Report for Apartment rental offers in Germany data analysis

Farzaneh Ahmadi (400422012)

April 7, 2022

We first import the required packages for our analysis.

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

As the first step, the goal is to have an overview of the data features, the number of columns/rows, etc.

0.1 Preview data

[4]:
```
#load and preview data
root = '/gdrive/MyDrive/Data-Mining/immo/'
immo= pd.read_csv(root + "immo_data.csv")
```

[5]:
```
immo.sample(5)
```

	regio1	serviceCharge	heatingType	telekomTvOffer	telekomHybridUploadSpeed	newlyConst	balcony	picturecount	pricetrend
158625	Nordrhein_Westfalen	105.00	central_heating	ONE_YEAR_FREE	NaN	False	True	7	2.87
19295	Sachsen	90.05	central_heating	ONE_YEAR_FREE	NaN	False	True	2	3.85
137409	Nordrhein_Westfalen	95.00	central_heating	ONE_YEAR_FREE	NaN	False	False	10	3.49
166031	Bayern	150.00	central_heating	ONE_YEAR_FREE	NaN	False	True	12	3.59
152884	Niedersachsen	143.00	central_heating	NaN	NaN	True	False	8	3.61

5 rows × 49 columns

[6]:
```
immo.shape
```

[6]: (268850, 49)

[7]:
```
#An overview to numerical features
immo.describe()
```

[8]:
```
immo.info()
```

[9]:
```
immo.dtypes
```

[I deleted the result of immo.decribe, immo.info() and immo.dtype, to make the report more readable]

Renaming columns

[10]:
```
# Renaming columns
immo.rename(columns = {"regio1": "state", "regio2": "city"}, inplace = True)
```

0.2 Data cleaning (Task 1)

In this part, the goal is to deal with missing values, outliers, unnecessary columns and duplication.

[11]:
```
#checking for missing value
immo.isna().sum()/len(immo) * 100
```

[11]:
```
state                       0.000000
serviceCharge               2.569834
heatingType                16.684397
telekomTvOffer             12.132788
telekomHybridUploadSpeed   83.254603
newlyConst                  0.000000
balcony                     0.000000
picturecount                0.000000
pricetrend                  0.681421
telekomUploadSpeed         12.407662
totalRent                  15.070485
yearConstructed            21.218151
scoutId                     0.000000
noParkSpaces               65.388879
firingTypes                21.188023
hasKitchen                  0.000000
geo_bln                     0.000000
cellar                      0.000000
yearConstructedRange       21.218151
baseRent                    0.000000
houseNumber                26.415473
livingSpace                 0.000000
geo_krs                     0.000000
condition                  25.474800
interiorQual               41.906267
petsAllowed                42.615957
street                      0.000000
streetPlain                26.413614
lift                        0.000000
```

```
baseRentRange              0.000000
typeOfFlat                13.618747
geo_plz                    0.000000
noRooms                    0.000000
thermalChar               39.615399
floor                     19.084620
numberOfFloors            36.351869
noRoomsRange               0.000000
garden                     0.000000
livingSpaceRange           0.000000
city                       0.000000
regio3                     0.000000
description                7.344988
facilities                19.685326
heatingCosts              68.191185
energyEfficiencyClass     71.066766
lastRefurbish             69.979171
electricityBasePrice      82.575414
electricityKwhPrice       82.575414
date                       0.000000
dtype: float64
```

[12]:
```
# Delete columns with more than 30% null data
immo = immo.drop(columns=immo.columns[((immo.isna().sum()/len(immo)*100) > 30.
 ↪0)])
immo.columns
```

[12]:
```
Index(['state', 'serviceCharge', 'heatingType', 'telekomTvOffer', 'newlyConst',
       'balcony', 'picturecount', 'pricetrend', 'telekomUploadSpeed',
       'totalRent', 'yearConstructed', 'scoutId', 'firingTypes', 'hasKitchen',
       'geo_bln', 'cellar', 'yearConstructedRange', 'baseRent', 'houseNumber',
       'livingSpace', 'geo_krs', 'condition', 'street', 'streetPlain', 'lift',
       'baseRentRange', 'typeOfFlat', 'geo_plz', 'noRooms', 'floor',
       'noRoomsRange', 'garden', 'livingSpaceRange', 'city', 'regio3',
       'description', 'facilities', 'date'],
      dtype='object')
```

[13]:
```
#Delete irrelevant columns
immo.
 ↪drop(columns=['livingSpaceRange','street','description','facilities','geo_krs','geo_plz','sco
                  ␣
 ↪'telekomTvOffer','pricetrend','noRoomsRange','picturecount','geo_bln','date',
                 ␣
 ↪'houseNumber','streetPlain','firingTypes','yearConstructedRange',"regio3"],inplace=True)
```

Deleting columns with not appropriate value.(Properties with zero livingSpace or zero totalRent)

```
[14]: immo[immo['livingSpace'] == 0.0].shape[0]
```

```
[14]: 75
```

```
[15]: immo[immo['totalRent']== 0.0].shape[0]
```

```
[15]: 236
```

```
[16]: immo = immo.drop(immo[immo['livingSpace'] == 0.0].index)
      immo = immo.drop(immo[immo['totalRent'] == 0.0].index)
      immo.shape
```

```
[16]: (268544, 19)
```

Because I want to predict rental price ('totalRent') so I should drop all the rows that doesn't consist totalRent

```
[17]: immo.dropna(subset=['totalRent'],inplace=True)
```

Checking for the duplications

```
[18]: immo.duplicated().sum()
```

```
[18]: 5833
```

Getting rid of duplicates

```
[19]: immo= immo.drop_duplicates()
      immo.shape
```

```
[19]: (222211, 19)
```

Outlier treatment

In this method, the mean and standard deviation of the residuals are calculated and compared. If a value is a 3 of standard deviations away from the mean, that data point is identified as an outlier.

```
[20]: from re import I
      for cols in immo.columns:
          if immo[cols].dtype == 'int64' or immo[cols].dtype == 'float64':
              upper_range = immo[cols].mean() + 3 * immo[cols].std()
              lower_range = immo[cols].mean() - 3 * immo[cols].std()

              indexs = immo[(immo[cols] > upper_range) | (immo[cols] < lower_range)].
       ↪index
              immo = immo.drop(indexs)
```

