Models used

1.Lasso & Ridge Regression 2. ElasticNet
3. KNN Regressor
4. Random Forest
5. Light GBM
6. XGBoost
7. Ensembling

In [1]:
```
url='https://www.kaggle.com/c/restaurant-revenue-prediction/rules'
```

In [2]:
```
import opendatasets as od
import os
```

In [5]:
```
od.download(url)
```

```
Please provide your Kaggle credentials to download this dataset. Learn mor
e: http://bit.ly/kaggle-creds (http://bit.ly/kaggle-creds)
Your Kaggle username: swaroopss
Your Kaggle Key: ········

  0%|
| 0.00/2.68M [00:00<?, ?B/s]

Downloading restaurant-revenue-prediction.zip to .\restaurant-revenue-predi
ction

100%|██████████████████████████████████████████
███████| 2.68M/2.68M [00:44<00:00, 63.5kB/s]


Extracting archive .\restaurant-revenue-prediction/restaurant-revenue-predi
ction.zip to .\restaurant-revenue-prediction
```

In [12]:
```
od.download(url)
```

```
Skipping, found downloaded files in ".\restaurant-revenue-prediction" (use
force=True to force download)
```

In [6]:
```
import json

with open('path/kaggle.json','r') as f:
    d=json.load(f)
    print(d)
```

...

In [ ]:
```
## alternatively we can use this
"""""
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

""""""
```

In [14]:
```
os.listdir('./restaurant-revenue-prediction')
```

Out[14]:
```
['sampleSubmission.csv', 'test.csv.zip', 'train.csv.zip']
```

In [608]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from matplotlib.pylab import rcParams
import matplotlib
%matplotlib inline
```

In [655]:
```
sns.set_style('darkgrid')
matplotlib.rcParams['figure.figsize']=(20,30)
matplotlib.rcParams['font.size']=(10)
```

In [656]:
```
data=pd.read_csv('./restaurant-revenue-prediction/train.csv.zip')
```

In [657]:
```
data.shape
```

Out[657]:
```
(137, 43)
```

In [658]:
```
pd.set_option('display.max_columns', None)
```

In [659]:
```
data
```

Out[659]:

	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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
132	132	06/25/2008	Trabzon	Other	FC	2	3.0	3.0	5.0	4	2	4	4	4	4	
133	133	10/12/2006	İzmir	Big Cities	FC	4	5.0	4.0	4.0	2	3	5	4	4	5	
134	134	07/08/2006	Kayseri	Other	FC	3	4.0	4.0	4.0	2	3	5	5	5	5	
135	135	10/29/2010	İstanbul	Big Cities	FC	4	5.0	4.0	5.0	2	2	5	5	5	5	
136	136	09/01/2009	İstanbul	Big Cities	FC	4	5.0	3.0	5.0	2	2	5	4	4	5	

137 rows × 43 columns

In [660]:
```
data.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  
```

In [661]:
```
data.describe()
```

Out[661]:

	Id	P1	P2	P3	P4	P5	P6	
count	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	137.000000	1
mean	68.000000	4.014599	4.408759	4.317518	4.372263	2.007299	3.357664	
std	39.692569	2.910391	1.514900	1.032337	1.016462	1.209620	2.134235	
min	0.000000	1.000000	1.000000	0.000000	3.000000	1.000000	1.000000	
25%	34.000000	2.000000	4.000000	4.000000	4.000000	1.000000	2.000000	
50%	68.000000	3.000000	5.000000	4.000000	4.000000	2.000000	3.000000	
75%	102.000000	4.000000	5.000000	5.000000	5.000000	2.000000	4.000000	
max	136.000000	12.000000	7.500000	7.500000	7.500000	8.000000	10.000000	

In [662]:
```
data.isna().sum()
```

Out[662]:
```
Id            0
Open Date     0
City          0
City Group    0
Type          0
P1            0
P2            0
P3            0
P4            0
P5            0
P6            0
P7            0
P8            0
P9            0
P10           0
P11           0
P12           0
P13           0
P14           0
```

In [663]:

```
display(data)
```

	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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
132	132	06/25/2008	Trabzon	Other	FC	2	3.0	3.0	5.0	4	2	4	4	4	4	
133	133	10/12/2006	İzmir	Big Cities	FC	4	5.0	4.0	4.0	2	3	5	4	4	5	
134	134	07/08/2006	Kayseri	Other	FC	3	4.0	4.0	4.0	2	3	5	5	5	5	
135	135	10/29/2010	İstanbul	Big Cities	FC	4	5.0	4.0	5.0	2	2	5	5	5	5	
136	136	09/01/2009	İstanbul	Big Cities	FC	4	5.0	3.0	5.0	2	2	5	4	4	5	

137 rows × 43 columns

In [664]:
```
test_df=pd.read_csv('./restaurant-revenue-prediction/test.csv.zip')
test_df
```

Out[664]:

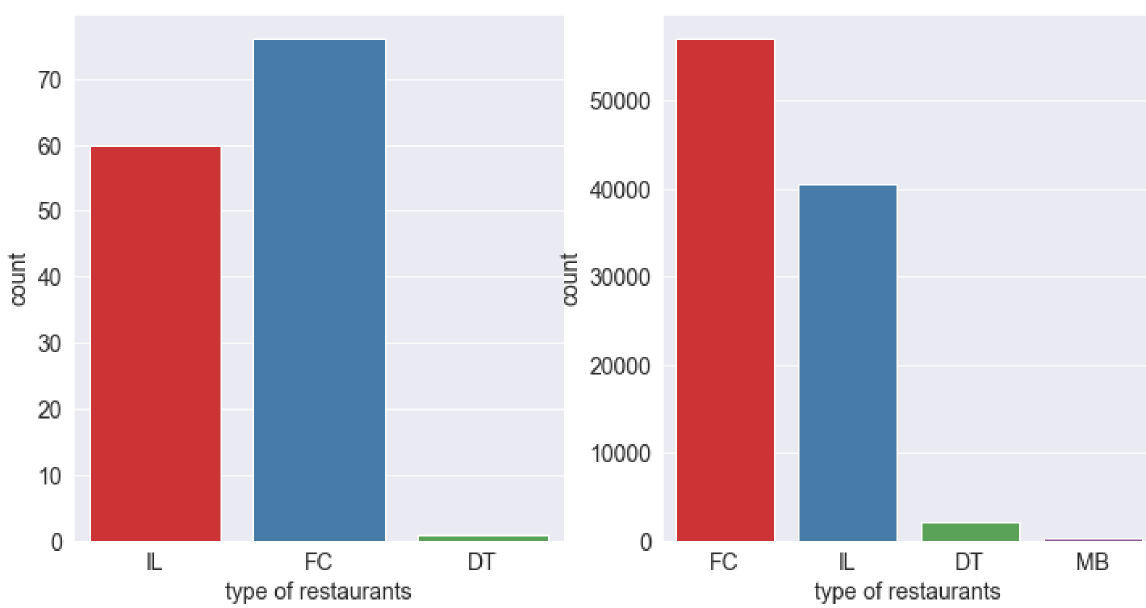
	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	P9
0	0	01/22/2011	Niğde	Other	FC	1	4.0	4.0	4.0	1	2	5	4	5
1	1	03/18/2011	Konya	Other	IL	3	4.0	4.0	4.0	2	2	5	3	4
2	2	10/30/2013	Ankara	Big Cities	FC	3	4.0	4.0	4.0	2	2	5	4	4
3	3	05/06/2013	Kocaeli	Other	IL	2	4.0	4.0	4.0	2	3	5	4	5
4	4	07/31/2013	Afyonkarahisar	Other	FC	2	4.0	4.0	4.0	1	2	5	4	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	99995	01/05/2000	Antalya	Other	FC	5	5.0	4.0	4.0	2	2	5	5	4
99996	99996	07/18/2011	Niğde	Other	IL	1	2.0	4.0	3.0	1	1	1	5	5
99997	99997	12/29/2012	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	1	2	5	3	4
99998	99998	10/12/2013	İstanbul	Big Cities	FC	12	7.5	6.0	6.0	4	4	10	10	10
99999	99999	10/05/2010	İstanbul	Big Cities	IL	2	5.0	4.0	4.0	2	2	5	5	5

100000 rows × 42 columns

In [665]:

```
#plt.figure(figsize=(30,5))
plt.rc('font', size=14)
fig, ax= plt.subplots(1,2, figsize=(12,6));
g1=sns.countplot(data['Type'],ax=ax[0], palette='Set1' );
ax[0].set_xlabel("type of restaurants", )
g1=sns.countplot(test_df['Type'], ax=ax[1], palette='Set1');
plt.xlabel("type of restaurants")
fig.show();
```

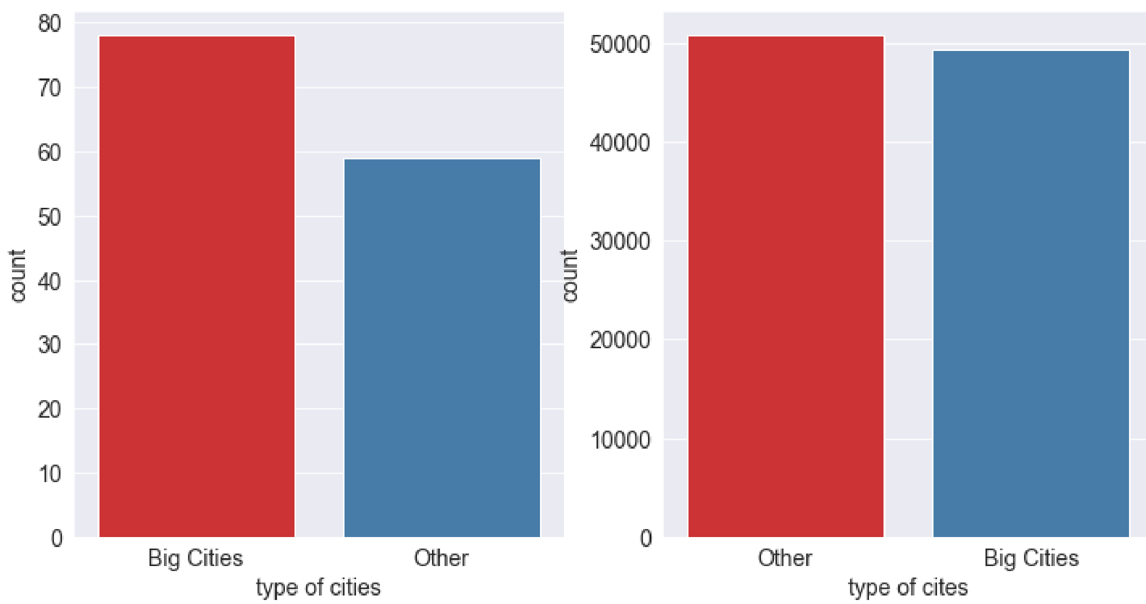
```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.

c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.

c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:8: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, whic
h is a non-GUI backend, so cannot show the figure.
```



In [666]:

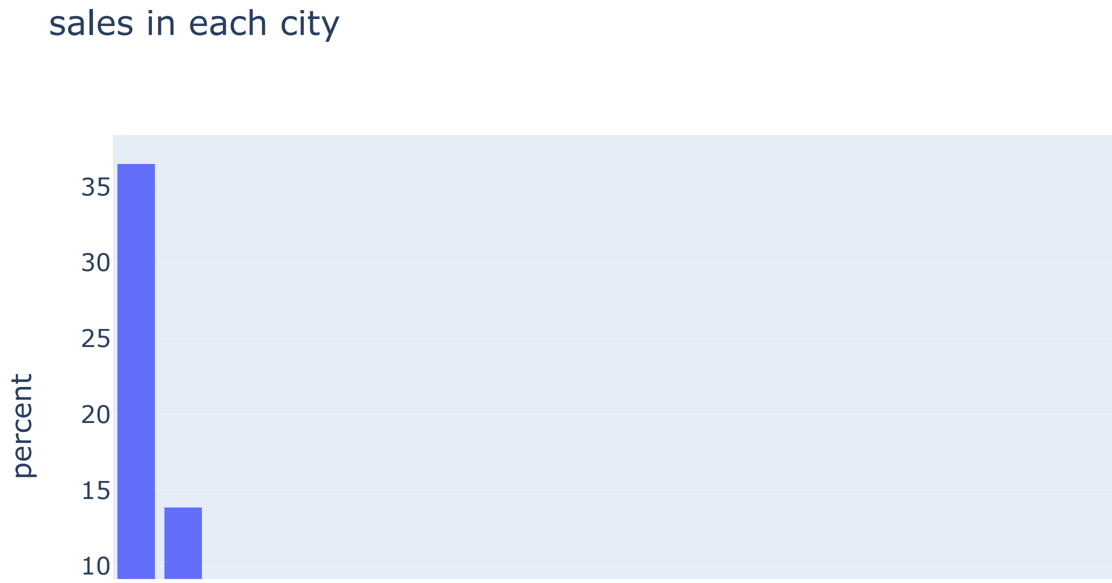
```
fig, ax=plt.subplots(1,2, figsize=(12,6))
sns.countplot(data['City Group'], palette='Set1', ax=ax[0],)
sns.countplot(test_df['City Group'], palette='Set1', ax=ax[1])
ax[0].set_xlabel('type of cities')
ax[1].set_xlabel('type of cites')
fig.show()
```

```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.

c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.

c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:6: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, whic
h is a non-GUI backend, so cannot show the figure.
```



In [667]:

```
fig1=px.histogram(data.City, x=data.City, title='sales in each city', histnor
fig1.show()
fig2=px.histogram(test_df.City, x=test_df.City, title='sales in each city', h
fig.show()
```




```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:4: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, whic
h is a non-GUI backend, so cannot show the figure.
```

\observation

number of cities in test df is more than train tf

In [668]:

```
data.City.nunique(), test_df.City.nunique()
```

Out[668]: (34, 57)

replacing MB(: mobile) with DT(: drive through) in test data as there are no category MB in train dataset

In [669]:
```
test_df.Type[test_df.Type=='MB']='DT'
```

```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
```

In [670]:
```
### we replaced 250 of such MB
test_df.Type[test_df.Type=='MB']
```

Out[670]:
```
Series([], Name: Type, dtype: object)
```

City column is useless as we have 34 unique cities in train data but 54 of them in test data. we cannot use that for prediction.

In [671]:
```
data.drop('City', axis=1, inplace=True)
test_df.drop('City', axis=1, inplace=True)
```

In [672]:
```
data.drop('Id', axis=1, inplace=True)
```

In [673]:
```
year=pd.to_datetime(data['Open Date']).dt.year
```

In [674]:
```
year.unique()
```

Out[674]:
```
array([1999, 2008, 2013, 2012, 2009, 2010, 2011, 2000, 2014, 2006, 1998,
       1996, 2004, 2007, 2005, 2002, 1997], dtype=int64)
```

As 'open Date' is not much usefull lets make it more usefull by doing some feature engineering

In [675]:
```
import datetime
data['Open Date']=pd.to_datetime(data['Open Date'])
### using base year to claculate number of days restaurant had been opened si
cal_year=datetime.datetime(2015,8,13)

