Exploratory Data Analysis and Visualization Part Two

Introduction

This article builds upon Part One [1] and revisits and expands on ideas related to exploratory data analysis (EDA). This tutorial is based on materials sourced from this primary source [2] and follows the general outline and steps described therein. As in the previous article, the datasets used and analysis followed reproduce the book's items to demonstrate the steps a data scientist would take when dealing with a new dataset.

Additionally, examples and discussions go beyond the primary source material to either extend or emphasize different aspects of EDA. This is done with either the supplied datasets from the book or analysis of additional supplementary datasets available.

This work with having data to analyze. Four different datasets [3,4,5,6] are used in the book to show how to analyze the data. The sources of data are either from github stored notebooks or kaggle notebooks. The datasets are in simple CSV formats and can be read into Pandas dataframes. From there, descriptive statistics and plots can be examined for each dataset.

Understanding the datasets and examining their meaning becomes important when the data is intended to be used for learning. Recall, the bulk of the data parameters (features listed in columns) are the independent variables. These are the X matrix of inputs supplied per example (row). The dependent variable is the target feature, the column vector y that is being predicted. This is typically the labels of the dataset.

These inputs are used to train models in supervised learning (with unsupervised and reinforcement learning being additional model training techniques). The model is then used to predict outputs for newly supplied data not found in the training dataset.

Theory

This article covers EDA topics by utilizing Jupyter notebooks to process and visualize the data. The datasets are each examined to determine the overall quality of the data. Basic information can identify missing data or wrongly formatted data that needs to be corrected. This data cleaning process is something that must be done prior to starting to use the dataset for any learning operations.

Let's start by reviewing each of the supplied datasets. The first one is a CSV file of AirBnB listing data for the city or Berlin, Germany. There are a number of features and many samples in this file. This section details how to take a quick look at different aspects of the data. These methods can be used with any of the datasets described earlier and are shown in the appendix.

In [25]:
```
import pandas as pd
import seaborn as sns
%matplotlib inline
listings_df = pd.read_csv('listings.csv')
listings_df.head()
```

Out[25]:

	id	listing_url	scrape_id	last_scraped	source	name	description	neighborhood_overview	
0	3176	https://www.airbnb.com/rooms/3176	20240622134424	2024-06-23	city scrape	Fabulous Flat in great Location	This beautiful first floor apartment is situa...	The neighbourhood is famous for its variety of...	https://a pictures
1	9991	https://www.airbnb.com/rooms/9991	20240622134424	2024-06-23	city scrape	Geourgeous flat - outstanding views	4 bedroom with very large windows and outstand...	Prenzlauer Berg is an amazing neighbourhood wh...	https://a pictures/
2	14325	https://www.airbnb.com/rooms/14325	20240622134424	2024-06-22	city scrape	Studio Apartment in Prenzlauer Berg	The apartment is located on the upper second f...	NaN	https://a pictures/
3	16644	https://www.airbnb.com/rooms/16644	20240622134424	2024-06-23	city scrape	In the Heart of Berlin - Kreuzberg	Light and sunny 2-Room-turn of the century-fla...	Our Part of Kreuzberg is just the best. Good v...	https://a pictures,
4	17904	https://www.airbnb.com/rooms/17904	20240622134424	2024-06-23	city scrape	Beautiful Kreuzberg studio - 3 months minimum	- apt is available starting September 1, 2024<...	The apartment is located in Kreuzberg, which i...	https://a pictures,

5 rows × 75 columns

The "shape", "columns", "info" and "describe" (for descriptive statistics) are all commands that can be used to for an initial analysis of the dataset.

As can be seen, the total number of rows is 13759 and the number of features is large with 75 columns. The describe command, supplies information on the columns that are numerical in nature. The info command gives the quantity of each non-null data. This means, if the number given for a feature is less than the total number of rows, then that implies there is missing data that needs to be handled.

In [28]:
```
listings_df.shape
```

Out[28]:
```
(13759, 75)
```

In [30]:
```
listings_df.columns
```

Out[30]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'source', 'name',
       'description', 'neighborhood_overview', 'picture_url', 'host_id',
       'host_url', 'host_name', 'host_since', 'host_location', 'host_about',
       'host_response_time', 'host_response_rate', 'host_acceptance_rate',
       'host_is_superhost', 'host_thumbnail_url', 'host_picture_url',
       'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhood',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude',
       'longitude', 'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month'],
      dtype='object')
```

In [32]:
```
listings_df.describe()
```

Out[32]:

	id	scrape_id	host_id	host_listings_count	host_total_listings_count	latitude	longitude
count	1.375900e+04	1.375900e+04	1.375900e+04	13750.000000	13750.000000	13759.000000	13759.000000
mean	3.529798e+17	2.024062e+13	1.618469e+08	19.459782	22.084582	52.509342	13.402723
std	4.588343e+17	5.929903e+00	1.822188e+08	86.314352	99.540534	0.033789	0.067338
min	3.176000e+03	2.024062e+13	1.581000e+03	1.000000	1.000000	52.340190	13.118150
25%	1.884756e+07	2.024062e+13	1.447609e+07	1.000000	1.000000	52.490077	13.362765
50%	4.234426e+07	2.024062e+13	6.789678e+07	1.000000	2.000000	52.509220	13.411275
75%	8.461250e+17	2.024062e+13	2.785392e+08	4.000000	6.000000	52.532147	13.438580
max	1.184030e+18	2.024062e+13	5.846847e+08	1195.000000	1448.000000	52.656110	13.721390

8 rows × 39 columns

In [34]:

```
listings_df.info()
```

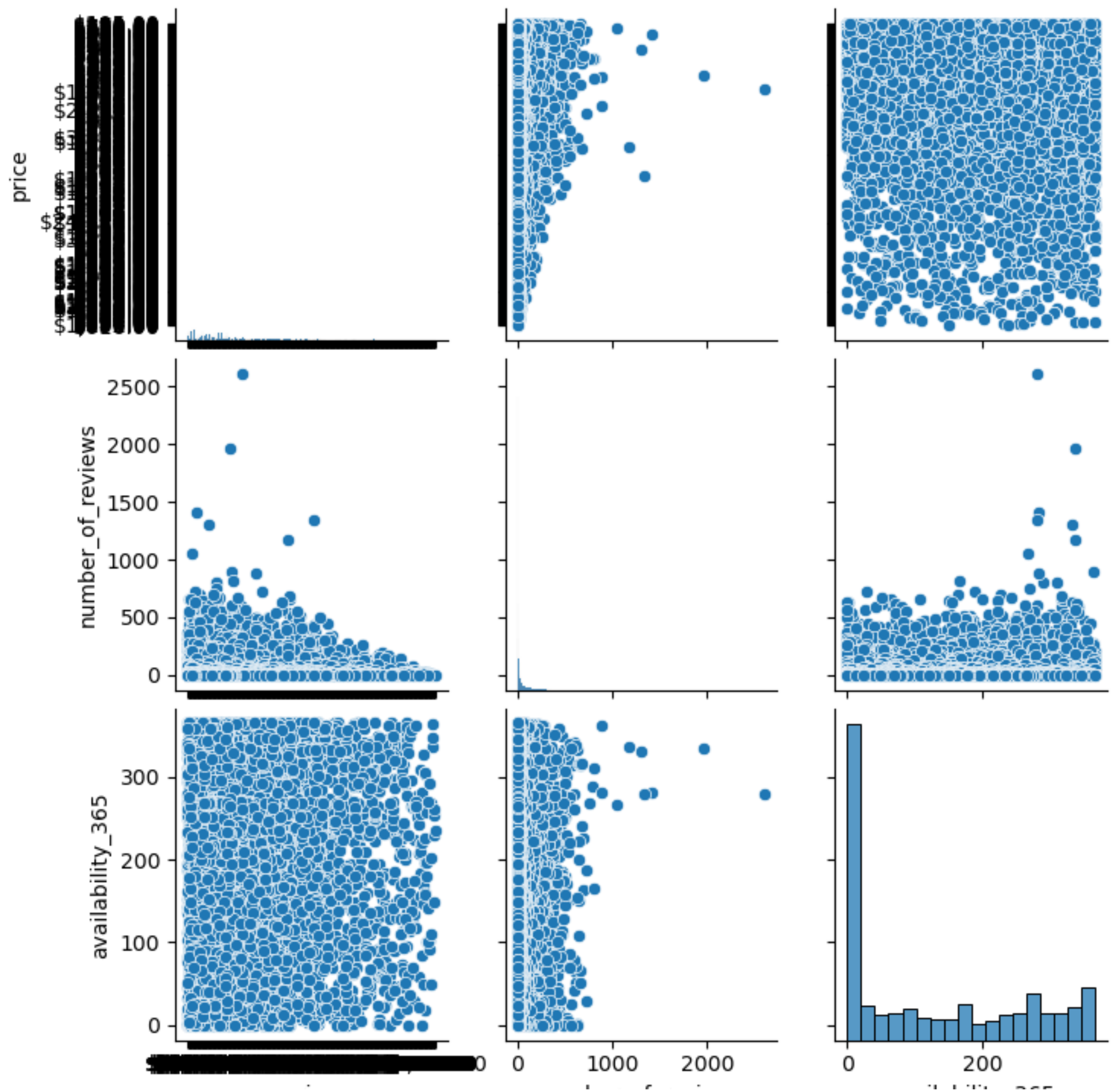
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13759 entries, 0 to 13758
Data columns (total 75 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   id                                            13759 non-null  int64
 1   listing_url                                   13759 non-null  object
 2   scrape_id                                     13759 non-null  int64
 3   last_scraped                                  13759 non-null  object
 4   source                                        13759 non-null  object
 5   name                                          13759 non-null  object
 6   description                                   13137 non-null  object
 7   neighborhood_overview                         7075 non-null   object
 8   picture_url                                   13759 non-null  object
 9   host_id                                       13759 non-null  int64
 10  host_url                                      13759 non-null  object
 11  host_name                                     13750 non-null  object
 12  host_since                                    13750 non-null  object
 13  host_location                                 11063 non-null  object
 14  host_about                                    7157 non-null   object
 15  host_response_time                            8893 non-null   object
 16  host_response_rate                            8893 non-null   object
 17  host_acceptance_rate                          9353 non-null   object
 18  host_is_superhost                             13641 non-null  object
 19  host_thumbnail_url                            13750 non-null  object
 20  host_picture_url                              13750 non-null  object
 21  host_neighbourhood                            6190 non-null   object
 22  host_listings_count                           13750 non-null  float64
 23  host_total_listings_count                     13750 non-null  float64
 24  host_verifications                            13759 non-null  object
 25  host_has_profile_pic                          13750 non-null  object
 26  host_identity_verified                        13750 non-null  object
 27  neighbourhood                                 7075 non-null   object
 28  neighbourhood_cleansed                        13759 non-null  object
 29  neighbourhood_group_cleansed                  13759 non-null  object
 30  latitude                                      13759 non-null  float64
 31  longitude                                     13759 non-null  float64
 32  property_type                                 13759 non-null  object
 33  room_type                                     13759 non-null  object
 34  accommodates                                  13759 non-null  int64
 35  bathrooms                                     8818 non-null   float64
 36  bathrooms_text                                13754 non-null  object
```

```
 37  bedrooms                                      11702 non-null  float64
 38  beds                                          8758 non-null   float64
 39  amenities                                     13759 non-null  object 
 40  price                                         8821 non-null   object 
 41  minimum_nights                                13759 non-null  int64  
 42  maximum_nights                                13759 non-null  int64  
 43  minimum_minimum_nights                        13759 non-null  int64  
 44  maximum_minimum_nights                        13759 non-null  int64  
 45  minimum_maximum_nights                        13759 non-null  int64  
 46  maximum_maximum_nights                        13759 non-null  int64  
 47  minimum_nights_avg_ntm                        13759 non-null  float64
 48  maximum_nights_avg_ntm                        13759 non-null  float64
 49  calendar_updated                              0 non-null      float64
 50  has_availability                              12863 non-null  object 
 51  availability_30                               13759 non-null  int64  
 52  availability_60                               13759 non-null  int64  
 53  availability_90                               13759 non-null  int64  
 54  availability_365                              13759 non-null  int64  
 55  calendar_last_scraped                         13759 non-null  object 
 56  number_of_reviews                             13759 non-null  int64  
 57  number_of_reviews_ltm                         13759 non-null  int64  
 58  number_of_reviews_l30d                        13759 non-null  int64  
 59  first_review                                  10521 non-null  object 
 60  last_review                                   10521 non-null  object 
 61  review_scores_rating                          10521 non-null  float64
 62  review_scores_accuracy                        10515 non-null  float64
 63  review_scores_cleanliness                     10517 non-null  float64
 64  review_scores_checkin                         10514 non-null  float64
 65  review_scores_communication                   10516 non-null  float64
 66  review_scores_location                        10514 non-null  float64
 67  review_scores_value                           10512 non-null  float64
 68  license                                       8803 non-null   object 
 69  instant_bookable                              13759 non-null  object 
 70  calculated_host_listings_count                13759 non-null  int64  
 71  calculated_host_listings_count_entire_homes   13759 non-null  int64  
 72  calculated_host_listings_count_private_rooms  13759 non-null  int64  
 73  calculated_host_listings_count_shared_rooms   13759 non-null  int64  
 74  reviews_per_month                             10521 non-null  float64
dtypes: float64(18), int64(21), object(36)
memory usage: 7.9+ MB
```

