Project Linear Regression

In this notebook a linear regression algortihm will be used to predict the units ordered for a specific medication type

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Dataset ¶

Below you can see the dataset in it's current state

In [2]:

```
df = pd.read_csv("pharmacy_with_pop.csv")
df.head()
```

Out[2]:

	Delivery Date	Pharmacy No	Pharmacy Post	YOB	Gender	CNK	Product Name	ATC
0	2017-01-01	7341765	21	1922	1	5520523	WACHTHONORARIUM	
1	2017-01-01	7341765	21	1925	1	1799931	ZALDIAR 37,5 MG/325 MG FILMOMH TABL 20	N02AJ13
2	2017-01-01	8272695	16	1932	2	1719400	VASEXTEN CAPS BLIST 28 X 10 MG	C08CA12
3	2017-01-01	8272695	16	1933	2	5520523	WACHTHONORARIUM	
4	2017-01-01	9111423	10	1931	1	1750132	AACIDEXAM 5MG/ML OPL INJ FL INJ 1 X 1ML	H02AB02

Selecting properties

Below you can select the prefered medication type and frequency (Y, M, W, D)

In [3]:

```
MedicationType = 'Psychoanaleptics'
Frequency= 'M'
```

After selecting a medication type, it will be filtered from the dataset

In [4]:

```
is_Med = df['Medication_Type'] == MedicationType
df2 = df[is_Med]
df2.head()
```

Out[4]:

	Delivery Date	Pharmacy No	Pharmacy Post	YOB	Gender	CNK	Product Name	ATC	l
40	2017-01-01	7641438	40	1969	2	3183092	CYMBALTA 60 MG MAAGSAPRESIST. CAPS 98 X 60 MG	N06AX21	
838	2017-01-01	7084071	86	1899	0	126987	REDOMEX DIFFUCAPS CAPS 40 X 25 MG	N06AA09	
839	2017-01-01	7056201	89	1899	0	127019	REDOMEX DIFFUCAPS CAPS 40 X 50 MG	N06AA09	
978	2017-01-01	7067208	20	1899	0	1390343	SERLAIN 50 MG COMP PELL 30 X 50 MG	N06AB06	
1269	2017-01-01	7122399	30	1899	0	1625672	FLUOXETINE EG CAPS 56 X 20 MG	N06AB03	

In [5]:

```
df2['Delivery Date'] = df2['Delivery Date'].astype('datetime64[ns]')
df2.head()
```

```
<ipython-input-5-40db6a7376c0>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df2['Delivery Date'] = df2['Delivery Date'].astype('datetime64[ns]')
```

Out[5]:

	Delivery Date	Pharmacy No	Pharmacy Post	YOB	Gender	CNK	Product Name	ATC	L
40	2017-01-01	7641438	40	1969	2	3183092	CYMBALTA 60 MG MAAGSAPRESIST. CAPS 98 X 60 MG	N06AX21	
838	2017-01-01	7084071	86	1899	0	126987	REDOMEX DIFFUCAPS CAPS 40 X 25 MG	N06AA09	
839	2017-01-01	7056201	89	1899	0	127019	REDOMEX DIFFUCAPS CAPS 40 X 50 MG	N06AA09	
978	2017-01-01	7067208	20	1899	0	1390343	SERLAIN 50 MG COMP PELL 30 X 50 MG	N06AB06	
1269	2017-01-01	7122399	30	1899	0	1625672	FLUOXETINE EG CAPS 56 X 20 MG	N06AB03	

Create a new dataframe

Now it has to create a new dataframe to sum the units on the selected frequency

In [6]:

```
res = df2.set_index('Delivery Date').groupby([pd.Grouper(freq=Frequency), 'Total Popula
tion', 'Age Group'])['Units'].sum().reset_index()

print(res)
```

```
    Delivery Date  Total Population Age Group   Units
0      2017-01-31           21822.0     95-99    5188
1      2017-01-31           94399.0     90-94   31220
2      2017-01-31          208401.0     85-89   91037
3      2017-01-31          322326.0     80-84  139935
4      2017-01-31          379764.0     75-79  152454
..            ...               ...       ...     ...
701    2019-12-31          751517.0     40-44  131912
702    2019-12-31          754578.0     35-39  107362
703    2019-12-31          772635.0     45-49  178354
704    2019-12-31          799615.0     55-59  256272
705    2019-12-31          802276.0     50-54  237066

[706 rows x 4 columns]
```

In [7]:

```
res['Month'] = 0
res['Month'] = pd.DatetimeIndex(res['Delivery Date']).month
res.head()
```

Out[7]:

	Delivery Date	Total Population	Age Group	Units	Month
0	2017-01-31	21822.0	95-99	5188	1
1	2017-01-31	94399.0	90-94	31220	1
2	2017-01-31	208401.0	85-89	91037	1
3	2017-01-31	322326.0	80-84	139935	1
4	2017-01-31	379764.0	75-79	152454	1

Converting the age groups

To be able to use the age groups it has to be converted to numerical data, so we do that by changing '0-4' to a 1, '5-9' to a 2, etc.

