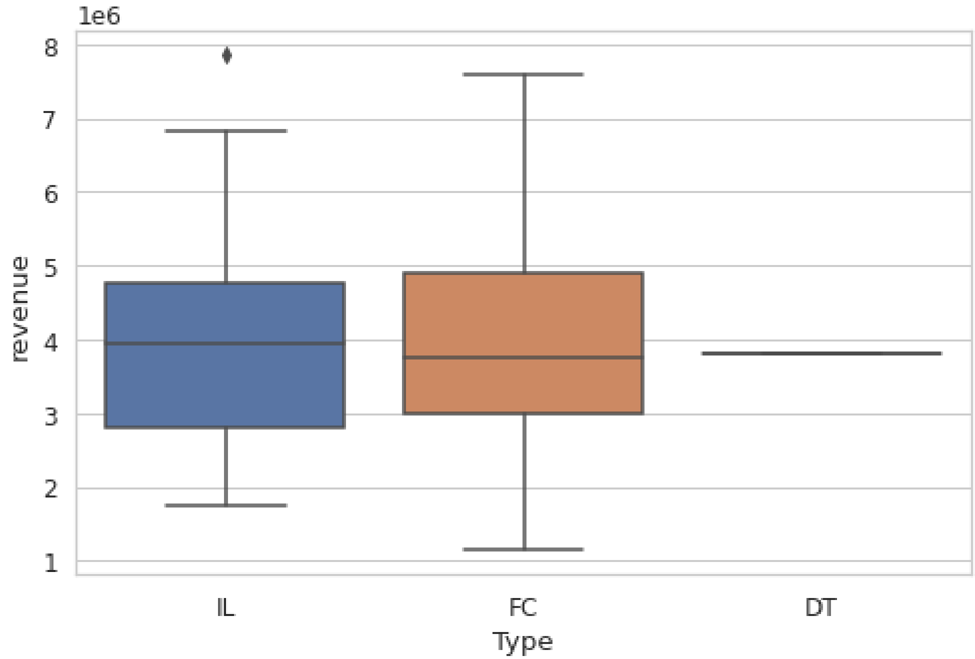
```
sns.countplot(data=df, x="Type", palette="Blues")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f842abeb160>




```
sns.boxplot(x="Type", y='revenue', data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f842abb55b0>