The pairwise plots of different features and a heatmap are two popular views of the data. These two can very quickly show different relationships between the variables. These can include variance and feature correlation.

In [37]:
```
sns.pairplot(listings_df, vars=['price', 'number_of_reviews', 'availability_365'])
```

Out[37]:
```
<seaborn.axisgrid.PairGrid at 0x73c0c0941270>
```


price
number_of_reviews
availability_365
2500
2000
1500
1000
500
0
300
200
100
0
1000
2000
200

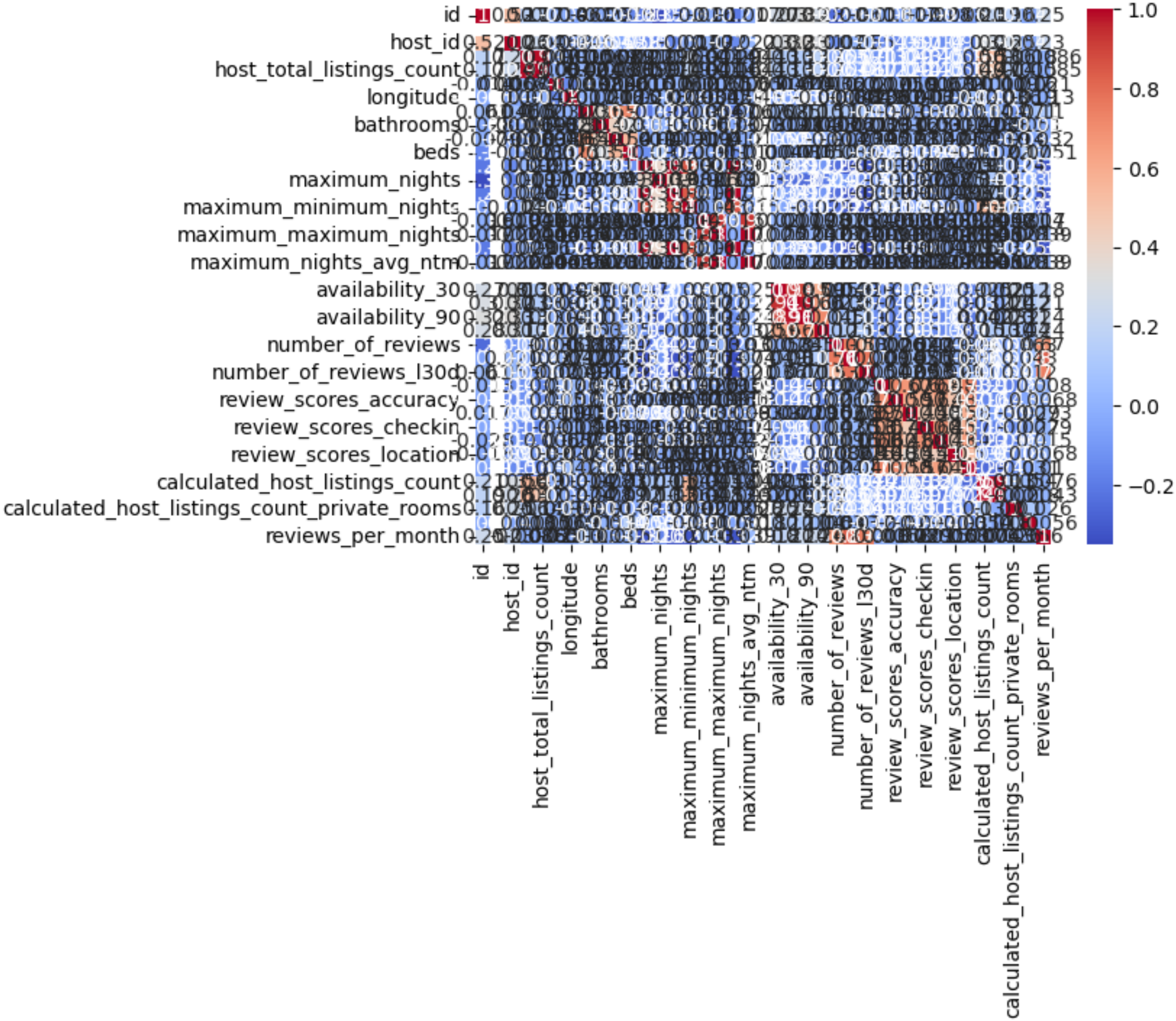
price number_of_reviews availability_365

In [38]:

```
import numpy as np
listings_df_numeric = listings_df.select_dtypes(include=np.number)
listings_df_corr = listings_df_numeric.corr()
sns.heatmap(listings_df_corr, annot=True, cmap='coolwarm')
```