[21]:
```
immo.shape
```

[21]:
```
(217098, 19)
```

Dealing with missing value (Fillna numeric data and categorical data)

[22]:
```
#Fill NaN values in numeric data by the mean
immo._get_numeric_data().mean()
```

[22]:
```
serviceCharge       146.769537
newlyConst            0.073879
balcony               0.615528
totalRent           773.206130
yearConstructed    1967.182693
hasKitchen            0.345733
cellar                0.649352
baseRent            605.405406
livingSpace          71.371320
lift                  0.230361
baseRentRange         3.671112
noRooms               2.586231
floor                 2.060264
garden                0.202508
dtype: float64
```

[23]:
```
immo.fillna(immo._get_numeric_data().mean(),inplace = True)
```

[24]:
```
#Fill NaN values in "heatingType" and "typeOfFlat" by the mode
immo['heatingType'].fillna(immo['heatingType'].mode()[0], inplace=True)
immo['typeOfFlat'].fillna(immo['typeOfFlat'].mode()[0], inplace=True)
```

[25]:
```
#Fill NAN values in "condition" by other
immo['condition'].fillna("other", inplace=True) # fill the NA by other
immo['condition'].value_counts()
```

[25]:
```
other                                 54206
well_kept                             53687
refurbished                           23218
fully_renovated                       21520
mint_condition                        17643
first_time_use                        16750
modernized                            14304
first_time_use_after_refurbishment    12933
negotiable                             1719
need_of_renovation                     1114
ripe_for_demolition                       4
```

```
Name: condition, dtype: int64
```

To reduce number of categories, the last 3 conditions which are not good conditions for the apartment will be grouped in 'other'

[26]:
```python
otherscondition = immo['condition'].value_counts().tail(3).index

def editcondition(dflist):
    if dflist in otherscondition:
        return 'other'
    else:
        return dflist

immo['condition'] =immo['condition'].apply(editcondition)
immo['condition'].value_counts()
```

[26]:
```
other                                 57043
well_kept                             53687
refurbished                           23218
fully_renovated                       21520
mint_condition                        17643
first_time_use                        16750
modernized                            14304
first_time_use_after_refurbishment    12933
Name: condition, dtype: int64
```

[27]:
```python
#checking for missing value
immo.isna().sum()
```

[27]:
```
state             0
serviceCharge     0
heatingType       0
newlyConst        0
balcony           0
totalRent         0
yearConstructed   0
hasKitchen        0
cellar            0
baseRent          0
livingSpace       0
condition         0
lift              0
baseRentRange     0
typeOfFlat        0
noRooms           0
floor             0
garden            0
```

```
city                   0
dtype: int64
```

Cleaned data:

[28]:
```
immo.head(10)
```

	state	serviceCharge	heatingType	newlyConst	balcony	totalRent	yearConstructed	hasKitchen	cellar	baseRent	livingSpace
0	Nordrhein_Westfalen	245.0	central_heating	False	False	840.00	1965.000000	False	True	595.00	86.00
2	Sachsen	255.0	floor_heating	True	True	1300.00	2019.000000	False	True	965.00	83.80
4	Bremen	138.0	self_contained_central_heating	False	True	903.00	1950.000000	False	False	765.00	84.97
6	Sachsen	70.0	self_contained_central_heating	False	False	380.00	1967.182693	False	True	310.00	62.00
7	Bremen	88.0	central_heating	False	True	584.25	1959.000000	False	True	452.25	60.30
8	Baden_Württemberg	110.0	oil_heating	False	False	690.00	1970.000000	True	True	580.00	53.00
10	Sachsen	88.0	central_heating	False	True	307.00	1930.000000	False	True	219.00	40.20
11	Sachsen	155.0	central_heating	False	False	555.00	1892.000000	False	True	400.00	80.00
12	Rheinland_Pfalz	270.0	oil_heating	False	False	920.00	1912.000000	False	False	650.00	100.00
13	Nordrhein_Westfalen	200.0	central_heating	False	False	1150.00	1951.000000	False	False	950.00	123.44

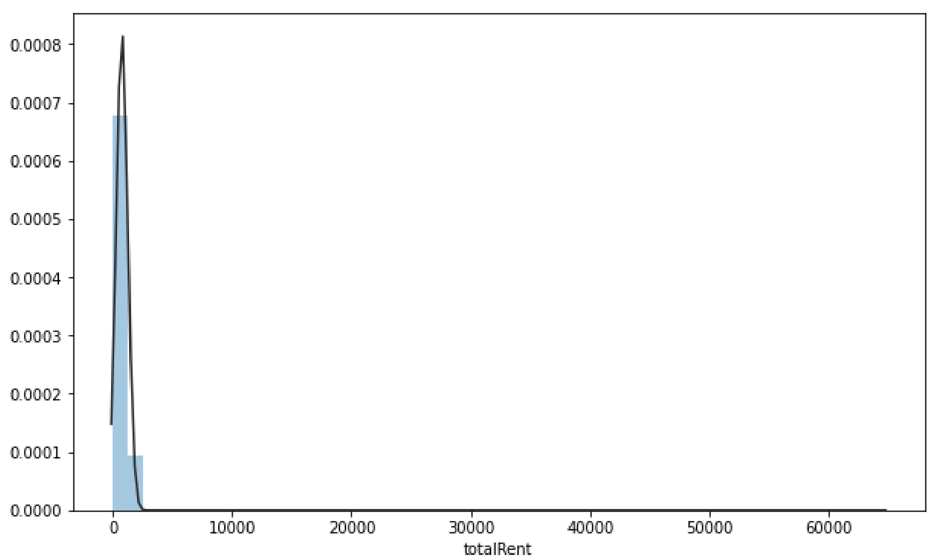
0.3 Data visualization

In this part the goal is to visualize data features and extract some useful information from the plots.

