Restaurant Reveune Analysis

30/06/2024

Nguyễn Bá Vinh
Trần Nguyễn Việt Anh

About us

With over 1,200 quick service restaurants across the globe, TFI is the company behind some of the world's most well-known brands: Burger King, Sbarro, Popeyes, Usta Donerci, and Arby’s. They employ over 20,000 people in Europe and Asia and make significant daily investments in developing new restaurant sites.

Problem

Currently, choosing new restaurant locations relies heavily on the gut feeling of development teams, making it inconsistent across different regions and cultures. This subjective approach is risky, as opening a restaurant in the wrong spot can lead to closure within a year and a half, wasting significant time and money. To improve investment efficiency and free up resources for other areas, a mathematical model that predicts restaurant sales based on various data points is needed. This analysis article aims to develop such a model by using data on demographics, real estate, and commerce to predict annual sales for 100,000 potential locations.

Report

01
Exploratory Data Analysis

02
Data Preprocessing

03
Model Employing

04
Suggestion and Conclusion

1

Exploratory Data Analysis

```
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
```

Information about dataset

Data shape:

- 42 features (38 numerical columns + 4 categorical columns)
- 137 records

Mini Preprocessing

This step include:

- Converting the column Open Date to DateTime.
- Deleting the columns that don't need to be analyzed.
- Modifying the columns for being suitable with model.

```
[6]  df = df.drop(columns = ['Id'])
```

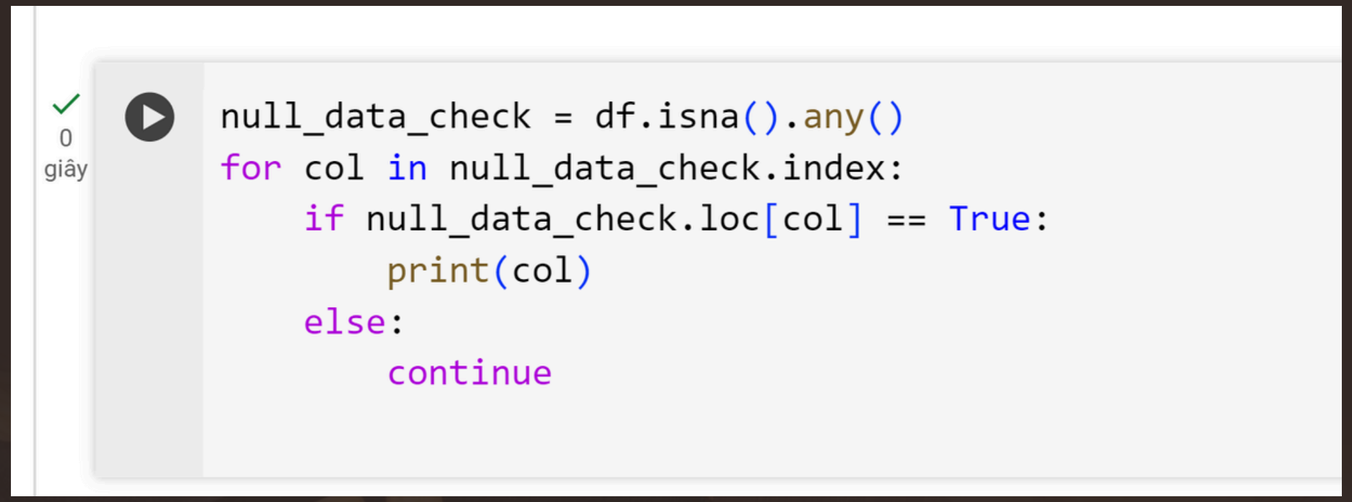
```
df['Open Date'] = pd.to_datetime(df['Open Date'], format= '%m/%d/%Y')
df['Opened_range_time'] = (df['Open Date'] - df['Open Date'].min()).dt.days
df = df.drop(columns=['Open Date'])
df['Opened_range_time']
```

```
0      1165
1      4299
2      6149
3      5748
4      4749
       ...
132    4431
133    3809
134    3713
135    5287
136    4864
Name: Opened_range_time, Length: 137, dtype: int64
```

```
df['City_Group'] = df['City Group']
df = df.drop(columns= ['City Group'])
```