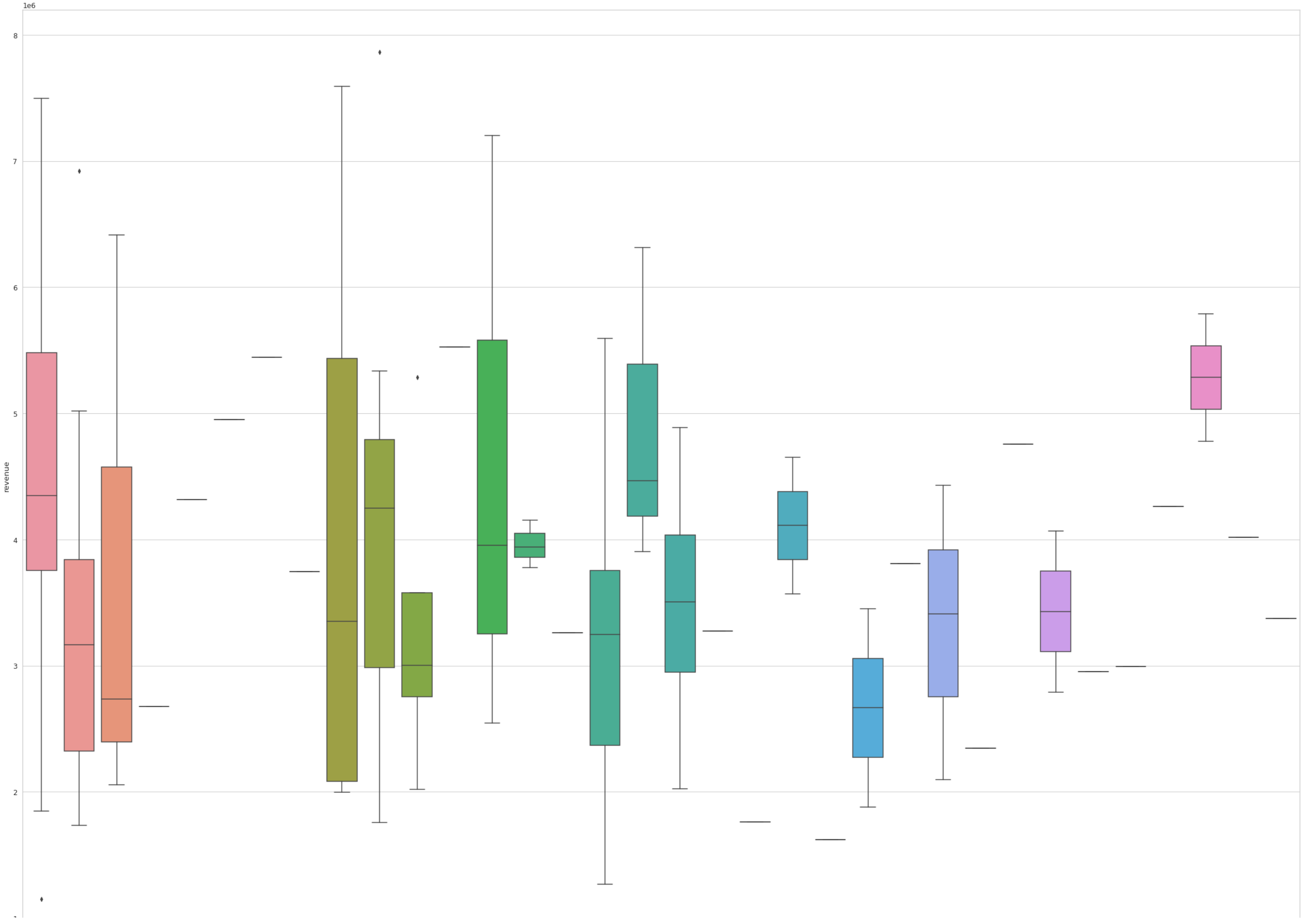


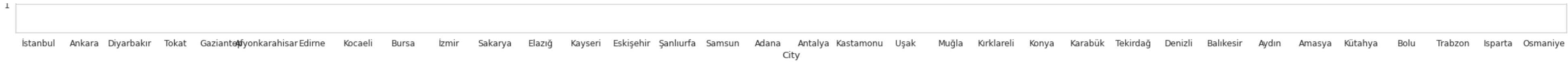

```
df['City'].value_counts()
```

```
İstanbul          43
Ankara            19
İzmir              7
Bursa              5
Samsun             5
Sakarya            4
Antalya            4
Kayseri            3
Eskişehir          3
Adana              3
Diyarbakır         3
Tekirdağ           3
Muğla              2
Trabzon            2
Aydın              2
Konya              2
Karabük            1
Isparta            1
Bolu               1
Kütahya            1
Amasya             1
Balıkesir          1
Denizli            1
Kocaeli            1
Kırklareli         1
Edirne             1
Uşak               1
Kastamonu          1
Tokat              1
Şanlıurfa          1
Elazığ             1
Gaziantep          1
Afyonkarahisar     1
Osmaniye           1
Name: City, dtype: int64
```

```
fig, ax = plt.subplots(1, 1, figsize=(40, 30))
sns.boxplot(x='City', y='revenue', data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f842ab21d30>





There are 63 different City values. I'd like to dropping it since there are too many factors and don't give much information of revenue. Besides, the 'City Group' feature can provide the effect of city as well.

