FINAL PROJECT

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test-Project1.csv to test-Project1.csv
Saving train-Project1.csv to train-Project1.csv
```

```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#importing datasets
data_train = pd.read_csv('train-Project1.csv')
data_test = pd.read_csv('test-Project1.csv')
data_train.head()
```

	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P
0	0	07/17/1999	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	2	5	4	5	5	
1	1	02/14/2008	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	2	5	5	5	5	
2	2	03/09/2013	Diyarbakır	Other	IL	2	4.0	2.0	5.0	2	3	5	5	5	5	
3	3	02/02/2012	Tokat	Other	IL	6	4.5	6.0	6.0	4	4	10	8	10	10	
4	4	05/09/2009	Gaziantep	Other	IL	3	4.0	3.0	4.0	2	2	5	5	5	5	

Visualizing the dataset

```
cities = data_train['City'].unique()
cities.sort()

groups = data_train['City Group'].unique()

types = data_train['Type'].unique()
```

```
citywisedata = data_train.groupby('City').mean()
citywisedata.head()
```

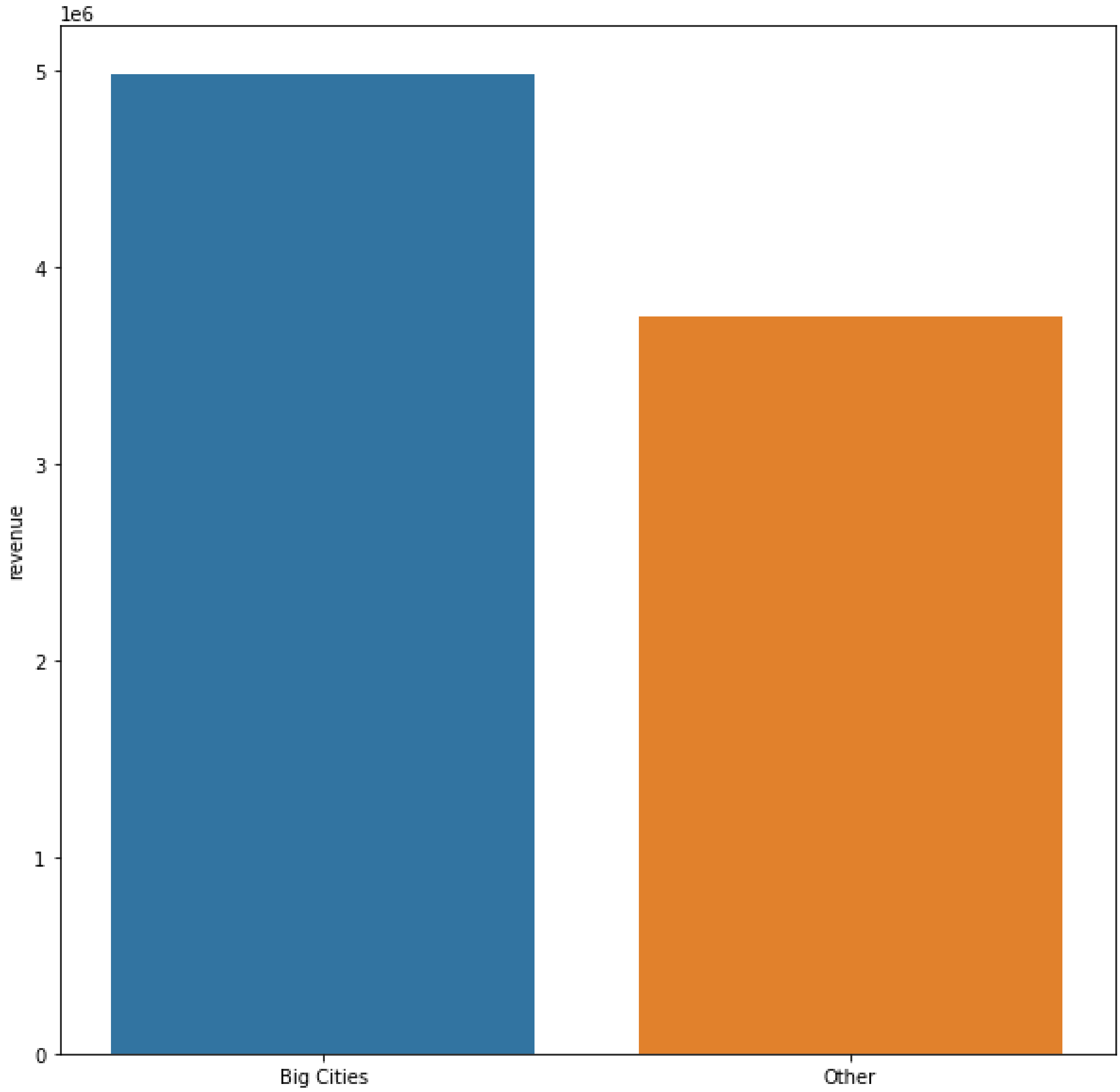
City	Id	P1	P2	P3	P4	P5	P6
Adana	72.666667	4.000000	5.000000	4.000000	3.000000	1.000000	2.666667
Afyonkarahisar	8.000000	1.000000	1.000000	4.000000	4.000000	1.000000	2.000000
Amasya	110.000000	6.000000	3.000000	6.000000	6.000000	4.000000	4.000000
Ankara	62.105263	3.526316	4.421053	4.078947	4.736842	2.789474	3.947368
Antalya	74.000000	3.000000	3.500000	5.250000	4.375000	2.000000	4.000000