### scaling date and producing new column
### 1000 scaling is used to produce better result while using in model
data['DaysOfOpen']=(cal_year-data['Open Date']).dt.days/1000
data.drop(['Open Date'], inplace= True, axis=1)
```

In [676]:

```
data
```

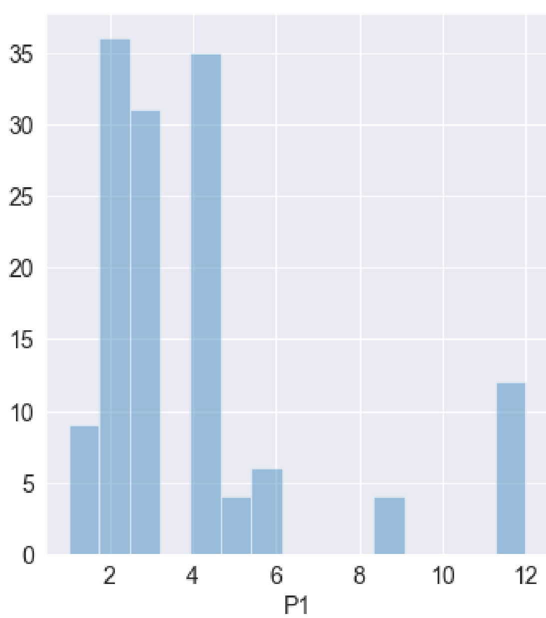
Out[676]:

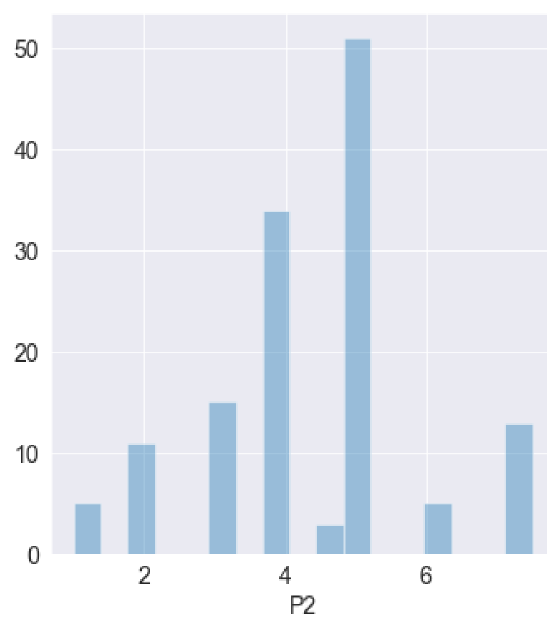
	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P1
0	Big Cities	IL	4	5.0	4.0	4.0	2	2	5	4	5	5	3	5	5.0	1	2	
1	Big Cities	FC	4	5.0	4.0	4.0	1	2	5	5	5	5	1	5	5.0	0	0	
2	Other	IL	2	4.0	2.0	5.0	2	3	5	5	5	5	2	5	5.0	0	0	
3	Other	IL	6	4.5	6.0	6.0	4	4	10	8	10	10	8	10	7.5	6	4	
4	Other	IL	3	4.0	3.0	4.0	2	2	5	5	5	5	2	5	5.0	2	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
132	Other	FC	2	3.0	3.0	5.0	4	2	4	4	4	4	4	4	4.0	0	0	
133	Big Cities	FC	4	5.0	4.0	4.0	2	3	5	4	4	5	5	4	5.0	0	0	
134	Other	FC	3	4.0	4.0	4.0	2	3	5	5	5	5	1	5	5.0	0	0	
135	Big Cities	FC	4	5.0	4.0	5.0	2	2	5	5	5	5	2	5	5.0	0	0	
136	Big Cities	FC	4	5.0	3.0	5.0	2	2	5	4	4	5	4	4	5.0	0	0	

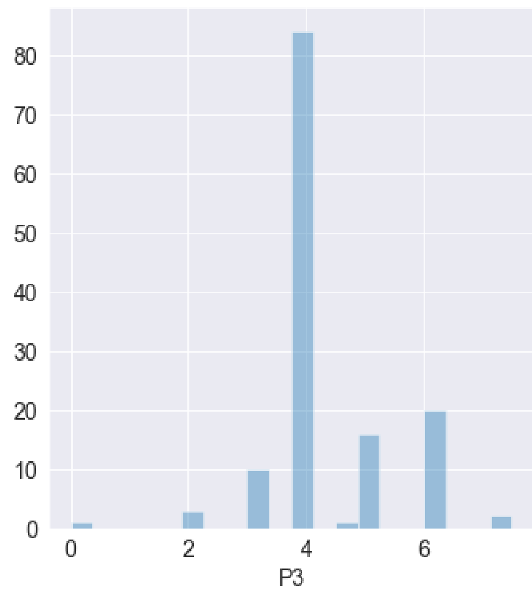
137 rows × 41 columns

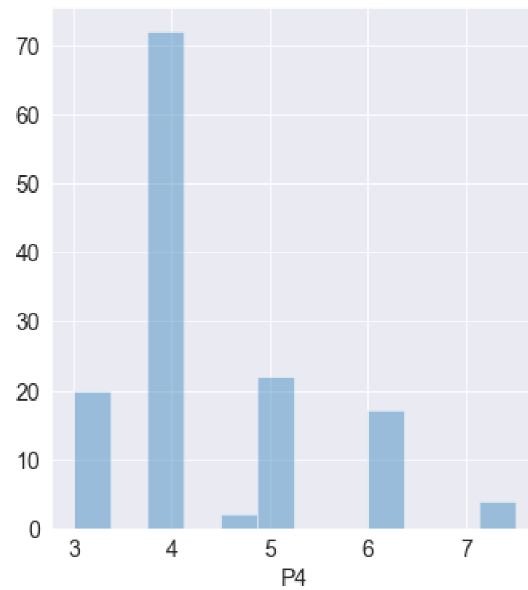
In [677]:
```
plt.rc('figure', max_open_warning=0)

for i in range(1,38,2):
    fig, ax=plt.subplots(1,2, figsize=(12,6))
    ax1=sns.distplot(data['P{}'.format(i)],kde=False, ax=ax[0])
    i=i+1
    if i<38:
        ax2=sns.distplot(data['P{}'.format(i)], kde=False, ax=ax[1])
        fig.show()
```

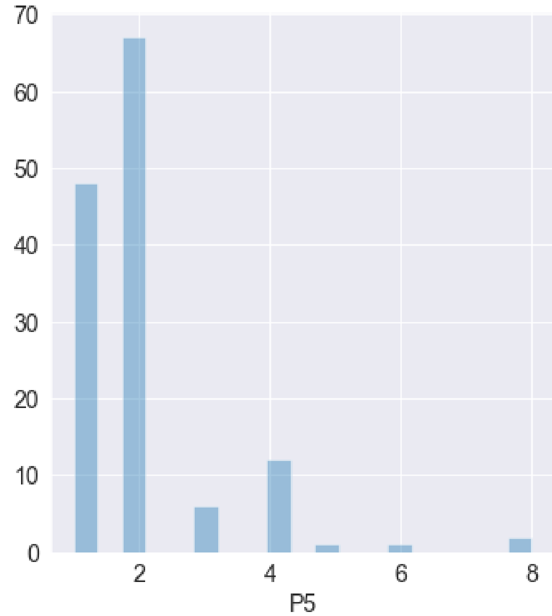
```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogr
ams).

