In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
matplotlib.rcParams["figure.figsize"] = (20,10)
from yellowbrick.datasets import load_concrete
from yellowbrick.regressor import ResidualsPlot
```

Importing data

In [2]:

```
#df = pd.read_csv('mumbai_prices.csv')
df = pd.read_csv(r'C:\Users\CG Lapy 2\Downloads\archive_mumbai.zip')
df.head()
```

Out[2]:

	area	bathroom_num	bedroom_num	city	desc	dev_name	floor_count	floor_num	furnishing	id	...	longitude	post_date
0	350.0	2.0	1	Mumbai	2 Bath,Unfurnished,East facing The project has...	NaN	NaN	NaN	Unfurnished	45349857	...	72.825882	2020-01-11
1	652.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,East facing A 1BHK apart...	Veena Group	NaN	NaN	Semi-Furnished	45960973	...	72.833592	2020-01-11
2	635.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,4 floor,West facing A be...	Agarwal Group	7.0	4.0	Semi-Furnished	46688849	...	72.801612	2019-12-13
3	540.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,East facing Essential Se...	NaN	NaN	NaN	Semi-Furnished	44696119	...	72.836006	2020-01-13
4	625.0	1.0	1	Mumbai	1 Bath,Furnished,2 floor,North facing 24 hours...	Millennium Group	7.0	2.0	Furnished	46742851	...	72.850167	2019-12-17

5 rows × 23 columns

In [3]:

```
df.shape
```

Out[3]:

```
(34348, 23)
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['area', 'bathroom_num', 'bedroom_num', 'city', 'desc', 'dev_name',
       'floor_count', 'floor_num', 'furnishing', 'id', 'id_string', 'latitude',
       'locality', 'longitude', 'post_date', 'poster_name', 'price', 'project',
       'title', 'trans', 'type', 'url', 'user_type'],
      dtype='object')
```

In [5]:

```
df.describe()
```

Out[5]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	id	latitude	longitude	price
count	33572.000000	34334.000000	34348.000000	31488.000000	31567.000000	3.434800e+04	34348.000000	34348.000000	3.434800e+04
mean	1177.387704	2.199278	2.076686	17.403551	8.577850	4.292717e+07	13.963273	52.420100	7.149190e+04
std	682.924385	0.880150	0.899821	13.996063	7.770904	6.954479e+06	9.131194	32.689143	7.717099e+04
min	10.000000	1.000000	1.000000	2.000000	-2.000000	2.074068e+06	0.000000	0.000000	2.200000e+03
25%	690.000000	2.000000	1.000000	7.000000	3.000000	4.249600e+07	0.000000	0.000000	3.200000e+04
50%	1040.000000	2.000000	2.000000	14.000000	6.000000	4.601324e+07	19.074359	72.839302	5.000000e+04
75%	1400.000000	3.000000	3.000000	22.000000	11.000000	4.678029e+07	19.150385	72.878328	8.000000e+04
max	9500.000000	8.000000	5.000000	120.000000	95.000000	4.733486e+07	73.071373	80.191436	1.200000e+06

In [6]:

```
df1 = df.copy()
```

In [7]:

```
(df1.city != 'Mumbai').unique()
```

Out[7]:

```
array([False])
```

In [8]:

```
df1.trans.unique()
```

Out[8]:

```
array(['Rent', nan], dtype=object)
```

In [9]:

```
df1.drop(['desc','dev_name','id','id_string','poster_name','project','url','trans', 'city','title','user_type','longitude','latitude'], a
df1.head()
```

Out[9]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing	locality	post_date	price	type
0	350.0	2.0	1	NaN	NaN	Unfurnished	Malad West	2020-01-11	9000	Apartment
1	652.0	2.0	1	NaN	NaN	Semi-Furnished	Vasai	2020-01-11	8060	Apartment
2	635.0	2.0	1	7.0	4.0	Semi-Furnished	Virar	2019-12-13	8000	Apartment
3	540.0	2.0	1	NaN	NaN	Semi-Furnished	Vasai East	2020-01-13	8000	Apartment
4	625.0	1.0	1	7.0	2.0	Furnished	Naigaon East	2019-12-17	9000	Apartment

Null values handling

In [10]:

```
df1.isnull().sum()
```

Out[10]:

```
area             776
bathroom_num      14
bedroom_num        0
floor_count     2860
floor_num       2781
furnishing        10
locality         208
post_date          0
price              0
type               0
dtype: int64
```

In [11]:

```
df1.dropna(inplace=True)
```

In [12]:

```
df1.shape
```

Out[12]:

```
(30507, 10)
```

In [13]:

```
df1.isnull().sum()
```

Out[13]:

```
area            0
bathroom_num    0
bedroom_num     0
floor_count     0
floor_num       0
furnishing      0
locality        0
post_date       0
price           0
type            0
dtype: int64
```

Feature Engineering

In [14]:

```
df1['year'] = df1['post_date'].apply(lambda x: str(x.split('-')[0]))
df1.year.unique()
```

Out[14]:

```
array(['2019', '2020'], dtype=object)
```

In [15]:

```
df1.drop('post_date',axis=1, inplace=True)
df1.head()
```

Out[15]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing	locality	price	type	year
2	635.0	2.0	1	7.0	4.0	Semi-Furnished	Virar	8000	Apartment	2019
4	625.0	1.0	1	7.0	2.0	Furnished	Naigaon East	9000	Apartment	2019
5	630.0	2.0	1	15.0	9.0	Unfurnished	Virar West	8000	Apartment	2020
6	690.0	2.0	1	7.0	4.0	Furnished	Virar	8500	Apartment	2020
7	338.0	1.0	1	5.0	2.0	Unfurnished	Royal Palms Estate	10000	Apartment	2020

In [16]:

```
df2 = df1.copy()
```

In [17]:

```
df2['price_per_area'] = df2['price']/df2['area']
df2.head(3)
```

Out[17]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing	locality	price	type	year	price_per_area
2	635.0	2.0	1	7.0	4.0	Semi-Furnished	Virar	8000	Apartment	2019	12.598425
4	625.0	1.0	1	7.0	2.0	Furnished	Naigaon East	9000	Apartment	2019	14.400000
5	630.0	2.0	1	15.0	9.0	Unfurnished	Virar West	8000	Apartment	2020	12.698413

In [18]:

```
df2.price_per_area.describe()
```

Out[18]:

```
count    30507.000000
mean        59.856951
std        161.541249
min          2.812500
25%         38.461538
50%         50.000000
75%         69.565217
max      20000.000000
Name: price_per_area, dtype: float64
```

In [19]:

```
df3 = df2[~(df2.area/df2.bedroom_num<200)]
df3.shape
```

Out[19]:

```
(30473, 11)
```

In [20]:

```
df3 = df2[~(df2.area/df2.bathroom_num<300)]
df3.shape
```

Out[20]:

```
(29318, 11)
```

In [21]:

```
df3 = df2[~(df2.bedroom_num/df2.bathroom_num < 1)]
df3.shape
```

Out[21]:

```
(25135, 11)
```

In [22]:

```
df3.head()
```

Out[22]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing	locality	price	type	year	price_per_area
4	625.0	1.0	1	7.0	2.0	Furnished	Naigaon East	9000	Apartment	2019	14.400000
7	338.0	1.0	1	5.0	2.0	Unfurnished	Royal Palms Estate	10000	Apartment	2020	29.585799
9	500.0	1.0	1	6.0	2.0	Semi-Furnished	Virar West	8000	Apartment	2019	16.000000
16	560.0	1.0	1	4.0	3.0	Furnished	Silver Park	10000	Apartment	2019	17.857143
17	600.0	1.0	1	7.0	4.0	Unfurnished	Vasai West	10000	Apartment	2019	16.666667

In [23]:

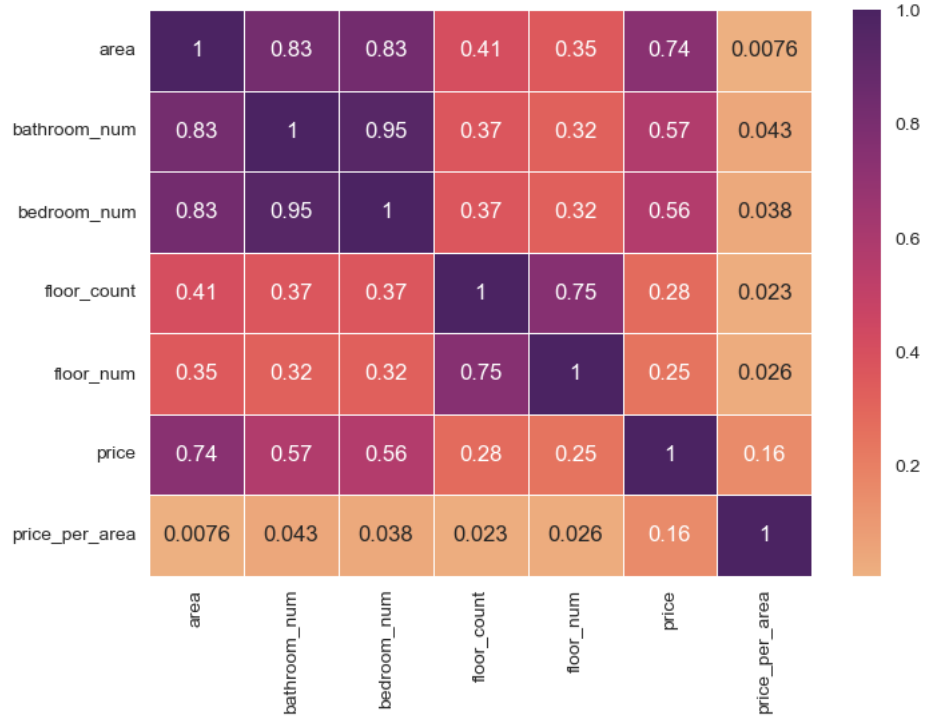
```
matrix = df3.corr()
matrix
```

Out[23]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	price	price_per_area
area	1.000000	0.827339	0.829663	0.407107	0.354728	0.738568	0.007576
bathroom_num	0.827339	1.000000	0.945993	0.366543	0.321268	0.574867	0.042804
bedroom_num	0.829663	0.945993	1.000000	0.368524	0.323175	0.563137	0.038087
floor_count	0.407107	0.366543	0.368524	1.000000	0.752303	0.279174	0.022561
floor_num	0.354728	0.321268	0.323175	0.752303	1.000000	0.245676	0.026069
price	0.738568	0.574867	0.563137	0.279174	0.245676	1.000000	0.155907
price_per_area	0.007576	0.042804	0.038087	0.022561	0.026069	0.155907	1.000000

In [24]:

```
sns.heatmap(matrix, cmap=sns.color_palette("flare", as_cmap=True), linewidth=0.5, annot=True);
```



In [25]:

```
#scatter plot to check for outliers
```

In [26]:

```
fig, ax = plt.subplots(5, figsize=(30, 25))
ax[0].scatter(x = df3['area'], y = df3['price'], color = 'slateblue')
ax[0].set_xlabel("Area")
ax[0].set_ylabel("Price")

ax[1].scatter(x = df3['floor_count'], y = df3['price'], color='rebeccapurple')
ax[1].set_xlabel("Number of floors")
ax[1].set_ylabel("Price")

ax[2].scatter(x = df3['floor_num'], y = df3['price'], color = 'darkviolet')
ax[2].set_xlabel(" Floor number")
ax[2].set_ylabel("Price")

ax[3].scatter(x = df3['bathroom_num'], y = df3['price'], color = 'violet')
ax[3].set_xlabel("Number of bathrooms")
ax[3].set_ylabel("Price")

ax[4].scatter(x = df3['bedroom_num'], y = df3['price'], color = 'fuchsia')
ax[4].set_xlabel("Number of bedrooms")
ax[4].set_ylabel("Price")


plt.show()
```

In [27]:

```
df4 = df3.copy()
```

In [28]:

```
#outlier removal
```

In [29]:

```
df3 = df3[(df3.floor_num < 60)]
df3 = df3[(df3.floor_count < 80)]
df3 = df3[(df3.area < 6001)]
```

In [30]:

```
df3.shape
```

Out[30]:

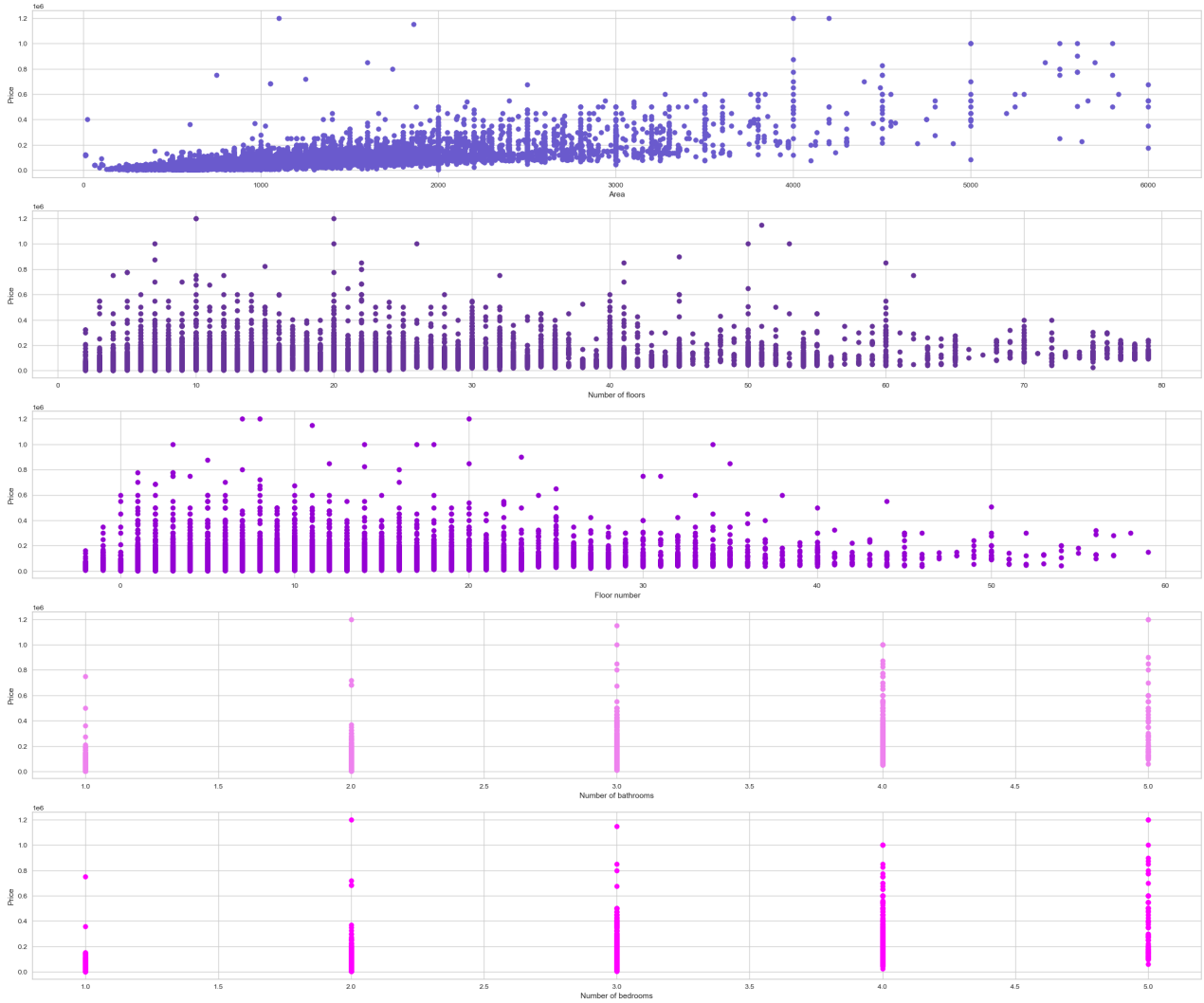
```
(25092, 11)
```

In [31]:

```
# scatter plot after outlier removal
```

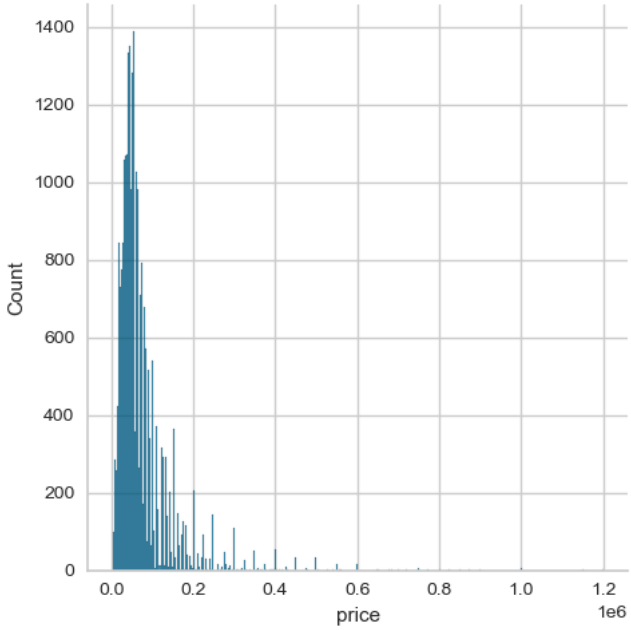
In [32]:

```
fig, ax = plt.subplots(5, figsize=(30, 25))
ax[0].scatter(x = df3['area'], y = df3['price'], color = 'slateblue')
ax[0].set_xlabel("Area")
ax[0].set_ylabel("Price")

ax[1].scatter(x = df3['floor_count'], y = df3['price'], color='rebeccapurple')
ax[1].set_xlabel("Number of floors")
ax[1].set_ylabel("Price")

ax[2].scatter(x = df3['floor_num'], y = df3['price'], color = 'darkviolet')
ax[2].set_xlabel(" Floor number")
ax[2].set_ylabel("Price")

ax[3].scatter(x = df3['bathroom_num'], y = df3['price'], color = 'violet')
ax[3].set_xlabel("Number of bathrooms")
ax[3].set_ylabel("Price")

ax[4].scatter(x = df3['bedroom_num'], y = df3['price'], color = 'fuchsia')
ax[4].set_xlabel("Number of bedrooms")
ax[4].set_ylabel("Price")

plt.show()
```



In [33]:

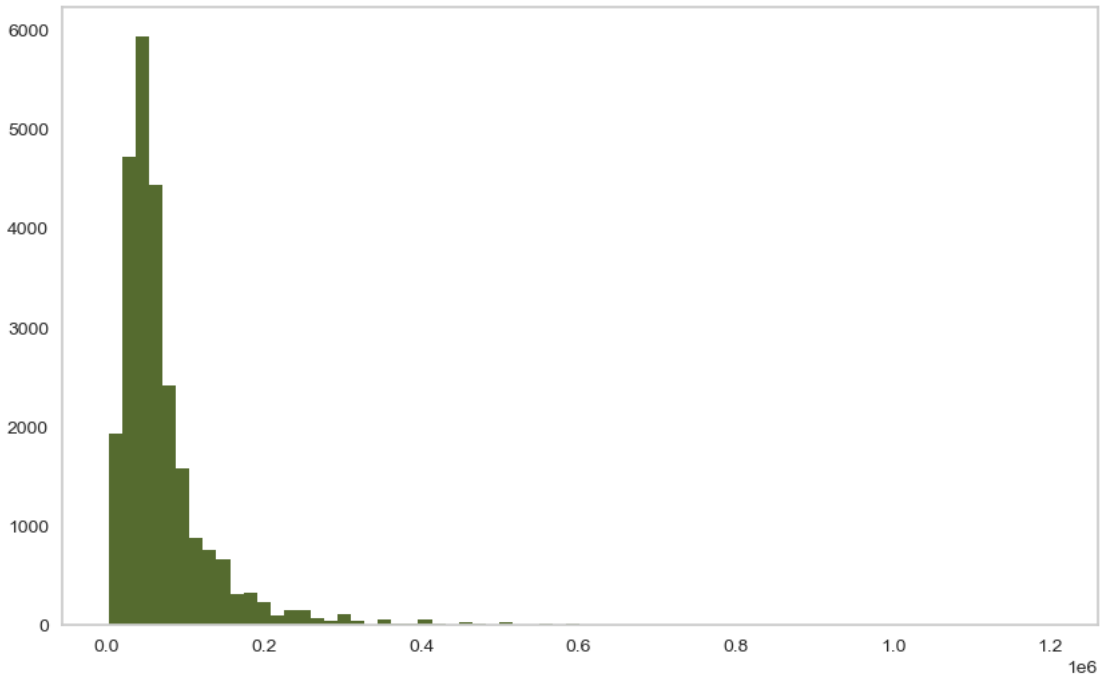
```
sns.displot(df3['price']);
fig = plt.figure(figsize=(20,10));
```




```
<Figure size 2000x1000 with 0 Axes>
```

In [34]:

```
df3['price'].hist(bins = 70, grid = False, color = 'darkolivegreen', figsize = (10,6))
plt.show()
```



In [35]:

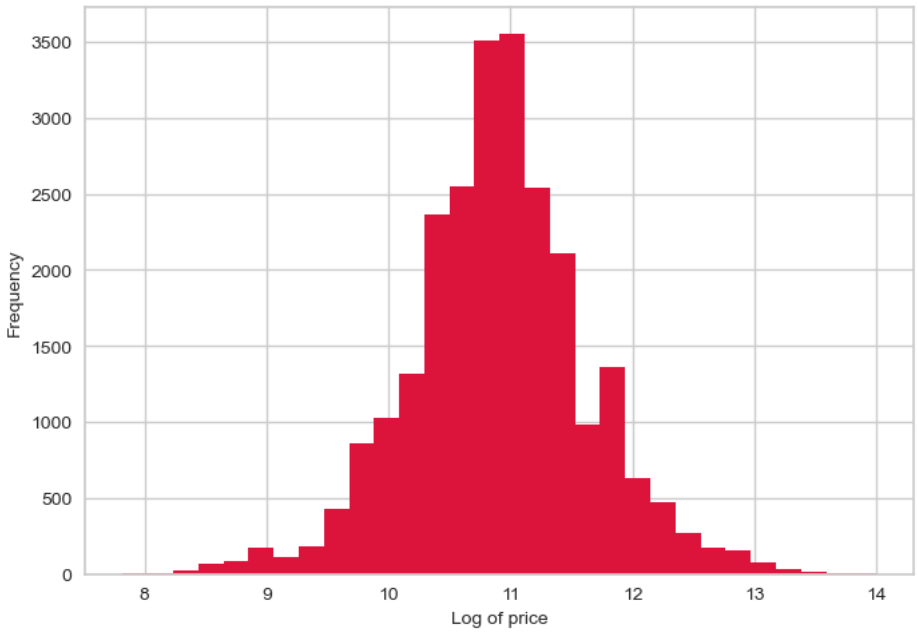
```
df3.agg({'price':['mean', 'median','skew','kurtosis', 'std','min','max']})
```

Out[35]:

	price
mean	7.191442e+04
median	5.300000e+04
skew	4.493041e+00
kurtosis	3.630693e+01
std	6.942018e+04
min	2.500000e+03
max	1.200000e+06

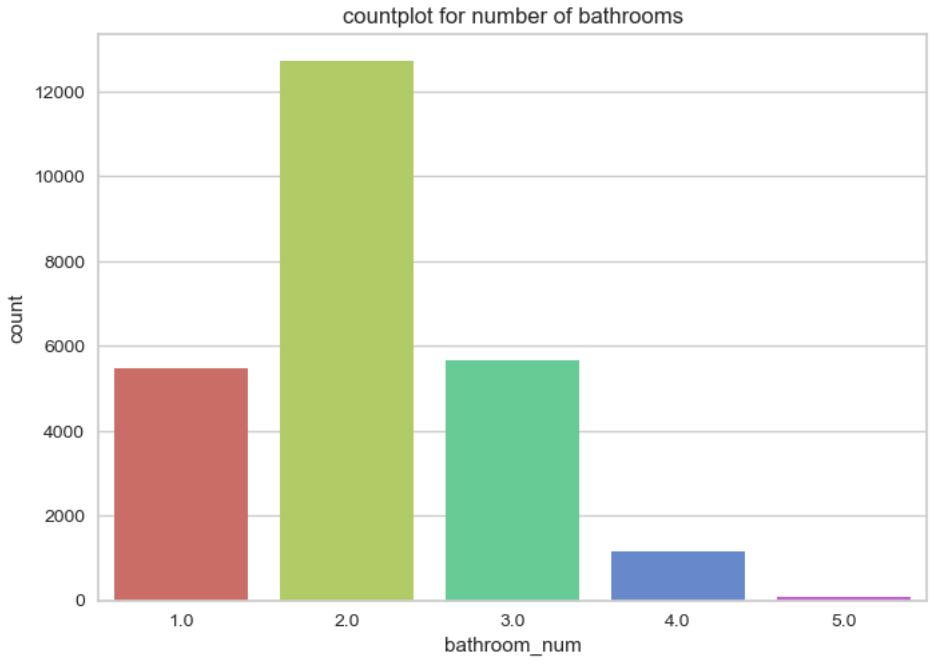
In [36]:

```
np.log(df3['price']).plot.hist(bins=30, color = 'crimson')
plt.xlabel('Log of price', fontsize=10)
plt.ylabel('Frequency', fontsize=10)
plt.xticks(fontsize =10)
plt.yticks(fontsize = 10)
plt.show()
```



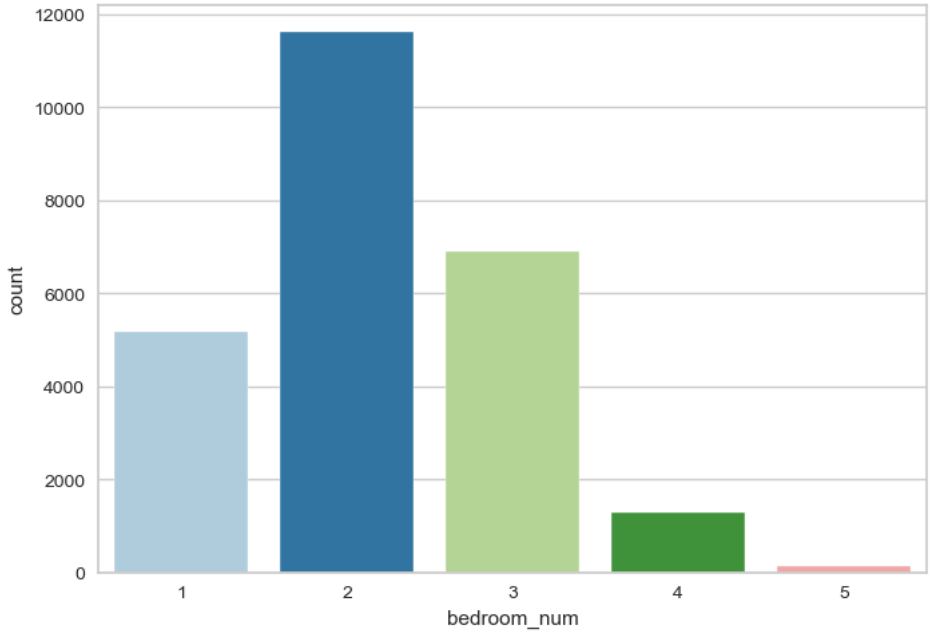
In [37]:

```
sns.countplot(x = 'bathroom_num', data=df3, palette='hls');
plt.title("countplot for number of bathrooms")
sns.set_style("whitegrid")
```



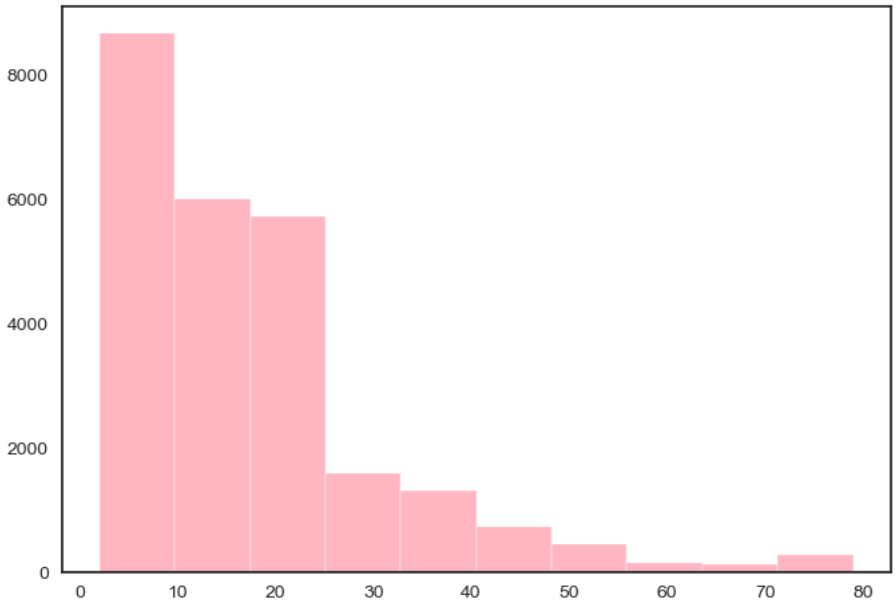
In [38]:

```
sns.countplot(x = 'bedroom_num', data=df3, palette='Paired');
sns.set_style("white")
```



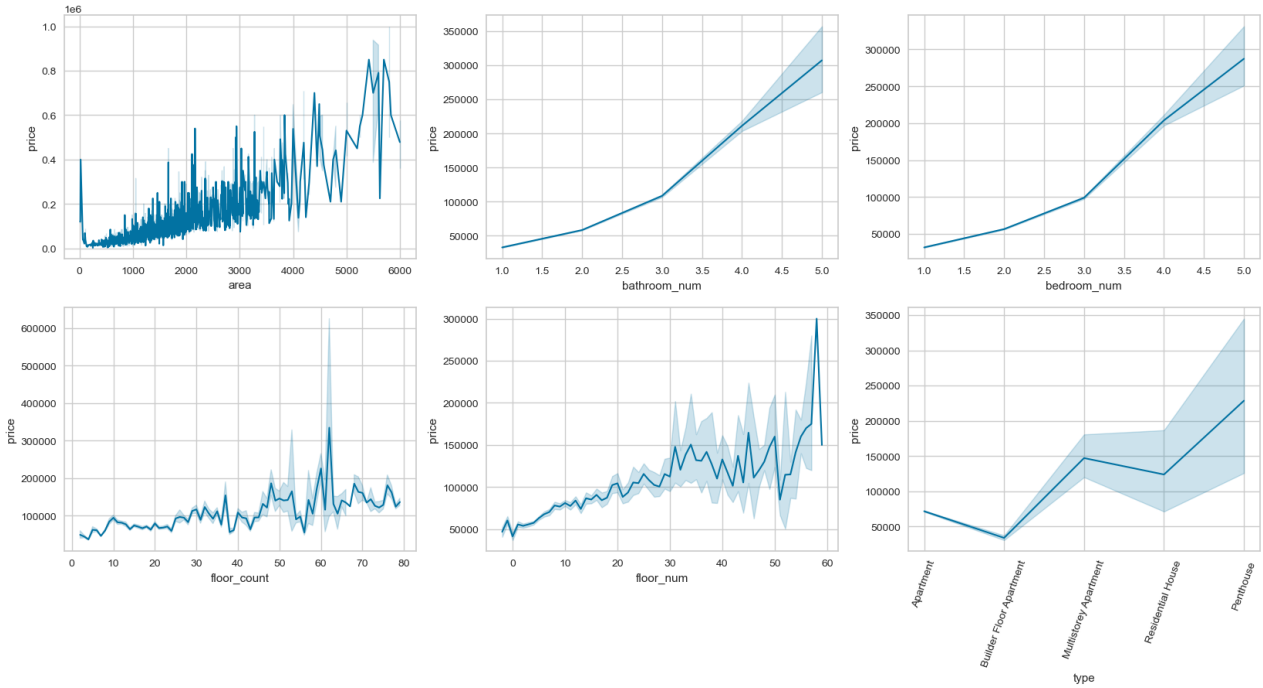
In [39]:

```
plt.hist(df3['floor_count'], edgecolor='white', color='lightpink');
sns.set_style("whitegrid")
```



In [40]:

```
fig, ax = plt.subplots(2, 3, figsize=(20, 9))
ax = ax.flatten()

sns.set()
sns.lineplot(data=df3, x="area", y="price", ax=ax[0]);

sns.lineplot(data=df3, x="bathroom_num", y="price", ax=ax[1])

sns.lineplot(data=df3, x="bedroom_num", y="price", ax=ax[2])

sns.lineplot(data=df3, x="floor_count", y="price", ax=ax[3])

sns.lineplot(data=df3, x="floor_num", y="price", ax=ax[4])

sns.lineplot(data=df3, x="type", y="price", ax=ax[5]);

plt.xticks(rotation=70)

plt.show();
```



In [41]:

```
df3.describe()
```

Out[41]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	price	price_per_area
count	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	2.509200e+04	25092.000000
mean	1193.128407	2.110792	2.187470	17.742707	8.730950	7.191442e+04	59.688686
std	612.136996	0.804317	0.835193	13.725897	7.594724	6.942018e+04	171.010888
min	11.000000	1.000000	1.000000	2.000000	-2.000000	2.500000e+03	3.076923
25%	812.000000	2.000000	2.000000	7.000000	4.000000	3.500000e+04	38.228737
50%	1090.000000	2.000000	2.000000	15.000000	7.000000	5.300000e+04	50.000000
75%	1425.000000	3.000000	3.000000	22.000000	12.000000	8.400000e+04	69.444444
max	6000.000000	5.000000	5.000000	79.000000	59.000000	1.200000e+06	20000.000000

One hot encoding

In [42]:

```
dummies1 = pd.get_dummies(df3.furnishing)
print(dummies1)
dummies2 = pd.get_dummies(df3.locality)
print(dummies2)
dummies3 = pd.get_dummies(df3.type)
print(dummies3)
```

```
       Furnished  Semi-Furnished  Unfurnished
4              1               0            0
7              0               0            1
9              0               1            0
16             1               0            0
17             0               0            1
...          ...             ...          ...
34302          0               0            1
34305          0               0            1
34309          0               0            1
34326          0               0            1
34347          0               0            1

[25092 rows x 3 columns]
       4 Bunglows  AAI Residential Complex  Aarey Milk Colony  Abhinav Nagar  \
4               0                        0                  0              0
7               0                        0                  0              0
9               0                        0                  0              0
16              0                        0                  0              0
```

In [43]:

```
df3 = pd.concat([df3, dummies1.drop('Unfurnished', axis='columns'), dummies2.drop('Zeezamata Nagar', axis='columns'), dummies3.drop('Resi
```

In [44]:

```
df3.drop(['furnishing', 'locality', 'type'], axis= 'columns', inplace=True)
df3.head()
```

Out[44]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	price	year	price_per_area	Furnished	Semi-Furnished	...	Yari Road	Yashavant Nagar	Yashwant Nagar	Ye
4	625.0	1.0	1	7.0	2.0	9000	2019	14.400000	1	0	...	0	0	0	
7	338.0	1.0	1	5.0	2.0	10000	2020	29.585799	0	0	...	0	0	0	
9	500.0	1.0	1	6.0	2.0	8000	2019	16.000000	0	1	...	0	0	0	
16	560.0	1.0	1	4.0	3.0	10000	2019	17.857143	1	0	...	0	0	0	
17	600.0	1.0	1	7.0	4.0	10000	2019	16.666667	0	0	...	0	0	0	

5 rows × 779 columns

In [45]:

```
df3.shape
```

Out[45]:

```
(25092, 779)
```

In [46]:

```
X = df3.drop('price', axis='columns')
X.head()
```

Out[46]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	year	price_per_area	Furnished	Semi-Furnished	4 Bunglows	...	Yari Road	Yashavant Nagar	Yashwant Nagar
4	625.0	1.0	1	7.0	2.0	2019	14.400000	1	0	0	...	0	0	0
7	338.0	1.0	1	5.0	2.0	2020	29.585799	0	0	0	...	0	0	0
9	500.0	1.0	1	6.0	2.0	2019	16.000000	0	1	0	...	0	0	0
16	560.0	1.0	1	4.0	3.0	2019	17.857143	1	0	0	...	0	0	0
17	600.0	1.0	1	7.0	4.0	2019	16.666667	0	0	0	...	0	0	0

5 rows × 778 columns

In [47]:

```
y = df3.price
y.head()
```

Out[47]:

```
4      9000
7     10000
9      8000
16    10000
17    10000
Name: price, dtype: int64
```

In [48]:

```
#Normalizing the data
```

In [49]:

```
from sklearn.preprocessing import Normalizer

# create the Normalizer object
normalizer = Normalizer()

# fit the normalizer to the data
normalizer.fit(X)

# transform the data
X_normalized = normalizer.transform(X)
```

In [50]:

```
X_normalized
```

Out[50]:

```
array([[2.95705786e-01, 4.73129258e-04, 4.73129258e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.65014508e-01, 4.88208604e-04, 4.88208604e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.40377337e-01, 4.80754674e-04, 4.80754674e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [2.84858469e-01, 4.74764114e-04, 4.74764114e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.06436058e-01, 4.71440090e-04, 4.71440090e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [4.32234132e-01, 7.85880240e-04, 7.85880240e-04, ...,
        0.00000000e+00, 3.92940120e-04, 0.00000000e+00]])
```

In [51]:

```
data = pd.DataFrame(X_normalized)
```

In [52]:

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

In [53]:

```
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2, random_state=0)
```

K-nearest neighbor

In [54]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error as mae
```

In [55]:

```
knn = KNeighborsRegressor(n_neighbors = 5)
```

In [56]:

```
knn.fit(X_train, y_train)
```

Out[56]:

```
KNeighborsRegressor()
```

In [57]:

```
pred = knn.predict(X_test)
```

In [58]:

```
RMSEknn = mse(y_test, pred, squared = False)
RMSEknn
```

Out[58]:

```
14254.017140610631
```

In [59]:

```
knnscore = knn.score(X_test, y_test)
knnscore
```

Out[59]:

```
0.9589643655222844
```

In [60]:

```
#using CV grid search to tune the hyperparameters
```

In [61]:

```
from sklearn.model_selection import GridSearchCV
```

In [62]:

```
params = {'n_neighbors':[1,2,3,4,5,6,7,8,9]}
```

In [63]:

```
model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

Out[63]:

```
{'n_neighbors': 2}
```

In [64]:

```
knn1 = KNeighborsRegressor(n_neighbors = 2)
```

In [65]:

```
knn1.fit(X_train, y_train)
```

Out[65]:

```
KNeighborsRegressor(n_neighbors=2)
```

In [66]:

```
pred_knn = knn1.predict(X_test)
```

In [67]:

```
knn_score = knn1.score(X_test, y_test)
print("Accuracy score for KNN:",knn_score)
```

```
Accuracy score for KNN: 0.9794981148324132
```

In [68]:

```
RMSE_knn = mse(y_test, pred_knn, squared = False)
print("Root Mean Squared Error for KNN:",RMSE_knn)
```

```
Root Mean Squared Error for KNN: 10075.198207417992
```

In [69]:

```
EPSILON =  1e-10
rmspe_knn = (np.sqrt(np.mean(np.square((y_test - pred_knn) / (y_test + EPSILON))))) * 100
print("Root Mean Squared Percentage Error for KNN:",rmspe_knn)
```

```
Root Mean Squared Percentage Error for KNN: 7.099692254357483
```

In [70]:

```
mae_knn = mae(pred_knn, y_test)
print("Mean Absolute Error for knn: ", mae_knn)
```

```
Mean Absolute Error for knn:  2504.348176927675
```

In [71]:

```
mape_knn = (np.mean(np.abs(y_test-pred_knn)/y_test)) * 100
print("Mean Absolute Percentage Error for knn: ", mape_knn)
```

```
Mean Absolute Percentage Error for knn:  3.1590220913873326
```

In [72]:

```
r2_knn = r2_score(pred_knn, y_test)
print("r2 score for knn: ", r2_knn)
```

```
r2 score for knn:  0.9780615277686634
```

In [73]:

```
data_knn = pd.DataFrame({'Actual':y_test, 'Predicted':pred_knn})
data_knn
```

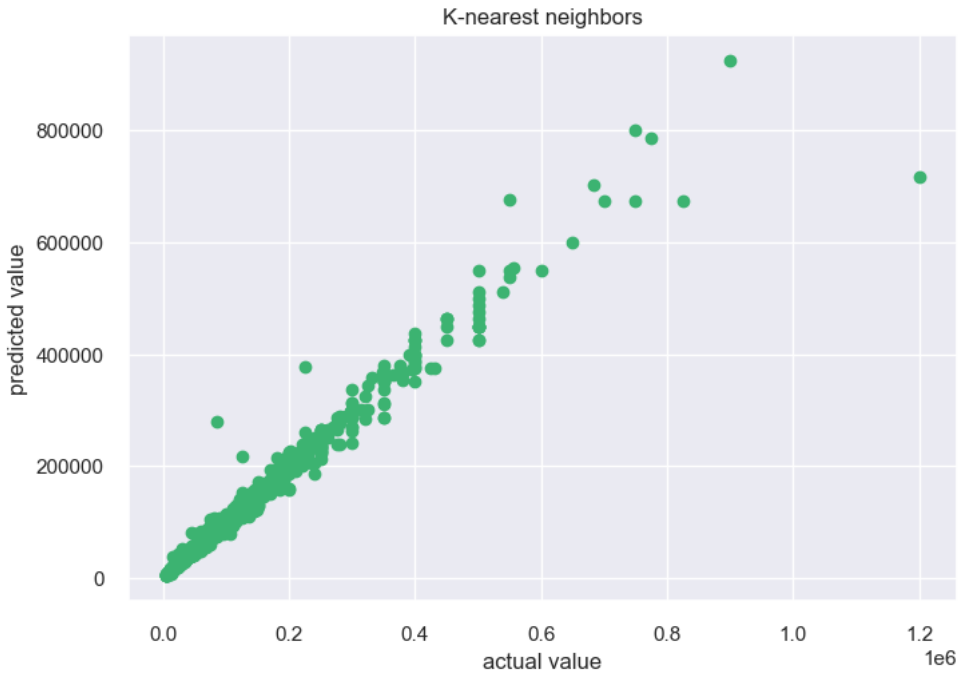
Out[73]:

	Actual	Predicted
33379	250000	235000.0
27648	55000	55000.0
26647	45600	45250.0
12216	13500	15500.0
31710	120000	122500.0
...	...	...
7245	30000	30000.0
17745	50000	50000.0
5144	21000	20000.0
16266	50000	50000.0
26556	40000	40000.0

5019 rows × 2 columns

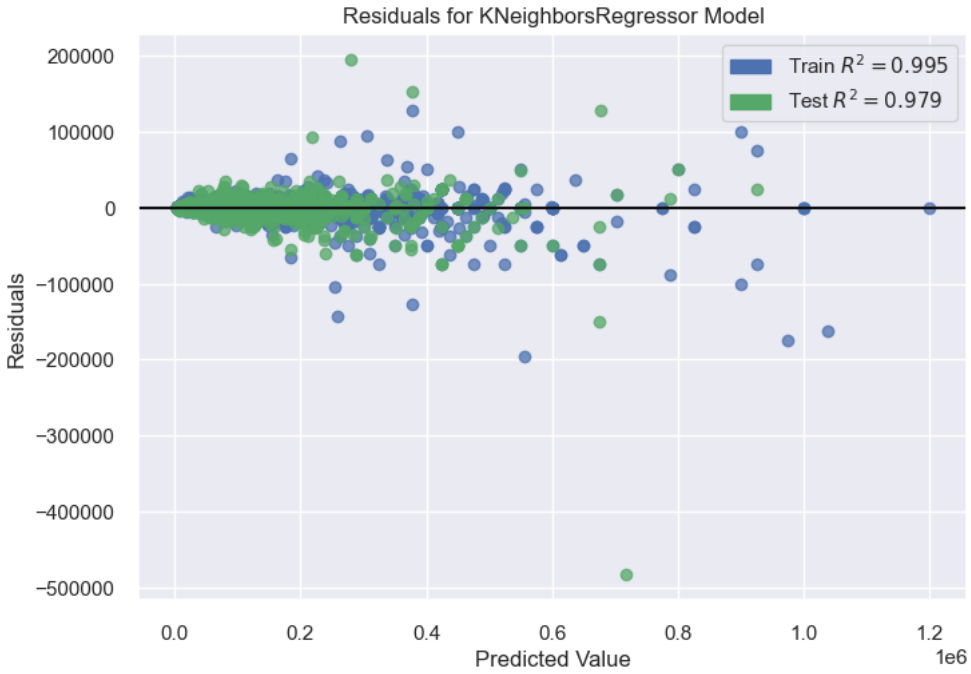
In [74]:

```
plt.scatter(y_test, pred_knn, color='mediumseagreen');
plt.title("K-nearest neighbors")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```



In [75]:

```
visualizer = ResidualsPlot(knn1, hist=False)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.show();
```



Decision Tree

In [76]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [77]:

```
tree = DecisionTreeRegressor(criterion='squared_error')
```

In [78]:

```
tree.fit(X_train, y_train)
```

Out[78]:

```
DecisionTreeRegressor()
```

In [79]:

```
pred_tree = tree.predict(X_test)
```

In [80]:

```
dec_tree_score = tree.score(X_test, y_test)
dec_tree_score
```

Out[80]:

```
0.981431121778113
```

In [81]:

```
RMSE_dec_tree = mse(y_test, pred_tree ,squared = False)
print("Root Mean Squared Error for decision tree:",RMSE_dec_tree)
```

```
Root Mean Squared Error for decision tree: 9588.47487023712
```

In [82]:

```
rmspe_tree = (np.sqrt(np.mean(np.square((y_test - pred_tree) / (y_test + EPSILON))))) * 100
print("Root Mean Squared Percentage Error for decision tree:",rmspe_tree)
```

```
Root Mean Squared Percentage Error for decision tree: 3.713248636653081
```

In [83]:

```
mae_tree = mae(pred, y_test)
print("Mean Absolute Error for decision tree:",mae_tree)
```

```
Mean Absolute Error for decision tree: 2916.4846782227532
```

In [84]:

```
mape_tree = (np.mean(np.abs(y_test-pred_tree)/y_test)) * 100
print("Mean Absolute Percentage Error for decision tree:",mape_tree)
```

```
Mean Absolute Percentage Error for decision tree: 1.189318136717846
```

In [85]:

```
r2_tree = r2_score(pred, y_test)
print("r2 score for decision tree:",r2_tree)
```

```
r2 score for decision tree: 0.9540652841875253
```

In [86]:

```
data_tree = pd.DataFrame({'Actual':y_test, 'Predicted':pred_tree})
data_tree
```

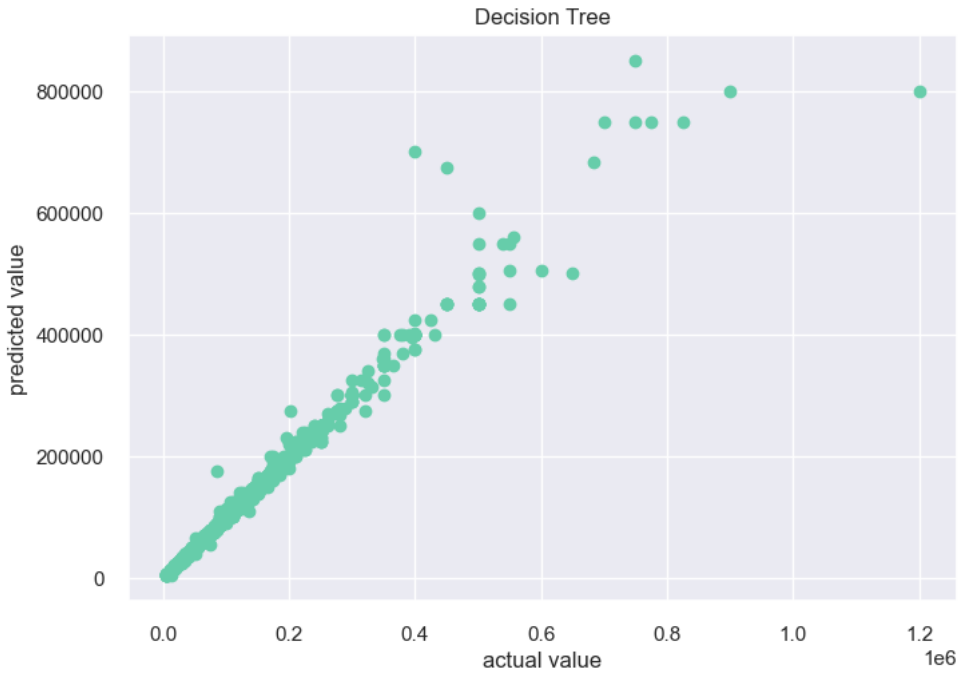
Out[86]:

	Actual	Predicted
33379	250000	250000.0
27648	55000	55000.0
26647	45600	45500.0
12216	13500	12000.0
31710	120000	120000.0
...	...	...
7245	30000	30000.0
17745	50000	50000.0
5144	21000	21000.0
16266	50000	50000.0
26556	40000	40000.0

