Aim: Predict the rental price of houses based on various features such as locality, number of bedrooms, floor number, etc.

The project utilizes machine learning models (k-NN, Decision Tree, Random Forest) trained on a dataset of rental properties of Mumbai city to predict the rental price for new properties.

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)
```

In [2]:
```
df = pd.read_csv('data_rent.csv')
df.head()
```

Out[2]:

	area	bathroom_num	bedroom_num	city	desc	dev_n
0	350.0	2.0	1	Mumbai	2 Bath,Unfurnished,East facing The project has...	
1	652.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,East facing A 1BHK apart...	Ve G
2	635.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,4 floor,West facing A be...	Aga G
3	540.0	2.0	1	Mumbai	2 Bath,Semi-Furnished,East facing Essential Se...	
4	625.0	1.0	1	Mumbai	1 Bath,Furnished,2 floor,North facing 24 hours...	Millenı G

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_st
ring', '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_n
count	33572.000000	34334.000000	34348.000000	31488.000000	31567.000(
mean	1177.387704	2.199278	2.076686	17.403551	8.5778
std	682.924385	0.880150	0.899821	13.996063	7.7709
min	10.000000	1.000000	1.000000	2.000000	-2.000(
25%	690.000000	2.000000	1.000000	7.000000	3.000(
50%	1040.000000	2.000000	2.000000	14.000000	6.000(
75%	1400.000000	3.000000	3.000000	22.000000	11.000(
max	9500.000000	8.000000	5.000000	120.000000	95.000(

In [6]:
```
df1 = df.copy()
```

In [7]:
```
(df1.city != 'Mumbai').unique()
```

Out[7]:
```
array([False])
```

In [8]:
```
df1.trans.unique()
```

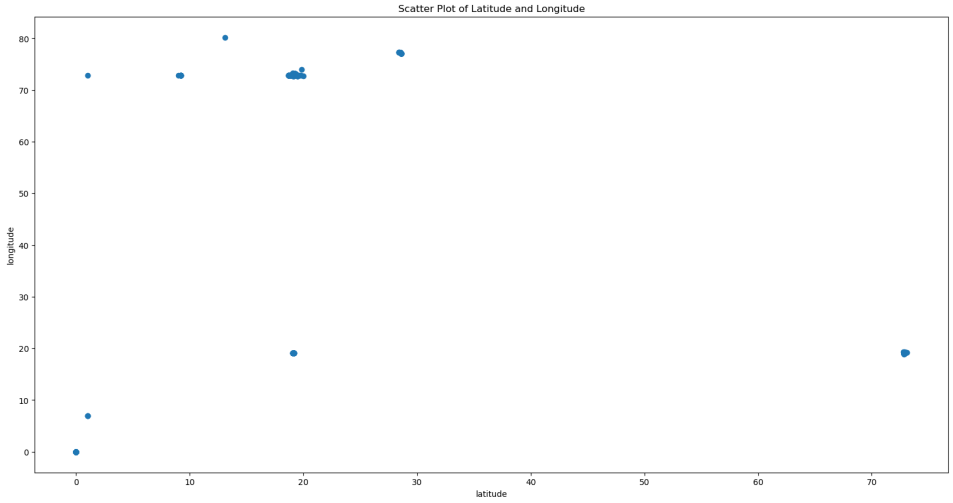
Out[8]:
```
array(['Rent', nan], dtype=object)
```

In [9]:
```
df1.drop(['desc','dev_name','id','id_string', 'post_date', 'pos
         axis = 1, inplace = True)
df1.head()
```

Out[9]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing
0	350.0	2.0	1	NaN	NaN	Unfurnished
1	652.0	2.0	1	NaN	NaN	Semi-Furnished
2	635.0	2.0	1	7.0	4.0	Semi-Furnished
3	540.0	2.0	1	NaN	NaN	Semi-Furnished
4	625.0	1.0	1	7.0	2.0	Furnished

In [10]:
```
plt.scatter(df1['latitude'], df1['longitude'])
plt.title('Scatter Plot of Latitude and Longitude')
plt.xlabel("latitude")
plt.ylabel("longitude")
plt.show()
```



latitude-longitude of mumbai- 19.0760° N, 72.8777° E

In the above plot, we can see that the longitude and latitude are all over the place. Hence we can get rid of these column since they does not seem reliable.

In [11]:
```
df1.drop(['latitude', 'longitude'], axis = 1, inplace = True)
df2 = df1.copy()
df2.head(3)
```

Out[11]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing
0	350.0	2.0	1	NaN	NaN	Unfurnished
1	652.0	2.0	1	NaN	NaN	Semi-Furnished
2	635.0	2.0	1	7.0	4.0	Semi-Furnished

Null values handling

In [12]:
```
df2.isna().sum()/len(df2) * 100
```

Out[12]:
```
area            2.259229
bathroom_num    0.040759
bedroom_num     0.000000
floor_count     8.326540
floor_num       8.096541
furnishing      0.029114
locality        0.605567
price           0.000000
type            0.000000
dtype: float64
```

In [13]:
```
df2['floor_num'].fillna(df2['floor_num'].mode()[0], inplace = T
df2['floor_count'].fillna(df2['floor_count'].mode()[0], inplace
df2['area'].fillna(df2['area'].mean(), inplace = True)
```

In [14]:
```
df2.isna().sum()/len(df2) * 100
```

Out[14]:
```
area            0.000000
bathroom_num    0.040759
bedroom_num     0.000000
floor_count     0.000000
floor_num       0.000000
furnishing      0.029114
locality        0.605567
price           0.000000
type            0.000000
dtype: float64
```

In [15]:
```
df2.dropna(inplace=True)
```

In [16]:
```
df2.shape
```

Out[16]:
```
(34117, 9)
```

In [17]:
```
df2.isnull().sum()
```

Out[17]:
```
area            0
bathroom_num    0
bedroom_num     0
floor_count     0
floor_num       0
furnishing      0
locality        0
price           0
type            0
dtype: int64
```

In [18]:
```
for i in df2.columns:
    print("Type of", i, "is: ", df2[i].dtype)
```

