Project 1

Author: Maciej Ogórek | WIMiIP| 306686

Git: https://github.com/MaciejOgorek/Project_FoDS

1. I started the work with deleting all restaurants not located in Poland to reduce the size of dataset and to speed up all the operations.

```
In [51]: df = df[df.country == 'Poland']
```

As a result I got a dataset consisting of 24698 restaurants.

```
In [54]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 24698 entries, 886009 to 910706
Data columns (total 42 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   restaurant_link                       24698 non-null  object
 1   restaurant_name                       24698 non-null  object
 2   original_location                     24698 non-null  object
 3   country                               24698 non-null  object
 4   region                                24698 non-null  object
 5   province                              24697 non-null  object
```

I took a look on the dataset and realized that dataset required lots of recoding. I decided to drop non relevant columns and columns consisting of data that couldn't be converted into numerical value:

```
In [519]: df = df.drop(columns = "region")
In [520]: df = df.drop(columns = "country")
In [521]: df = df.drop(columns = "restaurant_name")
In [522]: df = df.drop(columns = "original_location")
In [523]: df = df.drop(columns = "province")
In [524]: df = df.drop(columns = "city")
In [525]: df = df.drop(columns = "address")
In [526]: df = df.drop(columns = "restaurant_link")
In [527]: df = df.drop(columns = "default_language")
In [528]: df = df.drop(columns = "keywords")
In [529]: df = df.drop(columns = "latitude")
In [530]: df = df.drop(columns = "longitude")
In [531]: df = df.drop(columns = "original_open_hours")
In [532]: df = df.drop(columns = "popularity_detailed")
In [533]: df = df.drop(columns = "popularity_generic")
In [534]: df = df.drop(columns = "meals")
In [535]: df = df.drop(columns = "cuisines")
In [536]: df = df.drop(columns = "reviews_count_in_default_language")
In [537]: df = df.drop(columns = "total_reviews_count")
In [538]: df = df.drop(columns = "price_range")
In [539]: df = df.drop(columns = "atmosphere")
In [540]: df = df.drop(columns = "awards")
```

Since I didn't want to lose 20% of data I decided to replace all missing values of avg_rating with mean value of this variable divided by 2.

```
In [567]: df.avg_rating.mean()
Out[567]: 4.109489402697495
```

```
In [568]: df['avg_rating'] = df['avg_rating'].fillna(4.109489402697495/2)
```

Where it was possible I filled missing records with appropriate values.

```
In [552]: df['open_days_per_week'] = df['open_days_per_week'].fillna(7)
```

```
In [553]: df['working_shifts_per_week'] = df['working_shifts_per_week'].fillna(9)
```

```
In [554]: df['open_hours_per_week'] = df['open_hours_per_week'].fillna(56)
```

```
In [555]: df['excellent'] = df['excellent'].fillna(0)
```

```
In [556]: df['very_good'] = df['very_good'].fillna(0)
```

```
In [557]: df['average'] = df['average'].fillna(0)
```

```
In [558]: df['poor'] = df['poor'].fillna(0)
```

```
In [559]: df['terrible'] = df['terrible'].fillna(0)
```

```
In [560]: df['food'] = df['food'].fillna(0)
```

```
In [561]: df['service'] = df['service'].fillna(0)
```

```
In [562]: df['value'] = df['value'].fillna(0)
```

Where it was possible I recalculated the data to the values that were suitable for the analysis.

```
In [545]: df['vegetarian_friendly'] = df['vegetarian_friendly'].replace(['N', 'Y'], [0, 1]
```

```
In [546]: df['vegan_options'] = df['vegan_options'].replace(['N', 'Y'], [0, 1])
```

```
In [547]: df['gluten_free'] = df['gluten_free'].replace(['N', 'Y'], [0, 1])
```

```
In [548]: df['claimed'] = df['claimed'].fillna(0)
```

```
In [549]: df['price_level'] = df['price_level'].fillna(0)
```

```
In [550]: df['price_level'].value_counts()
Out[550]: €€-€€€     8976
          0          8670
          €          6710
          €€€€        342
          Name: price_level, dtype: int64
```

```
In [551]: df['price_level'] = df['price_level'].replace(['€', '€€-€€€','€€€€'], [0, 1, 2])
```