c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:9: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, whic
h is a non-GUI backend, so cannot show the figure.
```

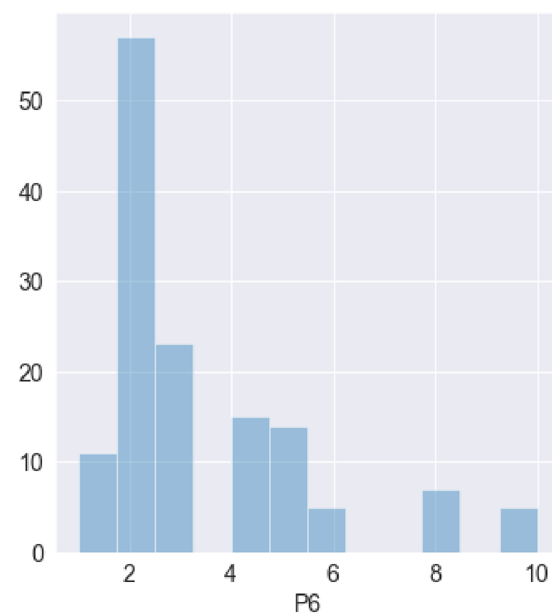


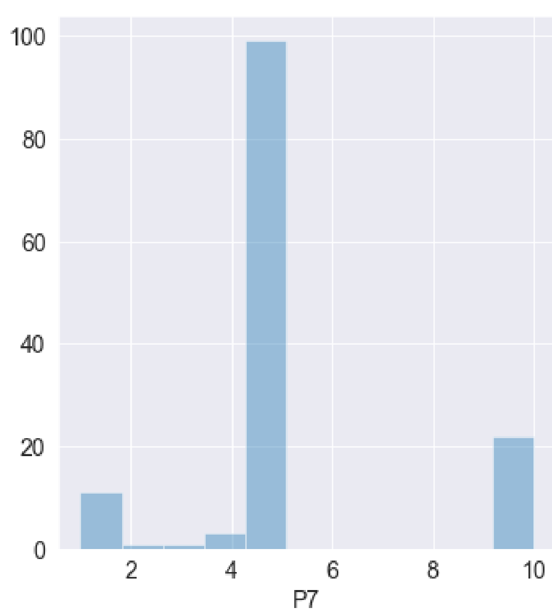


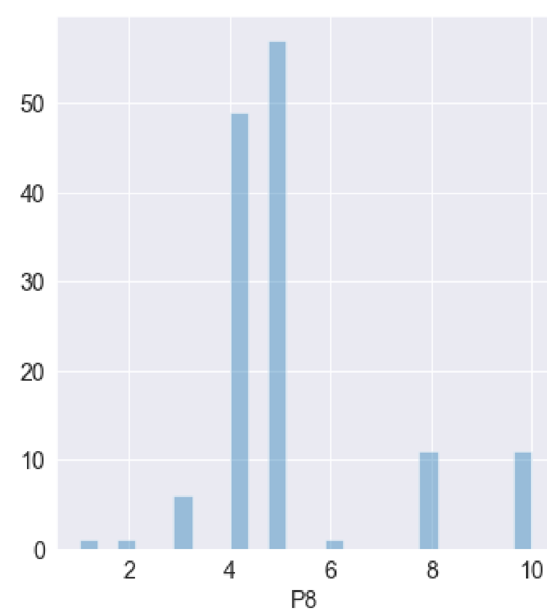

0
10
20
30
40
50
60
70
80
0 2 4 6
P3

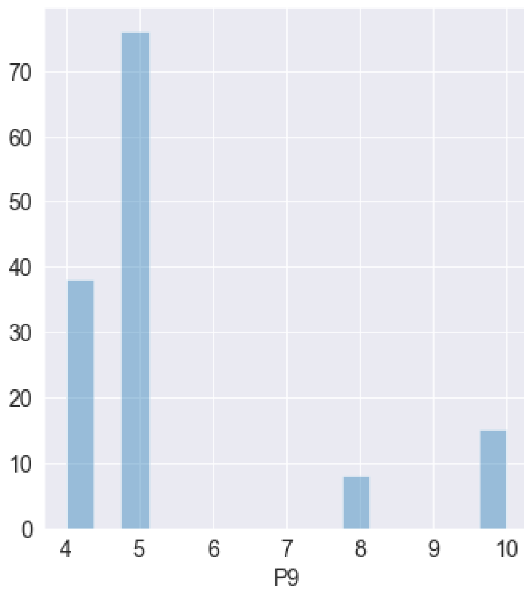

0
10
20
30
40
50
60
70
3 4 5 6 7
P4


0
10
20
30
40
50
60
70
2 4 6 8
P5

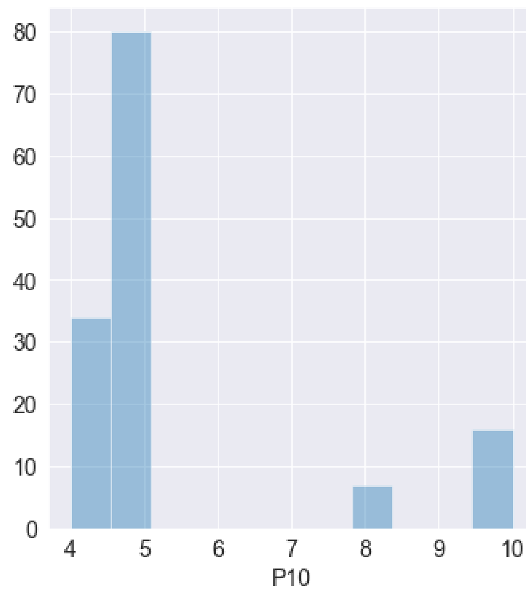

0
10
20
30
40
50
2 4 6 8 10
P6


0
20
40
60
80
100
2 4 6 8 10
P7

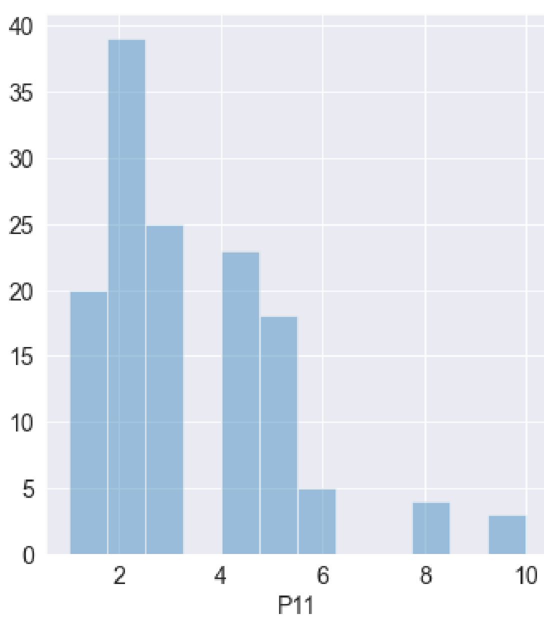

0
10
20
30
40
50
2 4 6 8 10
P8



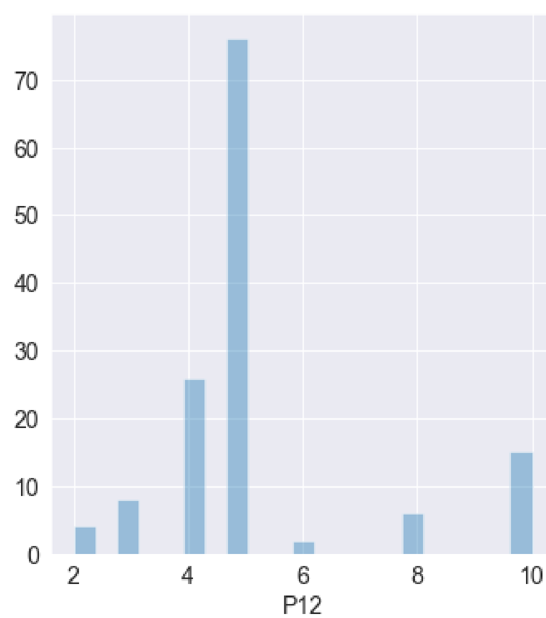
P9



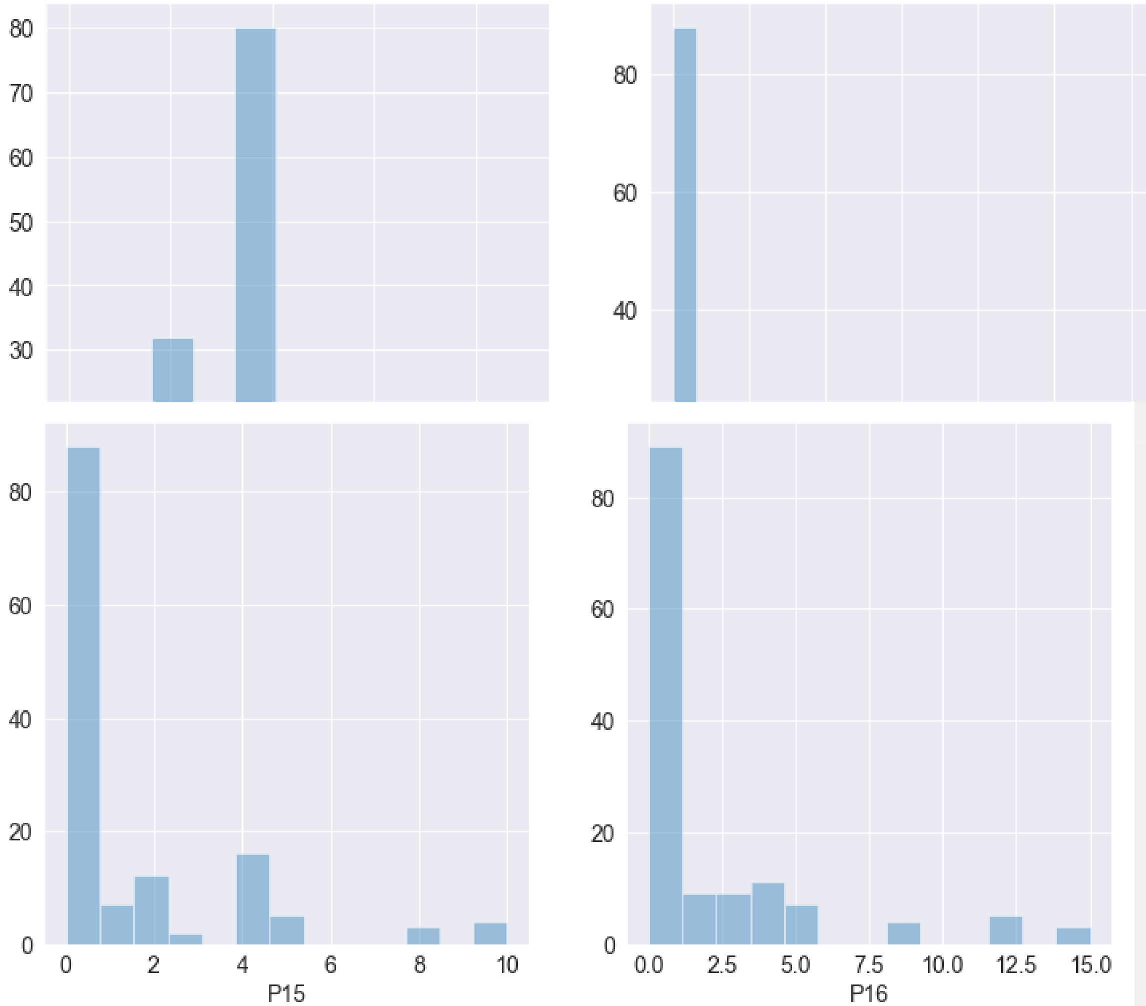
P10

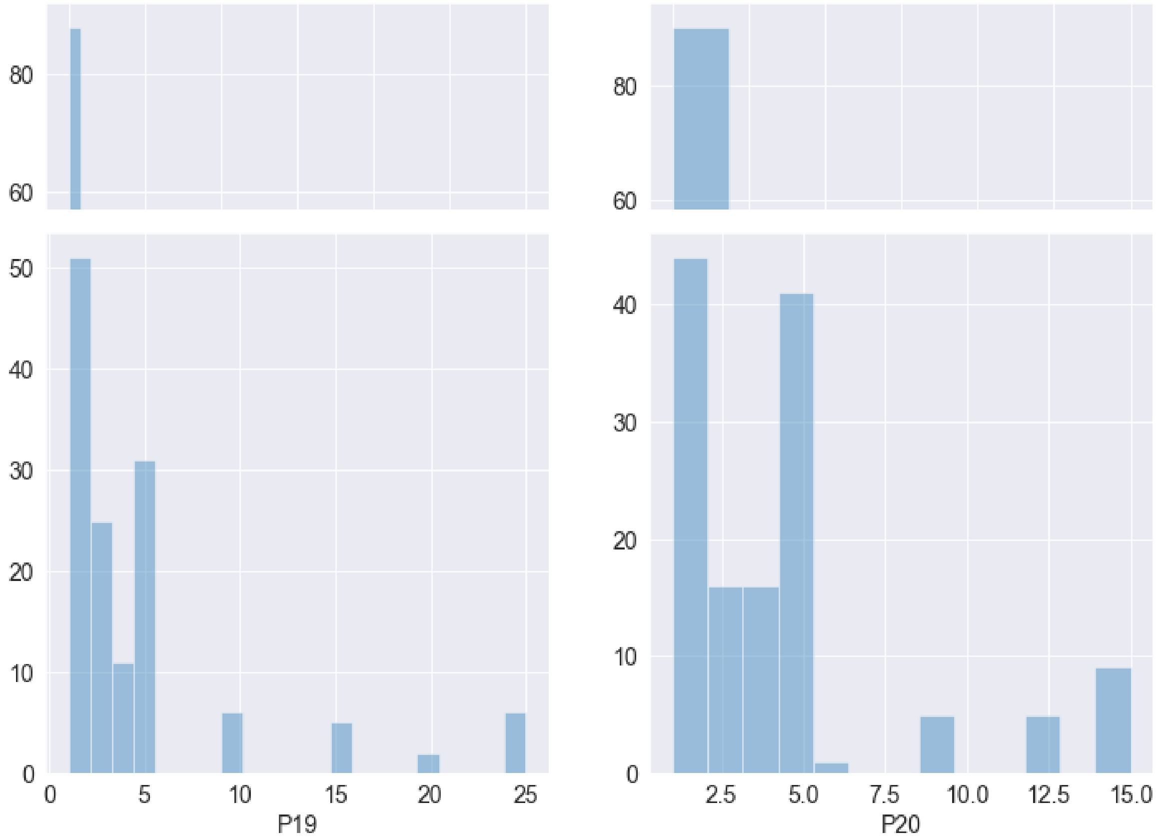


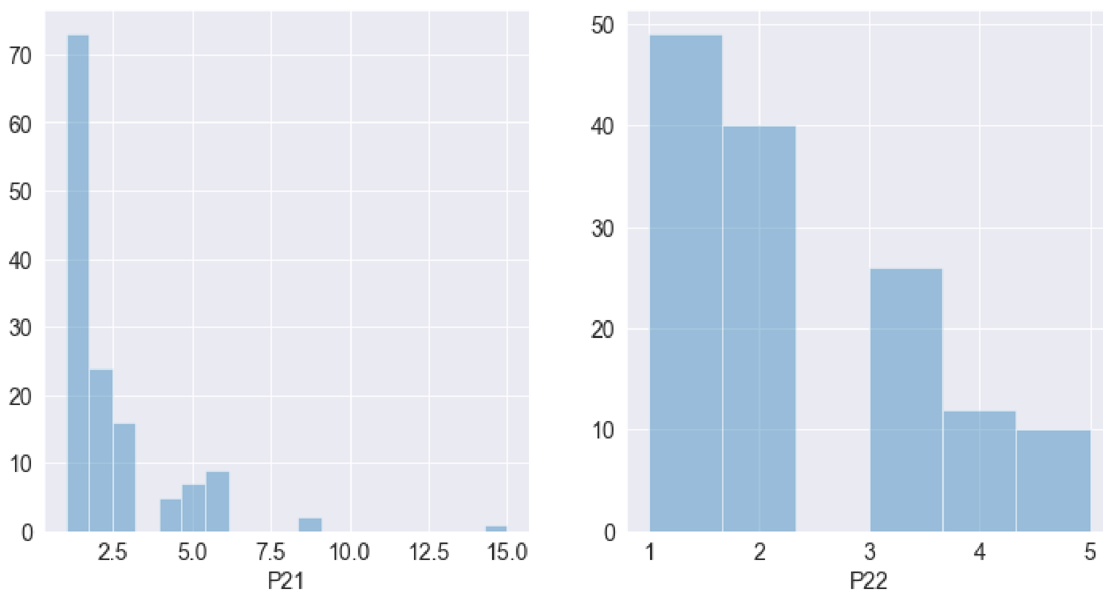
P11

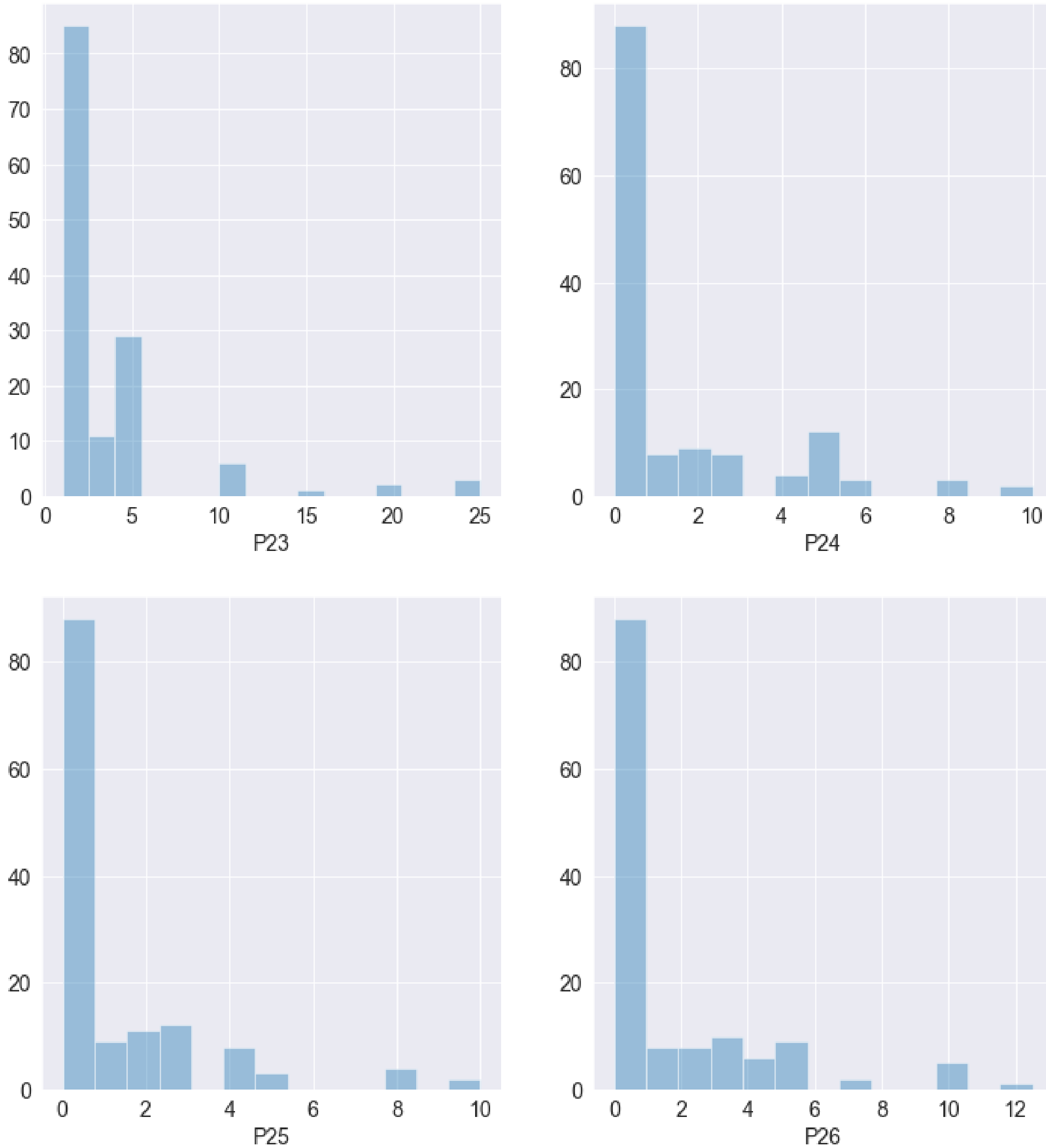


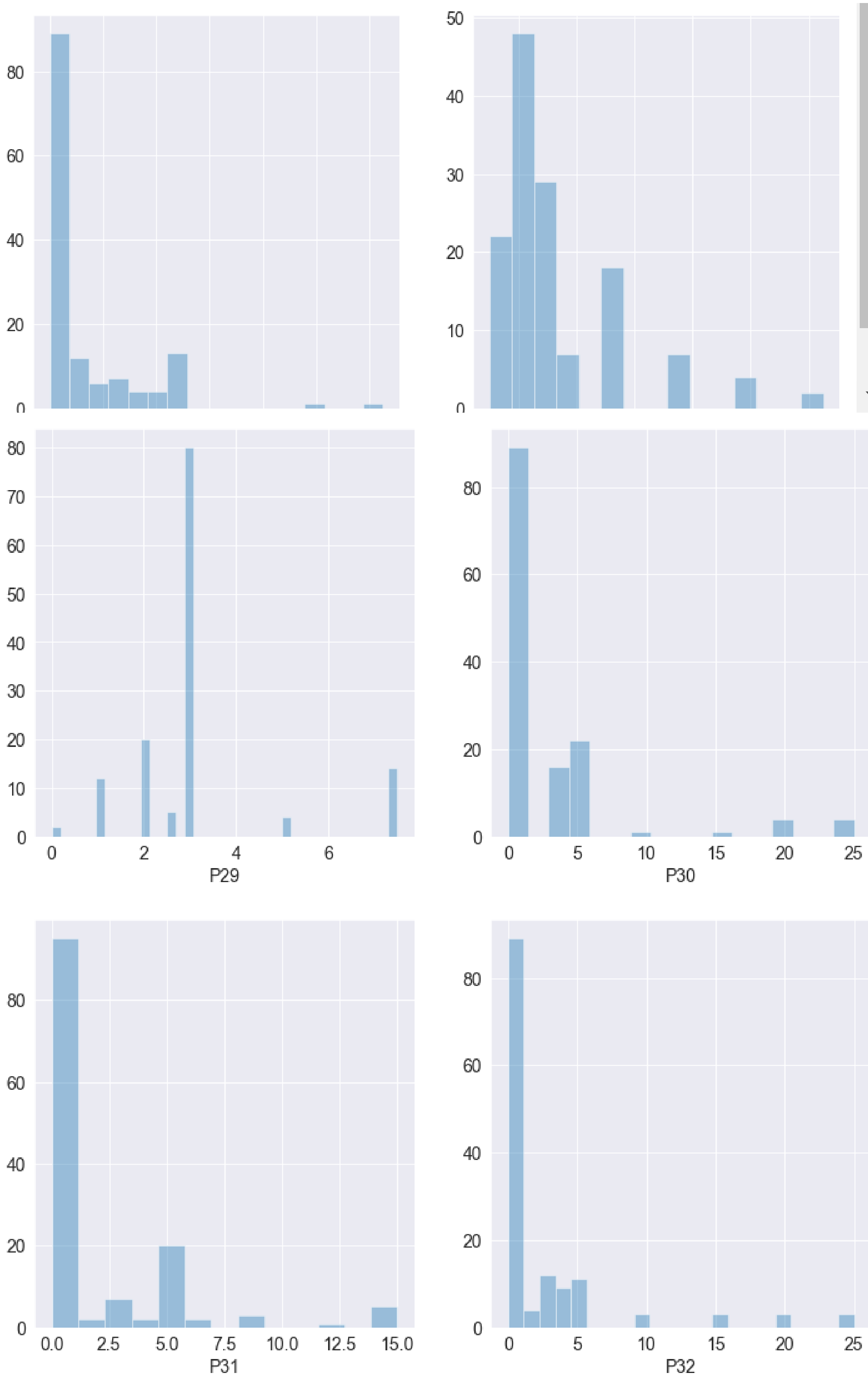
P12

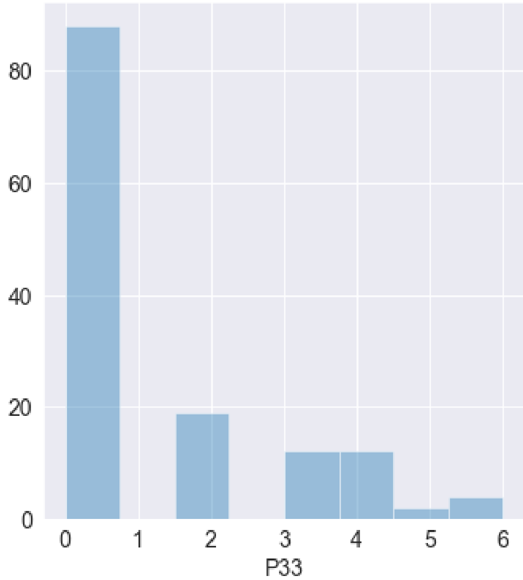

P15
P16

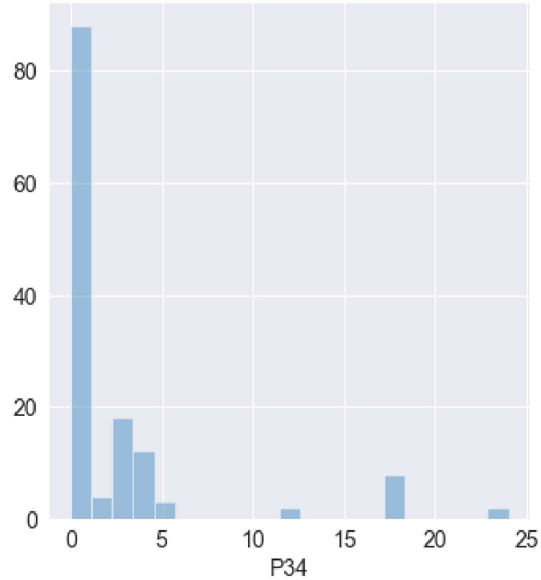

P19
P20

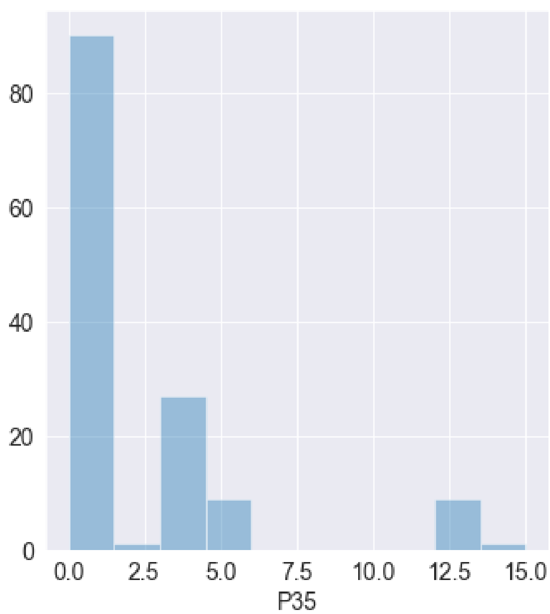

P21
P22

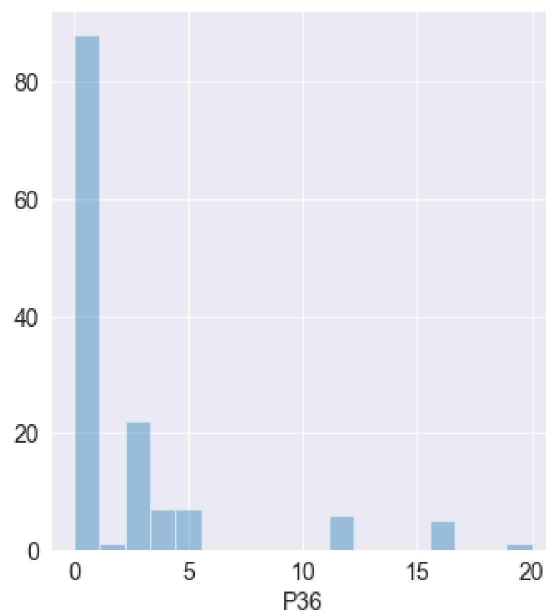

P23
P24
P25
P26

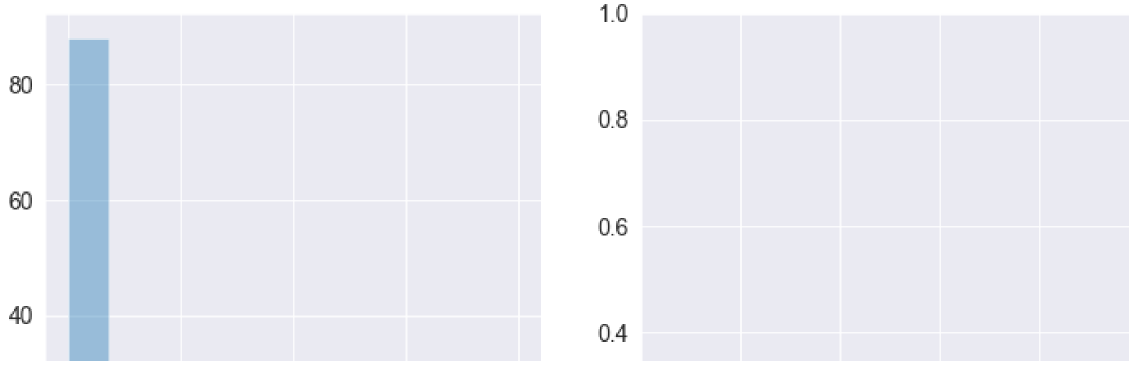

P29
P30
P31
P32


P33


P34


P35


P36



P1...p37 all this these values are not numerical, some p avlues are categorical and they are imputed. Basically multivariate imputation by chained equations (also known as MICE) was used to replace the missing values in some of these features.

Example *from sklearn.experimental import enable_iterative_imputer*
from sklearn.impute import IterativeImputer

imp_train = IterativeImputer(max_iter=30, missing_values=0, sample_posterior=True, min_value=1, random_state=37)
imp_test = IterativeImputer(max_iter=30, missing_values=0, sample_posterior=True, min_value=1, random_state=23)

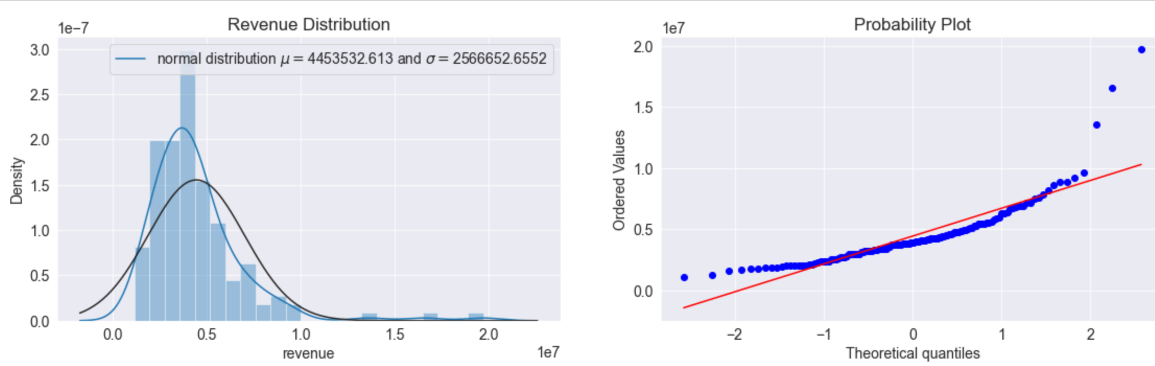
p_data = ['P'+str(i) for i in range(1,38)]
df[p_data] = np.round(imp_train.fit_transform(df[p_data]))
test_df[p_data] = np.round(imp_test.fit_transform(test_df[p_data]))

In [678]:

```
from scipy import stats
from scipy.stats import norm, skew
```

In [679]:

```
(mu,sigma)=norm.fit(data['revenue'])
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(19, 5))
ax1=sns.distplot(data['revenue'],fit=norm, ax=ax1)
ax1.legend([f'normal distribution $\mu=${mu:.3f} and $\sigma=${sigma:.4f}'],
ax1.set_xlabel('revenue')
ax1.set_title('Revenue Distribution')
ax2=stats.probplot(data['revenue'], plot=plt)
```



revenue is bit right skewed but to fit in th linear model we need normal distributed feature hence we use log transform the reveneue into normal distribution model

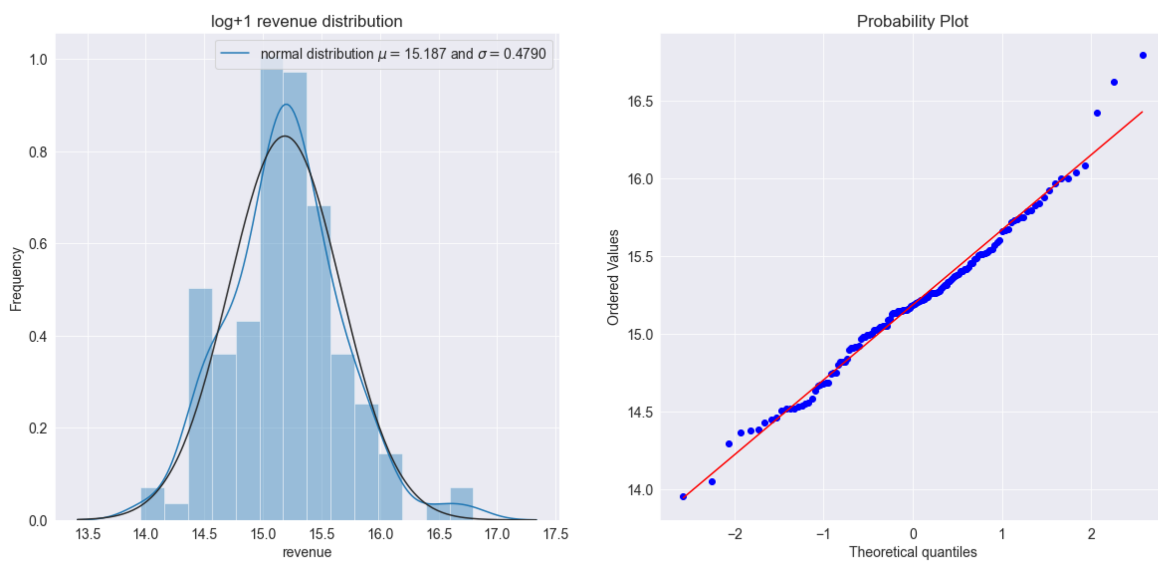
NOTE: remember to use exmp+1 while predicting

$$np.log1p = log(1 + x)$$

In [680]:

```
mu, sigma= norm.fit(np.log1p(data['revenue']))
fig, (ax1,ax2)=plt.subplots(1,2, figsize=(20,9))
ax1=sns.distplot(np.log(data['revenue']),fit=norm,ax=ax1)
ax1.legend([f'normal distribution $\mu=${mu:.3f} and $\sigma=${sigma:.4f}'],
ax1.set_title('log+1 revenue distribution')
ax1.set_xlabel('revenue')
ax1.set_ylabel('Frequency')
ax2=stats.probplot(np.log(data['revenue']), plot=plt)
```

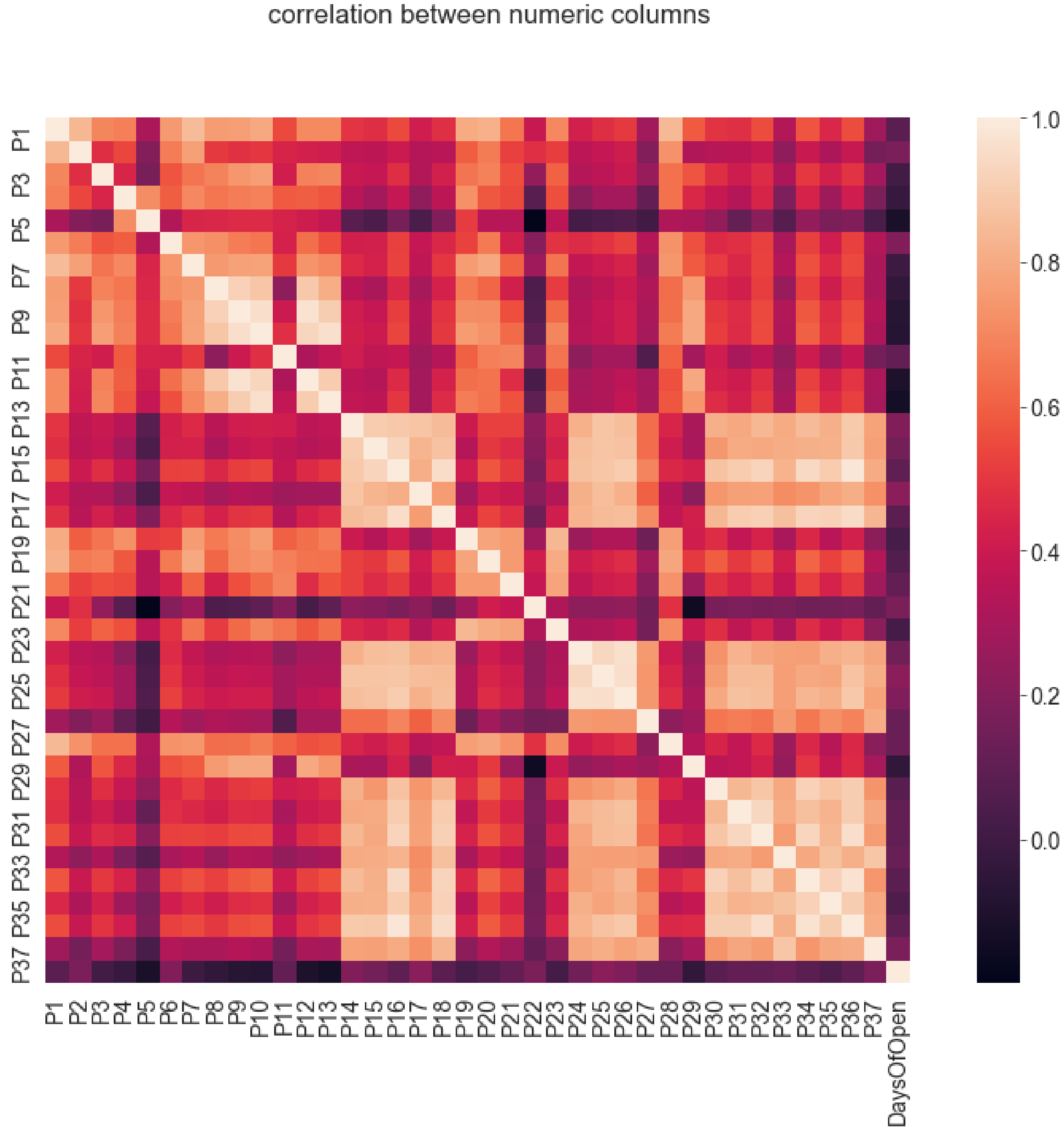
```
c:\users\swaro\appdata\local\programs\python\python37\lib\site-packages\sea
born\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
```



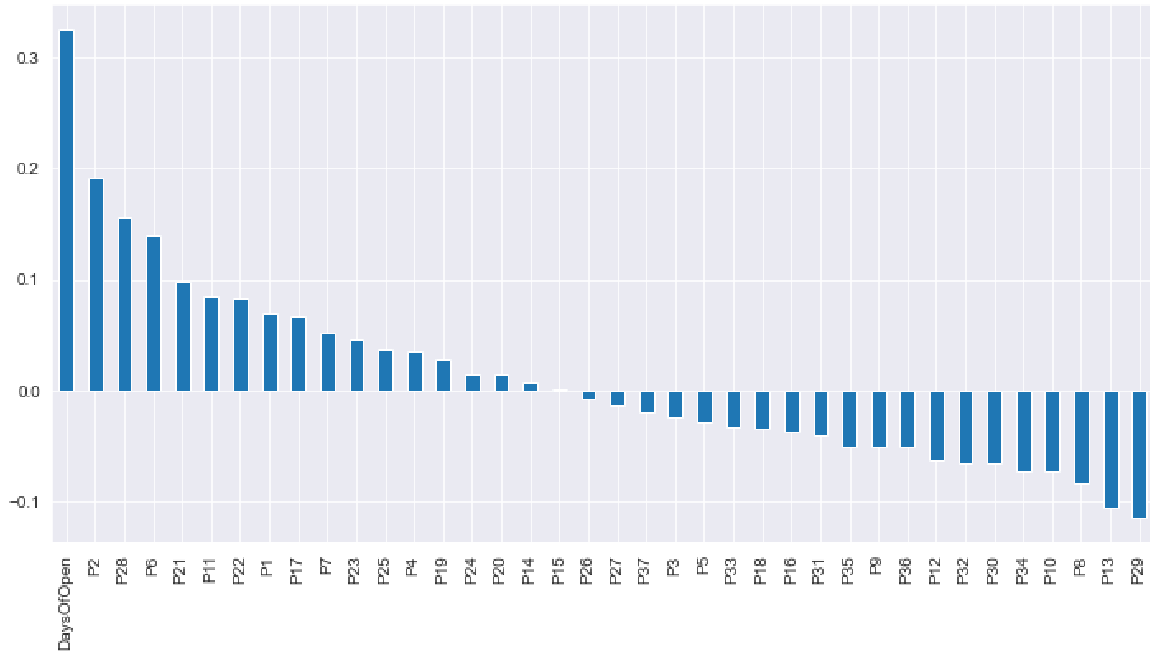
In [681]:

```
plt.figure(figsize=(15,10))
sns.heatmap(data.drop(['revenue','Type','City Group'], axis=1).corr(), square
plt.suptitle('correlation between numeric columns')
```

Out[681]: Text(0.5, 0.98, 'correlation between numeric columns')

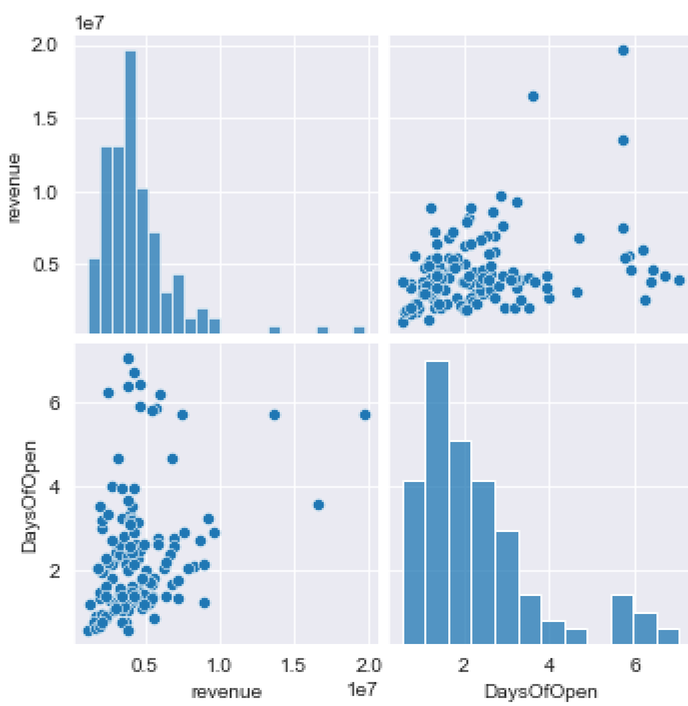


In [682]:
```
plt.figure(figsize=(12,6))
matplotlib.rc('font', size=10)
corr_with_revenue=data.drop(['Type','City Group'], axis=1).corr()['revenue'].
corr_with_revenue.drop('revenue').plot(kind='bar')
plt.show()
```



In [683]:
```
sns.pairplot(data[data.corr()['revenue'].sort_values(ascending=False).index[0
```

Out[683]: <seaborn.axisgrid.PairGrid at 0x1db2fafdac8>

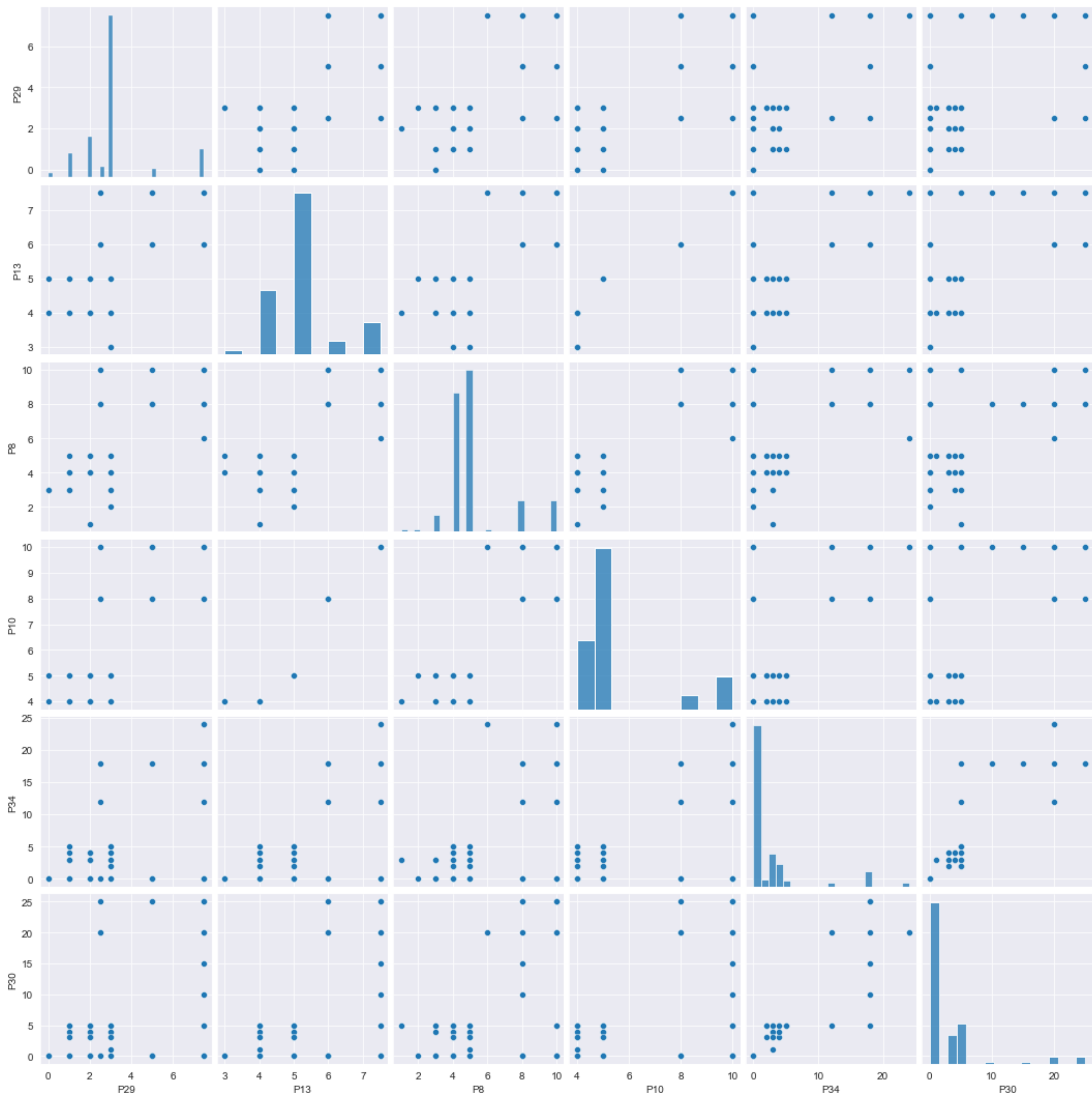


visualizing correlation in pair

In [684]:

```
sns.pairplot(data[data.corr()['revenue'].sort_values(ascending=True).index[0:
```

Out[684]: <seaborn.axisgrid.PairGrid at 0x1db24f5d908>



In [685]:

```
### to handle skewness,in this project it is not used
df1=data.copy()
t_df=test_df.copy()
n_ftr=data.dtypes[data.dtypes!='object'].index
skewed=df1[n_ftr].apply(lambda x: skew(x))
skewed=skewed[skewed>0.5].index
df1[skewed] = np.log1p(df1[skewed])
```

In [686]:
```
data.head(6)
```

Out[686]:

	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
0	Big Cities	IL	4	5.0	4.0	4.0	2	2	5	4	5	5	3	5	5.0	1	2	2
1	Big Cities	FC	4	5.0	4.0	4.0	1	2	5	5	5	5	1	5	5.0	0	0	0
2	Other	IL	2	4.0	2.0	5.0	2	3	5	5	5	5	2	5	5.0	0	0	0
3	Other	IL	6	4.5	6.0	6.0	4	4	10	8	10	10	8	10	7.5	6	4	9
4	Other	IL	3	4.0	3.0	4.0	2	2	5	5	5	5	2	5	5.0	2	1	2
5	Big Cities	FC	6	6.0	4.5	7.5	8	10	10	8	8	8	10	8	6.0	0	0	0

In [687]:
```
df1.head(6)### significantly revenue range has been changed
```

Out[687]:

	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	F
0	Big Cities	IL	1.609438	5.0	4.0	1.609438	1.098612	1.098612	1.791759	1.609438	1.79175
1	Big Cities	FC	1.609438	5.0	4.0	1.609438	0.693147	1.098612	1.791759	1.791759	1.79175
2	Other	IL	1.098612	4.0	2.0	1.791759	1.098612	1.386294	1.791759	1.791759	1.79175
3	Other	IL	1.945910	4.5	6.0	1.945910	1.609438	1.609438	2.397895	2.197225	2.39789
4	Other	IL	1.386294	4.0	3.0	1.609438	1.098612	1.098612	1.791759	1.791759	1.79175
5	Big Cities	FC	1.945910	6.0	4.5	2.140066	2.197225	2.397895	2.397895	2.197225	2.19722

Feature Engineering

left skewed-->> skewness will be negative
right skewed-->> skewness is positive

In [688]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

In [689]:
```
train_iter = IterativeImputer(max_iter=30, missing_values=0, sample_posterior
test_iter = IterativeImputer(max_iter=30, missing_values=0, sample_posterior=

p_value=['P'+str(i) for i in range(1,38)]
df1[p_value]=np.round(train_iter.fit_transform(df1[p_value]))
```

In [690]:
```
df1
```

Out[690]:

	City Group	Type	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
0	Big Cities	IL	2.0	5.0	4.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	1.0
1	Big Cities	FC	2.0	5.0	4.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	2.0	2.0
2	Other	IL	1.0	4.0	2.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	2.0	2.0
3	Other	IL	2.0	4.0	6.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
4	Other	IL	1.0	4.0	3.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
132	Other	FC	1.0	3.0	3.0	2.0	2.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
133	Big Cities	FC	2.0	5.0	4.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0
134	Other	FC	1.0	4.0	4.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	2.0	1.0
135	Big Cities	FC	2.0	5.0	4.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	2.0	2.0
136	Big Cities	FC	2.0	5.0	3.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0

137 rows × 41 columns

In [691]:
```
%%time
test_df[p_value]=np.round(test_iter.fit_transform(test_df[p_value]))
```

```
Wall time: 8min 52s
```

In [692]:
```
data.DaysOfOpen.isnull().sum()
```

Out[692]:
```
0
```

In [693]:
```
cat_cols=data.select_dtypes(include=['object']).columns
cat_cols
```

Out[693]:
```
Index(['City Group', 'Type'], dtype='object')
```

In [694]:

```
## get_dummies
##cat_cols=data.dtypes[data.dtypes=='object'].index--alternatively
cat_cols=data.select_dtypes(include=['object']).columns
data=pd.get_dummies(data, columns=cat_cols, drop_first=False)
```

In [695]:

```
data.head(10)
```

Out[695]:

P29	P30	P31	P32	P33	P34	P35	P36	P37	revenue	DaysOfOpen	City Group_Big Cities	Group_O
3.0	5	3	4	5	5	4	3	4	5653753.0	5.871	1	
3.0	0	0	0	0	0	0	0	0	6923131.0	2.737	1	
3.0	0	0	0	0	0	0	0	0	2055379.0	0.887	0	
7.5	25	12	10	6	18	12	12	6	2675511.0	1.288	0	
3.0	5	1	3	2	3	4	3	3	4316715.0	2.287	0	
5.0	0	0	0	0	0	0	0	0	5017319.0	2.008	1	
3.0	4	5	2	2	3	5	4	4	5166635.0	1.767	1	
2.0	0	0	0	0	0	0	0	0	4491607.0	1.514	1	
3.0	4	5	5	3	4	5	4	5	4952497.0	1.811	0	
2.5	0	0	0	0	0	0	0	0	5444227.0	1.366	0	

In [696]:

```
from sklearn.model_selection import train_test_split
data['revenue']=np.log1p(data['revenue'])
x,y=data.drop('revenue', axis=1), data['revenue']
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.2, random_sta
```

In [697]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
```

ridge and lasso regression (L1 and L2 regularization)

ridge--->> overfitting resolver

$$ridge = loss + \alpha \times (slope)$$

Lasso--->> can perform to resolve overfitting and also useful in feature selection(features with less slope value is eliminated)

$$lasso = loss + \alpha \times |slope|$$

In [698]:
```
params_ridge={
    'alpha':[.01, .1, .5, .7, .9, .95, .99, 1, 5, 10, 20],## alpha values bei
    'fit_intercept':[True,False],
    'normalize':[True, False],
    'solver':['svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
}

## above is used in gridsearchCV as param_grid
```

In [699]:
```
ridge_model=Ridge()
ridge_reg=GridSearchCV(ridge_model,params_ridge,scoring='neg_root_mean_square
ridge_reg.fit(x_train,y_train)
## lets look into the best parameters
print(f'optimal alpha:{ridge_reg.best_params_["alpha"]:.3f}')
print(f'optimal fit_intercept:{ridge_reg.best_params_["fit_intercept"]}')
print(f'optimal normlize:{ridge_reg.best_params_["normalize"]}')
print(f'optimal solver:{ridge_reg.best_params_["solver"]}')
print(f'Best_score:{ridge_reg.best_score_}')
```

```
optimal alpha:1.000
optimal fit_intercept:True
optimal normlize:True
optimal solver:saga
Best_score:-0.4422092367993636
```

In [700]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

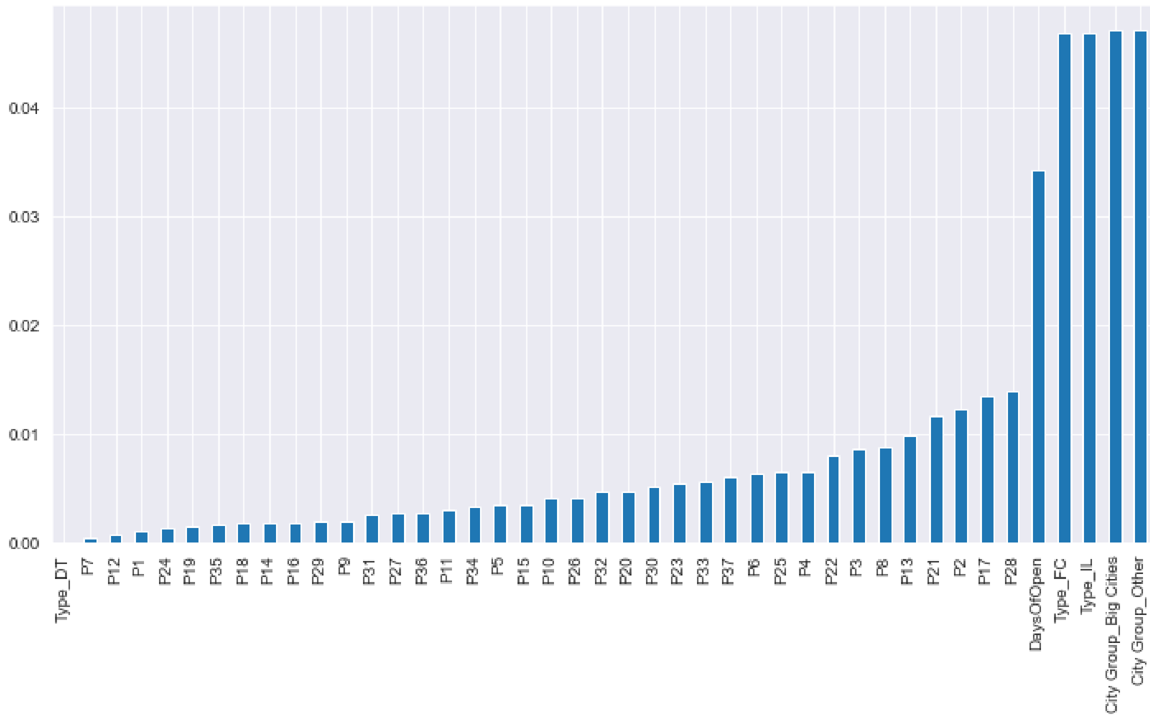
In [701]:
```
### lets fit the tuned parameters
ridge_model=Ridge(alpha=ridge_reg.best_params_["alpha"],fit_intercept=ridge_r
                  normalize=ridge_reg.best_params_["normalize"],solver=ridge_r
ridge_model.fit(x_train,y_train)

train_pred=ridge_model.predict(x_train)
val_pred=ridge_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('validation_score_r2:',r2_score(val_pred,y_test))
train_rmse=np.sqrt(mean_squared_error(train_pred,y_train))
test_rmse=np.sqrt(mean_squared_error(train_pred,y_train))
print(f'train_rmse: {train_rmse:.4f}')
print(f'test_rmse: {test_rmse:.4f}')
```

```
Train r2 score:  -9.53392819446919
validation_score_r2: -20.9442855778621
train_rmse: 0.4105
test_rmse: 0.4105
```

In [702]:

```python
## feature importance
feature_coef=pd.Series(data=np.abs(ridge_model.coef_), index=x_train.columns)
feature_coef.plot(kind='bar', figsize=(12,6))
```

Out[702]: <AxesSubplot:>



Lasso reg

In [703]:

```
params_grid={
    'alpha':[.01, .1, .5, .7, .9, .95, .99, 1, 5, 10, 20],
    'fit_intercept':[True,False],
    'normalize':[True,False]
}
lasso_model=Lasso()
lasso_reg=GridSearchCV(lasso_model,params_grid,scoring='neg_root_mean_squarec
lasso_reg.fit(x_train,y_train)
print(f'optimized alpha:{lasso_reg.best_params_["alpha"]}')
print(f'optimized fit_intercept:{lasso_reg.best_params_["fit_intercept"]}')
print(f'optimized normalize:{lasso_reg.best_params_["normalize"]}')
print(f'best score:{lasso_reg.best_score_}')
```

```
optimized alpha:0.1
optimized fit_intercept:True
optimized normalize:False
best score:-0.4452847908679775
```

In [704]:

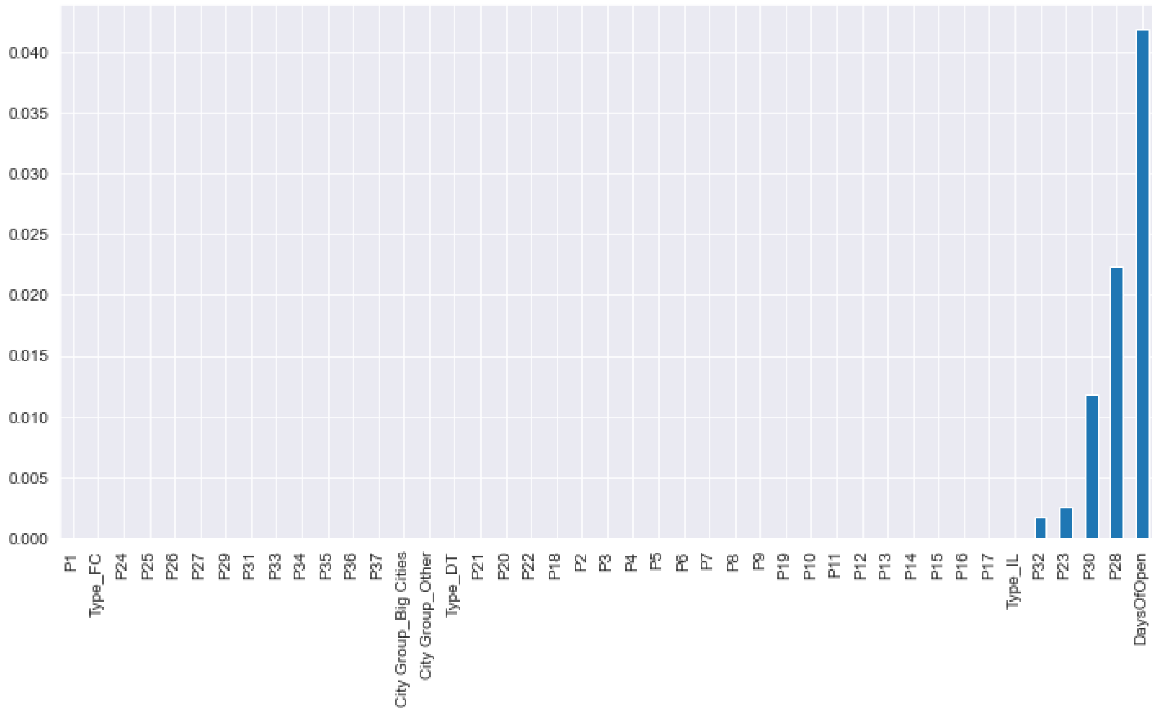
```
## fit tuned params in model
lasso_model=Lasso(alpha=lasso_reg.best_params_['alpha'],fit_intercept=lasso_r
                 normalize = lasso_reg.best_params_["normalize"])
lasso_model.fit(x_train,y_train)
train_pred=lasso_model.predict(x_train)
val_pred=lasso_model.predict(x_test)

print('train_score_r2:',r2_score(train_pred,y_train))
print('train_score_r2:',r2_score(val_pred,y_test))
train_rmse=np.sqrt(mean_squared_error(train_pred,y_train))
val_rmse=np.sqrt(mean_squared_error(val_pred,y_test))
print(f'train_rmse: {train_rmse:.4f}')
print(f'val_rmse: {train_rmse:.4f}')
```

```
train_score_r2: -19.357185893022447
train_score_r2: -40.167961857643796
train_rmse: 0.4288
val_rmse: 0.4288
```

In [705]:

```
## feature importance
feature_coef=pd.Series(data=np.abs(lasso_model.coef_), index=x_train.columns)
feature_coef.plot(kind='bar',figsize=(12,6))
```

Out[705]: <AxesSubplot:>



lets combine both Ridge and lasso reg using--->>> "ElasticNet"

In [706]:

```
from sklearn.linear_model import ElasticNet, ElasticNetCV

## using ElasticNetCV reduce redundancy of  tuning alpha values which we do i
elastic_model=ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1], eps=5e-2,
elastic_model.fit(x_train,y_train)
print(f'alpha: {elastic_model.alpha_:.6f}')
print(f'li_ratio: {elastic_model.l1_ratio_:.3f}')
print(f'number of iteration: {elastic_model.n_iter_}')
```

```
alpha: 0.654552
li_ratio: 0.100
number of iteration: 22
```

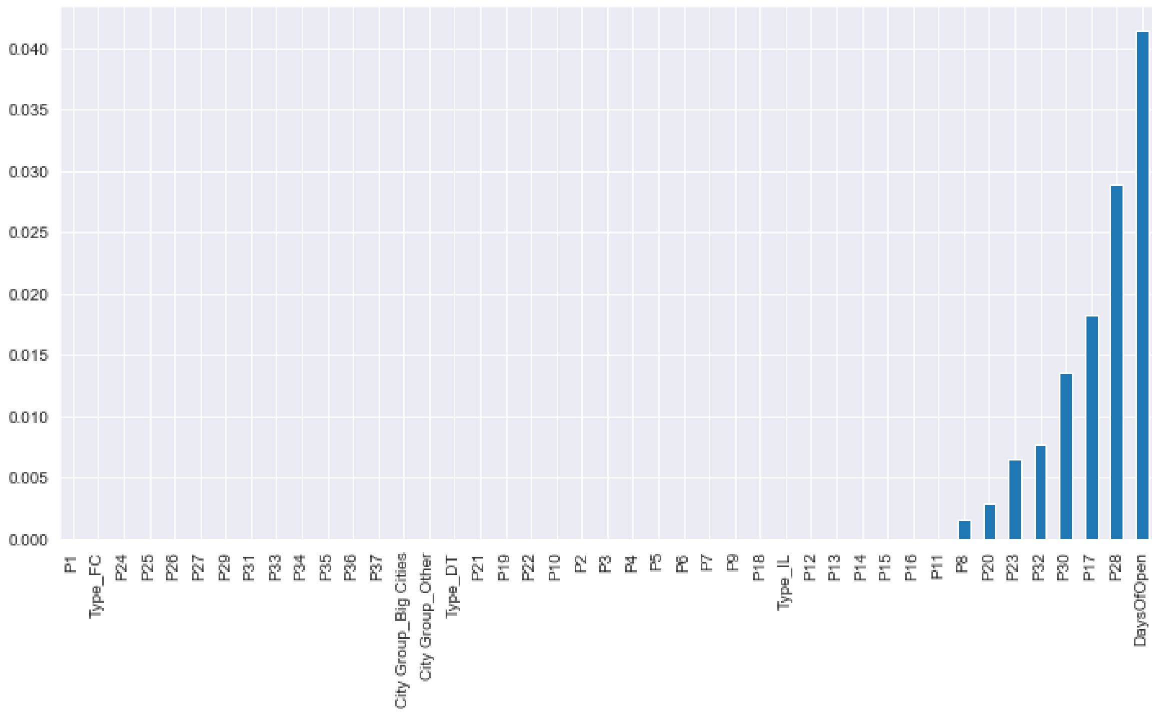
In [707]:

```
### lets predict
train_pred=elastic_model.predict(x_train)
val_pred=elastic_model.predict(x_test)
print(f'train_r2_score: {r2_score(train_pred,y_train)}')
print(f'val_r2_score: {r2_score(val_pred,y_test)}')
train_rmse=np.sqrt(mean_squared_error(train_pred,y_train))
val_rmse=np.sqrt(mean_squared_error(val_pred,y_test))
print(f'train_rms:{train_rmse:.3f}')
print(f'test_rmse:{val_rmse:.3f}')
```

```
train_r2_score: -13.303778284294895
val_r2_score: -25.09364620030457
train_rms:0.421
test_rmse:0.532
```

In [708]:
```
### feature importance
imp_coef=pd.Series(data=np.abs(elastic_model.coef_), index=x_train.columns).s
imp_coef.plot(kind='bar',figsize=(12,6))
```

Out[708]: <AxesSubplot:>



In [709]:
```
?ElasticNetCV
```

KNN (K-NearestNeighburs)