Out[38]:

```
<Axes: >
```

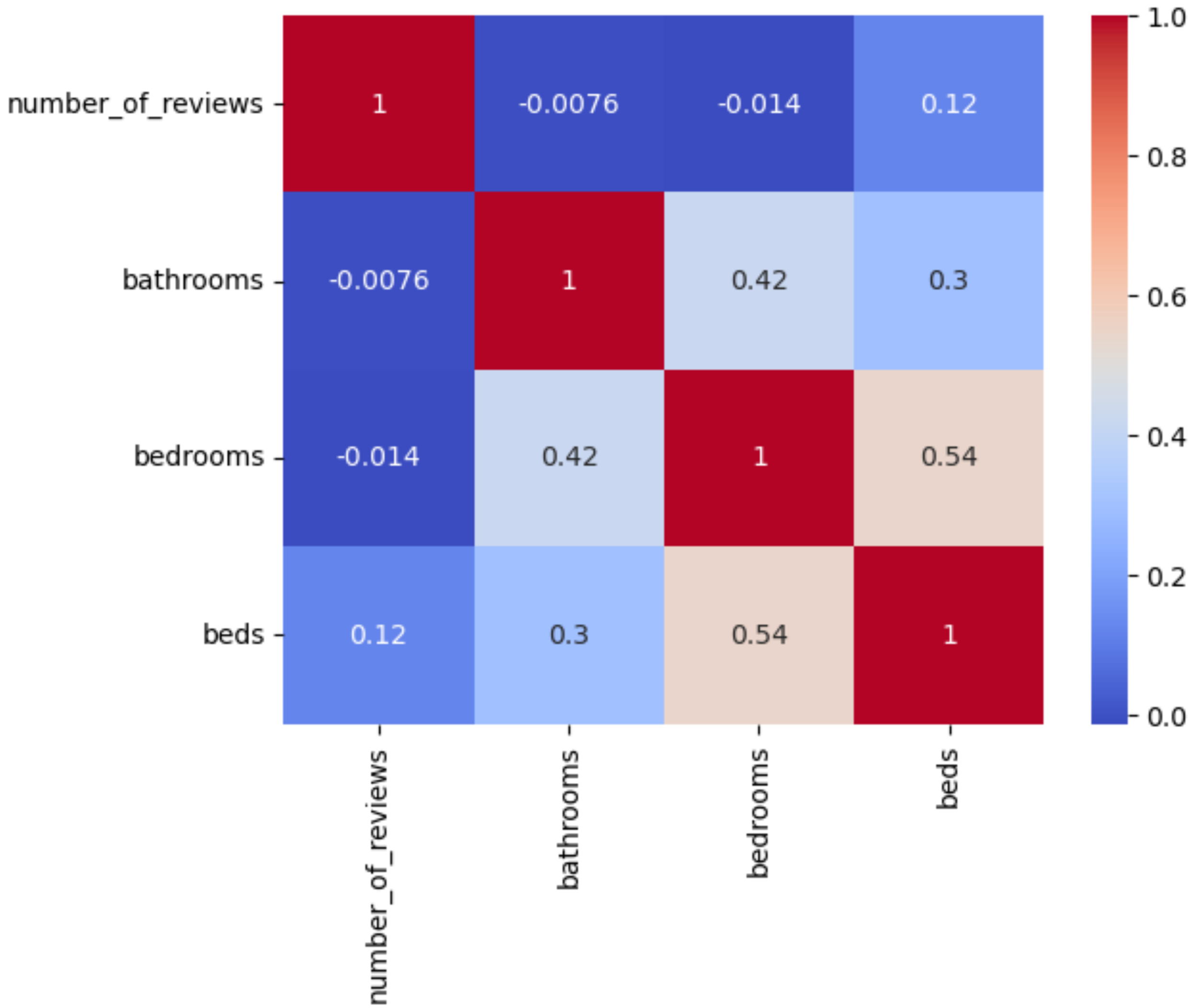

id
host_id
host_total_listings_count
longitude
bathrooms
beds
maximum_nights
maximum_minimum_nights
maximum_maximum_nights
maximum_nights_avg_ntm
availability_30
availability_90
number_of_reviews
number_of_reviews_l30d
review_scores_accuracy
review_scores_checkin
review_scores_location
calculated_host_listings_count
calculated_host_listings_count_private_rooms
reviews_per_month
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Clearly there is too much numeric data in the total available set of features displayed. It is impossible to see the subset of desired data. Supplying the dataset into the library can be done, but the library does not know exactly how to deal the data's full range and set of values. It dutifully attempts to process all of it, even though the cost of doing so in a limited area results in poor visualization.

An attempt was made to capture the numeric data only (by selecting types that are numbers), but the set of features is too large to plot cleanly. The best way to show a correlation heatmap in this case is to extract a subset of data to study. This will show if the limited subset of data has any inter-relationships. The operation of selecting a subset of data has to be done judiciously by the analyst to determine what features are relevant to compare or not.

In [40]:
```
listings_df_subset = listings_df[['number_of_reviews', 'bathrooms', 'bedrooms', 'beds']]
listings_df_corr = listings_df_subset.corr()
sns.heatmap(listings_df_corr, annot=True, cmap='coolwarm')
```

Out[40]:
```
<Axes: >
```



As expected, there is positive correlation between features like the quantity of beds, bedrooms and bathrooms. There is very little or negative correlation to the number of reviews feature. This kind examination can yield insights about which features are most likely to contribute to predicting the target variable. Possibly irrelevant features can be dropped from the feature matrix and not used as inputs to train a model. This could help improve the model's accuracy or make it more robust to future unseen data or speed up the training process because of a reduced dataset.

Analysis

This section covers a number of techniques that can be used in analyzing the data. This started in the previous section that used a number of existing outputs and graphs to show how the data was structured, organized and the theory of how to examine different aspects of the data.

The next set of steps to take involve improving the datasets to be more readable, displayable and usable when it comes time to feed it into a model. This aspect of data analysis also covers potentially looking at the volumn of data to determine how to change that.

The book uses the term "data scrubbing" for cleaning and manipulating the data in preparation for data analysis. The notion of filling in blanks or replacing poor values might have to be balanced by erasing examples or features.

A lot depends on the algorithms picked for model training. 1] Some require the concept of one-hot-encoding (using 1-hot, MCAR, MAR), 2] examining statistics for multi-collinearity, 3] and preparing for when and how to use dimension reduction (which will be the focus of the experiments done in the next section).

One-Hot Encoding

As mentioned previously, data scrubbing is the activity of cleaning up the dataset in preparation for its use. This is primarily to make its consumption by a training algorithm as seemless as possible. Considerations of what the algorithm's purpose is, regression or classification, must be taken into account. For regression based models, the target feature variable being predicted is a numeric continuous value (usually an integer or floating point number). For classification based models, the target feature variable being predicted is a discrete category or class (usually a character string label).

A one-hot encoding is a technique for converting a categorical variable (usually character string based) into a numeric form. This allows algorithms that process only numeric columns to operate on this data. Typically, the conversion is for a categorical variable to be converted into a binary form, of zero and one, for false and true respectively.

Examining the dataframe's info() earlier showed each column and its type. Features listed as 'object', indicates a non-numerical value. These are good candidates for conversion. The details of the specific columns has to be examined to see what the range of values can be set to. This will determine how many additional columns will be expanded into the feature matrix. This can significantly grow the size of the matrix, so applying this encoding has to be done strategically.

A simple optimization here is relying on multi-collinearity. The statistical concept being "the ability to predict a variable based on the value of other variables" (as defined in the book). Basically, it relies on the process of elimination from a limited set of n values that a variable can take on. If the other n-1 values are accounted for in the one-hot-encoding, then the nth value from the set's expansion is not required by virtue of the others being labeled true or false accordingly.

An example of doing this can be achieved with the Panda's get_dummies() method and using the drop_first argument to remove an expendable column. This can be seen in the expansion of the different neighbourhoods, where the n-1 values are 0 for the neigbourhood that is not relevant for the row, and set to 1 for the neighbourhood that is relevant for that row.

In [49]:
```
listings_df_onehot = pd.get_dummies(listings_df, columns=['neighbourhood_group_cleansed', 'neighbourhood'],
listings_df_onehot.head(2)
```

Out[49]:

	id	listing_url	scrape_id	last_scraped	source	name	description	neighborhood_overview	
0	3176	https://www.airbnb.com/rooms/3176	20240622134424	2024-06-23	city scrape	Fabulous Flat in great Location	This beautiful first floor apartment is situa...	The neighbourhood is famous for its variety of...	https://a0 pictures/
1	9991	https://www.airbnb.com/rooms/9991	20240622134424	2024-06-23	city scrape	Geourgeous flat - outstanding views	4 bedroom with very large windows and outstand...	Prenzlauer Berg is an amazing neighbourhood wh...	https://a0 pictures/4

2 rows × 113 columns

Missing Data

Cleaning the data can also includes efforts to remove values or fill in missing values. The book lists three categories of missing values. "Missing completely at random (MCAR), missing at random (MAR), and nonignorable."

These three categories require different approaches because of their different root causes. MCAR values have no relationship to other values. MAR values are related (not to themselves) to other variable values. Nonignorable values are related to the value itself (for what the question is asking) or related to the significance of the data.

Alternate methods of data collection have to be designed to work around missing data due to these three categories. Better approaches to impute the data can be used to account for the root cause of why the data is missing. Filling in the missing data with the appropriate values can be informed by the context of the questions that caused the data to be missing in the first place.

Examining the data to count up the amount of missing data is a good place to start for determining what can be done to fill-in or drop the missing values. One place to start with would include simple counting:

In [53]:
```
listings_df.isnull().sum()
```

Out[53]:
```
id                                                 0
listing_url                                        0
scrape_id                                          0
last_scraped                                       0
source                                             0
                                                ...
calculated_host_listings_count                     0
calculated_host_listings_count_entire_homes        0
calculated_host_listings_count_private_rooms       0
calculated_host_listings_count_shared_rooms        0
reviews_per_month                               3238
Length: 75, dtype: int64
```

This shows a way to examine each category that is missing in order to determine what to do. Examples of that can be to use the mean or mode to fill in missing values. Alternatively, if it can be justified, filling in with a constant, possibly zero can be used. Or just dropping the data can be used.

In [58]:
```
listings_df['reviews_per_month'].fillna((listings_df['reviews_per_month'].mean()), inplace=True)
listings_df['reviews_per_month'].fillna((listings_df['reviews_per_month'].mode()), inplace=True)
listings_df['reviews_per_month'].fillna((0), inplace=True)
listings_df.dropna(axis=0, how='any', subset=None, inplace=True)
```

Using these methods above, the data can be cleaned up, and values adjusted so they are best prepared for algorithm processing. The next logical thing to account for is possibly reducing the overall number of parmeters or dimensions of the data. That is the subject of the next set of experiments that demonstrate how to do it.