```
#Plotting the mean data against each city
fig, ax = plt.subplots()
fig.set_size_inches(10, 10)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
sns.barplot(x = cities, y = citywisedata['revenue'], ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06765ab0b8>
```

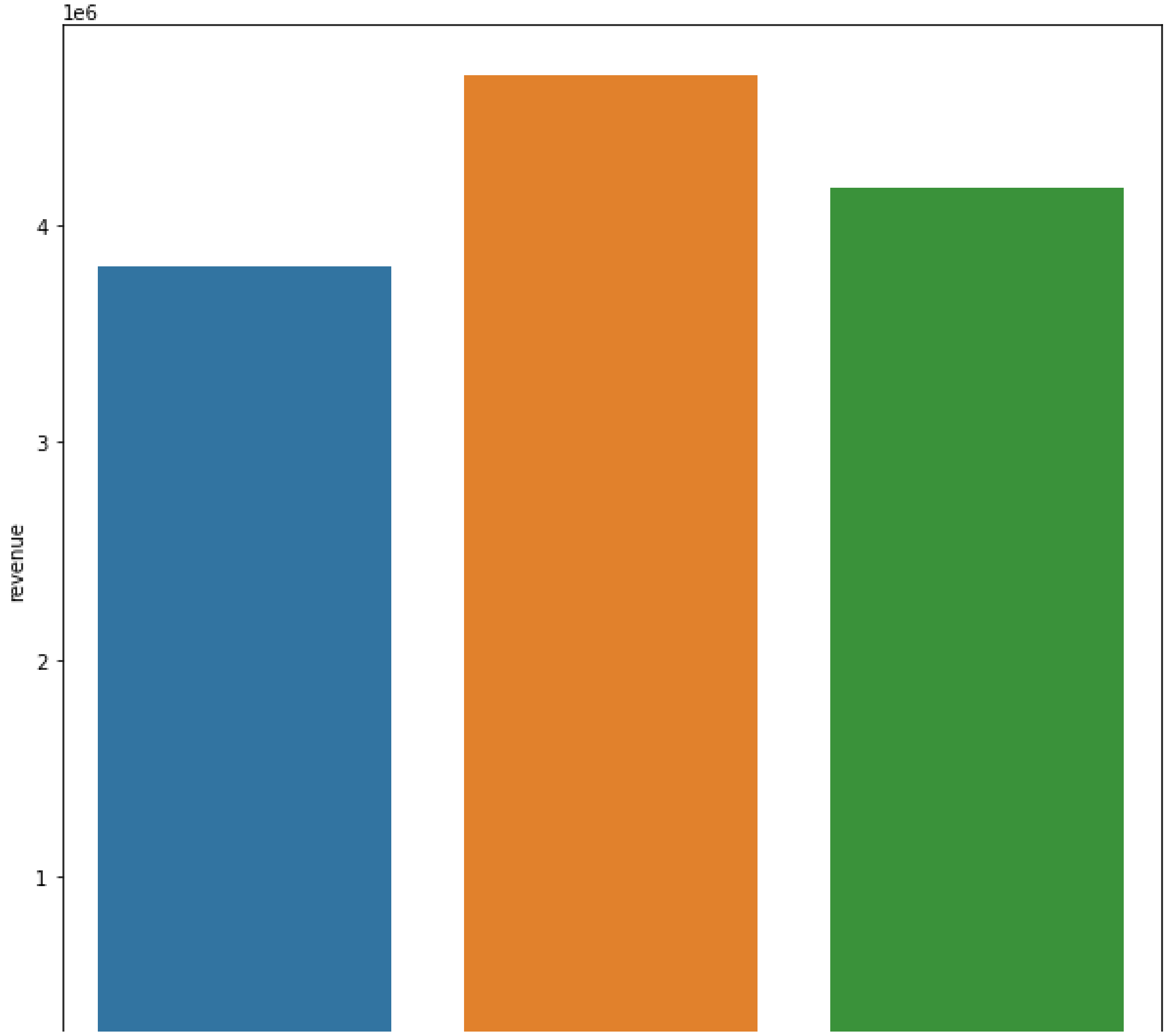
```
#PLotting the city type against its mean revenue
citytypewisedata = data_train.groupby('City Group').mean()