Dataframe Content Inspection

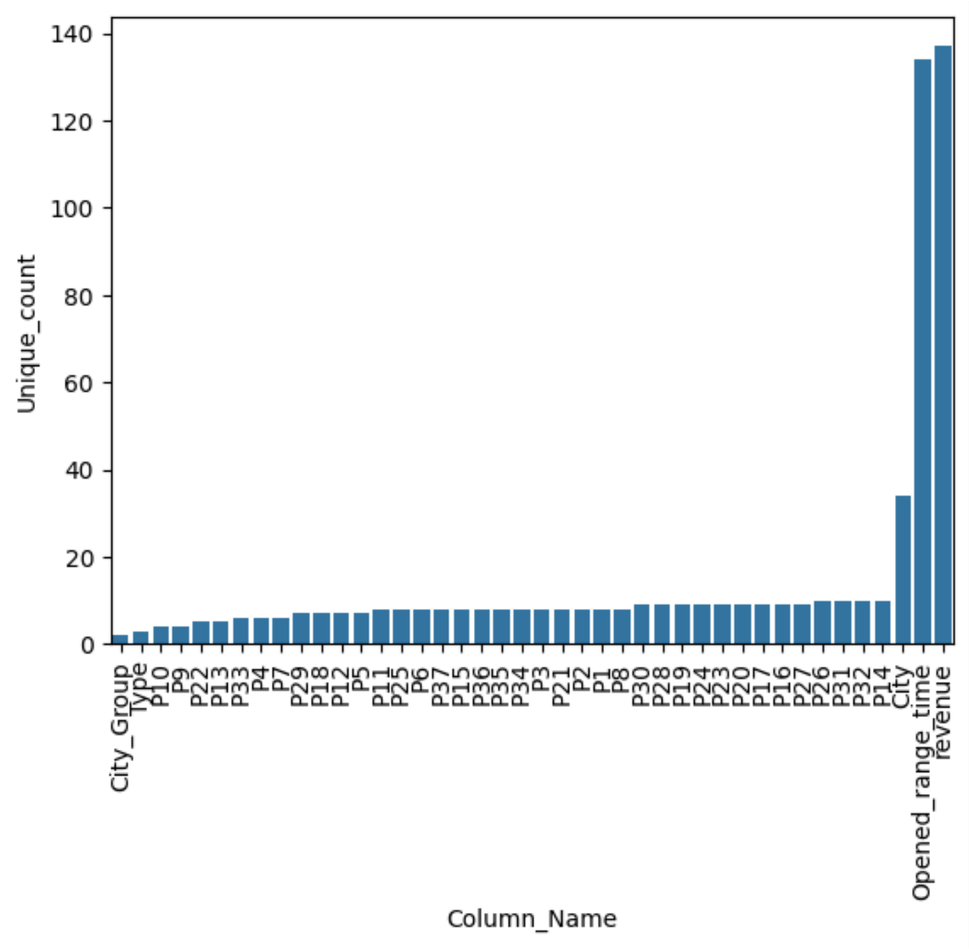
Null-values checks



This step is used to check for null value.

Result: No column containg null-values.

Dataframe Content Inspection



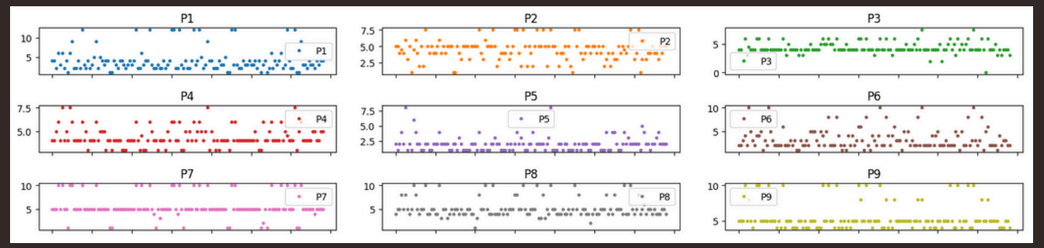
Unique Values Checks

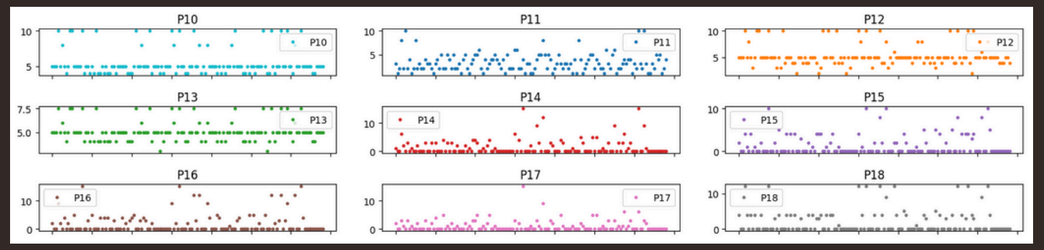
- Most columns have small number of unique values except 3 features: City, Opened_range_time, and revenue.

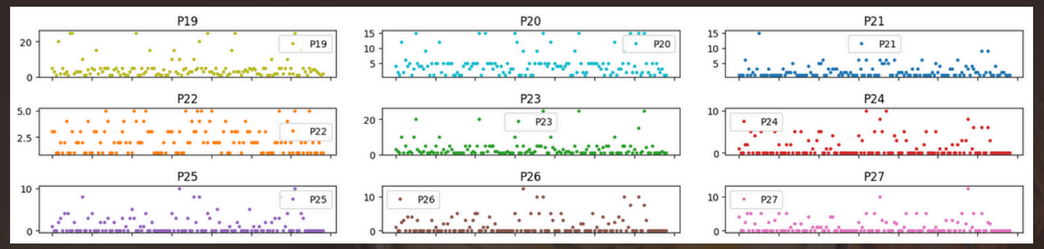
Dataframe Content Inspection

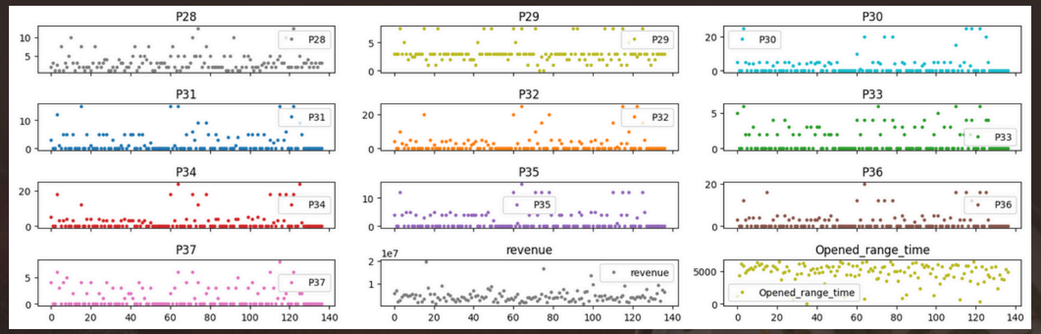
Value Behavior

- There are no clear trends in the data distribution for each feature.
- The density of data points is so thick that it is quite confusing.