2. When the data was ready I prepared statistical summary of the data.

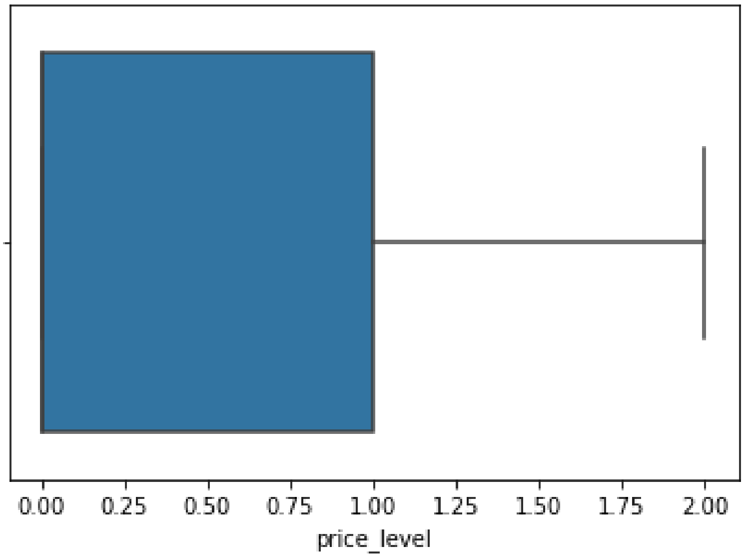
```
In [573]: df.describe().transpose()[['mean', 'std', 'min', 'max']]
```

Out[573]:

	mean	std	min	max
claimed	0.375658	0.484302	0.0	1.000000
price_level	0.391125	0.515608	0.0	2.000000
vegetarian_friendly	0.139404	0.346375	0.0	1.000000
vegan_options	0.058831	0.235312	0.0	1.000000
gluten_free	0.031582	0.174887	0.0	1.000000
open_days_per_week	6.852782	0.551613	1.0	7.000000
open_hours_per_week	67.513497	24.131297	0.0	167.883333
working_shifts_per_week	7.794599	1.289164	1.0	14.000000
avg_rating	3.781868	1.001736	1.0	5.000000
excellent	8.498502	45.176975	0.0	3120.000000
very_good	3.687424	17.116831	0.0	787.000000
average	1.318852	6.099053	0.0	435.000000
poor	0.628067	2.742321	0.0	164.000000
terrible	0.757227	3.234466	0.0	132.000000
food	1.697121	2.077531	0.0	5.000000
service	1.701879	2.071220	0.0	5.000000
value	1.661430	2.035065	0.0	5.000000

The results look solid all the numbers are logical. There are no values which would be totally abstract but of course there are some outliers.

