Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
```

Loading the data

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

In [3]:
```
df.shape
```

Out[3]: (3818, 33)

In [4]:
```
df1 = pd.read_csv('seattle.csv')
```

In [5]:
```
df1.shape
```

Out[5]: (3818, 92)

Level-0 Analysis and Data Cleaning

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3818 entries, 0 to 3817
Data columns (total 33 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   id                                3818 non-null   int64  
 1   name                              3818 non-null   object 
 2   host_id                           3818 non-null   int64  
 3   host_name                         3816 non-null   object 
 4   host_since                        3816 non-null   object 
 5   host_is_superhost                 3816 non-null   object 
 6   host_listings_count               3816 non-null   float64
 7   host_total_listings_count         3816 non-null   float64
 8   host_has_profile_pic              3816 non-null   object 
 9   host_identity_verified            3816 non-null   object 
 10  is_location_exact                 3818 non-null   object 
 11  property_type                     3817 non-null   object 
 12  room_type                         3818 non-null   object 
 13  accommodates                      3818 non-null   int64  
 14  bathrooms                         3802 non-null   float64
 15  bedrooms                          3812 non-null   float64
 16  beds                              3817 non-null   float64
 17  bed_type                          3818 non-null   object 
 18  amenities                         3818 non-null   object 
 19  square_feet                       97 non-null     float64
 20  price                             3818 non-null   object 
 21  guests_included                   3818 non-null   int64  
 22  extra_people                      3818 non-null   object 
 23  minimum_nights                    3818 non-null   int64  
 24  maximum_nights                    3818 non-null   int64  
 25  number_of_reviews                 3818 non-null   int64  
 26  review_scores_rating              3171 non-null   float64
 27  instant_bookable                  3818 non-null   object 
 28  cancellation_policy               3818 non-null   object 
 29  require_guest_profile_picture     3818 non-null   object 
 30  require_guest_phone_verification  3818 non-null   object 
 31  calculated_host_listings_count    3818 non-null   int64  
 32  reviews_per_month                 3191 non-null   float64
dtypes: float64(8), int64(8), object(17)
memory usage: 984.5+ KB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
id                                  0
name                                0
host_id                             0
host_name                           2
host_since                          2
host_is_superhost                   2
host_listings_count                 2
host_total_listings_count           2
host_has_profile_pic                2
host_identity_verified              2
is_location_exact                   0
property_type                       1
room_type                           0
accommodates                        0
bathrooms                          16
bedrooms                            6
beds                                1
bed_type                            0
amenities                           0
square_feet                      3721
price                               0
guests_included                     0
extra_people                        0
minimum_nights                      0
maximum_nights                      0
number_of_reviews                   0
review_scores_rating              647
instant_bookable                    0
cancellation_policy                 0
require_guest_profile_picture       0
require_guest_phone_verification    0
calculated_host_listings_count      0
reviews_per_month                 627
dtype: int64
```

In [8]:
```
df.head()
```

Out[8]:

	id	name	host_id	host_name	host_since	host_is_superhost	host_listings_count	host_total_listings_count	host_has_profi
0	241032	Stylish Queen Anne Apartment	956883	Maija	11-08-2011	f	3.0	3.0	
1	953595	Bright & Airy Queen Anne Apartment	5177328	Andrea	21-02-2013	t	6.0	6.0	
2	3308979	New Modern House-Amazing water view	16708587	Jill	12-06-2014	f	2.0	2.0	
3	7421966	Queen Anne Chateau	9851441	Emily	06-11-2013	f	1.0	1.0	
4	278830	Charming craftsman 3 bdm house	1452570	Emily	29-11-2011	f	2.0	2.0	

5 rows × 33 columns

In [9]:
```
cols = ['last_scraped','host_response_time','host_neighbourhood','neighbourhood_cleansed','city',
 'state','market','country','latitude','longitude','availability_365','cleaning_fee','security_deposit','calenda
```

In [10]:
```
df2 = df1[cols]
```

In [11]:
```
df3 = pd.concat([df,df2],axis=1)
```

In [12]:
```
df3.drop(['square_feet','review_scores_rating','reviews_per_month'],axis=1,inplace=True)
```

In [13]:
```
df3['cleaning_fee'].fillna('Not Available',inplace=True)
df3['security_deposit'].fillna('Not Available',inplace=True)
```

In [14]:
```
df3['host_response_time'].fillna('Unknown',inplace=True)
```

In [15]:
```
df3['host_neighbourhood'].fillna('Unknown',inplace=True)
```

In [16]:
```
df3.shape
```

Out[16]: (3818, 44)

In [17]:
```
df3.dropna(inplace=True)
```

In [18]:
```
df3.shape
```

Out[18]: (3793, 44)

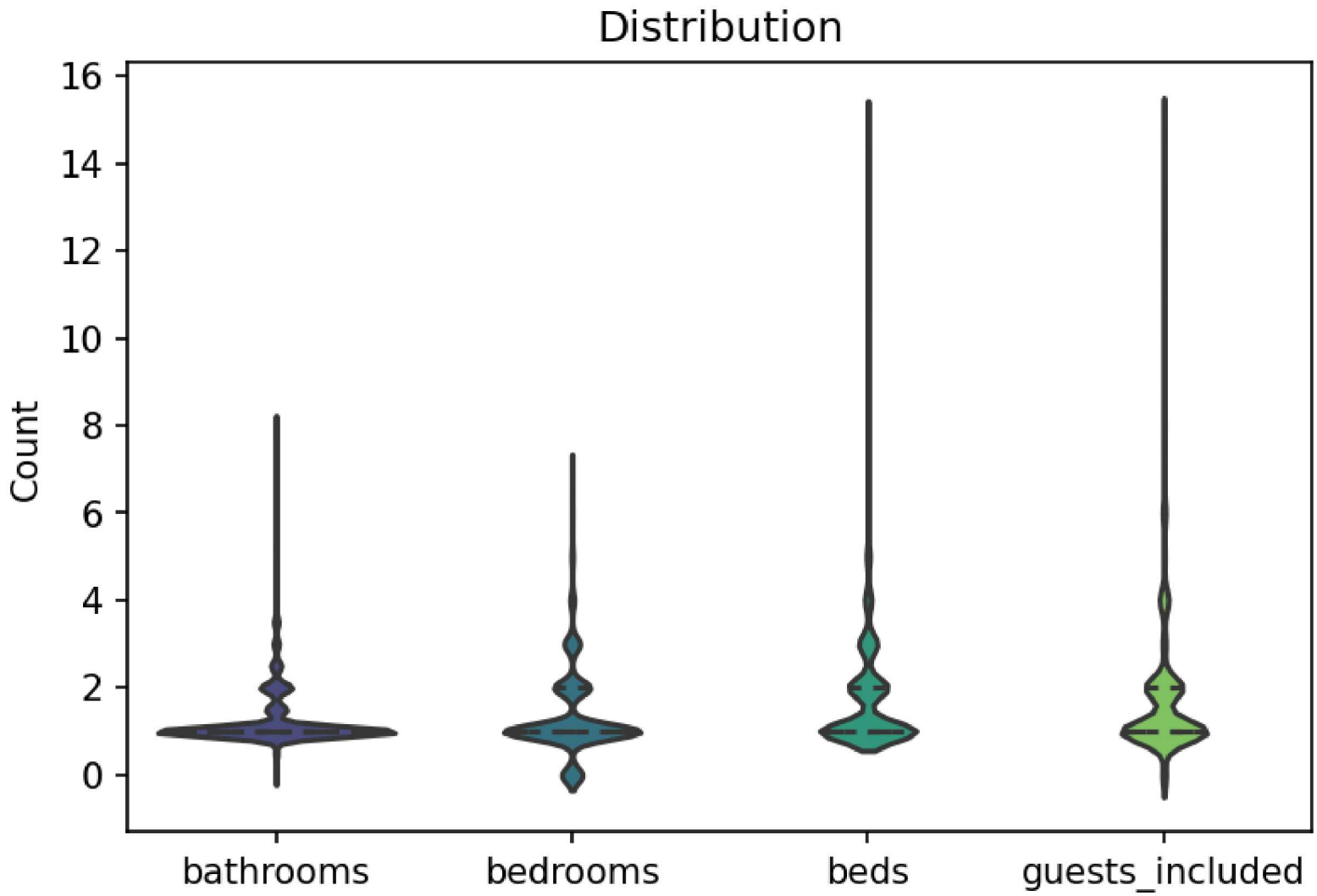
In [19]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3793 entries, 0 to 3817
Data columns (total 44 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   id                                3793 non-null   int64  
 1   name                              3793 non-null   object 
 2   host_id                           3793 non-null   int64  
 3   host_name                         3793 non-null   object 
 4   host_since                        3793 non-null   object 
 5   host_is_superhost                 3793 non-null   object 
 6   host_listings_count               3793 non-null   float64
 7   host_total_listings_count         3793 non-null   float64
 8   host_has_profile_pic              3793 non-null   object 
 9   host_identity_verified            3793 non-null   object 
 10  is_location_exact                 3793 non-null   object 
 11  property_type                     3793 non-null   object 
 12  room_type                         3793 non-null   object 
 13  accommodates                      3793 non-null   int64  
 14  bathrooms                         3793 non-null   float64
 15  bedrooms                          3793 non-null   float64
 16  beds                              3793 non-null   float64
 17  bed_type                          3793 non-null   object 
 18  amenities                         3793 non-null   object 
 19  price                             3793 non-null   object 
 20  guests_included                   3793 non-null   int64  
 21  extra_people                      3793 non-null   object 
 22  minimum_nights                    3793 non-null   int64  
 23  maximum_nights                    3793 non-null   int64  
 24  number_of_reviews                 3793 non-null   int64  
 25  instant_bookable                  3793 non-null   object 
 26  cancellation_policy               3793 non-null   object 
 27  require_guest_profile_picture     3793 non-null   object 
 28  require_guest_phone_verification  3793 non-null   object 
 29  calculated_host_listings_count    3793 non-null   int64  
 30  last_scraped                      3793 non-null   object 
 31  host_response_time                3793 non-null   object 
 32  host_neighbourhood                3793 non-null   object 
 33  neighbourhood_cleansed            3793 non-null   object 
 34  city                              3793 non-null   object 
```

```
 35  state                          3793 non-null   object
 36  market                         3793 non-null   object
 37  country                        3793 non-null   object
 38  latitude                       3793 non-null   float64
 39  longitude                      3793 non-null   float64
 40  availability_365               3793 non-null   int64
 41  cleaning_fee                   3793 non-null   object
 42  security_deposit               3793 non-null   object
 43  calendar_last_scraped          3793 non-null   object
dtypes: float64(7), int64(9), object(28)
memory usage: 1.3+ MB
```

In [20]:
```
df3.to_csv('cleaned_seattle_data.csv')
```

In [21]:
```
plt.figure(dpi=150)
sns.violinplot(data = df3[['bathrooms','bedrooms','beds','guests_included']],palette='viridis',inner='quartile')
plt.title('Distribution')
plt.ylabel('Count');
```

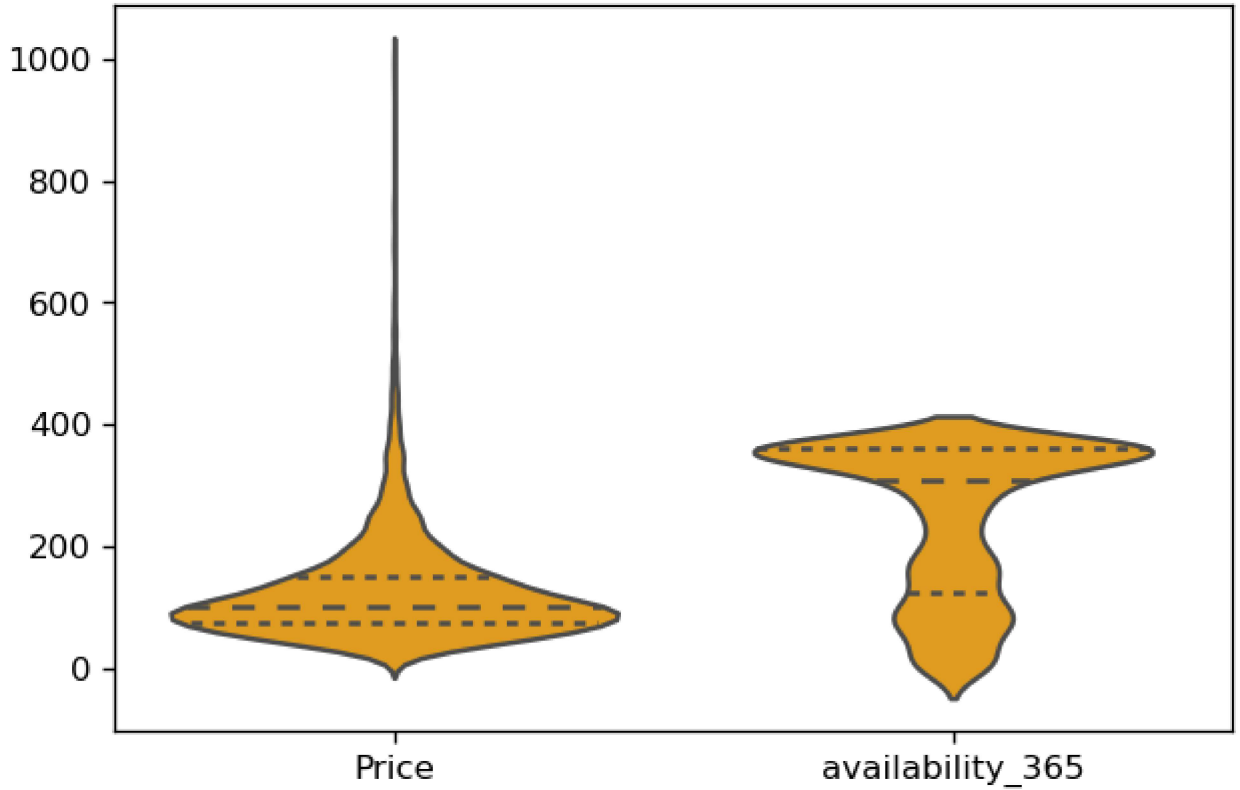


In [22]:
```
price = []
for i in df3.price:
    temp = 0
    if len(i) == 6 or len(i) == 7:
        temp = float(i[1:])
        price.append(temp)
    elif len(i) == 9:
        last = i[3:]
        first = i[1]
        complete = first+last
        temp = float(complete)
        price.append(temp)
```

In [23]:
```
df3['Price'] = price
```

In [24]:
```
df3.drop('price',axis=1,inplace=True)
```

In [25]:
```
plt.figure(dpi=120)
sns.violinplot(data = df3[['Price','availability_365']],color='orange',inner='quartile');
```



In [26]:
```
d=[]
for i in df3['extra_people']:
    d.append(i.replace("$","").replace(",",""))
b  = [float(d) for d in d]
df3.drop('extra_people',axis=1)
df3['extra_people'] = b
```

In [27]:
```
df3.to_csv('seattle_data_final.csv')
```

In [28]:
```
df_dep = df3[df3.security_deposit != 'Not Available'].security_deposit
idx = df_dep.index
```

In [29]:
```
df_nb = df3[df3.index.isin(idx)].host_neighbourhood
```

In [30]:
```
d=[]
for i in df_dep:
    d.append(i.replace("$","").replace(",",""))
b  = [float(d) for d in d]
df_dep = pd.DataFrame(data=b,columns=['deposit'])
```

In [31]:
```
df_dep
```

Out[31]:

	deposit
0	100.0
1	1000.0
2	700.0
3	150.0
4	150.0
...	...
1853	150.0
1854	500.0
1855	500.0
1856	250.0
1857	300.0

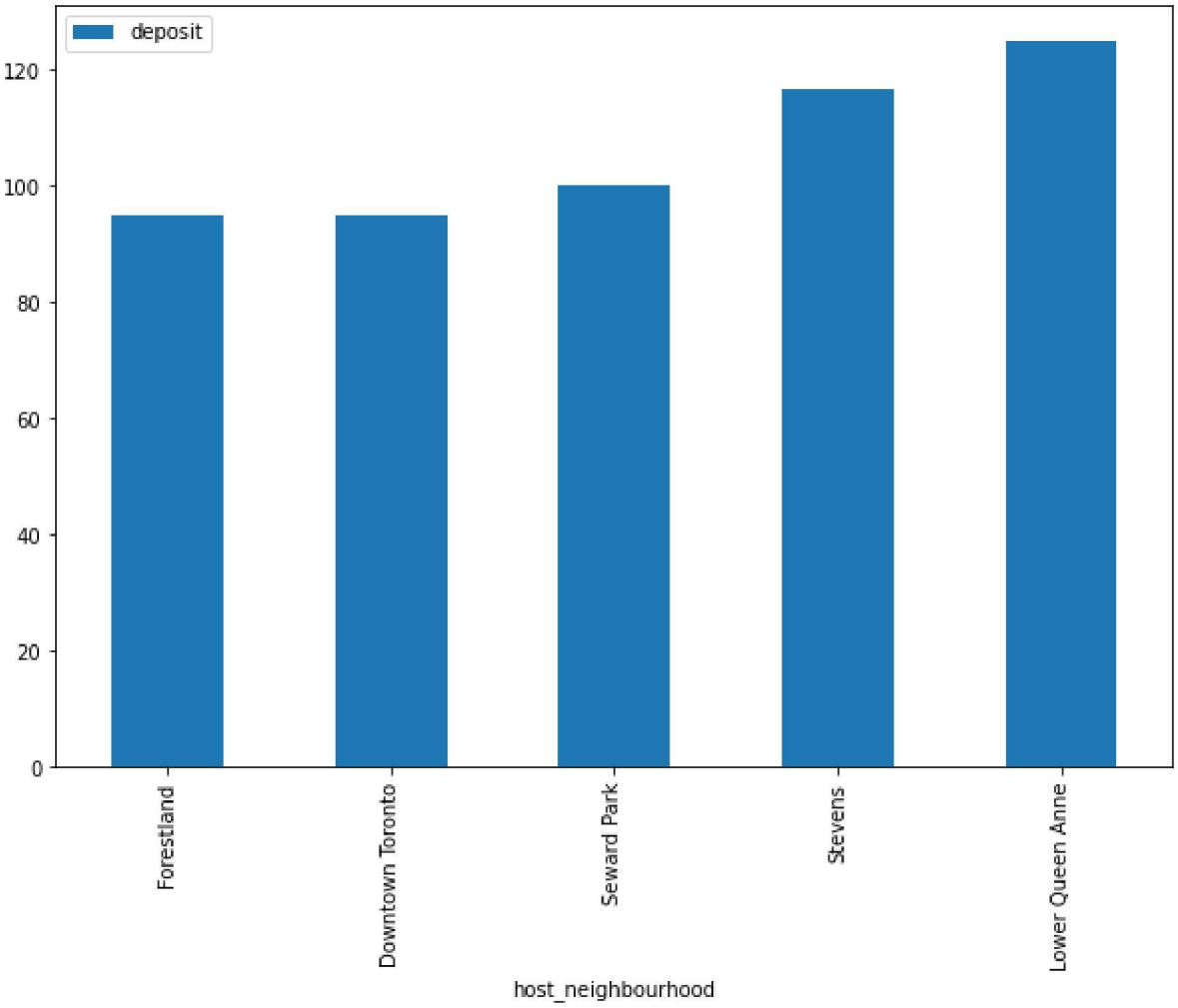
1858 rows × 1 columns

In [32]:
```
df_p = pd.concat([df_dep,df_nb],axis=1)
```

Neighbourhood with least security deposits

In [33]:
```
df_p.groupby(['host_neighbourhood']).mean().sort_values(by = 'deposit',ascending=True)[:5].plot(kind='bar',figsi
```

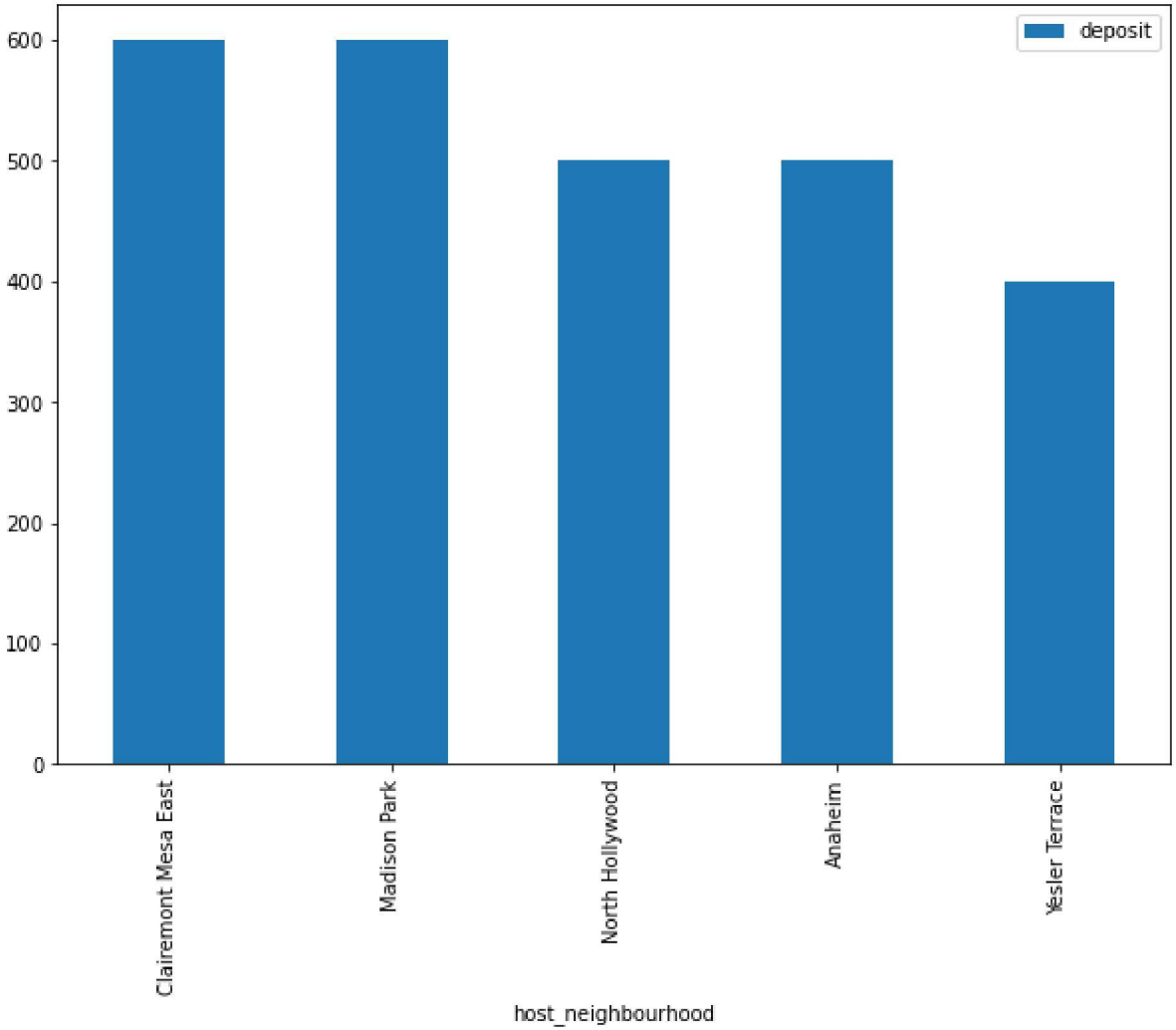
Out[33]: <AxesSubplot:xlabel='host_neighbourhood'>



Neighbourhoods with highest security deposits

In [34]:
```
df_p.groupby(['host_neighbourhood']).mean().sort_values(by = 'deposit',ascending=False)[:5].plot(kind='bar',figs
```

Out[34]: <AxesSubplot:xlabel='host_neighbourhood'>



In [35]:
```
df_cln = df3[df3.cleaning_fee != 'Not Available'].cleaning_fee
df_neighbourhood = df3[df3.security_deposit != 'Not Available'].host_neighbourhood
```

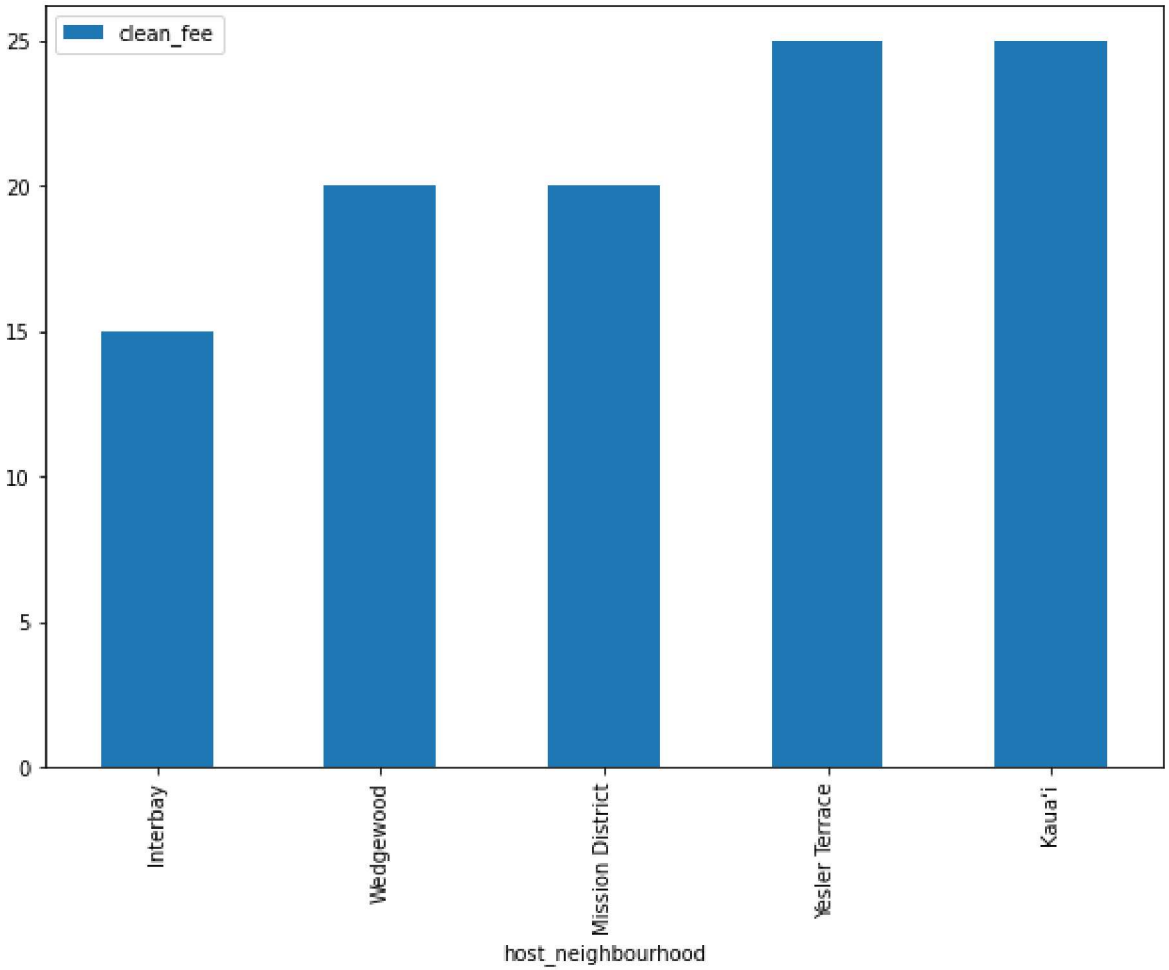
In [36]:
```
d=[]
for i in df_cln:
    d.append(i.replace("$","").replace(",",""))
b  = [float(d) for d in d]
df_cln = pd.DataFrame(data=b,columns=['clean_fee'])
```

In [37]:
```
df_c = pd.concat([df_cln,df_nb],axis=1)
```

Neighbourhood with least Cleaning Fee

In [38]:
```
df_c.groupby(['host_neighbourhood']).mean().sort_values(by = 'clean_fee',ascending=True)[:5].plot(kind='bar',fig
```

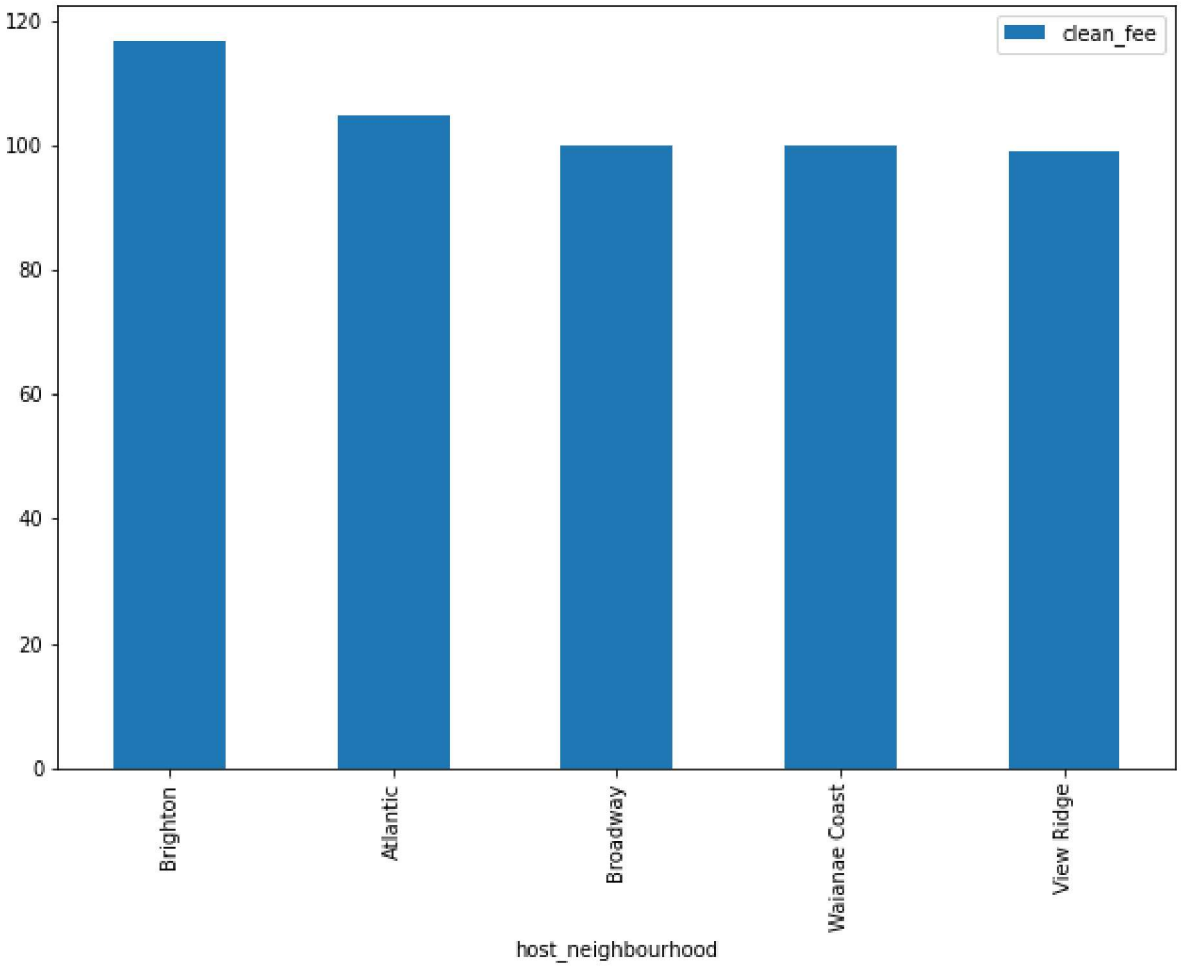
Out[38]: <AxesSubplot:xlabel='host_neighbourhood'>



Neighbourhood with Highest Cleaning Fee

In [39]:
```
df_c.groupby(['host_neighbourhood']).mean().sort_values(by = 'clean_fee',ascending=False)[:5].plot(kind='bar',fi
```

Out[39]: <AxesSubplot:xlabel='host_neighbourhood'>



Problem_Statement_4: Revenue Forecasting

Selected Seattle city data for its revenue forecasting

In [40]:
```
df_ts = df3[['host_since','Price']]
```

In [41]:
```
df_ts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3793 entries, 0 to 3817
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   host_since  3793 non-null   object 
 1   Price       3793 non-null   float64
dtypes: float64(1), object(1)
memory usage: 88.9+ KB
```

In [42]:
```
df_ts['host_since']=pd.to_datetime(df_ts['host_since'])
```

In [43]:
```
df_ts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3793 entries, 0 to 3817
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   host_since  3793 non-null   datetime64[ns]
 1   Price       3793 non-null   float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 88.9 KB
```

In [44]:
```
df_ts.head()
```

Out[44]:

	host_since	Price
0	2011-11-08	85.0
1	2013-02-21	150.0
2	2014-12-06	975.0
3	2013-06-11	100.0
4	2011-11-29	450.0

In [45]:
```
idx1 = df_ts[(df_ts['host_since'] =='2016-01-01' )|(df_ts.host_since =='2016-02-01')|(df_ts.host_since =='2016-0
```

In [46]:
```
len(idx1)
```

Out[46]: 5

In [47]:
```
df_ts.shape
```

Out[47]: (3793, 2)

In [48]:
```
df_ts.drop(idx1,inplace=True)
```

In [49]:
```
df_ts.shape
```

Out[49]:
```
(3788, 2)
```

In [50]:
```
df_ts.sort_values(by='host_since',inplace=True)
```

In [51]:
```
df_ts
```

Out[51]:

	host_since	Price
2190	2008-10-11	48.0
2192	2008-10-11	50.0
2194	2008-10-11	100.0
3118	2009-02-09	82.0
2425	2009-02-09	137.0
...	...	...
1139	2015-12-22	90.0
2589	2015-12-27	90.0
118	2015-12-28	65.0
3085	2015-12-29	50.0
3815	2015-12-30	93.0

3788 rows × 2 columns

In [52]:
```
df_ts.set_index('host_since',inplace=True)
```

In [53]:
```
df_ts.head()
```

Out[53]:

host_since	Price
2008-10-11	48.0
2008-10-11	50.0
2008-10-11	100.0
2009-02-09	82.0
2009-02-09	137.0

In [54]:
```
df_ts_sum = df_ts.resample(rule='MS').sum()
```

In [55]:
```
df_ts_sum.head()
```

Out[55]:

host_since	Price
2008-10-01	198.0
2008-11-01	0.0
2008-12-01	0.0
2009-01-01	0.0
2009-02-01	2048.0

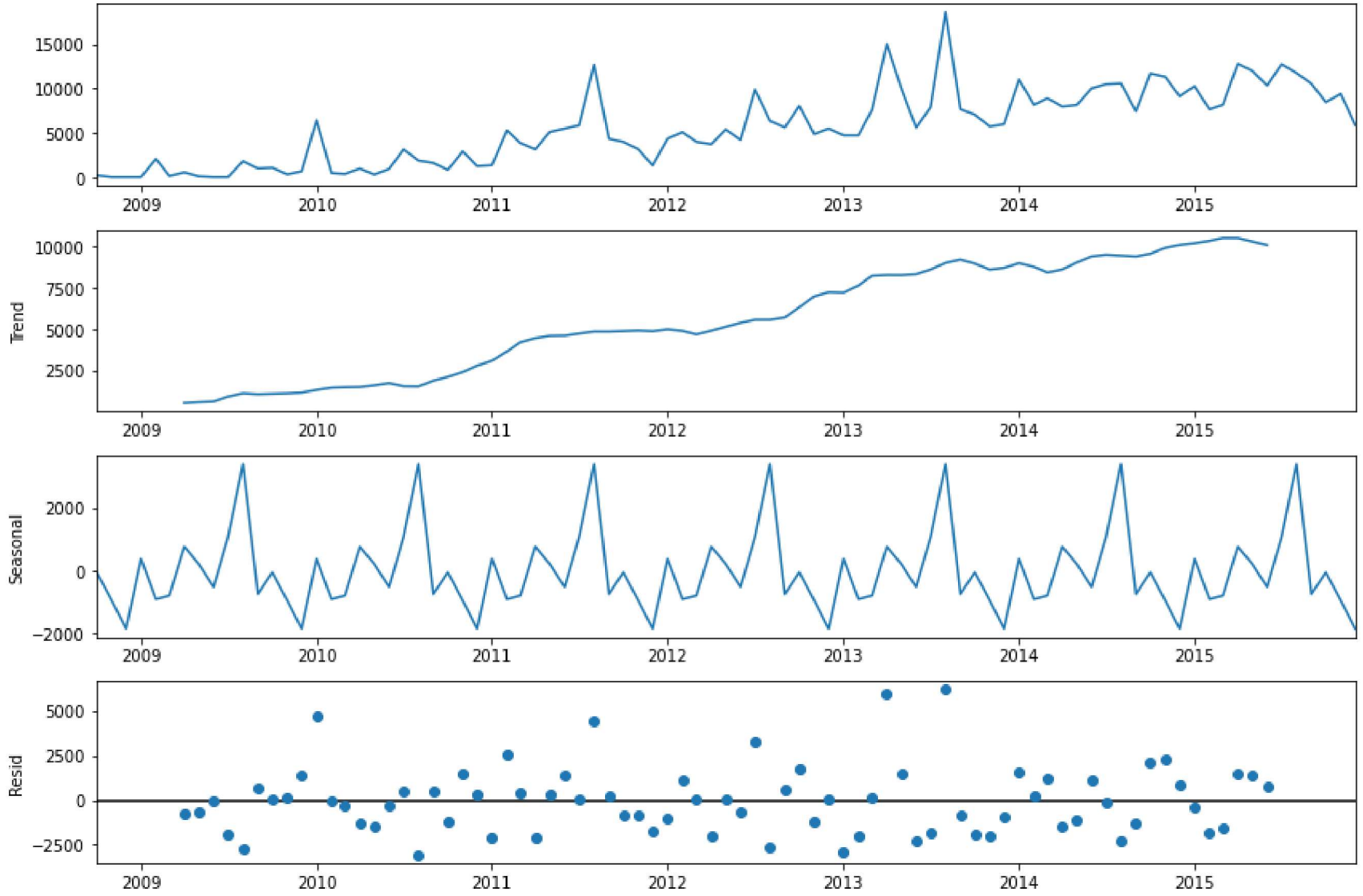
In [56]:
```
df_ts_sum.index.freq = 'MS'
```

In [57]:
```
df_ts_sum.index
```

Out[57]:
```
DatetimeIndex(['2008-10-01', '2008-11-01', '2008-12-01', '2009-01-01',
               '2009-02-01', '2009-03-01', '2009-04-01', '2009-05-01',
               '2009-06-01', '2009-07-01', '2009-08-01', '2009-09-01',
               '2009-10-01', '2009-11-01', '2009-12-01', '2010-01-01',
               '2010-02-01', '2010-03-01', '2010-04-01', '2010-05-01',
               '2010-06-01', '2010-07-01', '2010-08-01', '2010-09-01',
               '2010-10-01', '2010-11-01', '2010-12-01', '2011-01-01',
               '2011-02-01', '2011-03-01', '2011-04-01', '2011-05-01',
               '2011-06-01', '2011-07-01', '2011-08-01', '2011-09-01',
               '2011-10-01', '2011-11-01', '2011-12-01', '2012-01-01',
               '2012-02-01', '2012-03-01', '2012-04-01', '2012-05-01',
               '2012-06-01', '2012-07-01', '2012-08-01', '2012-09-01',
               '2012-10-01', '2012-11-01', '2012-12-01', '2013-01-01',
               '2013-02-01', '2013-03-01', '2013-04-01', '2013-05-01',
               '2013-06-01', '2013-07-01', '2013-08-01', '2013-09-01',
               '2013-10-01', '2013-11-01', '2013-12-01', '2014-01-01',
               '2014-02-01', '2014-03-01', '2014-04-01', '2014-05-01',
               '2014-06-01', '2014-07-01', '2014-08-01', '2014-09-01',
               '2014-10-01', '2014-11-01', '2014-12-01', '2015-01-01',
               '2015-02-01', '2015-03-01', '2015-04-01', '2015-05-01',
               '2015-06-01', '2015-07-01', '2015-08-01', '2015-09-01',
               '2015-10-01', '2015-11-01', '2015-12-01'],
              dtype='datetime64[ns]', name='host_since', freq='MS')
```

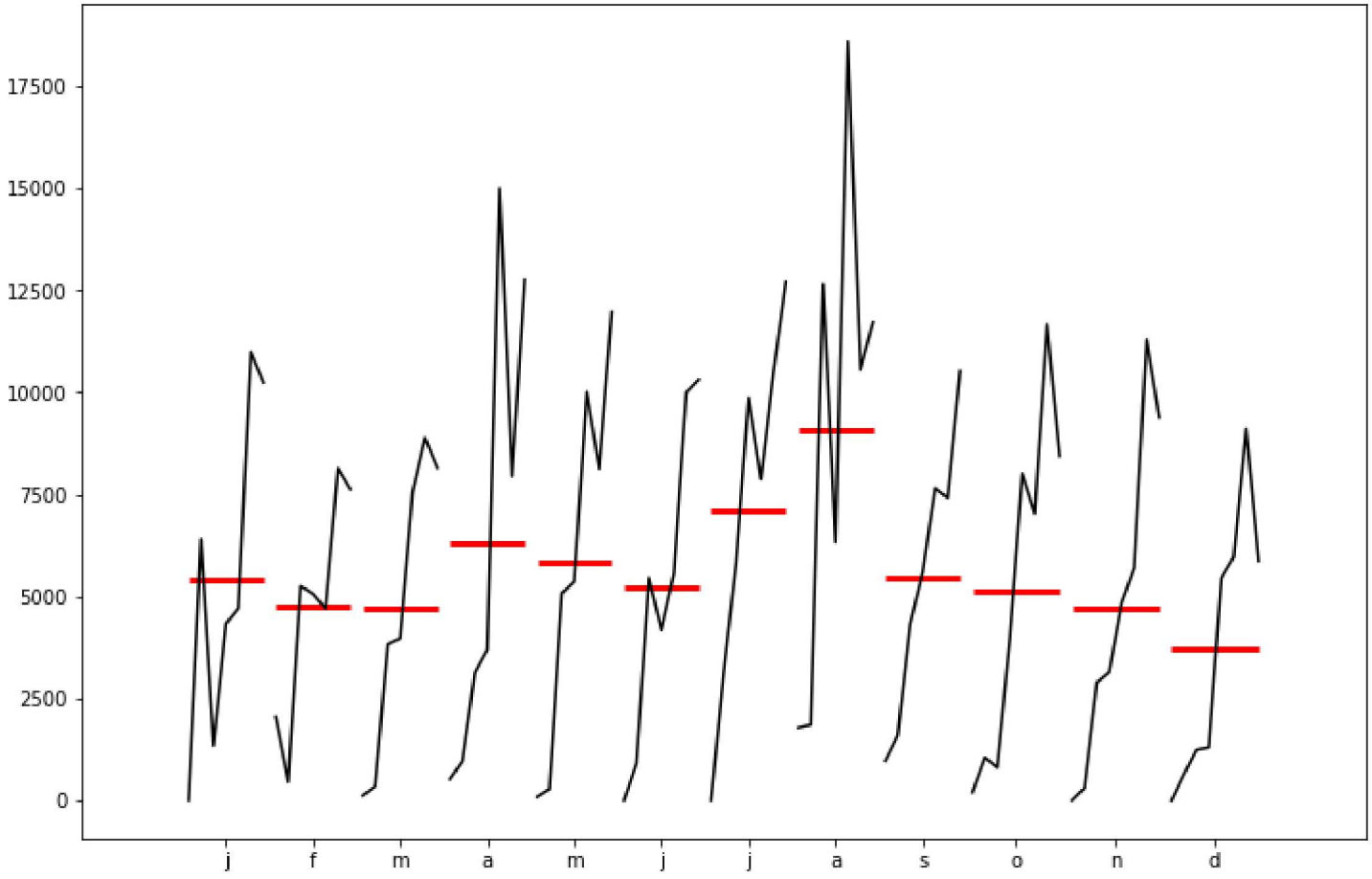
In [58]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [59]:
```
plt.rcParams["figure.figsize"] = (12,8)

result = seasonal_decompose(df_ts_sum.dropna(), model='add')
result.plot()
plt.show()
```

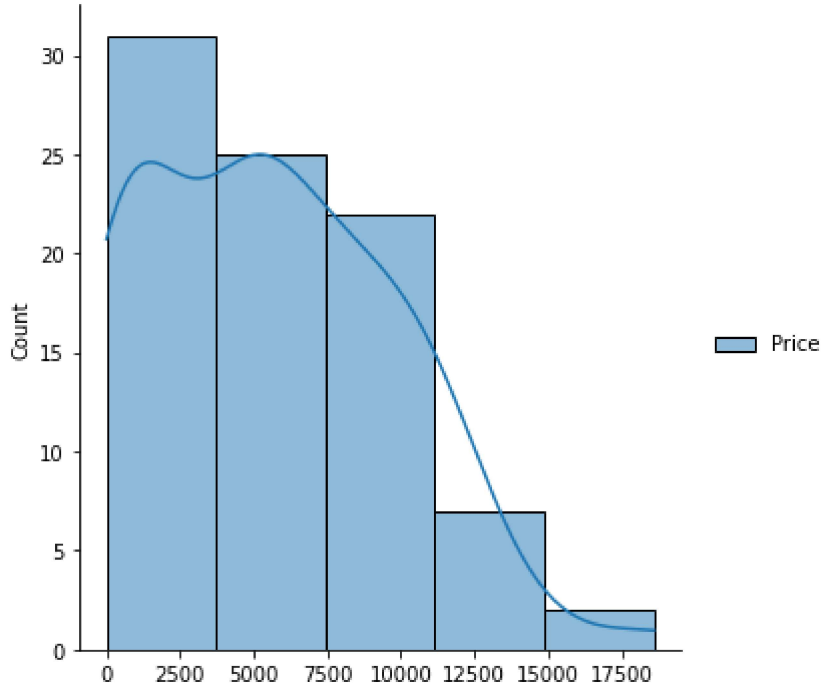


In [60]:
```
from statsmodels.graphics.tsaplots import month_plot
```

In [61]:
```
month_plot(df_ts_sum);
```

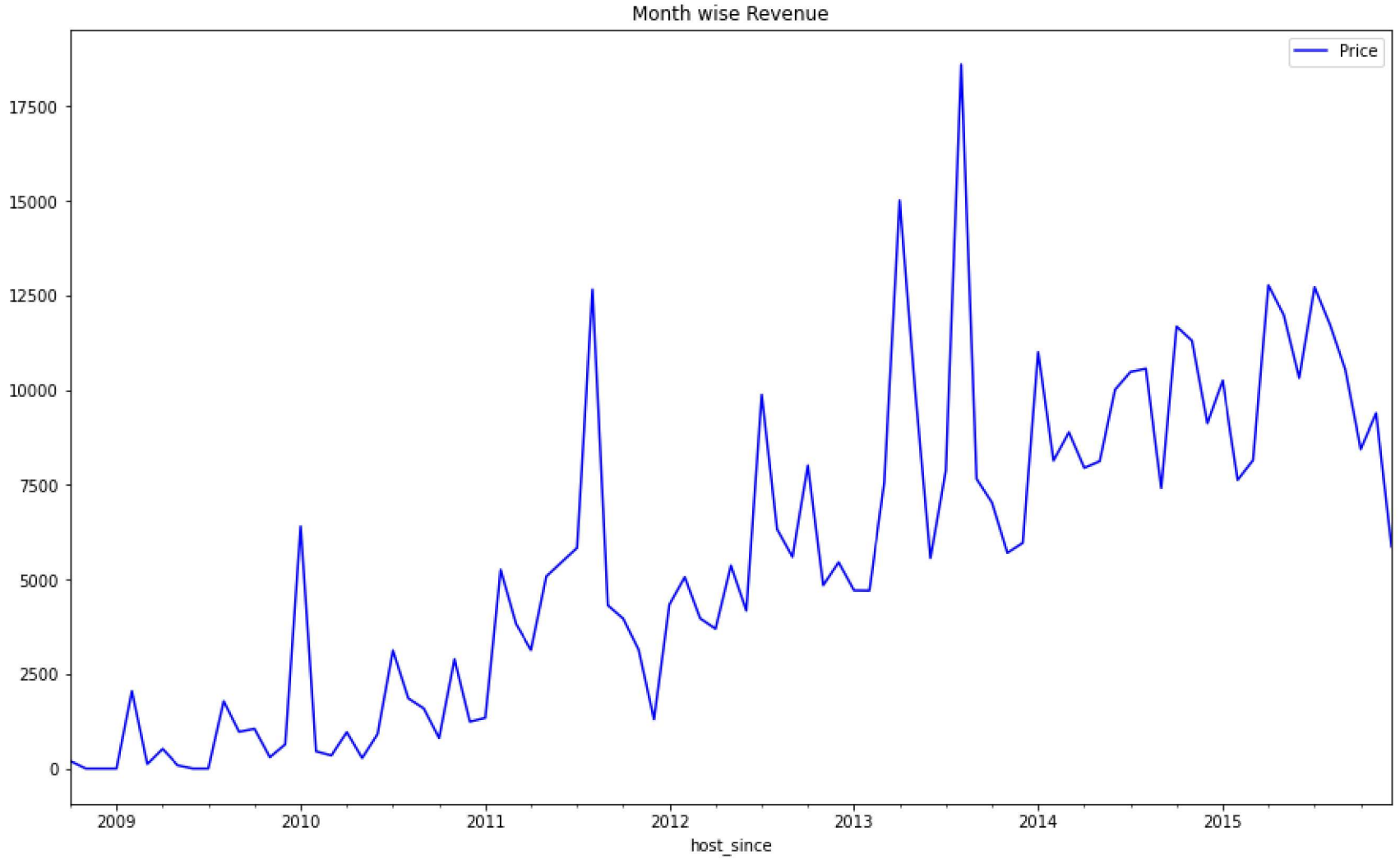


In [62]:
```
sns.displot(data=df_ts_sum,bins=5,kde=True);
```



In [63]:
```
plt.figure(dpi=150)
df_ts_sum.plot(figsize=(15,9),c='b')
plt.title('Month wise Revenue')
plt.show()
```

```
<Figure size 1800x1200 with 0 Axes>
```



In [64]:
```
len_train_data = round(len(df_ts_sum)*0.7)
```

In [65]:
```
len_train_data
```

Out[65]: 61

In [66]:
```
train_data = df_ts_sum.iloc[:len_train_data]
test_data = df_ts_sum.iloc[len_train_data:]
```

In [67]:
```
len(train_data),len(test_data)
```

Out[67]: (61, 26)

Model 1: Holt-Winters Method(Triple Smoothening Method)

In [68]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

fitted_model = ExponentialSmoothing(train_data['Price'],trend='add',seasonal='add',seasonal_periods=12).fit()
```

```
C:\Users\91630\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:920: ConvergenceWarning: Optim
ization failed to converge. Check mle_retvals.
  warnings.warn(
```

In [69]:
```
fitted_model.summary()
```

Out[69]:

ExponentialSmoothing Model Results

Dep. Variable:	Price	**No. Observations:**	61
Model:	ExponentialSmoothing	**SSE**	378539433.204
Optimized:	True	**AIC**	986.098
Trend:	Additive	**BIC**	1019.872
Seasonal:	Additive	**AICC**	1002.384
Seasonal Periods:	12	**Date:**	Wed, 27 Jul 2022
Box-Cox:	False	**Time:**	11:40:41
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.0757143	alpha	True
smoothing_trend	0.0757143	beta	True
smoothing_seasonal	0.1026984	gamma	True
initial_level	3510.0000	l.0	True
initial_trend	84.659722	b.0	True
initial_seasons.0	-3312.0000	s.0	True
initial_seasons.1	-3510.0000	s.1	True
initial_seasons.2	-3510.0000	s.2	True
initial_seasons.3	-3510.0000	s.3	True
initial_seasons.4	-1462.0000	s.4	True
initial_seasons.5	-3392.0000	s.5	True
initial_seasons.6	-2990.0000	s.6	True
initial_seasons.7	-3425.0000	s.7	True
initial_seasons.8	-3510.0000	s.8	True
initial_seasons.9	-3510.0000	s.9	True

initial_seasons.10	-1730.0000	s.10	True
initial_seasons.11	-2540.0000	s.11	True

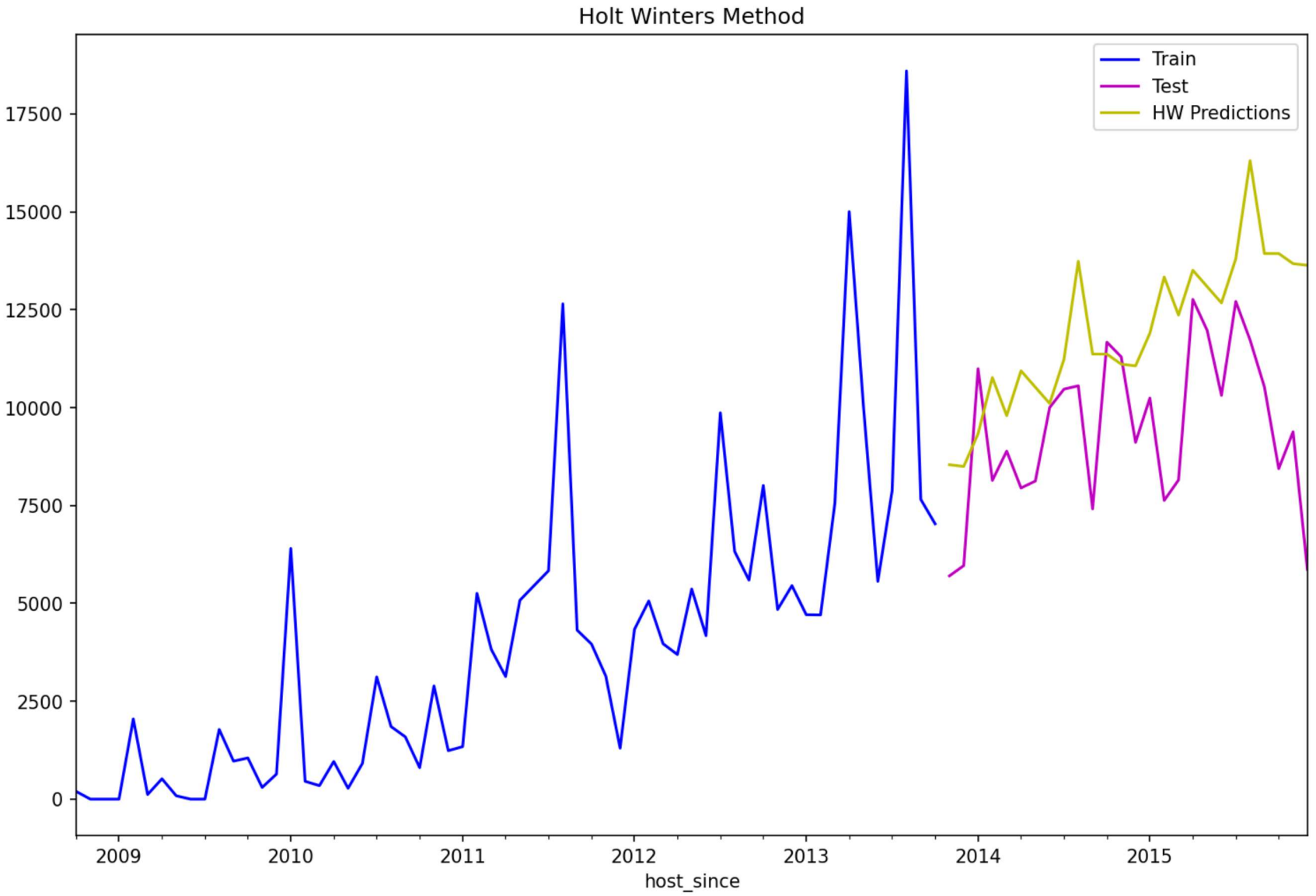
In [70]:
```
hw_aic = round(fitted_model.aic,1)
```

In [71]:
```
hw_aic
```

Out[71]: 986.1

In [72]:
```
hw_pred = fitted_model.forecast(26).rename('HW Predictions')
```

In [73]:
```
plt.figure(dpi=150)
train_data['Price'].plot(legend=True,label='Train',figsize=(12,8),c='b')
test_data['Price'].plot(legend=True,label='Test',c='m')
hw_pred.plot(legend=True,c='y')
plt.title('Holt Winters Method');
```



In [74]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [75]:
```
MAE_hw = mean_absolute_error(test_data['Price'],hw_pred)
```

In [76]:
```
MAE_hw
```

Out[76]: 2647.9869607347637

In [77]:
```
#RMSE
RMSE_hw = np.sqrt(mean_squared_error(test_data['Price'],hw_pred))
```

In [78]:
```
RMSE_hw
```

Out[78]:
```
3241.7150387620304
```

In [79]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf # for determining (p,q) orders
from pmdarima import auto_arima # for determining ARIMA orders
```

In [80]:
```
from statsmodels.tsa.stattools import adfuller

def adf_test(series):
    result=adfuller(series)
    print('ADF Statistics: {}'.format(result[0]))
    print('p- value: {}'.format(result[1]))
    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root an
    else:
        print("Weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationa
```

In [81]:
```
adf_test(df_ts_sum['Price'])
```

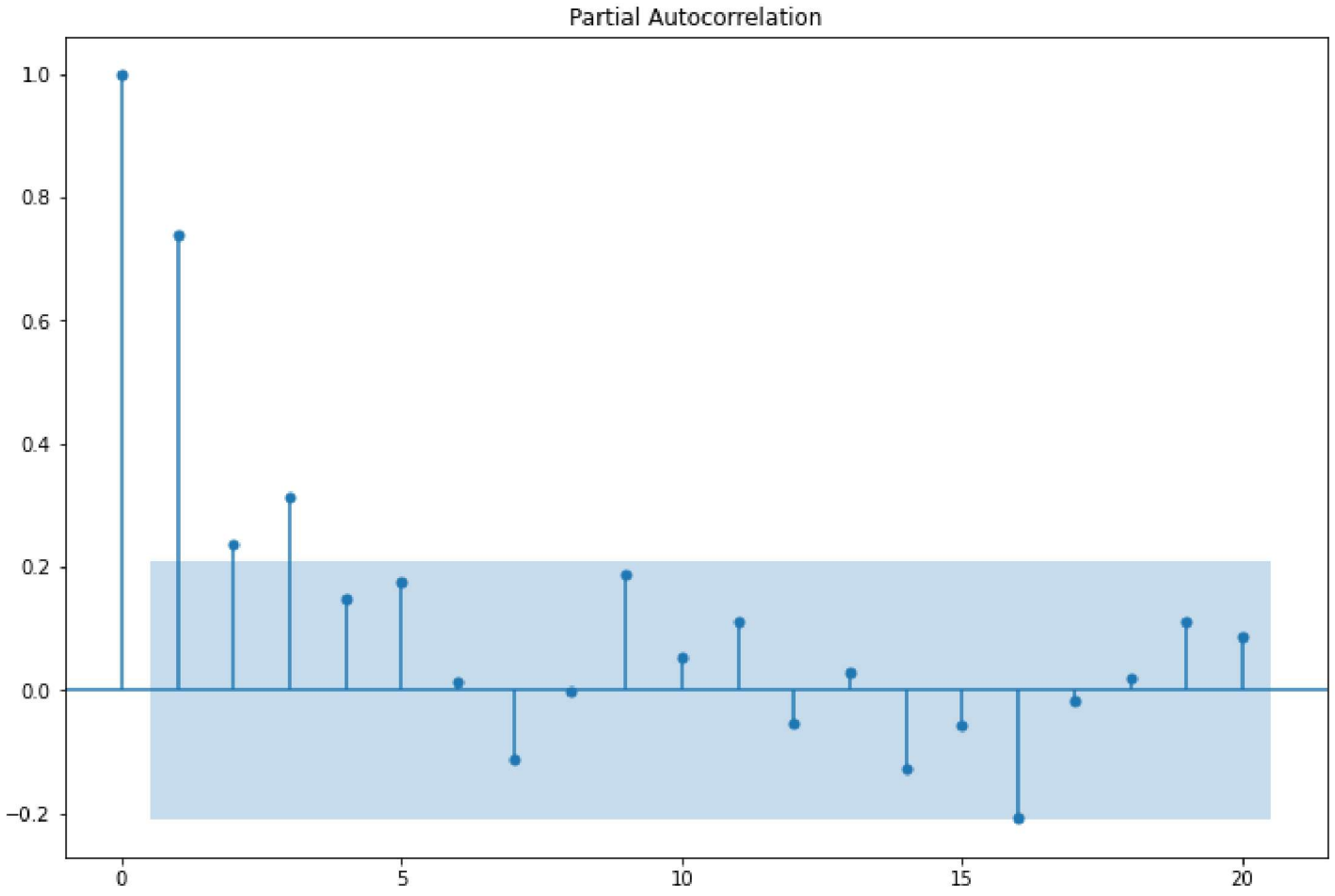
```
ADF Statistics: -1.4181405962348084
p- value: 0.5735490558649162
Weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

auto-ARIMA for best values of p,d,q

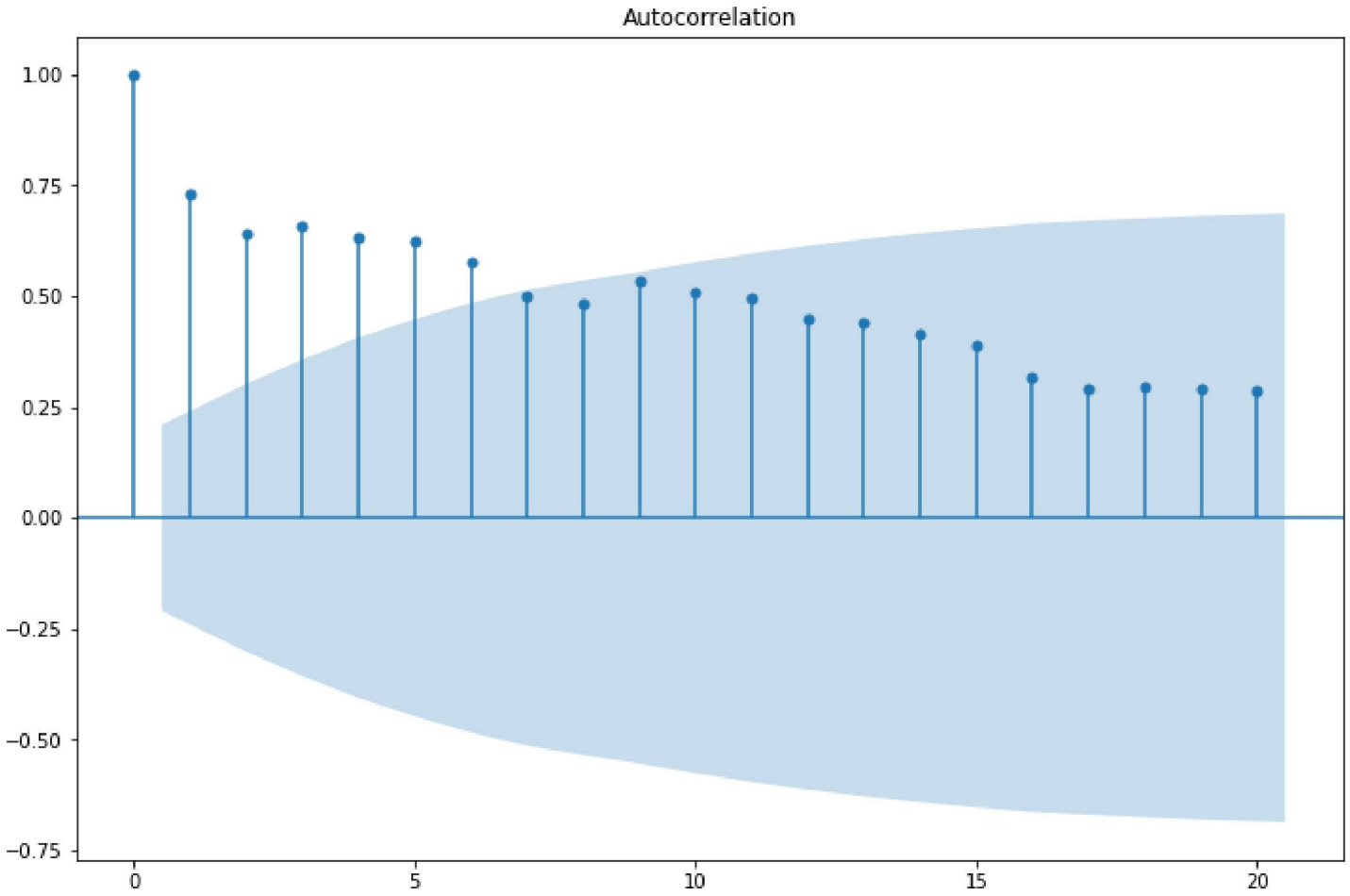
In [82]:
```
stepwise_fit = auto_arima(df_ts_sum['Price'],start_p=0,start_q=0,max_p=10,max_q=10,trace=True,seasonal=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1627.700, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1618.562, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.05 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1625.746, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1607.603, Time=0.02 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1605.781, Time=0.02 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=1603.498, Time=0.02 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=1604.617, Time=0.04 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1595.928, Time=0.15 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.14 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.28 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.21 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.21 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=1601.421, Time=0.04 sec

Best model:  ARIMA(3,1,1)(0,0,0)[0] intercept
Total fit time: 1.416 seconds
```

In [83]:
```
plot_pacf(df_ts_sum['Price'],lags=20);
```



In [84]:
```
plot_acf(df_ts_sum['Price'],lags=20);
```



In [85]:
```
p = 3
d = 1
q = 1
```

Model 2: SARIMA

In [86]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [87]:
```
import itertools
from itertools import product
from tqdm import tqdm_notebook
```

In [88]:
```
def optimize_SARIMA(parameters_list, d, D, s, data):
    """
        Return dataframe with parameters, corresponding AIC and SSE
        parameters_list - list with (p, q, P, Q) tuples
        d - integration order
        D - seasonal integration order
        s - length of season
    """
    results = []
    for param in tqdm_notebook(parameters_list):
        try:
            model = SARIMAX(data, order=(param[0], d, param[1]), seasonal_order=(param[2], D, param[3], s)).fit(
        except:
            continue
        aic = model.aic
        results.append([param, aic])
    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,q,P,Q)', 'AIC']
    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)
    return result_df
```

In [89]:
```
p = range(0, 4, 1)
d = 1
q = range(0, 4, 1)
P = range(0, 4, 1)
D = 1
Q = range(0, 4, 1)
s = 12
```

In [90]:
```
parameters = product(p, q, P, Q)
parameters_list = list(parameters)
```

In [91]:
```
# result_df = optimize_SARIMA(parameters_list, 0, 0, 12, train_data['Price'])
```

In [92]:
```
p = 0
d = 1
q = 2
P = 0
D = 1
Q = 2
s = 12
```

In [93]:
```
sarima_model = SARIMAX(train_data['Price'],order=(p,d,q),seasonal_order=(P,D,Q,s))
sarima_result = sarima_model.fit()
```

In [94]:
```
sarima_result.summary()
```

Out[94]:

SARIMAX Results

Dep. Variable:	Price	**No. Observations:**	61
Model:	SARIMAX(0, 1, 2)x(0, 1, 2, 12)	**Log Likelihood**	-447.381
Date:	Wed, 27 Jul 2022	**AIC**	904.763
Time:	11:40:45	**BIC**	914.119
Sample:	10-01-2008	**HQIC**	908.298
	- 10-01-2013		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.7047	0.244	-2.885	0.004	-1.183	-0.226
ma.L2	-0.2221	0.265	-0.839	0.402	-0.741	0.297
ma.S.L12	-0.9681	0.418	-2.315	0.021	-1.788	-0.148
ma.S.L24	0.6737	0.542	1.243	0.214	-0.389	1.736
sigma2	5.965e+06	2.25e+06	2.650	0.008	1.55e+06	1.04e+07

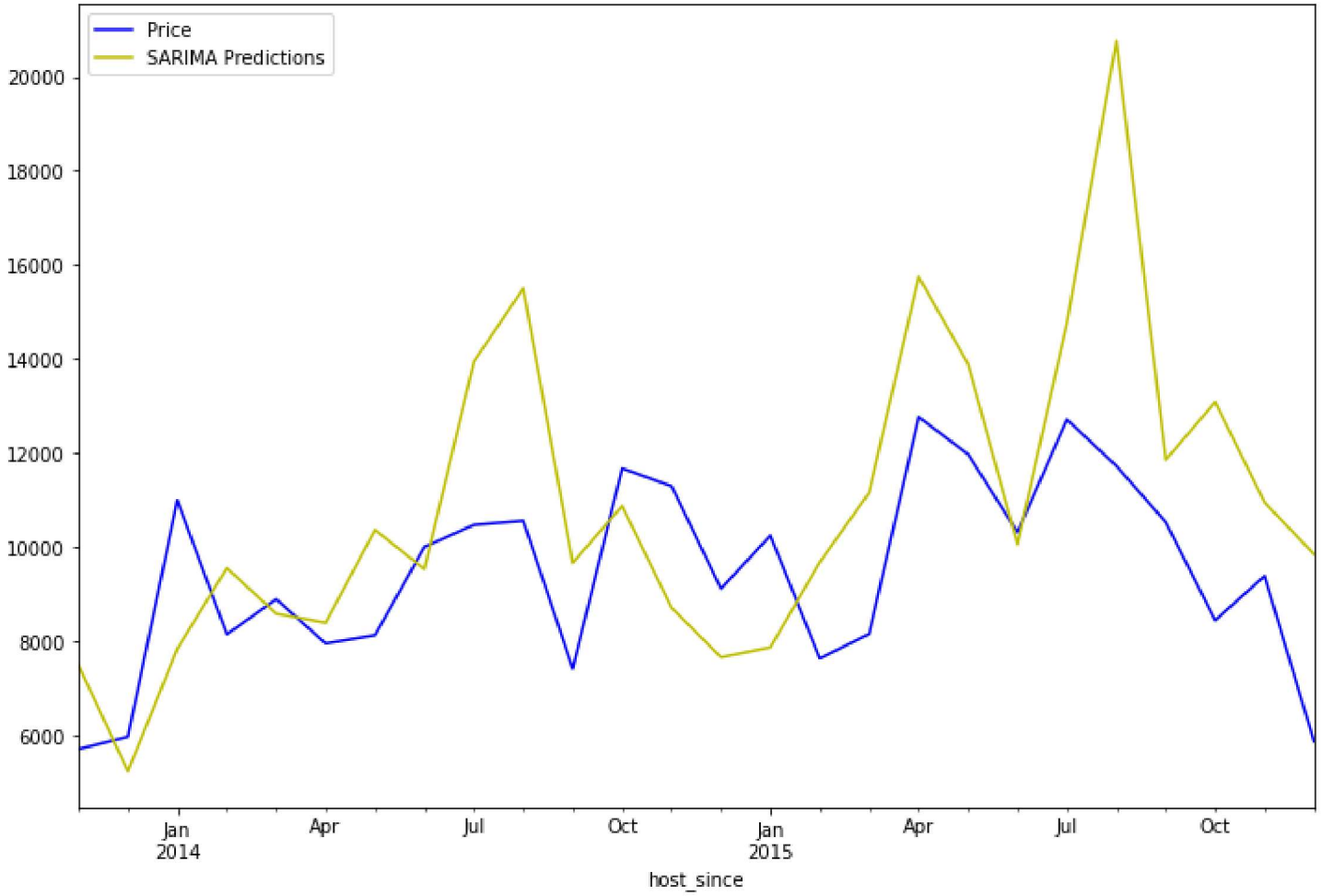
Ljung-Box (L1) (Q):	0.19	**Jarque-Bera (JB):**	18.33
Prob(Q):	0.66	**Prob(JB):**	0.00
Heteroskedasticity (H):	3.64	**Skew:**	1.24
Prob(H) (two-sided):	0.01	**Kurtosis:**	4.72

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

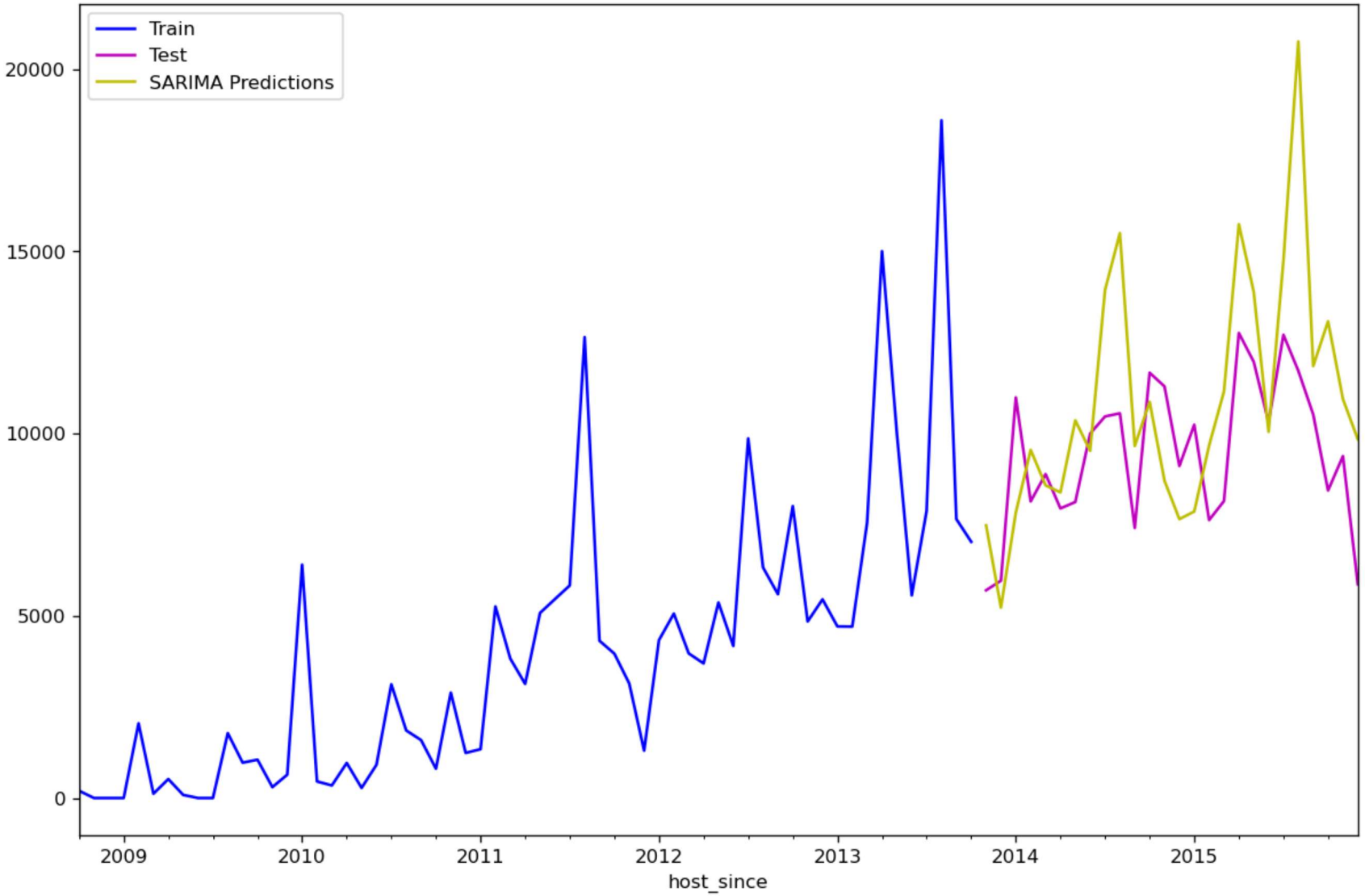
In [95]:
```
start = len(train_data)
end = len(train_data) + len(test_data) - 1
```

In [96]:
```
sarima_pred = sarima_result.predict(start,end).rename("SARIMA Predictions")
```

In [97]:
```
plt.figure(figsize=(9,6))
test_data['Price'].plot(figsize=(12,8),legend=True,c='b')
sarima_pred.plot(legend=True,c='y');
```



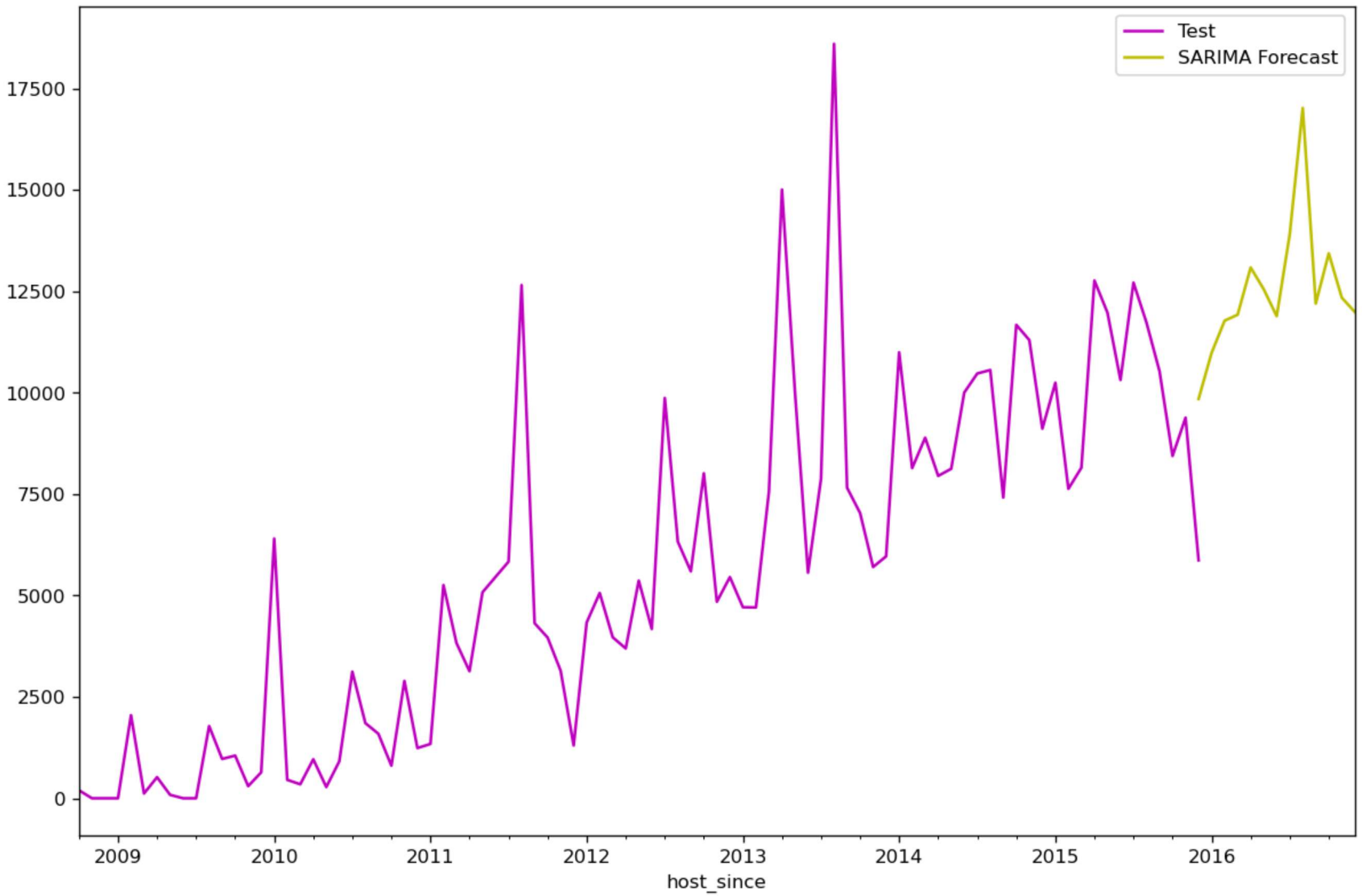
In [98]:
```
plt.figure(dpi=120)
train_data['Price'].plot(legend=True,label='Train',figsize=(12,8),c='b')
test_data['Price'].plot(legend=True,label='Test',c='m')
sarima_pred.plot(legend=True,c='y');
```



In [99]:
```
sarima_model_final = SARIMAX(df_ts_sum['Price'],order=(p,d,q),seasonal_order=(P,D,Q,s))
sarima_result = sarima_model_final.fit()
```

In [100]:
```
sarima_fcast = sarima_result.predict('2015-12-01','2016-12-01').rename("SARIMA Forecast")
```

In [101]:
```
plt.figure(dpi=120)
df_ts_sum['Price'].plot(legend=True,label='Test',c='m')
sarima_fcast.plot(legend=True,c='y');
```



In [102]:
```
MAE_SARIMA = mean_absolute_error(test_data['Price'],sarima_pred)
MAE_SARIMA
```

Out[102]: 2356.1076533028363

In [103]:
```
RMSE_SARIMA = np.sqrt(mean_squared_error(test_data['Price'],sarima_pred))
RMSE_SARIMA
```

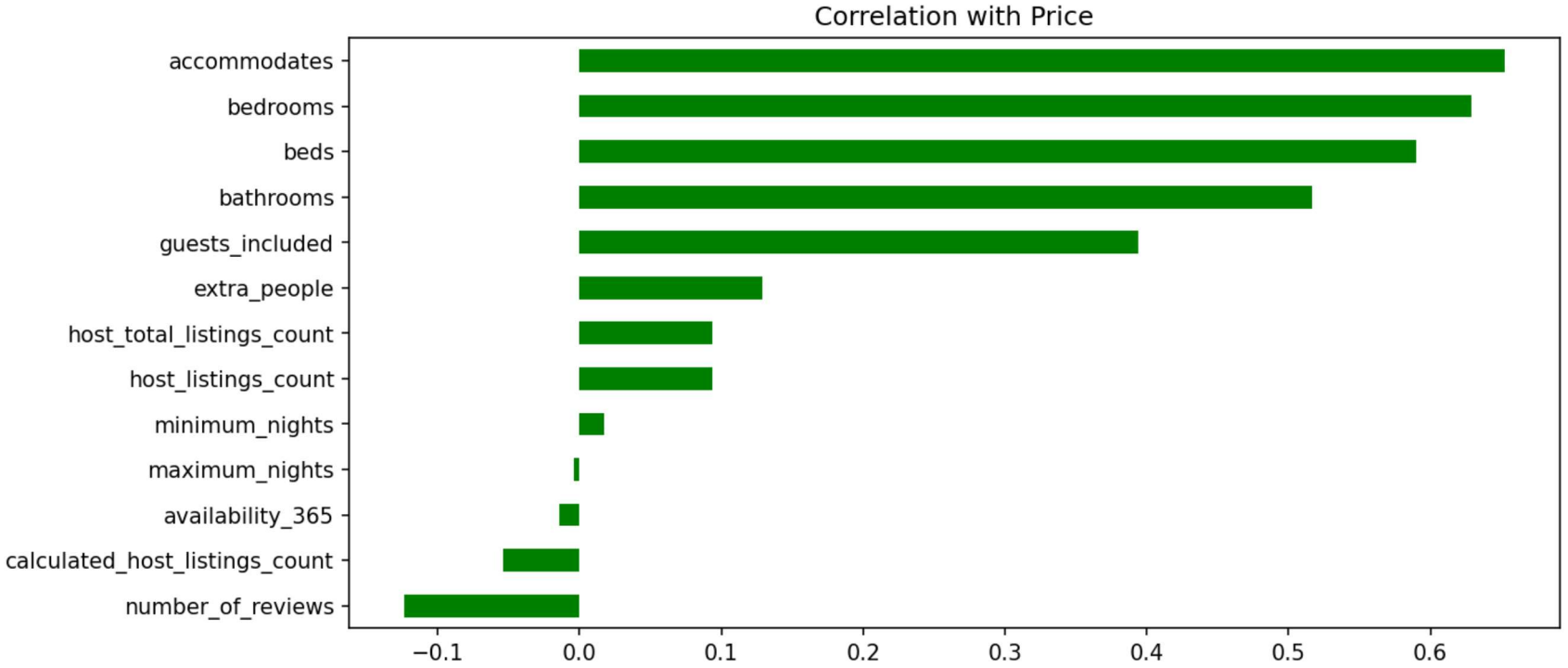
Out[103]: 2982.5159015369863

In [104]:
```
sarima_aic = round(sarima_result.aic,1)
sarima_aic
```

Out[104]: 1375.9

Model 4: SARIMAX

In [105]:
```
plt.subplots(dpi=150)
df3.corr()['Price'].drop(['host_id','id','Price','latitude','longitude']).sort_values().plot(kind='barh',color='
plt.title('Correlation with Price');
```



In [106]:
```
df_tsx = df3[['host_since','Price','accommodates','bathrooms','bedrooms','beds','guests_included']]
```

In [107]:
```
df_tsx.head()
```

Out[107]:

	host_since	Price	accommodates	bathrooms	bedrooms	beds	guests_included
0	11-08-2011	85.0	4	1.0	1.0	1.0	2
1	21-02-2013	150.0	4	1.0	1.0	1.0	1
2	12-06-2014	975.0	11	4.5	5.0	7.0	10
3	06-11-2013	100.0	3	1.0	0.0	2.0	1
4	29-11-2011	450.0	6	2.0	3.0	3.0	6

In [108]:
```
df_tsx['host_since']=pd.to_datetime(df_tsx['host_since'])
```

In [109]:
```
idx1 = df_tsx[(df_tsx['host_since'] =='2016-01-01' )|(df_tsx.host_since =='2016-02-01')|(df_tsx.host_since =='20
```

In [110]:
```
len(idx1)
```

Out[110]: 5

In [111]:
```
df_tsx.drop(idx1,inplace=True)
```

In [112]:
```
df_tsx.shape
```

Out[112]: (3788, 7)

In [113]:
```
df_tsx.sort_values('host_since',inplace=True)
```

In [114]:
```
df_tsx.set_index('host_since',inplace=True)
```

In [115]:
```
df_tsx_sum = df_tsx.resample(rule='MS').sum()
```

In [116]:
```
df_tsx_sum.index.freq = 'MS'
```

In [117]:
```
df_tsx_sum.head()
```

Out[117]:

host_since	Price	accommodates	bathrooms	bedrooms	beds	guests_included
2008-10-01	198.0	6	3.0	3.0	3.0	4
2008-11-01	0.0	0	0.0	0.0	0.0	0
2008-12-01	0.0	0	0.0	0.0	0.0	0
2009-01-01	0.0	0	0.0	0.0	0.0	0
2009-02-01	2048.0	61	23.5	24.0	32.0	43

In [118]:
```
train = df_tsx_sum.iloc[:len_train_data]
test = df_tsx_sum.iloc[len_train_data:]
```

In [119]:
```
len(train),len(test)
```

Out[119]:
```
(61, 26)
```

In [120]:
```
exog_train = train[['accommodates','bathrooms','bedrooms','beds','guests_included']]
exog_test = test[['accommodates','bathrooms','bedrooms','beds','guests_included']]
```

In [121]:
```
sarimax_model = SARIMAX(train.Price, order=(p,d,q), seasonal_order=(P,D,Q,s), exog = exog_train.values).fit()
```

```
C:\Users\91630\anaconda3\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likeliho
od optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [122]:
```
sarimax_model.summary()
```

Out[122]:

SARIMAX Results

Dep. Variable:	Price	**No. Observations:**	61
Model:	SARIMAX(0, 1, 2)x(0, 1, 2, 12)	**Log Likelihood**	-387.306
Date:	Wed, 27 Jul 2022	**AIC**	794.611
Time:	11:40:48	**BIC**	813.323
Sample:	10-01-2008	**HQIC**	801.683
	- 10-01-2013		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
x1	24.4022	26.420	0.924	0.356	-27.380	76.185
x2	22.0447	49.543	0.445	0.656	-75.058	119.147
x3	24.5473	56.237	0.436	0.662	-85.676	134.770
x4	-18.0253	44.352	-0.406	0.684	-104.953	68.903
x5	4.6573	23.995	0.194	0.846	-42.372	51.686
ma.L1	-0.9174	0.554	-1.655	0.098	-2.004	0.169
ma.L2	-0.0815	0.391	-0.208	0.835	-0.849	0.686
ma.S.L12	-0.6764	0.580	-1.166	0.244	-1.813	0.460
ma.S.L24	0.2956	0.742	0.398	0.691	-1.159	1.751
sigma2	8.011e+05	0.000	6.4e+09	0.000	8.01e+05	8.01e+05

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	4.22
Prob(Q):	0.92	**Prob(JB):**	0.12
Heteroskedasticity (H):	4.13	**Skew:**	0.33
Prob(H) (two-sided):	0.01	**Kurtosis:**	4.29

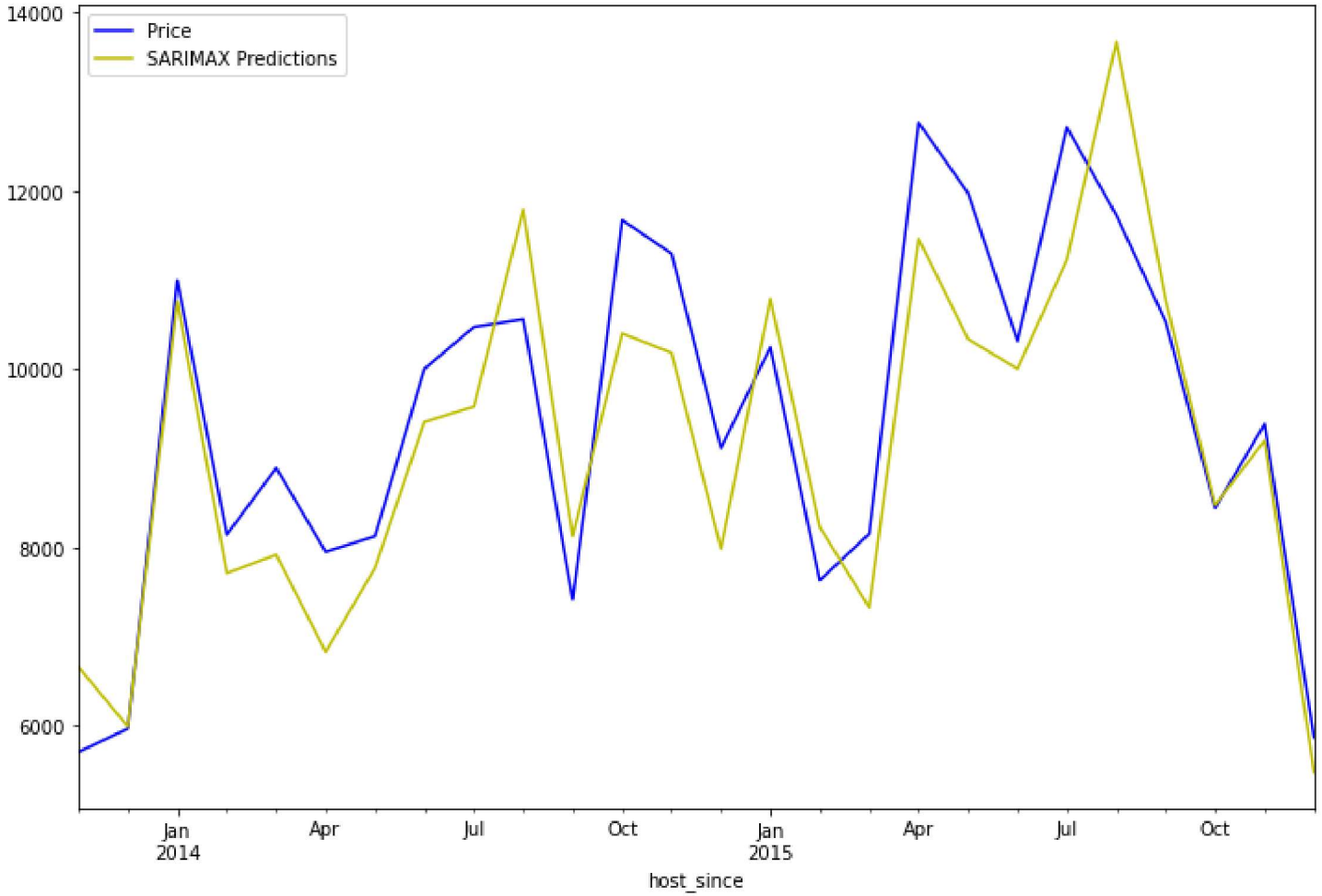
Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.21e+27. Standard errors may be unstable.

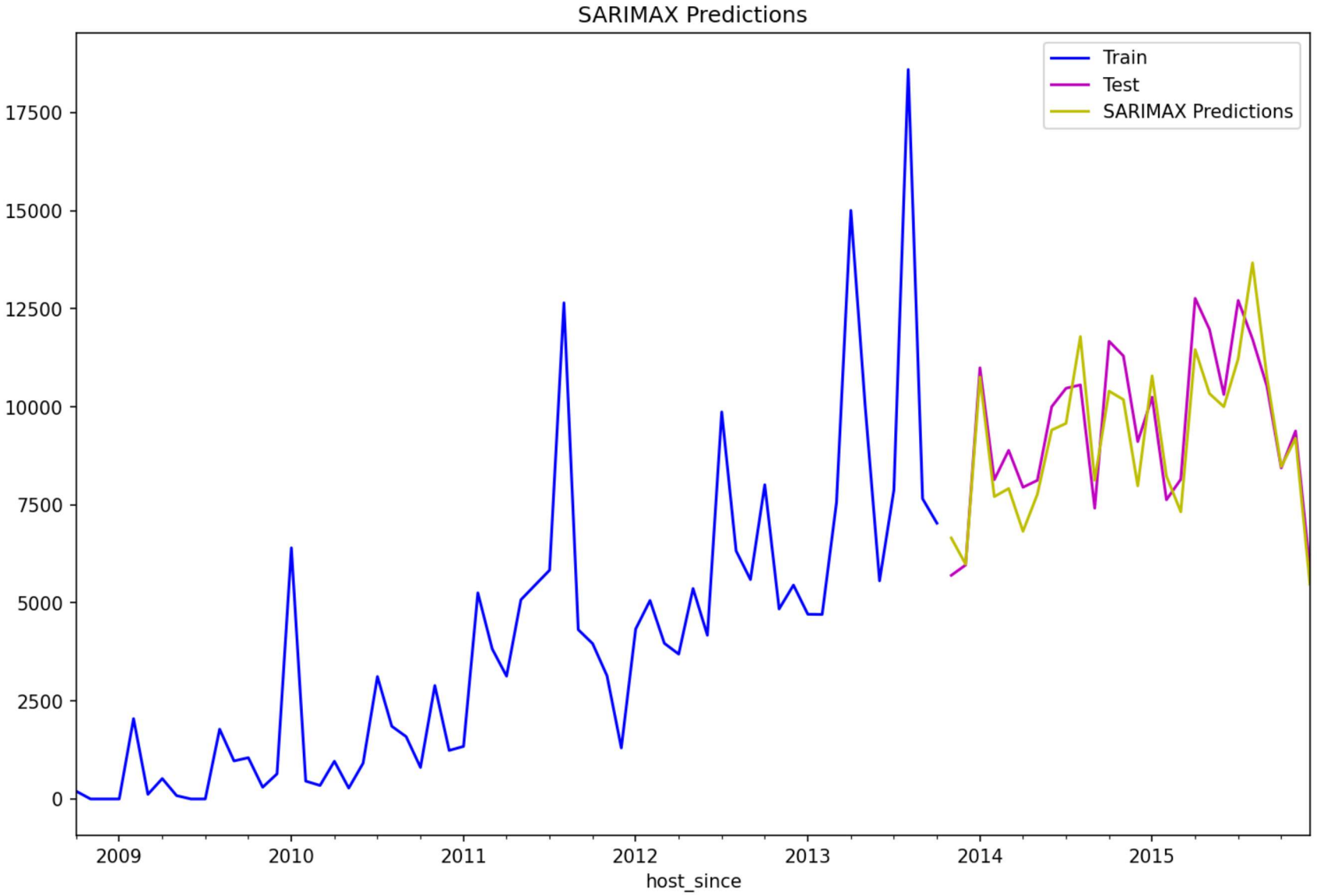
In [123]:
```
sarimax_pred = sarimax_model.predict(start,end, exog=exog_test.values).rename("SARIMAX Predictions")
```

In [124]:
```
plt.figure(figsize=(9,6))
plt.figure(figsize=(9,6))
test['Price'].plot(figsize=(12,8),legend=True,c='b')
sarimax_pred.plot(legend=True,c='y');
```

```
<Figure size 648x432 with 0 Axes>
```



In [125]:
```
plt.figure(figsize=(15,9),dpi=150)

train['Price'].plot(legend=True,label='Train',figsize=(12,8),c='b')
test['Price'].plot(legend=True,label='Test',c='m')
sarimax_pred.plot(legend=True,c='y')
plt.title('SARIMAX Predictions');
```



In [126]:
```
MAE_SARIMAX = mean_absolute_error(test['Price'],sarimax_pred)
MAE_SARIMAX
```

Out[126]: 790.809872749124

In [127]:
```
#RMSE
RMSE_SARIMAX = np.sqrt(mean_squared_error(test['Price'],sarimax_pred))
RMSE_SARIMAX
```

Out[127]: 937.9929763689117

In [128]:
```
sarimax_model_aic = round(sarimax_model.aic,1)
sarimax_model_aic
```

Out[128]: 794.6

In [129]:
```
list_aic = [hw_aic,sarima_aic,sarimax_model_aic]
list_aic
```

Out[129]: [986.1, 1375.9, 794.6]

In [130]:
```
list_mae = [MAE_hw,MAE_SARIMA,MAE_SARIMAX]
list_mae
```

Out[130]: [2647.9869607347637, 2356.1076533028363, 790.809872749124]

In [131]:
```
list_rmse = [RMSE_hw,RMSE_SARIMA,RMSE_SARIMAX]
list_rmse
```

Out[131]: [3241.7150387620304, 2982.5159015369863, 937.9929763689117]

In [132]:
```
df_final = pd.DataFrame(data=list_aic,index=['Holt-Winters','SARIMA','SARIMAX'])
```

In [133]:
```
df_final.rename(columns={0:'AIC'},inplace=True)
```

In [134]:
```
df_final['Mean Absolute Error'] = list_mae
df_final['Root Mean Squared Error'] = list_rmse
```

Conclusions:

In [135]:
```
df_final
```

Out[135]:

	AIC	Mean Absolute Error	Root Mean Squared Error
Holt-Winters	986.1	2647.986961	3241.715039
SARIMA	1375.9	2356.107653	2982.515902
SARIMAX	794.6	790.809873	937.992976

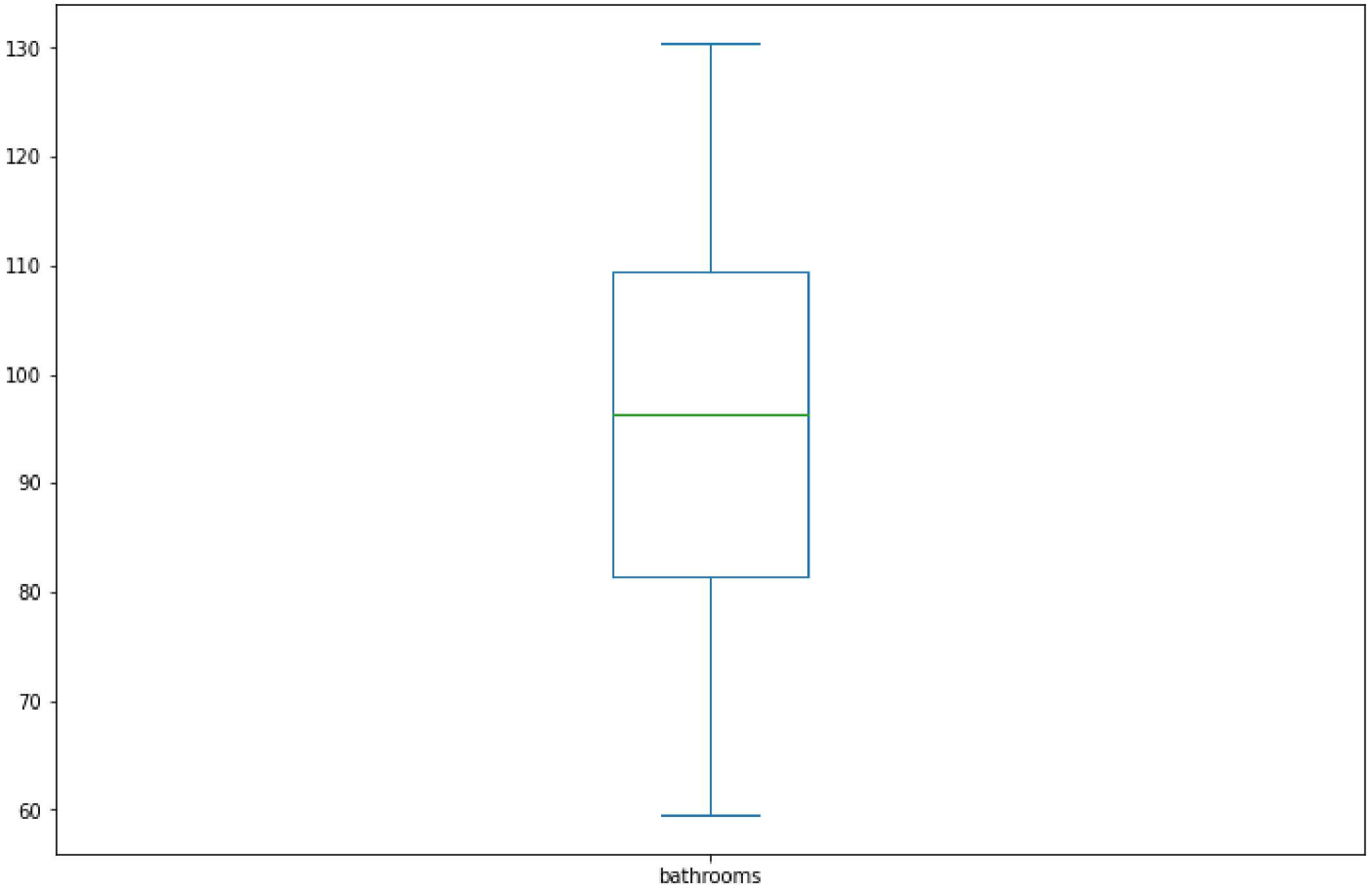
Conclusion: From the comparison DataFrame, for the Seattle dataset taken, SARIMAX model algorithm has yielded minimum error and AIC and thus we can finalize SARIMAX model for revenue forecasting.

Forecasting using SARIMAX

In [136]:
```
exog_total = df_tsx_sum.drop('Price',axis=1)
```

In [137]:
```
test['bathrooms'].plot(kind='box')
```

Out[137]: <AxesSubplot:>



In [138]:
```
forecast_dates = pd.date_range(start='2016-01-01',end='2016-12-01', freq='MS')
```

In [139]:
```
accommodates_range = np.random.randint(150,360,12)
accommodates_range
```

Out[139]:
```
array([161, 152, 306, 243, 328, 275, 169, 240, 345, 273, 331, 330])
```

In [140]:
```
bathrooms_range = np.random.randint(60,130,12)
bathrooms_range
```

Out[140]:
```
array([105, 111,  67,  78, 122,  82,  64,  67,  61,  82, 126,  61])
```

In [141]:
```
bedrooms_range = np.random.randint(58,137,12)
bedrooms_range
```

Out[141]:
```
array([129, 113,  94, 128, 101, 118,  77,  63,  85,  86, 114,  72])
```

In [142]:
```
beds_range = np.random.randint(83,191,12)
beds_range
```

Out[142]:
```
array([149, 172, 170, 108, 166,  92,  94, 178, 138, 167, 151, 145])
```

In [143]:
```
guests_included_range = np.random.randint(79,177,12)
guests_included_range
```

Out[143]:
```
array([ 92, 172, 100,  79, 160, 101, 151, 159,  96,  81, 130, 118])
```

In [144]:
```
exog_forecast = pd.DataFrame(data=[accommodates_range,bathrooms_range,bedrooms_range,beds_range,guests_included_
```

In [145]:
```
exog_forecast.reset_index(inplace=True)
```

In [146]:
```
exog_forecast.columns
```

Out[146]:
```
Index(['level_0', 'Accommodates', 'bathrooms', 'bedrooms', 'beds',
       'guests_included'],
      dtype='object')
```

In [147]:
```
exog_forecast.rename(columns = {'level_0':'forecast_dates'},inplace=True)
```

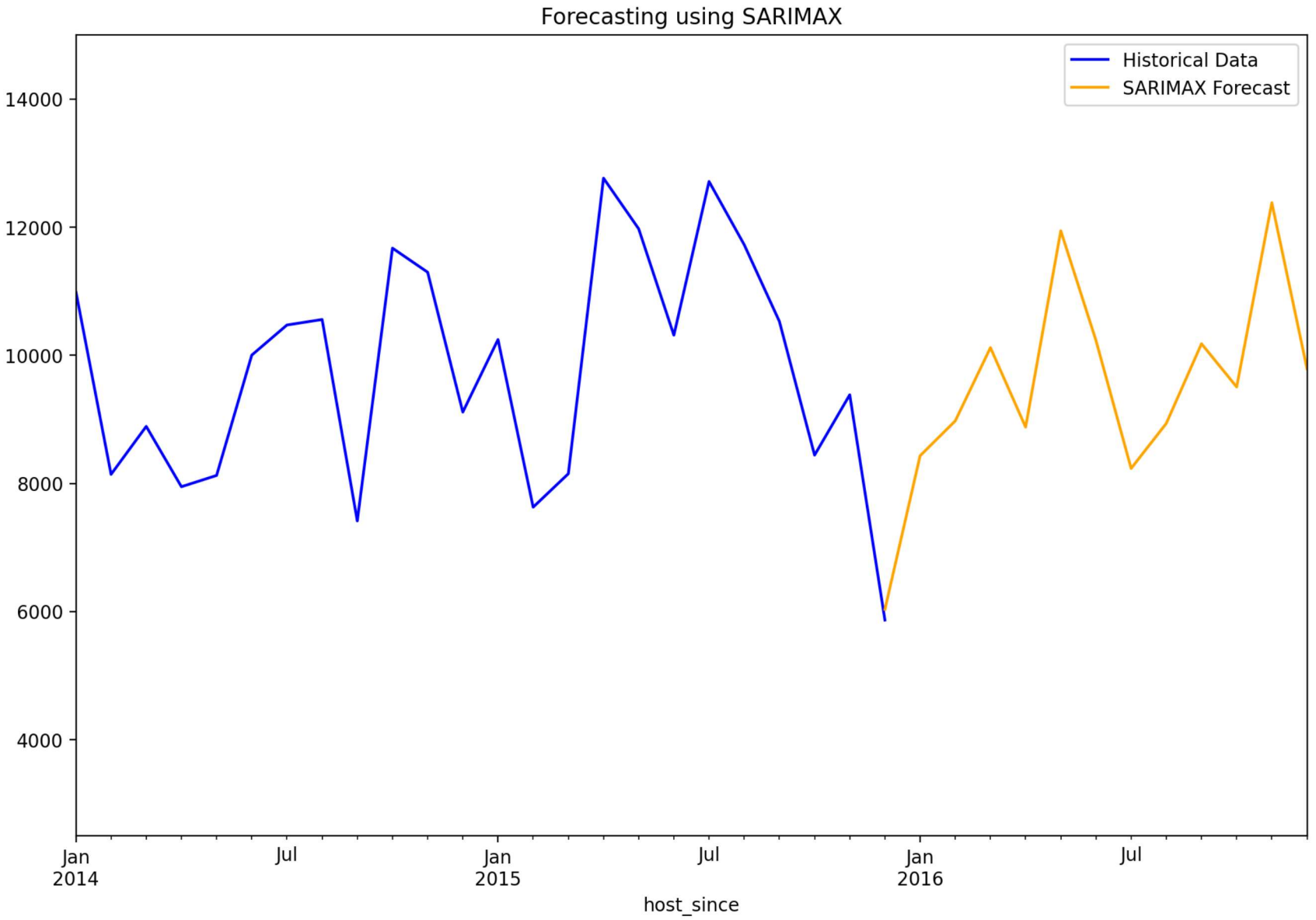
In [148]:
```
exog_forecast.set_index('forecast_dates',inplace=True)
```

In [149]:
```
sarimax_model_final = SARIMAX(df_tsx_sum.Price, order=(p,d,q), seasonal_order=(P,D,Q,s), exog = exog_total.value
```

```
C:\Users\91630\anaconda3\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likeliho
od optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [150]:
```
fcast_sarimax = sarimax_model_final.predict('2015-12-01','2016-12-01', exog=exog_forecast.values).rename("SARIMA
```

In [151]:
```
plt.figure(figsize=(15,9),dpi=200)
df_tsx_sum['Price'].plot(legend=True,label='Historical Data',figsize=(12,8),c='b')
fcast_sarimax.plot(legend=True,c='orange')
plt.title('Forecasting using SARIMAX')
plt.xlim('2014-01-01','2016-12-01')
plt.ylim(2500,15000);
```



In [152]:
```
fcast_sarimax
```

Out[152]:
```
2015-12-01     6033.243210
2016-01-01     8430.855072
2016-02-01     8975.355619
2016-03-01    10118.801330
2016-04-01     8878.246296
2016-05-01    11942.892973
2016-06-01    10231.022494
2016-07-01     8233.690222
2016-08-01     8936.942436
2016-09-01    10179.723215
2016-10-01     9505.188161
2016-11-01    12380.510220
2016-12-01     9790.515995
Freq: MS, Name: SARIMAX Forecast, dtype: float64
```

1.Identify if there are any trends and seasons when travelers book these accommodations:

From the seasonal decomposition and month_plot we can see that there is an overall upward trend showing rise in bookings of these accomodations and the seasonality shows that more bookings were done in the month's of July and August maybe due to the warm weather and especially to attend Seafair festival where people majorly come to watch the hydroplane races and the Blue Angels(a flight demonstration squadron of the United States Navy) which is a Seattle tradition.

2.Does this business model also indicate trends that shows demand for these houses ?

Yes SARIMAX model is able to indicate the trends and seasonality of the demands for the houses.

Problem_Statement_5: Prediction of property_type:

host_is_superhost, host_total_listings_count, room_type, accommodates, 'bathrooms', 'bedrooms', 'beds', 'bed_type', number_of_reviews, instant_bookable, cancellation_policy, host_response_time, availability_365, Price

In [153]:
```
df4 = df3.drop(['id','host_id','latitude','longitude'],axis=1)
```

In [154]:
```
df5 = df4[['host_is_superhost', 'host_total_listings_count', 'room_type', 'accommodates', 'bathrooms', 'bedrooms
       'number_of_reviews', 'instant_bookable', 'cancellation_policy', 'host_response_time', 'availability_365'
```

In [155]:
```
df5['property_type'].replace(to_replace =['Cabin', 'Condominium', 'Camper/RV',
       'Bungalow', 'Townhouse', 'Loft', 'Boat', 'Bed & Breakfast',
       'Other', 'Dorm', 'Treehouse', 'Yurt', 'Chalet', 'Tent'],
                 value ="Others",inplace=True)
```

In [156]:
```
df5.property_type.unique()
```

Out[156]:
```
array(['Apartment', 'House', 'Others'], dtype=object)
```

In [157]:
```
df5.property_type.value_counts()
```

Out[157]:
```
House        1723
Apartment    1696
Others        374
Name: property_type, dtype: int64
```

In [158]:
```
df5.select_dtypes(exclude=['int','float']).columns
```

Out[158]:
```
Index(['host_is_superhost', 'room_type', 'bed_type', 'instant_bookable',
       'cancellation_policy', 'host_response_time', 'property_type'],
      dtype='object')
```

In [159]:
```
df_dummies = pd.get_dummies(df5[['host_is_superhost', 'room_type', 'bed_type', 'instant_bookable',
       'cancellation_policy', 'host_response_time']],drop_first=True)
```

In [160]:
```
df6 = pd.concat([df5,df_dummies],axis=1).drop(['host_is_superhost', 'room_type', 'bed_type', 'instant_bookable',
       'cancellation_policy', 'host_response_time'],axis=1)
```

In [161]:
```
df6.columns
```

Out[161]:
```
Index(['host_total_listings_count', 'accommodates', 'bathrooms', 'bedrooms',
       'beds', 'number_of_reviews', 'availability_365', 'Price',
       'property_type', 'host_is_superhost_t', 'room_type_Private room',
       'room_type_Shared room', 'bed_type_Couch', 'bed_type_Futon',
       'bed_type_Pull-out Sofa', 'bed_type_Real Bed', 'instant_bookable_t',
       'cancellation_policy_moderate', 'cancellation_policy_strict',
       'host_response_time_a few days or more',
       'host_response_time_within a day',
       'host_response_time_within a few hours',
       'host_response_time_within an hour'],
      dtype='object')
```

In [162]:
```
X = df6.drop('property_type',axis=1)
y = df6['property_type']
```

Feature Selector Algorithms

In [163]:
```
def ExtraTree_Selector(X,y):
    from sklearn.ensemble import ExtraTreesClassifier
    model = ExtraTreesClassifier()
    model.fit(X, y)
    print(model.feature_importances_)
    feat_importances = pd.Series(model.feature_importances_, index=X.columns)
    feat_importances.sort_values().plot(kind='barh');
    plt.show()
    list1=feat_importances.keys().to_list()
    return list1,feat_importances
```

In [164]:
```
def forward_selector(X,y,k):
    from sklearn.ensemble import RandomForestClassifier
    from mlxtend.feature_selection import SequentialFeatureSelector
    forward_feature_selector = SequentialFeatureSelector(RandomForestClassifier(n_jobs=-1),
              k_features=k,
              forward=True,
              verbose=2,
              scoring='accuracy',
              cv=5)
    fselector = forward_feature_selector.fit(X,y)
    print(fselector.k_feature_names_)
```

In [165]:
```
def chi2_selector(X,y,k):
    from sklearn.feature_selection import SelectKBest, chi2
    chi2_features = SelectKBest(chi2, k = k)
    X_kbest_features = chi2_features.fit_transform(X, y)
    mask=chi2_features.get_support()
    new_feature=[]
    for bool,feature in zip(mask,X.columns):
        if bool:
            new_feature.append(feature)
    list3=new_feature
    print(list3)
```

Classifiers

logistic regression

In [287]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix, ConfusionMatrixDisplay
# cm = confusion_matrix(pred_labels, true_labels, labels=range(len(label_list)))
# disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_list)
# disp.plot()
```

In [288]:
```
def log_function(X,y):
    scaler = StandardScaler()
    penalty = ['l1', 'l2']
    C = np.logspace(0, 1,10)
    param_grid = {'penalty':penalty,'C':C}
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
    scaled_X_train = scaler.fit_transform(X_train)
    scaled_X_test = scaler.transform(X_test)
    log_model = LogisticRegression(max_iter=5000)
    grid_model1 = GridSearchCV(estimator=log_model,param_grid=param_grid,cv=5)
    grid_model1.fit(scaled_X_train,y_train)
    print(grid_model1.best_score_)
    print(f'Best Parameters: {grid_model1.best_params_}\n')
    y_pred1 = grid_model1.predict(scaled_X_test)
    print(classification_report(y_test,y_pred1))
```

KNN

In [306]:
```
def knn_function(X,y):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score
    scaler = StandardScaler()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
    knn = KNeighborsClassifier()
    operations = [('scaler',scaler),('knn',knn)]
    from sklearn.pipeline import Pipeline
    pipe = Pipeline(operations)   ## our model to be passed as estimator for GridSearchCV
    k_values = list(range(1,20))
    param_grid = {'knn__n_neighbors': k_values}
    full_cv_classifier = GridSearchCV(pipe,param_grid,cv=5,scoring='accuracy')
    full_cv_classifier.fit(X_train,y_train)
    full_cv_classifier.best_estimator_.get_params()
    ## Elbow Method to find optimal K

    test_error_rates = []


    for k in range(1,20):
        knn_model = KNeighborsClassifier(n_neighbors=k)
        knn_model.fit(X_train,y_train)

        y_pred_test = knn_model.predict(X_test)

        test_error = 1 - accuracy_score(y_test,y_pred_test)
        test_error_rates.append(test_error)

    plt.figure(figsize=(10,6),dpi=200)
    plt.plot(range(1,20),test_error_rates,label='Test Error')
    plt.legend()
    plt.ylabel('Error Rate')
    plt.xlabel("K Value")


    knn= KNeighborsClassifier(n_neighbors=19)
    operations = [('scaler',scaler),('knn',knn)]

    pipe = Pipeline(operations)
    pipe.fit(X_train,y_train)
    pipe_pred = pipe.predict(X_test)
```

```
    print(classification_report(y_test,pipe_pred))
    print(accuracy_score(y_test, pipe_pred))
```

SVC

In [290]:
```
def svc_function(X,y):
    from sklearn.svm import SVC # Support Vector Classifier
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
    from sklearn.preprocessing import StandardScaler  # Using standardization for scaling feature
    scaler = StandardScaler()          # Created an instance of our StandardScaler()
    scaled_X_train = scaler.fit_transform(X_train)        # Fit-Transforming our training set
    scaled_X_test = scaler.transform(X_test)      # Transforming our test set
    svm = SVC()
    param_grid1 = {'C':[0.001,0.01,0.1,0.5,1],'gamma':['scale','auto']}
    grid_model2 = GridSearchCV(svm,param_grid1,cv=5)
    grid_model2.fit(scaled_X_train,y_train)
    print(grid_model2.best_params_)
    y_pred2 = grid_model2.predict(scaled_X_test)
    print(classification_report(y_test,y_pred2))
    print(grid_model2.best_score_)
```

RandomForestClassifier .

In [297]:
```
def randomforest(X,y):
    from sklearn.ensemble import RandomForestClassifier
    rfc = RandomForestClassifier()
    test_error = []

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
    from sklearn.preprocessing import StandardScaler  # Using standardization for scaling feature
    scaler = StandardScaler()          # Created an instance of our StandardScaler()
    scaled_X_train = scaler.fit_transform(X_train)        # Fit-Transforming our training set
    scaled_X_test = scaler.transform(X_test)      # Transforming our test set

    for n in range(1,40):
        # Use n random trees
        model = RandomForestClassifier(n_estimators=n,max_features='sqrt')
        model.fit(scaled_X_train,y_train)
        test_preds = model.predict(scaled_X_test)
        test_error.append(1-accuracy_score(test_preds,y_test))

    plt.plot(range(1,40),test_error,label='Test Error')
    plt.legend()

    n_estimators=list(range(10,40))
    max_features= list(range(2,5))
    bootstrap = [True,False]



    param_grid2 = {'n_estimators':n_estimators,'max_features':max_features,'bootstrap':bootstrap}

    grid_model3 = GridSearchCV(rfc,param_grid2,cv=5)
    grid_model3.fit(scaled_X_train,y_train)
    print(grid_model3.best_params_)
    print(classification_report(y_test,test_preds))
    print(grid_model3.best_score_)
```

GradientBoostingClassifier

In [298]:
```
def gradient_boost_function(X,y):
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
    from sklearn.preprocessing import StandardScaler  # Using standardization for scaling feature
    scaler = StandardScaler()          # Created an instance of our StandardScaler()
    scaled_X_train = scaler.fit_transform(X_train)        # Fit-Transforming our training set
    scaled_X_test = scaler.transform(X_test)      # Transforming our test set
    gb_model = GradientBoostingClassifier()
    param_grid3 = {"n_estimators":[1,5,10,20,40,100],'max_depth':[3,4,5,6]}
    grid_model4 = GridSearchCV(gb_model,param_grid3,cv=5)
    grid_model4.fit(scaled_X_train,y_train)
    print(grid_model4.best_params_)
    y_pred6 = grid_model4.predict(scaled_X_test)
    print(y_pred6)
    print(classification_report(y_test,y_pred6))
    print(grid_model4.best_score_)
```

AdaBoostClassifier

In [310]:
```
def adaboost_function(X,y):
    from sklearn.ensemble import AdaBoostClassifier

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)

    from sklearn.preprocessing import StandardScaler  # Using standardization for scaling feature
    scaler = StandardScaler()          # Created an instance of our StandardScaler()
    scaled_X_train = scaler.fit_transform(X_train)        # Fit-Transforming our training set
    scaled_X_test = scaler.transform(X_test)      # Transforming our test set

    error_rates = []
    for n in range(1,20):

        model = AdaBoostClassifier(n_estimators=n)
        model.fit(scaled_X_train,y_train)
        preds = model.predict(scaled_X_test)
        err = 1 - accuracy_score(y_test,preds)

        error_rates.append(err)
    plt.plot(range(1,20),error_rates)

    model = AdaBoostClassifier(n_estimators=17)
    model.fit(scaled_X_train,y_train)
    y_pred5 = model.predict(scaled_X_test)
    print(accuracy_score(y_test, y_pred5))
    print(classification_report(y_test,y_pred5))
```

Goal: Binary classification

In [180]:
```
df4.property_type.value_counts()
```

Out[180]:
```
House              1723
Apartment          1696
Townhouse           118
Condominium          91
Loft                 40
Bed & Breakfast      37
Cabin                21
Other                21
Camper/RV            13
Bungalow             13
Boat                  8
Tent                  5
Treehouse             3
Dorm                  2
Yurt                  1
Chalet                1
Name: property_type, dtype: int64
```

House : Townhouse, Loft, Camper/RV, Cabin, Bungalow, Boat, Treehouse, Bed&Breakfast, Tent, House, Yurt, Chalet

Apartment: Condominium, Dorm

In [187]:
```
df4.shape
```

Out[187]:
```
(3793, 40)
```

In [192]:
```
index1 = df4[df4.property_type == 'Other'].index
```

In [194]:
```
df4.drop(index1,axis=0,inplace=True)
```

In [205]:
```
df4['property_type'].replace(to_replace =['Condominium','Dorm'],
                 value ="Apartment",inplace=True)
```

In [211]:
```
df4['property_type'].replace(to_replace =['Townhouse', 'Loft', 'Camper/RV', 'Cabin', 'Bungalow', 'Boat', 'Treeho
                                          'Bed & Breakfast', 'Tent', 'House', 'Yurt', 'Chalet'],
                 value ="House",inplace=True)
```

In [212]:
```
df4.property_type.value_counts()
```

Out[212]:
```
House        1983
Apartment    1789
Name: property_type, dtype: int64
```

In [213]:
```
df4 = df4[['host_is_superhost', 'host_total_listings_count', 'room_type', 'accommodates', 'bathrooms', 'bedrooms
       'number_of_reviews', 'instant_bookable', 'cancellation_policy', 'host_response_time', 'availability_365'
```

In [214]:
```
df_dummies = pd.get_dummies(df4[['host_is_superhost', 'room_type', 'bed_type', 'instant_bookable',
      'cancellation_policy', 'host_response_time']],drop_first=True)
```

In [215]:
```
df_v = pd.concat([df4,df_dummies],axis=1).drop(['host_is_superhost', 'room_type', 'bed_type', 'instant_bookable'
      'cancellation_policy', 'host_response_time'],axis=1)
```

In [223]:
```
X = df_v.drop('property_type',axis=1)
y = df_v['property_type']
```

In [224]:
```
X.columns
```

Out[224]:
```
Index(['host_total_listings_count', 'accommodates', 'bathrooms', 'bedrooms',
       'beds', 'number_of_reviews', 'availability_365', 'Price',
       'host_is_superhost_t', 'room_type_Private room',
       'room_type_Shared room', 'bed_type_Couch', 'bed_type_Futon',
       'bed_type_Pull-out Sofa', 'bed_type_Real Bed', 'instant_bookable_t',
       'cancellation_policy_moderate', 'cancellation_policy_strict',
       'host_response_time_a few days or more',
       'host_response_time_within a day',
       'host_response_time_within a few hours',
       'host_response_time_within an hour'],
      dtype='object')
```

In [230]:
```
forward_selector(X,y,4)
```

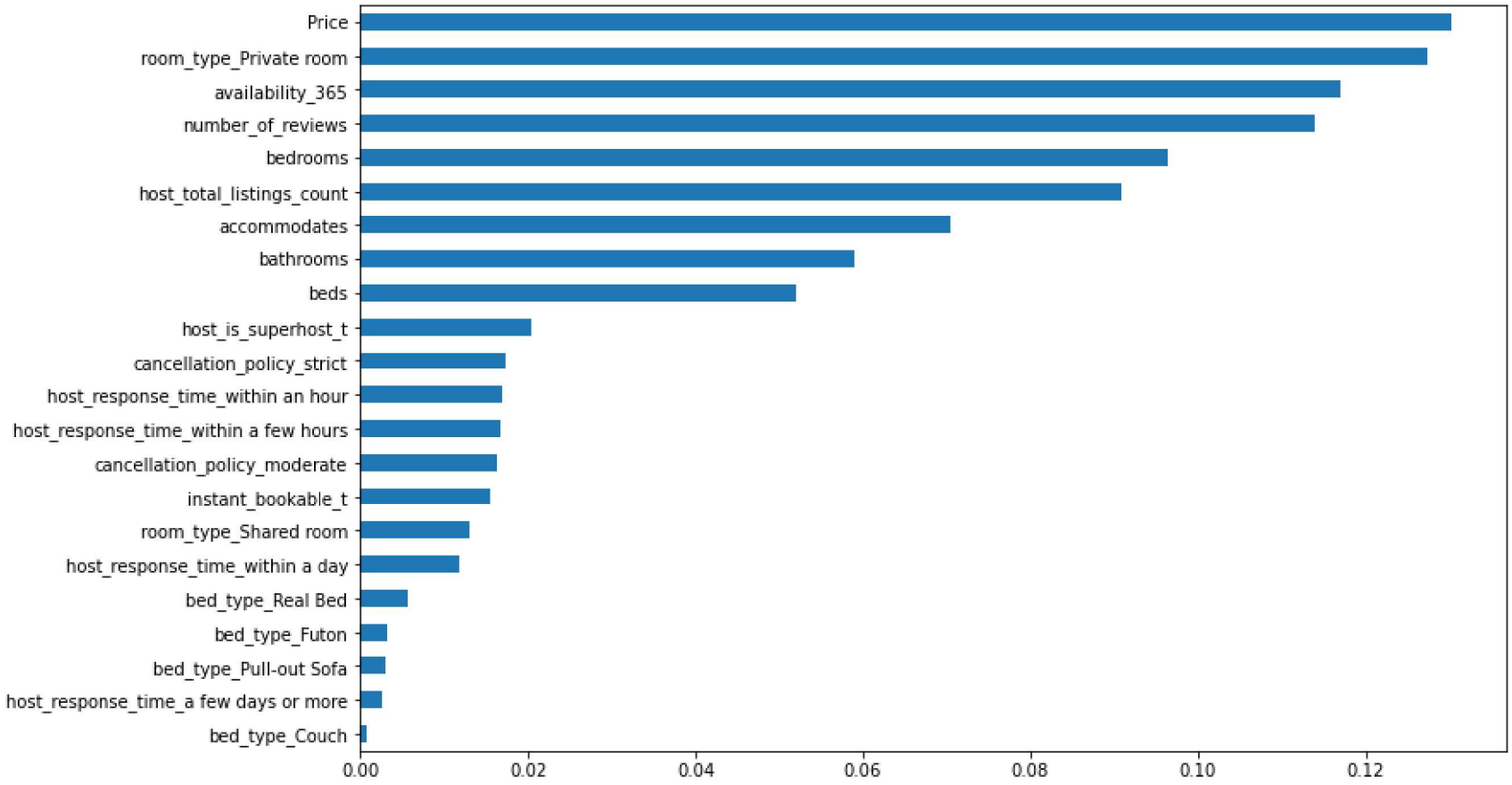
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  22 out of  22 | elapsed:   20.1s finished

[2022-07-27 12:29:18] Features: 1/4 -- score: 0.6468909304899257[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed:   17.4s finished

[2022-07-27 12:29:36] Features: 2/4 -- score: 0.7330395067366979[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   16.6s finished

[2022-07-27 12:29:52] Features: 3/4 -- score: 0.7698852916893566[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.7s remaining:    0.0s


('host_total_listings_count', 'bedrooms', 'room_type_Private room', 'room_type_Shared room')

[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:   15.4s finished

[2022-07-27 12:30:08] Features: 4/4 -- score: 0.773862314894514
```

In [231]:
```
chi2_selector(X,y,4)
```

```
['host_total_listings_count', 'number_of_reviews', 'availability_365', 'Price']
```

In [232]:
```
feat_imp = ExtraTree_Selector(X,y)
```

```
[0.09079761 0.07044933 0.05903731 0.09627979 0.05201084 0.11391642
 0.11699348 0.13029271 0.02031968 0.127365   0.01299433 0.00075352
 0.00310846 0.0030954  0.00555523 0.0154031  0.01621724 0.01735295
 0.00264159 0.01182291 0.01672922 0.01686386]
```



In [ ]:
```
# log_function(X,y)
# knn_function(X,y)
# svc_function(X,y)
# randomforest(X,y)
```

Combination_1

- forward_selector

In [ ]:
```
# 'host_total_listings_count', 'bedrooms', 'room_type_Private room', 'room_type_Shared room', 'bed_type_Real Bed
```

In [251]:
```
# X = df_v.drop('property_type',axis=1)
X = df_v[['host_total_listings_count', 'bedrooms', 'room_type_Private room','Price']]
y = df_v['property_type']
```

In [252]:
```
log_function(X,y)
```

```
Best Parameters: {'C': 1.0, 'penalty': 'l2'}

              precision    recall  f1-score   support

   Apartment       0.74      0.73      0.73       523
       House       0.77      0.78      0.77       609

    accuracy                           0.75      1132
   macro avg       0.75      0.75      0.75      1132
weighted avg       0.75      0.75      0.75      1132
```

In [253]:
```
svc_function(X,y)
```

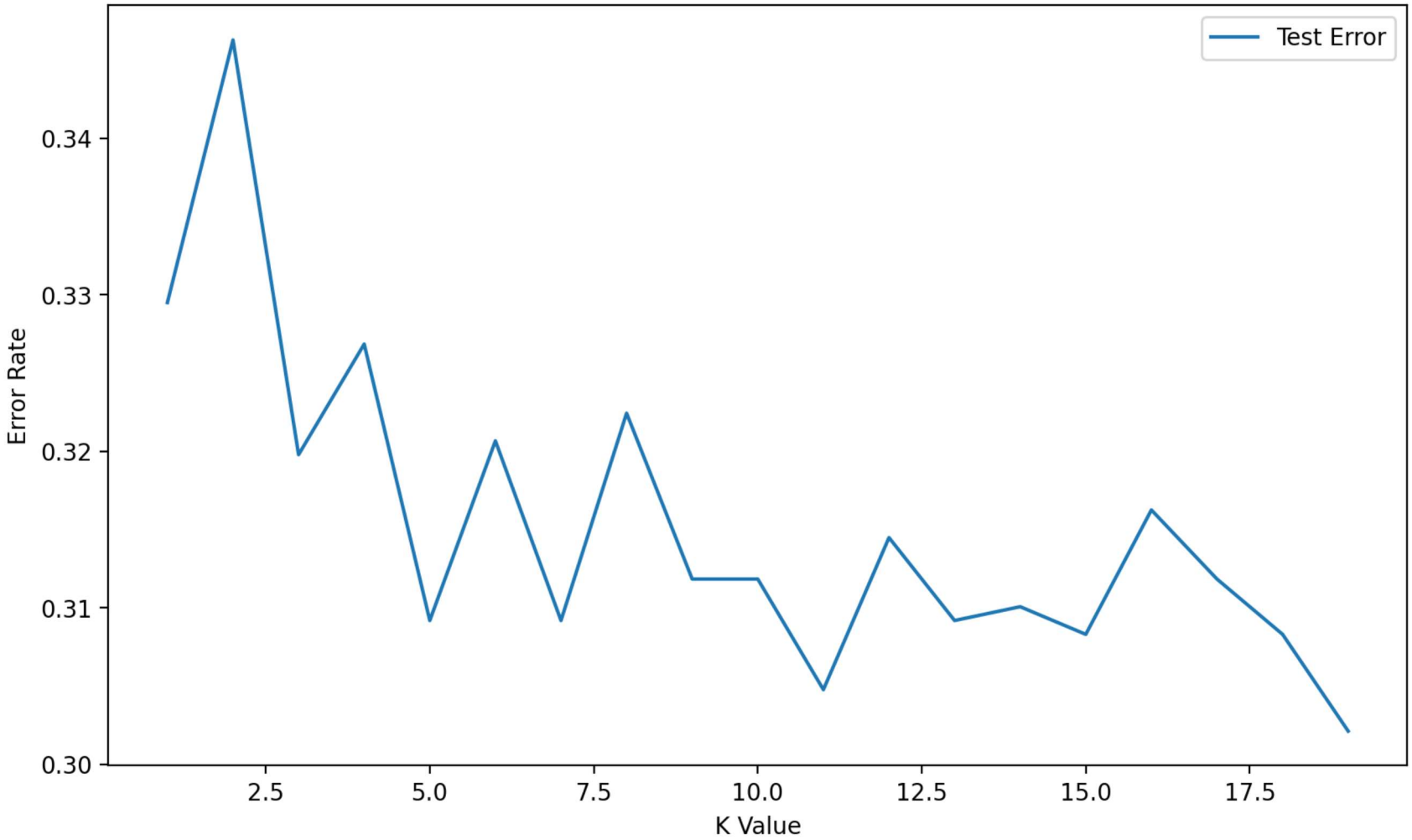
```
{'C': 1, 'gamma': 'scale'}
              precision    recall  f1-score   support

   Apartment       0.77      0.72      0.75       523
       House       0.77      0.81      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```

In [254]:
```
gradient_boost_function(X,y)
```

```
{'max_depth': 4, 'n_estimators': 40}
['Apartment' 'Apartment' 'House' ... 'Apartment' 'House' 'Apartment']
              precision    recall  f1-score   support

   Apartment       0.75      0.76      0.75       523
       House       0.79      0.78      0.78       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```

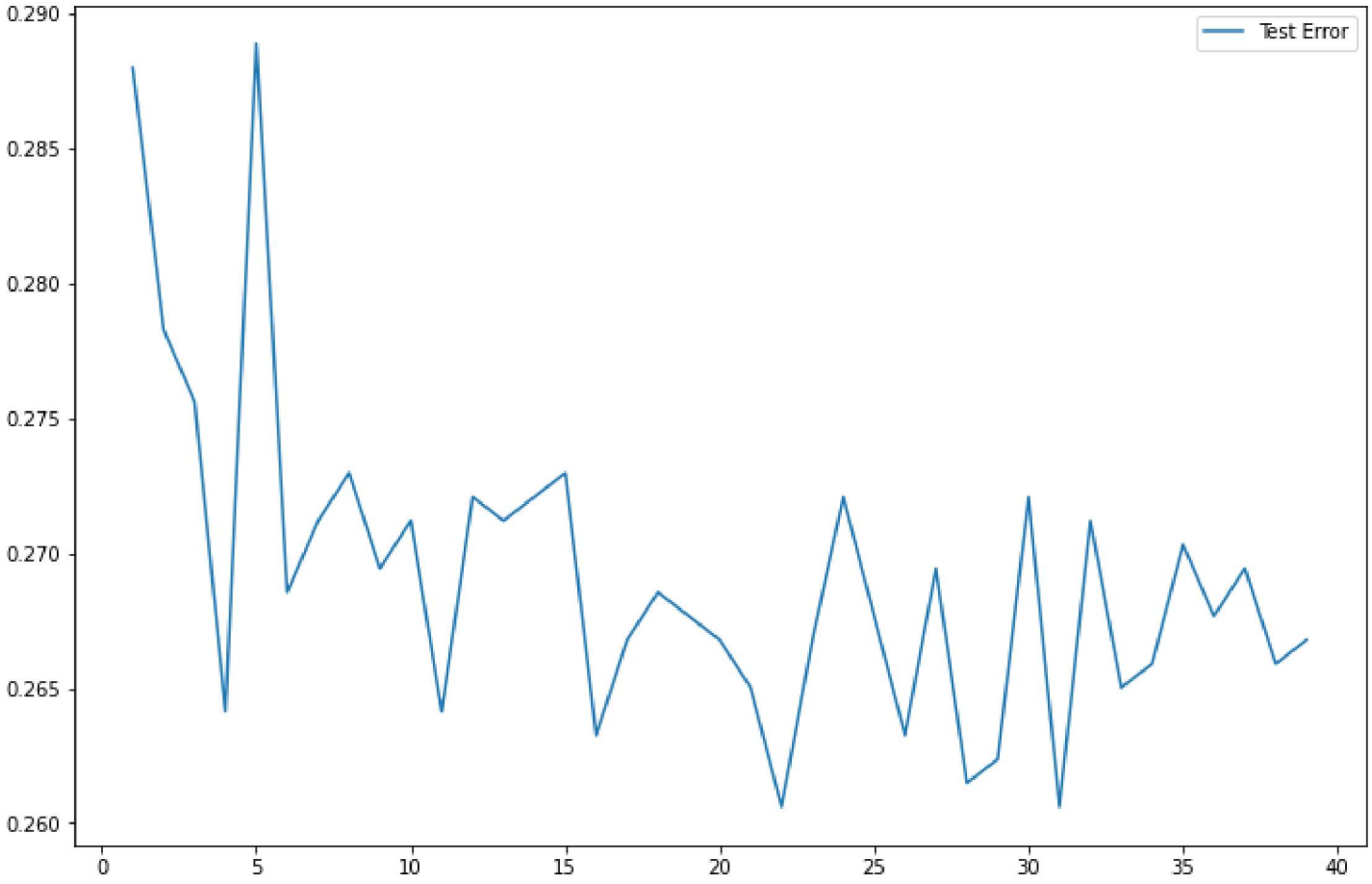
In [255]:
```
knn_function(X,y)
```

```
              precision    recall  f1-score   support

   Apartment       0.75      0.75      0.75       523
       House       0.79      0.79      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```



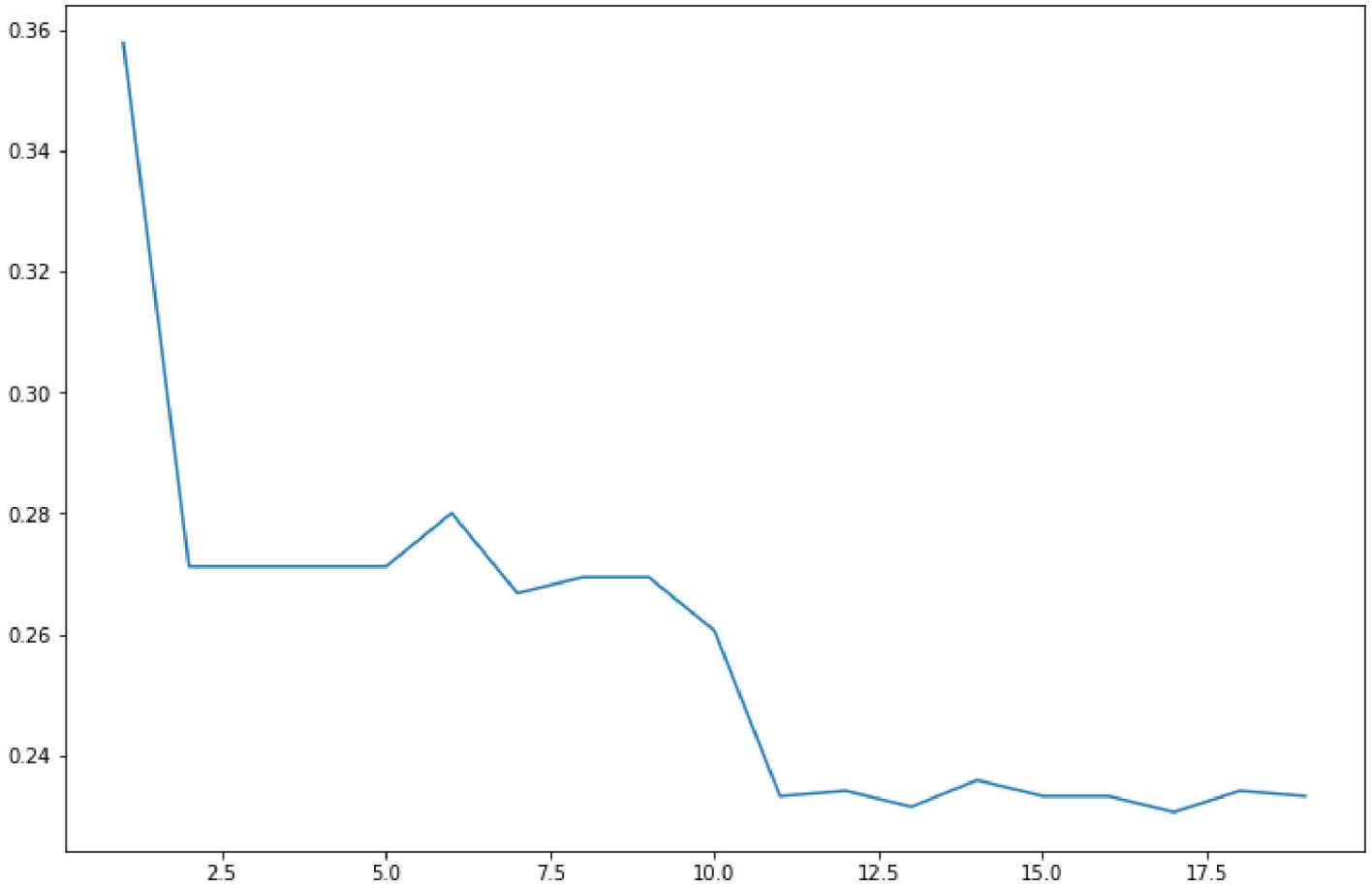
In [256]:
```
randomforest(X,y)
```

```
{'bootstrap': True, 'max_features': 2, 'n_estimators': 38}
              precision    recall  f1-score   support

   Apartment       0.71      0.72      0.72       523
       House       0.76      0.74      0.75       609

    accuracy                           0.73      1132
   macro avg       0.73      0.73      0.73      1132
weighted avg       0.73      0.73      0.73      1132
```



In [257]:
```
adaboost_function(X,y)
```

```
              precision    recall  f1-score   support

   Apartment       0.76      0.73      0.75       523
       House       0.78      0.80      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```



Combination_2

- chi square

In [261]:
```
X = df_v[['host_total_listings_count', 'bedrooms', 'accommodates', 'Price', 'room_type_Private room']]
y = df_v['property_type']
```

In [262]:
```
log_function(X,y)
```

```
Best Parameters: {'C': 1.0, 'penalty': 'l2'}

              precision    recall  f1-score   support

   Apartment       0.73      0.72      0.73       523
       House       0.76      0.78      0.77       609

    accuracy                           0.75      1132
   macro avg       0.75      0.75      0.75      1132
weighted avg       0.75      0.75      0.75      1132
```

In [263]:
```
svc_function(X,y)
```

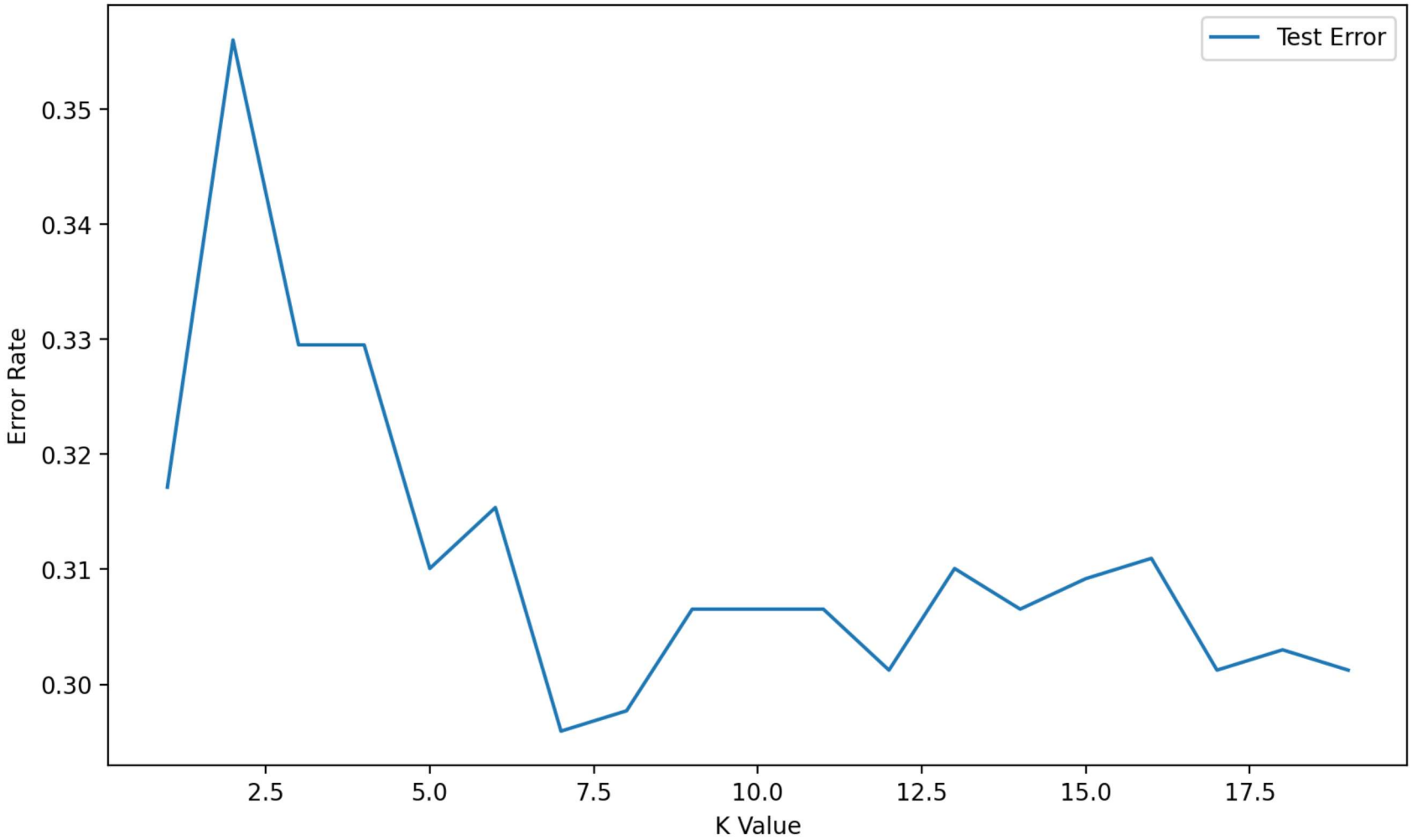
```
{'C': 1, 'gamma': 'auto'}
              precision    recall  f1-score   support

   Apartment       0.77      0.72      0.75       523
       House       0.77      0.81      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```

In [264]:
```
gradient_boost_function(X,y)
```

```
{'max_depth': 3, 'n_estimators': 40}
['Apartment' 'Apartment' 'House' ... 'Apartment' 'House' 'Apartment']
              precision    recall  f1-score   support

   Apartment       0.75      0.75      0.75       523
       House       0.79      0.79      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```

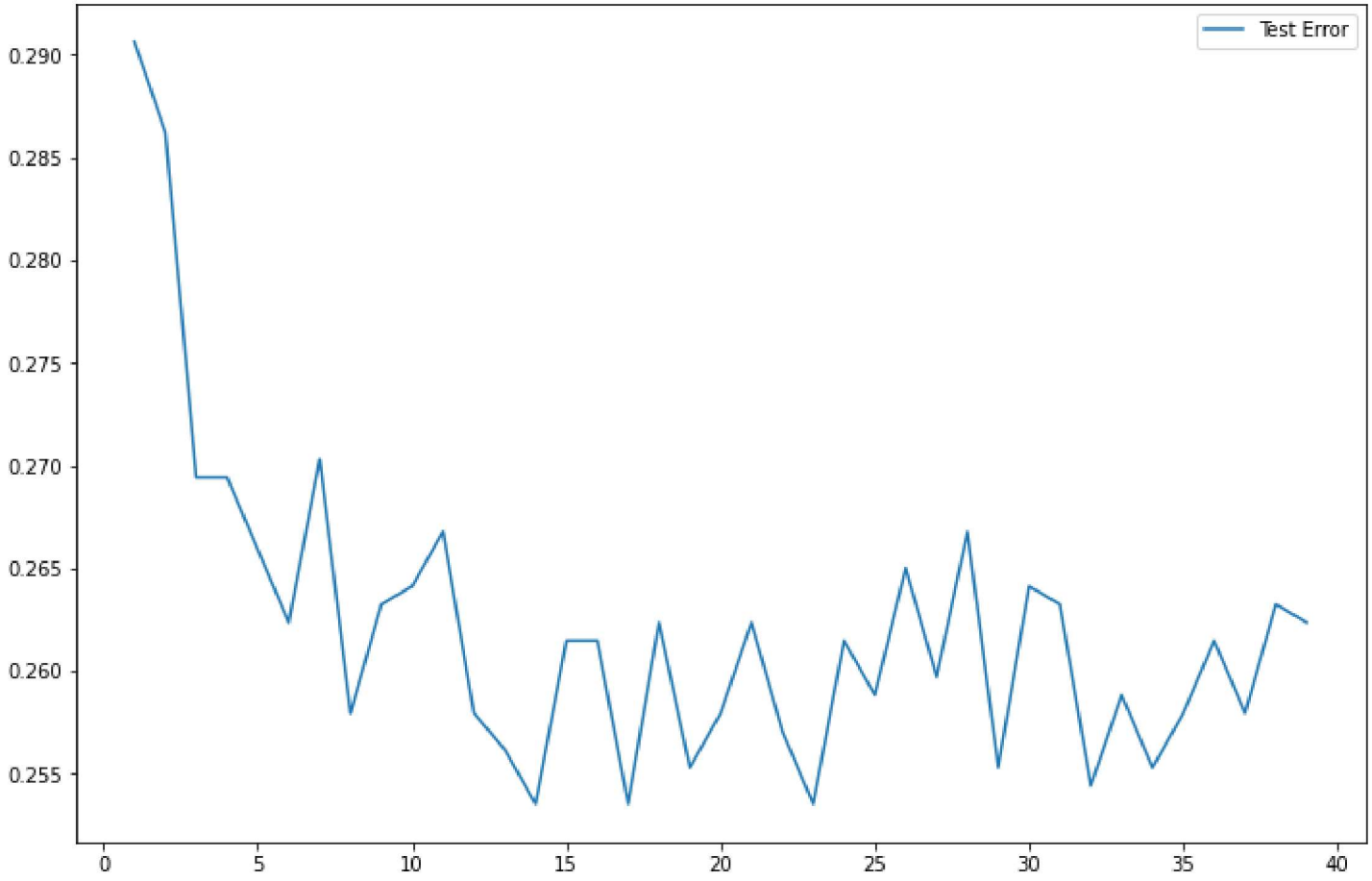
In [265]:
```
knn_function(X,y)
```

```
              precision    recall  f1-score   support

   Apartment       0.75      0.75      0.75       523
       House       0.78      0.79      0.78       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```



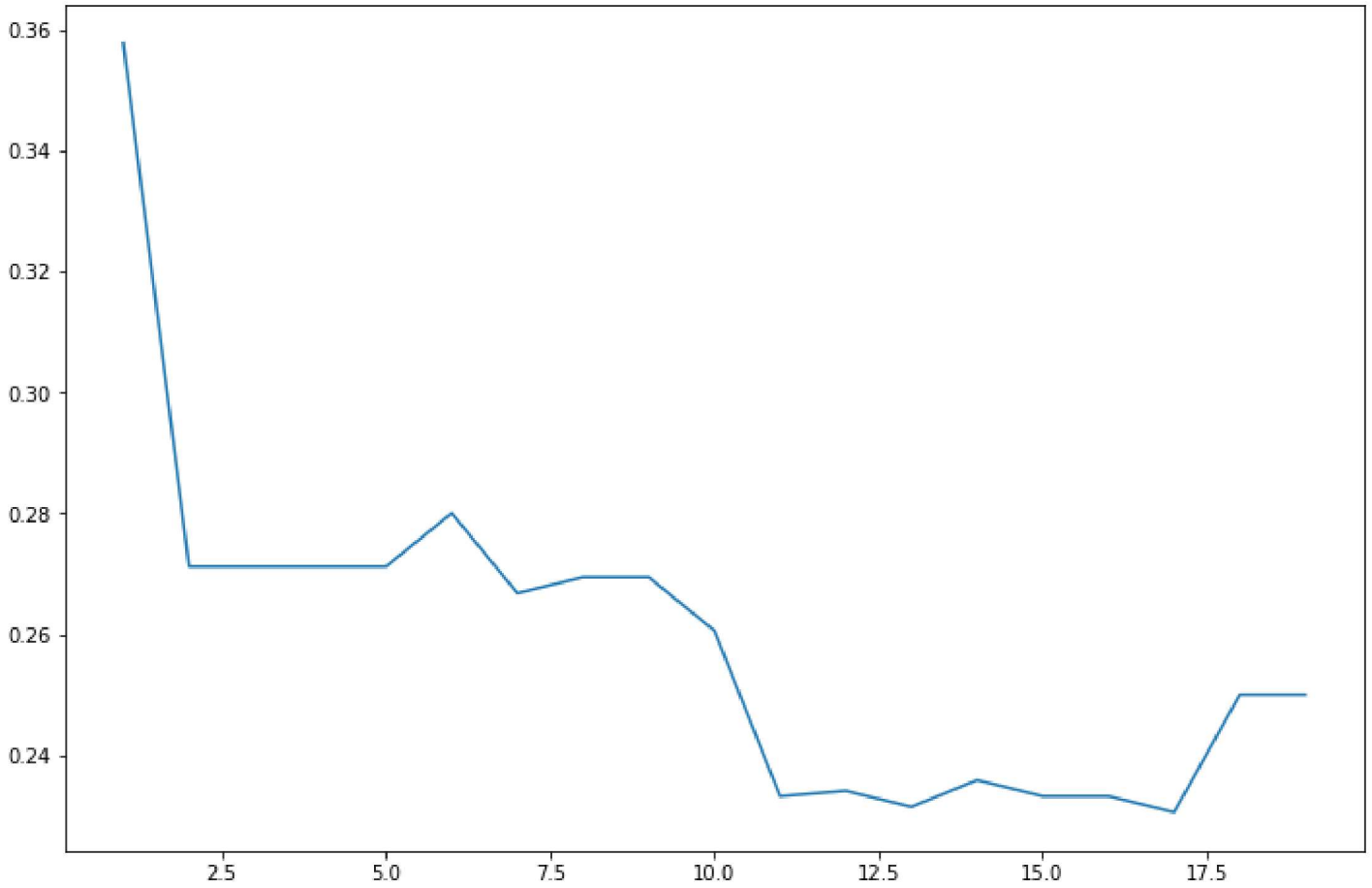
In [266]:
```
randomforest(X,y)
```

```
{'bootstrap': True, 'max_features': 4, 'n_estimators': 38}
              precision    recall  f1-score   support

   Apartment       0.71      0.73      0.72       523
       House       0.76      0.75      0.75       609

    accuracy                           0.74      1132
   macro avg       0.74      0.74      0.74      1132
weighted avg       0.74      0.74      0.74      1132
```



In [267]:
```
adaboost_function(X,y)
```

	precision	recall	f1-score	support
Apartment	0.76	0.73	0.75	523
House	0.78	0.80	0.79	609
accuracy			0.77	1132
macro avg	0.77	0.77	0.77	1132
weighted avg	0.77	0.77	0.77	1132



Combination_4

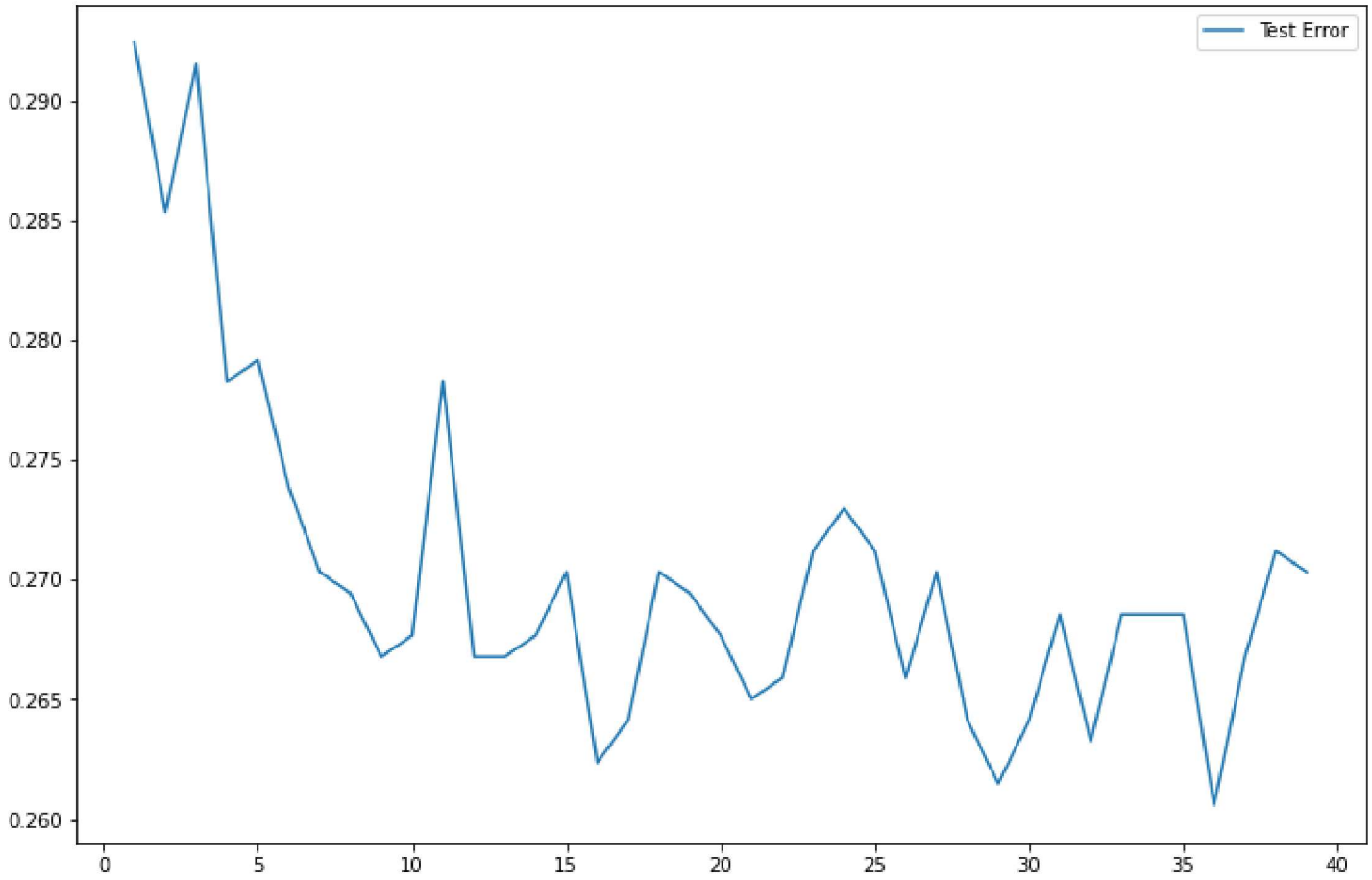
In [299]:
```
X = df_v[['host_total_listings_count', 'bedrooms', 'Price', 'room_type_Private room']]
y = df_v['property_type']
```

In [300]:
```
log_function(X,y) #76
```

```
0.759469696969697
Best Parameters: {'C': 1.0, 'penalty': 'l2'}

              precision    recall  f1-score   support

   Apartment       0.74      0.73      0.73       523
       House       0.77      0.78      0.77       609

    accuracy                           0.75      1132
   macro avg       0.75      0.75      0.75      1132
weighted avg       0.75      0.75      0.75      1132
```

In [301]:
```
randomforest(X,y) # 75.6
```

```
{'bootstrap': True, 'max_features': 3, 'n_estimators': 18}
              precision    recall  f1-score   support

   Apartment       0.70      0.72      0.71       523
       House       0.76      0.73      0.74       609

    accuracy                           0.73      1132
   macro avg       0.73      0.73      0.73      1132
weighted avg       0.73      0.73      0.73      1132

0.7556818181818181
```

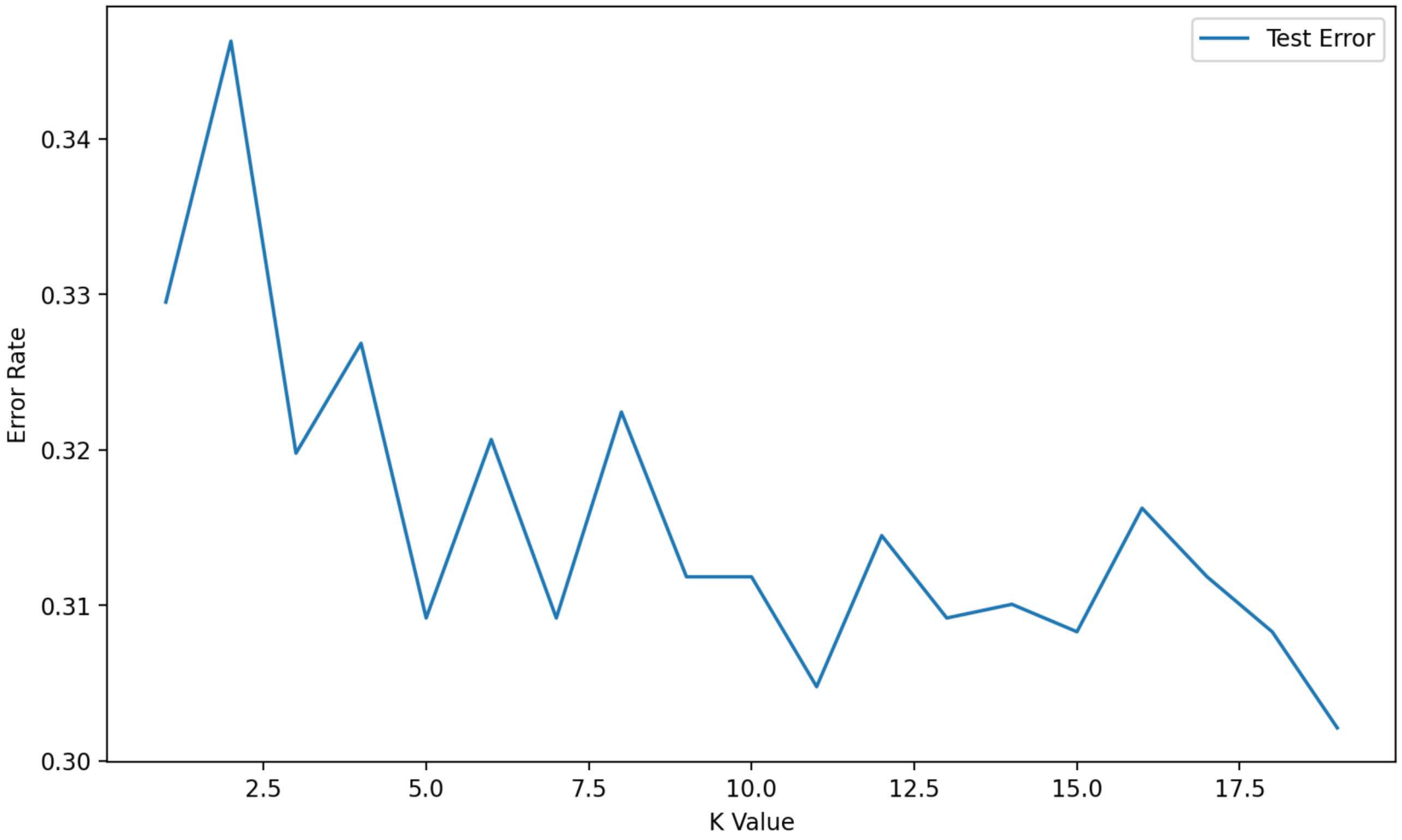


In [302]:
```
svc_function(X,y) #76.82
```

```
{'C': 1, 'gamma': 'scale'}
              precision    recall  f1-score   support

   Apartment       0.77      0.72      0.75       523
       House       0.77      0.81      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132

0.7681818181818183
```

In [307]:
```
knn_function(X,y) # 77.21
```

```
              precision    recall  f1-score   support

   Apartment       0.75      0.75      0.75       523
       House       0.79      0.79      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132

0.7720848056537103
```

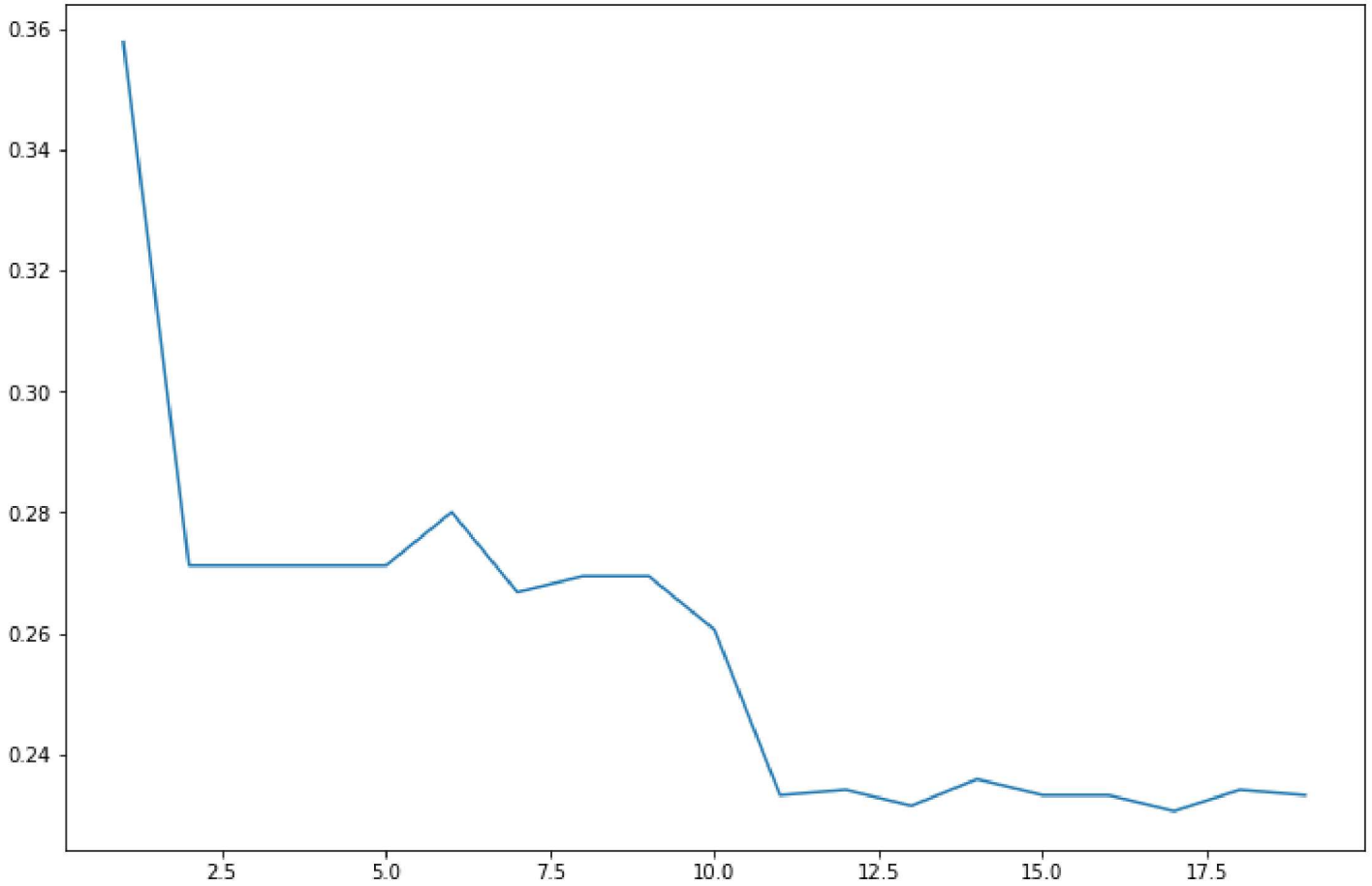


In [304]:
```
gradient_boost_function(X,y) # 77.87
```

```
{'max_depth': 4, 'n_estimators': 40}
['Apartment' 'Apartment' 'House' ... 'Apartment' 'House' 'Apartment']
              precision    recall  f1-score   support

   Apartment       0.75      0.76      0.75       523
       House       0.79      0.78      0.78       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132

0.7787878787878787
```

In [311]:
```
adaboost_function(X,y) #76.9
```

```
0.769434628975265
              precision    recall  f1-score   support

   Apartment       0.76      0.73      0.75       523
       House       0.78      0.80      0.79       609

    accuracy                           0.77      1132
   macro avg       0.77      0.77      0.77      1132
weighted avg       0.77      0.77      0.77      1132
```



In [312]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [313]:
```
final_model = GradientBoostingClassifier(n_estimators=40,max_depth=4,random_state=101)
```

In [314]:
```
X = df_v[['host_total_listings_count', 'bedrooms', 'Price', 'room_type_Private room']]
y = df_v['property_type']
```

In [315]:
```
final_model.fit(X,y)
```

Out[315]:
```
GradientBoostingClassifier(max_depth=4, n_estimators=40, random_state=101)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [316]:
```
import joblib
```

In [317]:
```
joblib.dump(final_model,'final_model.pkl')
```

Out[317]:
```
['final_model.pkl']
```

In [318]:
```
joblib.dump(list(X.columns),'column_names.pkl')
```

Out[318]:
```
['column_names.pkl']
```

In [319]:
```
model = joblib.load("final_model.pkl")
```

In [320]:
```
model.predict([[1,2,20,0]])
```

Out[320]:
```
array(['Apartment'], dtype=object)
```

In [321]:
```
df4.Price.max()
```

Out[321]:
```
1000.0
```

In [ ]:
```

```