Univariate Analysis

Summary for Numerical Columns

Comment

	P1	P2	P3	P4	P5	P6	...	P33	P34	P35	P36	P37	revenue
count	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	...	137.000000	137.000000	137.000000	137.000000	137.000000	1.370000e+02
mean	4.014599	4.408759	4.317518	4.372263	2.007299	3.357664	...	1.138686	2.489051	2.029197	2.211679	1.116788	4.453533e+06
std	2.910391	1.514900	1.032337	1.016462	1.209620	2.134235	...	1.698540	5.165093	3.436272	4.168211	1.790768	2.576072e+06
min	1.000000	1.000000	0.000000	3.000000	1.000000	1.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	1.149870e+06
25%	2.000000	4.000000	4.000000	4.000000	1.000000	2.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	2.999068e+06
50%	3.000000	5.000000	4.000000	4.000000	2.000000	3.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	3.939804e+06
75%	4.000000	5.000000	5.000000	5.000000	2.000000	4.000000	...	2.000000	3.000000	4.000000	3.000000	2.000000	5.166635e+06
max	12.000000	7.500000	7.500000	7.500000	8.000000	10.000000	...	6.000000	24.000000	15.000000	20.000000	8.000000	1.969694e+07

Univariate Analysis

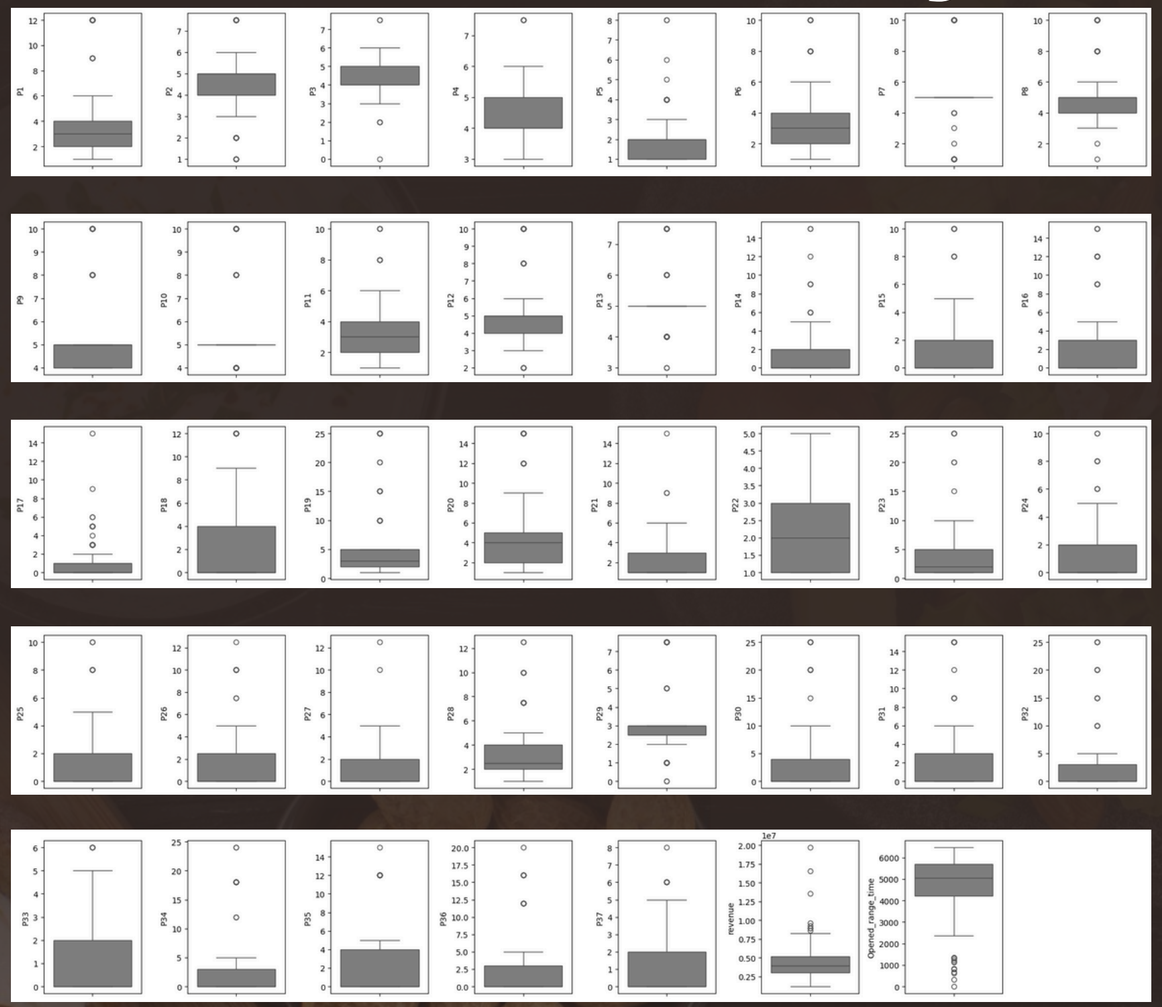
Summary for Categorical columns

As we can see in the table below, Istanbul appears 50 times out of 137, Type is mainly FC with 76 out of 137.