```
Type of area is:  float64
Type of bathroom_num is:  float64
Type of bedroom_num is:  int64
Type of floor_count is:  float64
Type of floor_num is:  float64
Type of furnishing is:  object
Type of locality is:  object
Type of price is:  int64
Type of type is:  object
```

In [19]:
```
df2['bathroom_num'] = df2['bathroom_num'].astype(int)
df2['floor_count'] = df2['floor_count'].astype(int)
df2['floor_num'] = df2['floor_num'].astype(int)
df2['price'] = df2['price'].astype(float)
```

In [20]:
```
df2.head()
```

Out[20]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing
0	350.0	2	1	7	5	Unfurnished
1	652.0	2	1	7	5	Semi-Furnished
2	635.0	2	1	7	4	Semi-Furnished
3	540.0	2	1	7	5	Semi-Furnished
4	625.0	1	1	7	2	Furnished

Feature Engineering

In [21]:
```
df2['price_per_area'] = df2['price']/df2['area']
df2.head()
```

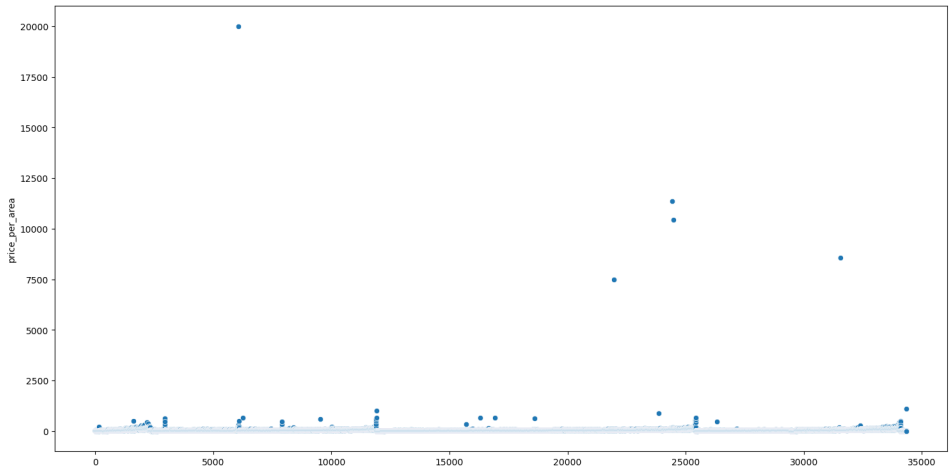
Out[21]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing
0	350.0	2	1	7	5	Unfurnished
1	652.0	2	1	7	5	Semi-Furnished
2	635.0	2	1	7	4	Semi-Furnished
3	540.0	2	1	7	5	Semi-Furnished
4	625.0	1	1	7	2	Furnished

In [22]:
```
df2.price_per_area.describe()
```

Out[22]:
```
count    34117.000000
mean        58.655956
std        153.334619
min          2.760348
25%         37.692308
50%         49.214660
75%         67.796610
max      20000.000000
Name: price_per_area, dtype: float64
```

In [23]:
```
plt.figure(figsize=(18,9))
sns.scatterplot(df2['price_per_area']);
```



In [24]:
```
df2[df2.price_per_area > 2500]
```

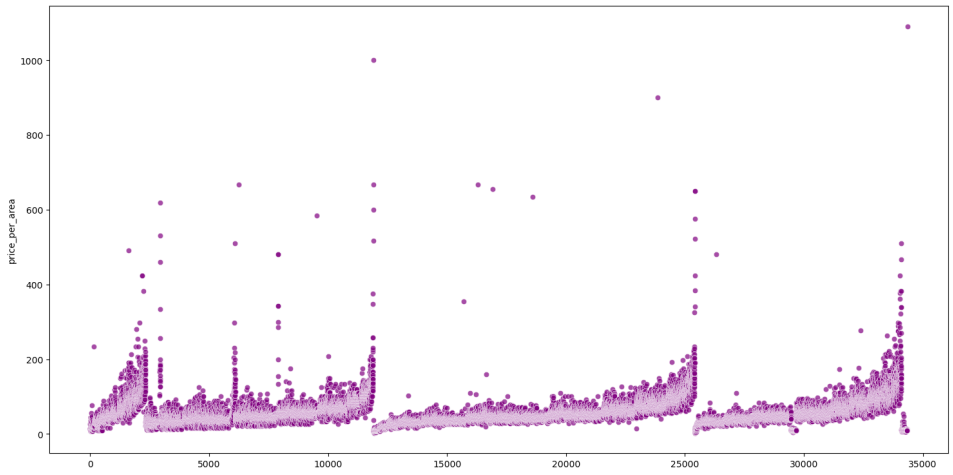
Out[24]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishi
6055	20.0	5	5	12	6	Unfurnish
21953	10.0	3	2	22	5	Furnish
24435	11.0	2	2	30	25	Furnish
24477	11.0	2	2	25	17	Ser Furnish
31555	14.0	3	3	34	22	Furnish

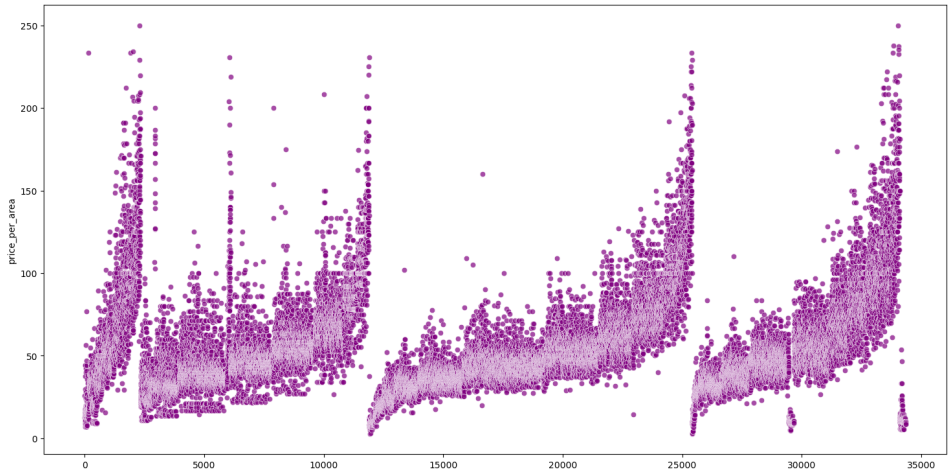
In [25]:
```
df2 = df2[~(df2.price_per_area > 2500)]
df2.price_per_area.describe()
```

Out[25]:
```
count    34112.000000
mean        56.967509
std         34.519307
min          2.760348
25%         37.692308
50%         49.214660
75%         67.757453
max       1090.909091
Name: price_per_area, dtype: float64
```

In [26]:
```
plt.figure(figsize=(18,9))
sns.scatterplot(df2['price_per_area'], color = 'purple', alpha
```



In [27]:
```
df2 = df2[~(df2.price_per_area > 250)]
plt.figure(figsize=(18,9))
sns.scatterplot(df2['price_per_area'], color = 'purple', alpha
```



In [28]:
```
df2 = df2[~(df2.area < 150)]
```

In [29]:
```
df2.describe()
```

Out[29]:

	area	bathroom_num	bedroom_num	floor_count	floor_n
count	34040.000000	34040.000000	34040.000000	34040.000000	34040.000(
mean	1180.158854	2.200823	2.078261	16.572797	8.3053
std	674.909906	0.878653	0.898529	13.713800	7.5159
min	150.000000	1.000000	1.000000	2.000000	-2.000(
25%	700.000000	2.000000	1.000000	7.000000	4.000(
50%	1050.000000	2.000000	2.000000	12.000000	6.000(
75%	1400.000000	3.000000	3.000000	22.000000	11.000(
max	9500.000000	8.000000	5.000000	120.000000	95.000(

In [30]:
```
df3 = df2.copy()
print(df3.shape)
df3.head()
```