In [710]:
```
from sklearn.neighbors import KNeighborsRegressor

knn_params={
    'n_neighbors':[3,5,7,9,11]
}
knn_model=KNeighborsRegressor()
knn_reg=GridSearchCV(knn_model, knn_params, scoring='neg_root_mean_squared_er
knn_reg.fit(x_train,y_train)
print(f'optimal neighbors:{knn_reg.best_params_["n_neighbors"]}')
print(f'best_score {knn_reg.best_score_}')
```

```
optimal neighbors:9
best_score -0.41107510768078015
```

In [711]:

```
knn_model=KNeighborsRegressor(n_neighbors=9)
knn_model.fit(x_train, y_train)
train_pred=knn_model.predict(x_train)
val_pred=knn_model.predict(x_test)
print(f'train_r2_score:{r2_score(train_pred,y_train)}')
print(f'test_r2_score: {r2_score(val_pred,y_test)}')
train_rmse=np.sqrt(mean_squared_error(train_pred,y_train))
test_rmse=np.sqrt(mean_squared_error(val_pred,y_test))
print(f'train_rmse: {train_rmse:.3f}')
print(f'test_rmse: {test_rmse:.3f}')
```

```
train_r2_score:-3.444336745211438
test_r2_score: -9.506698190128576
train_rmse: 0.393
test_rmse: 0.534
```

Random Forest

In [715]:

```
from sklearn.ensemble import RandomForestRegressor

params_rf = {
    'max_depth': [10, 30, 35, 50, 65, 75, 100],
    'max_features': [.3, .4, .5, .6],
    'min_samples_split': [6, 9, 14],
    'n_estimators': [30, 60, 120, 210],
    'min_samples_leaf':[3,4,5]
}

rf_model= RandomForestRegressor()
rf_reg=GridSearchCV(rf_model, params_rf, cv=10, n_jobs=-1, scoring='neg_root_
rf_reg.fit(x_train, y_train)
print(f'Optimal depth: {rf_reg.best_params_["max_depth"]}')
print(f'Optimal max_features: {rf_reg.best_params_["max_features"]}')
print(f'Optimal min_sample_leaf: {rf_reg.best_params_["min_samples_leaf"]}')
print(f'Optimal min_samples_split: {rf_reg.best_params_["min_samples_split"]}
print(f'Optimal n_estimators: {rf_reg.best_params_["n_estimators"]}')
print(f'Best score: {rf_reg.best_score_}')
```

```
Optimal depth: 10
Optimal max_features: 0.3
Optimal min_sample_leaf: 4
Optimal min_samples_split: 14
Optimal n_estimators: 30
Best score: -0.3958821958607686
```

In [716]:

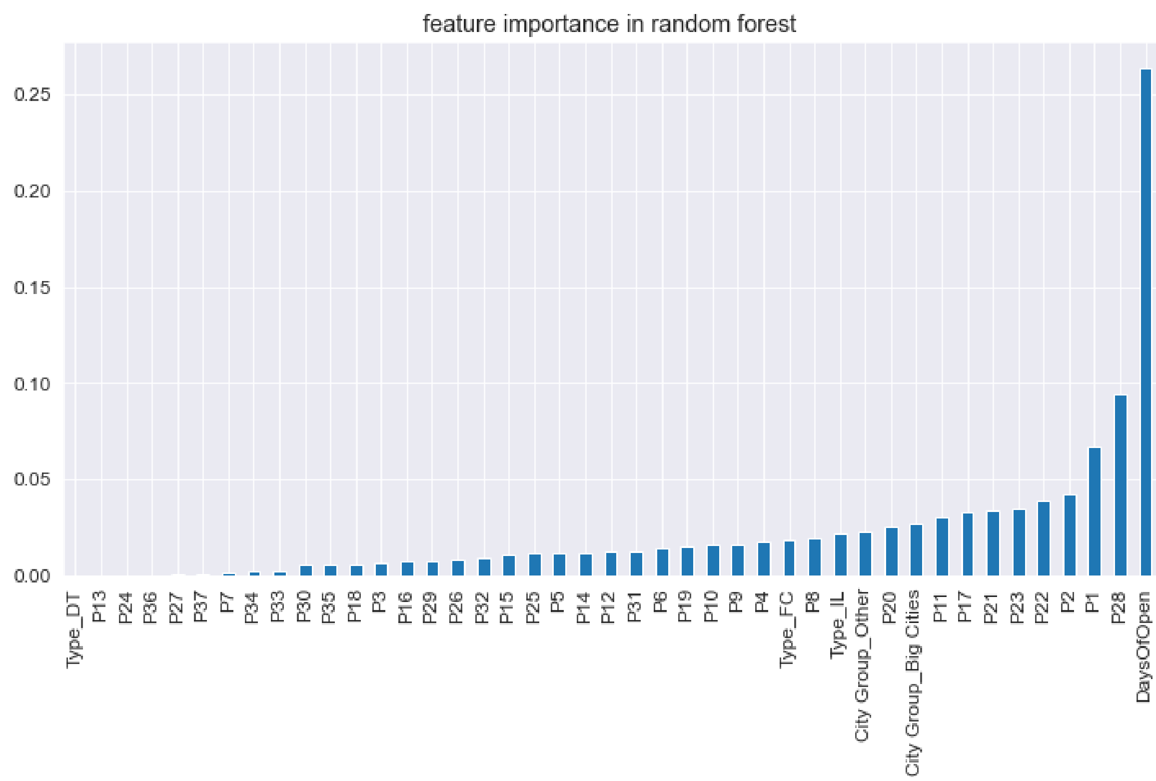
```
rf_model=RandomForestRegressor(max_depth=75,max_features=0.4,min_samples_leaf
                               n_estimators=30, random_state=42,n_jobs=-1,oob
rf_model.fit(x_train,y_train)
train_pred=rf_model.predict(x_train)
val_pred=rf_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

```
Train r2 score:  -1.459306395998862
Test r2 score:  0.27684416302974013
Train RMSE: 0.3141
Test RMSE: 0.4700
```

In [717]:

```
matplotlib.rc('font',size=12)
plt.title('feature importance in random forest')
feature_imp=pd.Series(data=np.abs(rf_model.feature_importances_), index=x_tra
feature_imp.plot(kind='bar',figsize=(12,6),)

n=(feature_imp>0).sum()
print(f'{n} features with reduction of {(1-n/len(feature_imp))*100:2.2f}%')
```

```
39 features with reduction of 9.30%
```



Light GBM

In [718]:

```
import lightgbm as lgbm

lgbm_params={
    'learning_rate':[.01,.1,.5,.8,.9,.95,.99,1],
    'boosting':['gbdt'],
    'metric':['l1'],
    'feature_fraction':[.3,.4,.5,.7,1],
    'num_leaves':[20],
    'min_data':[10],
    'max_depth':[10],
    'n_estimators':[10,30,50,70,100]
}

lgb_model=lgbm.LGBMRegressor()
lgb_reg=GridSearchCV(lgb_model,lgbm_params, cv=10, n_jobs=-1,scoring='neg_roo
lgb_reg.fit(x_train, y_train)
## finding optimum parameters
print(f'optimal Learning Rate: {lgb_reg.best_params_["learning_rate"]}')
print(f'Optimal feature_fraction: {lgb_reg.best_params_["feature_fraction"]}'
print(f'Optimal n_estimators: {lgb_reg.best_params_["n_estimators"]}')
print(f'Best score: {lgb_reg.best_score_}')
```

```
[LightGBM] [Warning] min_data_in_leaf is set with min_child_samples=20, wil
l be overridden by min_data=10. Current value: min_data_in_leaf=10
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignor
ed. Current value: boosting=gbdt
[LightGBM] [Warning] feature_fraction is set=0.4, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.4
optimal Learning Rate: 0.1
Optimal feature_fraction: 0.4
Optimal n_estimators: 50
Best score: -0.39542147006788125
```

In [719]:

```
lgb_model=lgbm.LGBMRegressor(learning_rate=lgb_reg.best_params_['learning_rat
                      feature_fraction=lgb_reg.best_params_["feature_fraction
                      n_estimators=lgb_reg.best_params_["n_estimators"],
                      max_depth=10,
                      min_data=10,
                      num_leaves=20, n_jobs=-1, boosting=['gbdt'],metric='l1'
lgb_model.fit(x_train,y_train)
train_pred=lgb_model.predict(x_train)
val_pred=lgb_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

```
Train r2 score:  0.6457001610093012
Test r2 score:  0.27852844219846784
Train RMSE: 0.1950
Test RMSE: 0.4694
```

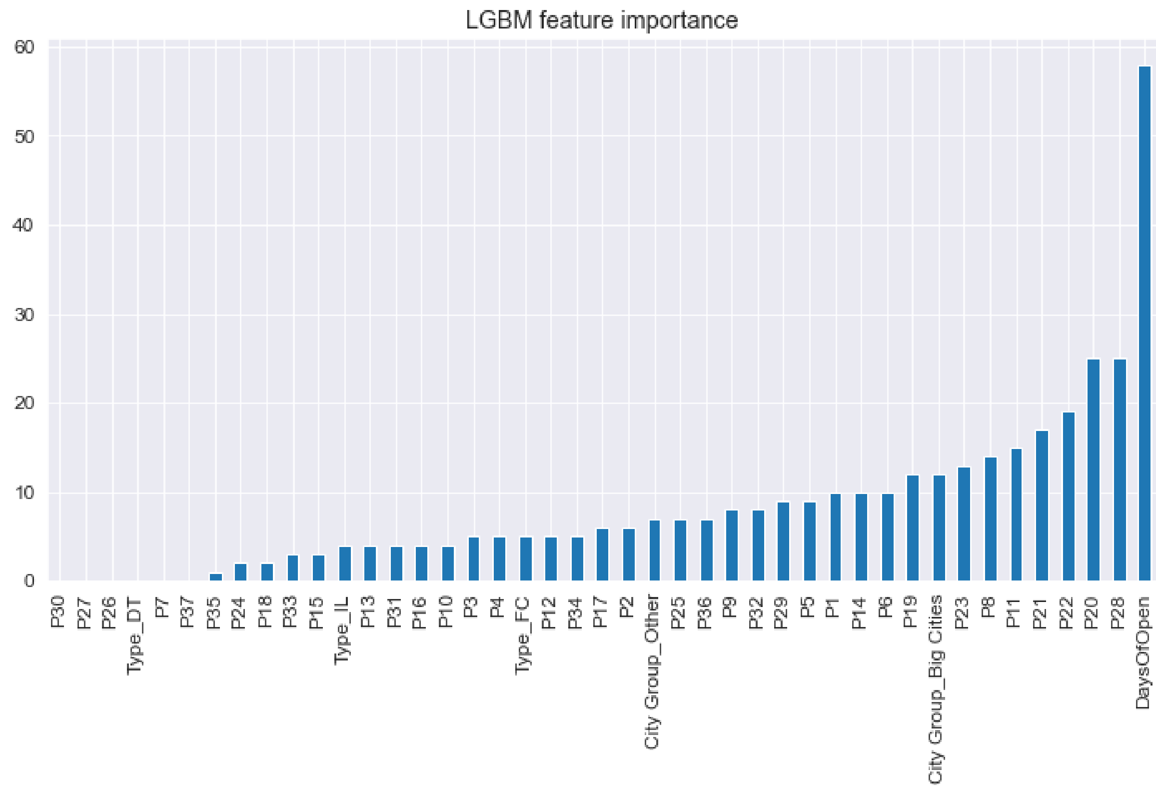
Here it seems like model is overfitted lets see what we can get in xgboost

In [720]:

```
## lgbm regressor feature importance
lgb_features=pd.Series(data=np.abs(lgb_model.feature_importances_), index=x_t
n_feat=(lgb_features>0).sum()
print(f'{n_feat} features with reduction of {(1-n_feat/len(lgb_features))*100
plt.title('LGBM feature importance')
lgb_features.sort_values().plot(kind='bar', figsize=(12,6))
```

```
37 features with reduction of 13.95%
```

Out[720]: <AxesSubplot:title={'center':'LGBM feature importance'}>



GBM using XGboost

In [721]:

```
from xgboost import XGBRegressor
```

In [722]:

```
params_xgb = {
    'learning_rate': [.1, .5, .7, .9, .95, .99, 1],
    'colsample_bytree': [.3, .4, .5, .6],
    'max_depth': [4],
    'alpha': [3],
    'subsample': [.5],
    'n_estimators': [30, 70, 100, 150]
}

xgb_model = XGBRegressor()
xgb_regressor = GridSearchCV(xgb_model, params_xgb, scoring='neg_root_mean_sc
xgb_regressor.fit(x_train, y_train)
print(f'Optimal learning_rate: {xgb_regressor.best_params_["learning_rate"]}'
print(f'Optimal colsample_bytree: {xgb_regressor.best_params_["colsample_bytr
print(f'Optimal n_estimators: {xgb_regressor.best_params_["n_estimators"]}')
print(f'Best score: {xgb_regressor.best_score_}')
```

```
Optimal learning_rate: 0.9
Optimal colsample_bytree: 0.3
Optimal n_estimators: 30
Best score: -0.40213172143940856
```

In [723]:

```

xgb_model = XGBRegressor(learning_rate=xgb_regressor.best_params_["learning_r
                         colsample_bytree=xgb_regressor.best_params_["colsamp
                         max_depth=4, alpha=3, subsample=.5,
                         n_estimators=xgb_regressor.best_params_["n_estimator
xgb_model.fit(x_train, y_train)
train_pred = xgb_model.predict(x_train)
val_pred = xgb_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

```
Train r2 score:  -0.8849982006504784
Test r2 score:  0.15411538877217035
Train RMSE: 0.3361
Test RMSE: 0.5083
```

In [724]:

```
xgb_model = XGBRegressor(learning_rate=0.95,
                         colsample_bytree=0.5,
                         max_depth=4, alpha=3, subsample=.5,
                         n_estimators=100, n_jobs=-1, random_state=43)
xgb_model.fit(x_train, y_train, eval_metric='rmse',verbose=True,
              early_stopping_rounds=4,eval_set=[(x_test,y_test)])
train_pred = xgb_model.predict(x_train)
val_pred = xgb_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

```
[0]	validation_0-rmse:1.11313
[1]	validation_0-rmse:0.55352
[2]	validation_0-rmse:0.54211
[3]	validation_0-rmse:0.52358
[4]	validation_0-rmse:0.52358
[5]	validation_0-rmse:0.52309
[6]	validation_0-rmse:0.51741
[7]	validation_0-rmse:0.51010
[8]	validation_0-rmse:0.51237
[9]	validation_0-rmse:0.51101
[10]	validation_0-rmse:0.52993
Train r2 score:  -2.4779563722592313
Test r2 score:  0.14813384823827946
Train RMSE: 0.3686
Test RMSE: 0.5101
```

using early stop in xgboost while fitting data into the model, this avoids model overfitting

1. It works by monitoring the performance of the model that is being trained on a separate test dataset and stopping the training procedure once the performance on the test dataset has not improved after a fixed number of training iterations.
2. It avoids overfitting by attempting to automatically select the inflection point where performance on the test dataset starts to decrease while performance on the training dataset continues to improve as the model starts to overfit.