This can create a large gap between Istanbul and other cities

	City	Type	City_Group
count	137	137	137
unique	34	3	2
top	İstanbul	FC	Big Cities
freq	50	76	78

Univariate Analysis



Column Distribution

- All columns contains outlier values.
- The value of the opening hour subtraction box is quite low, close to the min value.

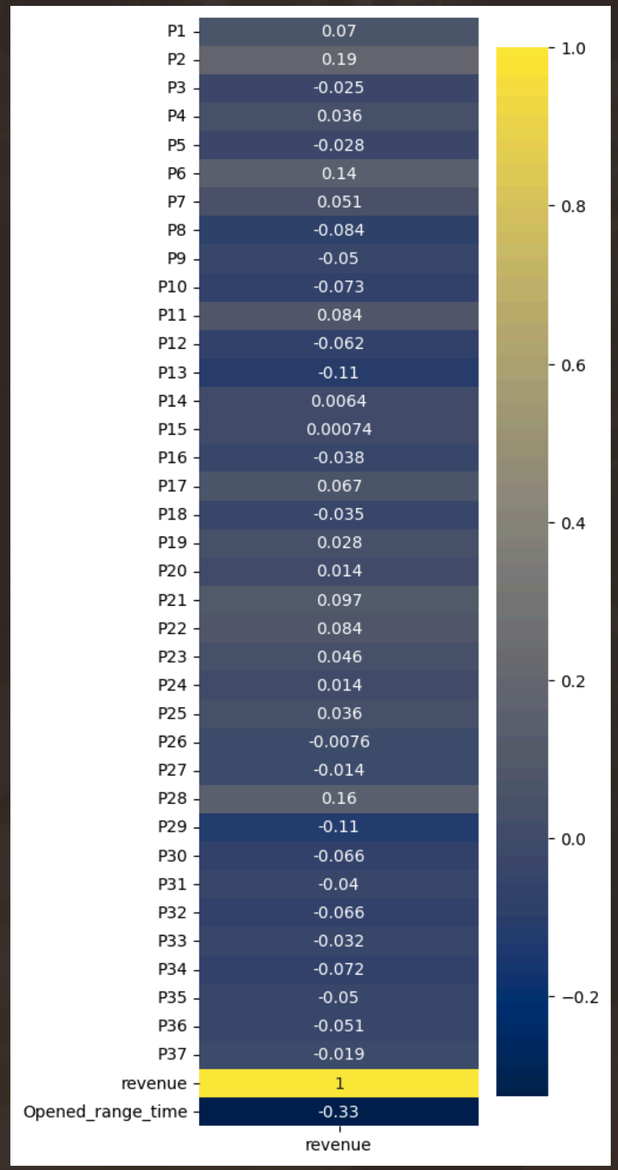
Analysis againts Revenue

Correlation between revenue and others

```
df.select_dtypes(['int64', 'float64']).corr().loc[:,['revenue']]['revenue'].mean()
```

```
0.027721471882549514
```

- Mean of correlation values is 0.02.
- Almost independent variables have no significantly effect to dependent variables.



2

Preprocessing

Preprocessing: Remove Duplicated records

```
# Handle duplicates
df.drop_duplicates()
```

	Id	Open Date	City	Type	P1	P2	P3	P4	P5	P6	...	P30	P31	P32	P33	P34	P35	P36	P37	revenue	City_Group
0	0	07/17/1999	İstanbul	IL	4	5.0	4.0	4.0	2	2	...	5	3	4	5	5	4	3	4	5653753.0	Big Cities
1	1	02/14/2008	Ankara	FC	4	5.0	4.0	4.0	1	2	...	0	0	0	0	0	0	0	0	6923131.0	Big Cities
2	2	03/09/2013	Diyarbakır	IL	2	4.0	2.0	5.0	2	3	...	0	0	0	0	0	0	0	0	2055379.0	Other
3	3	02/02/2012	Tokat	IL	6	4.5	6.0	6.0	4	4	...	25	12	10	6	18	12	12	6	2675511.0	Other
4	4	05/09/2009	Gaziantep	IL	3	4.0	3.0	4.0	2	2	...	5	1	3	2	3	4	3	3	4316715.0	Other
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
132	132	06/25/2008	Trabzon	FC	2	3.0	3.0	5.0	4	2	...	0	0	0	0	0	0	0	0	5787594.0	Other
133	133	10/12/2006	İzmir	FC	4	5.0	4.0	4.0	2	3	...	0	0	0	0	0	0	0	0	9262754.0	Big Cities
134	134	07/08/2006	Kayseri	FC	3	4.0	4.0	4.0	2	3	...	0	0	0	0	0	0	0	0	2544857.0	Other
135	135	10/29/2010	İstanbul	FC	4	5.0	4.0	5.0	2	2	...	0	0	0	0	0	0	0	0	7217634.0	Big Cities
136	136	09/01/2009	İstanbul	FC	4	5.0	3.0	5.0	2	2	...	0	0	0	0	0	0	0	0	6363241.0	Big Cities

Preprocessing

For numerical columns

```
[28] def log_transform(X):
         X = X.apply(lambda x: np.log(x+3))
         return X

     def linear_scaling(X):
         min_value = X.min(axis = 0)
         max_value = X.max(axis = 0)
         X = (X-min_value) / (max_value - min_value)
         return X

     def to_dataframe(X):
         return pd.DataFrame(X, columns=X.columns)
```

- Applies a logarithmic transformation to all values in the DataFrame X.
- Performs linear scaling (normalization) on the DataFrame X.
- Converts X into a DataFrame with the same columns as X.