Experimental Results

The primary source refers to dimension reduction as "descending dimension algorithms". As has been mentioned earlier, these steps help reduce the quantity of data to input into a training algorithm. More importantly, once the number of dimensions increases beyond what can be graphed, it becomes very difficult to visualize the data and understand it.

The goal so far has been to explore the datasets and exploratory graphs have been used to visualize the datasets. The data also has to be shared and understood by others, and this involves creating explanatory graphs that can be consumed by readers. The graph types might be the same, but the purpose and audience differs. Additionally, explanatory graphs can only contain a subset of data that can be visualized. Hence the need to reduce dimensions.

PCA

The idea behind data scrubbing is to do the required steps to distill the input feature set into the minimal and cleanest dataset possible. This way, the data is optimized for processing by different model algorithms. This idea falls under the the heading of "Principal Component Analysis" or also known as "general factor analysis".

The basic idea in PCA is to examine subsets of the datapoints available in the dataset such that there is maximal variance amongst the datapoints being compared. This can be achieved in different ways, but one way is to plot data against its regression line. Then project the data onto an orthogonal line and examine the variance on the new axis. For situations where datapoints are not yielding increased variance in the projection, then those datapoints can become candidates for removal. The goal being to retain those datapoints on the principal component that increased variance in comparison to the original values that appeared on the normal x and y axis.

An example using the synthetic dataset helps to demonstrate the process. It uses the advertising dataset [3] csv file. It is imported, cleaned, scaled, reduced in dimension and visualized.

In [66]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
advertising_df = pd.read_csv('advertising.csv')
advertising_df.head()
```

Out[66]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

In [68]:

```
# The data is cleaned up by removing character string columns and binary encoded Male column.
del advertising_df['Ad Topic Line']
del advertising_df['City']
del advertising_df['Country']
del advertising_df['Timestamp']
del advertising_df['Male']
# Then the data is scaled with zero mean and unit variance (normal distribution)
from sklearn.preprocessing import StandardScaler
advertising_df_scaler = StandardScaler()
advertising_df_scaler.fit(advertising_df)
advertising_df_scaled_data = advertising_df_scaler.transform(advertising_df)
print(advertising_df.shape)
```

```
(1000, 5)
```

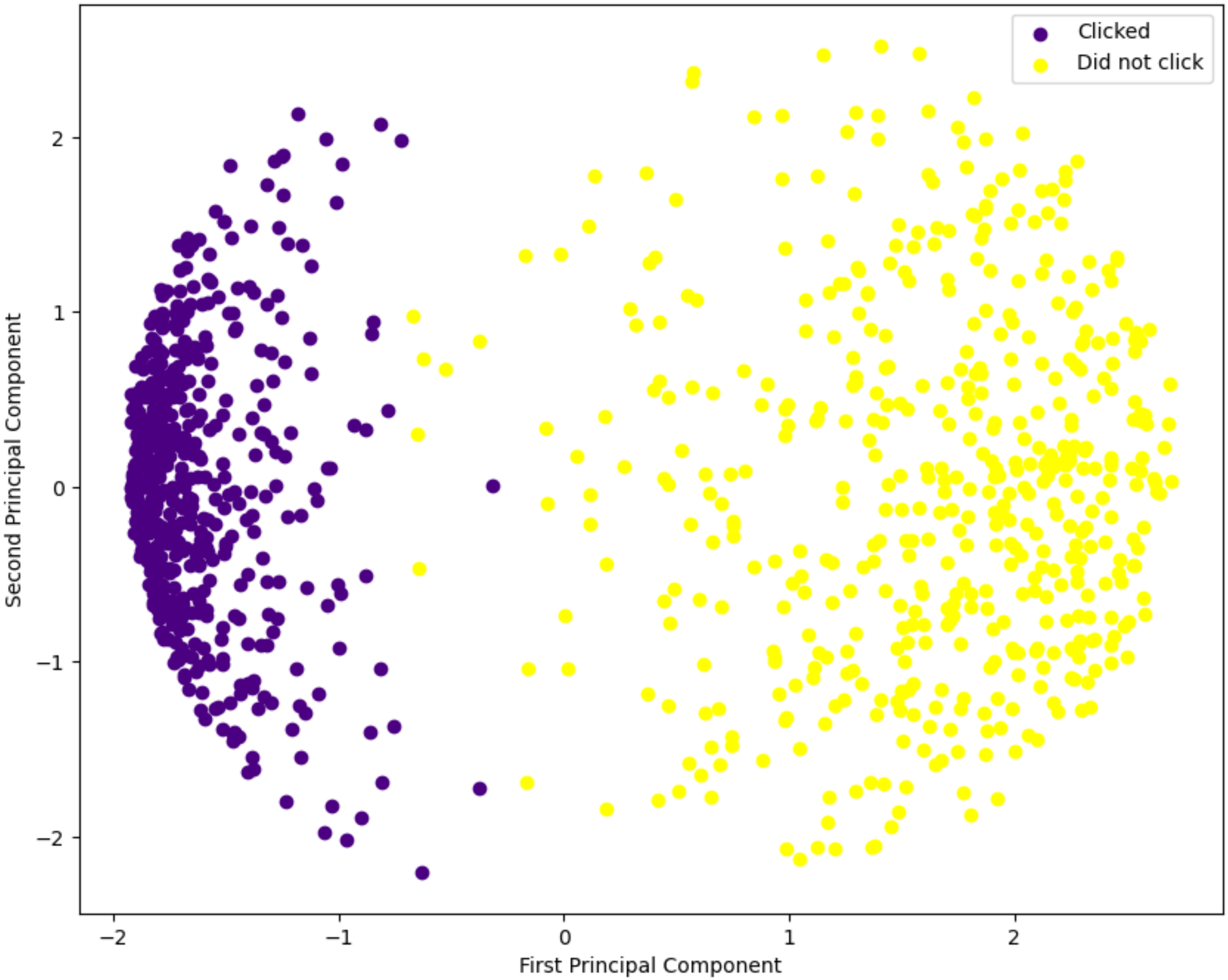
Now the PCA algorithm is applied followed by visualization. The previous cell showed that the deletion of columns caused the dataset to go from 10 to 5 columns. This will be further reduced by PCA next.

In [71]:
```python
from sklearn.decomposition import PCA
ad_df_pca = PCA(n_components=2)
# These two components need to be fit and then all data recreated by using transform
ad_df_pca.fit(advertising_df_scaled_data)
ad_df_pca_scaled = ad_df_pca.transform(advertising_df_scaled_data)
print(f'shape of the scaled pca data is {ad_df_pca_scaled.shape}')
```

```
shape of the scaled pca data is (1000, 2)
```

Now this scaled data is visualized as follows. The colors represent the target variable outcome of 'clicked on ad' or 'did not click on ad'. The shape is changed and 5 columns are being represented as 2.

In [74]:
```python
plt.figure(figsize=(10,8))
legend = advertising_df['Clicked on Ad']
colors = {0:'#4B0082', 1:'#FFFF00'}
labels = {0:'Clicked', 1:'Did not click'}
for t in np.unique(legend):
    ix = np.where(legend == t)
    plt.scatter(ad_df_pca_scaled[ix,0], ad_df_pca_scaled[ix,1], c=colors[t], label=labels[t])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.legend()
plt.show()
```


Clicked
Did not click
Second Principal Component
First Principal Component
2
1
0
−1
−2
−2
−1
0
1
2

This analysis shows the effects of using PCA in preparation for additional analysis using the k-Means Clustering algorithm.

k-Means Clustering

Another technique (a form of unsupervised learning because there is no target variable labeled with ground truth for training purposes) is k-means clustering. Its purpose is to segregate datapoints into subgroups without any knowledge of the category or class the datapoint belongs to.

The algorithm requires seeding with an initial k number of clusters to form. Each cluster utilizes a random centroid and proximity of a datapoint to the centroid (a distance calculation) dictates membership to a particular cluster. There can be an update of the centroid based on the mean of the new cluster. There can be shifting of points between clusters if the proximity to a newer centroid helps improve the cohesion of that datapoint to a cluster. This process iterates until all datapoints are grouped and remain stable in a cluster.

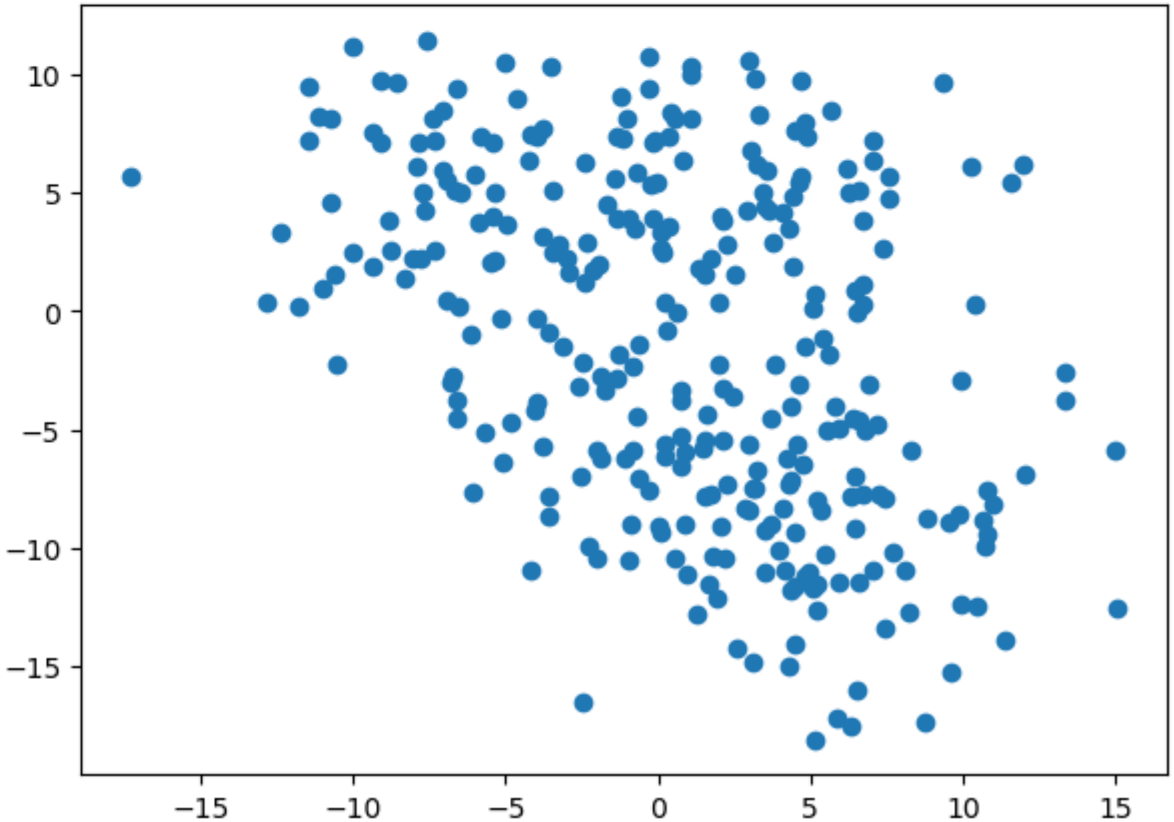
This process is illustrated with a synthetic dataset to show how datapoints form into clusters.

In [79]:

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
%matplotlib inline
X, y = make_blobs(n_samples=300, n_features=2, centers=4, cluster_std=4, random_state=10)
plt.figure(figsize=(7,5))
plt.scatter(X[:,0],X[:,1])
```

