In [38]:
```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(test)
mse = mean_squared_error(target_test, preds)
rmse = np.sqrt(mse)
rmse
```

Out[38]:
```
67879.86844243007
```

TODO: Applying the end-end ML steps to a different dataset.

We will apply what we've learnt to another dataset (airbnb dataset). We will predict airbnb price based on other features.

[35 pts] Visualizing Data

[5 pts] Load the data + statistics

- load the dataset
- display the first few rows of the data

In [39]:
```
DATASET_PATH = os.path.join("datasets", "airbnb")
def load_airbnb_data(airbnb_path):
    '''
        loads airbnb.csv dataset stored

        Args:
            airbnb_path (str): path to folder containing aribnb dataset

        Returns:
            pd.DataFrame
    '''
    csv_path = os.path.join(airbnb_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)

airbnb = load_airbnb_data(DATASET_PATH)
airbnb.head()
```

Out[39]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitud
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitud
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.9597
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.943

- pull up info on the data type for each of the data fields. Will any of these be problemmatic feeding into your model (you may need to do a little research on this)? Discuss:

In [40]:
```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   name                            48879 non-null  object 
 2   host_id                         48895 non-null  int64  
 3   host_name                       48874 non-null  object 
 4   neighbourhood_group             48895 non-null  object 
 5   neighbourhood                   48895 non-null  object 
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object 
 9   price                           48895 non-null  int64  
 10  minimum_nights                  48895 non-null  int64  
 11  number_of_reviews               48895 non-null  int64  
 12  last_review                     38843 non-null  object 
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64  
 15  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

Problematic Features: last_review: A date format must be dropped or converted into a numerical value(ie years on airbnb) room_type, neighborhood, neihborhood_group: Seems like a few categories that should be one-hot-encoded host_name: useless string should be dropped host_id, id: drop as little/no meaning attached besides some sort of ordering of sign-up dates name: could do some sort of semantic encoding to give float value on prestige or something, most likely drop though

- drop the following columns: name, host_id, host_name, and last_review
- display a summary of the statistics of the loaded data

In [41]:
```
airbnb = airbnb.drop(["name", "host_id", "host_name", "last_review"], axis=1)
```

In [42]:
```
airbnb.describe()
```

Out[42]:

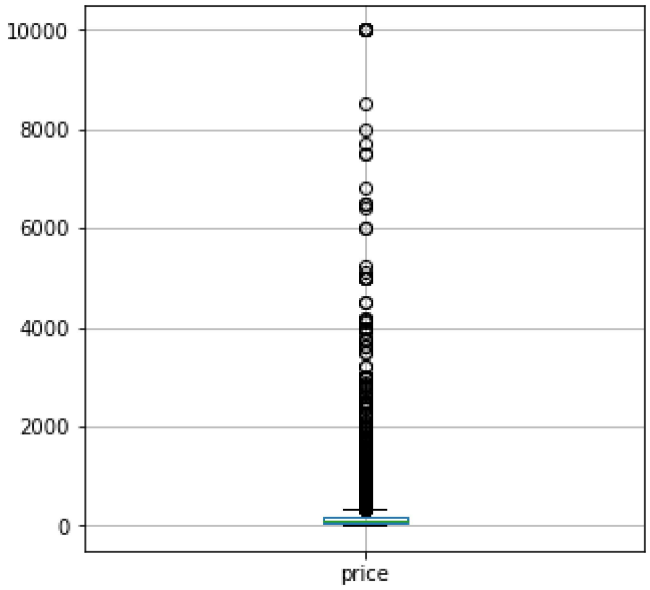
	id	latitude	longitude	price	minimum_nights	number_of_reviews	re

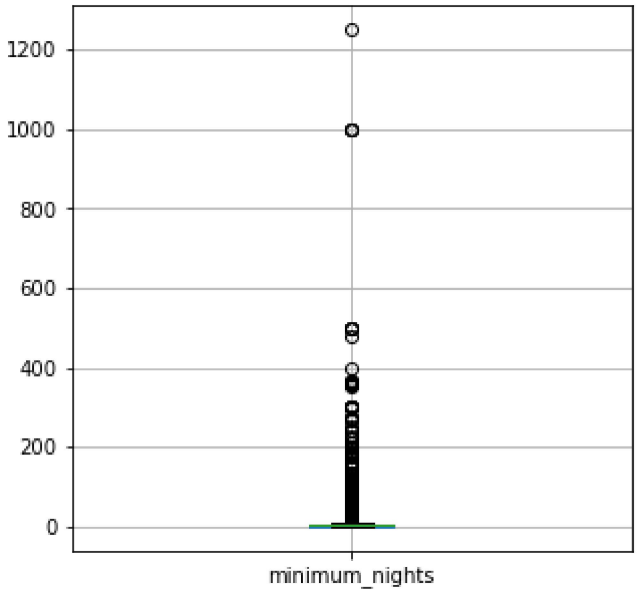
	id	latitude	longitude	price	minimum_nights	number_of_reviews	re
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	44.550582	
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	24.000000	
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	629.000000	