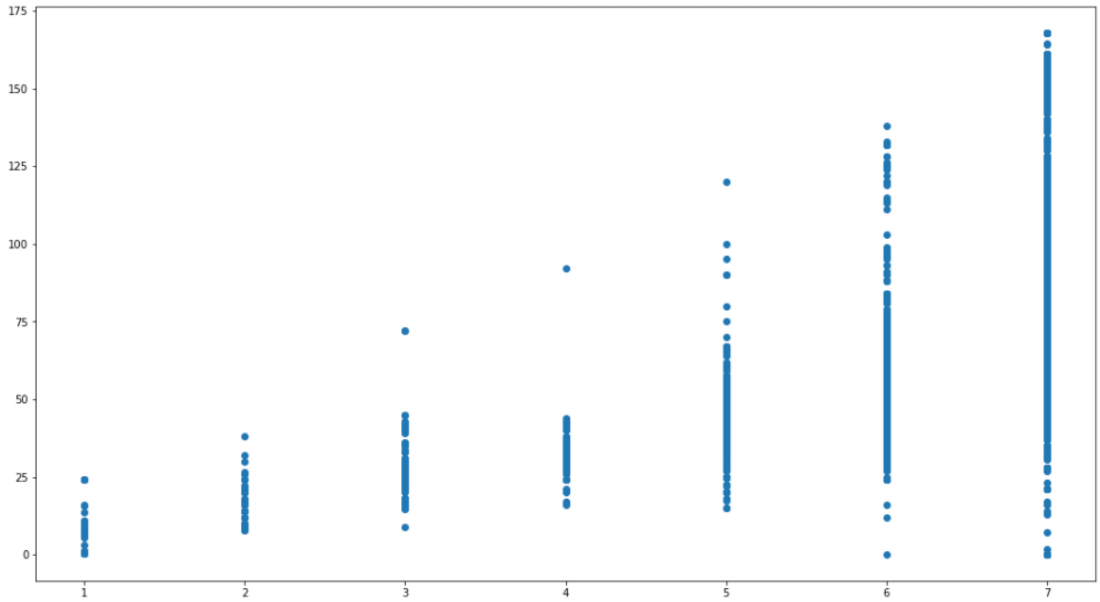
3. I started to identify outliers using univariate and multivariate methods:
Variable: price_level
Method: univariate



Variables: open_days_per_week, open_hours_per_week
Method: Multivariate

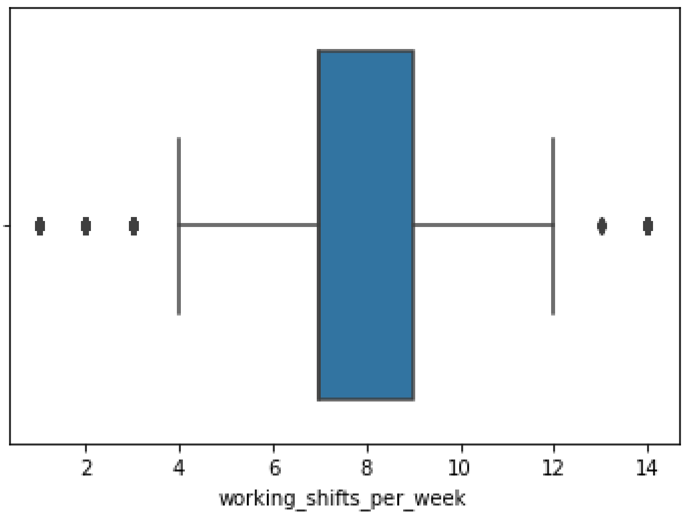
In [579]:
```
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(df['open_days_per_week'], df['open_hours_per_week'])
```

Out[579]: <matplotlib.collections.PathCollection at 0x1f521de5730>



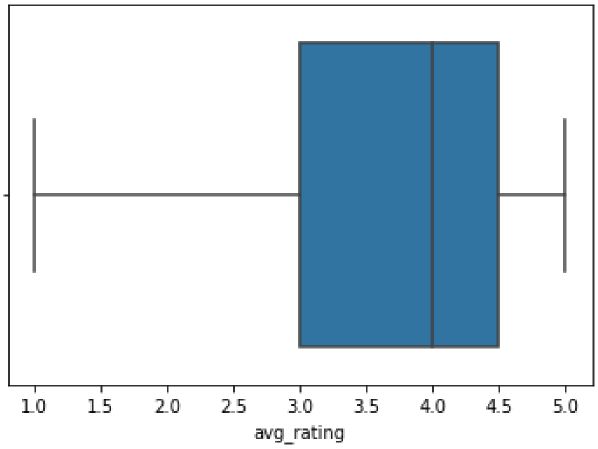
Variable: working_shifts_per_week
Method: univariate