Out[79]:

```
<matplotlib.collections.PathCollection at 0x73c0b78413c0>
```



Now the k-means algorithm will be used in the model and the synthetic data (stored in matrix X) will be fit (or trained). This allows for doing a prediction on the input data to generate the centroids. These centroids can be used to plot on top of the of data to see in which clusters the data points reside.

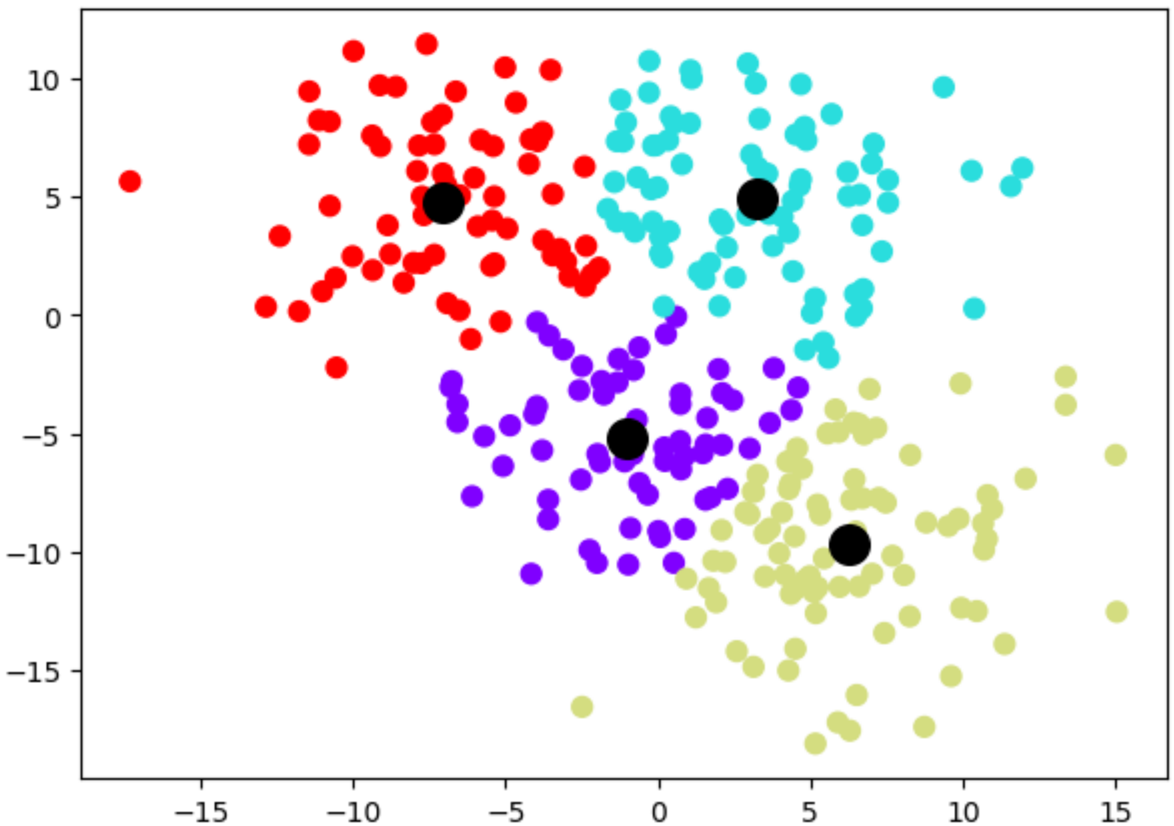
This ultimately is a method for reducing the dimension associated with the total number of rows in the dataset. This dataset can be shrunk down to just those four centroids representing a summary of a number of rows that all affiliate with the same cluster. Since the large subset of points all fall into a limited number of clusters, the representative point in each cluster (the centroid) can be a proxy for that large subset and be the single row used in a subsequent supervised training algorithm.

In [82]:
```
k_means_model = KMeans(n_clusters=4, n_init=10)
k_means_model.fit(X)
k_means_model_predict = k_means_model.predict(X)
k_means_model_centroids = k_means_model.cluster_centers_
print(k_means_model.cluster_centers_)
```

```python
plt.figure(figsize=(7,5))
plt.scatter(X[:,0],X[:,1],c=k_means_model_predict, s=50, cmap='rainbow')
plt.scatter(k_means_model_centroids[:,0], k_means_model_centroids[:,1], c='black', s=200, alpha=1)
```

```
[[-1.01492539 -5.23271226]
 [ 3.23209343  4.94623366]
 [ 6.24946744 -9.70847466]
 [-7.03502629  4.80055552]]
```

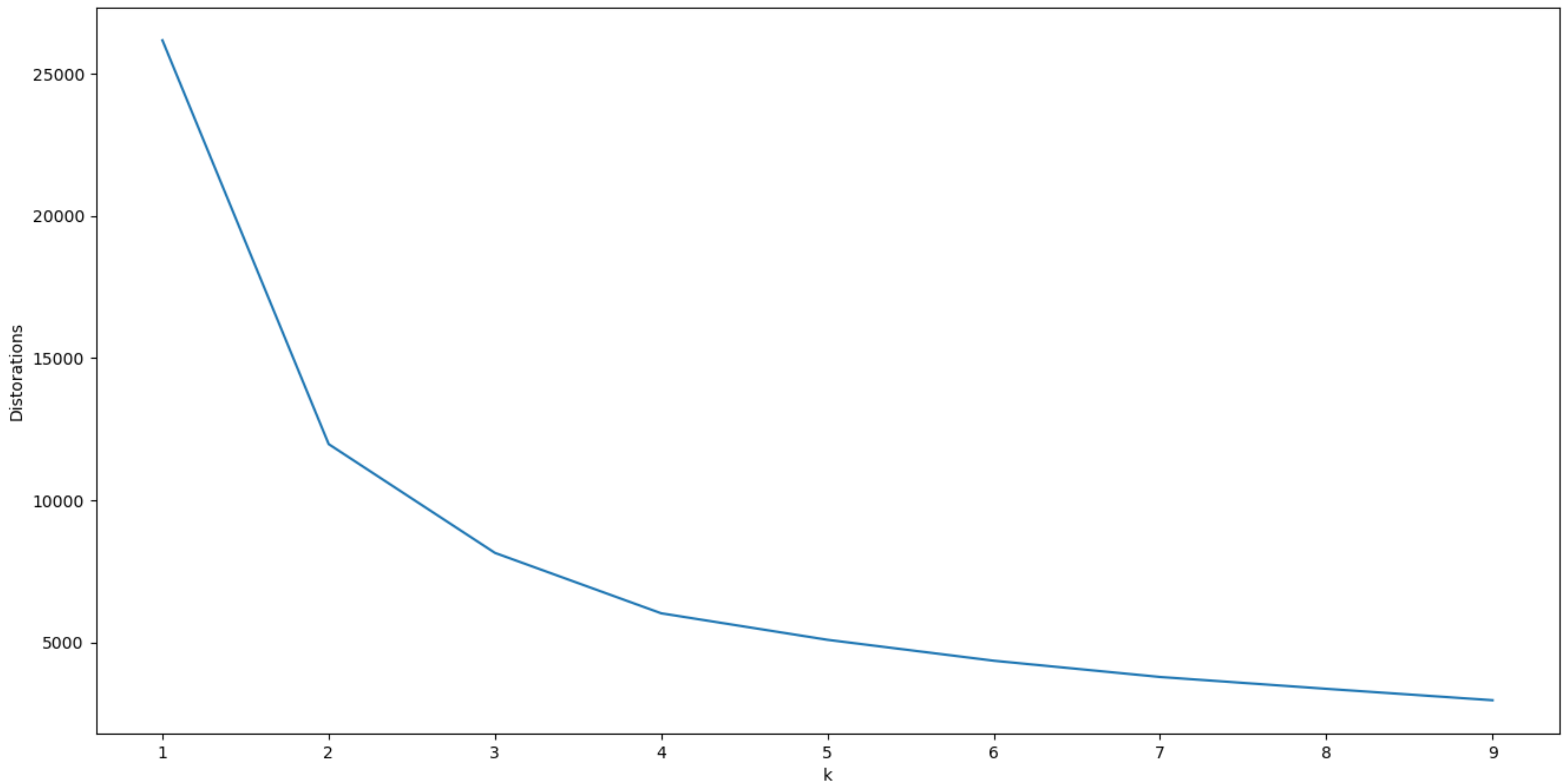
Out[82]: <matplotlib.collections.PathCollection at 0x73c0b7883c70>



In this case, the value of k=4 was selected as centers in the beginning when the syntheic blob data was created to analyze. But is that value of k correct or optimal? To make that determination, there is a value of inertia calculated in the model along with the centroids that were generated. This value can be utilized as a measure of 'distortion' and plotted into a Scree plot for finding the optimal value of k. This plot captures the degree of scattering (in essense the variance of the datapoints from the centroid) by comparing the distortion for each variation of clusters. This distortion measure is the Euclidean distance between the centroid and cluster datapoints.

The optimal value of k is the elbow in the Scree plot, which indicates the variation has subsided. The distortion value does not change much or changes minimally going left to right. This can be seen in the Scree plot shown and lines up with the fact that the synthetic data was created with four blobs. This was generated by cycling through a range of k values and running the algorithm to fit the data and gathering up the inertia value from the models generated. These collected values are then plotted.

In [85]:
```
distortions = []
K=range(1,10)
for k in K:
    model = KMeans(n_clusters=k, n_init=10)
    model.fit(X,y)
    distortions.append(model.inertia_)
plt.figure(figsize=(16,8))
plt.plot(K, distortions)
plt.xlabel('k')
plt.ylabel('Distorations')
plt.show()
```



It may have been noticed in the earlier synthetic data generation that the blobs were created with a built in function that returned two variables. Namely X and y. The X is the feature matrix and y is the column vector of target variables that are the value (or label) being predicted. This concept of having the data available in these two variables comes from linear algebra. In its most simple form, the familiar straight-line regression curve attempting to be fit to the data would have a slope and y-intercept, where values in the range x would be used to generated output values y (y=mx+b, where b is some offset constant).

When this is generalized to many features and examples, then a system of linear equations has to be solved to generate the outputs in a vector y. This can go further as higher order (degree) polynomial is needed to fit the supplied data to solve the equation.

When training a model, the full set of data in X and y is not used directly. This could lead to a model that is overfit on the training data and essentially creates a perfect fit for only that dataset. It will not generalize to new data the model has not seen before and it will not be able to make an accurate prediction. The balance between a model that overfits versus underfits is known as the bias-variance tradeoff.

Data Partitioning

An operation that is done on a total dataset is to partition it into two parts. This is known as 'split validation'. The first part is known as the training set and is used to train the model. The second part is known as the test set and is used to determine the accuracy of the model created from the training data. This allows for determining how well the model performs on data is has not seen before. This split is typically done roughly as 70 or 80 % for training data and 30 or 20 % for test data.

Another split on the test data is also done to create a validation set. The purpose of this is to see how the model performs on unknown data to help guide how much optimization needs to be done on the model to make it more accurate. Then the final test data partition is used to determine the model's prediction error. This splitting is accomplished using built in library functionality. This is shown using the same blob data generated from before that appears in X and y for the 300 samples.

In [89]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=10, shuffle=True)
print(X_train.shape)
print(X_test.shape)
# to do a validation set partition, split up the test set in half for validation purposes
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5)
print(X_test.shape)
print(X_val.shape)
print(len(y_test))
print(len(y_val))
```

```
(180, 2)
(120, 2)
(60, 2)
(60, 2)
60
60
```

Conclusions

A number of topics have been covered that have shown different aspects of exploratory data analysis and visualization. The focus has been on building up steps in building a model. The primary source [2] gives an general breakdown of these steps as:

1. import libraries
2. import dataset
3. EDA (and doing necessary visualization)
4. data scrubbing
5. pre-model algorithm (PCA and related discussed above, extends step 4)
6. split validation
7. set algorithm (related to training the model)
8. predict
9. evaluate
10. optimize (hyperparameter tuning)

Most of these steps have been examined in this article (except steps 7-10).

Step 9 on evaluation is important to determine the quality of the results of the prediction. Recall the attempt to use the Scree plot as a measure of evaluating what the optimal number of clusters should be for the k-means algorithm. Similar other techniques exist to evalute the results of the prediction (depending on if it is a regression model yielding a numeric value or a classification model yielding a class label). Some items to examine include the following measures. A number of these will be covered in future articles.

Accuracy = (number predicted correctly / number of cases) is a simple account of correct predictions.

Confusion matrix (or error matrix) is used to give the number of false-positives and false-negatives. This can provide the total misclassification by using (False-positives + False-negatives)/Total data points. The complement of this number is the model's accuracy score.

Classification report provides 4 metrics: precision, recall, f1-score, and support. The defintions are:

precision = number of true positives / number of predicted positive cases So a high precision score indicates low occurance of false-positives.

recall = number of true positives / number of action positive cases So this tells how many of the positive outcomes were correctly classified as positive f1-score = weighted average of precision and recall. Used to compare between models.

support = count of number of positive and negative cases

For most regression models that are predicting a numeric target variable value, the most common measures used are MAE (mean absolute error) and RMSE (root mean square error). The MAE is the average error of data points from the regression line. The RMSE measures the standard deviation of the prediction errors.

This concludes the survey of EDA techniques and metrics to examine when preparing datasets for model consumption. Future articles will revisit these ideas and demonstrate additional techniques found in the libraries for managing the datasets.

References

1. Patel, Amit. EDA and Visualization Part One. https://ap20.github.io/nnj/NL/edutechrev/ExploreDAandVisPart1_apatel.html
2. Theobald, Oliver. Machine Learning with Python, 2024. scatterplotspress.com. https://scatterplotpress.teachable.com/
3. This project was part of Python for Data Science and Machine Learning Bootcamp on Udemy by Jose Portilla. advertising dataset https://github.com/tyonas9/Logistic-Regression-ML-Model
4. Melbourne Housing. https://www.kaggle.com/datasets/anthonypino/melbourne-housing-market
5. AirBnB. https://www.kaggle.com/code/mahmoudkhater99/berlin-airbnb-eda/notebook
6. Kickstarter. https://www.kaggle.com/datasets/tayoaki/kickstarter-dataset

Appendix

Additional Datasets

This section covers an additional group of datasets mentioned in the reference. The same operations as done in the Theory section are conducted on each of these datasets to gain similar insights before applying the techniques of the Analysis section to curate the datasets.

This appendix covers the remaining datasets related to kickstarter projects and Melbourne (Austrailia) housing data. The synthetic advertising data was treated earlier in the PCA section.

In [101…
```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
# gather up all the datasets
kickstarter_df = pd.read_csv('18k_Projects.csv', low_memory=False)
#advertising_df = pd.read_csv('advertising.csv')
melbournehousing_df = pd.read_csv('Melbourne_housing_FULL.csv')
```

In [103…
```
kickstarter_df.shape
```

Out[103…
```
(18142, 35)
```

In [105…
```
kickstarter_df.columns
```

Out[105…
```
Index(['Id', 'Name', 'Url', 'State', 'Currency', 'Top Category', 'Category',
       'Creator', 'Location', 'Updates', 'Comments', 'Rewards', 'Goal',
       'Pledged', 'Backers', 'Start', 'End', 'Duration in Days',
       'Facebook Connected', 'Facebook Friends', 'Facebook Shares',
       'Has Video', 'Latitude', 'Longitude', 'Start Timestamp (UTC)',
       'End Timestamp (UTC)', 'Creator Bio', 'Creator Website',
       'Creator - # Projects Created', 'Creator - # Projects Backed',
       '# Videos', '# Images', '# Words (Description)',
       '# Words (Risks and Challenges)', '# FAQs'],
      dtype='object')
```

In [107…
```
kickstarter_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18142 entries, 0 to 18141
Data columns (total 35 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Id                                18142 non-null  int64  
 1   Name                              18142 non-null  object 
 2   Url                               18142 non-null  object 
 3   State                             18142 non-null  object 
 4   Currency                          18142 non-null  object 
 5   Top Category                      18142 non-null  object 
 6   Category                          18142 non-null  object 
 7   Creator                           18142 non-null  object 
 8   Location                          18142 non-null  object 
 9   Updates                           18142 non-null  int64  
 10  Comments                          18142 non-null  int64  
 11  Rewards                           18142 non-null  int64  
 12  Goal                              18142 non-null  int64  
 13  Pledged                           18142 non-null  int64  
 14  Backers                           18142 non-null  int64  
 15  Start                             18142 non-null  object 
 16  End                               18142 non-null  object 
 17  Duration in Days                  18142 non-null  int64  
 18  Facebook Connected                18142 non-null  object 
 19  Facebook Friends                  12290 non-null  float64
 20  Facebook Shares                   18142 non-null  int64  
 21  Has Video                         18142 non-null  object 
 22  Latitude                          9803 non-null   float64
 23  Longitude                         9803 non-null   float64
 24  Start Timestamp (UTC)             18142 non-null  object 
 25  End Timestamp (UTC)               18142 non-null  object 
 26  Creator Bio                       18142 non-null  object 
 27  Creator Website                   11475 non-null  object 
 28  Creator - # Projects Created      18142 non-null  int64  
 29  Creator - # Projects Backed       13898 non-null  float64
 30  # Videos                          18041 non-null  float64
 31  # Images                          18142 non-null  int64  
 32  # Words (Description)             18142 non-null  int64  
 33  # Words (Risks and Challenges)    18041 non-null  float64
 34  # FAQs                            18142 non-null  int64  
dtypes: float64(6), int64(13), object(16)
memory usage: 4.8+ MB
```

In [109…