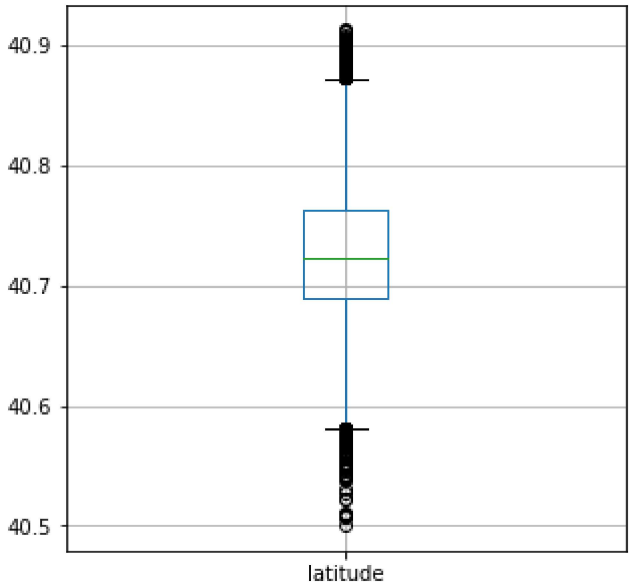
[5 pts] Boxplot 3 features of your choice

- plot boxplots for 3 features of your choice

In [43]:
```
feature_choices = ["price", "minimum_nights", "latitude"]
for feature in feature_choices:
    airbnb.boxplot(feature, figsize=(5, 5))
    plt.show()
```







- describe what you expected to see with these features and what you actually observed

With of each of these features, I was expecting a median near the means reported by describe. This was true for all three features.

I expected Latitude to be have the most unifrom boxplot of the three, with its box lying in the middle of the values. I did not expect there to be too many outliers, and was given the compact graph I believed I would get. One thing to note, however, is the two separate clusters of outliers around (40.85, 40.92) and (40.5, 40.6) respectfully. This seems to indicated two neighborhood clusters far away from all the others in the new york area. It also seems like 75% of listings are clustered into a small area or around .05 latitude which makes sense for the compactness of new york city.

I was shocked by the large range of the outliers for both minimum_nights and price features. The boxes in both boxplots seem to be very compacted around the 0 mark, with price having a bit larger of a box width, due to the extreme outliers in both features. I would love to see the listing for the

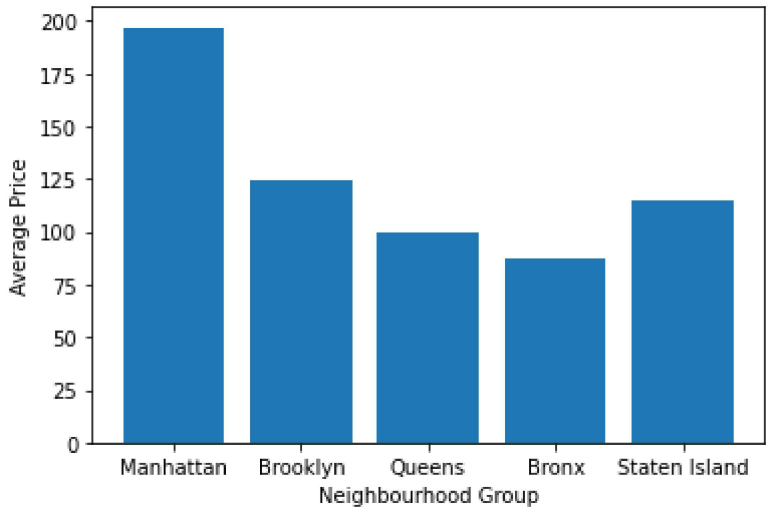
highest price outlier of $10,000 a night! It's roughly 2,000 more than the next nearest outlier. Both features have extremely large variance.