In [726]:

```
xgb_model.fit(x_train,y_train,early_stopping_rounds=4, eval_set=[(x_test,y_te
              eval_metric='rmse',verbose=True)
train_pred = xgb_model.predict(x_train)
val_pred = xgb_model.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

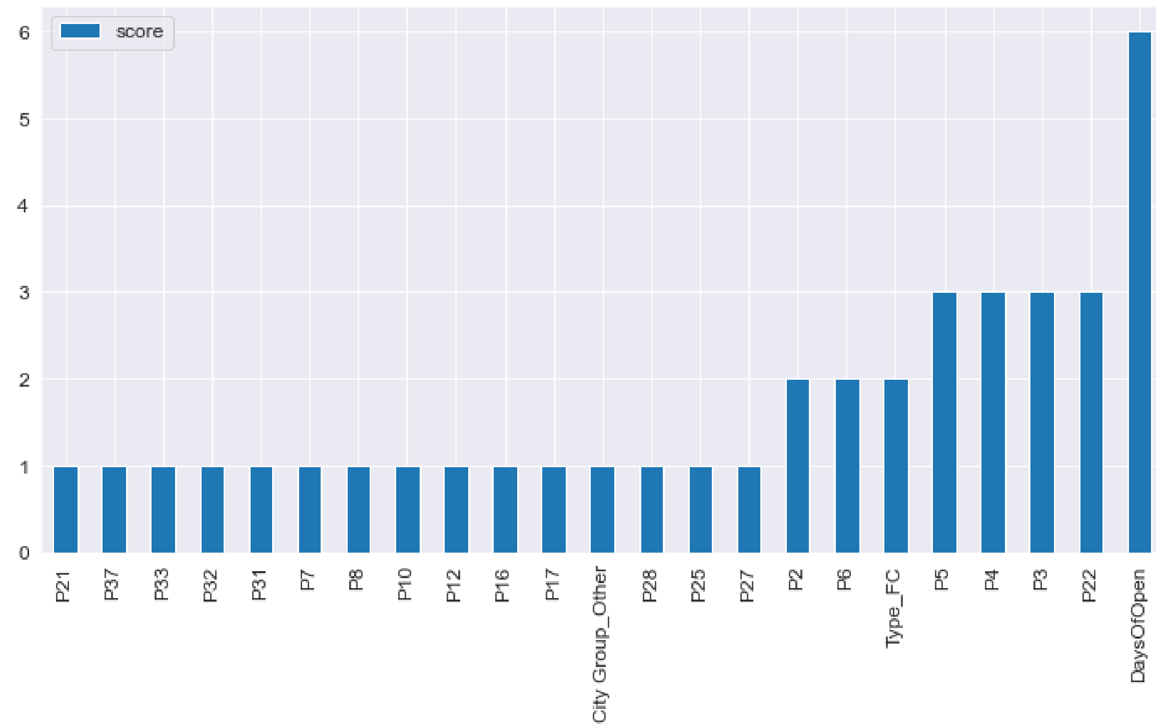
```
[0]	validation_0-rmse:1.11313
[1]	validation_0-rmse:0.55352
[2]	validation_0-rmse:0.54211
[3]	validation_0-rmse:0.52358
[4]	validation_0-rmse:0.52358
[5]	validation_0-rmse:0.52309
[6]	validation_0-rmse:0.51741
[7]	validation_0-rmse:0.51010
[8]	validation_0-rmse:0.51237
[9]	validation_0-rmse:0.51101
[10]	validation_0-rmse:0.52993
[11]	validation_0-rmse:0.52993
Train r2 score:  -2.4779563722592313
Test r2 score:  0.14813384823827946
Train RMSE: 0.3686
Test RMSE: 0.5101
```

early stop did not yeild much, still model is worse. r2_score is negative this means curve is not following trend whatsoever

In [727]:

```
xgb_feat=xgb_model.get_booster().get_fscore()
keys=list(xgb_feat.keys())
values=list(xgb_feat.values())
## Be careful while using python object here I used main data frame name wher
data=pd.DataFrame(data=values, index=keys, columns=['score']).sort_values(by=
data.plot(kind='bar', figsize=(12,6))
```

Out[727]: <AxesSubplot:>



xgboot is not working for us

Regressor ensembling

By for whatever model is used RFRegressor ensembling worked fine for us

In [728]:

```
rf_model_en = RandomForestRegressor(max_depth=200, max_features=0.4, min_samp
                                    min_samples_split=6, n_estimators=30, n_jobs
rf_model_en.fit(x_train, y_train)
train_pred = rf_model_en.predict(x_train)
val_pred = rf_model_en.predict(x_test)
print('Train r2 score: ', r2_score(train_pred, y_train))
print('Test r2 score: ', r2_score(y_test, val_pred))
train_rmse = np.sqrt(mean_squared_error(train_pred, y_train))
test_rmse = np.sqrt(mean_squared_error(y_test, val_pred))
print(f'Train RMSE: {train_rmse:.4f}')
print(f'Test RMSE: {test_rmse:.4f}')
```

```
Train r2 score:  -0.056758366720200029
Test r2 score:  0.26801077978249066
Train RMSE: 0.2604
Test RMSE: 0.4728
```

In [729]:

```
from numpy import mean, std
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.datasets import make_regression
from sklearn.ensemble import StackingRegressor
```

In [730]:

```
## getting ensembled model
def get_stacking():
    base_models=list()
    base_models.append(('ridge',ridge_model))
    base_models.append(('lasso',lasso_model))
    base_models.append(('random_forest',rf_model))
    ## defining Linearregression
    learner=LinearRegression()
    ## final ensemble model
    ## estimators--->>> base models which gave good accuracy
    ## final_estimator---->>> regressor used that will be cobined with base m
    ###passthrough --->>> set false beacuse just the final_estimator results
    model=StackingRegressor(estimators=base_models,final_estimator=learner, c
    return model

## models are used to evaluate each individual models performance
def get_model():
    models=dict()
    models['ridge']=ridge_model
    models['lasso']=lasso_model
    models['random_f']=rf_model
    models['stacking']=get_stacking()
    return models

##evaluating using cross validation score
def evaluate(model, x,y):
    cv=RepeatedKFold(n_splits=10 ,n_repeats=5 ,random_state=43)
    scores=cross_val_score(model,x,y,scoring='neg_mean_absolute_error',cv=cv,
    return scores

models=get_model()
result,names=list(),list()
for name, model in models.items():
    score=evaluate(model, x_train,y_train)
    result.append(score)
    names.append(name)
    print(f'{names} {mean(score)} {std(score)}')
```

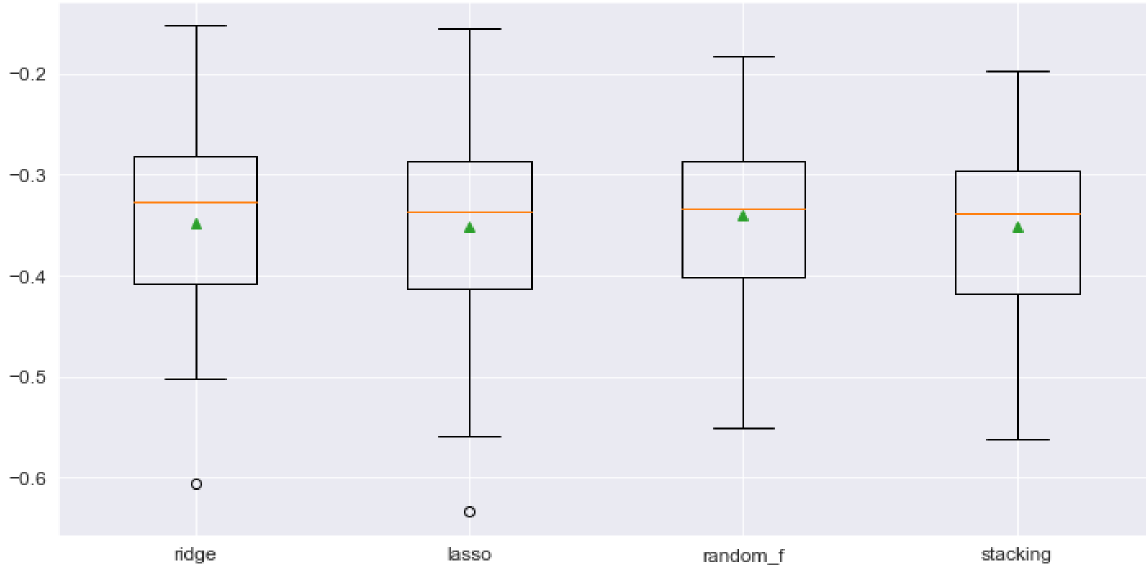
```
['ridge'] -0.34875699298135737 0.08764053152415771
['ridge', 'lasso'] -0.3512285235818889 0.09253584917552803
['ridge', 'lasso', 'random_f'] -0.33944773394721267 0.0828251774349566
['ridge', 'lasso', 'random_f', 'stacking'] -0.35159376177787366 0.083273874
71963954
```

In [731]:

```
name=['ridge', 'lasso', 'random_f', 'stacking']
```

In [732]:

```
## plot frequency digram to compare
plt.figure(figsize=(12,6))
plt.boxplot(result, labels=name,vert=True, showmeans=True)
plt.show()
```



In [747]:
```
### storing base models
base_models=list()
base_models.append(('ridge', ridge_model))
base_models.append(('lasso',ridge_model))
base_models.append(('random forest', rf_model))

## final learner/ meta LinearRegression
learner=LinearRegression()
stck_model=StackingRegressor(estimators=base_models, final_estimator=learner,
##fitting whole data irrespective of train and val
stck_model.fit(x,y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    2.2s finished
```

Out[747]:
```
StackingRegressor(cv=10,
                  estimators=[('ridge',
                               Ridge(alpha=1, normalize=True, solver='sag
a')),
                              ('lasso',
                               Ridge(alpha=1, normalize=True, solver='sag
a')),
                              ('random forest',
                               RandomForestRegressor(max_depth=75,
                                                     max_features=0.4,
                                                     min_samples_leaf=5,
                                                     min_samples_split=9,
                                                     n_estimators=30, n_job
s=-1,
                                                     oob_score=True,
                                                     random_state=42))],
                  final_estimator=LinearRegression(), verbose=True)
```

In [748]:
```
score=evaluate(stck_model,x,y)
```

In [750]:
```
mean(score), std(score)
```

Out[750]:
```
(-0.3548553427596707, 0.06741499622395684)
```

By far whatever model we used RF came out best in terms of R2_score and even in balancing biase and varience

In [ ]:

In [758]:

```
### lets try only rf models and ensemble them
### below is the optimized model got form stacking
stk_rf_model=RandomForestRegressor(max_depth=75,max_features=0.4,
                                                     min_samples_leaf=5,
                                                     min_samples_split=9,
                                                     n_estimators=50, n_jobs=
                                                     oob_score=True,
                                                     random_state=42)


## base models
base_model_rf=list()
base_model_rf.append(('rf1',rf_model))
base_model_rf.append(('rf2', rf_model))
base_model_rf.append(('rf3',stk_rf_model))

## final stimator/ LR
learner=LinearRegression()
## stacking
stk_model=StackingRegressor(estimators=base_model_rf, final_estimator=learner
stk_model.fit(x,y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    2.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    2.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    2.9s finished
```

Out[758]:

```
StackingRegressor(cv=12,
                  estimators=[('rf1',
                               RandomForestRegressor(max_depth=75,
                                                     max_features=0.4,
                                                     min_samples_leaf=5,
                                                     min_samples_split=9,
                                                     n_estimators=30, n_job
s=-1,
                                                     oob_score=True,
                                                     random_state=42)),
                              ('rf2',
                               RandomForestRegressor(max_depth=75,
                                                     max_features=0.4,
                                                     min_samples_leaf=5,
                                                     min_samples_split=9,
                                                     n_estimators=30, n_job
s=-1,
                                                     oob_score=True,
                                                     random_state=42)),
                              ('rf3',
                               RandomForestRegressor(max_depth=75,
                                                     max_features=0.4,
                                                     min_samples_leaf=5,
                                                     min_samples_split=9,
                                                     n_estimators=50, n_job
```

```
s=-1,
                                                    oob_score=True,
                                                    random_state=42))],
                  final_estimator=LinearRegression(), verbose=True)
```

In [829]:

```
score_rf=evaluate(stk_model,x,y)
print(f"mean:{mean(score_rf)} std: {std(score_rf)}")
```

```
mean:-0.3494324659723111 std: 0.06731955522537553
```

lets process test data and do some prediction on it

In [733]:

```
test_df.Type.value_counts()## we were already replace MB with DT
```

Out[733]:

```
FC    57019
IL    40447
DT     2534
Name: Type, dtype: int64
```

In [738]:

```
import datetime
test_df['Open Date']=pd.to_datetime(test_df['Open Date'])
base_date=datetime.datetime(2015,8,24)
##to find numbers of days it have had been open, since its inception
test_df['DaysOfOpen']=(base_date-test_df['Open Date']).dt.days/1000 ### 1000
```

In [832]:

```
### encoding OHE
categorical=test_df.select_dtypes(include=['object']).columns.tolist()
test_df=pd.get_dummies(test_df, columns=categorical, drop_first=False)
```

In [834]:

```
test_df.drop('Open Date', axis=1, inplace=True)
```

predicting revenue by using test data and saving prediction as csv

In [839]:
```
## creating submission df and equating Id in submission
submission_df=pd.DataFrame(columns=['Id', 'pred_revenue'])
submission_df['Id']=test_df['Id']

#### ridge model

ridge_pred=ridge_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(ridge_pred)
submission_df.to_csv('ridge_predict.csv', index=False)

### lasso model

lasso_pred=lasso_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(lasso_pred)
submission_df.to_csv('lasso_predict.csv', index=False)

### random forest

rf_pred=rf_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(rf_pred)
submission_df.to_csv('rf_predict.csv', index=False)

## LightGBM

lgb_pred=lgb_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(lgb_pred)
submission_df.to_csv('lightGBM_predict.csv', index=False)

## Xgboost GBM

xgb_pred=xgb_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(xgb_pred)
submission_df.to_csv('xgboost_predict.csv', index=False)

## stacked model/ ensemble model

stack_pred=stck_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(stack_pred)
submission_df.to_csv('ensemble_stack_predict.csv', index=False)

#### stacked model (only random forest is ued for stacking)

rf_ensemble_pre=stk_model.predict(test_df.drop('Id', axis=1))
submission_df['pred_revenue']=np.expm1(rf_ensemble_pre)
submission_df.to_csv('rf_ensemble_predict.csv', index=False)
```

Conlusion

1. Here skew is not performed on numerical data , this can be tried for result
2. scaling all numerical to same range and each model performace can be checked on scaled data
3. LightGBM model is overfitting hence we can use it in stacking model by performing intense hyperparameters tuning

4. xgboost does' not yeild much and it can be replaced with other model.
5. All models can be stored in "joblib" and can be reused but file will be too huge as we have too many models in here