In [1]:

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

Importing data sets

In [2]:

```
price_train=pd.read_csv(r"price_train.csv")
price_test=pd.read_csv(r"price_test.csv")
price_train.head()
```

Out[2]:

	X1	id	host_is_superhost	host_response_rate	host_response_time	host_listings_count	host_identity_verified	accc
0	5460	22146017	False	99%	within an hour	521	False	
1	4143	18638163	True	100%	within an hour	1	True	
2	5802	22734110	True	100%	within an hour	1	False	
3	776	3946674	True	90%	within a few hours	1	True	
4	6064	23610186	True	100%	within an hour	4	True	

5 rows × 25 columns

In [3]:

```
# droping irrevelant columns
price_train.drop(['X1',"id"],axis=1,inplace=True)
price_test.drop(["X1"],axis=1,inplace=True)
price_test_id=price_test.pop('id')
print(price_train.shape,price_test.shape)
price_train.head()
```

```
(3466, 23) (1734, 22)
```

Out[3]:

	host_is_superhost	host_response_rate	host_response_time	host_listings_count	host_identity_verified	accommodates	neigl
0	False	99%	within an hour	521	False	5	
1	True	100%	within an hour	1	True	2	
2	True	100%	within an hour	1	False	4	
3	True	90%	within a few hours	1	True	2	
4	True	100%	within an hour	4	True	8	

5 rows × 23 columns

Exploratory Data Analysis and Data engineering

In [4]:

```
# function to check for missing values
def check_missing_data(df):
    mis_col=[]
    for i in df.columns:
        percent=df[i].isnull().sum()
        if percent !=0:
            mis_col.append(i)
    if len(mis_col)==0:
        print('no columns with missing value')
    else:
        print('number of columns with missing value : ',len(mis_col))
        return mis_col
```

In [5]:

```
#checking for missing values
check_missing_data(price_train)
```

```
no columns with missing value
```

In [6]:

```
# preprocessing cleaning_fee and price for train
price_train[['cleaning_fee','price']] = price_train[['cleaning_fee','price']].replace('[
\$,]','',regex=True).astype(float)
```

In [7]:

```
price_train['price'].isnull().sum()
```

Out[7]:

```
0
```

In [8]:

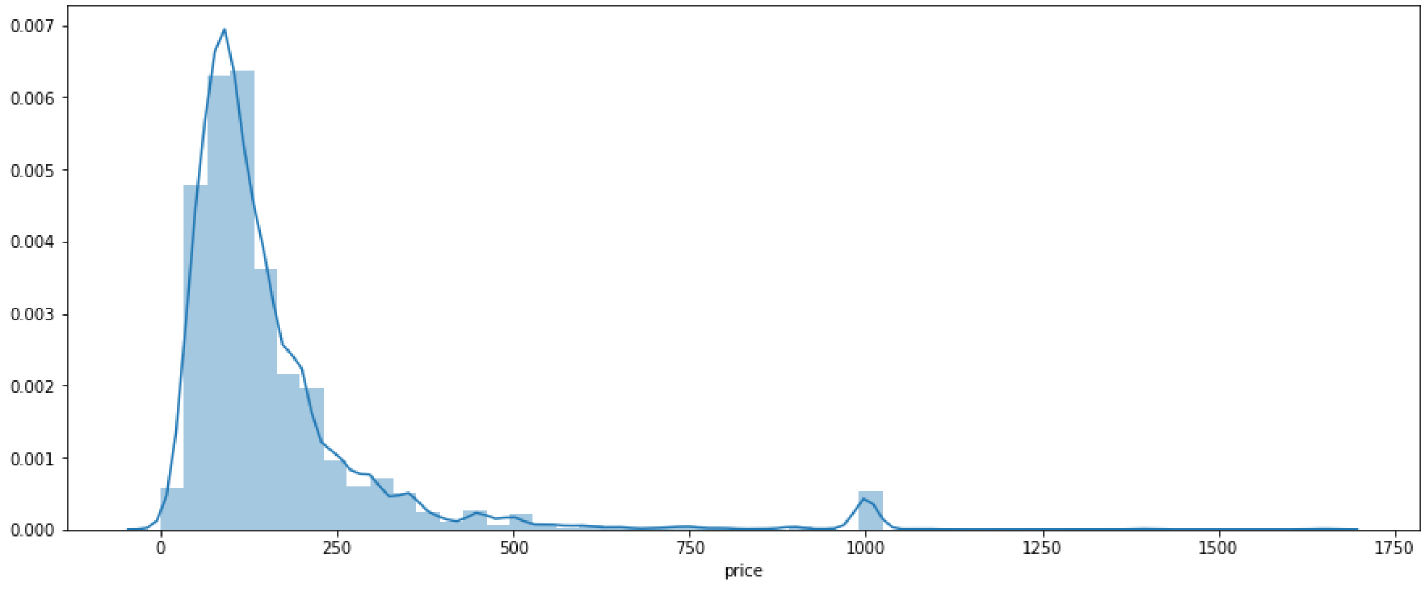
```
# preprocessing cleaning_fee for test
price_test['cleaning_fee'] = price_test['cleaning_fee'].replace('[\$,]','',regex=True).a
stype(float)
```

In [9]:

```
# distribution of price
plt.figure(figsize=(15,6))
sns.distplot(price_train['price'])
```

Out[9]:

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

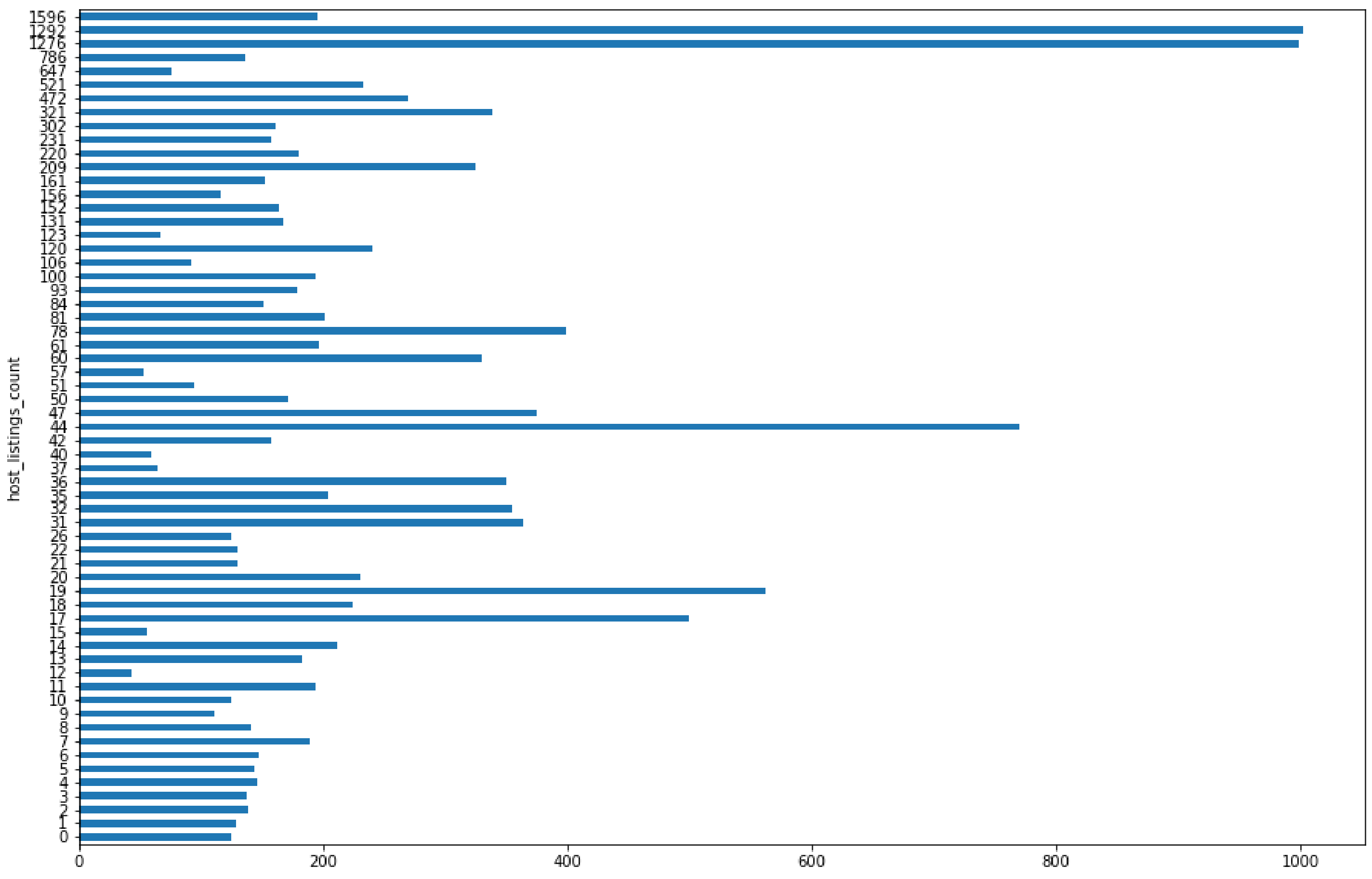


In [10]:

```python
# average price based on host_listings_count
plt.figure(figsize=(15,10))
price_train.groupby('host_listings_count')["price"].mean().plot.barh()
```

Out[10]:

```
<AxesSubplot:ylabel='host_listings_count'>
```

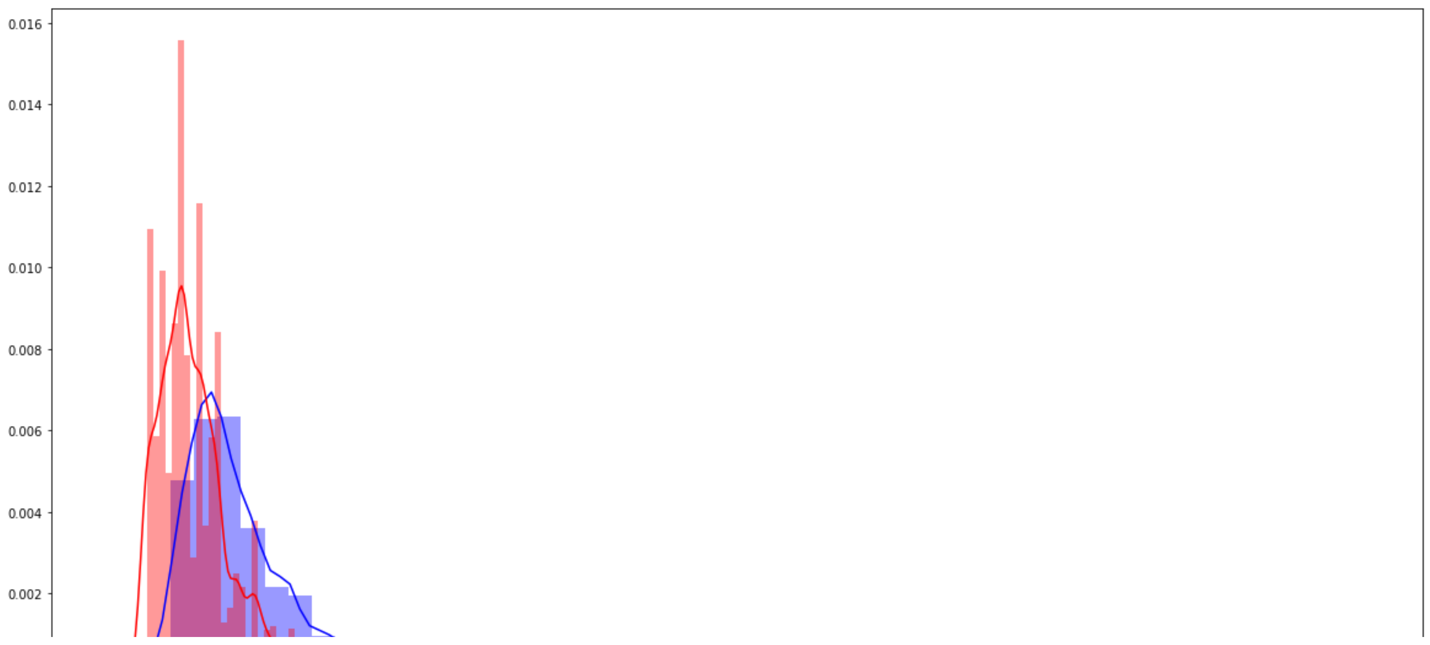


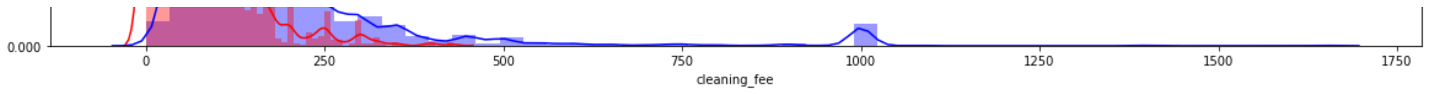
In [11]:

```python
# distribution comparison between cleaning_fee and price
plt.figure(figsize=(20,10))
sns.distplot(price_train['price'],color='blue')
sns.distplot(price_train['cleaning_fee'],color='red')
```

Out[11]:

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



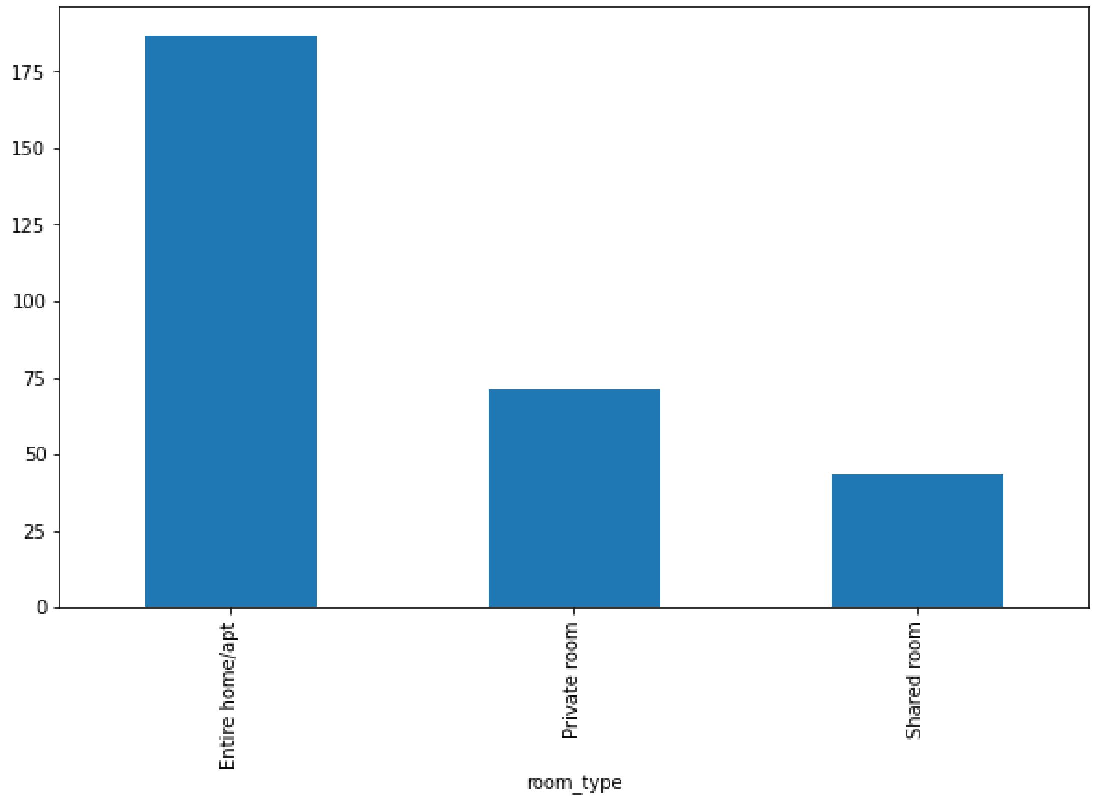


In [12]:

```python
# average price based on the room_type
plt.figure(figsize=(10,6))
price_train.groupby('room_type')["price"].mean().plot.bar()
```

Out[12]:

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

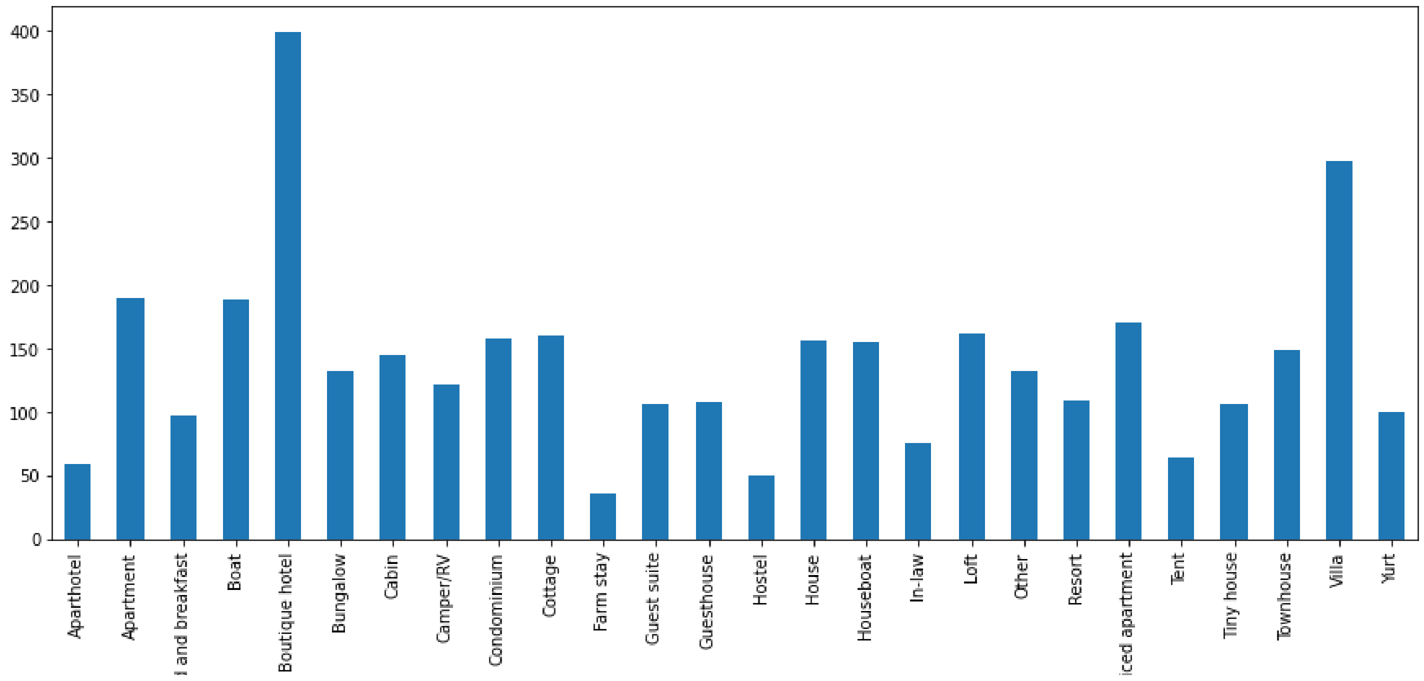


In [13]:

```python
# average price based on the property_type
plt.figure(figsize=(15,6))
price_train.groupby('property_type')["price"].mean().plot.bar()
```

Out[13]:

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



In [14]:

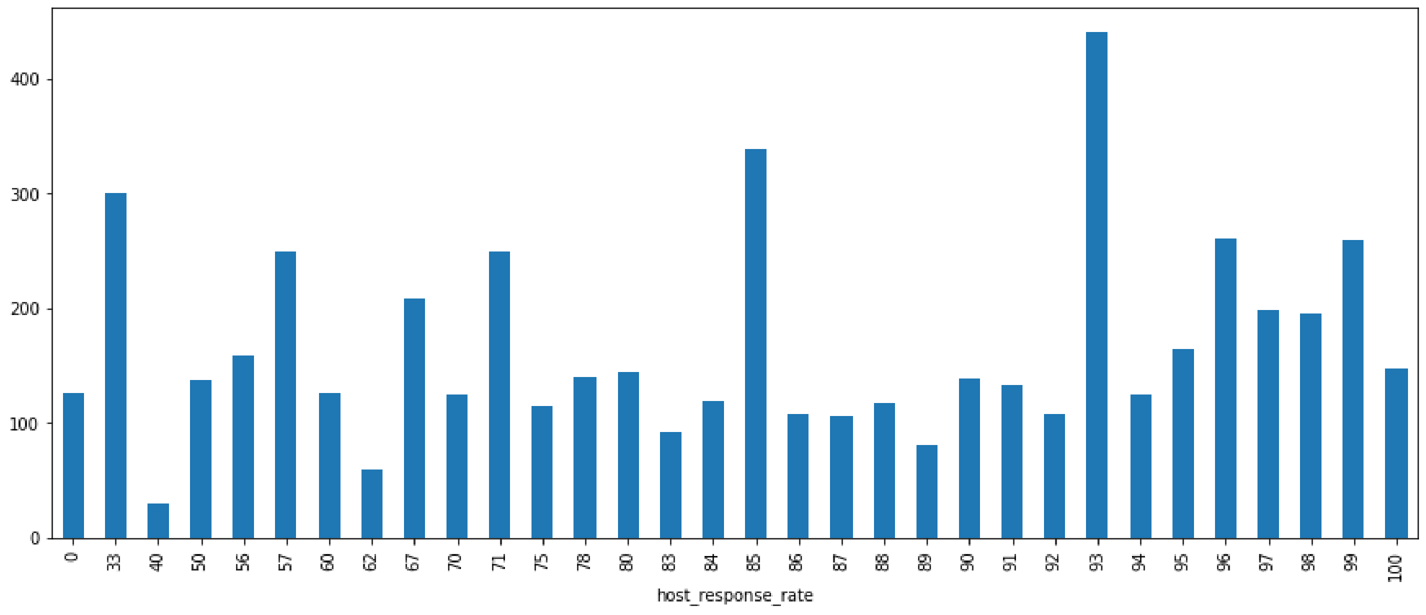
```
# function to preprocess host_response_rate column into int type
def host_response_rate_preprocess(df):
    df['host_response_rate']=df['host_response_rate'].str.replace('%','').astype(int)
    return df
price_train=host_response_rate_preprocess(price_train)
price_test=host_response_rate_preprocess(price_test)
```

In [15]:

```
# average price based on host_response_rate
plt.figure(figsize=(15,6))
price_train.groupby('host_response_rate')["price"].mean().plot.bar()
```

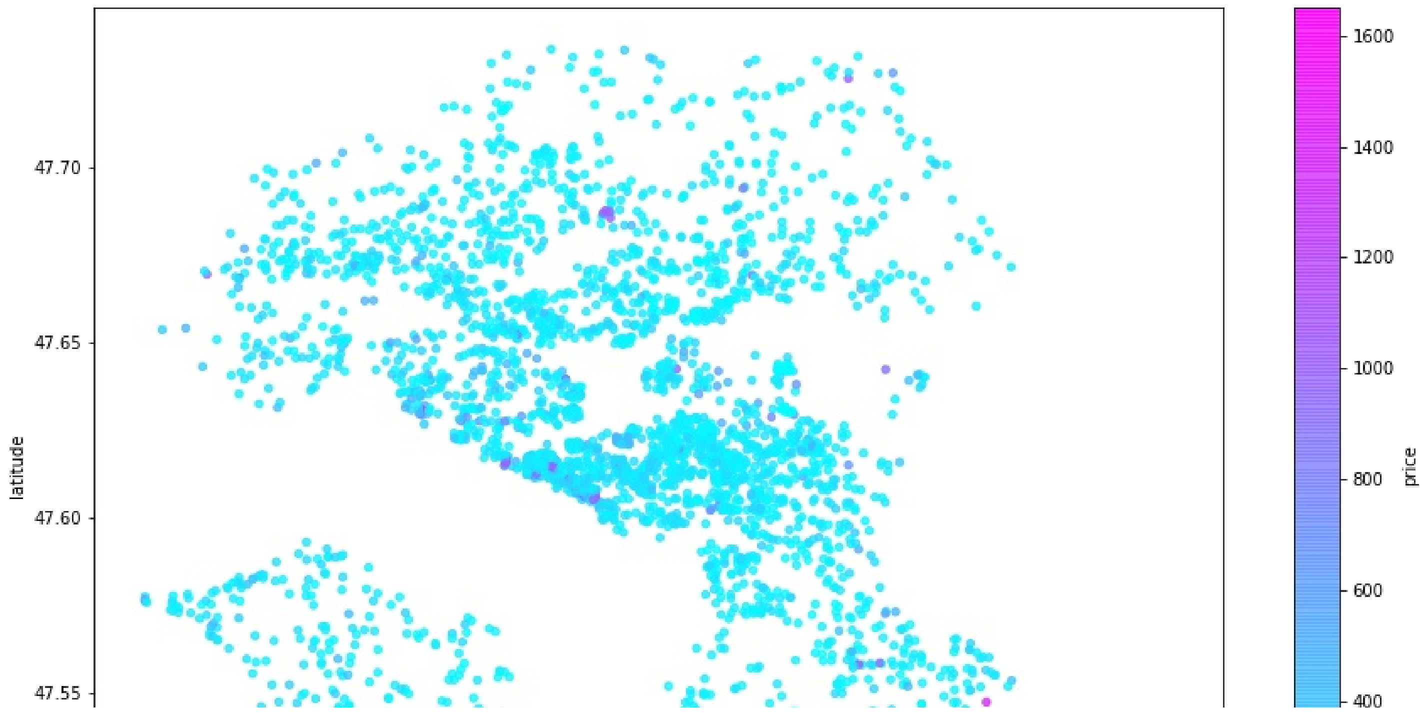
Out[15]:

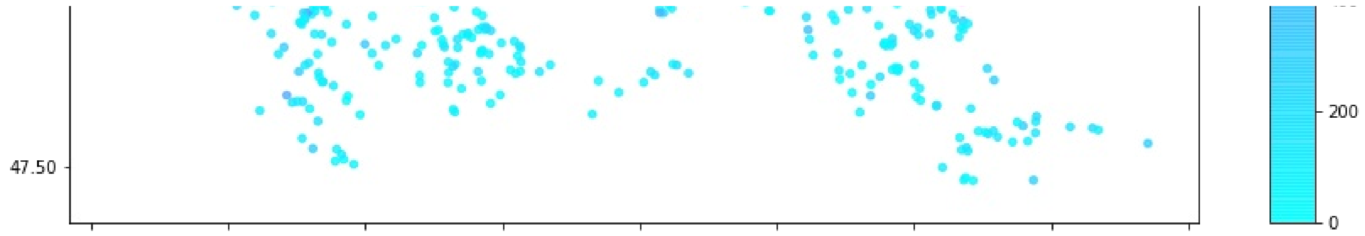
```
<AxesSubplot:xlabel='host_response_rate'>
```



In [16]:

```
# distribution of price variation in different locations
price_train.plot.scatter(x='longitude', y='latitude', c='price', figsize=(15,10), cmap='
cool', alpha=0.8);
```



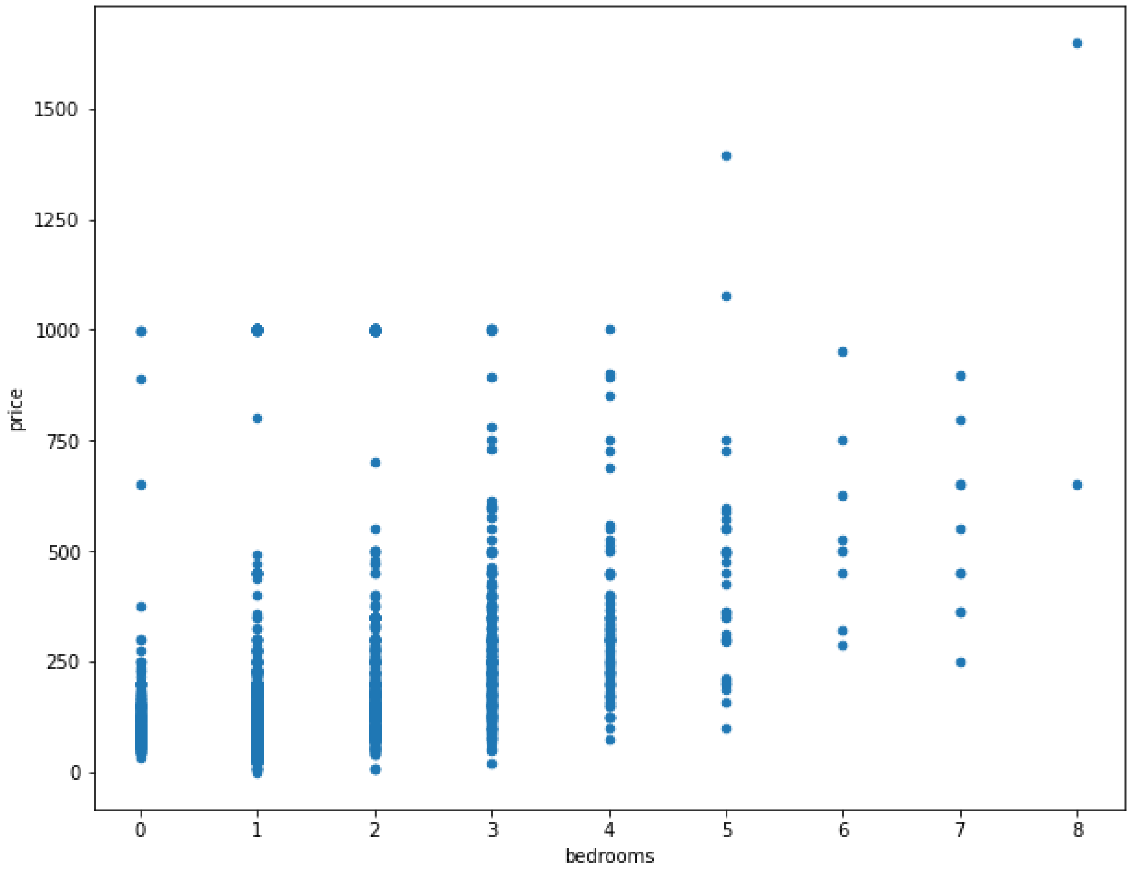


In [17]:

```
# price variation plot depending on bedrooms
price_train.plot.scatter(x='bedrooms', y='price', figsize=(10,8))
```

Out[17]:

```
<AxesSubplot:xlabel='bedrooms', ylabel='price'>
```



In [18]:

```
#function to clean amenities column values
def amenities_clean(df):
    df["amenities"] = df["amenities"].str.lower().str.replace('{','').str.replace('}',''
).str.replace('"','').str.replace(' ','_').str.split(',')
    return df
price_train=amenities_clean(price_train)
price_test=amenities_clean(price_test)
price_train["amenities"]
```

Out[18]:

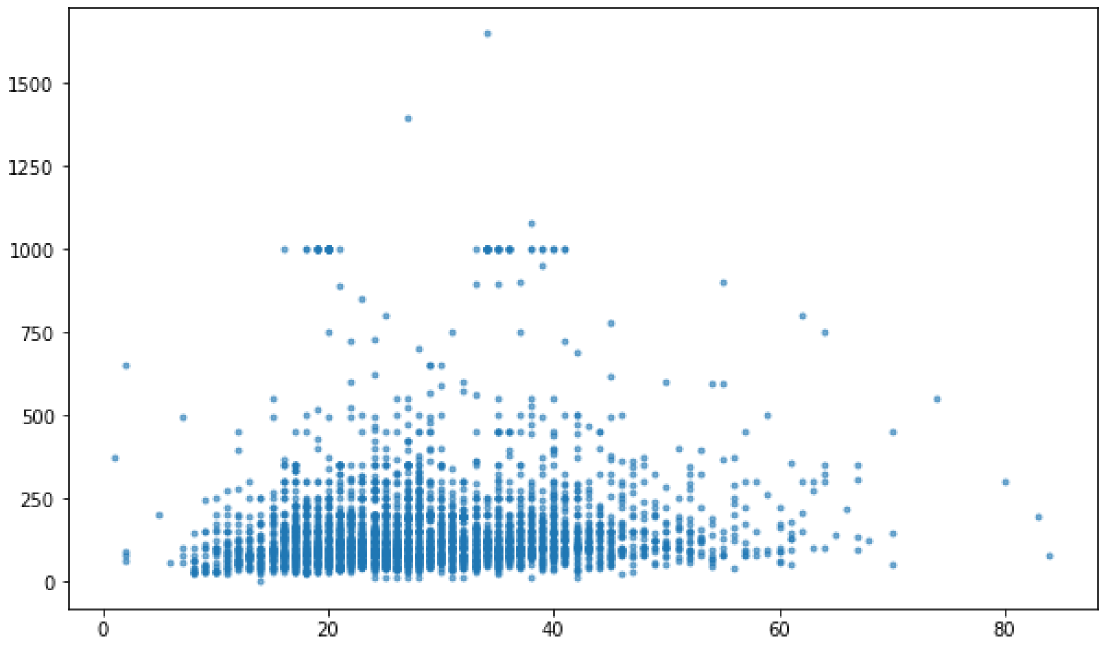
```
0       [tv, internet, wifi, kitchen, elevator, heatin...
1       [tv, cable_tv, internet, wifi, air_conditionin...
2       [tv, wifi, air_conditioning, kitchen, free_str...
3       [internet, wifi, kitchen, pets_live_on_this_pr...
4       [tv, wifi, kitchen, free_parking_on_premises, ...
                              ...
3461    [tv, internet, wifi, air_conditioning, kitchen...
3462    [wifi, kitchen, heating, essentials, shampoo, ...
3463    [wifi, kitchen, free_parking_on_premises, pets...
3464    [tv, wifi, free_parking_on_premises, indoor_fi...
3465    [wifi, kitchen, free_street_parking, heating, ...
Name: amenities, Length: 3466, dtype: object
```

In [19]:

```
# price variation based on the no of ameneties availabe
plt.figure(figsize=(10,6))
number_of_amenities=price_train["amenities"].apply(len)
price=price_train['price']
plt.scatter(number_of_amenities,price,alpha=0.6,marker='.')
```

Out[19]:

```
<matplotlib.collections.PathCollection at 0x7efcfb83a250>
```



In [20]:

```
#function to convert bool values into int
def bool_to_int(df):
    boolean_col=[x for x in df if df[x].dtype==bool]
    if len(boolean_col)==0:
        print("No column with boolean value")
    else:
        for col in boolean_col:
            df[col]=df[col].astype(int)
    return df

price_train=bool_to_int(price_train)
price_test=bool_to_int(price_test)
```

In [21]:

```
# listing out categorical valued columns
def categorical_check(df):
    categ_col=[x for x in df if df[x].dtype=='O']
    if len(categ_col)==0:
        print("No categorical valued columns")
    else:
        print(len(categ_col),'categorical valued columns')
    return categ_col

categ_col_train=categorical_check(price_train)
categ_col_test=categorical_check(price_test)
```

```
7 categorical valued columns
7 categorical valued columns
```

In [22]:

```
#preprocessing amenities column to the number of amenities available to avoid increasing
the dimesnion of the data set
def amenities_len(df):
    df["number_of_amenities"]=df["amenities"].apply(len)
    df.drop(["amenities"],axis=1,inplace=True)
    return df
price_train=amenities_len(price_train)
price_test=amenities_len(price_test)
```

#preprocessing amenities column to Multi Label for deep learning(increases the number of features) def amenities_multilabel(df): from sklearn.preprocessing import MultiLabelBinarizer mlb=MultiLabelBinarizer() label_df=pd.DataFrame(mlb.fit_transform(df.pop("amenities")),columns=mlb.classes_) df=df.join(label_df.drop([label_df.columns[0]],axis=1)) return df price_train=amenities_multilabel(price_train) price_test=amenities_multilabel(price_test)

In [23]:

```
#function for label encoding
def label_encoding_categorical_column(df):
    categ_col=[x for x in df if df[x].dtype=='O']
    from sklearn.preprocessing import LabelEncoder
    for col in categ_col:
        le=LabelEncoder()
        df[col+'label']=le.fit_transform(df[col])
        df.drop([col],axis=1,inplace=True)
    return df

price_train=label_encoding_categorical_column(price_train)
price_test=label_encoding_categorical_column(price_test)
```

In [24]:

```
#checking categorical columns if present
categ_col_train=categorical_check(price_train)
categ_col_test=categorical_check(price_test)
```

```
No categorical valued columns
No categorical valued columns
```

In [25]:

```
#### checking for multicollinearity

#Creating Correlation matrix
corr_matrix=price_train.iloc[:,0:-1].corr().abs()

#selecting upper triangle of correlation matrix
upper_tri=corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(np.bool))

#finding columns to drop whuch have correlation greater than 0.7
to_drop=[x for x in upper_tri.columns if any(upper_tri[x]>0.7)]
print(to_drop)
print('correlated columns to be dropped :',len(to_drop))

price_train.drop(price_train[to_drop],axis=1,inplace=True)
price_test.drop(price_test[to_drop],axis=1,inplace=True)
```

```
['bedrooms', 'beds']
correlated columns to be dropped : 2
```

import csv price_train.to_csv("price_train_clean.csv",index=False) price_test.to_csv("price_test_clean.csv",index=False)