High variability in price with long tail values, review numbers much more compact, however availability has a wider variance.

[10 pts] Plot average price of a listing per neighbourhood_group

In [44]:
```
ng_types = airbnb["neighbourhood_group"].value_counts().index #get list of each type of
ng_mean_price = []
for n in ng_types:
    ng_mean_price.append(airbnb[airbnb["neighbourhood_group"] == n]["price"].mean()) #f
plt.bar(ng_types, ng_mean_price)
plt.ylabel("Average Price")
plt.xlabel("Neighbourhood Group")
```

Out[44]:
```
Text(0.5, 0, 'Neighbourhood Group')
```

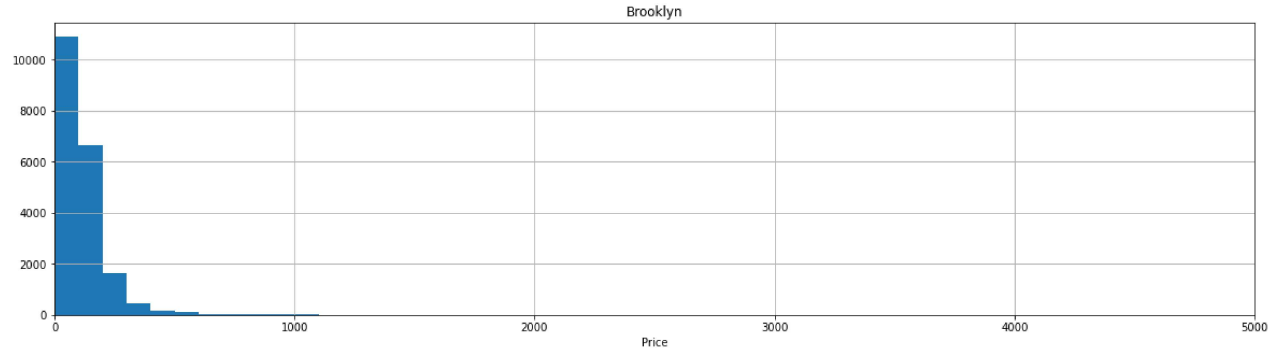


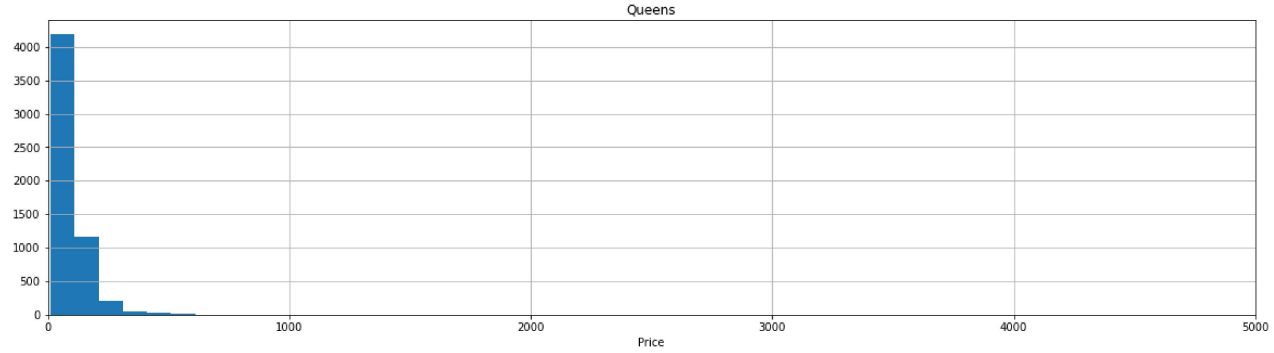
- describe what you expected to see with these features and what you actually observed

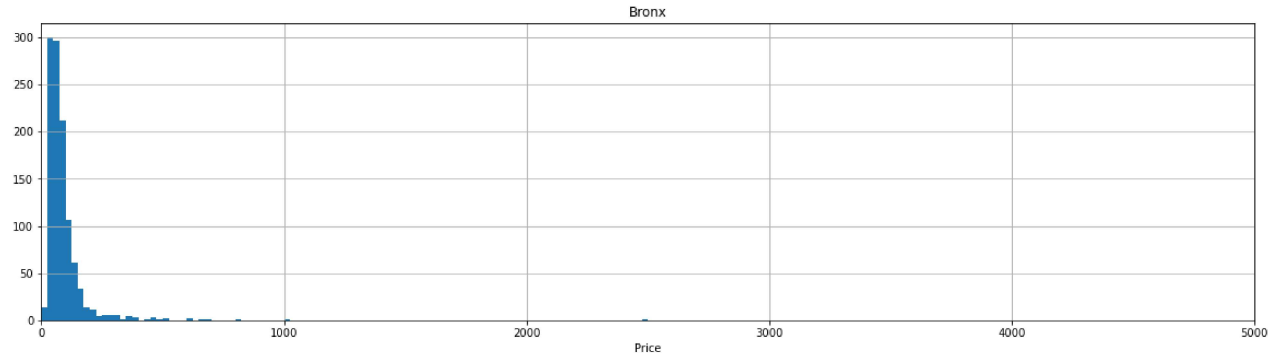
I expected so see all the averages within $200 of each other, which was true. Manhattan had the highest average, which made sense as its known as a high-end neighbourhood. Bronx and Queens has the lowest and second lowest averages which is in line with what I've been told of NY.

- So we can see different neighborhoods have dramatically different pricepoints, but how does the price breakdown by range. To see let's do a histogram of price by neighborhood to get a better sense of the distribution.

In [45]:
```
for n in ng_types:
    airbnb[airbnb["neighbourhood_group"] == n]["price"].hist( bins=100, figsize=(20,5))
    plt.title(n)
    plt.xlabel("Price")
    plt.xlim([0, 5000])
    plt.show()
```





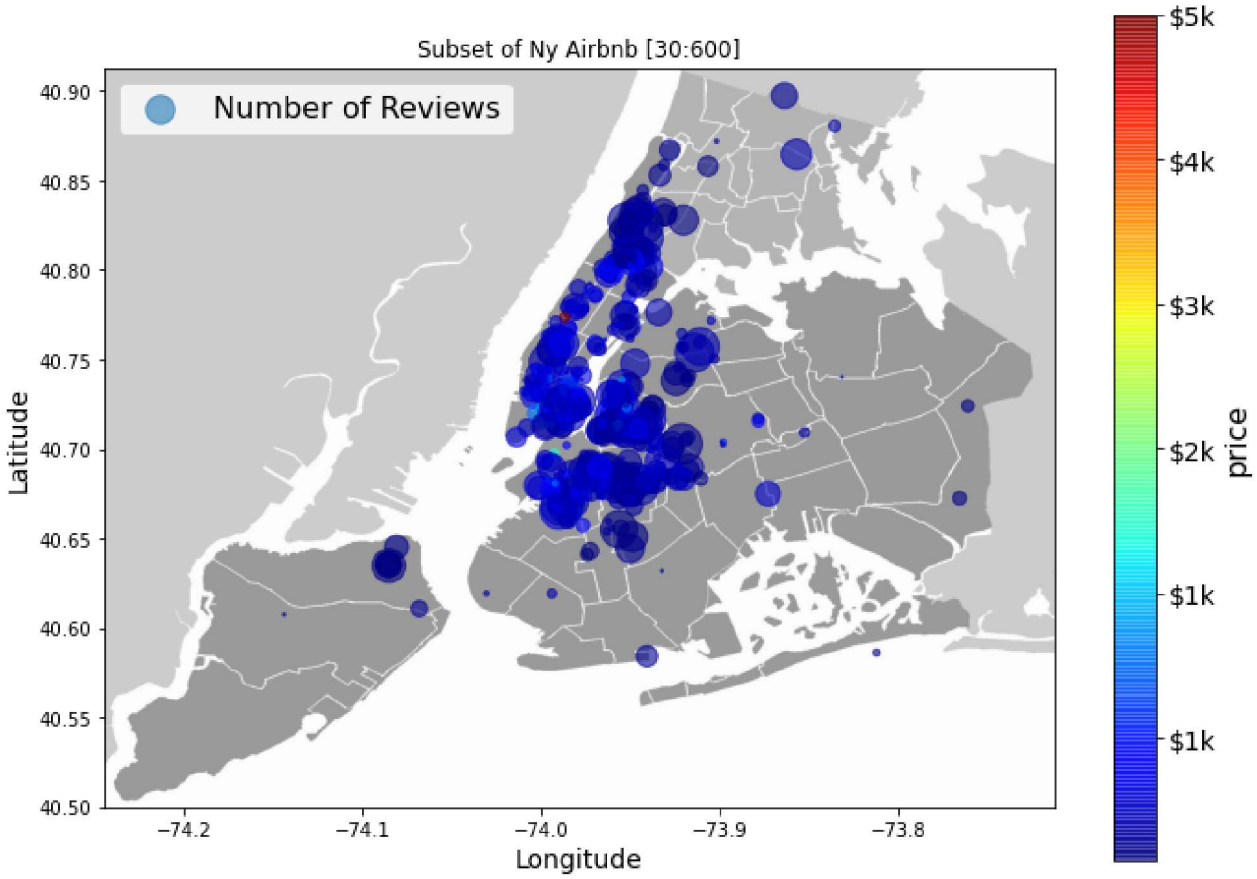


[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if

you can :)).

In [46]:
```
#COPIED FROM EXAMPLE ABOVE AND CHANGED NAMES AND LONG/LAT ranges