In [8]:

```
res['Age'] = 0
res['Age'] = res['Age Group'].map( {'0-4': 1, '5-9': 2, '10-14': 3, '15-19': 4, '20-24'
: 5, '25-29': 6, '30-34': 7, '35-39': 8, '40-44': 9, '45-49': 10, '50-54': 11, '55-59':
12, '60-64': 13, '65-69': 14, '70-74': 15, '75-79': 16, '80-84': 17, '85-89': 18, '90-9
4': 19, '95-99': 20, '100+': 20, } ).astype(int)
res.head(5)
```

Out[8]:

	Delivery Date	Total Population	Age Group	Units	Month	Age
0	2017-01-31	21822.0	95-99	5188	1	20
1	2017-01-31	94399.0	90-94	31220	1	19
2	2017-01-31	208401.0	85-89	91037	1	18
3	2017-01-31	322326.0	80-84	139935	1	17
4	2017-01-31	379764.0	75-79	152454	1	16

In [ ]:

```

```

In [9]:

```
res['Total Population'] = res['Total Population'].astype(float)
res.dtypes
res = res.reset_index()
res.head()
```

Out[9]:

	index	Delivery Date	Total Population	Age Group	Units	Month	Age
0	0	2017-01-31	21822.0	95-99	5188	1	20
1	1	2017-01-31	94399.0	90-94	31220	1	19
2	2	2017-01-31	208401.0	85-89	91037	1	18
3	3	2017-01-31	322326.0	80-84	139935	1	17
4	4	2017-01-31	379764.0	75-79	152454	1	16

Swapping column placement

To normalize the data it is easier when the columns that are to be normalized are al next to each other so Age Group and Age are swapped

In [10]:

```
cols = list(res.columns)
a, b = cols.index('Age Group'), cols.index('Age')
cols[b], cols[a] = cols[a], cols[b]
res = res[cols]
```

Normalizing the data

To make sure that the data shares a common scale we use the minmaxscaler to normalize the data

In [11]:

```
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
res[[i for i in list(res.columns)[2:5]]] = minmax.fit_transform(res[[i for i in list(re
s.columns)[2:5]]])
print(res)
```

```
     index Delivery Date  Total Population       Age     Units  Month  \
0        0    2017-01-31          0.000000  1.000000  0.018426      1
1        1    2017-01-31          0.091636  0.947368  0.111238      1
2        2    2017-01-31          0.235574  0.894737  0.324504      1
3        3    2017-01-31          0.379416  0.842105  0.498841      1
4        4    2017-01-31          0.451937  0.789474  0.543475      1
..     ...           ...               ...       ...       ...    ...
701    701    2019-12-31          0.921312  0.421053  0.470237     12
702    702    2019-12-31          0.925177  0.368421  0.382708     12
703    703    2019-12-31          0.947976  0.473684  0.635817     12
704    704    2019-12-31          0.982041  0.578947  0.913620     12
705    705    2019-12-31          0.985401  0.526316  0.845144     12

    Age Group
0       95-99
1       90-94
2       85-89
3       80-84
4       75-79
..        ...
701     40-44
702     35-39
703     45-49
704     55-59
705     50-54

[706 rows x 7 columns]
```

Creating the model

After all the preprocessing the model can be made, we use Total Population and Age to predict the Units

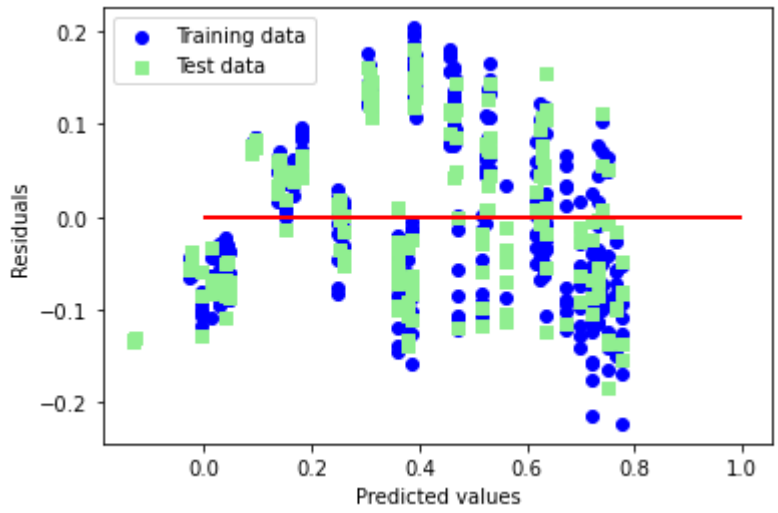
In [12]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X = res[['Total Population', 'Age']].values
y = res['Units'].values
X_train, X_test, y_train, y_test = train_test_split(
     X, y, test_size=0.3, random_state=0
)
slr = LinearRegression()
slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
print('Slope: %.3f' % slr.coef_[0])
print('Intercept: %.3f' % slr.intercept_)
```