In [26]:

```
#setting train and test data for model training
X,Y=price_train.iloc[:,0:-1],price_train['price']

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=.2)
print(X_train.shape,X_test.shape)
print(Y_train.shape,Y_test.shape)
```

```
(2772, 20) (694, 20)
```

```
(2772,) (694,)
```

In [27]:

```
# Standardizing Data before Training and testing
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
price_test_scaled=scaler.transform(price_test)
```

In [28]:

```
pd.DataFrame(X_train_scaled).describe()
```

Out[28]:

	0	1	2	3	4	5	6	7	
count	2.772000e+03	2.772000e+03	2.772000e+03	2.772000e+03	2.772000e+03	2.772000e+03	2.772000e+03	2.772000e+03	2.
mean	4.085236e-16	-2.081868e-16	-3.393149e-16	1.898433e-16	-1.797254e-16	-7.056216e-14	2.691830e-13	-6.448265e-18	
std	1.000180e+00	1.000180e+00	1.000180e+00	1.000180e+00	1.000180e+00	1.000180e+00	1.000180e+00	1.000180e+00	1.
min	-9.971181e-01	-1.004636e+01	-3.178043e-01	-9.336532e-01	-1.125681e+00	-2.843220e+00	-2.678597e+00	-6.122285e-01	1.
25%	-9.971181e-01	2.021229e-01	-3.132322e-01	-9.336532e-01	-7.166105e-01	-4.420907e-01	-6.200294e-01	-6.122285e-01	
50%	-9.971181e-01	2.021229e-01	-3.086600e-01	-9.336532e-01	-3.075404e-01	-9.912441e-02	2.251998e-02	-6.122285e-01	
75%	1.002890e+00	2.021229e-01	-2.720828e-01	1.071061e+00	2.037972e-01	7.540685e-01	6.774103e-01	-6.389332e-02	
max	1.002890e+00	2.021229e-01	6.979354e+00	1.071061e+00	9.919212e+00	2.374619e+00	3.196944e+00	7.612800e+00	1.

Initial Modelling

In [29]:

```
from sklearn.linear_model import LinearRegression,ElasticNet
from sklearn import svm
import xgboost
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error
```

In [30]:

```
# functionh to train,predict and display results
def model_training(model_dict,X,Y,X_test,Y_test):
    import numpy as np
    from sklearn.linear_model import LinearRegression,ElasticNet
    from sklearn import svm
    import xgboost
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import r2_score,mean_squared_error

    R2_score=[]
    Mean_squared_error=[]
    Root_mean_squared_error=[]
    Models=[]
    for name,reg in model_dict.items():
        Models.append(name)
        regressor=reg()
        regressor.fit(X,Y)
        print("fitted",name)
        Y_pred=regressor.predict(X_test)
        R2_score.append(r2_score(Y_test,Y_pred))
        Mean_squared_error.append(mean_squared_error(Y_test,Y_pred))
        Root_mean_squared_error.append(np.sqrt(mean_squared_error(Y_test,Y_pred)))
    results={'Models':Models,"r2_score":R2_score,"mean_squared_error":Mean_squared_error,
"root_mean_squared_error":Root_mean_squared_error}
    return pd.DataFrame(results)
```

In [31]:

```
# model training scaled data set
model_dict={'Linear':LinearRegression,'SVM':svm.SVR,'ElasticNet':ElasticNet,"Random Fores
t":RandomForestRegressor,'XGboost':xgboost.XGBRegressor}
result=model_training(model_dict,X_train_scaled,Y_train,X_test_scaled,Y_test)
result
```

```
fitted Linear
fitted SVM
fitted ElasticNet
fitted Random Forest
fitted XGboost
```

Out[31]:

	Models	r2_score	mean_squared_error	root_mean_squared_error
0	Linear	1.000000	3.309306e-26	1.819150e-13
1	SVM	0.231592	1.365089e+04	1.168370e+02
2	ElasticNet	0.879408	2.142343e+03	4.628545e+01
3	Random Forest	0.998114	3.349763e+01	5.787714e+00
4	XGboost	0.999184	1.448842e+01	3.806365e+00

In [32]:

```
# model training un-scaled data set
tree_model_dict={"Random Forest":RandomForestRegressor,'XGboost':xgboost.XGBRegressor}
tree_result=model_training(tree_model_dict,X_train,Y_train,X_test,Y_test)
tree_result
```

```
fitted Random Forest
fitted XGboost
```

Out[32]:

	Models	r2_score	mean_squared_error	root_mean_squared_error
0	Random Forest	0.997731	40.311474	6.349132
1	XGboost	0.999195	14.305331	3.782239

Model Selection

since for linear regression the r-square value is high and the errors were low,hence,Linear Regression was selected for the problem statement

In [33]:

```
# Fitting and training the model
linear_regression=LinearRegression()
linear_regression.fit(X_train_scaled,Y_train)
print("fitted")
```

Final Prediction

In [33]:

```
price_test_predicted=linear_regression.predict(price_test_scaled)
price_test_predicted
```

```
fitted
```

Out[33]:

```
array([24., 27., 24., ..., 33., 30., 50.])
```

In [35]:

```
#converting to csv file
df=pd.DataFrame({'id':price_test_id,'price':price_test_predicted})
df.to_csv("submission.csv",index=False)
```

In [ ]:

```

```