# load an image of NY
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "newyork.png"

import matplotlib.image as mpimg
ny_img=mpimg.imread(os.path.join(images_path, filename))
ax = airbnb[30:600].plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                         s=airbnb[30:600]['number_of_reviews'], label="Number of Reviews"
                         c="price", cmap=plt.get_cmap("jet"),
                         colorbar=False, alpha=0.6,
                        )

plt.imshow(ny_img, extent=[-74.244420, -73.712990, 40.499790, 40.913060], alpha=0.7,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)


prices = airbnb[30:600]["price"]
tick_values = np.linspace(prices.min(), prices.max())
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/50)) for v in tick_values], fontsize=14)
cb.set_label('price', fontsize=16)

plt.legend(fontsize=16)
plt.title("Subset of Ny Airbnb [30:600]")
save_fig("ny_housing_prices_plot")
plt.show()
```

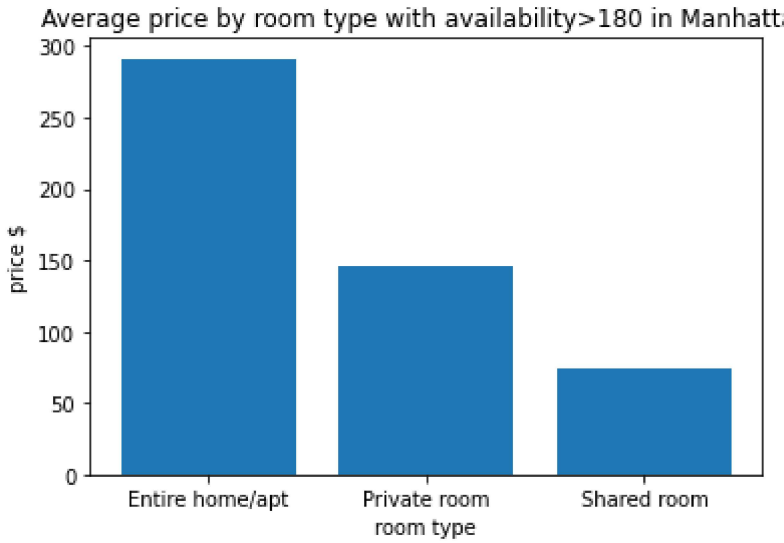
```
<ipython-input-46-1312a22e726b>:26: UserWarning: FixedFormatter should only be used toge
ther with FixedLocator
  cb.ax.set_yticklabels(["$%dk"%(round(v/50)) for v in tick_values], fontsize=14)
Saving figure ny_housing_prices_plot
```



[5 pts] Plot average price of room types who have availability greater than 180 days and neighbourhood_group is Manhattan

In [47]:
```
mrt = airbnb[airbnb["neighbourhood_group"]=="Manhattan"]
mrt = mrt[mrt["availability_365"] > 180]
room_types = mrt["room_type"].value_counts().index
room_types_price = []
for r in room_types:
    room_types_price.append(mrt[mrt["room_type"] == r]["price"].mean())
plt.bar(room_types, room_types_price)
plt.xlabel("room type")
plt.ylabel("price $")
plt.title("Average price by room type with availability>180 in Manhattan")
```

Out[47]:
```
Text(0.5, 1.0, 'Average price by room type with availability>180 in Manhattan')
```



[5 pts] Plot correlation matrix

- which features have positive correlation?
- which features have negative correlation?

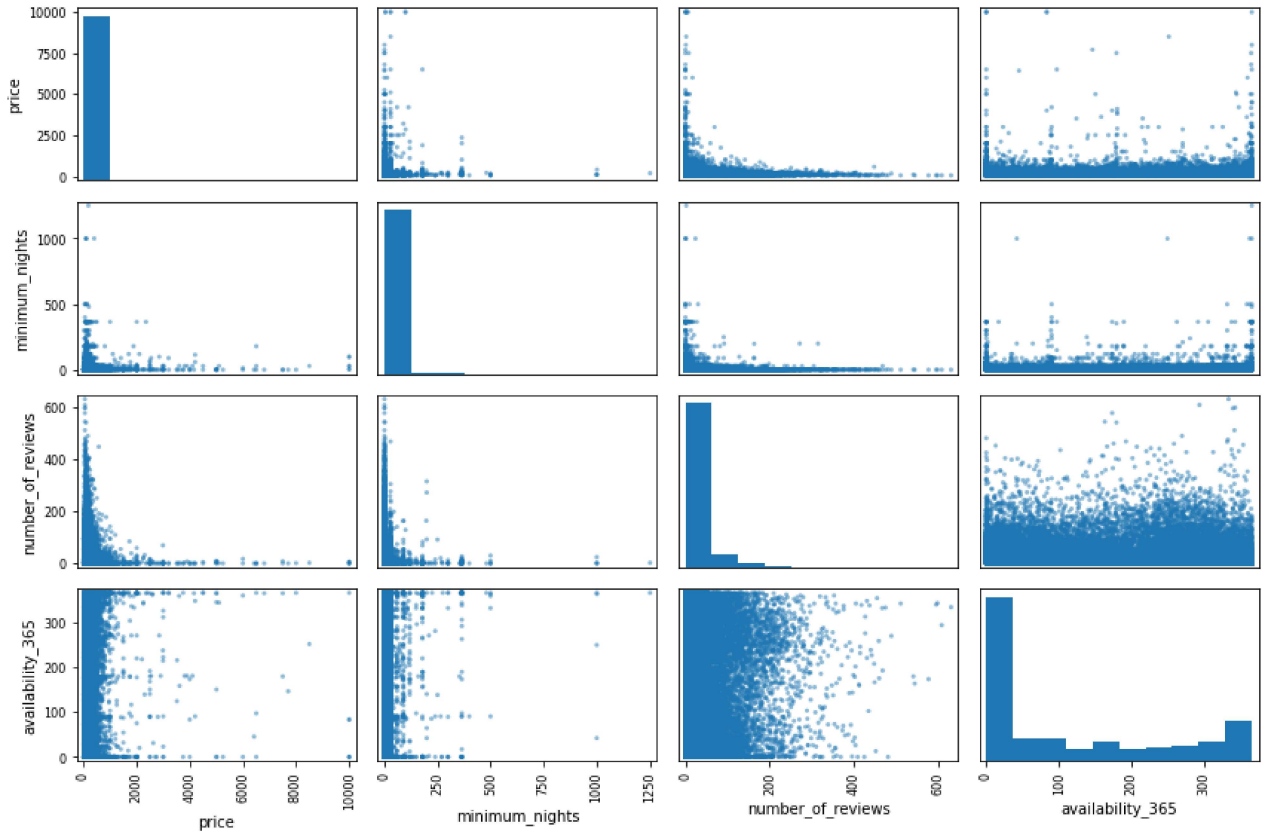
In [48]:
```
corr_matrix = airbnb.corr()
corr_matrix
```

Out[48]:

	id	latitude	longitude	price	minimum_nights	number_of_re
id	1.000000	-0.003125	0.090908	0.010619	-0.013224	-0.3
latitude	-0.003125	1.000000	0.084788	0.033939	0.024869	-0.0
longitude	0.090908	0.084788	1.000000	-0.150019	-0.062747	0.0
price	0.010619	0.033939	-0.150019	1.000000	0.042799	-0.0
minimum_nights	-0.013224	0.024869	-0.062747	0.042799	1.000000	-0.0
number_of_reviews	-0.319760	-0.015389	0.059094	-0.047954	-0.080116	1.0
reviews_per_month	0.291828	-0.010142	0.145948	-0.030608	-0.121702	0.5
calculated_host_listings_count	0.133272	0.019517	-0.114713	0.057472	0.127960	-0.0
availability_365	0.085468	-0.010983	0.082731	0.081829	0.144303	0.1

In [49]:
```
attributes = [ "price",
               "minimum_nights", "number_of_reviews", "availability_365"]
scatter_matrix(airbnb[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

```
Saving figure scatter_matrix_plot
```



Entries in the matrix > 0 have positive correlation between the row and column indicies

entires in the matrix < 0 have negative correlation between the row and column indices

NOTE: correlation != causation as seen between positive correlation of id and price

[30 pts] Prepare the Data

[5 pts] Augment the dataframe with two other features which you think would be useful

In [50]:
```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   neighbourhood_group             48895 non-null  object 
 2   neighbourhood                   48895 non-null  object 
 3   latitude                        48895 non-null  float64
 4   longitude                       48895 non-null  float64
 5   room_type                       48895 non-null  object 
 6   price                           48895 non-null  int64  
 7   minimum_nights                  48895 non-null  int64  
 8   number_of_reviews               48895 non-null  int64  
 9   reviews_per_month               38843 non-null  float64
 10  calculated_host_listings_count  48895 non-null  int64  
 11  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(6), object(3)
memory usage: 4.5+ MB
```

In [51]:
```
airbnb["max_yearly_bookings"] = airbnb["availability_365"]/airbnb["minimum_nights"]
airbnb["min_total_nights_booked"] = airbnb["number_of_reviews"]*airbnb["minimum_nights"
```

[5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

In [52]:
```
airbnb_incomplete = airbnb[airbnb.isnull().any(axis=1)]
airbnb_incomplete.head()
#airbnb_incomplete["reviews_per_month"].isna().sum()
```

Out[52]:

	id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nig
2	3647	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
19	7750	Manhattan	East Harlem	40.79685	-73.94872	Entire home/apt	190	
26	8700	Manhattan	Inwood	40.86754	-73.92639	Private room	80	
36	11452	Brooklyn	Bedford-Stuyvesant	40.68876	-73.94312	Private room	35	
38	11943	Brooklyn	Flatbush	40.63702	-73.96327	Private room	150	