1. TotalRent range distribution

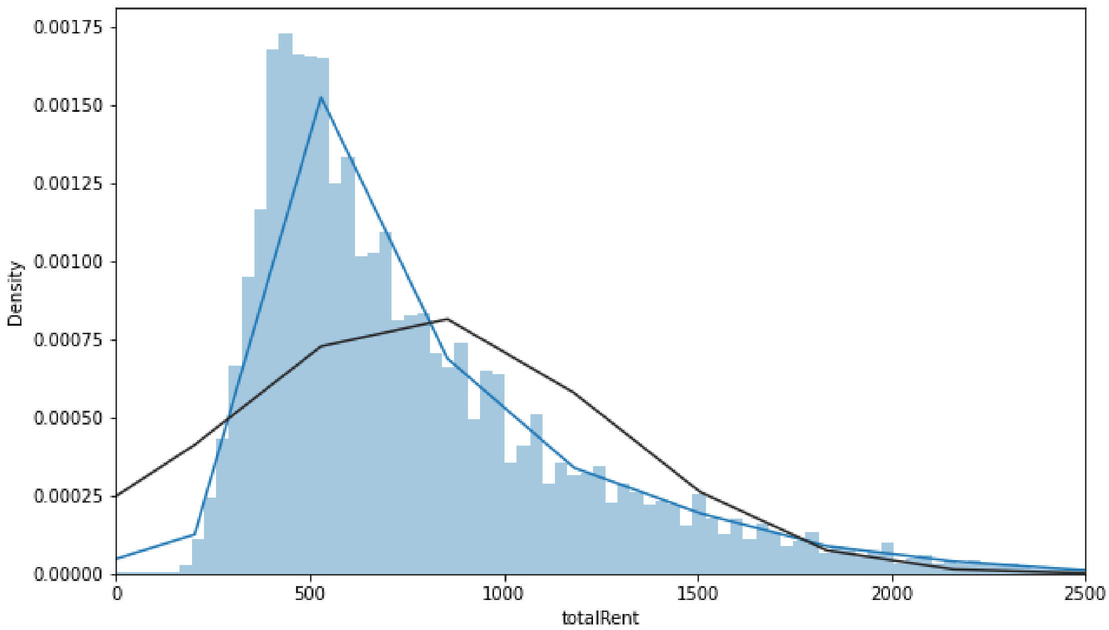
[29]:
```
#TotalRent range distribution plot
from scipy.stats import norm
fig,ax = plt.subplots(figsize=(10,6))
sns.distplot(a=immo.totalRent,kde= False, fit=norm)
```

[29]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fc9163b2850>`



[30]:
```
#totalRent in range (0, 2500) distribution
from scipy.stats import norm
fig,ax = plt.subplots(figsize=(10,6))
sns.distplot(immo['totalRent'],fit=norm, bins=2000)
ax.set_xlim(0, 2500)
```

[30]: (0.0, 2500.0)



The plot shows that:

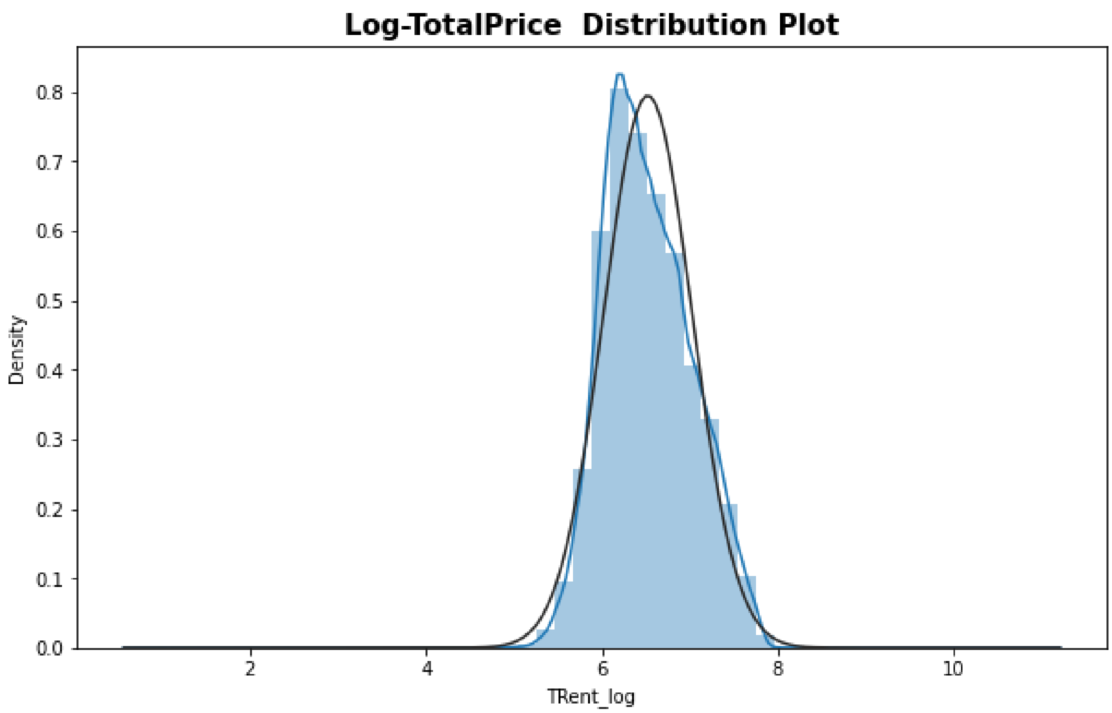
There is a right-skewed distribution on totalRent

TotalRent factor has an unstable distribution

[31]:
```
#Log transformation
immo['TRent_log'] = np.log(immo.totalRent+1)
```

[32]:
```
plt.figure(figsize=(10,6))
sns.distplot(immo['TRent_log'], fit=norm)
plt.title("Log-TotalPrice  Distribution Plot",size=15, weight='bold')
```

[32]: Text(0.5, 1.0, 'Log-TotalPrice Distribution Plot')