Preprocessing

For categorical columns

```
[ ]  preprocessin_pipeline_categorical = Pipeline([("encoding", FunctionTransformer(func= encode))])
     X_test_preprocessed_categorical = preprocessin_pipeline_categorical.fit_transform(X_test)
     X_train_preprocessed_categorical = preprocessin_pipeline_categorical.fit_transform(X_train)
```

```
[ ]  X_test_preprocessed = pd.concat([X_test_preprocessed_numerical, X_test_preprocessed_categorical], axis = 1)
     X_train_preprocessed = pd.concat([X_train_preprocessed_numerical, X_train_preprocessed_categorical], axis= 1)
```

Using pipeline ensures that both the training and testing datasets undergo the same transformations, which is crucial for maintaining the integrity of machine learning models.

3

Model Prediction

Deploy Model and Evaluate

Linear Regression model (Base)

```
[30] linear_model = LinearRegression()
     linear_model.fit(X_train_preprocessed, y_train)
     y_train_pred = linear_model.predict(X_train_preprocessed)
     y_test_pred = linear_model.predict(X_test_preprocessed)
     # Output of data: "revenue" has oulier => using MAE
     print(f"In term of training: {mean_absolute_error(y_pred= y_train_pred, y_true= y_train)}")
     print(f"In term of testing: {mean_absolute_error(y_pred= y_test_pred, y_true= y_test)}")
     print(f"Mean of y_train: {y_train.mean()}")
     print(f"Mean of y_test: {y_test.mean()}")
```

```
In term of training: 1325300.598026269
In term of testing: 3527993.6329613975
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
```

MAE(Training) << mean(revenue) (Training)
=> Model has good performance at training data.

MAE(Testing) << mean(revenue) (Testing)
=> Model has good performance at testing data.

Deploy Model and Evaluate

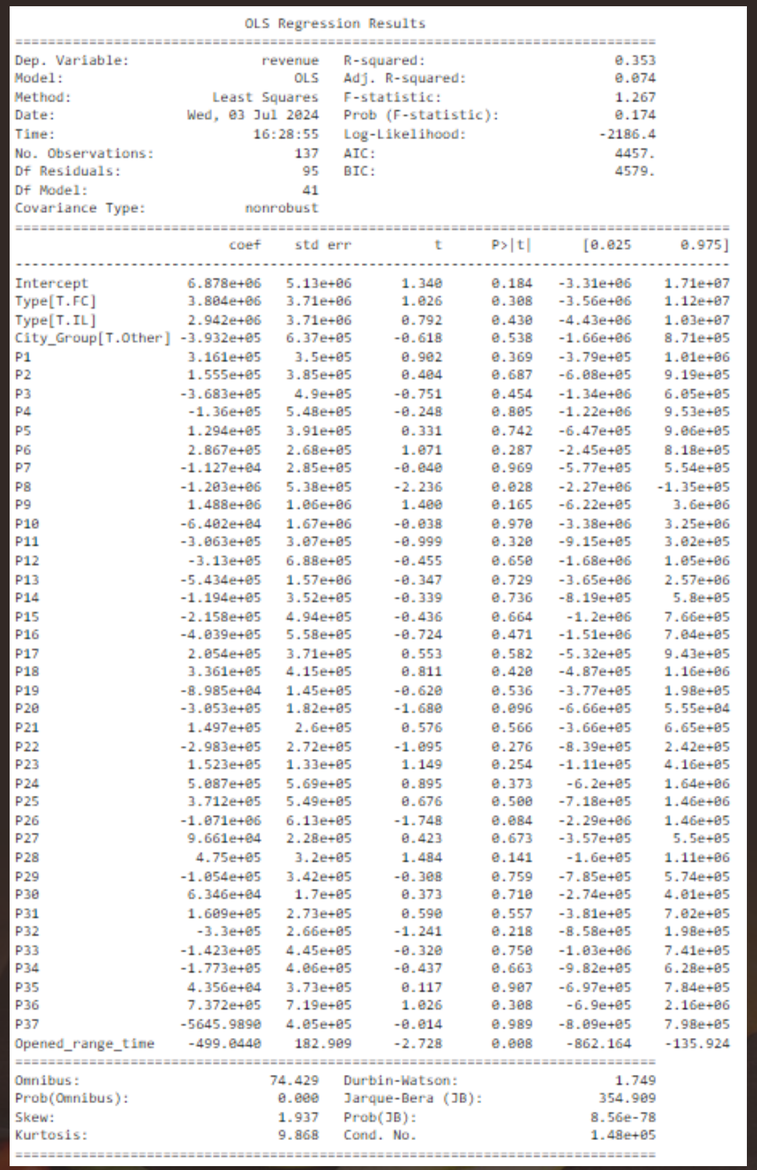
Multiple Linear Regression model

- Should consider finding other independent variables or other modeling methods to improve your model because p-value > 0,05