In [53]:
```
#As more than 1/5th of reviews_per_month data is missing, seems like I should just remo
airbnb = airbnb.drop("reviews_per_month", axis=1)
```

In [54]:
```
airbnb.head()
```

Out[54]:

	id	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_night
0	2539	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	3647	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	3831	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	1(

[15 pts] Code complete data pipeline using sklearn mixins

In [67]:
```
airbnb_features = airbnb.drop("price", axis=1)
airbnb_labels = airbnb["price"].copy()
airbnb_num = airbnb_features.drop(["neighbourhood_group", "neighbourhood", "room_type"]
```

```
# airbnb_cat = airbnb["id", "latitude", "longitude", "minimum_nights", "number_of_revie
#                      "availability_365", "max_yearly_bookings", "min_total_nights_book
```

In [68]:
```
imputer = SimpleImputer(strategy="median")

minimum_nights_idx, number_of_reviews_idx, availability_365_idx = 3, 4, 6 #indexes of n
```

In [69]:
```
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    airbnb["max_yearly_bookings"] = airbnb["availability_365"]/airbnb["minimum_nights"]
    airbnb["min_total_nights_booked"] = airbnb["number_of_reviews"]*airbnb["minimum_nig
    '''
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self  # nothing else to do

    def transform(self, X):
        max_yearly_bookings = X[:, availability_365_idx] / max(X[:, minimum_nights_idx]
        min_total_nights_booked = X[:, number_of_reviews_idx] * X[:, minimum_nights_idx
        return np.c_[X, max_yearly_bookings, min_total_nights_booked]
```

In [70]:
```
attr_adder = AugmentFeatures()
airbnb_extra_attribs = attr_adder.transform(airbnb.values)
```

In [71]:
```
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])
```

In [72]:
```
airbnb_num_tr = num_pipeline.fit_transform(airbnb_num)

numerical_features = list(airbnb_num)
categorical_features = ["neighbourhood_group", "neighbourhood", "room_type"]
```

In [73]:
```
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

airbnb_prepared = full_pipeline.fit_transform(airbnb_features)
```

In [74]:
```
airbnb_prepared[:10][:10] #????? How can I tell if prepared it right or now
```

Out[74]:
```
<10x240 sparse matrix of type '<class 'numpy.float64'>'
        with 140 stored elements in Compressed Sparse Row format>
```

[5 pts] Set aside 20% of the data as test test (80% train, 20% test).

In [75]:
```
data_target = airbnb['price']
train, test, target, target_test = train_test_split(airbnb_prepared, data_target, test_
```

[15 pts] Fit a model of your choice

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using MSE. Provide both test and train set MSE values.

In [76]:
```
lin_reg = LinearRegression()
lin_reg.fit(train, target)

# let's try the full preprocessing pipeline on a few training instances
data = test
labels = target_test

print("Predictions:", lin_reg.predict(data)[:5])
print("Actual labels:", list(labels)[:5])
```

```
Predictions: [527.96074055 320.07133802 191.37637447  88.12363003 146.15464863]
Actual labels: [225, 649, 300, 26, 125]
```

In [77]:
```
preds = lin_reg.predict(test)
mse = mean_squared_error(target_test, preds)
rmse = np.sqrt(mse)
print("Test Set:")
print("MSE:", mse)
print("RMSE:", rmse)
```

```
Test Set:
MSE: 47991.36784940251
RMSE: 219.06932201794598
```

In [78]:
```
preds = lin_reg.predict(train)
mse = mean_squared_error(target, preds)
rmse = np.sqrt(mse)
print("Train Set:")
print("MSE:", mse)
print("RMSE:", rmse)
```

```
Train Set:
MSE: 51559.335201108086
RMSE: 227.06680779257036
```

Strangely enough, my model performs better on the test set then the train set, indicating my model is not overfitted

In [ ]:
```

```