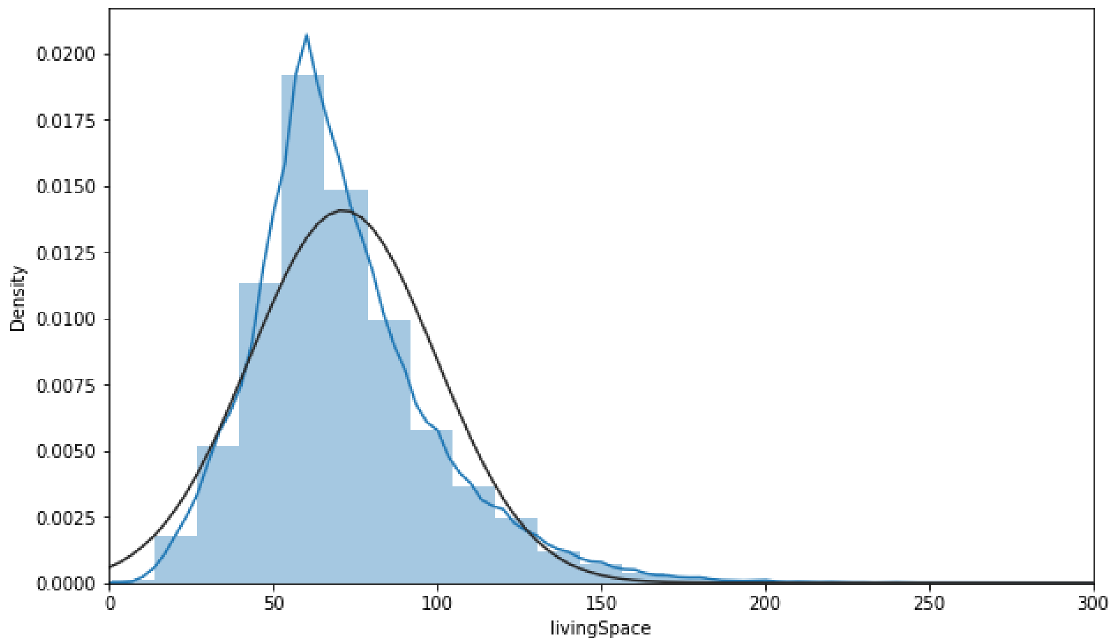


With log transformation, TotalRent feature has normal distribution.

2. Living space distribution

[33]:
```
fig,ax = plt.subplots(figsize=(10,6))
sns.distplot(immo['livingSpace'],fit=norm)
ax.set_xlim(0, 300)
```

[33]:
```
(0.0, 300.0)
```



3. How many properties are there in each state?

[34]:
```
#How many states are there?
immo.state.unique()
```

[34]:
```
array(['Nordrhein_Westfalen', 'Sachsen', 'Bremen', 'Baden_Württemberg',
       'Rheinland_Pfalz', 'Thüringen', 'Hessen', 'Niedersachsen',
       'Schleswig_Holstein', 'Bayern', 'Hamburg', 'Sachsen_Anhalt',
       'Mecklenburg_Vorpommern', 'Berlin', 'Brandenburg', 'Saarland'],
      dtype=object)
```

[35]:
```
# How many properties are there in each state?
statecount = immo.groupby('state').size()
state_count = pd.DataFrame({'count' : statecount}).reset_index()
state_count
```

[35]:

	state	count
0	Baden_Württemberg	12670
1	Bayern	17058
2	Berlin	8632
3	Brandenburg	6175
4	Bremen	2455
5	Hamburg	3069
6	Hessen	13562
7	Mecklenburg_Vorpommern	5751
8	Niedersachsen	11983

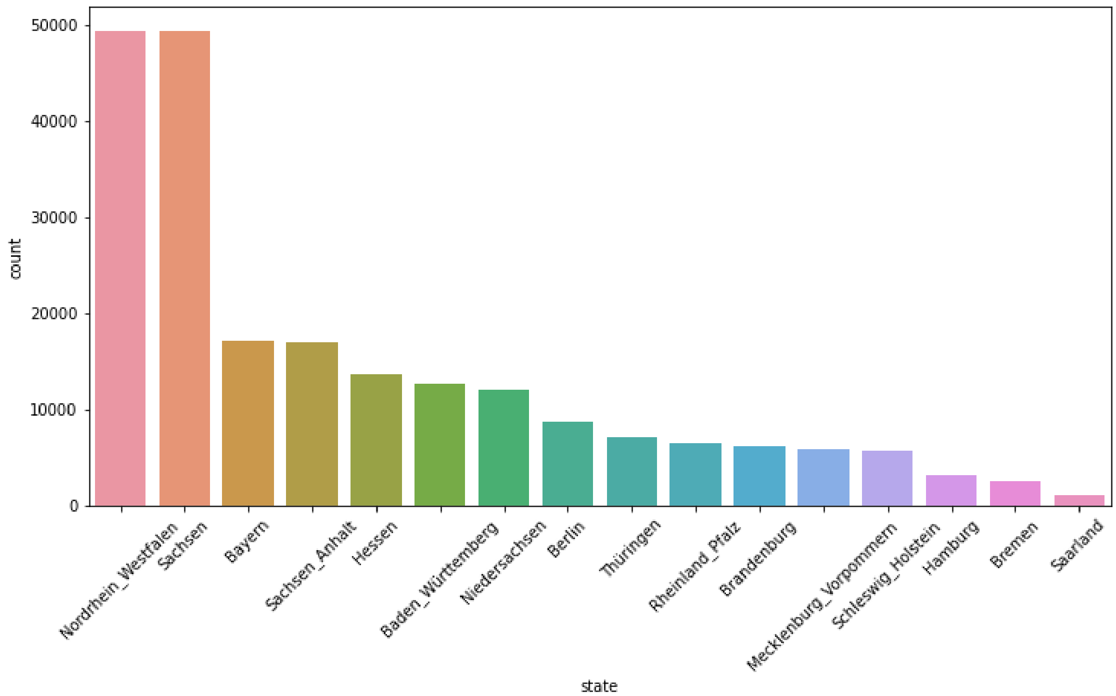
```
9      Nordrhein_Westfalen  49315
10         Rheinland_Pfalz   6411
11                Saarland   1008
12                 Sachsen  49307
13          Sachsen_Anhalt  16931
14      Schleswig_Holstein   5707
15               Thüringen   7064
```

[36]:
```python
from IPython.core.pylabtools import figsize
plt.figure(figsize=(12, 6))
#sns.catplot(x='state', kind="count", data= immo)

sns.countplot(immo["state"] ,order = immo['state'].value_counts().index )
plt.xticks(rotation = 45)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

[36]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 <a list of 16 Text major ticklabel objects>)
```



The plot shows that Nordrhein_ Westfalen is the state having the most properties in immo.