fig, ax = plt.subplots()
fig.set_size_inches(10, 10)

sns.barplot(x = groups, y = citytypewisedata['revenue'], ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0679952828>
```



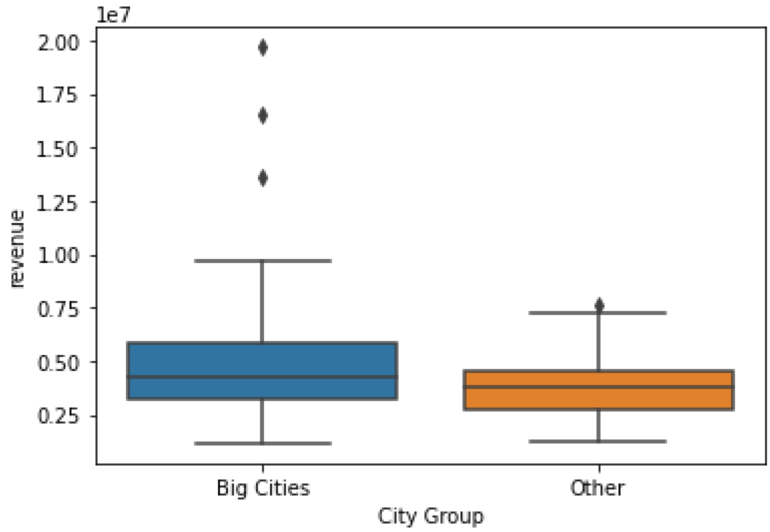

```
#Plotting each city group against its mean
citygroupwisedata = data_train.groupby('Type').mean()

fig, ax = plt.subplots()
fig.set_size_inches(10, 10)

sns.barplot(x = types, y = citygroupwisedata['revenue'], ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f067c4d9400>
```



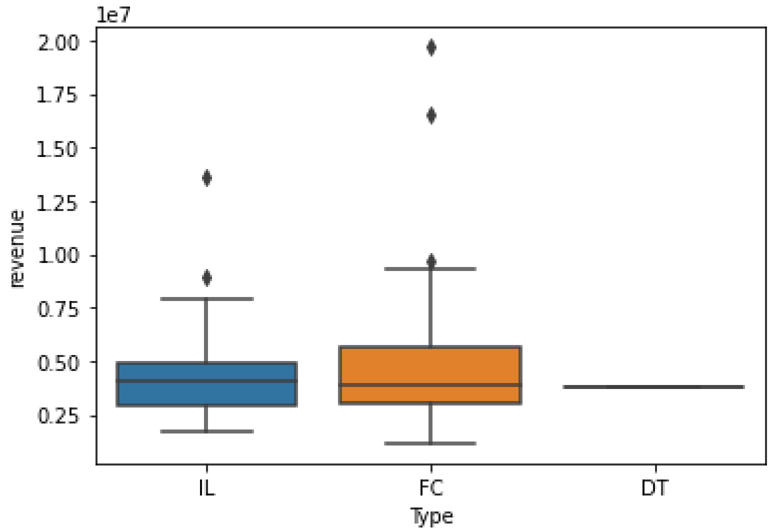