F-statistic = 1.267

p-value of mlr = 0.174 => This model has no statistical meaning

=> Reject model mlr



OLS Regression Results

Dep. Variable:	revenue	R-squared:	0.353
Model:	OLS	Adj. R-squared:	0.074
Method:	Least Squares	F-statistic:	1.267
Date:	Wed, 03 Jul 2024	Prob (F-statistic):	0.174
Time:	16:28:55	Log-Likelihood:	-2186.4
No. Observations:	137	AIC:	4457.
Df Residuals:	95	BIC:	4579.
Df Model:	41		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.878e+06	5.13e+06	1.340	0.184	-3.31e+06	1.71e+07
Type[T.FC]	3.804e+06	3.71e+06	1.026	0.308	-3.56e+06	1.12e+07
Type[T.IL]	2.942e+06	3.71e+06	0.792	0.430	-4.43e+06	1.03e+07
City_Group[T.Other]	-3.932e+05	6.37e+05	-0.618	0.538	-1.66e+06	8.71e+05
P1	3.161e+05	3.5e+05	0.902	0.369	-3.79e+05	1.01e+06
P2	1.555e+05	3.85e+05	0.404	0.687	-6.08e+05	9.19e+05
P3	-3.683e+05	4.9e+05	-0.751	0.454	-1.34e+06	6.05e+05
P4	-1.36e+05	5.48e+05	-0.248	0.805	-1.22e+06	9.53e+05
P5	1.294e+05	3.91e+05	0.331	0.742	-6.47e+05	9.06e+05
P6	2.867e+05	2.68e+05	1.071	0.287	-2.45e+05	8.18e+05
P7	-1.127e+04	2.85e+05	-0.040	0.969	-5.77e+05	5.54e+05
P8	-1.203e+06	5.38e+05	-2.236	0.028	-2.27e+06	-1.35e+05
P9	1.488e+06	1.06e+06	1.400	0.165	-6.22e+05	3.6e+06
P10	-6.402e+04	1.67e+06	-0.038	0.970	-3.38e+06	3.25e+06
P11	-3.063e+05	3.07e+05	-0.999	0.320	-9.15e+05	3.02e+05
P12	-3.13e+05	6.88e+05	-0.455	0.650	-1.68e+06	1.05e+06
P13	-5.434e+05	1.57e+06	-0.347	0.729	-3.65e+06	2.57e+06
P14	-1.194e+05	3.52e+05	-0.339	0.736	-8.19e+05	5.8e+05
P15	-2.158e+05	4.94e+05	-0.436	0.664	-1.2e+06	7.66e+05
P16	-4.039e+05	5.58e+05	-0.724	0.471	-1.51e+06	7.04e+05
P17	2.054e+05	3.71e+05	0.553	0.582	-5.32e+05	9.43e+05
P18	3.361e+05	4.15e+05	0.811	0.420	-4.87e+05	1.16e+06
P19	-8.985e+04	1.45e+05	-0.620	0.536	-3.77e+05	1.98e+05
P20	-3.053e+05	1.82e+05	-1.680	0.096	-6.66e+05	5.55e+04
P21	1.497e+05	2.6e+05	0.576	0.566	-3.66e+05	6.65e+05
P22	-2.983e+05	2.72e+05	-1.095	0.276	-8.39e+05	2.42e+05
P23	1.523e+05	1.33e+05	1.149	0.254	-1.11e+05	4.16e+05
P24	5.087e+05	5.69e+05	0.895	0.373	-6.2e+05	1.64e+06
P25	3.712e+05	5.49e+05	0.676	0.500	-7.18e+05	1.46e+06
P26	-1.071e+06	6.13e+05	-1.748	0.084	-2.29e+06	1.46e+05
P27	9.661e+04	2.28e+05	0.423	0.673	-3.57e+05	5.5e+05
P28	4.75e+05	3.2e+05	1.484	0.141	-1.6e+05	1.11e+06
P29	-1.054e+05	3.42e+05	-0.308	0.759	-7.85e+05	5.74e+05
P30	6.346e+04	1.7e+05	0.373	0.710	-2.74e+05	4.01e+05
P31	1.609e+05	2.73e+05	0.590	0.557	-3.81e+05	7.02e+05
P32	-3.3e+05	2.66e+05	-1.241	0.218	-8.58e+05	1.98e+05
P33	-1.423e+05	4.45e+05	-0.320	0.750	-1.03e+06	7.41e+05
P34	-1.773e+05	4.06e+05	-0.437	0.663	-9.82e+05	6.28e+05
P35	4.356e+04	3.73e+05	0.117	0.907	-6.97e+05	7.84e+05
P36	7.372e+05	7.19e+05	1.026	0.308	-6.9e+05	2.16e+06
P37	-5645.9890	4.05e+05	-0.014	0.989	-8.09e+05	7.98e+05
Opened_range_time	-499.0440	182.909	-2.728	0.008	-862.164	-135.924

Omnibus:	74.429	Durbin-Watson:	1.749
Prob(Omnibus):	0.000	Jarque-Bera (JB):	354.909
Skew:	1.937	Prob(JB):	8.56e-78
Kurtosis:	9.868	Cond. No.	1.48e+05

Evaluate

Multiple Linear Regression model

```
y_train_pred = mlr.predict(pd.concat([X_train_preprocessed.drop(columns= ['City', 'Type', 'Ci
y_test_pred = mlr.predict(pd.concat([X_test_preprocessed.drop(columns= ['City', 'Type', 'City
# Output of data: "revenue" has oulier => using MAE
print(f"In term of training: {mean_absolute_error(y_pred= y_train_pred, y_true= y_train)}")
print(f"In term of testing: {mean_absolute_error(y_pred= y_test_pred, y_true= y_test)}")
print(f"Mean of y_train: {y_train.mean()}")
print(f"Mean of y_test: {y_test.mean()}")
```

This model is not statistically significant and does not have good performance in modeling training data and predicting testing data

```
In term of training: 5249851.477238716
In term of testing: 4810390.465060848
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
```

MAE(Training) > mean(revenue) (Training)
=> Model has bad performance at training data.