4. Top 10 cities having the most property

[37]:
```
#Top 10 cities having the most property
top_city=immo.city.value_counts().head(10)
top_city
```

[37]:
```
Leipzig              11985
Chemnitz             10703
Berlin                8632
Dresden               6074
Magdeburg             4380
Halle_Saale           3714
Essen                 3488
Frankfurt_am_Main     3077
Hamburg               3069
München               2965
Name: city, dtype: int64
```

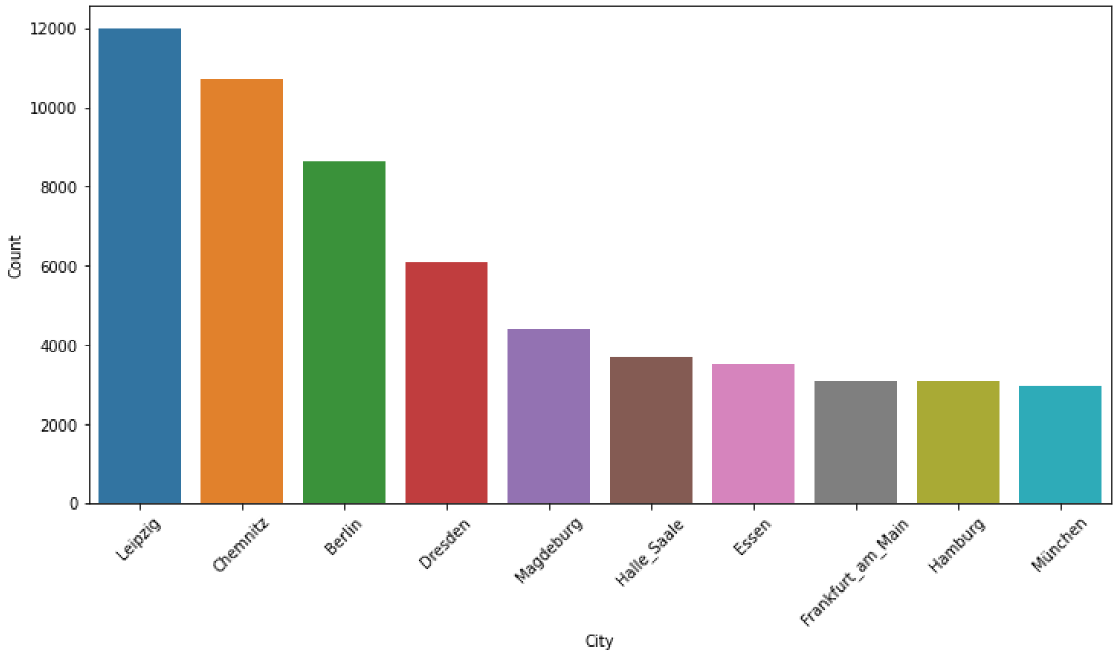
[38]:
```
# turning to data frame
top_city_df=pd.DataFrame(top_city)
top_city_df.reset_index(inplace=True)
top_city_df.rename(columns={'index':'City', 'city':'Count'}, inplace=True)
top_city_df
```

[38]:

	City	Count
0	Leipzig	11985
1	Chemnitz	10703
2	Berlin	8632
3	Dresden	6074
4	Magdeburg	4380
5	Halle_Saale	3714
6	Essen	3488
7	Frankfurt_am_Main	3077
8	Hamburg	3069
9	München	2965

[39]:
```
#Histogram
plt.figure(figsize=(12, 6))
sns.barplot(x='City',y='Count', data=top_city_df , order=top_city_df.
 ↪sort_values('Count',ascending = False).City)
plt.xticks(rotation = 45)
```

[39]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



The plot shows the top 10 cities having the most properties in immo.

Leipzing is the city having the most properties in immo.

5. Top 10 cities having the most total rent

[40]:
```
topCityPrice= immo[['city', 'totalRent']]
topCityPrice = immo.groupby(['city'])[['totalRent']].mean()
topCityPrice.totalRent.sort_values(ascending=False).head(10)
```

[40]:
```
city
München                     1508.230614
Freiburg_im_Breisgau        1425.895500
München_Kreis               1415.663138
Starnberg_Kreis             1400.007814
Frankfurt_am_Main           1334.584478
Miesbach_Kreis              1323.147500
Stuttgart                   1309.311071
Ebersberg_Kreis             1270.423081
Fürstenfeldbruck_Kreis      1265.542338
Hamburg                     1252.689677
Name: totalRent, dtype: float64
```

[41]:
```
#Turn to data frame
topCityPrice_df=pd.DataFrame(topCityPrice.totalRent.sort_values(ascending=False).
 ↪head(10))
topCityPrice_df.reset_index(inplace=True)
```

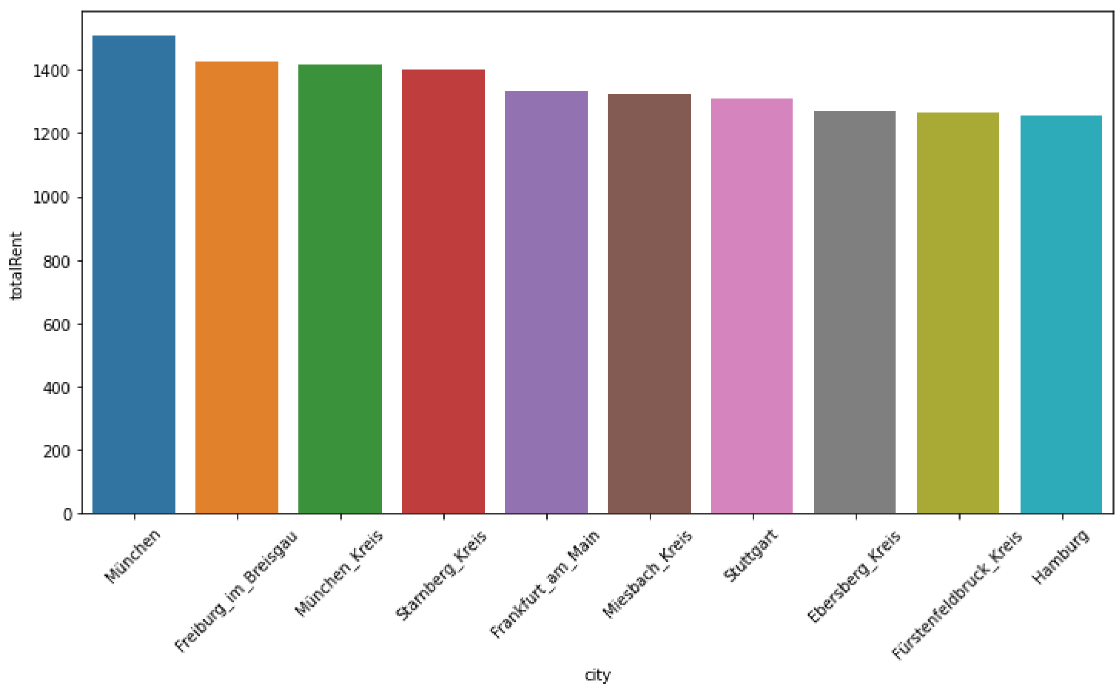
```
topCityPrice_df.rename(columns={'index':'City'}, inplace=True)
topCityPrice_df
```

[41]:

	city	totalRent
0	München	1508.230614
1	Freiburg_im_Breisgau	1425.895500
2	München_Kreis	1415.663138
3	Starnberg_Kreis	1400.007814
4	Frankfurt_am_Main	1334.584478
5	Miesbach_Kreis	1323.147500
6	Stuttgart	1309.311071
7	Ebersberg_Kreis	1270.423081
8	Fürstenfeldbruck_Kreis	1265.542338
9	Hamburg	1252.689677

[42]:
```
# Histogram
plt.figure(figsize=(12, 6))
sns.barplot(x='city',y='totalRent', data = topCityPrice_df)
plt.xticks(rotation = 45)
```

[42]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



The plot shows the 10 cities having the most expensive properties in immo in descending order. Munchen is the city with the most expensive properties in immo.

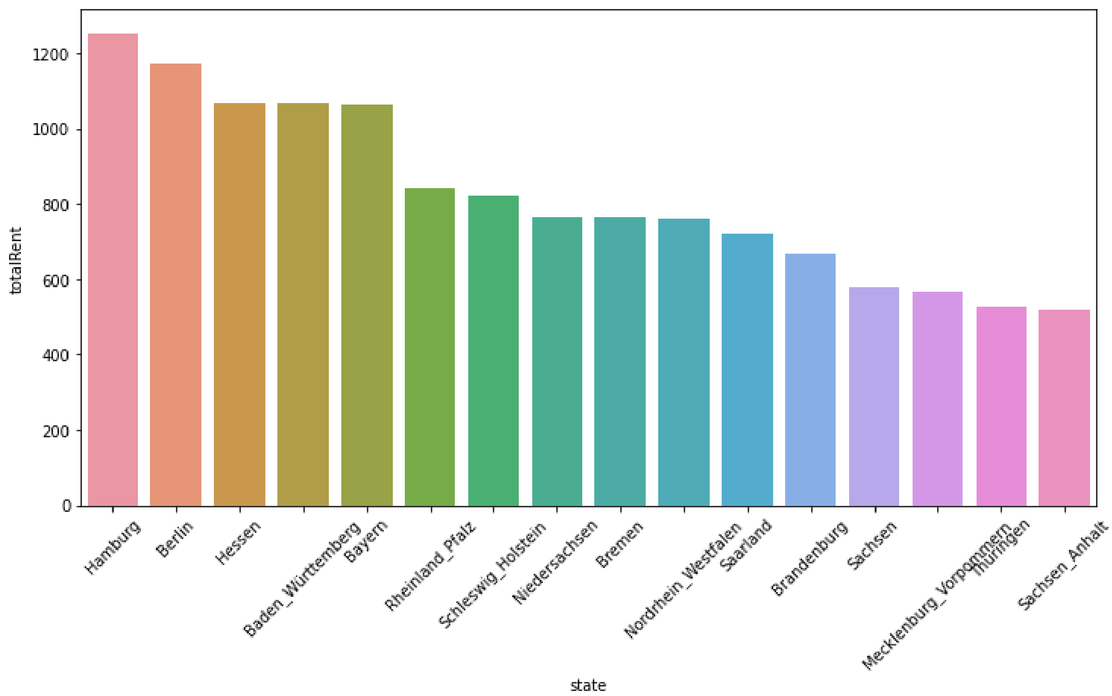
[43]:
```
state_rent= immo[['state', 'totalRent']]
state_rent = immo.groupby(['state'], as_index=False)[['totalRent']].mean()
state_rent
```

[43]:

	state	totalRent
0	Baden_Württemberg	1066.514131
1	Bayern	1062.311375
2	Berlin	1170.526045
3	Brandenburg	667.153338
4	Bremen	764.048705
5	Hamburg	1252.689677
6	Hessen	1068.476896
7	Mecklenburg_Vorpommern	568.522203
8	Niedersachsen	765.903618
9	Nordrhein_Westfalen	761.005055
10	Rheinland_Pfalz	841.537849
11	Saarland	719.303552
12	Sachsen	578.029478
13	Sachsen_Anhalt	518.882384
14	Schleswig_Holstein	822.161116
15	Thüringen	526.415089

[44]:
```
#Histogram
plt.figure(figsize=(12, 6))
sns.barplot(x="state",y="totalRent", data=state_rent, order=state_rent.
 ↪sort_values('totalRent',ascending = False).state)
plt.xticks(rotation = 45)
```

[44]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 <a list of 16 Text major ticklabel objects>)
```



The plot shows the distribution of totalRent across the states.

Hamborg is the state with the most expensive properties in immo.

0.4 Model building

[45]:
```
# Import usefull libraries
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

[46]:
```
# copy the DataFrame for building model
dfModel = immo.copy()
dfModel.head()
```

	state	serviceCharge	heatingType	newlyConst	balcony	totalRent	yearConstructed	hasKitchen	cellar	baseRent	livingSpace
0	Nordrhein_Westfalen	245.0	central_heating	False	False	840.00	1965.000000	False	True	595.00	86.00
2	Sachsen	255.0	floor_heating	True	True	1300.00	2019.000000	False	True	965.00	83.80
4	Bremen	138.0	self_contained_central_heating	False	True	903.00	1950.000000	False	False	765.00	84.97
6	Sachsen	70.0	self_contained_central_heating	False	False	380.00	1967.182693	False	True	310.00	62.00
7	Bremen	88.0	central_heating	False	True	584.25	1959.000000	False	True	452.25	60.30

Feature engineering: (Bonus task)

- Reduce number of categories:

Selecting only highest 20 city by quantity of data

[47]:
```
othersregion = list(dfModel['city'].value_counts().iloc[20:,].index)
def edit_region(dflist):
    if dflist in othersregion:
        return 'Other'
    else:
        return dflist

dfModel['city'] =dfModel['city'].apply(edit_region)
dfModel['city'].value_counts()
```

[47]:
```
Other                   135468
Leipzig                  11985
Chemnitz                 10703
Berlin                    8632
Dresden                   6074
Magdeburg                 4380
Halle_Saale               3714
Essen                     3488
Frankfurt_am_Main         3077
Hamburg                   3069
München                   2965
Düsseldorf                2934
Duisburg                  2756
Dortmund                  2531
Gelsenkirchen             2404
Mittelsachsen_Kreis       2364
Recklinghausen_Kreis      2235
Köln                      2148
Zwickau                   2145
Zwickau_Kreis             2105
Leipzig_Kreis             1921
Name: city, dtype: int64
```

- Create new columns:

Creating a new variable to tell the duration since last renovated till today

[48]:
```
import time
import datetime
from datetime import date
dfModel['numberOfYear'] = date.today().year - dfModel["yearConstructed"]
```

Create new columns for the price per square meter and addition cost (utilities).

[49]:
```python
dfModel['Price_m2'] = dfModel['baseRent'] / dfModel['livingSpace']
dfModel['additioncost'] = dfModel['totalRent'] - dfModel['baseRent']
```

	state	serviceCharge	heatingType	newlyConst	balcony	totalRent	hasKitchen	cellar	livingSpace	condition	...	baseRent
0	Nordrhein_Westfalen	245.0	central_heating	False	False	840.00	False	True	86.00	well_kept	...	
2	Sachsen	255.0	floor_heating	True	True	1300.00	False	True	83.80	first_time_use	...	
4	Bremen	138.0	self_contained_central_heating	False	True	903.00	False	False	84.97	refurbished	...	
6	Sachsen	70.0	self_contained_central_heating	False	False	380.00	False	True	62.00	fully_renovated	...	
7	Bremen	88.0	central_heating	False	True	584.25	False	True	60.30	other	...	

5 rows × 21 columns

Normalizing numeric data

[53]:
```python
for cols in dfModel.columns:
    if dfModel[cols].dtype == 'int64' or dfModel[cols].dtype == 'float64':
        if cols != 'totalRent':
            dfModel[cols] = ((dfModel[cols] - dfModel[cols].mean())/
 ↪(dfModel[cols].std()))

dfModel.head()
```

Convert categorical data to dummies variables

[54]:
```python
dfmodel_new = pd.get_dummies(dfModel, columns=['state', 'heatingType',␣
 ↪'newlyConst',
                                                'balcony', 'hasKitchen',␣
 ↪'cellar', 'condition', 'lift', 'typeOfFlat', 'garden', 'city'])
dfmodel_new.head()
```

	serviceCharge	totalRent	livingSpace	baseRentRange	noRooms	floor	TRent_log	numberOfYear	Price_m2	additioncost	...	city_Köln	city_Leipzig
0	1.319944	840.00	0.516404	0.155208	1.500857	-7.611405e-01	0.423552	6.411628e-02	-0.284916	0.302900	...	0	0
2	1.454316	1300.00	0.438742	1.099042	0.439257	6.746163e-01	1.292326	-1.522125e+00	0.483649	0.656025	...	0	0
4	-0.117838	903.00	0.480044	0.627125	0.439257	-7.611405e-01	0.567395	5.047390e-01	0.063607	-0.116926	...	0	0
6	-1.031569	380.00	-0.330815	-0.788626	-0.622343	-7.611405e-01	-1.153102	-1.165276e-15	-0.605690	-0.383732	...	0	0
7	-0.789699	584.25	-0.390826	-0.316709	0.439257	1.448146e-14	-0.298380	2.403654e-01	-0.187711	-0.140468	...	0	0

5 rows × 90 columns

[55]:
```python
# Split dataset into test and training data
y = dfmodel_new['totalRent']
x = dfmodel_new.drop(columns = ['totalRent'])
print(x.shape)
print(y.shape)
```

```
(217098, 89)
(217098,)
```

[56]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=0)
```

[57]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

Tree Regression

[58]:
```
#Preparing a Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
DTree=DecisionTreeRegressor(min_samples_leaf=.0001)
DTree.fit(x_train,y_train)
y_pred=DTree.predict(x_test)
```

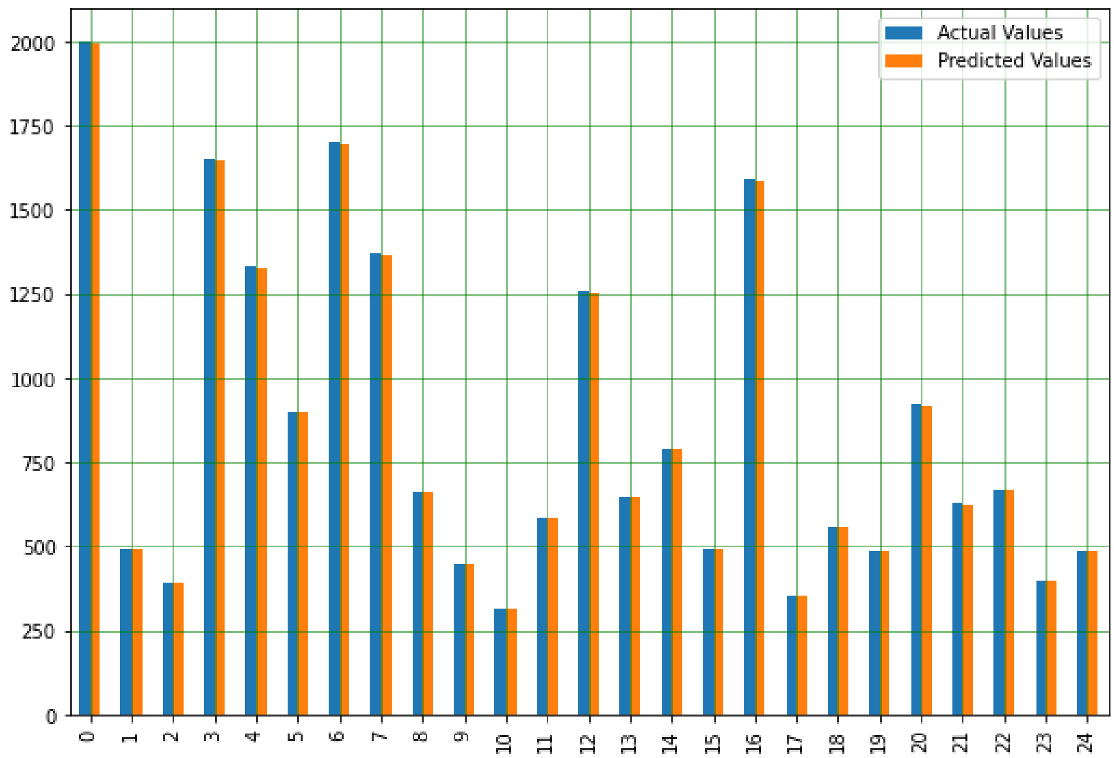
[59]:
```
from sklearn.metrics import r2_score
print("R2 score: ",r2_score(y_test,y_pred)*100)
print("RMSE: ",np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
R2 score:  71.18140938439608
RMSE:  277.3495559228842
```

[60]:
```
#Error
errordf2 = pd.DataFrame({'Actual Values': np.array(y_test).flatten(), 'Predicted␣
↪Values': y_pred.flatten()})
print(errordf2.head(5))
```