```
cat_col=data_train.select_dtypes(include='object').columns
num_col=data_train.select_dtypes(exclude='object').columns


#Plotting the categorical data against the target values

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f067c484c88>
```

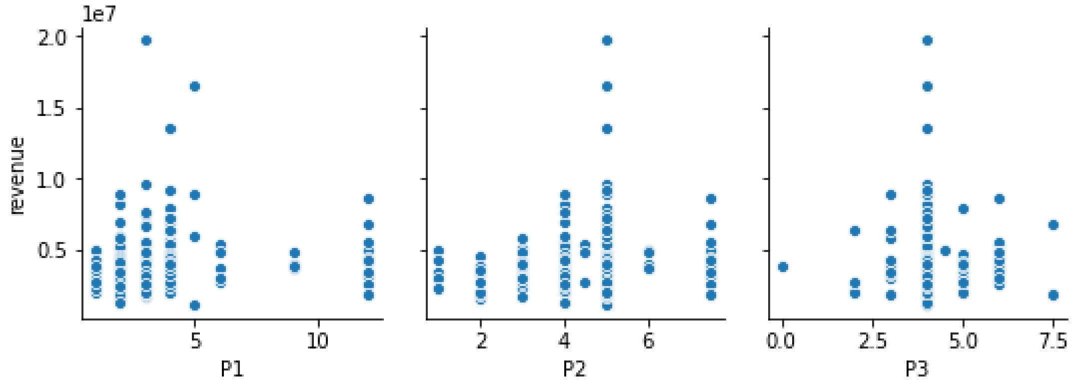



```
sns.boxplot(x = 'Type', y = 'revenue', data = data_train)
```

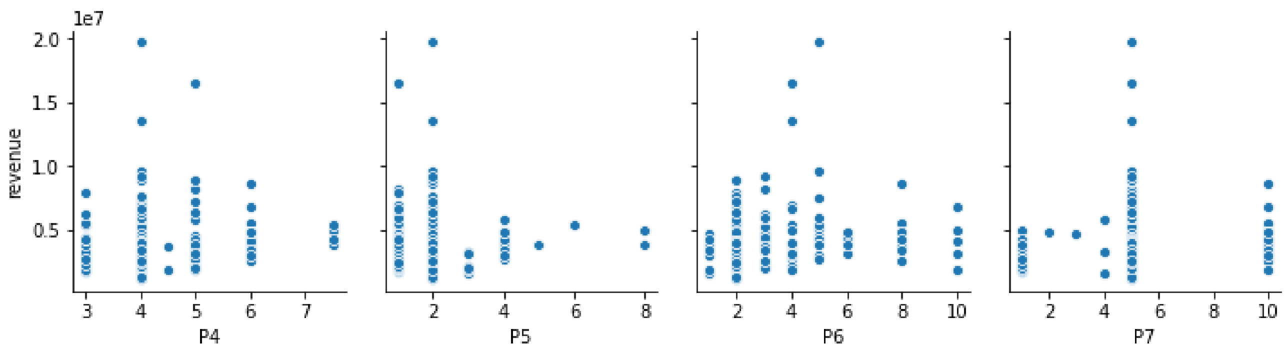
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06796c41d0>
```