```
(34040, 10)
```

Out[30]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	furnishing
0	350.0	2	1	7	5	Unfurnished
1	652.0	2	1	7	5	Semi-Furnished
2	635.0	2	1	7	4	Semi-Furnished
3	540.0	2	1	7	5	Semi-Furnished
4	625.0	1	1	7	2	Furnished

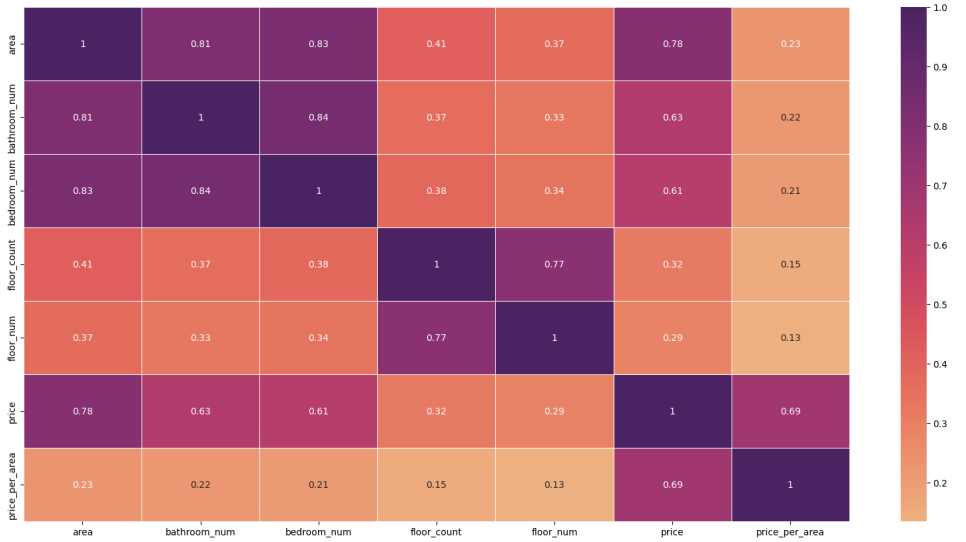
In [31]:
```
num_df = df3.select_dtypes(include = ['number'])
num_df
```

Out[31]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	price
0	350.0	2	1	7	5	9000.0
1	652.0	2	1	7	5	8060.0
2	635.0	2	1	7	4	8000.0
3	540.0	2	1	7	5	8000.0
4	625.0	1	1	7	2	9000.0
...	...	...	...	...	...	...
34342	680.0	2	1	15	8	6500.0
34343	700.0	2	1	7	5	7000.0
34344	750.0	2	1	12	7	6500.0
34345	700.0	2	1	12	4	6500.0
34346	750.0	2	1	12	4	6500.0

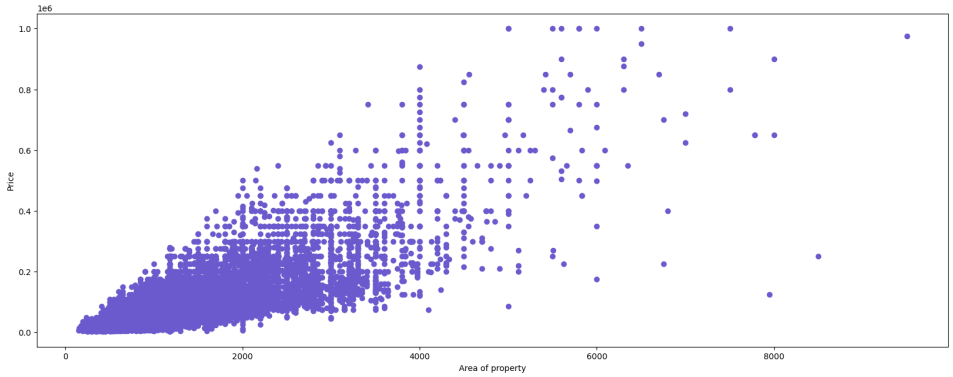
34040 rows × 7 columns

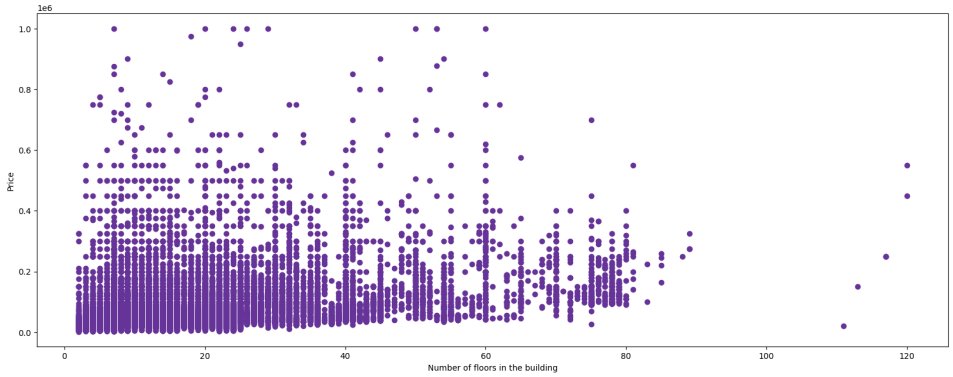
In [32]:
```
matrix = num_df.corr()
sns.heatmap(matrix, cmap=sns.color_palette("flare", as_cmap=Tru
```

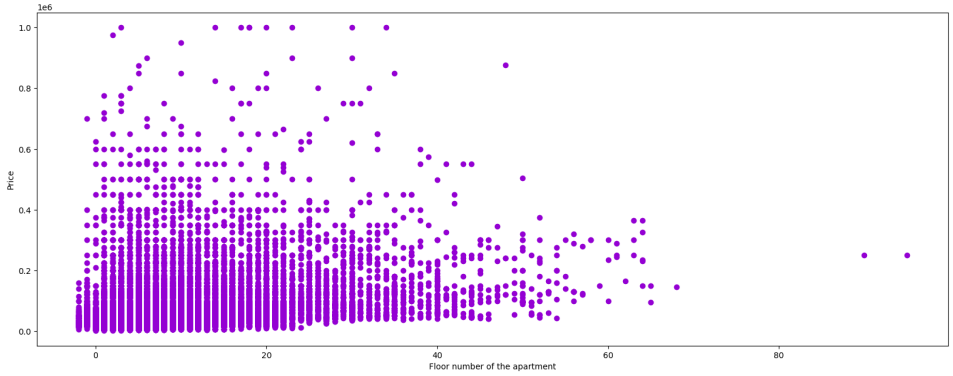


In [33]:

```
#outliers

fig, ax = plt.subplots(3, figsize=(20, 25))
ax[0].scatter(x = df3['area'], y = df3['price'], color = 'slate
ax[0].set_xlabel("Area of property")
ax[0].set_ylabel("Price")

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

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

plt.show()
```







In [34]:
```
df4 = df3.copy()
```

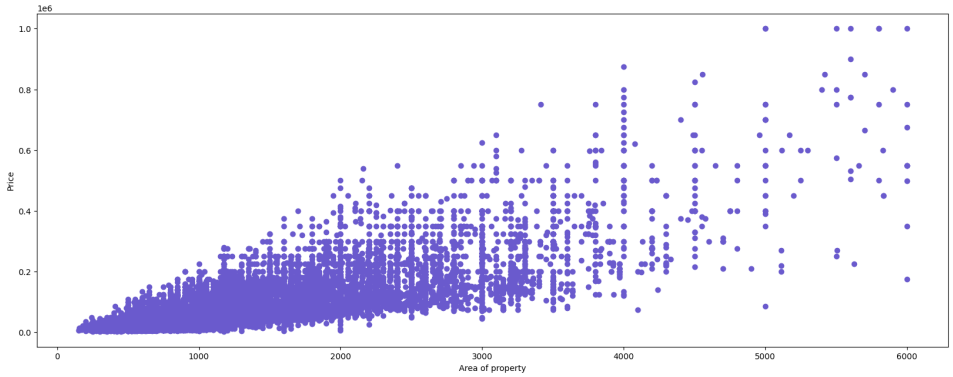
In [35]:
```
#outlier removal

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

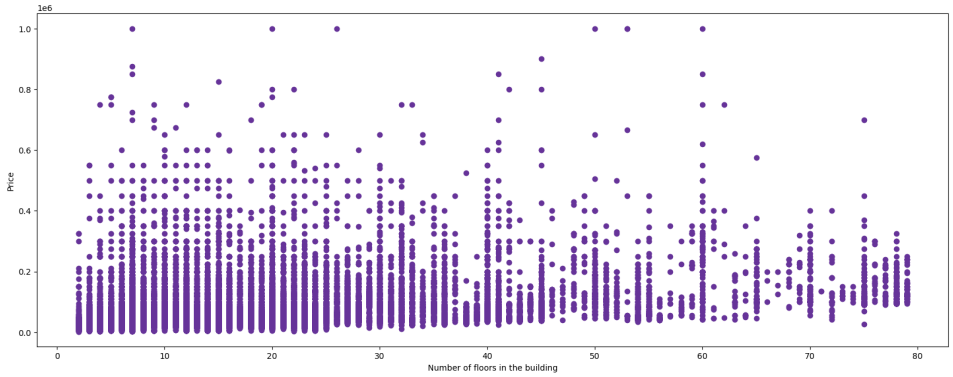
In [36]:
```
df3.shape
```

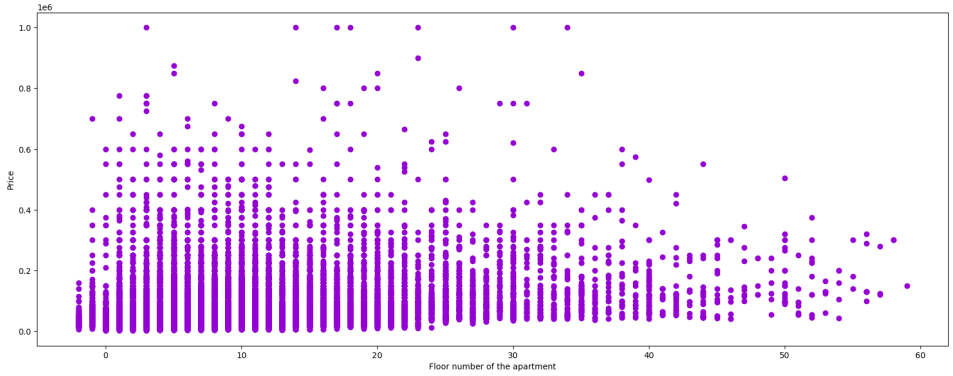
Out[36]: (33962, 10)

In [37]:

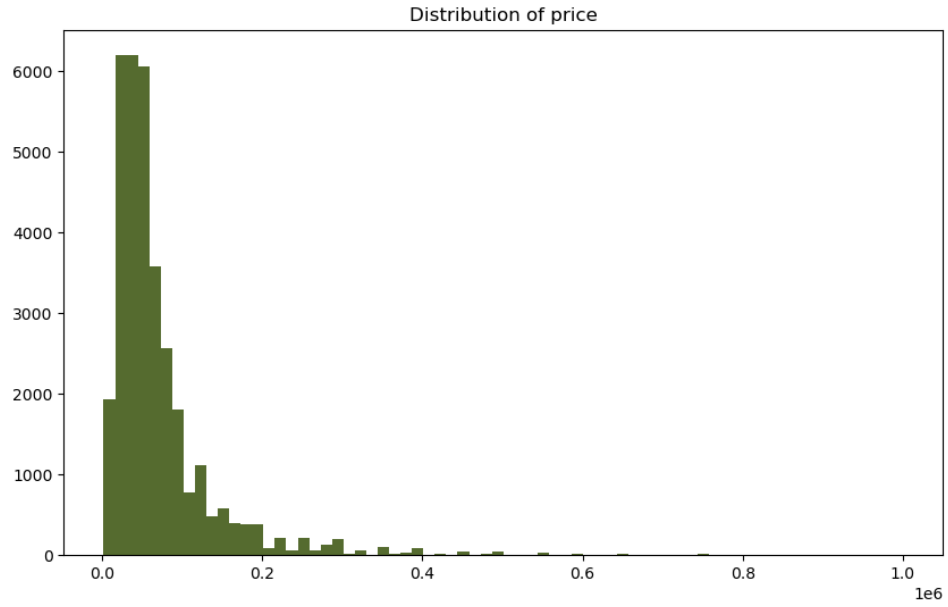
```
# scatter plot after outlier removal

fig, ax = plt.subplots(3, figsize=(20, 25))
ax[0].scatter(x = df3['area'], y = df3['price'], color = 'slate
ax[0].set_xlabel("Area of property")
ax[0].set_ylabel("Price")

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

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

plt.show()
```


1e6
1.0
0.8
0.6
0.4
0.2
0.0
Price
0
1000
2000
3000
4000
5000
6000
Area of property

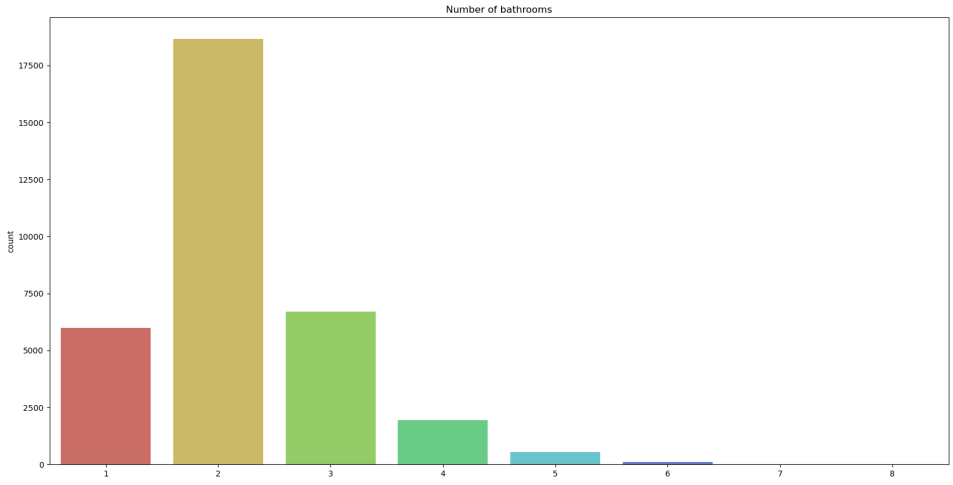

1e6
1.0
0.8
0.6
0.4
0.2
0.0
Price
0
10
20
30
40
50
60
70
80
Number of floors in the building


1e6
1.0
0.8
0.6
0.4
0.2
0.0
Price
0
10
20
30
40
50
60
Floor number of the apartment

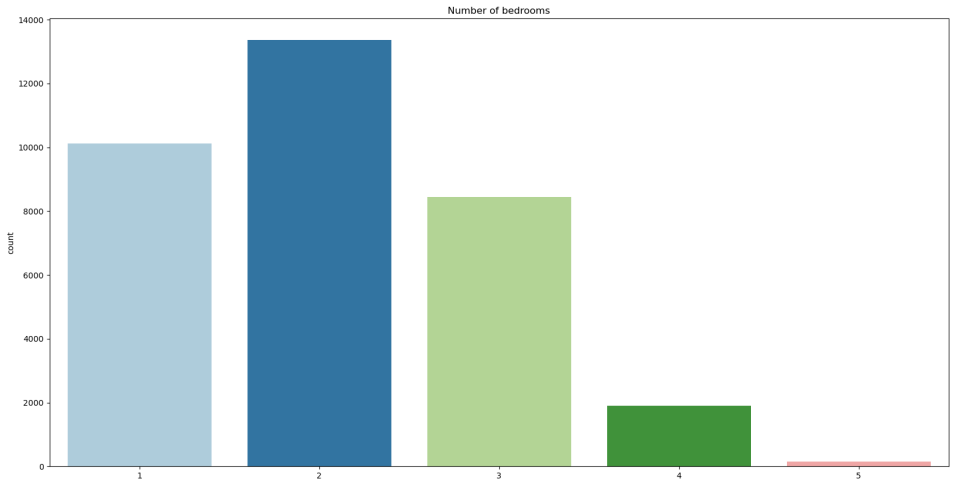
In [38]:
```
df3['price'].hist(bins = 70, grid = False, color = 'darkolivegr
plt.title("Distribution of price")
plt.show()
```



In [39]:
```
sns.countplot(x = 'bathroom_num', data=df3, palette='hls');
plt.title("Number of bathrooms")
plt.xlabel("")
plt.show()
```



In [40]:
```
sns.countplot(x = 'bedroom_num', data=df3, palette='Paired')
plt.title("Number of bedrooms")
plt.xlabel("")
plt.show()
```



Most of the houses have 2 bedrooms and bathrooms

In [41]:
```
df5 = df4.copy()
```

In [42]:
```
df5['locality'].value_counts()
```

Out[42]:
```
locality
Powai                  1479
Chembur                1418
Andheri West           1416
Andheri East           1114
Worli                  1035
                       ...
Nehru Nagar - Juhu        1
Ganesh Nagar              1
Virat Nagar               1
Anushakti Colony          1
Nimoni Baug               1
Name: count, Length: 825, dtype: int64
```

Since there are a lot of localities, we can club together the localities consisting of less number of properties.

So, all the localities with less than or equal to 10 properties are clubbed together as 'Others'.