5019 rows × 2 columns

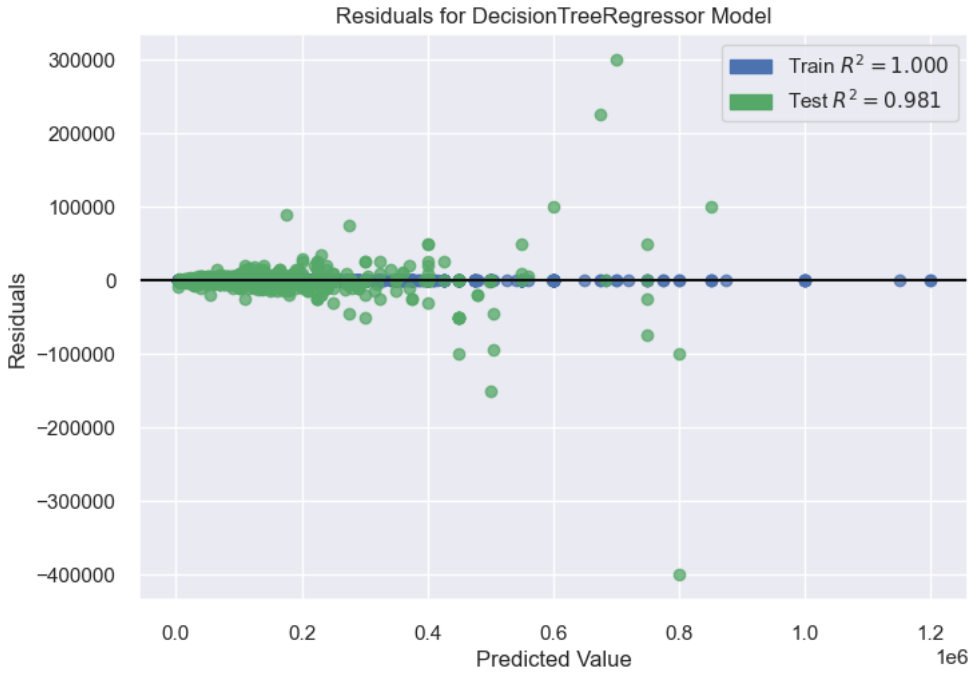
In [87]:

```
plt.scatter(y_test, pred_tree, color='mediumaquamarine');
plt.title("Decision Tree")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```



In [88]:

```
visualizer = ResidualsPlot(tree, hist=False)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.show();
```



Random forest

In [89]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [90]:

```
forest = RandomForestRegressor(n_estimators = 10, random_state = 0)
```

In [91]:

```
forest.fit(X_train, y_train)
```

Out[91]:

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

In [92]:

```
pred_forest = forest.predict(X_test)
```

In [93]:

```
forest_score = forest.score(X_test, y_test)
forest_score
```

Out[93]:

```
0.9884626419211885
```

In [94]:

```
RMSE_forest = mse(y_test, pred_forest ,squared = False)
print("Root Mean Squared Error for random forest:",RMSE_forest)
```

```
Root Mean Squared Error for random forest: 7558.05218586742
```

In [95]:

```
rmspe_forest = (np.sqrt(np.mean(np.square((y_test - pred_forest) / (y_test + EPSILON))))) * 100
print("Root Mean Squared Percentage Error for random forest:",rmspe_forest)
```

```
Root Mean Squared Percentage Error for random forest: 3.3587460147853507
```

In [96]:

```
mae_forest = mae(y_test, pred_forest)
print("Mean Absolute Error for random forest:",mae_forest)
```

```
Mean Absolute Error for random forest: 1060.2390549246197
```

In [97]:

```
mape_forest = (np.mean(np.abs(y_test-pred_forest)/y_test)) * 100
print("Mean Absolute Percentage Error for random forest:",mape_forest)
```

```
Mean Absolute Percentage Error for random forest: 0.9844430811517859
```

In [98]:

```
r2_forest = r2_score(pred_forest, y_test)
print("r2 score for random forest:",r2_forest)
```

```
r2 score for random forest: 0.9879143238134978
```

In [99]:

```
data_forest = pd.DataFrame({'Actual':y_test, 'Predicted': pred_forest})
data_forest
```

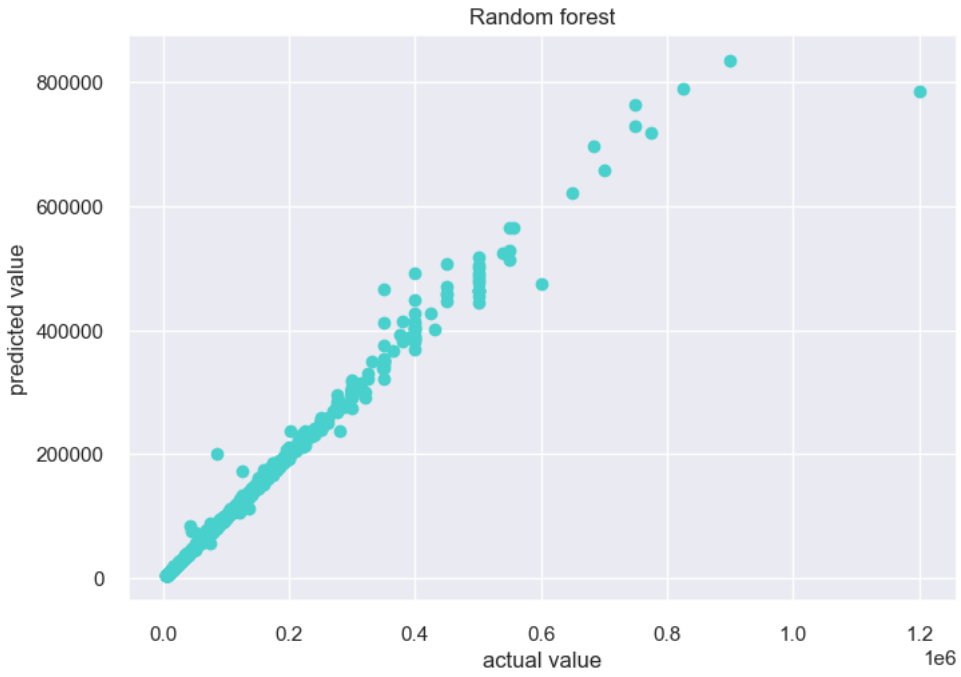
Out[99]:

	Actual	Predicted
33379	250000	256500.0
27648	55000	54800.0
26647	45600	45600.0
12216	13500	14499.9
31710	120000	120000.0
...	...	...
7245	30000	30000.0
17745	50000	50000.0
5144	21000	21200.0
16266	50000	49950.0
26556	40000	40000.0

5019 rows × 2 columns

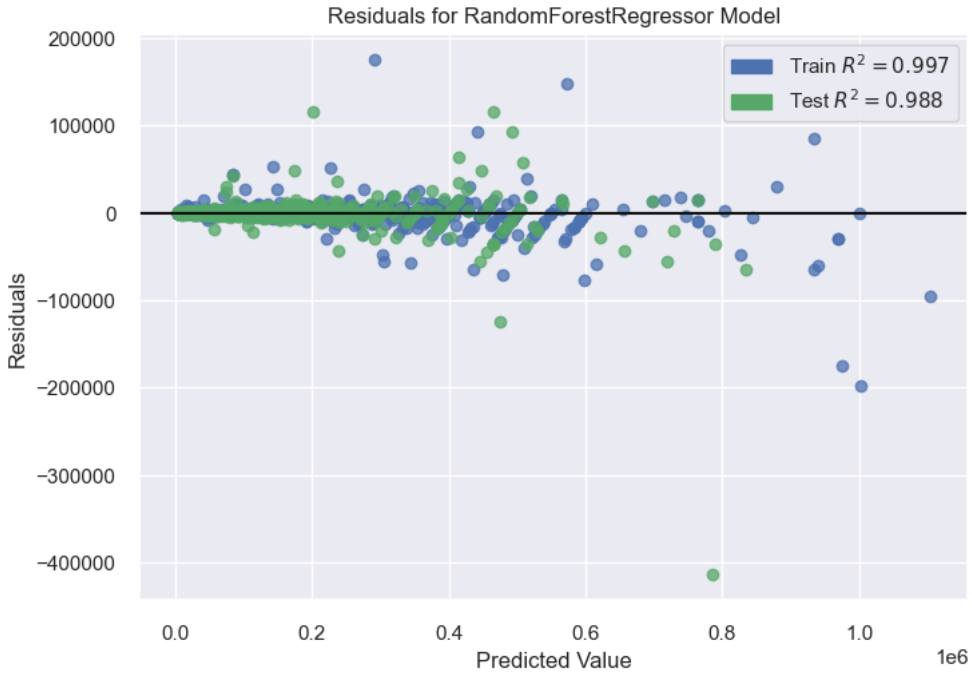
In [100]:

```
plt.scatter(x = y_test , y = pred_forest , color='mediumturquoise');
plt.title("Random forest")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```



In [101]:

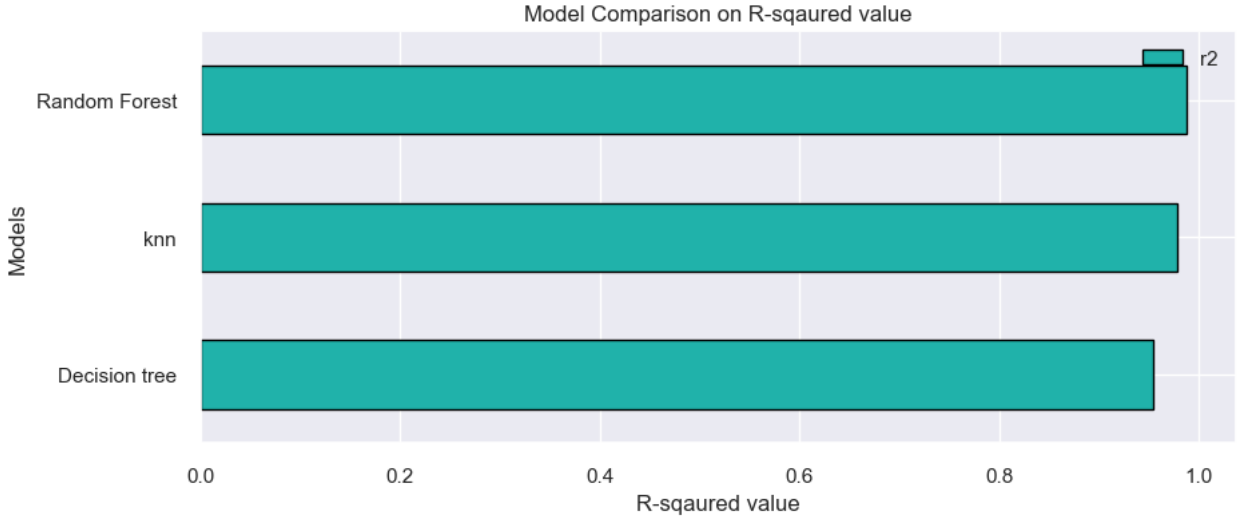
```
visualizer = ResidualsPlot(forest, hist=False)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.show();
```



Model Comparison

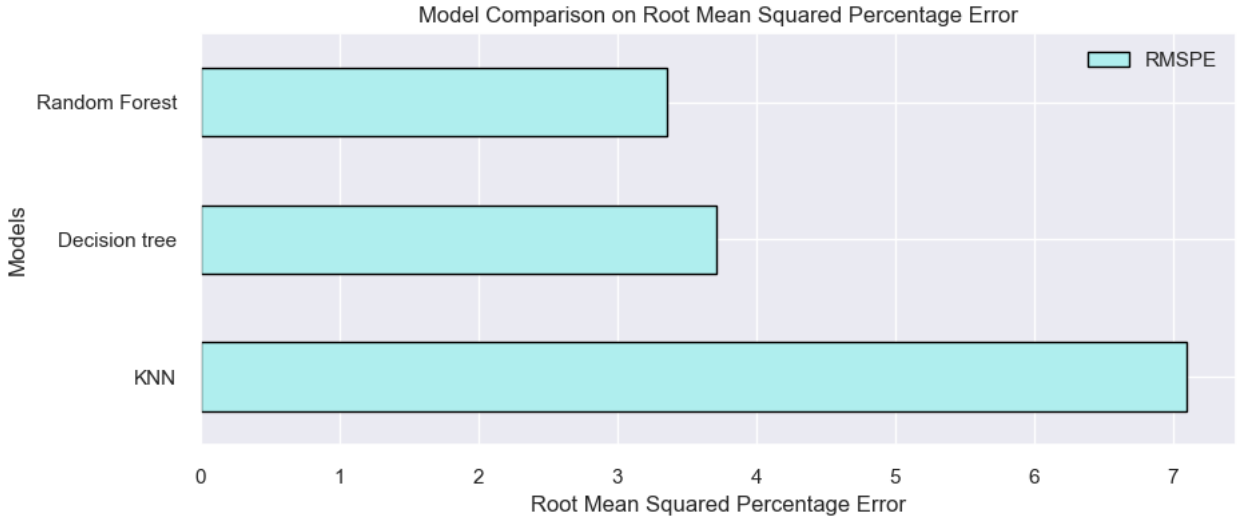
In [102]:

```
model_comparison_r2 = pd.DataFrame({'Models': [ 'knn', 'Decision tree',
                                               'Random Forest'],
                                    'r2': [r2_knn, r2_tree, r2_forest]})

model_comparison_r2.sort_values('r2', ascending = True).plot(x = 'Models',
                                                             y = 'r2', kind = 'barh', color = 'lightseagreen',
                                                             edgecolor = 'black', figsize = (10,4))
plt.xlabel('R-sqaured value')
plt.title('Model Comparison on R-sqaured value')
plt.show()
```



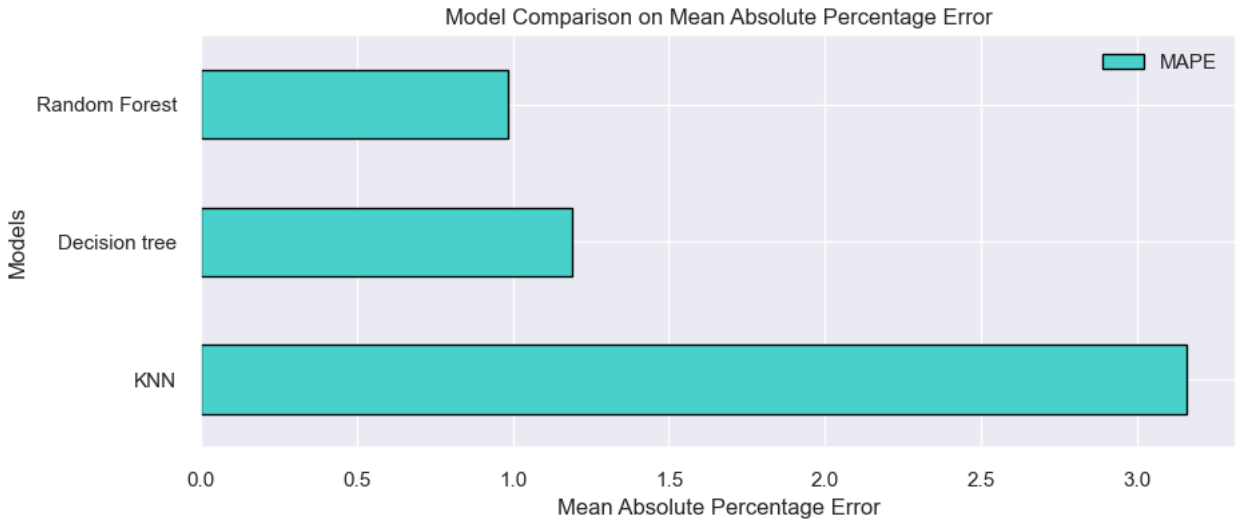
In [103]:

```
model_comparison_rmspe = pd.DataFrame({'Models': [ 'KNN', 'Decision tree',
                                    'Random Forest'],
                                'RMSPE': [rmspe_knn, rmspe_tree, rmspe_forest]})

model_comparison_rmspe.sort_values('RMSPE', ascending = False).plot(x = 'Models',
                                                        y = 'RMSPE', kind = 'barh', color = 'paleturquoise',
                                                        edgecolor = 'black', figsize = (10,4))
plt.xlabel('Root Mean Squared Percentage Error')
plt.title('Model Comparison on Root Mean Squared Percentage Error')
plt.show()
```



In [104]:

```
model_comparison_mape = pd.DataFrame({'Models': [ 'KNN', 'Decision tree',
                                    'Random Forest'],
                                'MAPE': [mape_knn, mape_tree, mape_forest]})

model_comparison_mape.sort_values('MAPE', ascending = False).plot(x = 'Models',
                                                        y = 'MAPE', kind = 'barh', color = 'mediumturquoise',
                                                        edgecolor = 'black', figsize = (10,4))
plt.xlabel('Mean Absolute Percentage Error')
plt.title('Model Comparison on Mean Absolute Percentage Error')
plt.show()
```



In [ ]: