

**ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ**



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS



ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΕΠΙΧΕΙΡΗΜΑΤΙΚΗ ΕΥΦΥΪΑ ΚΑΙ ΑΝΑΛΥΣΗ ΜΕΓΑΛΩΝ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ:

Data Warehousing and Data Mining

Επιμέλεια εργασίας:

Ιωάννης Καλαντζής (8200235)

Οδυσσέας Σπυρόπουλος (3200183)

Καθηγητής: Χατζηαντωνίου Δαμιανός

Αθήνα, Ιανουάριος 2024

Data Extraction and Cleaning

Για την εύρεση των δεδομένων μας, χρησιμοποιήσαμε τον ιστότοπο της Kaggle (<https://www.kaggle.com/>). Συγκεκριμένα, εργαστήκαμε πάνω σε ένα dataset, το οποίο αποτελείται από καταχωρήσεις κατοικιών-ακινήτων, στις Ηνωμένες Πολιτείες της Αμερικής. Ο σύνδεσμος για το συγκεκριμένο online dataset είναι ο ακόλουθος: https://www.kaggle.com/datasets/austinreese/usa-housing-listings?fbclid=IwAR3-tif1QU43u4R1AbmmuK9kZboVGdYfgAG3AGTJK_yEm2b3aK_ZLHbhF6U.

Το παρόν σετ δεδομένων αποτελείται από **385.000 instances** και **22 features**. Πιο συγκεκριμένα, τα δεδομένα που μας παρέχει το dataset είναι τα εξής:

1. **id**: Αναγνωριστικό για κάθε καταχώρηση.
2. **region**: Περιοχή του ακινήτου.
3. **price**: Τιμή ενοικίασης.
4. **type**: Τύπος του ακινήτου(π.χ. διαμέρισμα, σπίτι κλπ.).
5. **sqfeet**: Τετραγωνικά πόδια του ακινήτου.
6. **beds**: Αριθμός υπνοδωματίων.
7. **baths**: Αριθμός μπάνιων.
8. **cats_allowed**: Εάν επιτρέπονται οι γάτες(δυαδική τιμή, 1: Ναι και 0: Όχι).
9. **dogs_allowed**: Εάν επιτρέπονται οι σκύλοι(δυαδική τιμή, 1: Ναι και 0: Όχι).
10. **wheelchair_access**: Εάν υπάρχει πρόσβαση για άτομα με αναπηρικά αμαξίδια(δυαδική τιμή, όπου 1: Ναι και 0: Όχι).
11. **electric_vehicle_charge**: Εάν υπάρχει φόρτιση για ηλεκτρικά οχήματα(δυαδική τιμή, 1: Ναι και 0: Όχι).
12. **comes_furnished**: Εάν προσφέρεται επιπλωμένο(δυαδική τιμή, 1: Ναι και 0: Όχι).
13. **laundry_options**: Επιλογή πλυντηρίου(π.χ. w/d in unit, w/d hookups, laundry on site κλπ.).
14. **parking_options**: Επιλογές στάθμευση (π.χ. carport, attached garage, off-street parking κλπ.).
15. **smoking_allowed**: Εάν επιτρέπεται το κάπνισμα(δυαδική τιμή, 1: Ναι και 0: Όχι).
16. **lat**: Γεωγραφικός πλάτος.
17. **long**: Γεωγραφικό μήκος.
18. **state**: Πολιτεία καταχώρησης.
19. **description**: Περιγραφή.
20. **image_url**: Το URL της εικόνας της καταχώρησης.
21. **url**: Το URL της καταχώρησης.
22. **region_url**: Το URL της περιοχής.

Προφανώς, προτού εισάγουμε τα δεδομένα μας στο Data Warehouse, πρέπει να εξετάσουμε τις τιμές των βασικών features που θα χρειαστούμε για την ανάλυσή μας, να αφαιρέσουμε εκείνα τα features που δεν θα μας ωφελήσουν περαιτέρω. Επιπλέον, θα αφαιρέσουμε ακραίες (outliers) και κενές (null) τιμές.

Για τον σκοπό αυτό χρησιμοποιήσαμε το Jupyter Notebook, ένα εργαλείο open-source, το οποίο είναι δημοφιλές για την ανάλυση δεδομένων και την διαχείρισή τους.

Αφού πλοηγηθήκαμε και εξοικειωθήκαμε με τα δεδομένα, τις στήλες και τις γραμμές του dataset μας, δημιουργήσαμε το αντίστοιχο DataFrame και ξεκινήσαμε την διαδικασία καθαρισμού και επεξεργασίας.

```
In [1]: import pandas as pd
```

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

```
Out[2]: (384977, 22)
```

```
In [3]: pd.options.display.max_columns = 500
```

```
In [4]: df
```

```
Out[4]:
```

	id	url	region	region_url	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed
0	7049044568	https://reno.craigslist.org/apa/d/reno-beautif...	reno / tahoe	https://reno.craigslist.org	1148	apartment	1078	3	2.0	1	1
1	7049047186	https://reno.craigslist.org/apa/d/reno-reduced...	reno / tahoe	https://reno.craigslist.org	1200	condo	1001	2	2.0	0	0
2	7043634882	https://reno.craigslist.org/apa/d/sparks-state...	reno / tahoe	https://reno.craigslist.org	1813	apartment	1683	2	2.0	1	1
3	7049045324	https://reno.craigslist.org/apa/d/reno-1x1-fir...	reno / tahoe	https://reno.craigslist.org	1095	apartment	708	1	1.0	1	1
4	7049043759	https://reno.craigslist.org/apa/d/reno-no-long...	reno / tahoe	https://reno.craigslist.org	289	apartment	250	0	1.0	1	1
...
384972	7049053337	https://reno.craigslist.org/apa/d/reno-2x2-thi...	reno / tahoe	https://reno.craigslist.org	1295	apartment	957	2	2.0	1	1
384973	7049052968	https://reno.craigslist.org/apa/d/sparks-over-...	reno / tahoe	https://reno.craigslist.org	1549	apartment	1034	2	2.0	1	1
384974	7049050454	https://reno.craigslist.org/apa/d/sparks-1mont...	reno / tahoe	https://reno.craigslist.org	1249	apartment	840	2	1.0	1	1

Στην συνέχεια, έπρεπε να επεξεργαστούμε και να αφαιρέσουμε τις κενές τιμές, δηλαδή όσες τιμές είναι NaN.

NULL Elements

```
In [5]: df.isna().any()
```

```
Out[5]: id                False
url                  False
region              False
region_url          False
price               False
type                False
sqfeet              False
beds                False
baths               False
cats_allowed        False
dogs_allowed        False
smoking_allowed     False
wheelchair_access   False
electric_vehicle_charge False
comes_furnished     False
laundry_options     True
parking_options     True
image_url           False
description          True
lat                 True
long                True
state               False
dtype: bool
```

Από το παραπάνω αποτέλεσμα, παρατηρούμε ότι οι στήλες που έχουν κενές τιμές είναι οι στήλες laundry_options, parking_options, description, lat και long. Ας δούμε και το άθροισμα αυτών των τιμών για να μπορέσουμε να τις διαχειριστούμε αναλόγως.

```
In [6]: df.isna().sum()
```

```
Out[6]: id                0
url                  0
region              0
region_url          0
price               0
type                0
sqfeet              0
beds                0
baths               0
cats_allowed        0
dogs_allowed        0
smoking_allowed     0
wheelchair_access   0
electric_vehicle_charge 0
comes_furnished      0
laundry_options      79026
parking_options      140687
image_url            0
description           2
lat                  1918
long                 1918
state                0
dtype: int64
```

Έτσι, επικεντρωθήκαμε στις στήλες laundry_options και parking_options, οι οποίες είχαν και τις περισσότερες τιμές NaN. Προκειμένου να μην διαγράψουμε όλες αυτές τις τιμές από το σετ δεδομένων μας, γιατί θα μίκραινε αρκετά σε μέγεθος, αποφασίσαμε να τις μετατρέψουμε σε 'Unknown'. Τις στήλες long και lat, επειδή είχαν τόσο λίγες παρατηρήσεις με κενές τιμές, αποφασίσαμε να τις αφαιρέσουμε τελείως. Την στήλη με τις περιγραφές, την αφαιρούμε εξολοκλήρου από τα δεδομένα μας, αφού, δεν θα την χρειαζούμαστε περαιτέρω.

```
In [7]: # Replace NaN
df["laundry_options"] = df["laundry_options"].fillna('Unknown')
df["parking_options"] = df["parking_options"].fillna('Unknown')

# Drop NaN
df.dropna(subset=['lat', 'long'], inplace=True)
```

```
In [8]: df.isna().sum()
```

```
Out[8]: id                0
url                  0
region              0
region_url          0
price               0
type                0
sqfeet              0
beds                0
baths               0
cats_allowed        0
dogs_allowed        0
smoking_allowed     0
wheelchair_access   0
electric_vehicle_charge 0
comes_furnished      0
laundry_options      0
parking_options      0
image_url            0
description           0
lat                  0
long                 0
state                0
dtype: int64
```

Επιπλέον, πραγματοποιήσαμε ελέγχους στις στήλες που το θεωρήσαμε αναγκαίο, όπως για παράδειγμα στις στήλες με δυαδικές τιμές για να βεβαιωθούμε ότι όλες οι μεταβλητές είναι πράγματι δυαδικές.

- Check that binary columns are indeed binary

```
In [11]: def check_binary_columns(df, columns):
          for column in columns:
              unique_values = df[column].unique()
              if set(unique_values) != {0, 1}:
                  print(f"Not all values in {column} are binary.")
                  return False

              print(f"All specified columns are binary.")
              return True

          check_binary_columns(df, ['cats_allowed', 'dogs_allowed', 'smoking_allowed', 'wheelchair_access', 'electric_vehicle_charge',
          <
          >

          All specified columns are binary.
```

Out[11]: True

Επίσης, αφαιρέσαμε στήλες οι οποίες, όπως αναφέραμε και παραπάνω, δεν θα ευνοήσουν την ανάλυσή μας.

Dropping not necessary columns

```
In [13]: df.drop(columns=["url", "region_url", "image_url", "description"],axis=1,inplace=True)
```

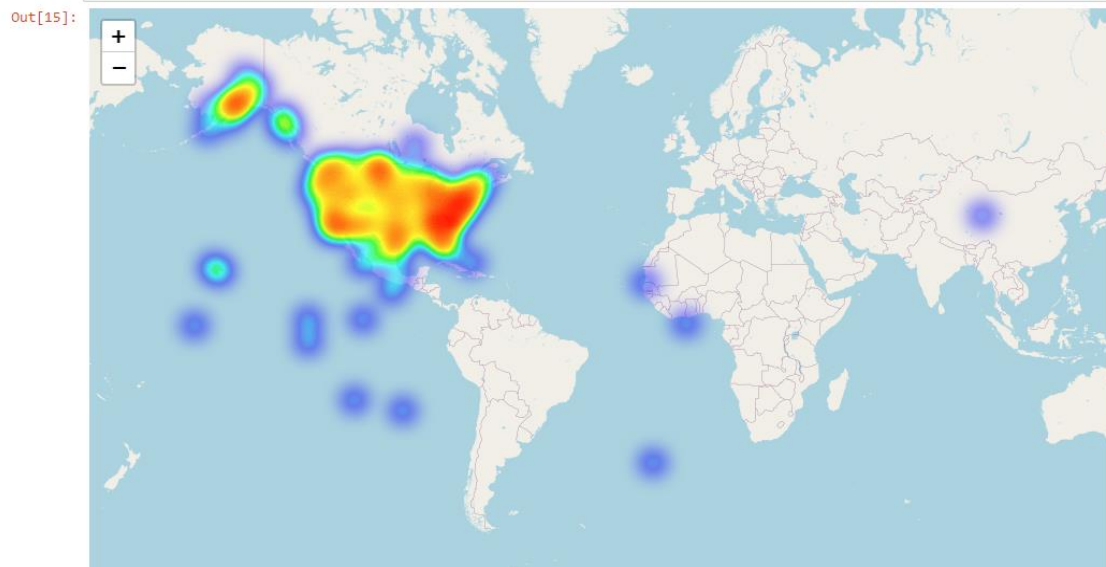
Έπειτα, με την βοήθεια ενός HeatMap, όπως φαίνεται και παρακάτω, παρατηρήσαμε ότι μερικές συντεταγμένες δεν αποτυπώνονται στον χάρτη πάνω στις ΗΠΑ, αλλά σε διαφορετικές χώρες του κόσμου. Αυτό οφείλεται σε ορισμένες ακραίες τιμές, οι οποίες έχουν καταχωρηθεί στα δεδομένα μας λάθος και πρέπει να αφαιρεθούν.

Dropping Outliers

```
In [15]: import folium
          from folium import plugins
          import numpy as np
          import pandas as pd

          heatMap = folium.Map([41, -96], zoom_start=4)
          heatMapCleanLoc = df[np.isfinite(df['lat'])].sample(n=300000)

          heatArr = heatMapCleanLoc[['lat', 'long']].to_numpy()
          heatMap.add_child(plugins.HeatMap(heatArr, radius=15))
```



Ουσιαστικά, η διαδικασία που ακολουθούμε βασίζεται στην απόρριψη όσων τιμών δεν βρίσκονται στα προσδοκώμενα σύνορα, με βάση το γεωγραφικό πλάτος και μήκος των ΗΠΑ.

```
In [16]: # usa_lat_range_contiguous = (24.396308, 49.384358)
        # usa_long_range_contiguous = (-125.000000, -66.934570)

        # Additional ranges for Alaska and Hawaii
        usa_lat_range_alaska = (51.8819, 71.5388)
        usa_long_range_alaska = (-179.148909, -129.986651)

        usa_lat_range_hawaii = (18.7763, 20.9957)
        usa_long_range_hawaii = (-156.000000, -154.8067)

        # Filter DataFrame to keep only rows within the extended USA boundaries
        df_usa = df[((df['lat'] >= usa_lat_range_contiguous[0]) & (df['lat'] <= usa_lat_range_contiguous[1]) &
            ((df['long'] >= usa_long_range_contiguous[0]) & (df['long'] <= usa_long_range_contiguous[1])) |
            ((df['lat'] >= usa_lat_range_alaska[0]) & (df['lat'] <= usa_lat_range_alaska[1]) &
            ((df['long'] >= usa_long_range_alaska[0]) & (df['long'] <= usa_long_range_alaska[1])) |
            ((df['lat'] >= usa_lat_range_hawaii[0]) & (df['lat'] <= usa_lat_range_hawaii[1]) &
            (df['long'] >= usa_long_range_hawaii[0]) & (df['long'] <= usa_long_range_hawaii[1]))))

        # Now df_usa_all contains only rows with latitude and longitude values within the extended USA boundaries

        print(f"{df.shape[0] - df_usa.shape[0]} rows removed!")

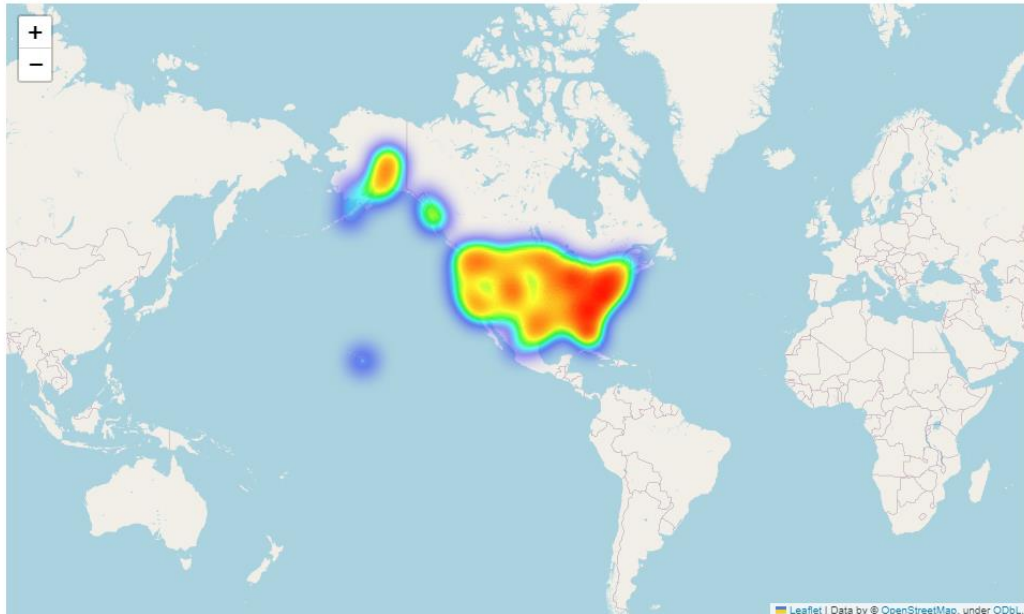
1971 rows removed!
```

Πράγματι, αφαιρέθηκαν όλες οι καταχωρήσεις των οποίων οι γεωγραφικές συντεταγμένες δεν ενθυλακώνονταν στις ΗΠΑ, όπως φαίνεται παρακάτω.

```
In [17]: # heatMap = folium.Map([41, -96], zoom_start=4)
        # heatMapCleanLoc = df_usa[np.isfinite(df_usa['lat'])].sample(n=300000)

        heatArr = heatMapCleanLoc[["lat", "long"]].to_numpy()
        heatMap.add_child(plugins.HeatMap(heatArr, radius=15))
```

Out[17]:



Τέλος, θα αφαιρέσουμε τις ακραίες τιμές από τις βασικές μας στήλες με αριθμητικές τιμές. Αυτές είναι τα δωμάτια(beds), τα μπάνια(baths), η τιμή(price) και τα τετραγωνικά πόδια(sqfeet). Για τον σκοπό αυτό δημιουργήσαμε τον παρακάτω κώδικα. Η τεχνική που ακολουθήσαμε για την αφαίρεση των ακραίων τιμών βασίζεται στο Interquartile Range(IQR) και ονομάζεται IQR rule ή IQR method. Αυτή η τεχνική χρησιμοποιεί τα προκαθορισμένα όρια $Q1 - 1.5 * IQR$ και $Q3 + 1.5 * IQR$, για να αποφασίσει ποιες τιμές θεωρούνται ακραίες.

```
In [22]: def filter_outliers(df, column):
    data = df[column]
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1

    # Set thresholds for outliers
    lower_bound = max(Q1 - (1.5 * IQR), 0)
    upper_bound = Q3 + (1.5 * IQR)

    print(f"{column} Lower Bound: {lower_bound}")
    print(f"{column} Upper Bound: {upper_bound}")

    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

    return df_filtered

# Specify the columns to filter
columns_to_filter = ['price', 'sqfeet', 'baths', 'beds']

# Delete the $1 Listings
df_usa = df_usa[df_usa["price"] > 1]

# Apply the filtering function for each column
df_new = df_usa.copy() # Create a copy to keep the original DataFrame intact

for column in columns_to_filter:
    df_new = filter_outliers(df_new, column)

# Print the resulting DataFrame
print("Shape of df_new:", df_new.shape)

price Lower Bound: 0
price Upper Bound: 2272.5
sqfeet Lower Bound: 187.5
sqfeet Upper Bound: 1687.5
baths Lower Bound: 0
baths Upper Bound: 3.5
beds Lower Bound: 0
beds Upper Bound: 3.5
Shape of df_new: (340828, 18)
```

```
In [23]: print(f"{df_usa.shape[0] - df_new.shape[0]} rows cleaned!")

38231 rows cleaned!
```

Τελικά, από το αρχικό σετ δεδομένων μας διαγράψαμε 42.000 γραμμές μέσω της εκθάρσης και της επεξεργασίας.

Final Results

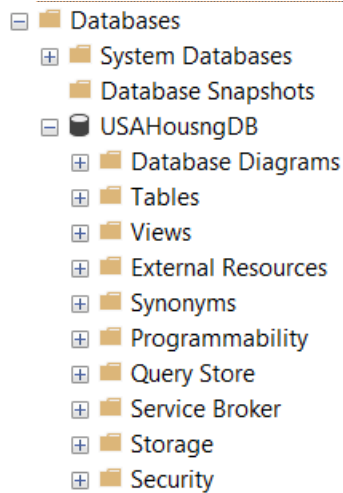
- So, after all this cleaning, we cleaned:

```
In [28]: print(f"{df.shape[0] - df_new.shape[0]} rows !!")

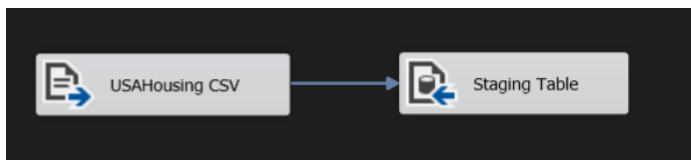
42231 rows !!
```

Data Warehousing

Ξεκινήσαμε με την δημιουργία της βάσης δεδομένων στο Microsoft SQL Server Management Studio.



Έπειτα δημιουργήσαμε ένα Data Flow Task όπου έχουμε ένα Excel Sources που εμπεριέχει το dataset και ένα SQL Server Destination με προορισμό την βάση μας. Έτσι δημιουργούμε τον πίνακα staging με τα στοιχεία των καταχωρήσεων των κατοικιών.



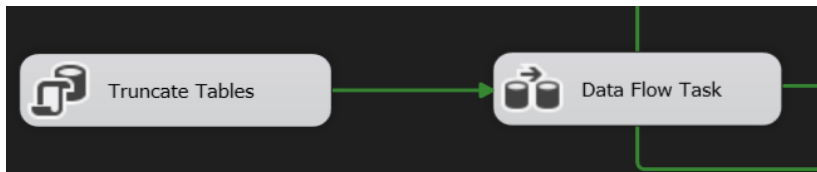
Specify a connection manager, data source, or data source view, and select the table or the view into which the data is copied. Click New to create a new table or view.

Connection manager:
LocalHost.USAHousngDB ▼ New...

Use a table or view:
[staging] ▼ New...

Preview

Επόμενο βήμα ήταν η δημιουργία ενός Execute SQL Task, το οποίο συνδέουμε με το παρακάτω Data Flow Task. Σκοπός είναι να πραγματοποιήσουμε truncate για τον πίνακά μας, τον οποίο δημιουργήσαμε παραπάνω, για να αποφεύγεται η επανειλημμένη εισαγωγή στοιχείων σε αυτόν, χωρίς την διαγραφή των προηγούμενων εγγραφών.




Τα SQL Queries που περιέχονται είναι τα παρακάτω.

General	
Name	Truncate Tables
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1253
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	LocalHost.USAHousngDB
SQLSourceType	Direct input
SQLStatement	TRUNCATE TABLE staging;TRUNCATE TAB
IsQueryStoredProcedure	
BypassPrepare	
<div>Enter SQL Query</div> <div>TRUNCATE TABLE staging; TRUNCATE TABLE listings_fact;</div>	
SQLStatement	

Στη συνέχεια, ασχοληθήκαμε με τα dimensions. Ξεχωρίσαμε τα εξής dimensions:

1. Location dimension

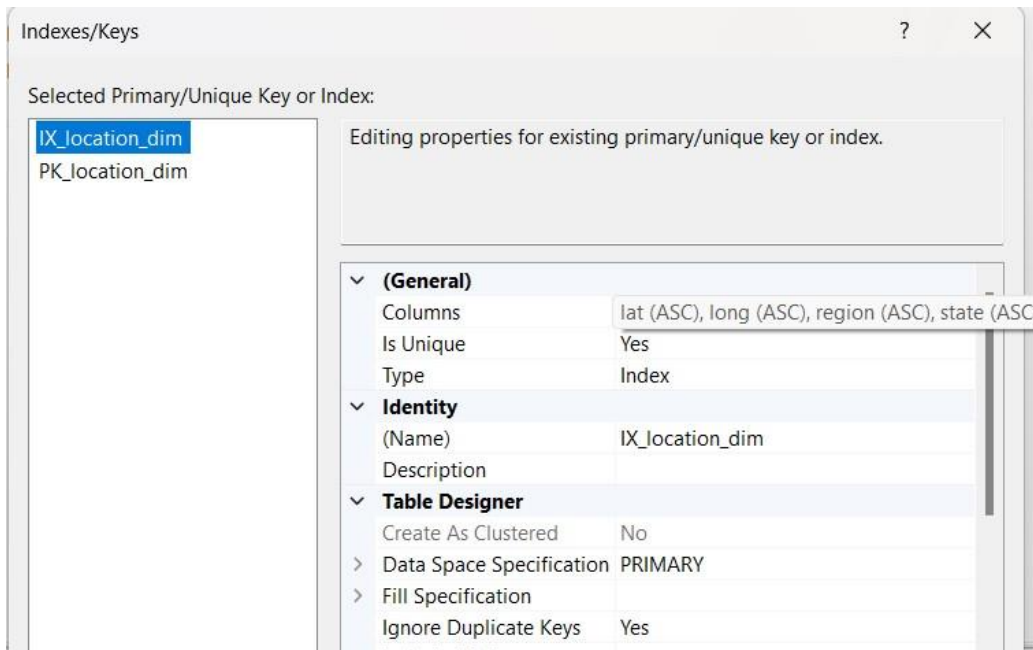
Το συγκεκριμένο dimension περιγράφει την τοποθεσία κάθε καταχώρησης και χαρακτηρίζεται από την περιοχή(region), την πολιτεία(state), το γεωγραφικό μήκος(long) και πλάτος(lat).

	Column Name	Data Type	Allow Nulls
	location_id	int	<input type="checkbox"/>
	region	varchar(64)	<input type="checkbox"/>
	state	varchar(64)	<input type="checkbox"/>
	lat	float	<input type="checkbox"/>
	long	float	<input type="checkbox"/>
			<input type="checkbox"/>

Ορίσαμε ως primary key την στήλη location_id, η οποία δεν πρέπει να περιέχει null τιμές, ενώ είναι τύπου integer και αυξάνεται κατά μία μονάδα κάθε φορά.

▼ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Τέλος, ορίσαμε index, δηλώνοντας ότι δέχεται μοναδικές τιμές και πρέπει να παραβλέπει διπλότυπες τιμές, εφόσον με βάση αυτή την στήλη θα παίρνει τιμές η στήλη location_id.



2. Property dimension

Το property dimension περιγράφει τα στοιχεία της κάθε ιδιοκτησίας που έχει καταχωρηθεί και χαρακτηρίζεται από τον τύπο της κατοικίας(type), τον αριθμό των κρεβατιών(beds) και των μπάνιων(baths).

	Column Name	Data Type	Allow Nulls
🔑	property_id	int	<input type="checkbox"/>
	type	varchar(64)	<input type="checkbox"/>
	beds	int	<input type="checkbox"/>
	baths	float	<input type="checkbox"/>
			<input type="checkbox"/>

Ορίσαμε ως primary key την στήλη property_id, η οποία δεν πρέπει να δέχεται null τιμές, ενώ είναι integer και αυξάνεται κατά 1 κάθε φορά.

▼ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Τέλος, πάλι ορίσαμε index, δηλώνοντας ότι δέχεται μοναδικές τιμές και πρέπει να παραβλέπει διπλότυπες τιμές, εφόσον με βάση αυτή την στήλη θα παίρνει τιμές η στήλη property_id.

Indexes/Keys

Selected Primary/Unique Key or Index:

IX_property_dim

PK_property_dim

Editing properties for existing primary/unique key or index.

▼ (General)

Columnsbaths (ASC), beds (ASC), type (ASC)

Is UniqueYes

TypeIndex

▼ Identity

(Name)IX_property_dim

Description

▼ Table Designer

Create As ClusteredNo

> Data Space SpecificationPRIMARY

> Fill Specification

Ignore Duplicate KeysYes

3. Amenities dimension

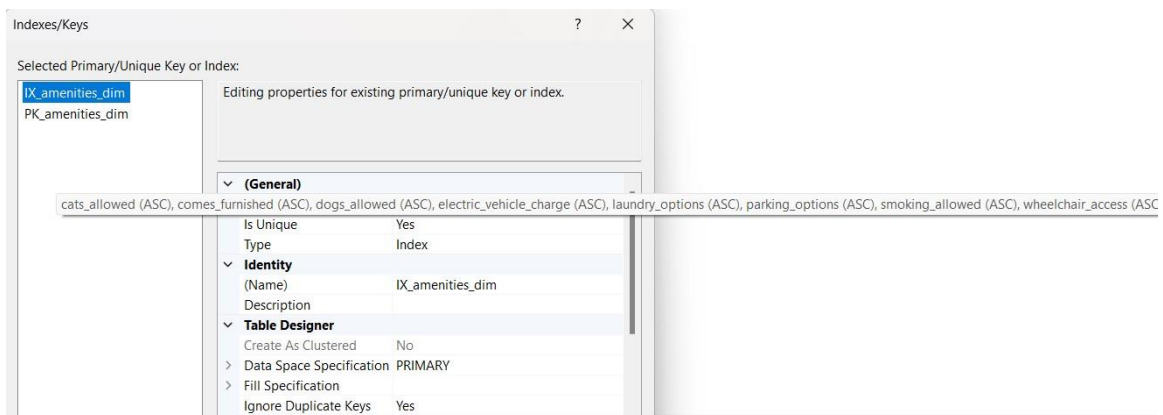
Το τελευταίο μας dimension, αυτό των amenities, περιγράφει τις παροχές της κάθε κατοικίας και χαρακτηρίζεται από τις στήλες της παρακάτω εικόνας.

	Column Name	Data Type	Allow Nulls
▶	amenities_id	int	<input type="checkbox"/>
	cats_allowed	bit	<input type="checkbox"/>
	dogs_allowed	bit	<input type="checkbox"/>
	smoking_allowed	bit	<input type="checkbox"/>
	wheelchair_access	bit	<input type="checkbox"/>
	electric_vehicle_charge	bit	<input type="checkbox"/>
	comes_furnished	bit	<input type="checkbox"/>
	laundry_options	varchar(64)	<input type="checkbox"/>
	parking_options	varchar(64)	<input type="checkbox"/>
			<input type="checkbox"/>

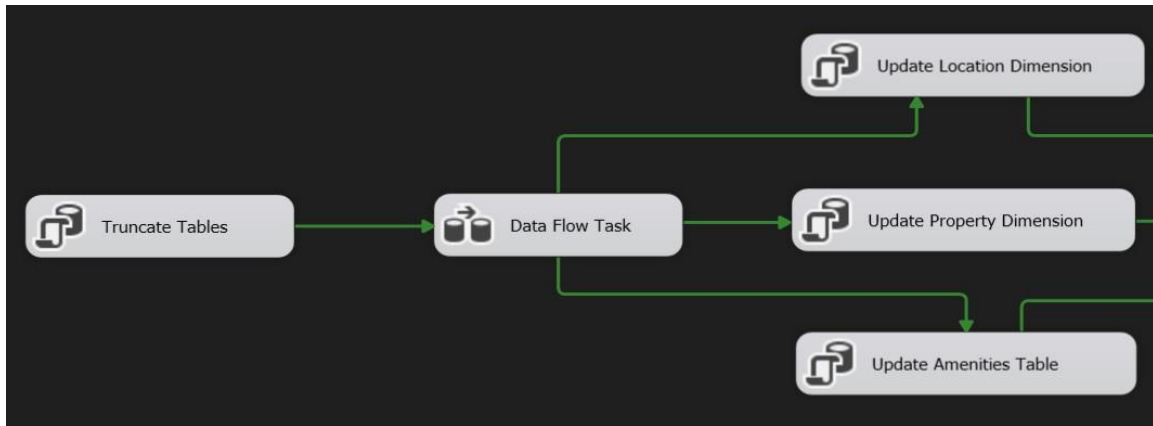
Για το τρίτο dimension μας, λοιπόν, ορίσαμε ως primary key την στήλη amenities_id, η οποία δεν πρέπει να δέχεται null τιμές, ενώ είναι integer και αυξάνεται κατά 1 κάθε φορά.

▼ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Τέλος, πάλι ορίσαμε index, δηλώνοντας ότι δέχεται μοναδικές τιμές και πρέπει να παραβλέπει διπλότυπες τιμές, εφόσον με βάση αυτή την στήλη θα παίρνει τιμές η στήλη amenities_id.



Εν συνεχεία δημιουργήσαμε τρία Execute SQL Task, ένα για κάθε dimension, τα οποία και συνδέσαμε με το Data Flow Task που δημιουργήσαμε παραπάνω.



Σκοπός αυτών των tasks είναι η ενημέρωση των πινάκων και την εισαγωγή των δεδομένων σε αυτούς, οι οποίοι αφορούν τα dimensions.

1. Update Location dimension

General	
Name	Update Location Dimension
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1253
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	LocalHost.USAHousngDB
SQLSourceType	Direct input
SQLStatement	INSERT INTO location_dim (region, state, lat, long)
IsQueryStoredProcedure	
BypassPrepare	SELECT DISTINCT region, state, lat, long
FROM staging	
WHERE NOT EXISTS (SELECT *	
FROM location_dim	
WHERE region= region	
AND state= state	
AND lat=lat	
AND long=long);	
<div> <div> Name </div> <div> Specifies the name of the task. </div> </div> <div> <div>Browse...</div> <div>Build Query</div> </div>	

Ο κώδικας στο δεξιά κάτω πλαίσιο εισάγει τις ανάλογες τιμές του πίνακα του Location dimension(location_dim).

2. Property dimension

General	
Name	Update Property Dimension
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1253
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	LocalHost.USAHousngDB
SQLSourceType	Direct input
SQLStatement	INSERT INTO property_dim (type, beds, baths)
IsQueryStoredProcedure	
BypassPrepare	SELECT DISTINCT type, beds, baths
	FROM staging
Name	
Specifies the name of the task.	WHERE NOT EXISTS (SELECT *
	FROM property_dim
	WHERE type= type
	AND beds= beds
	AND baths=baths)
Browse...	Build Qu

Εισάγουμε τις τιμές του Property dimension στον αντίστοιχο πίνακα property_dim.

3. Amenities dimension

Τελικά, εισάγουμε και τα αντίστοιχα δεδομένα που χρειάζεται στον πίνακα amenities_dim.

General	
Name	Update Amenities Table
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1253
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	LocalHost.USAHousngDB
SQLSourceType	Direct input

Enter SQL Query

```

INSERT INTO amenities_dim (cats_allowed, dogs_allowed, smoking_allowed,
wheelchair_access, electric_vehicle_charge, comes_furnished, laundry_options,
parking_options)

SELECT DISTINCT cats_allowed, dogs_allowed, smoking_allowed, wheelchair_access,
electric_vehicle_charge, comes_furnished, laundry_options, parking_options

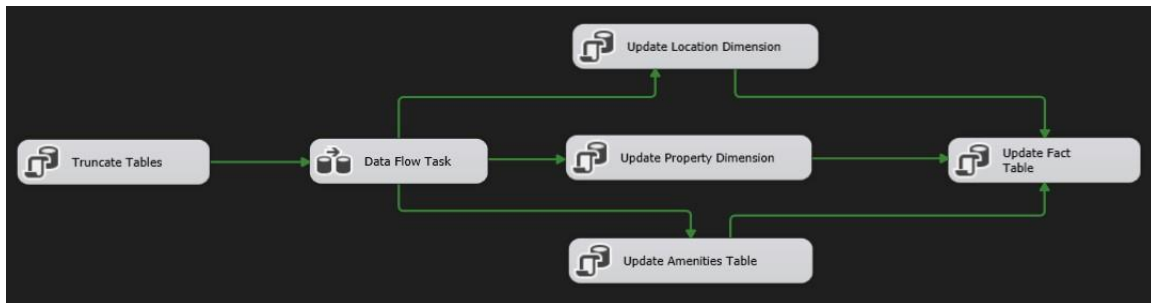
FROM staging

WHERE NOT EXISTS (SELECT *

FROM amenities_dim
|
WHERE cats_allowed= cats_allowed
AND dogs_allowed= dogs_allowed
AND wheelchair_access = wheelchair_access
AND electric_vehicle_charge = electric_vehicle_charge
AND comes_furnished = comes_furnished
AND laundry_options = laundry_options
AND parking_options= parking_options);

```

Αφού εισάγουμε τα δεδομένα μας τους πίνακες των dimensions, το τελευταίο στάδιο είναι η δημιουργία του Fact table. Έτσι, δημιουργούμε ένα ακόμα Execute SQL Task, το οποίο συνδέεται με καθένα από τα Execute SQL Task που αφορούν τα dimensions.



Με το task αυτό δημιουργούμε το Fact table, χρησιμοποιώντας τον παρακάτω κώδικα και ολοκληρώνοντας την διαδικασία του Warehouse της εργασίας.

General	
Name	Update Fact Table
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1253
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	LocalHost.USAHousngDB

Enter SQL Query

```

INSERT INTO listings_fact (listing_id, location, property, amenities, sqfeet, price)
SELECT
    staging.id,
    location_dim.location_id AS location,
    property_dim.property_id AS property,
    amenities_dim.amenities_id AS amenities,
    staging.sqfeet,
    staging.price
FROM
    staging
INNER JOIN
    location_dim ON staging.region = location_dim.region AND staging.state = location_dim.state AND staging.lat =
location_dim.lat AND staging.long = location_dim.long
INNER JOIN
    property_dim ON staging.type = property_dim.type AND staging.beds = property_dim.beds AND staging.baths =
property_dim.baths
INNER JOIN
    amenities_dim ON staging.cats_allowed = amenities_dim.cats_allowed AND staging.dogs_allowed =
amenities_dim.dogs_allowed AND staging.smoking_allowed = amenities_dim.smoking_allowed AND
staging.wheelchair_access = amenities_dim.wheelchair_access AND staging.electric_vehicle_charge =
amenities_dim.electric_vehicle_charge AND staging.comes_furnished = amenities_dim.comes_furnished AND
staging.laundry_options = amenities_dim.laundry_options AND staging.parking_options =
amenities_dim.parking_options;

```


Cube

Για να δημιουργήσουμε τον κύβο των δεδομένων μας στο Visual Studio θα δημιουργήσουμε ένα Analysis Services Multidimensional Project.

Στη συνέχεια, πατάμε στο data source και ανοίγει το παρακάτω Data Source Wizard και, έτσι, ξεκινάμε την υλοποίηση του κύβου.

Select how to define the connection
You can select from a number of ways in which your data source will define its

☐ Create a data source based on another object

☒ Create a data source based on an existing or new connection

Data connections:

LocalHost.USAHousngDB

Data connection properties:

Property	Value
Data Source	.
Initial Catalog	USAHousngDB
Integrated Se...	SSPI
Provider	SQLOLEDB.1

New... Delete

< Back Next > Finish >>| Cancel

Στην πρώτη σελίδα επιλέγουμε το table που περιέχει τα measures, δηλαδή το Fact table.

Cube Wizard

Select Measure Group Tables

Select a data source view or diagram and then select the tables that will be used for measure groups.

Data source view:
USA Housng DB

Measure group tables:

☒ listings_fact
☐ amenities_dim
☐ location_dim
☐ property_dim

[Suggest](#)

[< Back](#) [Next >](#) [Finish >>|](#) [Cancel](#)

Moves to the next wizard page

Στην συνέχεια επιλέγουμε τα measures που θέλουμε να μετράει ο κύβος μας, δηλαδή τα Sqfeet, το Price και το Listings Fact Count.

Cube Wizard

Select Measures

Select measures that you want to include in the cube.

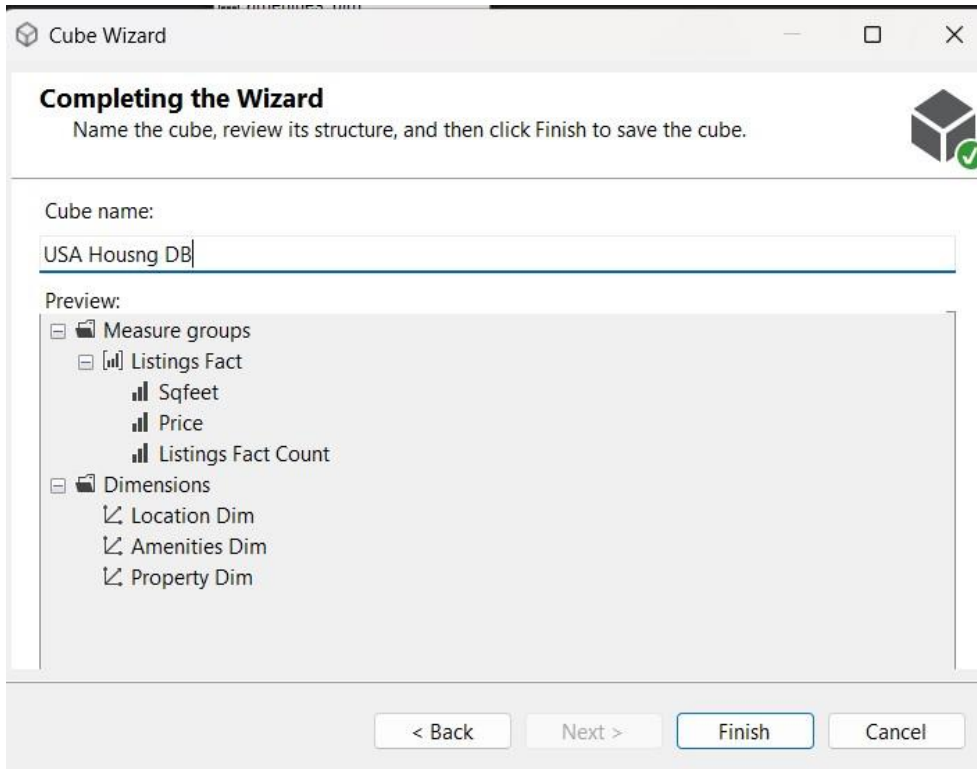
☒ Measure

☒ Listings Fact

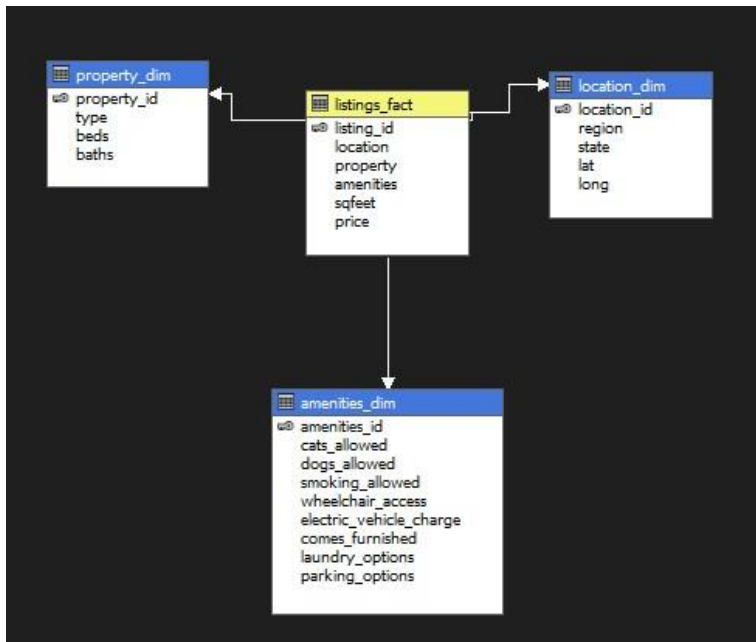
☒ Sqfeet
☒ Price
☒ Listings Fact Count

[< Back](#) [Next >](#) [Finish >>|](#) [Cancel](#)

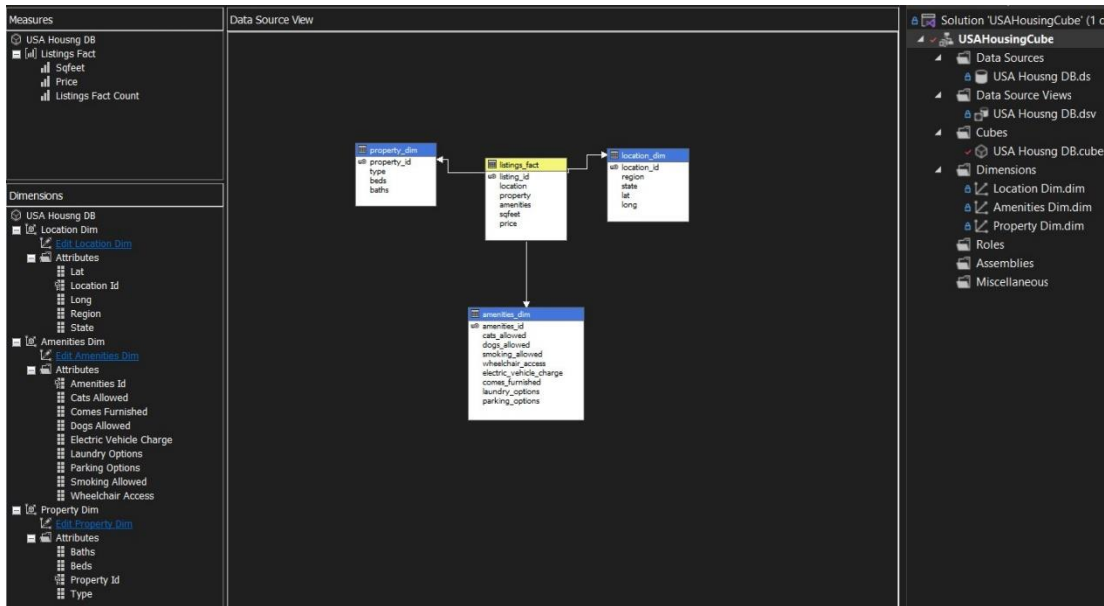
Μετά επιλέγουμε τα dimension table μας και, τελικά, αποθηκεύουμε τον κύβο μας και κλείνουμε τον Wizard.



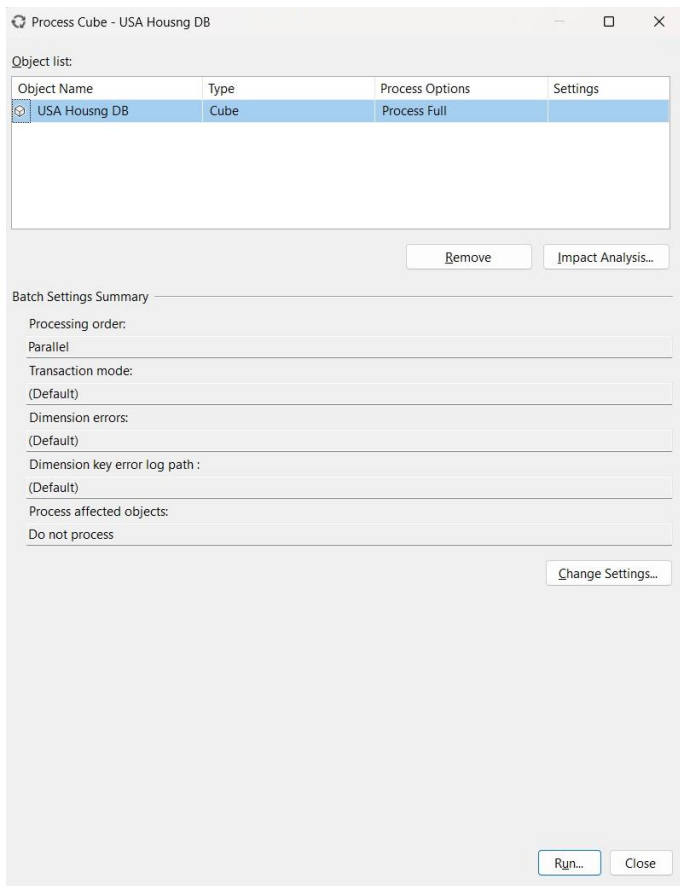
Επιστρέφοντας στο cube structure παρατηρούμε ότι το Visual Studio δημιούργησε το star schema μας.



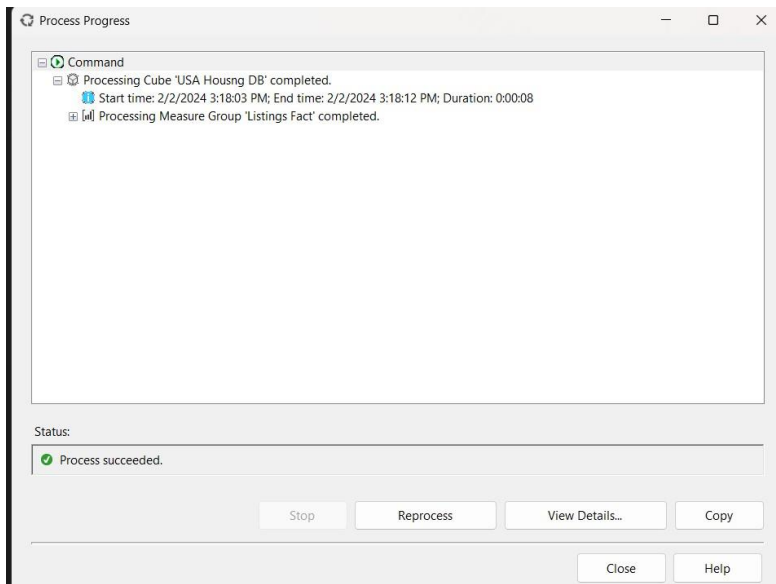
Πριν κάνουμε process τον κύβο πρέπει να προσθέσουμε στα dimensions μας τα υπόλοιπα attributes εκτός από το id του dimension, για να έχουμε πρόσβαση σε αυτά στη συνέχεια στον browser. Αυτό κάνουμε παρακάτω για το location dimension και αντίστοιχα για τα υπόλοιπα.



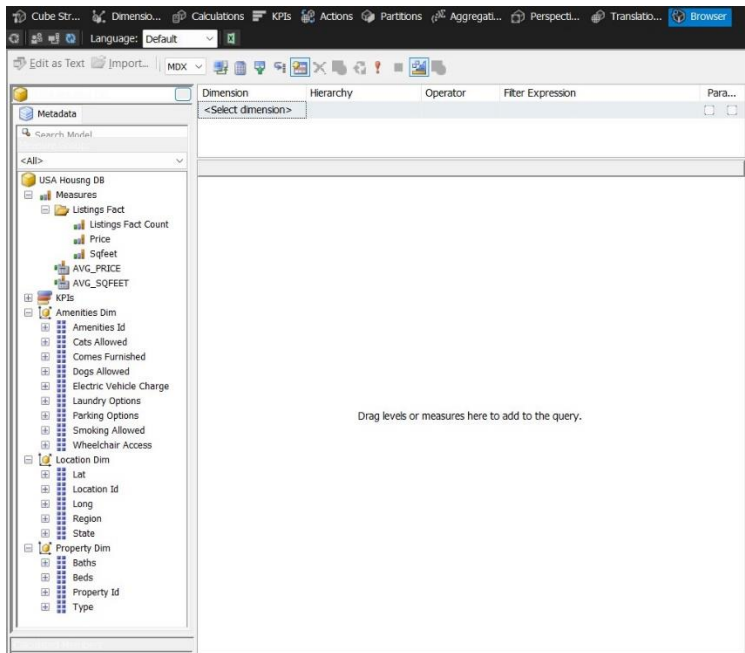
Και τώρα είμαστε έτοιμοι να κάνουμε process τον κύβο πατώντας το κουμπί “process”, το οποίο ανοίγει το αντίστοιχο dialog, μετά από επιτυχές deployment.



Πατάμε run για να τρέξει ο κύβος μας.



Ο κύβος δημιουργήθηκε με επιτυχία. Πηγαίνοντας στο Browser μπορούμε να δούμε όλες τις επιλογές που υπάρχουν ώστε να κάνουμε queries και aggregations.

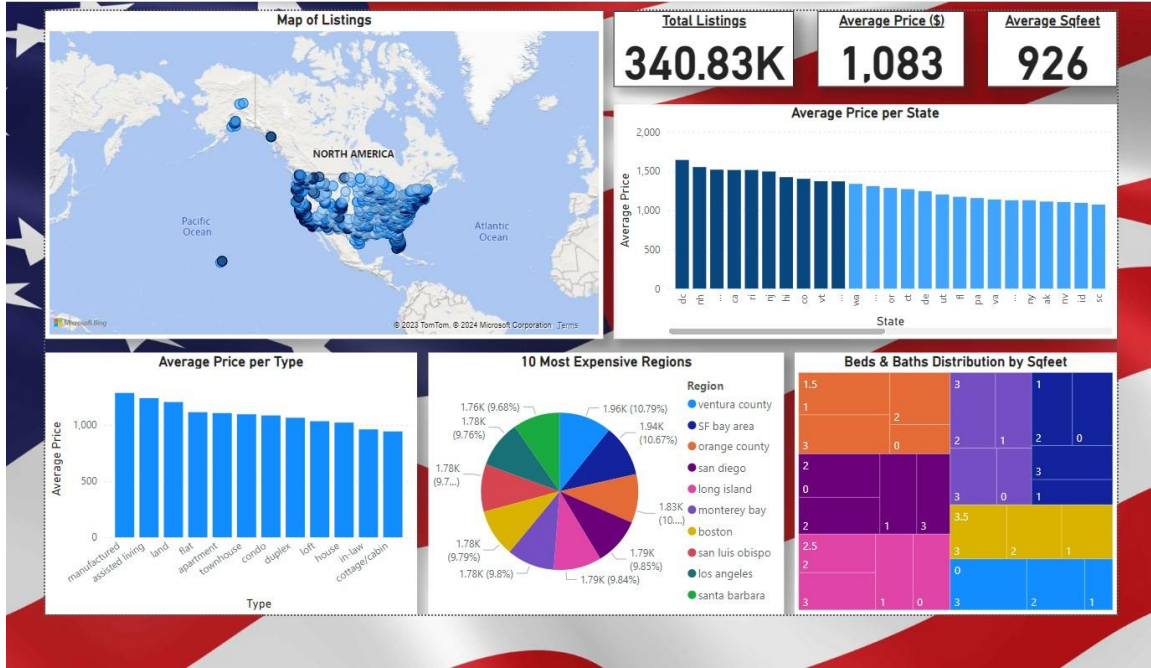


Τοποθετώντας ορισμένα attributes, μπορούμε να εφαρμόσουμε aggregations και ένα τέτοιο παράδειγμα είναι το παρακάτω, με τις μέσες τιμές (avg_price) και τα μέσα τετραγωνικά πόδια (avg_sqfeet) ανά περιοχή.

State	Price	AVG_PRICE	Sqfeet	AVG_SQFEET
ak	216...	1111.4490...	1642...	841.837519...
al	676...	872.43550...	7815...	1008.25141...
ar	226...	836.13794...	2623...	970.068786...
az	637...	1020.5967...	5121...	819.231003...
ca	362...	1514.4155...	2064...	861.865851...
co	139...	1401.5824...	8727...	879.348513...
ct	422...	1269.8453...	3044...	913.964875...
dc	319...	1642.2153...	1644...	845.275950...
de	236...	1243.7130...	1662...	873.622700...
fl	336...	1172.2022...	2823...	982.092503...
ga	122...	934.99778...	1268...	966.867251...
hi	118...	1423.0481...	73674	887.638554...
ia	642...	912.82493...	6487...	921.127218...
id	415...	1095.4069...	3689...	973.811823...
il	734...	893.20617...	7463...	907.335399...
in	506...	843.85759...	5633...	938.300466...
ks	532...	720.79366...	6334...	857.600866...
ky	435...	846.24237...	4665...	906.737414...
la	622...	923.26619...	6238...	924.656291...
ma	546...	1518.3551...	3266...	908.265294...
md	952...	1369.5222...	6367...	915.356382...
me	448...	1308.6122...	329687	961.186588...
mi	131...	984.99208...	1266...	945.431300...
mn	751...	1127.9461...	6288...	944.151929...
mo	139...	731.24267...	1784...	933.303870...
ms	368...	794.48759...	4601...	992.693851...
mt	117...	1060.5221...	1074...	970.945799...
nc	170...	973.49525...	1694...	969.624270...

Data Visualization

Μετά την δημιουργία του κύβου χρησιμοποιήθηκε το PowerBI για την δημιουργία dashboard και visualizations. Αφού, λοιπόν, εισήγαμε τα δεδομένα μας από τον κύβο μας στο PowerBI, δημιουργήσαμε την παρακάτω οπτική απεικόνιση.



Ας περιγράψουμε τα visualizations που επιλέξαμε:

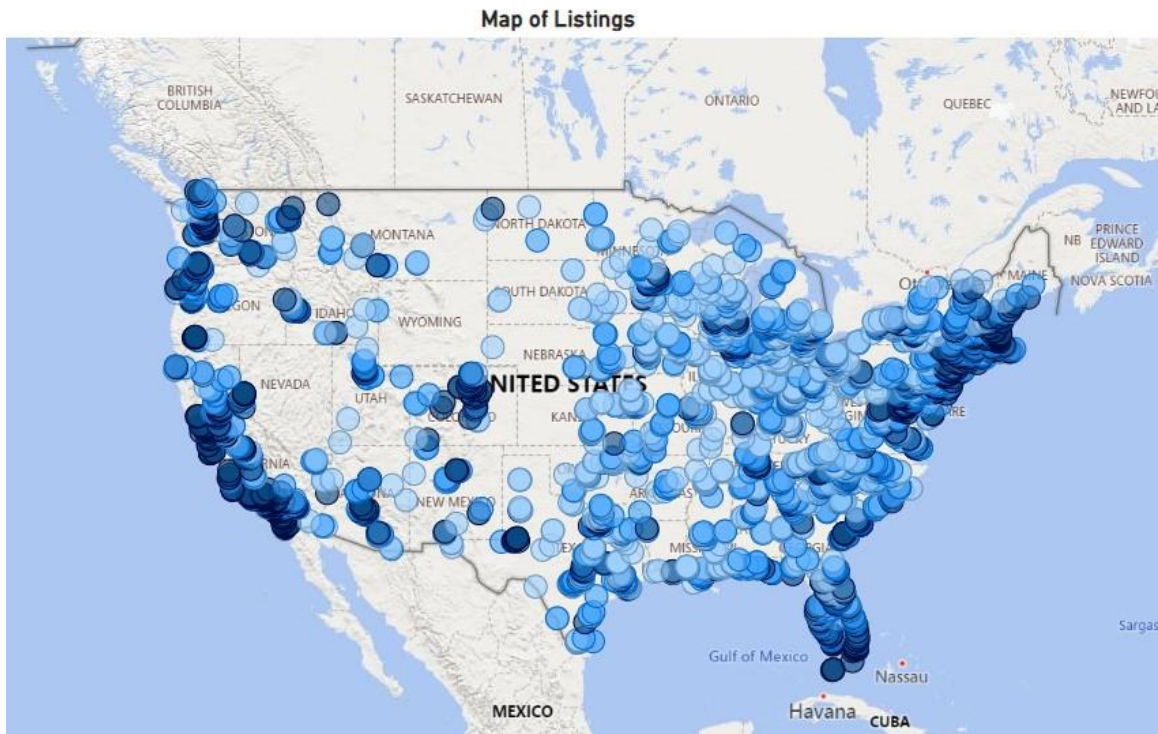
1. Average Price per State



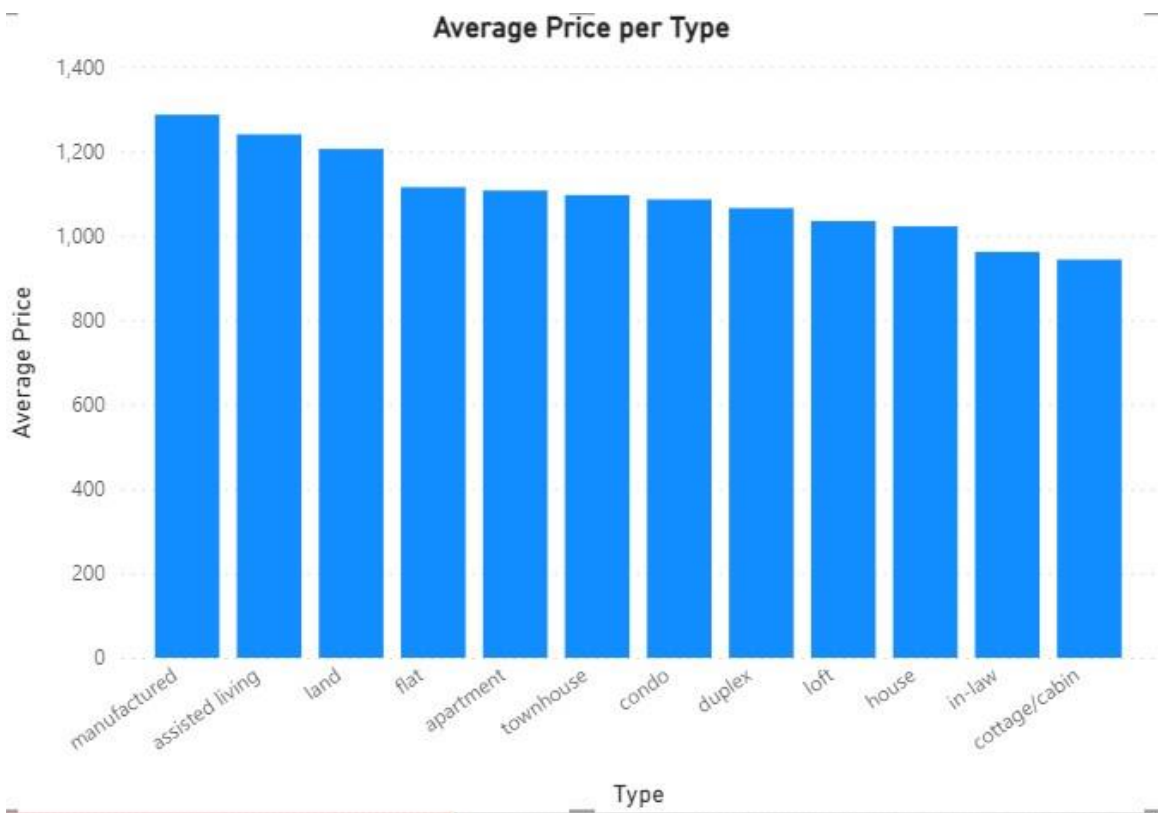
Το συγκεκριμένο διάγραμμα μας δείχνει την μέση τιμή κάθε πολιτείας, σε φθίνουσα σειρά.

2. Map of listings

Η παρακάτω εικόνα μας δείχνει τις καταχωρήσεις κατοικιών(listings) σε όλο τον χάρτη της Αμερικής με βάση τις γεωγραφικές συντεταγμένες(γεωγραφικό μήκος και πλάτος) κάθε καταχώρησης. Όσο πιο έντονα χρωματισμένη είναι μια κουκίδα, τόσο περισσότερες κατοικίες συγκεντρώνονται κοντά σε εκείνη την περιοχή.

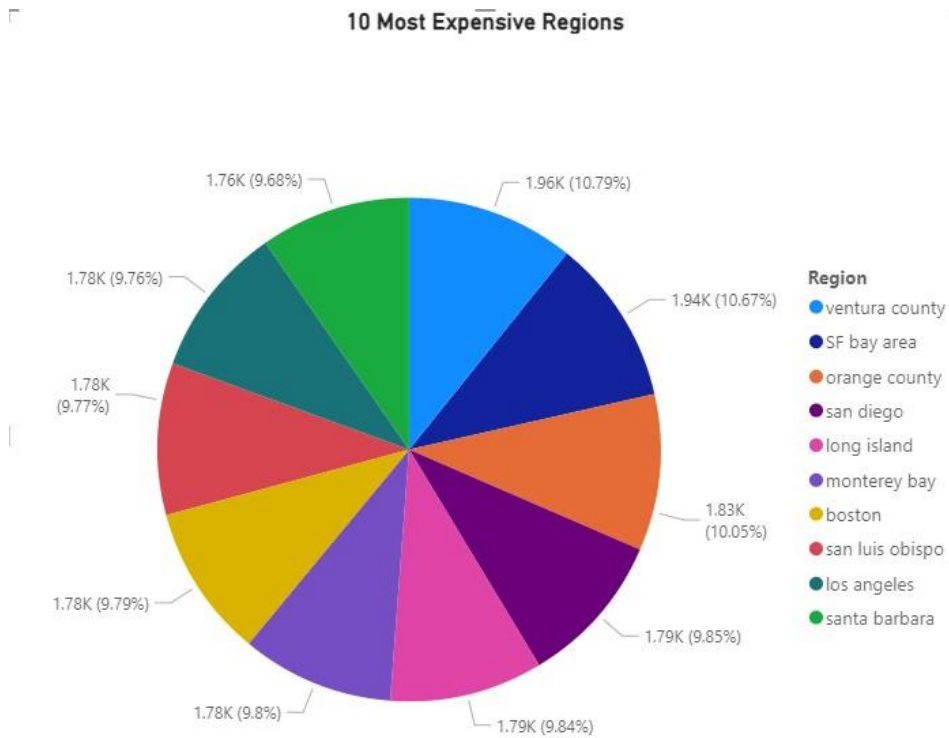


3. Average Price per Type



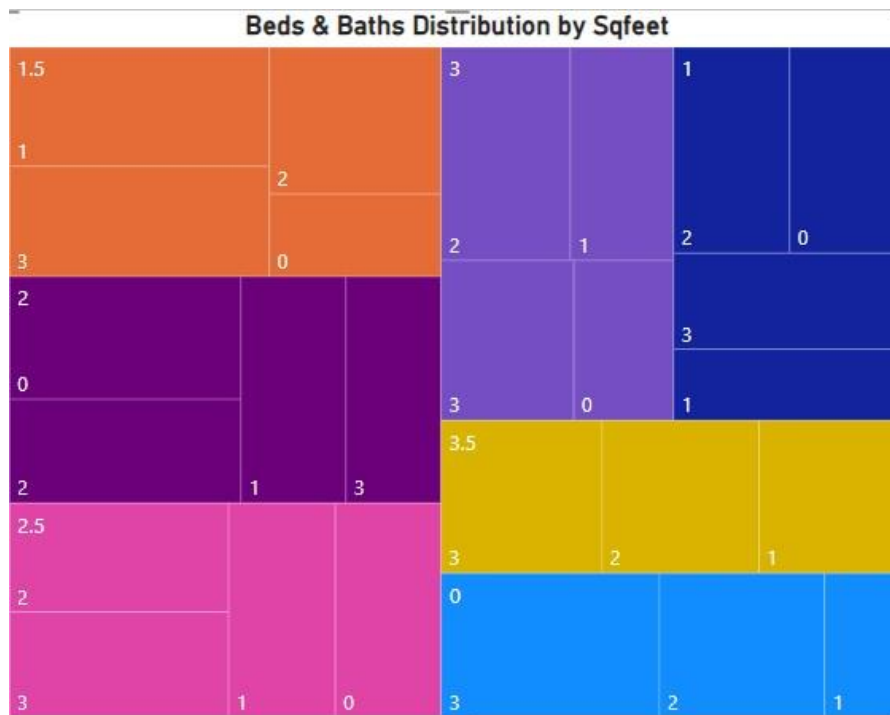
Ανάλογα με τον τύπο της κατοικίας που έχει κατοχυρωθεί υπολογίζουμε την μέση τιμή για τον συγκεκριμένο τύπο.

4. 10 Most Expensive Regions



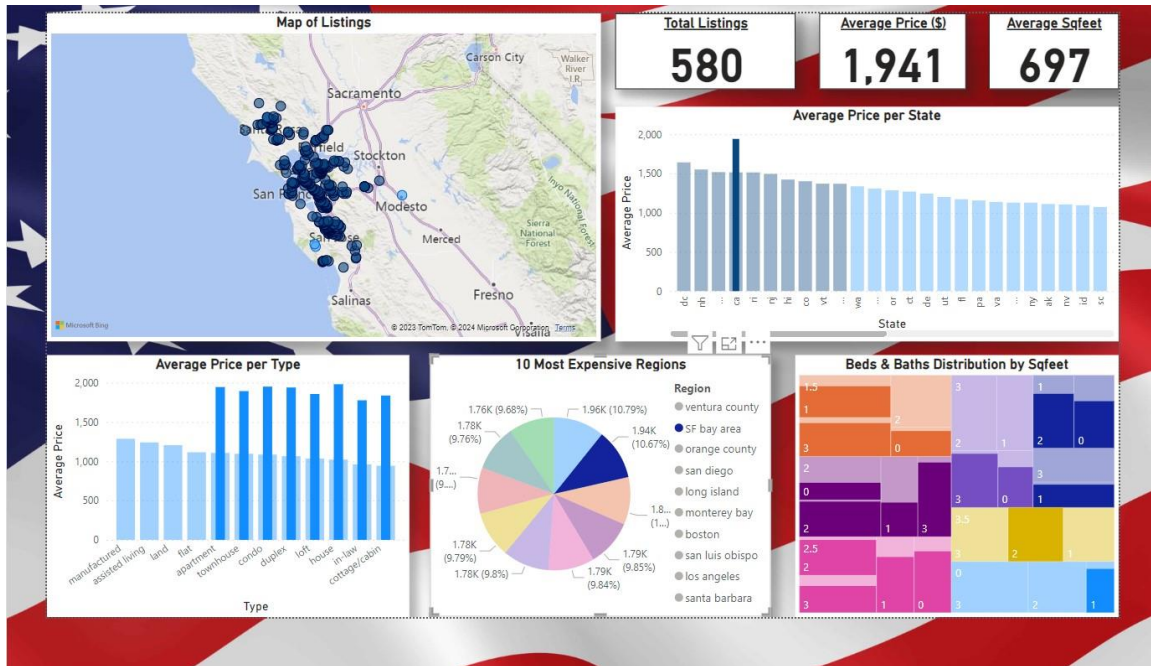
Το συγκεκριμένο pie chart μας δείχνει τις πιο ακριβές περιοχές, με την αντίστοιχη μέση τιμή και το όνομα της κάθε περιοχής στην στήλη στα δεξιά.

5. Beds and Baths Distribution by Sqfeet



Η συγκεκριμένη οπτικοποίηση μας περιγράφει την κατανομή των κρεβατιών και των μπάνιων μιας καταχώρησης ανάλογα με τα τετραγωνικά πόδια.

Οι οπτικοποιήσεις μας είναι όλες διαδραστικές μέσω του PowerBI. Για παράδειγμα, αν κάνουμε κλικ στην πολιτεία της California, τα διαγράμμάτα μας θα αλλάξουν όπως παρακάτω:



Παρατηρούμε, αρχικά, ότι ο χάρτης εστίασε στην πολιτεία της California, ενώ, ο τύπος των κατοικιών είναι μεγαλύτερος από την μέση τιμή σε κάποιες κατηγορίες. Παράλληλα, μια από τις 10 πιο ακριβές περιοχές βρίσκεται στην California και είναι το SF Bay Area, με μέση τιμή ενοικίασης τα 1.94 χιλιάδες δολάρια. Τέλος, οι χρωματισμοί στο διάγραμμα των κατανομών των δωματίων και των μπάνιων διαφοροποιούνται από την αρχική οπτικοποίηση.

Data Mining Tasks

Για την διαδικασία του Data Mining, αποφασίσαμε να εργαστούμε με τα open-source εργαλεία Jupyter Notebook και RapidMiner. Το Jupyter Notebook έχει χρησιμοποιηθεί και προηγουμένως στην παρούσα εργασία για τον καθαρισμό και την διαχείριση των δεδομένων, ενώ το RapidMiner είναι ένα επιπλέον open-source εργαλείο που χρησιμοποιείται για την ανάλυση δεδομένων και την εξόρυξη γνώσης. Οι λειτουργίες εξόρυξης δεδομένων που χρησιμοποιήθηκαν είναι η μέθοδος της παλινδρόμησης, η κατηγοριοποίηση, καθώς και η συσταδοποίηση.

1. Μέθοδοι Παλινδρόμησης

Για την πρώτη λειτουργία εξόρυξης, αποφασίσαμε να εστιάσουμε σε διάφορες μορφές παλινδρόμησης, δημιουργώντας προβλεπόμενες τιμές, και στις συσχετίσεις μεταξύ της τιμής μιας καταχώρησης και των παροχών της. Ας εξηγήσουμε, όμως, τον τρόπο με τον οποίο εργαστήκαμε.

Αρχικά, φορτώσαμε και διαβάσαμε σε ένα Jupyter Notebook αρχείο τα καθαρισμένα και επεξεργασμένα δεδομένα μας.

```
In [1]: import pandas as pd
df = pd.read_csv('housing_cleaned.csv')
df
```

Out[1]:

	id	region	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_access	electric_vehicle_charg
0	7049044568	reno / tahoe	1148	apartment	1078	3	2.0	1	1	0	0	
1	7049047186	reno / tahoe	1200	condo	1001	2	2.0	0	0	0	0	
2	7043634882	reno / tahoe	1813	apartment	1683	2	2.0	1	1	1	0	
3	7049045324	reno / tahoe	1095	apartment	708	1	1.0	1	1	1	0	
4	7049043759	reno / tahoe	289	apartment	250	0	1.0	1	1	1	1	
...
340823	7049053337	reno / tahoe	1295	apartment	957	2	2.0	1	1	1	0	
340824	7049052968	reno / tahoe	1549	apartment	1034	2	2.0	1	1	0	0	
340825	7049050454	reno / tahoe	1249	apartment	840	2	1.0	1	1	1	0	
340826	7049050149	reno / tahoe	1429	apartment	976	2	2.0	1	1	1	0	
340827	7049050010	reno / tahoe	1295	apartment	957	2	2.0	1	1	1	0	

340828 rows x 13 columns

Ύστερα, προκειμένου να μπορέσουμε να εφαρμόσουμε τις τεχνικές παλινδρόμησης, θα χρειαστεί να προσαρμόσουμε τα δεδομένα μας καταλλήλως. Γί' αυτόν τον σκοπό, θα χρησιμοποιήσουμε τη βιβλιοθήκη scikit-learn(sklearn) για την επεξεργασία των δεδομένων. Συγκεκριμένα, χρησιμοποιείται ο LabelEncoder από το sklearn.preprocessing για τη μετατροπή των *κατηγορικών μεταβλητών* σε *αριθμητικές* τιμές. Επίσης, αποθηκεύουμε τα αρχικά states σε έναν πίνακα, μαζί με τις καινούργιες αριθμητικές τιμές, γιατί θα τα χρειαστούμε στην συνέχεια της ανάλυσής μας.

```
In [2]: from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
state_mapping = pd.DataFrame({
    "state": df["state"]
})
df["state"]=le.fit_transform(df["state"])
state_mapping["encoded_state"]=df["state"]

df["region"]=le.fit_transform(df["region"])
df["type"]=le.fit_transform(df["type"])
df["laundry_options"]=le.fit_transform(df["laundry_options"])
df["parking_options"]=le.fit_transform(df["parking_options"])
```

```
In [3]: state_mapping = state_mapping.sort_values(by="encoded_state", ascending=False)
state_mapping = state_mapping.drop_duplicates(subset="state", keep="first")
state_mapping
```

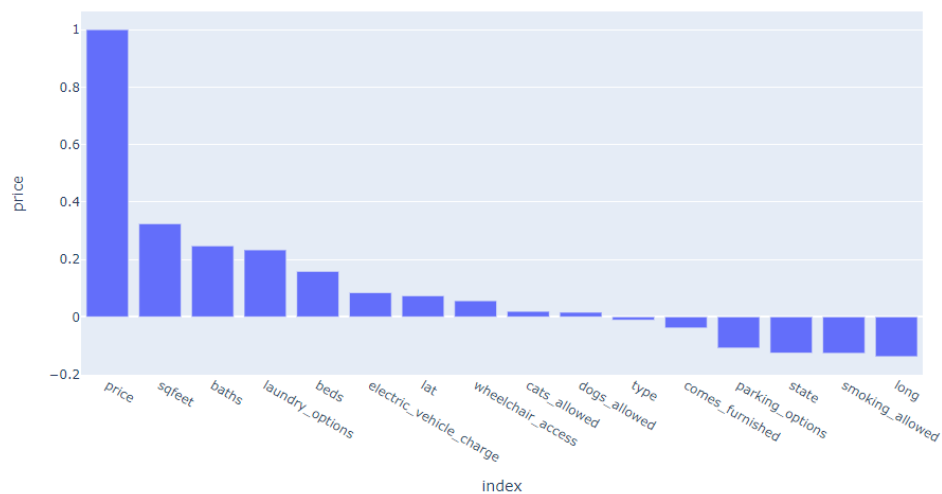
```
Out[3]:
```

	state	encoded_state
308226	wy	50
301967	wv	49
303948	wi	48
297519	wa	47
285150	vt	46
288529	va	45
281560	ut	44
260785	tx	43
243969	tn	42
239238	sd	41
232515	sc	40

```
In [4]: df.drop("region", axis=1, inplace=True)
df.drop("id", axis=1, inplace=True)
```

Στην συνέχεια, ο κώδικας χρησιμοποιεί τη βιβλιοθήκη Plotly Express για να δημιουργήσει ένα διάγραμμα με μπάρες που απεικονίζει τη συσχέτιση των χαρακτηριστικών με τη μεταβλητή "price", δηλαδή την τιμή κάθε καταχώρησης.

```
In [5]: import plotly.express as px
correlation=df.corr()["price"].reset_index().sort_values("price",ascending=False)
fig=px.bar(correlation, x="index", y="price")
fig.show()
```



Από το παραπάνω διάγραμμα, συγκεντρώνουμε πληροφορία για την συσχέτιση μεταξύ της τιμής μίας καταχώρησης και των λοιπών παροχών της. Προκειμένου να μην εστιάσουμε ξεχωριστά σε κάθε μία από

τις παροχές, μπορούμε να τις ομαδοποιήσουμε και να δημιουργήσουμε κατηγορίες ανάλογα με την ισχύ της συσχέτισης. Έτσι, έχουμε τις εξής κατηγορίες:

1. **Ισχυρή συσχέτιση:** Στην συγκεκριμένη κατηγορία εντάσσονται τα τετραγωνικά πόδια(sqfeet) του κτίσματος, τα baths και αν είναι διαθέσιμες οι επιλογές πλυντηρίου, όπου η τιμή της συσχέτισης είναι μεγαλύτερη από 0,2($correlation > 0.2$).
2. **Μέτρια συσχέτιση:** Σε αυτή την κατηγορία εντάσσεται ο αριθμός των υπνοδωματίων, το γεωγραφικό πλάτος, η παροχή φορτιστή για ηλεκτρικά αυτοκίνητα, η προσβασιμότητα για άτομα με αναπηρίες, καθώς και η άδεια κατοχής κατοικίδιων. Η τιμή της συσχέτισης είναι μεγαλύτερη του 0 και μικρότερη το 0,2($0 < correlation < 0.2$).
3. **Αρνητική συσχέτιση:** Στην συγκεκριμένη κατηγορία ενθυλακώνεται ο τύπος της καταχώρησης, η επίπλωση, το parking, η πολιτεία, η άδεια για κάπνισμα και το γεωγραφικό μήκος. Η τιμή της συσχέτισης φαίνεται να έχει τιμές μικρότερες του 0 και μεγαλύτερες του $-0,2$ ($-0.2 < correlation < 0$).

Προφανώς, η συσχέτιση δείχνει την σχέση που υπάρχει μεταξύ των εκάστοτε παροχών και της τιμής. Έτσι, το παραπάνω πόρισμα, είναι αρκετά ωφέλιμο για τους επενδυτές και τους κατόχους κατοικιών οι οποίοι επιθυμούν να δημιουργήσουν μια καταχώρηση και να νοικιάσουν τον χώρο τους. Γνωρίζοντας την συσχέτιση μεταξύ τιμών και παροχών, μπορούν να βελτιστοποιήσουν την ενοικίαση του διαμερίσματός τους, επιτρέποντας, για παράδειγμα, την κατοχή κατοικίδιων, ή εξοπλίζοντας τον χώρο με φορτιστή για ηλεκτρικά αμάξια(θετική συσχέτιση). Επίσης, μπορούν να εστιάσουν στην μετατροπή των παροχών οι οποίες μειώνουν την τιμή, όπως για παράδειγμα, να μην επιπλώνουν τον χώρο ή να μην επιτρέπουν το κάπνισμα(αρνητική συσχέτιση).