```
df=df.drop('City',axis=1)
df
```

	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	...	P31	P32	P33	P34	P35	P36	P37	revenue	Month	Year
0	Big Cities	IL	4	5.0	4.0	4.0	2	2	5	4	...	3	4	5	5	4	3	4	5653753.0	7	3
1	Big Cities	FC	4	5.0	4.0	4.0	1	2	5	5	...	0	0	0	0	0	0	0	6923131.0	2	10
2	Other	IL	2	4.0	2.0	5.0	2	3	5	5	...	0	0	0	0	0	0	0	2055379.0	3	15
131	Big Cities	FC	3	4.0	4.0	5.0	3	4	5	4	...	0	0	0	0	0	0	0	3199619.0	11	5

Encode categorical features

```
le=LabelEncoder()
```

```
df['Type']=le.fit_transform(df['Type'])
df['City  Group']=le.fit_transform(df['City  Group'])
df
```

	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	...	P31	P32	P33	P34	P35	P36	P37	revenue	Month	Year
2	1	2	2	4.0	2.0	5.0	2	3	5	5	...	0	0	0	0	0	0	0	2055379.0	3	15

Feature Importance

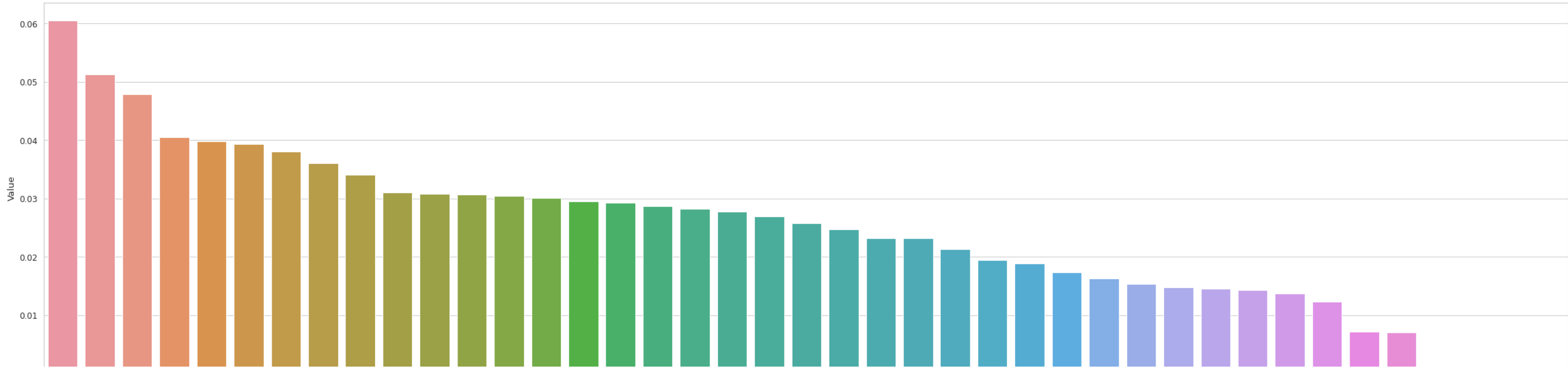
```
#Calculate F Score using XGB Regressor
x=df.drop('revenue',axis=1)
y=df['revenue']
xgb = XGBRegressor()
xgb.fit(x, y)

print(xgb.feature_importances_)
```

```
[18:37:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[0.00708976 0.01428882 0.02925263 0.06052972 0.01370331 0.02690866
 0.0213252  0.03048534 0.01736989 0.03011752 0.02946245 0.03599986
 0.0188325  0.02320181 0.         0.03103615 0.01944258 0.03070349
 0.02771336 0.01535809 0.02319274 0.03076216 0.02468628 0.02872329
 0.0147084  0.02820621 0.03936569 0.00720977 0.03400399 0.05126117
 0.04045502 0.03973033 0.03806062 0.         0.         0.01454267
 0.         0.0163374  0.01232213 0.02576109 0.0478499 ]
```

```
f_xgb = pd.DataFrame(data={'Feature':x.columns,'Value':xgb.feature_importances_})
f_xgb = f_xgb.sort_values(['Value'],ascending=False )
plt.figure(figsize=(15,8))
sns.barplot(f_xgb['Feature'],f_xgb['Value'])
plt.gcf().set_size_inches(40,10)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From ve
  warnings.warn(
```



The plot shows that the there are 4 variables are not important, so we want to drop them to better fit models.

```
df=df.drop(['P13'],axis=1)
df=df.drop(['P32'],axis=1)
df=df.drop(['P33'],axis=1)
df=df.drop(['P35'],axis=1)
```

Modeling

Train-Test Split

```
from sklearn.model_selection import train_test_split
x1=df.drop('revenue',axis=1)
y1=df['revenue']
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(102, 41)
(102,)
(26, 41)
(26,)
```

Baseline Models

Random Forest Regressor

```
RFR = RandomForestRegressor()
RFR = RFR.fit(X_train,y_train)


pred = RFR.predict(X_test)
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))

print("The MAE with the RF regressor is: "+str(mae))
print("The MSE with the RF regressor is: "+str(mse))
print("The R2_Score with the RF regressor is: "+str(r2))
print("The RMSE withe the RF regressor is:"+str(rmse))
```

```
The MAE with the RF regressor is: 913711.7453846154
The MSE with the RF regressor is: 1349095349956.7031
The R2_Score with the RF regressor is: 0.241789951835654
The RMSE withe the RF regressor is:1161505.6392272501
```

K-Neighbors Regressor

```
knn=KNeighborsRegressor()
knn=knn.fit(X_train,y_train)
```

```
pred = knn.predict(X_test)
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))

print("The MAE with the KNN regressor is: "+str(mae))
print("The MSE with the KNN regressor is: "+str(mse))
print("The R2_Score with the KNN regressor is: "+str(r2))
print("The RMSE withe the KNN regressor is:"+str(rmse))
```

```
The MAE with the KNN regressor is: 1171152.8000000003
The MSE with the KNN regressor is: 1802807234361.139
The R2_Score with the KNN regressor is: -0.013202335950426969
The RMSE withe the KNN regressor is:1342686.5733897614
```

Lasso

```
l = Lasso()
l=l.fit(X_train,y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You
  model = cd_fast.enet_coordinate_descent(
```

```
pred = l.predict(X_test)
```

```
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)

print("The MAE with the Lasso is: "+str(mae))
print("The MSE with the Lasso is: "+str(mse))
print("The R2_Score with the Lasso is: "+str(r2))
print("The RMSE withe the Lasso is:"+str(rmse))
```

```
The MAE with the Lasso is: 2252247.189497925
The MSE with the Lasso is: 7238794991960.511
The R2_Score with the Lasso is: -3.068301843663126
The RMSE withe the Lasso is:1342686.5733897614
```

Ridge

```
r = Ridge()
r = r.fit(X_train,y_train)


pred = r.predict(X_test)
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))

print("The MAE with the Ridge is: "+str(mae))
print("The MSE with the Ridge is: "+str(mse))
print("The R2_Score with the Ridge is: "+str(r2))
print("The RMSE withe the Lasso is:"+str(rmse))
```

```
The MAE with the Ridge is: 2003038.7538788451
The MSE with the Ridge is: 5847565587567.929
The R2_Score with the Ridge is: -2.2864118803288016
The RMSE withe the Lasso is:2418174.019289747
```

XGB Regressor

```
xgb=XGBRegressor()
xgb=xgb.fit(X_train,y_train)
```

```
[18:37:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
pred = xgb.predict(X_test)
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))

print("The MAE with the XGB regressor is: "+str(mae))
print("The MSE with the XGB regressor is: "+str(mse))
print("The R2_Score with the XGB regressor is: "+str(r2))
print("The RMSE withe the XGB regressor is:"+str(rmse))
```

```
The MAE with the XGB regressor is: 1036332.1490384615
The MSE with the XGB regressor is: 1644881322806.063
The R2_Score with the XGB regressor is: 0.07555418745647313
The RMSE withe the XGB regressor is:1282529.267816553
```

Comparing Models

```
regressors = {"RandomForestRegressor": RandomForestRegressor(),
              "KNN": KNeighborsRegressor(),
              "Ridge": Ridge(),
              "Lasso": Lasso(),
              "XGBoostRegressor": XGBRegressor()}
```

```
results=pd.DataFrame(columns=['MAE','MSE','R2-score','RMSE'])
```

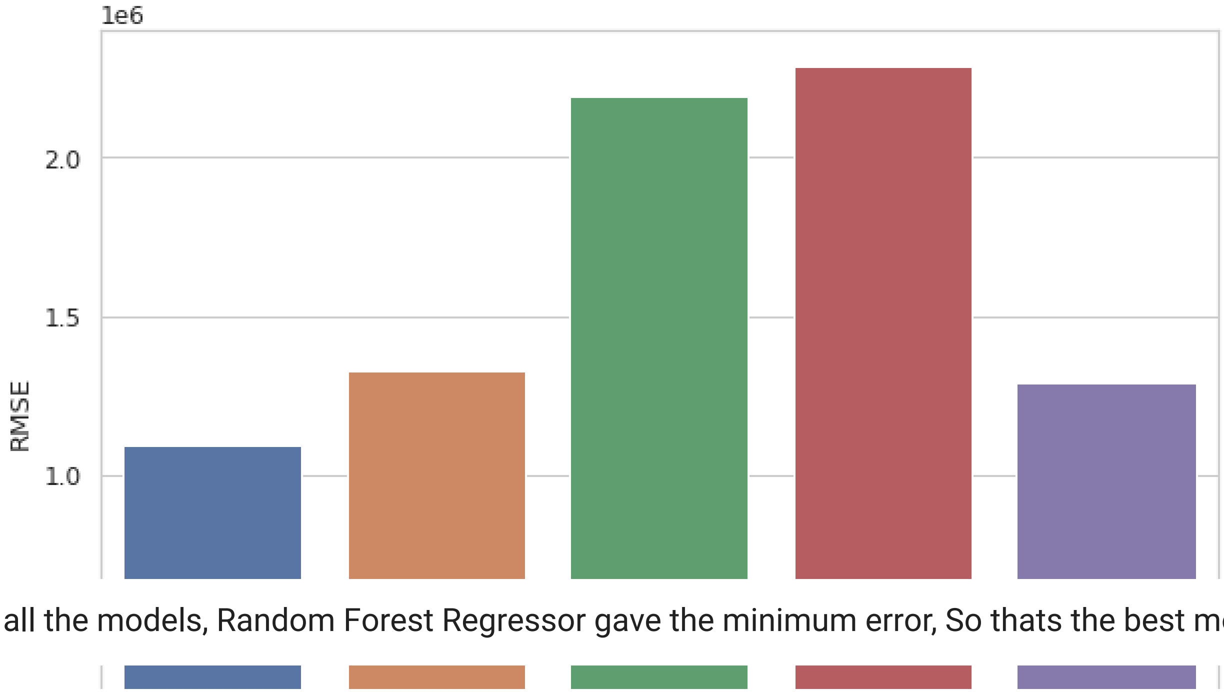
```
results=pd.DataFrame(columns=['MAE','MSE','R2-score','RMSE'])
for  method,func  in  regressors.items():
        func.fit(X_train,y_train)
        pred  =  func.predict(X_test)
        results.loc[method]=  [mean_absolute_error(y_test,pred),
                                                    mean_squared_error(y_test,pred),
                                                    r2_score(y_test,pred),
                                                    np.sqrt(mean_squared_error(y_test,  pred))
                                                  ]
results
```

```
[18:38:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You
  model = cd_fast.enet_coordinate_descent(
```

	MAE	MSE	R2-score	RMSE
RandomForestRegressor	8.385172e+05	1.258408e+12	0.292758	1.121788e+06
KNN	1.171153e+06	1.802807e+12	-0.013202	1.342687e+06
Ridge	2.003039e+06	5.847566e+12	-2.286412	2.418174e+06
Lasso	2.252247e+06	7.238795e+12	-3.068302	2.690501e+06
XGBoostRegressor	1.036332e+06	1.644881e+12	0.075554	1.282529e+06

```
fig,  ax  =  plt.subplots(1,1,sharey=False,figsize=(10,7))
bar  =  sns.barplot(keys,values,ax=ax)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From ve
  warnings.warn(
```



From all the models, Random Forest Regressor gave the minimum error, So thats the best model and should be chosen as the final model.

Hyperparameters tuning with GridSearchCV

```
from sklearn.model_selection import GridSearchCV
params = {
        "max_depth": ["None",10, 30, 50, 75, 100],
        "max_features": ["auto",0.3, 0.6],
        "min_samples_leaf": [1,3,5,7],
        "min_samples_split": [2, 4, 8, 12],
        "n_estimators": [30, 50, 100, 200]
}

## RandomForestRegressor
RFR = RandomForestRegressor()
RFR_grid = GridSearchCV(RFR, params, scoring='neg_root_mean_squared_error', cv=3, n_jobs=-1)
```

```
RFR_grid.fit(X_train,  y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
576 fits failed out of a total of 3456.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
576 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_forest.py", line 450, in fit
    trees = Parallel(
  File "/usr/local/lib/python3.8/dist-packages/joblib/parallel.py", line 1085, in __call__
    if self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python3.8/dist-packages/joblib/parallel.py", line 901, in dispatch_one_batch
    self._dispatch(tasks)
  File "/usr/local/lib/python3.8/dist-packages/joblib/parallel.py", line 819, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/usr/local/lib/python3.8/dist-packages/joblib/_parallel_backends.py", line 208, in apply_async
    result = ImmediateResult(func)
  File "/usr/local/lib/python3.8/dist-packages/joblib/_parallel_backends.py", line 597, in __init__
    self.results = batch()
  File "/usr/local/lib/python3.8/dist-packages/joblib/parallel.py", line 288, in __call__
    return [func(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/joblib/parallel.py", line 288, in <listcomp>
    return [func(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/utils/fixes.py", line 216, in __call__
    return self.function(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/ensemble/_forest.py", line 185, in _parallel_build_trees
    tree.fit(X, y, sample_weight=curr_sample_weight, check_input=False)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/tree/_classes.py", line 1315, in fit
    super().fit(
  File "/usr/local/lib/python3.8/dist-packages/sklearn/tree/_classes.py", line 305, in fit
    if max_depth <= 0:
TypeError: '<=' not supported between instances of 'str' and 'int'

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py:969: UserWarning: One or more of the test scores are non-finite: [
 -1461387.37310333 -1478422.02743076 -1465090.77100244]
  warnings.warn(
GridSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'max_depth': ['None', 10, 30, 50, 75, 100],
                         'max_features': ['auto', 0.3, 0.6],
                         'min_samples_leaf': [1, 3, 5, 7],
                         'min_samples_split': [2, 4, 8, 12],
                         'n_estimators': [30, 50, 100, 200]},
             scoring='neg_root_mean_squared_error')
```

```
print(RFR_grid.best_score_)
print(RFR_grid.best_params_)
print(RFR_grid.best_estimator_)
```

```
-1422954.5334620455
{'max_depth': 30, 'max_features': 0.6, 'min_samples_leaf': 5, 'min_samples_split': 12, 'n_estimators': 50}
RandomForestRegressor(max_depth=30, max_features=0.6, min_samples_leaf=5,
                      min_samples_split=12, n_estimators=50)
```

Model Performance with Best Hyperparameters

```
RFR=RandomForestRegressor(max_depth=30, max_features=0.6, min_samples_leaf=5,
                                            min_samples_split=12, n_estimators=50)
RFR=RFR.fit(X_train,y_train)
```

```
pred = RFR.predict(X_test)
mae=mean_absolute_error(y_test,pred)
mse=mean_squared_error(y_test,pred)
r2=r2_score(y_test,pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))

print("The MAE with the RFR regressor is: "+str(mae))
```

```
print("The  MSE  with  the  RFR  regressor  is:  "+str(mse))
print("The  R2_Score  with  the  RFR  regressor  is:  "+str(r2))
print("The  RMSE  withe  the  RFR  regressor  is:"+str(rmse))
```

```
The MAE with the RFR regressor is: 964629.310433871
The MSE with the RFR regressor is: 1464390064333.9844
The R2_Score with the RFR regressor is: 0.1769927446227625
The RMSE withe the RFR regressor is:1210119.8553589575
```

Submission