```
Slope: 1.193
Intercept: -1.058
```

In [13]:

```
plt.scatter(y_train_pred, y_train_pred - y_train,
    c='blue', marker='o', label='Training data'
)
plt.scatter(y_test_pred,  y_test_pred - y_test,
    c='lightgreen', marker='s', label='Test data'
)
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-0, xmax=1, lw=2, color='red')
plt.show()
```



Measuring accuracy

After making the model we can check it's accuracy and visualize it by plotting a bar chart

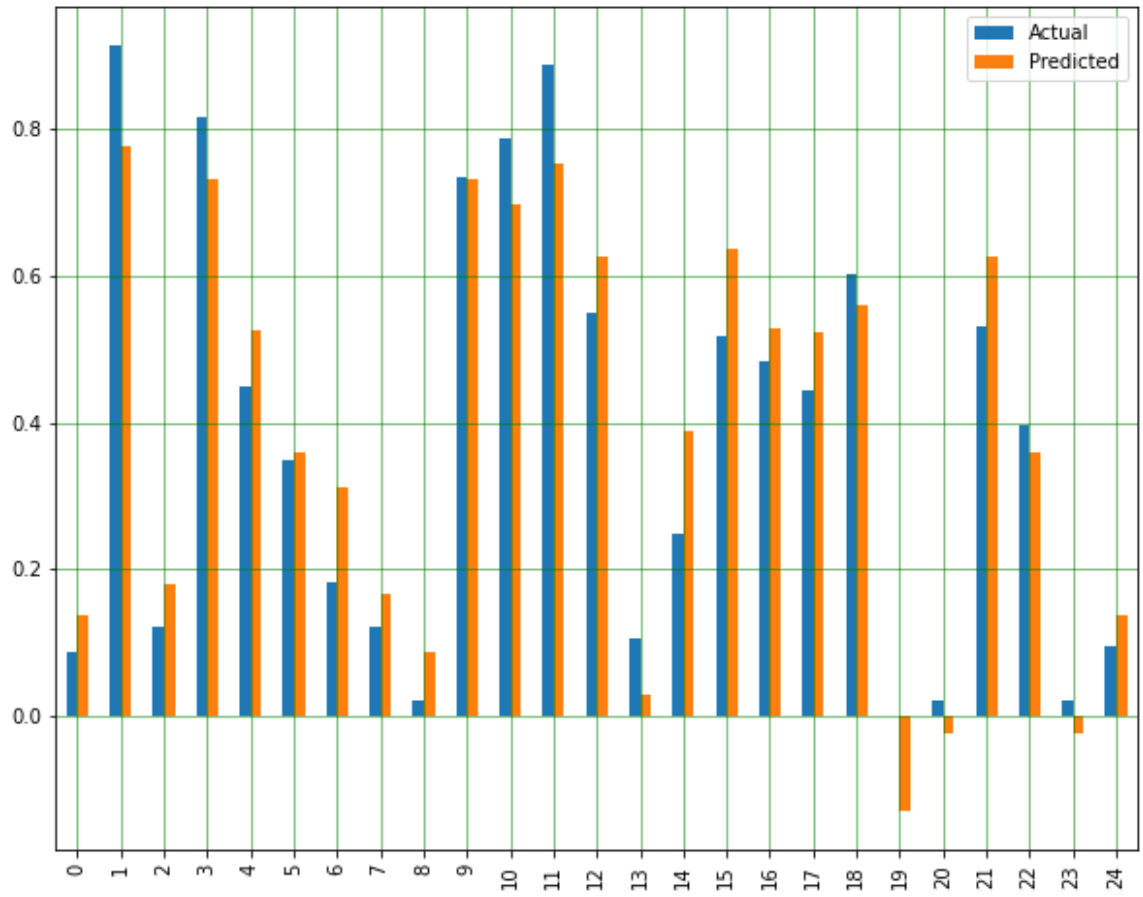
In [14]:

```
from sklearn.metrics import median_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
print("Mean Squared Error: ",mean_squared_error(y_test, y_test_pred))
errors = abs(y_test_pred-y_test)
print('Mean Absolute Error:', round(np.mean(errors), 2))
print('R2 score: ',r2_score(y_test, y_test_pred))
#print('Median Absolute Error: ',median_absolute_error(y_test, y_test_pred))
print('Linear Regression Accuracy: ', slr.score(X_test,y_test)*100)
```

```
Mean Squared Error:  0.007453403110110693
Mean Absolute Error: 0.07
R2 score:  0.8953789950699395
Linear Regression Accuracy:  89.53789950699395
```

In [15]:

```
df1 = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})
df2 = df1.head(25)
df2.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



In [ ]:

In [ ]:

```

```