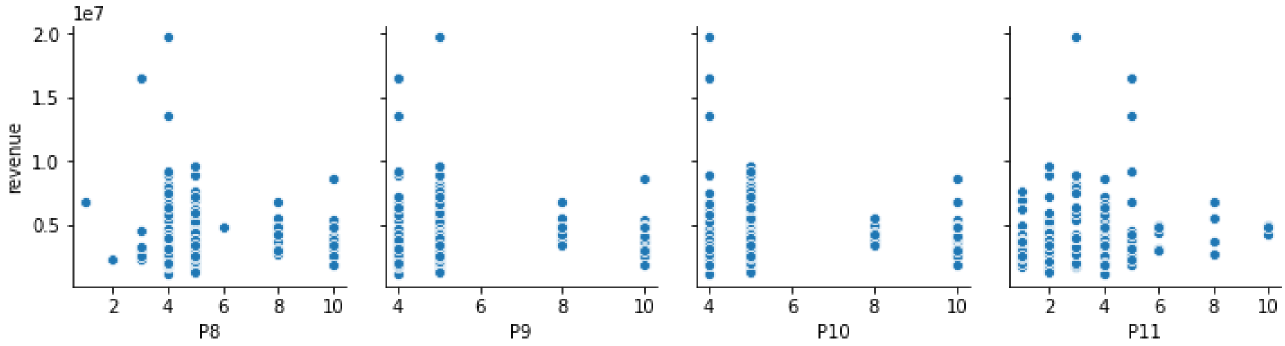

```
#Plotting the Numerical data against the Target Variable
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P1', 'P2', 'P3'])
```



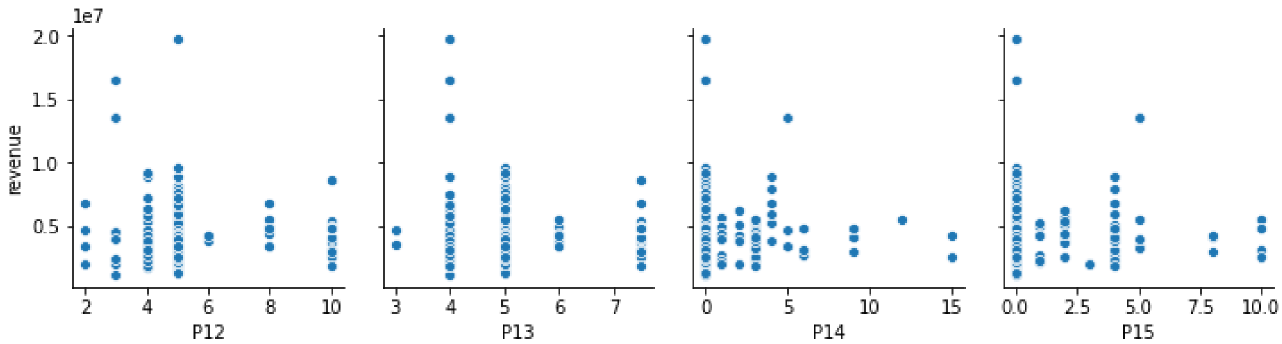

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P4', 'P5', 'P6', 'P7'])
```



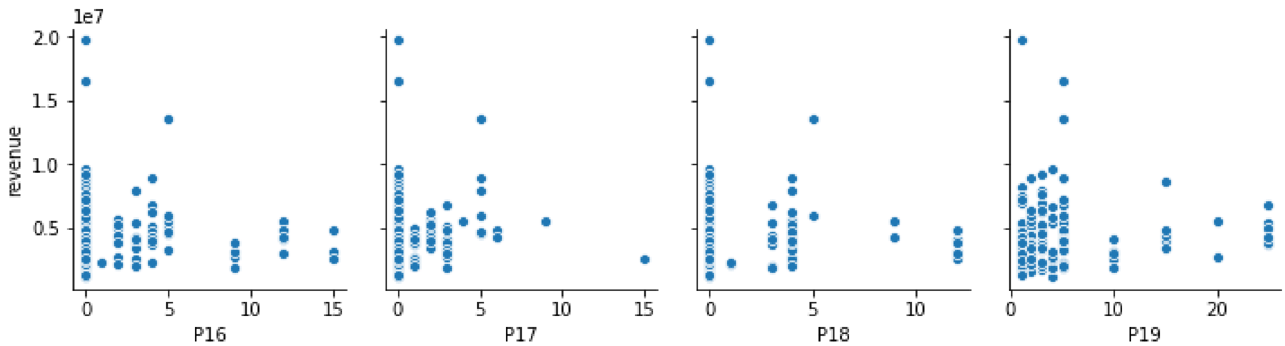

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P8', 'P9', 'P10', 'P11'])
```