In [43]:
```
loc_count = df5['locality'].value_counts()
loc_count_less_than_10 = loc_count[loc_count <= 10]
loc_count_less_than_10
```

Out[43]:
```
locality
Dahisar               10
Mehboob Studio        10
Upper Worli           10
Virar East            10
Juhu Beach Area       10
                      ..
Nehru Nagar - Juhu     1
Ganesh Nagar           1
Virat Nagar            1
Anushakti Colony       1
Nimoni Baug            1
Name: count, Length: 548, dtype: int64
```

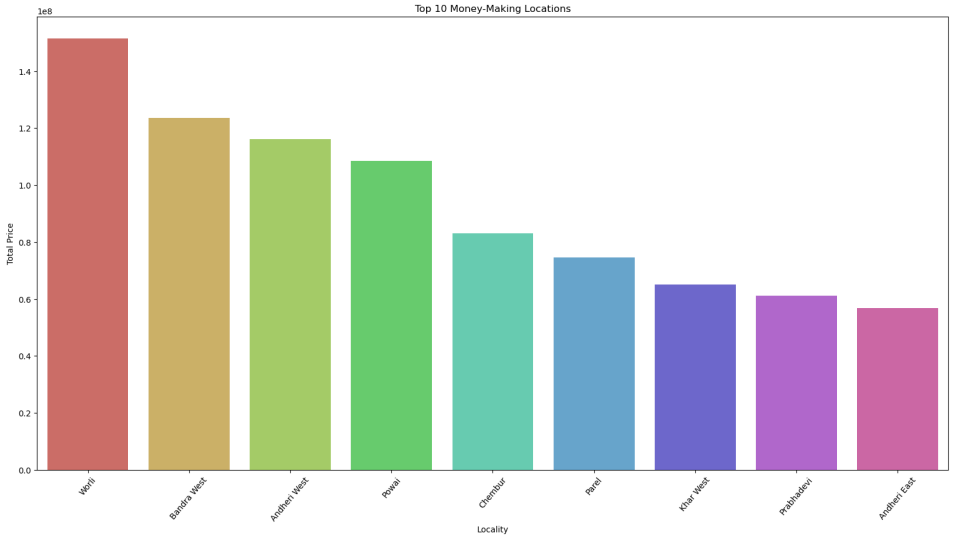
In [44]:
```
df5['locality'] = df5['locality'].apply(lambda x : 'Others' if
```

In [45]:
```
df5['locality'].value_counts()
```

Out[45]:
```
locality
Others            1703
Powai             1479
Chembur           1418
Andheri West      1416
Andheri East      1114
                  ...
Chinchpokli         11
Garodia Nagar       11
Shimpoli            11
Premier Colony      11
Piramal Nagar       11
Name: count, Length: 278, dtype: int64
```

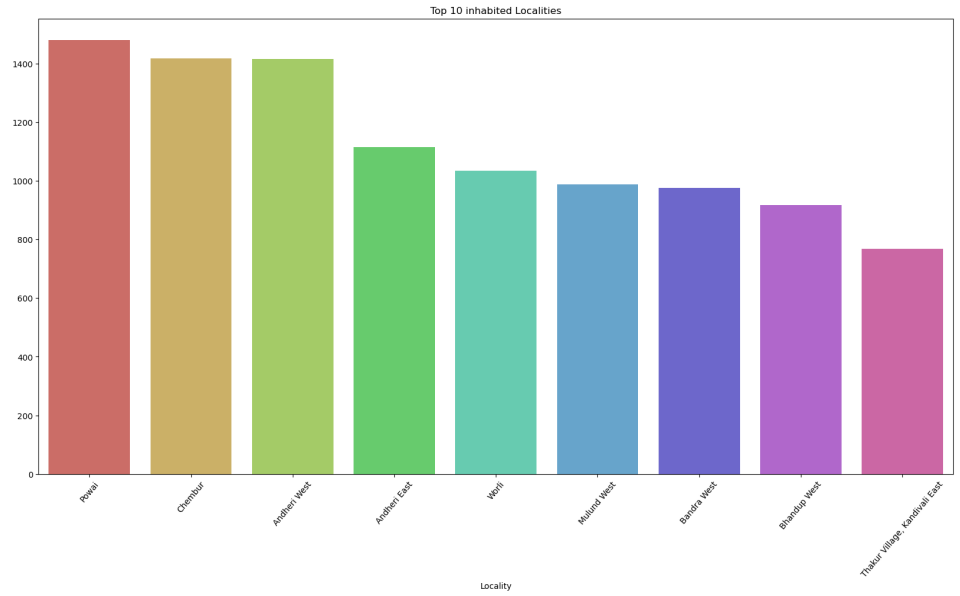
In [46]:
```
df6 = df5.copy()
```

In [47]:
```
grp_loc = df6.groupby('locality')['price'].sum().reset_index()
sort_loc = grp_loc.sort_values(by='price', ascending=False)
top_10_localities = sort_loc.head(10)
top_10_localities = top_10_localities[top_10_localities['locali

# Create a bar plot
sns.barplot(x='locality', y='price', data=top_10_localities, pa

# Adjust plot aesthetics
plt.title('Top 10 Money-Making Locations')
plt.xlabel('Locality')
plt.ylabel('Total Price')
plt.xticks(rotation=50)

# Show the plot
plt.show()
```



By 'largest pool of money', I mean the areas which recives highest amount either because of large number of tenants or because of expensive properties

In [48]:
```
loc = df6['locality'].value_counts().nlargest(10)
loc = loc[loc.index != 'Others']
# Create a bar plot
sns.barplot(x = loc.index, y = loc.values, palette = 'hls')

# Adjust plot aesthetics
plt.title('Top 10 inhabited Localities')
plt.xlabel('Locality')
plt.xticks(rotation=50)

# Show the plot
plt.show()
```



One Hot Encoding

In [49]:
```
from sklearn.preprocessing import OneHotEncoder, Normalizer

cat_col = ['furnishing', 'locality', 'type']

num_col = ['bathroom_num', 'bedroom_num', 'floor_count', 'floor

ohe = OneHotEncoder(sparse_output = False, drop = 'first')
df7 = pd.get_dummies(df6, columns = cat_col, drop_first = True)

df7.head()
```

Out[49]:

	area	bathroom_num	bedroom_num	floor_count	floor_num	price	price
0	350.0	2	1	7	5	9000.0	
1	652.0	2	1	7	5	8060.0	
2	635.0	2	1	7	4	8000.0	
3	540.0	2	1	7	5	8000.0	
4	625.0	1	1	7	2	9000.0	

5 rows × 291 columns

In [50]:
```
X = df7.drop(['price'], axis = 1)
y = df7.price
```

In [51]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

In [52]:
```
print("Shape of X_train:", X_train.shape, "\nShape of y_train:"
      X_test.shape, "\nShape of y_test:", y_test.shape)
```

```
Shape of X_train: (27232, 290)
Shape of y_train: (27232,)
Shape of X_test: (6808, 290)
Shape of y_test: (6808,)
```

In [53]:
```
from sklearn.preprocessing import MinMaxScaler

scaler_X = MinMaxScaler()
X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)

# Min-Max scaling for y_train and y_test using the same scaler
scaler_y = MinMaxScaler()
y_train = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test = scaler_y.transform(y_test.values.reshape(-1, 1))
```

kNN

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)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

In [56]:
```
knnscore = knn.score(X_test, y_test)
knnscore
```

Out[56]: 0.8603567543280682

In [57]:
```
RMSEknn = mse(y_test, pred, squared = False)
RMSEknn
```

Out[57]: 0.0281808423626602

In [58]:
```
#using CV grid search to tune the hyperparameters
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[1,2,3,4,5,6,7,8,9]}
model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

Out[58]: {'n_neighbors': 1}

In [59]:
```
knn1 = KNeighborsRegressor(n_neighbors = 2)
knn1.fit(X_train, y_train)
pred_knn = knn1.predict(X_test)
knn_score = knn1.score(X_test, y_test)
print("Accuracy score for KNN:",knn_score)
```

```
Accuracy score for KNN: 0.8772729010829576
```

In [60]:
```
mse_knn = mse(pred_knn, y_test)
print("Mean Absolute Error for knn: ", mse_knn)
```

```
Mean Absolute Error for knn:  0.0006979566911514066
```

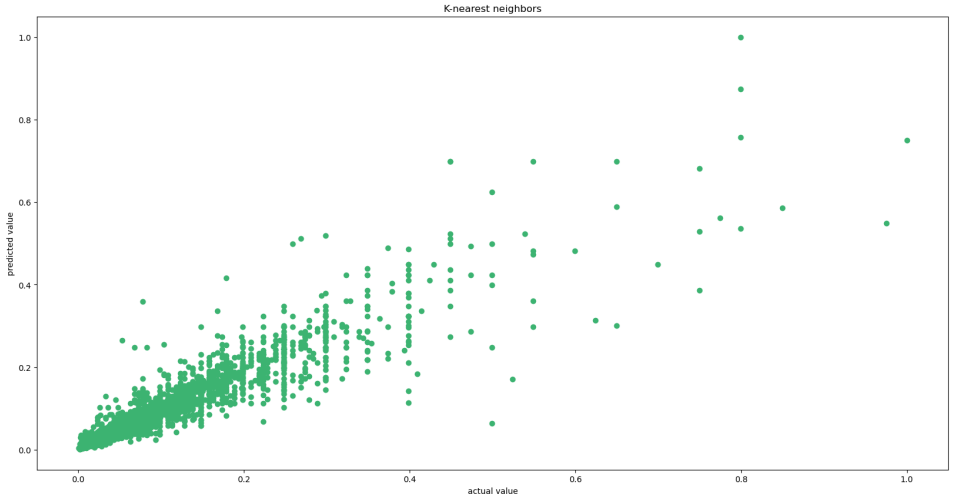
In [61]:
```
mae_knn = mae(pred_knn, y_test)
print("Mean Absolute Error for knn: ", mae_knn)
```

```
Mean Absolute Error for knn:  0.010220433102060727
```

In [62]:
```
r2_knn = r2_score(pred_knn, y_test)
print("r2 score for knn: ", r2_knn)
```

```
r2 score for knn:  0.8554164427042694
```

In [63]:
```
plt.scatter(y_test, pred_knn, color='mediumseagreen');
plt.title("K-nearest neighbors")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```



Decision Tree

In [64]:
```
from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor(criterion='squared_error')
tree.fit(X_train, y_train)
pred_tree = tree.predict(X_test)
```

In [65]:
```
dec_tree_score = tree.score(X_test, y_test)
dec_tree_score
```

Out[65]: 0.9917936777088733

In [66]:
```
MSE_dec_tree = mse(y_test, pred_tree)
print("Mean Squared Error for decision tree:",MSE_dec_tree)
```

```
Mean Squared Error for decision tree: 4.6669868377711904e-05
```

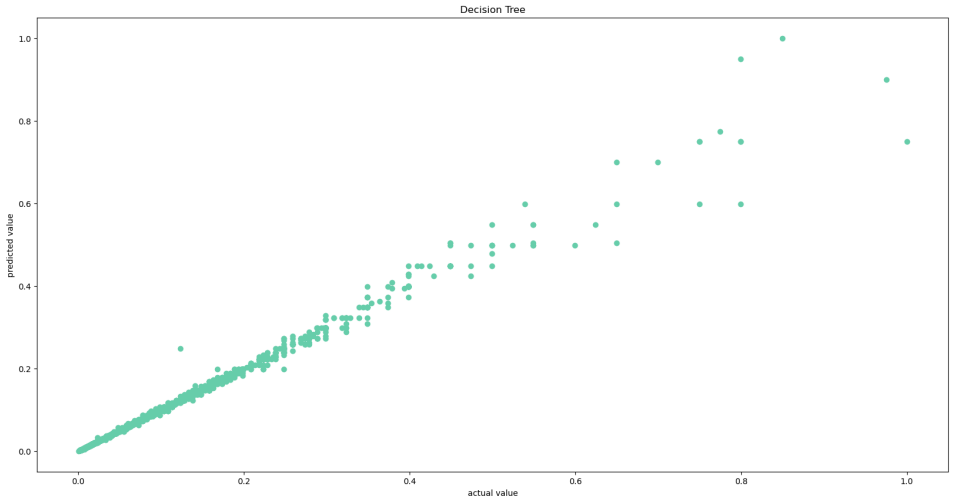
In [67]:
```
mae_tree = mae(pred, y_test)
print("Mean Absolute Error for decision tree:",mae_tree)
```

```
Mean Absolute Error for decision tree: 0.01106157153257731
```

In [68]:
```
r2_tree = r2_score(pred, y_test)
print("r2 score for decision tree:",r2_tree)
```

```
r2 score for decision tree: 0.7982816337205713
```

In [69]:
```
plt.scatter(y_test, pred_tree, color='mediumaquamarine');
plt.title("Decision Tree")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```



Random Forest

In [70]:
```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators = 10, random_state
forest.fit(X_train, y_train)
pred_forest = forest.predict(X_test)
```

```
C:\Users\2501a\anaconda3\Lib\site-packages\sklearn\base.py:11
52: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

In [71]:
```
forest_score = forest.score(X_test, y_test)
forest_score
```

Out[71]:
```
0.9954859745412009
```

In [72]:
```
MSE_forest = mse(y_test, pred_forest)
print("Mean Squared Error for random forest:", MSE_forest)
```

```
Mean Squared Error for random forest: 2.567154524793502e-05
```

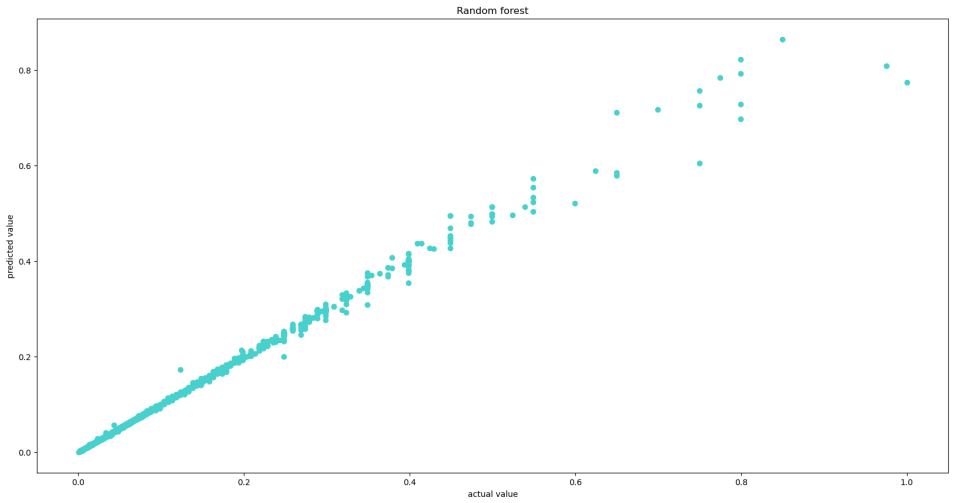
In [73]:
```
mae_forest = mae(y_test, pred_forest)
print("Mean Absolute Error for random forest:",mae_forest)
```

```
Mean Absolute Error for random forest: 0.0007109241241424472
```

In [74]:
```
r2_forest = r2_score(pred_forest, y_test)
print("r2 score for random forest:",r2_forest)
```

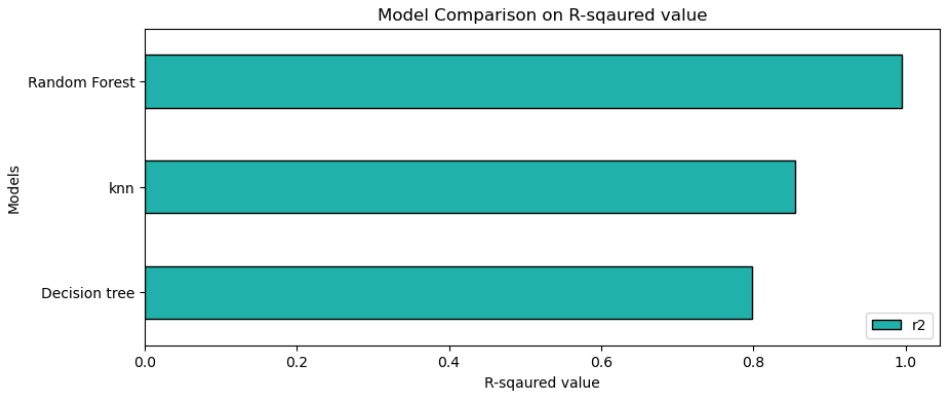
```
r2 score for random forest: 0.9953411129042474
```

In [75]:
```
plt.scatter(x = y_test , y = pred_forest , color='mediumturquoi
plt.title("Random forest")
plt.xlabel("actual value")
plt.ylabel("predicted value");
```

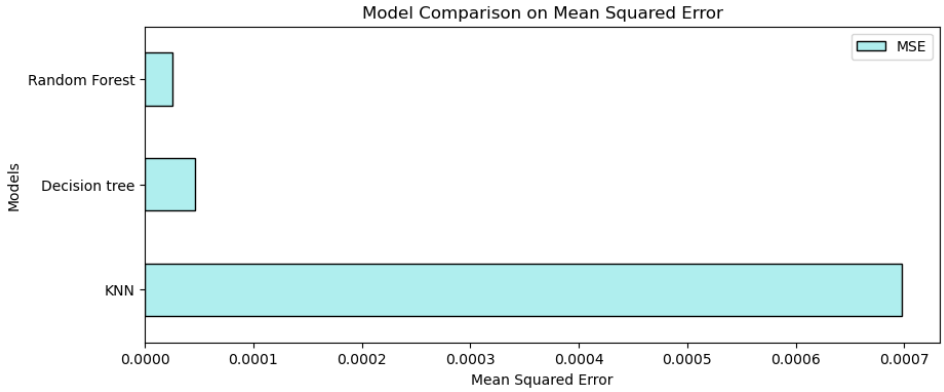


Model Comparison

In [76]:
```
model_comparison_r2 = pd.DataFrame({'Models': [ 'knn', 'Decisic
 'Random Forest'],
 'r2': [r2_knn, r2_tree, r2_forest]})
model_comparison_r2.sort_values('r2', ascending = True).plot(x
 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 [77]:
```
model_comparison_mse = pd.DataFrame({'Models': [ 'KNN', 'Decisi
 'Random Forest'],
 'MSE': [mse_knn, MSE_dec_tree, MSE_forest]})
model_comparison_mse.sort_values('MSE', ascending = False).plot
 y = 'MSE', kind = 'barh', color = 'paleturquoise',
 edgecolor = 'black', figsize = (10,4))
plt.xlabel('Mean Squared Error')
plt.title('Model Comparison on Mean Squared Error')
plt.show()
```



In [78]:
```
model_comparison_mape = pd.DataFrame({'Models': [ 'KNN', 'Decis
 'Random Forest'],
 'MAE': [mae_knn, mae_tree, mae_forest]})
model_comparison_mape.sort_values('MAE', ascending = False).plo
 y = 'MAE', kind = 'barh', color = 'mediumturquoise',
 edgecolor = 'black', figsize = (10,4))
plt.xlabel('Mean Absolute Error')
plt.title('Model Comparison on Mean Absolute Error')
plt.show()
```