```
kickstarter_df.describe()
```

Out[109…

	Id	Updates	Comments	Rewards	Goal	Pledged	Backers	Duration in Days	
count	1.814200e+04	18142.000000	18142.000000	18142.000000	1.814200e+04	1.814200e+04	18142.000000	18142.000000	1
mean	1.073471e+09	3.368647	34.243027	10.042002	2.653121e+04	1.102364e+04	138.070279	31.398468	
std	6.181166e+08	5.547975	539.161283	5.889806	7.583874e+05	7.855300e+04	633.787780	10.058827	
min	1.061440e+05	0.000000	0.000000	2.000000	1.000000e+02	1.000000e+00	1.000000	1.000000	
25%	5.385158e+08	0.000000	0.000000	6.000000	2.000000e+03	2.600000e+02	7.000000	29.000000	
50%	1.078580e+09	1.000000	0.000000	9.000000	5.000000e+03	1.722000e+03	29.000000	30.000000	
75%	1.606254e+09	5.000000	3.000000	12.000000	1.500000e+04	6.335000e+03	89.000000	32.000000	
max	2.147445e+09	128.000000	30341.000000	131.000000	1.000000e+08	6.224955e+06	35383.000000	60.000000	4

In [111…

```
kickstarter_df.head()
```

Out[111…

	Id	Name	Url	State	Currency	Top Category	Category	Creator	Location	Updates	...	Tim
0	1007121454	Nail Art and Photos Printed on your Nails w/ E...	https://www.kickstarter.com/projects/137019948...	failed	USD	Art	Art	Dodie Egolf	Puyallup	0	...	201 01:5
1	2032015036	Cold Again	https://www.kickstarter.com/projects/737783165...	failed	USD	Film & Video	Short Film	James Jacobs	Boston	0	...	201 02:3
2	733782855	Uchu Bijin Jewelry	https://www.kickstarter.com/projects/uchubijin...	failed	USD	Fashion	Fashion	Uchu Bijin	New York	1	...	201 01:2
3	514687871	Poetically Speaking: Stories of Love, Triumph ...	https://www.kickstarter.com/projects/tylicee/p...	failed	USD	Publishing	Poetry	Tylicee Mysreign	Detroit	0	...	201 01:1
4	683545993	Stranger Travels: Teachings from the Heart of ...	https://www.kickstarter.com/projects/197270300...	failed	USD	Publishing	Nonfiction	Ian Driscoll	Pucallpa	0	...	201 01:1

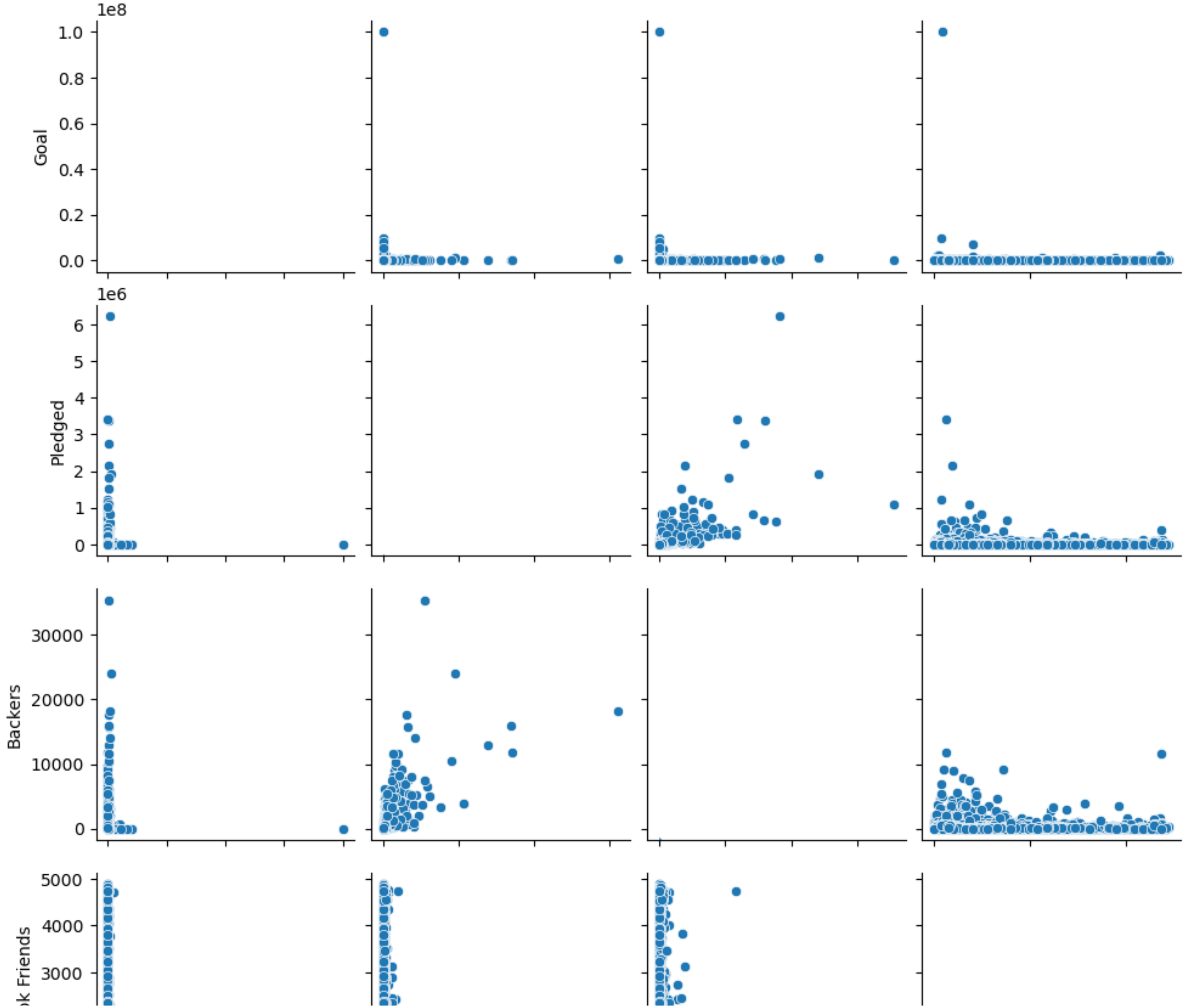
5 rows × 35 columns

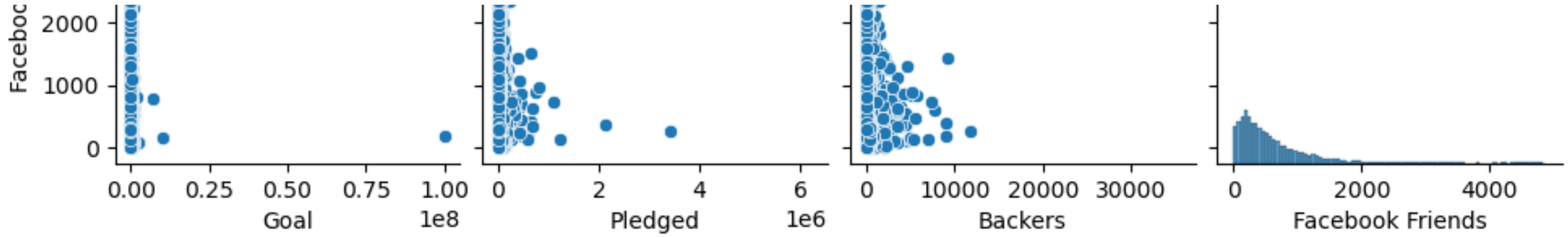
In [113…

```
sns.pairplot(kickstarter_df, vars=['Goal', 'Pledged', 'Backers', 'Facebook Friends'])
```

Out[113…