Out[580]: <AxesSubplot:xlabel='working_shifts_per_week'>



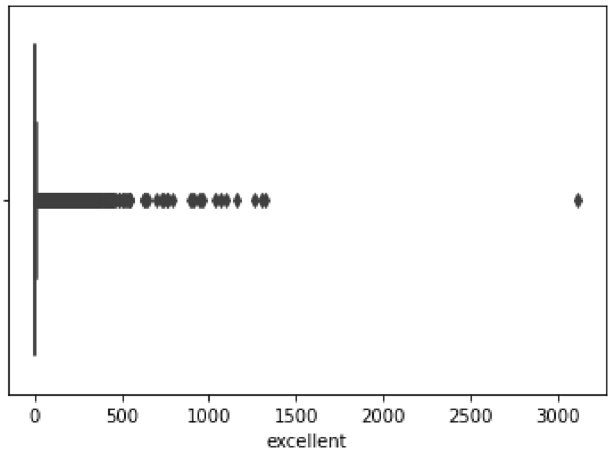
Variable: avg_rating
Method: univariate

```
: <AxesSubplot:xlabel='avg_rating'>
```



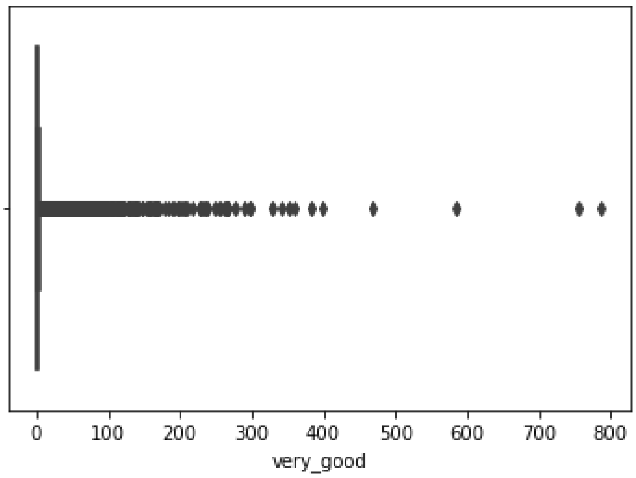
Variable: excellent
Method: univariate

```
Out[582]: <AxesSubplot:xlabel='excellent'>
```

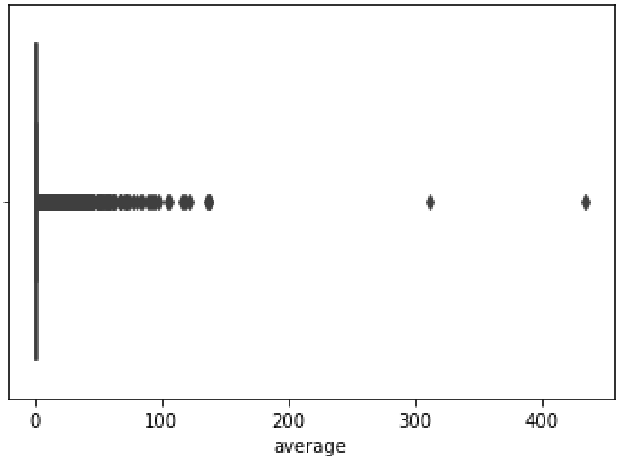


Variable: very_good
Method: univariate

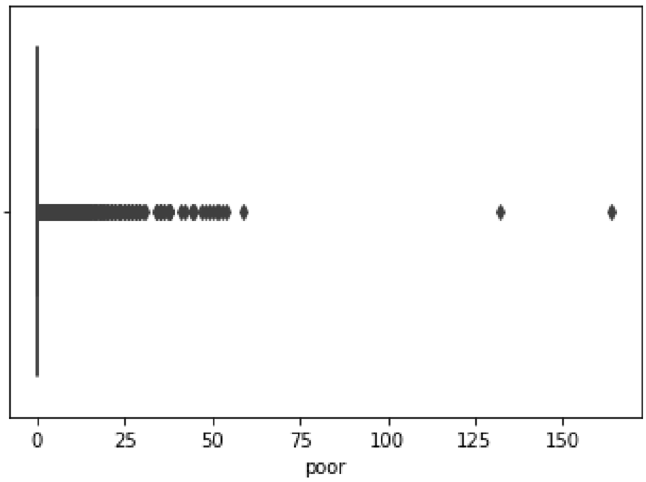
```
<AxesSubplot:xlabel='very_good'>
```