MAE(Testing) << mean(revenue) (Testing)
=> Model has good performance at testing data.

Deploy Model

KNeighborRegression model

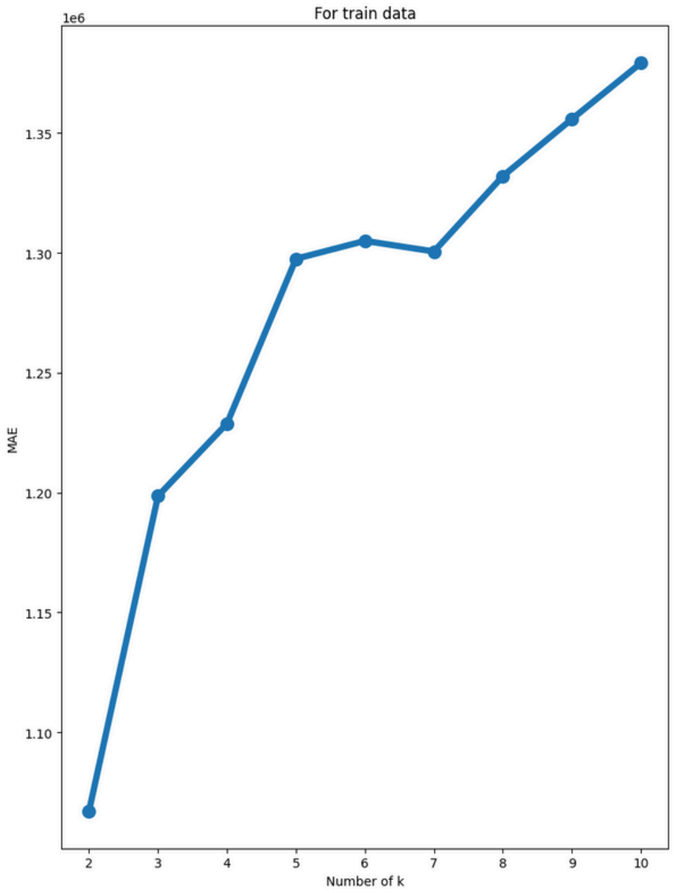
```
mae_train = []
mae_test = []
for k in range(2, 11):
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train_preprocessed, y_train)
    y_train_pred = knn.predict(X_train_preprocessed)
    y_test_pred = knn.predict(X_test_preprocessed)
    print(f"k = {k}:\n MAE_train = {mean_absolute_error(y_pred=y_train_pred, y_true= y_train)}\n MAE_tes
    mae_train.append(mean_absolute_error(y_pred=y_train_pred, y_true= y_train))
    mae_test.append(mean_absolute_error(y_pred=y_test_pred, y_true= y_test))

fig, axs = plt.subplots(1, 2, figsize = (15, 10))

axs[0].plot(range(2,11), mae_train, marker = "o", lw = 5, markersize = 10)
axs[0].set_xlabel("Number of k")
axs[0].set_ylabel("MAE")
axs[0].set_title("For train data")

axs[1].plot(range(2,11), mae_test, marker = "o", lw = 5, markersize = 10)
axs[1].set_xlabel("Number of k")
axs[1].set_ylabel("MAE")
axs[1].set_title("For test data")

plt.tight_layout()
plt.show()
```

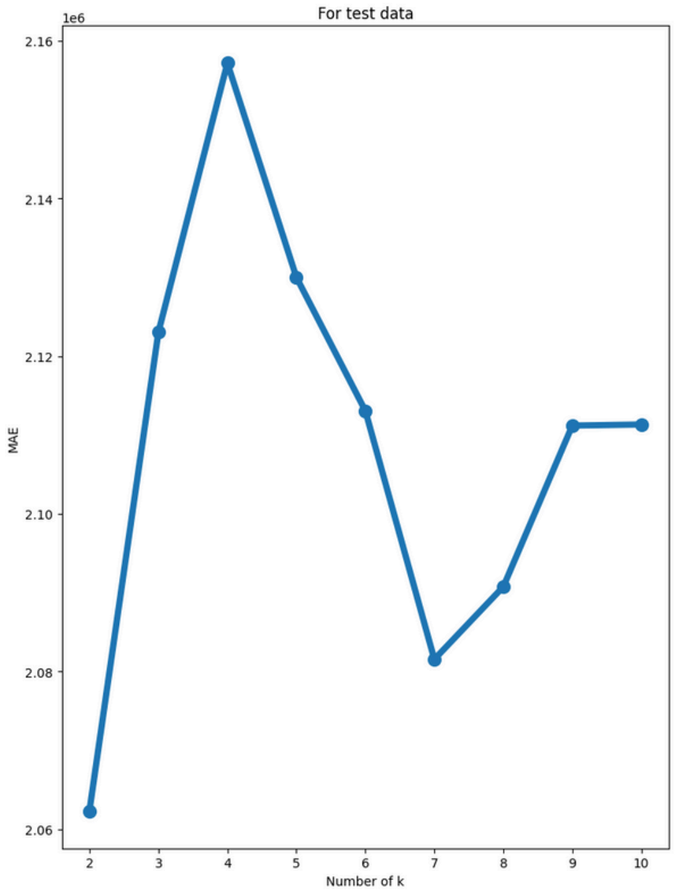
This code performs training and evaluates the performance of a regression KNN model with k values from 2 to 10. It calculates MAE for training and testing data, then plots it to compare MAE between k values. This helps determine the optimal k value for your model.