```
<seaborn.axisgrid.PairGrid at 0x73c0b7183520>
```


1e8
1e6
Goal
Pledged
Backers
ok Friends
1.0
0.8
0.6
0.4
0.2
0.0
6
5
4
3
2
1
0
30000
20000
10000
0
5000
4000
3000

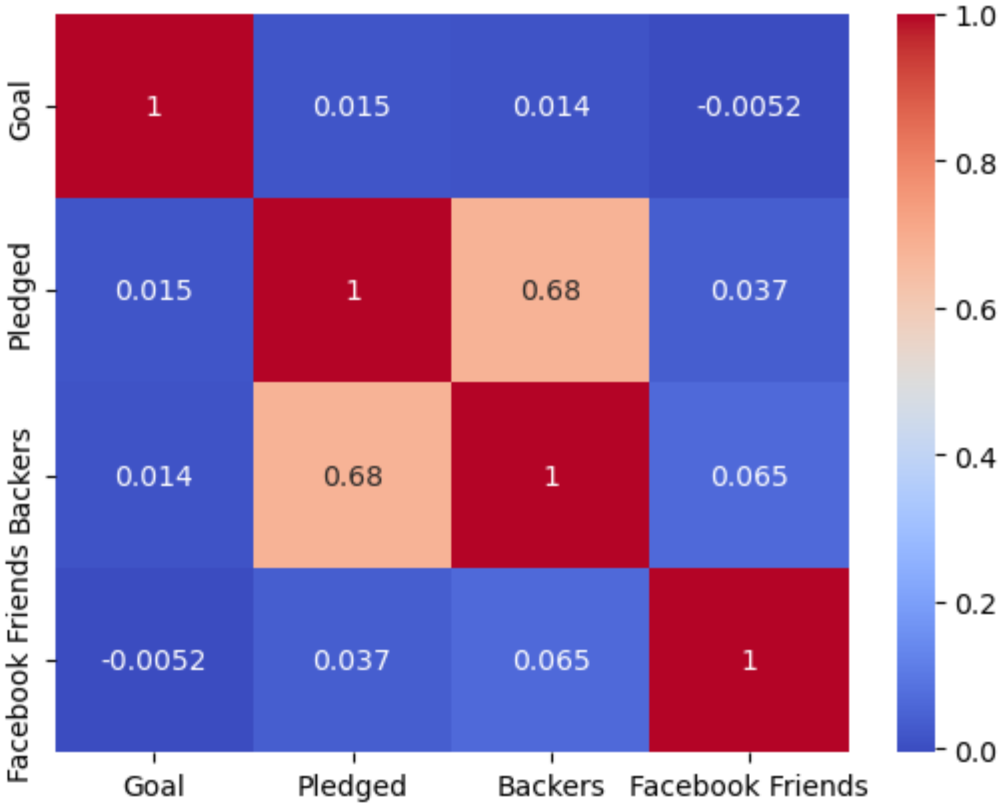


In [115…

```
kickstarter_df_subset = kickstarter_df[['Goal', 'Pledged', 'Backers', 'Facebook Friends']]
sns.heatmap(kickstarter_df_subset.corr(), annot=True, cmap='coolwarm')
```

Out[115…

```
<Axes: >
```



In [117…

```
melbournehousing_df.shape
```

Out[117…

```
(34857, 21)
```

In [119…
```
melbournehousing_df.columns
```

Out[119…
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

In [121…
```
melbournehousing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         34857 non-null  object 
 1   Address        34857 non-null  object 
 2   Rooms          34857 non-null  int64  
 3   Type           34857 non-null  object 
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object 
 6   SellerG        34857 non-null  object 
 7   Date           34857 non-null  object 
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object 
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object 
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
```

In [123…
```
melbournehousing_df.describe()
```

Out[123…

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize
count	34857.000000	2.724700e+04	34856.000000	34856.000000	26640.000000	26631.000000	26129.000000	23047.000000
mean	3.031012	1.050173e+06	11.184929	3116.062859	3.084647	1.624798	1.728845	593.598993
std	0.969933	6.414671e+05	6.788892	109.023903	0.980690	0.724212	1.010771	3398.841946
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	6.350000e+05	6.400000	3051.000000	2.000000	1.000000	1.000000	224.000000
50%	3.000000	8.700000e+05	10.300000	3103.000000	3.000000	2.000000	2.000000	521.000000
75%	4.000000	1.295000e+06	14.000000	3156.000000	4.000000	2.000000	2.000000	670.000000
max	16.000000	1.120000e+07	48.100000	3978.000000	30.000000	12.000000	26.000000	433014.000000

In [125…

```
melbournehousing_df.head()
```

Out[125…

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	La
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	3067.0	...	1.0	1.0	
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	3067.0	...	2.0	1.0	
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	

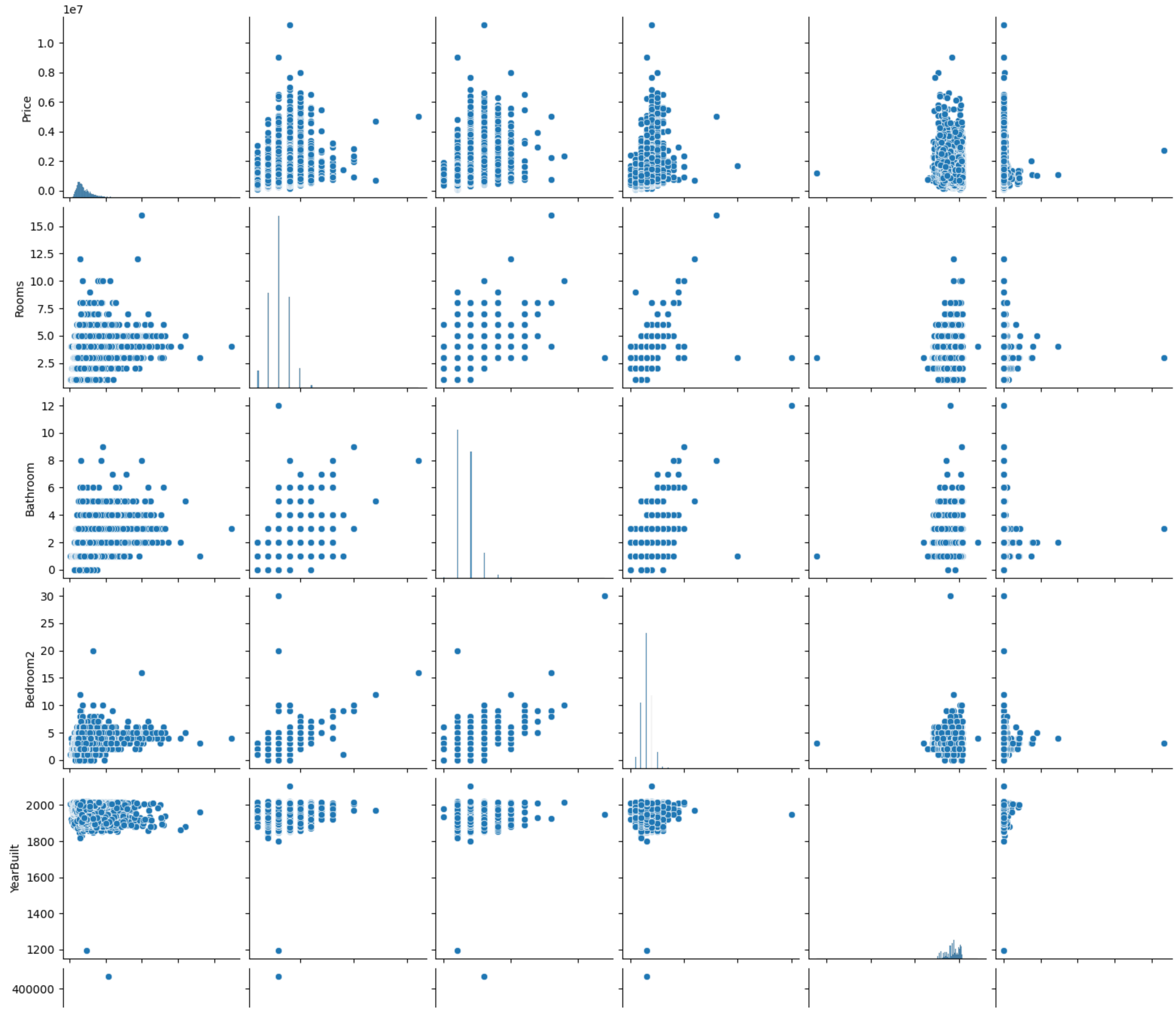
5 rows × 21 columns

In [127…

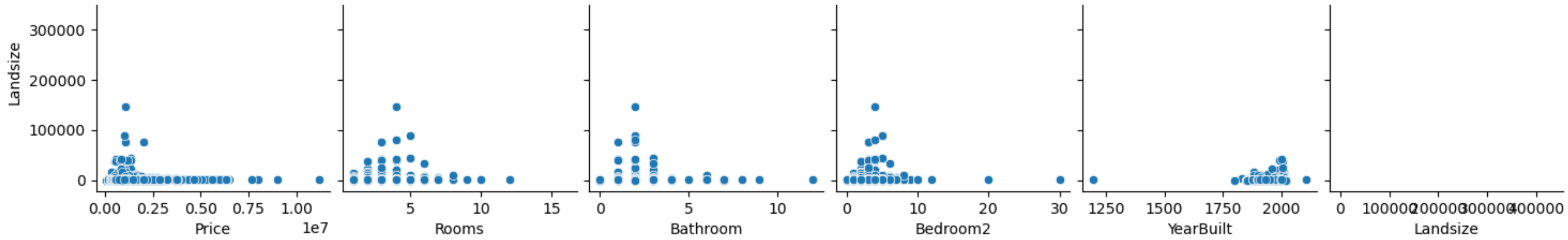
```
sns.pairplot(melbournehousing_df, vars=['Price', 'Rooms', 'Bathroom', 'Bedroom2', 'YearBuilt', 'Landsize'])
```

Out[127…

```
<seaborn.axisgrid.PairGrid at 0x73c06f320250>
```



1e7
Price
Rooms
Bathroom
Bedroom2
YearBuilt



In [129…
```
melbournehousing_df_subset = melbournehousing_df[['Price', 'Rooms', 'Bathroom', 'Bedroom2', 'YearBuilt', 'L
sns.heatmap(melbournehousing_df_subset.corr(), annot=True, cmap='coolwarm')
```

Out[129…
```
<Axes: >
```