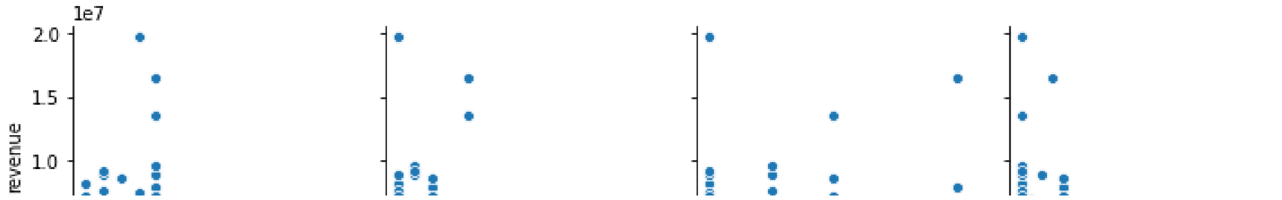

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P12', 'P13', 'P14', 'P15'])
```



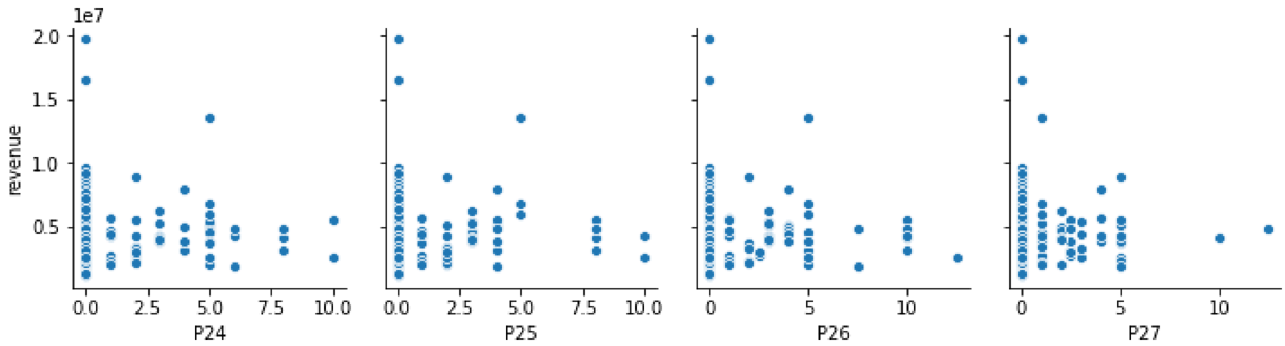

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P16', 'P17','P18', 'P19'])
```