Deploy Model

KNeighborRegression model

```
k = 2:
 MAE_train = 1067178.4633027522
 MAE_test = 2062324.482142857
k = 3:
 MAE_train = 1198589.8134556573
 MAE_test = 2123032.511904762
k = 4:
 MAE_train = 1228827.7110091744
 MAE_test = 2157146.1696428573
k = 5:
 MAE_train = 1297550.6733944952
 MAE_test = 2129989.764285714
k = 6:
 MAE_train = 1305006.0733944958
 MAE_test = 2113054.339285714
k = 7:
 MAE_train = 1300596.5937090435
 MAE_test = 2081546.6275510206
k = 8:
 MAE_train = 1332055.5481651376
 MAE_test = 2090750.8080357143
k = 9:
 MAE_train = 1355844.0356778798
 MAE_test = 2111186.075396825
k = 10:
 MAE_train = 1379198.480733945
 MAE_test = 2111319.45
```





Deploy Model and Evaluate

KNeighborRegression model

```
knn = KNeighborsRegressor(n_neighbors= 7)
knn.fit(X_train_preprocessed, y_train)
y_train_pred = knn.predict(X_train_preprocessed)
y_test_pred = knn.predict(X_test_preprocessed)
# Output of data: "revenue" has oulier => using MAE
print(f"In term of training: {mean_absolute_error(y_pred= y_train_pred, y_true= y_train)}")
print(f"In term of testing: {mean_absolute_error(y_pred= y_test_pred, y_true= y_test)}")
print(f"Mean of y_train: {y_train.mean()}")
print(f"Mean of y_test: {y_test.mean()}")
```

```
In term of training: 1300596.5937090435
In term of testing: 2081546.6275510206
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
```

```
[ ]  knr_model = KNeighborsRegressor()
     param_grid_knr= {"n_neighbors" : [i for i in range(2,11)]}
     grid_search_knr = GridSearchCV(estimator= knr_model, param_grid= param_grid_knr, scoring = 'ne
     grid_search_knr.fit(X_train_preprocessed, y_train)
     print(f"Best n_estimator: {grid_search_knr.best_params_['n_neighbors']}")

     Best n_estimator: 7
```

Deploy Model and Evaluate

KNeighborRegression model

```
knn = grid_search_knr.best_estimator_
# knn.fit(X_train_preprocessed, y_train)
y_train_pred = knn.predict(X_train_preprocessed)
y_test_pred = knn.predict(X_test_preprocessed)
# Output of data: "revenue" has oulier => using MAE
print(f"In term of training: {mean_absolute_error(y_pred= y_train_pred, y_true= y_train)}")
print(f"In term of testing: {mean_absolute_error(y_pred= y_test_pred, y_true= y_test)}")
print(f"Mean of y_train: {y_train.mean()}")
print(f"Mean of y_test: {y_test.mean()}")
```

```
In term of training: 1300596.5937090435
In term of testing: 2081546.6275510206
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
```

MAE(Training) << mean(revenue) (Training)

=> Model has good performance at training data.

MAE(Testing) << mean(revenue) (Testing)

=> Model has good performance at testing data.

Deploy Model and Evaluate

RandomForest model

```
[ ]  rf_model = RandomForestRegressor(random_state= 27)
     param_grid = {"n_estimators" : [10, 50, 100, 150, 200]}

     grid_search = GridSearchCV(estimator= rf_model, param_grid= param_grid, scoring
     grid_search.fit(X_train_preprocessed, y_train)
     print(f"Best n_estimator: {grid_search.best_params_['n_estimators']}")
```

```
Best n_estimator: 50
```

```
grid_search.best_estimator_.fit(X_train_preprocessed, y_train)
y_train_pred = grid_search.best_estimator_.predict(X_train_preprocessed)
y_test_pred = grid_search.best_estimator_.predict(X_test_preprocessed)
# Output of data: "revenue" has oulier => using MAE
print(f"In term of training: {mean_absolute_error(y_pred= y_train_pred, y_true= y_train)}")
print(f"In term of testing: {mean_absolute_error(y_pred= y_test_pred, y_true= y_test)}")
print(f"Mean of y_train: {y_train.mean()}")
print(f"Mean of y_test: {y_test.mean()}")
```

```
In term of training: 564684.466055046
In term of testing: 1874973.3214285711
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
```

MAE(Training) << mean(revenue) (Training)

=> Model has good performance at training data.

MAE(Testing) << mean(revenue) (Testing)

=> Model has good performance at testing data.

4

Suggestion

Conclusion

1.About reveune

For restaurants in cities with good revenue, they should catch up to stabilize this revenue level. For restaurants in other cities, you should learn more about the factors that affect revenue for a deeper analysis. Not only distinguishing cities but also distinguishing city groups and restaurant types

2.With model to predict

There need to be more factors to analyze such as menus, ingredients, and customer profiles. Through many such small factors, the models created can more accurately predict revenue and growth direction for restaurant businesses. In this case, the best model is Random Forest.

```
Mean of y_train: 4216079.743119266
Mean of y_test: 5377902.714285715
           LinearRegression  KNeighborRegression  RandomForest
For train      1.325301e+06         1.300597e+06  5.646845e+05
For test       3.527994e+06         2.081547e+06  1.874973e+06
```

Reference

Data source: Kaggle/Restaurant_revenue

Thank you

for your attention