```
   Actual Values  Predicted Values
0         2001.0       1994.319091
1          490.0        489.604333
2          395.0        394.567778
3         1650.0       1644.938800
4         1328.9       1325.060526
```

[61]:
```
#Error visualization
df2 = errordf2.head(25)
df2.plot(kind='bar',figsize=(10,7))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



The plot shows the relationship between Actual Values and Predicted Values in tree regression model.In this model R squared equals to 0.71 which shows accuracy is almost good.

Lasso Regression

[62]:
```
regL1 = Lasso(alpha=0.01)
regL1.fit(x_train, y_train)

y_pred=regL1.predict(x_test)
```

[63]:
```
from sklearn.metrics import r2_score
print("R2 score: ",r2_score(y_test,y_pred)*100)
print("RMSE: ",np.sqrt(mean_squared_error(y_test,y_pred)))
```

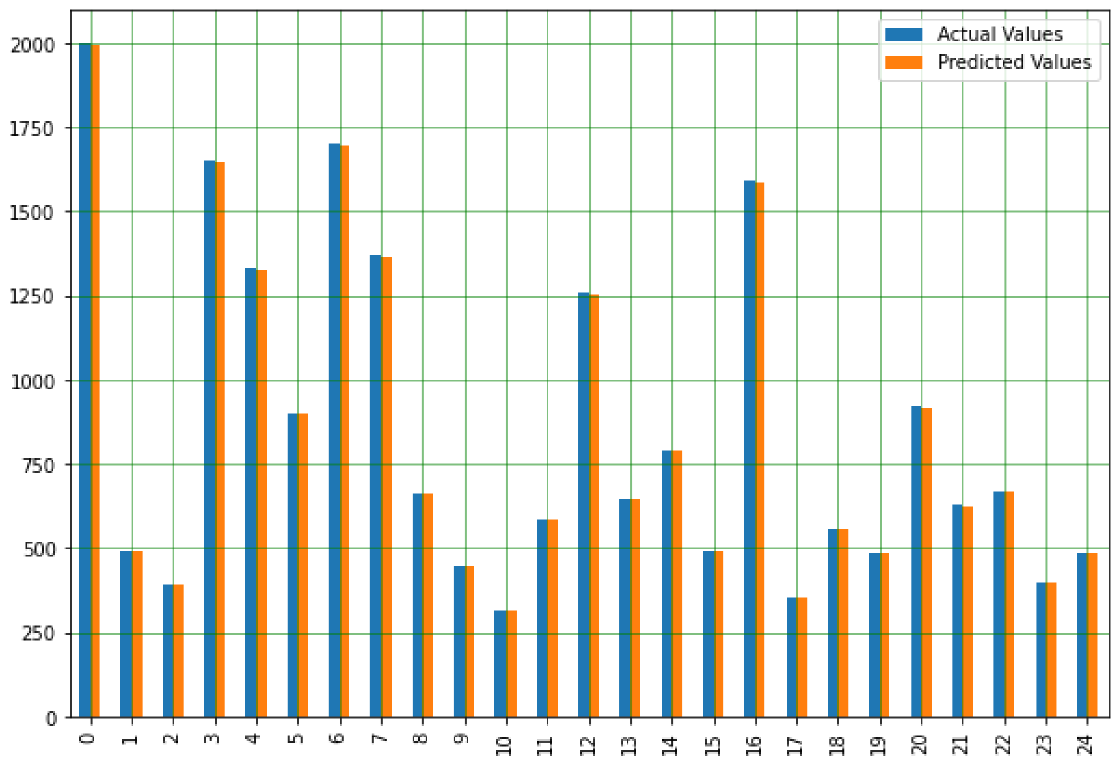
```
R2 score:  93.95143511703841
RMSE:  127.06246544119904
```

[64]:
```
#Error
errordf3 = pd.DataFrame({'Actual Values': np.array(y_test).flatten(), 'Predicted␣
↪Values': y_pred.flatten()})
print(errordf3.head(5))
```

```
   Actual Values  Predicted Values
0         2001.0       1753.739166
```

```
1         490.0        430.324166
2         395.0        330.581171
3        1650.0       1429.570368
4        1328.9       1287.655961
```

[65]:
```
#Error visualization
df2 = errordf2.head(25)
df2.plot(kind='bar',figsize=(10,7))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



The plot shows the relationship between Actual Values and Predicted Values in lasso regression model.In this model R squared equals to 0.93 which shows accuracy is pretty good.

Conclusion: In compare to tree regrassion lasso regression is better prediction model, as its R2 is 0.93