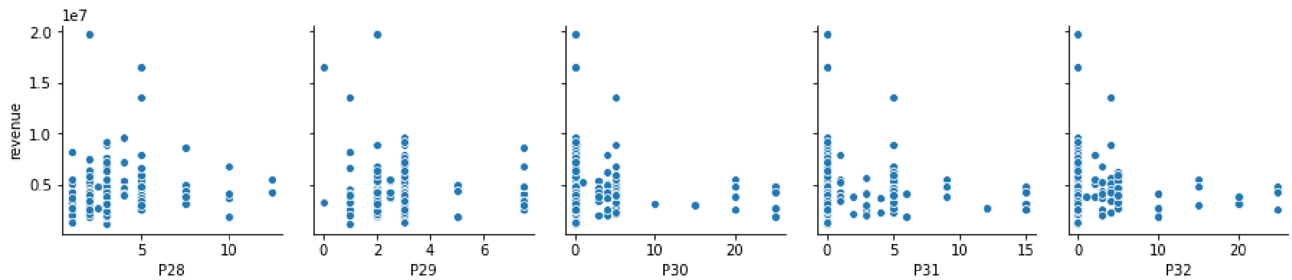

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P20', 'P21', 'P22', 'P23'])
```



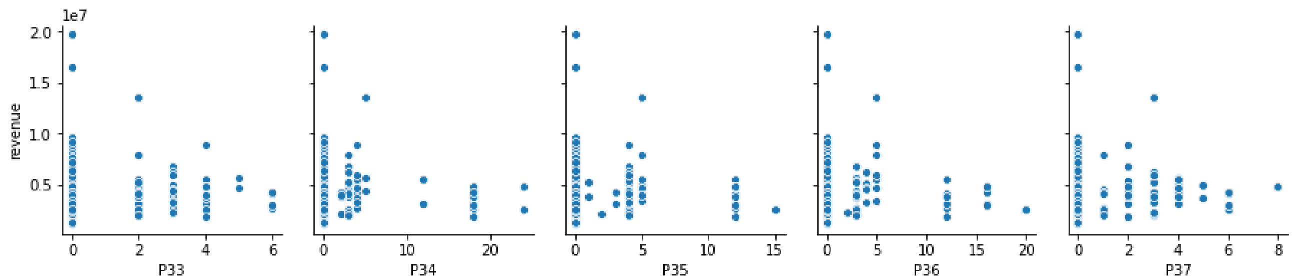

```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P24', 'P25', 'P26', 'P27'])
```




```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P28', 'P29', 'P30', 'P31', 'P32'])
```




```
pp = sns.pairplot(data=data_train,
                  y_vars=['revenue'],
                  x_vars=['P33', 'P34', 'P35', 'P36', 'P37'])
```



Preprocessing the dataset

```
#Creating a flag for each type of restaurant
data_train['Type_IL'] = np.where(data_train['Type'] == 'IL', 1, 0)
data_train['Type_FC'] = np.where(data_train['Type'] == 'FC', 1, 0)
data_train['Type_DT'] = np.where(data_train['Type'] == 'DT', 1, 0)

#Creating a flag for 'Big Cities'
data_train['Big_Cities'] = np.where(data_train['City Group'] == 'Big Cities', 1, 0)

#Converting Open_Date into day count
#Considering the same date the dataset was made available
data_train['Days_Open'] = (pd.to_datetime('2015-03-23') - pd.to_datetime(data_train['Open

#Removing unused columns
data_train = data_train.drop('Type', axis=1)
data_train = data_train.drop('City Group', axis=1)
data_train = data_train.drop('City', axis=1)
data_train = data_train.drop('Open Date', axis=1)

#Adjusting test data as well
data_test['Type_IL'] = np.where(data_test['Type'] == 'IL', 1, 0)
data_test['Type_FC'] = np.where(data_test['Type'] == 'FC', 1, 0)
data_test['Type_DT'] = np.where(data_test['Type'] == 'DT', 1, 0)
data_test['Big_Cities'] = np.where(data_test['City Group'] == 'Big Cities', 1, 0)
data_test['Days_Open'] = (pd.to_datetime('2015-03-23') - pd.to_datetime(data_test['Open Da
data_test = data_test.drop('Type', axis=1)
data_test = data_test.drop('City Group', axis=1)
data_test = data_test.drop('City', axis=1)
data_test = data_test.drop('Open Date', axis=1)
data_train.head()
```

	Id	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17
0	0	4	5.0	4.0	4.0	2	2	5	4	5	5	3	5	5.0	1	2	2	2
1	1	4	5.0	4.0	4.0	1	2	5	5	5	5	1	5	5.0	0	0	0	0
2	2	2	4.0	2.0	5.0	2	3	5	5	5	5	2	5	5.0	0	0	0	0
3	3	6	4.5	6.0	6.0	4	4	10	8	10	10	8	10	7.5	6	4	9	3
4	4	3	4.0	3.0	4.0	2	2	5	5	5	5	2	5	5.0	2	1	2	1

Implementing the models