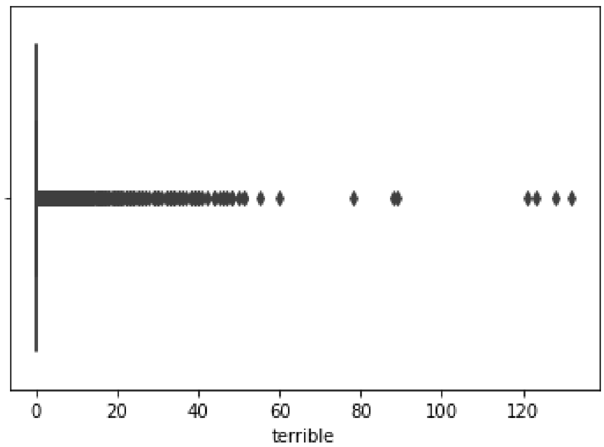
Variable: average
Method: univariate
<AxesSubplot:xlabel='average'>



Variable: poor
Method: univariate
<AxesSubplot:xlabel='poor'>



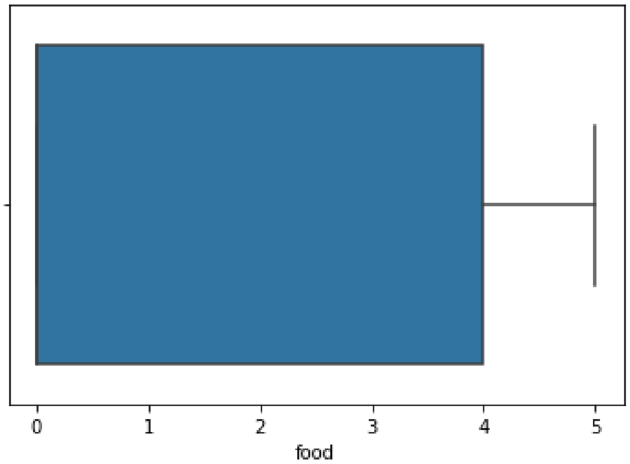
Variable: terrible
Method: univariate
<AxesSubplot:xlabel='terrible'>



Variable: food

Method: univariate

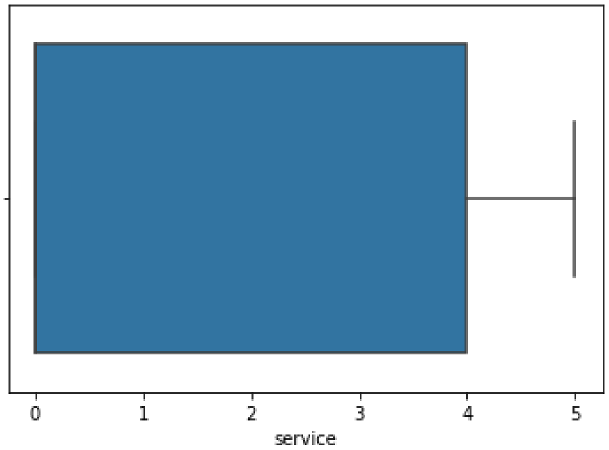
```
<AxesSubplot:xlabel='food'>
```



Variable: service

Method: univariate

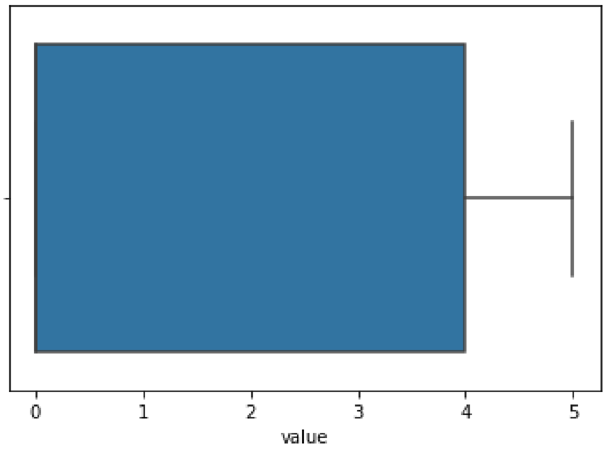
```
<AxesSubplot:xlabel='service'>
```



Variable: value

Method: univariate

```
<AxesSubplot:xlabel='value'>
```



On the charts we can see that the outliers can be seen when a variable has a wide range. Not all variables have an outliers but when data has strict limits for example grading scale its very hard to get an outlier.