```
test=  pd.read_csv('/content/test.csv')
```

```
test.shape
```

```
(100000, 42)
```

```
test['Open  Date']=pd.to_datetime(test['Open  Date'])
test['Month']=[x.month  for  x  in  test['Open  Date']]
test['Year']=[x.year  for  x  in  test['Open  Date']]
test['Year']=le.fit_transform(test['Year'])
test=test.drop(['Open  Date'],axis=1)
```

```
test['Type']=le.fit_transform(test['Type'])
test['City  Group']=le.fit_transform(test['City  Group'])
test=test.drop(['City'],axis=1)
```

```
test=test.drop(['P13'],axis=1)
test=test.drop(['P32'],axis=1)
test=test.drop(['P33'],axis=1)
test=test.drop(['P35'],axis=1)
```

```
test_id = test['Id'].tolist()
test.drop('Id',axis=1, inplace=True)
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 37 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   City Group  100000 non-null  int64
 1   Type        100000 non-null  int64
 2   P1          100000 non-null  int64
 3   P2          100000 non-null  float64
 4   P3          100000 non-null  float64
 5   P4          100000 non-null  float64
 6   P5          100000 non-null  int64
 7   P6          100000 non-null  int64
 8   P7          100000 non-null  int64
 9   P8          100000 non-null  int64
 10  P9          100000 non-null  int64
 11  P10         100000 non-null  int64
 12  P11         100000 non-null  int64
 13  P12         100000 non-null  int64
 14  P14         100000 non-null  int64
 15  P15         100000 non-null  int64
 16  P16         100000 non-null  int64
 17  P17         100000 non-null  int64
 18  P18         100000 non-null  int64
 19  P19         100000 non-null  int64
 20  P20         100000 non-null  int64
 21  P21         100000 non-null  int64
 22  P22         100000 non-null  int64
 23  P23         100000 non-null  int64
 24  P24         100000 non-null  int64
 25  P25         100000 non-null  int64
 26  P26         100000 non-null  float64
 27  P27         100000 non-null  float64
 28  P28         100000 non-null  float64
 29  P29         100000 non-null  float64
```

```
30  P30         100000 non-null  int64
31  P31         100000 non-null  int64
32  P34         100000 non-null  int64
33  P36         100000 non-null  int64
34  P37         100000 non-null  int64
35  Month       100000 non-null  int64
36  Year        100000 non-null  int64
dtypes: float64(7), int64(30)
memory usage: 28.2 MB
```

```
RFR=RandomForestRegressor(max_depth=30, max_features=0.6, min_samples_leaf=5,
                                   min_samples_split=12, n_estimators=50)
RFR=RFR.fit(x1,y1)
prediction = RFR.predict(test)
```

```
prediction.shape
```

```
(100000,)
```

```
ID = np.arange(0,prediction.shape[0])

d = {'Id': ID, 'Prediction': prediction}
out = pd.DataFrame(d)
out.to_csv('/content/prediction.csv',
        index = False)
```

Conclusion

Our best performance model is Random Forest Regressor and our Kaggle get a best score of 1880581. This high rmse is because transformed revenue vairable.
To get a better model, feature selection is very important. Fo

Our best performance model is Random Forest Regressor and our submission in Kaggle get a best score of 1880581. This high rmse is

how we work with date, how to handle City and City Group is
We tried to convert date to open days for restaurant and keep
first try, but the result is not very good. I believe is beca
vairables makes the model messy but don't give much information,
target of this project is to help TFI deciding when and where
restaurants, the open days might not helpful. Also, the dataset
so how to train the data and maxmium the use of data is also
consider about in the whole process.

because of the transformed revenue vairable. To get a better model, feature selection is very important. For this project, how we work with date, how to handle City and City Group is crucial. We tried to convert date to open days for restaurant and keep the city in our first try, but the result is not very good. I believe is because so many city vairables makes the model messy but don't give much information, and as the target of this project is to help TFI deciding when and where to open new restaurants, the open days might not helpful. Also, the dataset is very small, so how to train the data and maxmium the use of data is also the thing we consider about in the whole process.