```
X = data_train.drop(['Id', 'revenue'], axis=1)
Y = data_train.revenue
```

```
#Implementing Multiple Regression
from sklearn.linear_model import LinearRegression
from sklearn import metrics
regressor = LinearRegression()
regressor.fit(X, Y)

test_predicted_mreg = pd.DataFrame()
test_predicted_mreg['Id'] = data_test.Id
test_predicted_mreg['Prediction'] = regressor.predict(data_test.drop('Id', axis=1))
test_predicted_mreg.head()
```

	Id	Prediction
0	0	4.669238e+06
1	1	2.741557e+06
2	2	1.815584e+06
3	3	5.312600e+06
4	4	5.493380e+06

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = LinearRegression()
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 3004513.8682692987
Mean Squared Error: 19196941411388.652
Root Mean Squared Error: 4381431.434062235
```

```
#Implementing Logistic Regression
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(X, Y)

test_predicted_lreg = pd.DataFrame()
test_predicted_lreg['Id'] = data_test.Id
test_predicted_lreg['Prediction'] = reg.predict(data_test.drop('Id', axis=1))
test_predicted_lreg.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

	Id	Prediction
0	0	4888774.0
1	1	4067566.0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = LogisticRegression()
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 2065004.2142857143
Mean Squared Error: 13982633811870.072
Root Mean Squared Error: 3739336.012164469
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
#Implementing Support Vector Machines
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state = 1)
classifier.fit(X, Y)

test_predicted_svm = pd.DataFrame()
test_predicted_svm['Id'] = data_test.Id
test_predicted_svm['Prediction'] = classifier.predict(data_test.drop('Id', axis=1))
test_predicted_svm.head()
```

	Id	Prediction

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = SVC(kernel='rbf', random_state = 1)
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 2357957.785714286
Mean Squared Error: 9352554362343.428
Root Mean Squared Error: 3058194.624667212
```

```
#Implementing Ridge and Lasso Model
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn import metrics

#Lasso Regression
model = Lasso(alpha=5.5)
model.fit(X, Y)

test_predicted = pd.DataFrame()
test_predicted['Id'] = data_test.Id
test_predicted['Prediction'] = model.predict(data_test.drop('Id', axis=1))
test_predicted.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:47
  positive)
```

	Id	Prediction
0	0	4.669774e+06
1	1	2.740680e+06
2	2	1.815398e+06
3	3	5.311360e+06
4	4	5.492584e+06

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = Lasso(alpha=5.5)
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 3004980.5481886053
Mean Squared Error: 19201868342792.414
Root Mean Squared Error: 4381993.649332734
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:47
  positive)
```

```
#Ridge Regression
model = Ridge(alpha=330)
model.fit(X, Y)

test_predicted_ridge = pd.DataFrame()
test_predicted_ridge['Id'] = data_test.Id
test_predicted_ridge['Prediction'] = model.predict(data_test.drop('Id', axis=1))
test_predicted_ridge.head()
```

	Id	Prediction
0	0	4.229698e+06
1	1	3.822859e+06
2	2	3.636879e+06
3	3	3.882530e+06
4	4	3.957615e+06

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = Ridge(alpha=330)
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1930832.797409726
Mean Squared Error: 10032310013899.816
Root Mean Squared Error: 3167382.202055795
```

```
#Random Forest Regression Implementation
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=150)
model.fit(X, Y)

test_predicted_forest = pd.DataFrame()
```

```
test_predicted_forest['Id'] = data_test.Id
test_predicted_forest['Prediction'] = model.predict(data_test.drop('Id', axis=1))
test_predicted_forest.head()
```

	Id	Prediction
0	0	4.057964e+06
1	1	3.493131e+06
2	2	3.528119e+06
3	3	3.420063e+06
4	4	3.953846e+06

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = RandomForestRegressor(n_estimators=150)
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1612414.5004761906
Mean Squared Error: 9956576656014.11
Root Mean Squared Error: 3155404.3569745715
```

```
#Decision tree Regression Model
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, Y)

test_predicted_tree = pd.DataFrame()
test_predicted_tree['Id'] = data_test.Id
test_predicted_tree['Prediction'] = model.predict(data_test.drop('Id', axis=1))
test_predicted_tree.head()
```

	Id	Prediction
0	0	4.057964e+06
1	1	3.493131e+06
2	2	3.528119e+06
3	3	3.420063e+06
4	4	3.953846e+06

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

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regressor_accuracy = DecisionTreeRegressor(random_state = 0)
regressor_accuracy.fit(X_train, y_train)

y_pred = regressor_accuracy.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 1917038.7142857143
Mean Squared Error: 10582062293044.715
Root Mean Squared Error: 3253008.1913583796
```