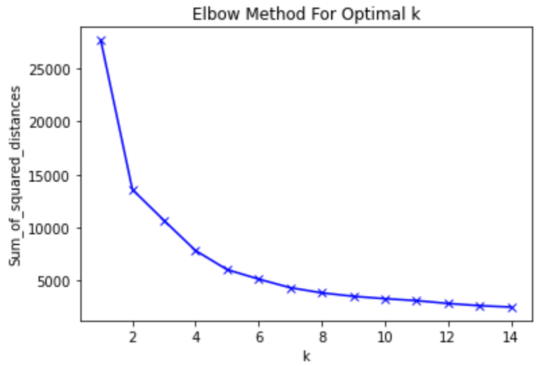
4. I used an elbow method to get an optimal K for K-means method of clustering

```
In [591]: cluster = df
```

```
In [592]: mms = MinMaxScaler()
          mms.fit(cluster)
          data_transformed = mms.transform(cluster)
```

```
In [597]: Sum_of_squared_distances = []
          K = range(1,15)
          for k in K:
              km = KMeans(n_clusters=k)
              km = km.fit(data_transformed)
              Sum_of_squared_distances.append(km.inertia_)
```

```
In [598]: plt.plot(K, Sum_of_squared_distances, 'bx-')
          plt.xlabel('k')
          plt.ylabel('Sum_of_squared_distances')
          plt.title('Elbow Method For Optimal k')
          plt.show()
```



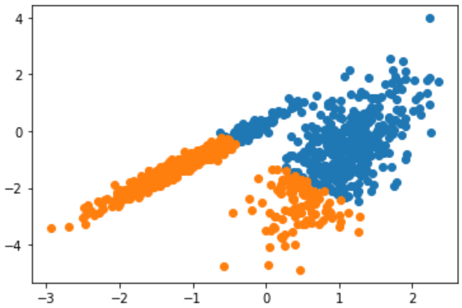
5. Using K =2 I clustered a dataset:

```
In [600]: X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)
```

```
In [607]: model = KMeans(n_clusters=2)
          model.fit(X)
Out[607]: KMeans(n_clusters=2)
```

```
In [608]: yhat = model.predict(X)
          clusters = unique(yhat)
```

```
In [609]: for cluster in clusters:
              row_ix = where(yhat == cluster)
              plt.scatter(X[row_ix, 0], X[row_ix, 1])
          plt.show()
```



6. I calculated Pearson's coefficient for variable avg_rating and every other variable.

   Variables: avg_rating, claimed
   Coefficient: 0.26849398078191256
   Correlation level: low

   Variables: avg_rating, price_level
   Coefficient: 0.20351258296287972
   Correlation level: low

   Variables: avg_rating, vegetarian_friendly
   Coefficient: 0.18230732451365278
   Correlation level: low

   Variables: avg_rating, vegan_options
   Coefficient: 0.1407207567915803
   Correlation level: low

   Variables: avg_rating, gluten_free
   Coefficient: 0.10776230260222618
   Correlation level: low

   Variables: avg_rating, open_days_per_week
   Coefficient: -0.08531272277307349
   Correlation level: low

   Variables: avg_rating, open_hours_per_week
   Coefficient: -0.1959115717325771
   Correlation level: low

   Variables: avg_rating, working_shifts_per_week
   Coefficient: -0.11487516310817568
   Correlation level: low

   Variables: avg_rating, very_good
   Coefficient: 0.07826583524069713
   Correlation level: low

   Variables: avg_rating, average
   Coefficient: 0.04541742766669098
   Correlation level: low

   Variables: avg_rating, poor
   Coefficient: 0.02073421845377288
   Correlation level: low

Variables: avg_rating, terrible
Coefficient: -0.028069473993855537
Correlation level: low

Variables: avg_rating, food
Coefficient: 0.2996473700779129
Correlation level: low

Variables: avg_rating, service
Coefficient: 0.2991681221982498
Correlation level: low

Variables: avg_rating, value
Coefficient: 0.3020495451493288
Correlation level: low

I estimated correlation level on a fact that Pearson's coefficient indicates strong link between variables when is between <-1;-0,5> and <0,5;1>. Weaker connection between variables is between <-0,5;0) and(0;0,5>. If the coefficient is equal to 0 two variables are not correlated.

7. I divided the dataset into two parts and created regression model

```
In [699]: train_dataset = df.sample(frac=0.8, random_state=0)
          test_dataset = df.drop(train_dataset.index)
```

```
In [700]: train_features = train_dataset.copy()
          test_features = test_dataset.copy()

          train_labels = train_features.pop('avg_rating')
          test_labels = test_features.pop('avg_rating')
```

```
In [701]: normalizer = preprocessing.Normalization()
          normalizer.adapt(np.array(train_features))
```

```
In [702]: print(normalizer.mean.numpy())

          [3.7878329e-01 3.9093027e-01 1.4050005e-01 6.0279381e-02 3.2543778e-02
           6.8520598e+00 6.7443253e+01 7.7924385e+00 8.5532446e+00 3.7136855e+00
           1.3156190e+00 6.2577182e-01 7.5463104e-01 1.6960977e+00 1.7019688e+00
           1.6601629e+00]
```

```
In [703]: first = np.array(train_features[:1])
          with np.printoptions(precision=2, suppress=True):
              print('First example:', first)
              print()
              print('Normalized:', normalizer(first).numpy())

          First example: [[ 1.   1.   1.   0.   0.   7.  56.   9.   5.   0.   0.   0.   0.   5.
             5.   4.5]]

          Normalized: [[ 1.28  1.18  2.47 -0.25 -0.18  0.27 -0.48  0.93 -0.09 -0.22 -0.22 -0.24
            -0.25  1.59  1.59  1.39]]
```

In [713]:
```python
def plot_loss(histpry):
    plt.plot(history.history['loss'], label = 'loss')
    plt.plot(history.history['val_loss'], label = 'val_loss')
    #plt.ylim([0, 40])
    plt.ylim([0, 2])
    plt.xlabel('Epoch')
    plt.ylabel('Error ')
    plt.legend()
    plt.grid(True)
```

In [714]:
```python
def build_and_compile_model(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])

    model.compile(
        #loss='mean_absolute_percentage_error',
        loss='mean_absolute_error',
        optimizer = tf.keras.optimizers.Adam(0.001))
    return model
```

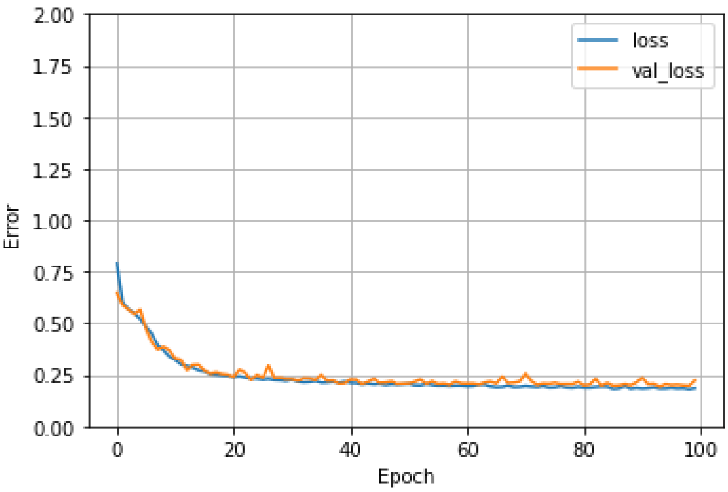
In [715]:
```python
dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()
```

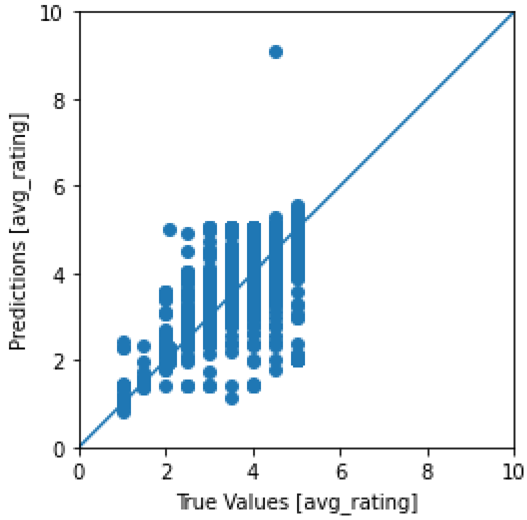
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
normalization (Normalization (None, 16)                33
_________________________________________________________________
dense_3 (Dense)              (None, 64)                1088
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 9,506
Trainable params: 9,473
Non-trainable params: 33
_________________________________________________________________
```

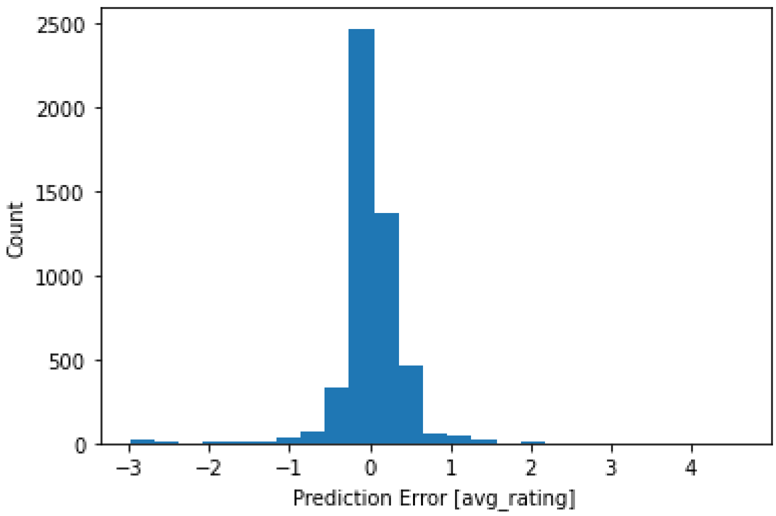
In [716]:
```python
%%time
history = dnn_model.fit(
    train_features, train_labels,
    validation_split = 0.2,
    verbose=1, epochs = 100)
```

```
Epoch 92/100
494/494 [==============================] - 0s 536us/step - loss: 0.1874 - val_loss: 0.2041
Epoch 93/100
494/494 [==============================] - 0s 548us/step - loss: 0.1846 - val_loss: 0.2060
Epoch 94/100
494/494 [==============================] - 0s 576us/step - loss: 0.1788 - val_loss: 0.1922
Epoch 95/100
494/494 [==============================] - 0s 558us/step - loss: 0.1784 - val_loss: 0.2044
Epoch 96/100
494/494 [==============================] - 0s 560us/step - loss: 0.1804 - val_loss: 0.1999
Epoch 97/100
494/494 [==============================] - 0s 570us/step - loss: 0.1831 - val_loss: 0.2015
Epoch 98/100
494/494 [==============================] - 0s 609us/step - loss: 0.1804 - val_loss: 0.1984
Epoch 99/100
494/494 [==============================] - 0s 561us/step - loss: 0.1765 - val_loss: 0.1965
Epoch 100/100
494/494 [==============================] - 0s 557us/step - loss: 0.1825 - val_loss: 0.2218
Wall time: 28.1 s
```

```python
plot_loss(history)
```







As it can be seen on the charts regression model returned a value 0.2 which is not a wanted value. However we can see that the value of error is very close to the predicted value. The unwanted result could be a result of some mistake made during the preparation of dataset. I don't think that the model was over-taught.