Τώρα, προχωρώντας με τις παλινδρομήσεις, ο παραπάνω κώδικας διαχωρίζει το σύνολο δεδομένων σε σύνολα εκπαίδευσης και ελέγχου, όπου οι μεταβλητές εξαρτώμενης και ανεξάρτητης μεταβλητής ορίζονται ως y και x αντίστοιχα. Στη συνέχεια, χρησιμοποιείται η συνάρτηση train_test_split από τη βιβλιοθήκη scikit-learn για να διαχωριστούν τα δεδομένα σε τυχαία εκπαιδευτικά και δοκιμαστικά σύνολα, όπου το 30% των δεδομένων προορίζεται για το δοκιμαστικό σύνολο, ενώ το random_state ορίζει μια σταθερά για την αναπαραγωγή των αποτελεσμάτων.

```
In [6]: > y=df["price"]
> x=df.drop("price", axis=1)

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 = 0)
```

Ο παρακάτω κώδικας εκπαιδεύει ένα μοντέλο γραμμικής παλινδρόμησης στα δεδομένα εκπαίδευσης και στη συνέχεια το χρησιμοποιεί για να προβλέψει τις τιμές στα δεδομένα ελέγχου, υπολογίζοντας το R^2 , MSE και RMSE για να αξιολογήσει την απόδοση του μοντέλου.

```
In [7]: > import numpy as np
> from sklearn import metrics
> from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train, y_train)
y_pred=lr.predict(X_test)
lr_r2=metrics.r2_score(y_test, y_pred)
lr_MSE=metrics.mean_squared_error(y_test, y_pred)
lr_RMSE=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

```
In [8]: > print("R^2 Score:", lr_r2)
> print("MSE Score:", lr_MSE)
> print("RMSE Score:", lr_RMSE)

R^2 Score: 0.23673349871138516
MSE Score: 110576.50940930205
RMSE Score: 332.5304638815849
```

Θα δημιουργήσουμε τον αντίστοιχο πίνακα με τις κανονικές τιμές και τις προβλέψεις που προκύπτουν από το μοντέλο της παλινδρόμησης, καθώς και ένα διάγραμμα που θα αναπαριστά τις τιμές αυτές.

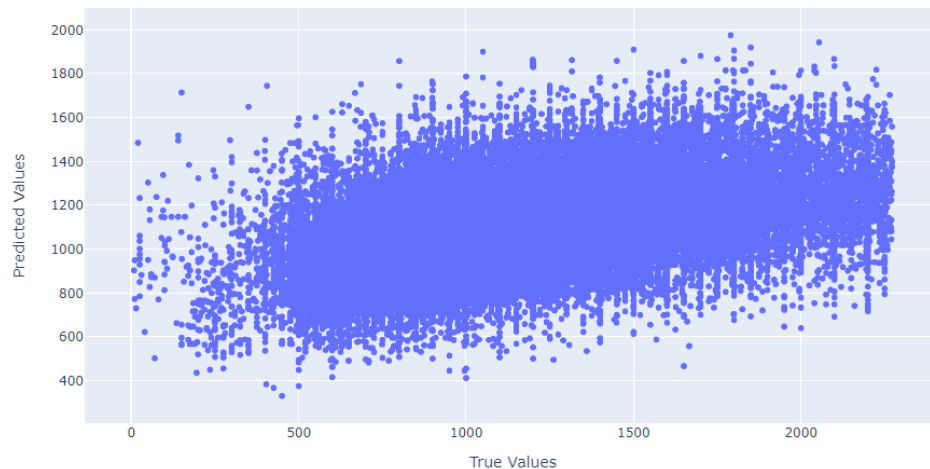
```
In [9]: pred_graph_lr=pd.DataFrame({"True Values":y_test, "Predicted Values":y_pred})
pred_graph_lr
```

Out[9]:

	True Values	Predicted Values
220842	1214	1058.745791
253104	785	1028.022949
100939	649	907.958657
252601	615	864.115558
326050	750	1404.716762
...
267496	650	634.704528
73309	675	1000.966410
2615	1468	1229.551097
39466	775	1084.283010
100110	575	920.486203

102249 rows x 2 columns

```
In [10]: import matplotlib.pyplot as plt
import seaborn as sns
fig=px.scatter(pred_graph_lr, x="True Values", y="Predicted Values")
fig.show()
```



Τα αποτελέσματα των μοντέλων παλινδρόμησης θα εξεταστούν συνολικά αφού εφαρμοστούν όλα. Έτσι, θα παραθέσουμε τις αντίστοιχες τεχνικές, όπως ακολουθούνται και παραπάνω, για κάθε μοντέλο. Στην συνέχεια, θα χρησιμοποιήσουμε το μοντέλο παλινδρόμησης XGBoost (Extreme Gradient Boosting) για να εκπαιδεύσουμε ένα μοντέλο πρόβλεψης. Το XGBoost είναι ένα δημοφιλές μοντέλο μηχανικής μάθησης που χρησιμοποιείται για προβλέψεις σε προβλήματα παλινδρόμησης και ταξινόμησης.

```
In [11]: from xgboost import XGBRegressor
xgb=XGBRegressor()
xgb.fit(X_train, y_train)
y_pred=xgb.predict(X_test)
xgb_r2=metrics.r2_score(y_test, y_pred)
xgb_MSE=metrics.mean_squared_error(y_test, y_pred)
xgb_RMSE=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

```
In [12]: print("R^2 Score:", xgb_r2)
print("MSE Score:", xgb_MSE)
print("RMSE Score:", xgb_RMSE)

R^2 Score: 0.813320905775275
MSE Score: 27044.711885311437
RMSE Score: 164.45276490625335
```

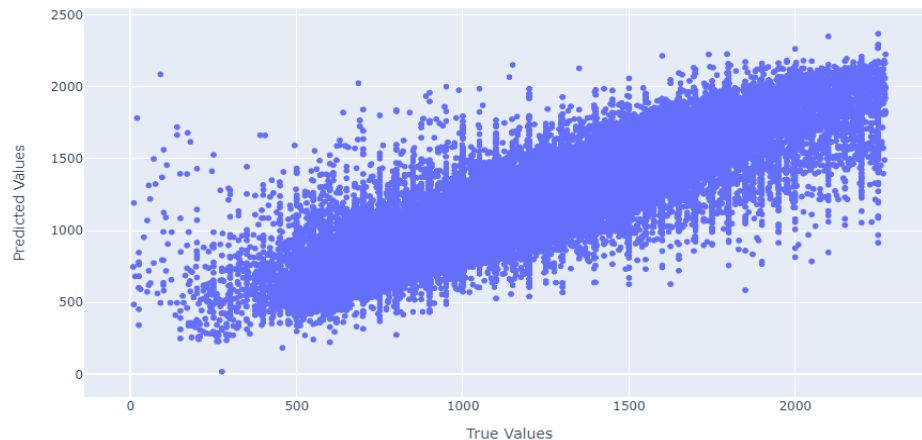
```
In [13]: pred_graph_xgb=pd.DataFrame({"True Values":y_test, "Predicted Values":y_pred})
pred_graph_xgb
```

Out[13]:

	True Values	Predicted Values
220842	1214	1248.773071
253104	785	820.112366
100939	649	534.584839
252601	615	615.280457
326050	750	1026.179688
...
267496	650	568.069824
73309	675	702.775330
2615	1468	1516.135498
39466	775	946.678101
100110	575	541.367432

102249 rows x 2 columns

```
In [14]: fig=px.scatter(pred_graph_xgb, x="True Values", y="Predicted Values")
fig.show()
```



Στον παρακάτω κώδικα χρησιμοποιείται το μοντέλο παλινδρόμησης Gradient Boosting, συγκεκριμένα η κλάση GradientBoostingRegressor από τη βιβλιοθήκη scikit-learn, για να εκπαιδευτεί ένα μοντέλο πρόβλεψης. Το Gradient Boosting είναι μια τεχνική αναβάθμισης δέντρων αποφάσεων, όπου τα δέντρα εκπαιδεύονται στη σειρά και κάθε δέντρο επιδιορθώνει τα σφάλματα του προηγούμενου.

```
In [15]: from sklearn.ensemble import GradientBoostingRegressor
gbr=GradientBoostingRegressor(n_estimators=100)
gbr.fit(X_train,y_train)
y_pred= gbr.predict(X_test)
gbr_r2=metrics.r2_score(y_test, y_pred)
gbr_MSE=metrics.mean_squared_error(y_test, y_pred)
gbr_RMSE=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```

```
In [16]: print("R^2 Score:", gbr_r2)
print("MSE Score:", gbr_MSE)
print("RMSE Score:", gbr_RMSE)

R^2 Score: 0.6197658507663555
MSE Score: 55085.56304971851
RMSE Score: 234.7031381335122
```

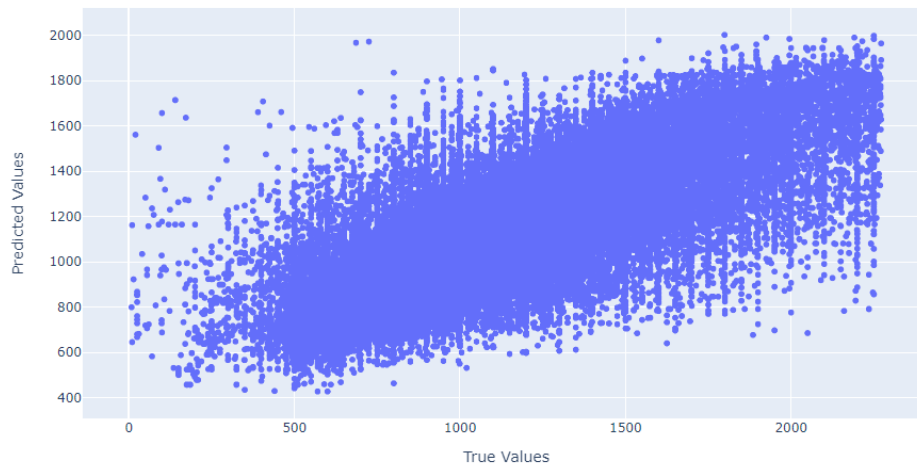
```
In [17]: pred_graph_gbr=pd.DataFrame({"True Values":y_test, "Predicted Values":y_pred})
pred_graph_gbr
```

Out[17]:

	True Values	Predicted Values
220842	1214	1269.551383
253104	785	956.307209
100939	649	658.312447
252601	615	652.203595
326050	750	1141.776516
...
267496	650	645.196224
73309	675	913.537253
2615	1468	1590.806601
39466	775	923.379267
100110	575	674.750231

102249 rows x 2 columns

```
In [18]: fig=px.scatter(pred_graph_gbr, x="True Values", y="Predicted Values")
fig.show()
```



Τέλος, ο παραπάνω κώδικας χρησιμοποιεί το μοντέλο παλινδρόμησης RandomForestRegressor από τη βιβλιοθήκη scikit-learn για να εκπαιδεύσει ένα μοντέλο πρόβλεψης. Το RandomForestRegressor χρησιμοποιεί ένα σύνολο τυχαίων δέντρων αποφάσεων και συνδυάζει τις προβλέψεις τους για να παράγει μια τελική πρόβλεψη.

```
In [19]: from sklearn.ensemble import RandomForestRegressor
ran=RandomForestRegressor()
ran.fit(X_train, y_train)
y_pred= ran.predict(X_test)
ran_r2=metrics.r2_score(y_test, y_pred)
ran_MSE=metrics.mean_squared_error(y_test, y_pred)
ran_RMSE=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
```



```
In [20]: print("R^2 Score:", ran_r2)
print("MSE Score:", ran_MSE)
print("RMSE Score:", ran_RMSE)
```

```
R^2 Score: 0.9022790204552237
MSE Score: 14157.106064364303
RMSE Score: 118.98363780101995
```

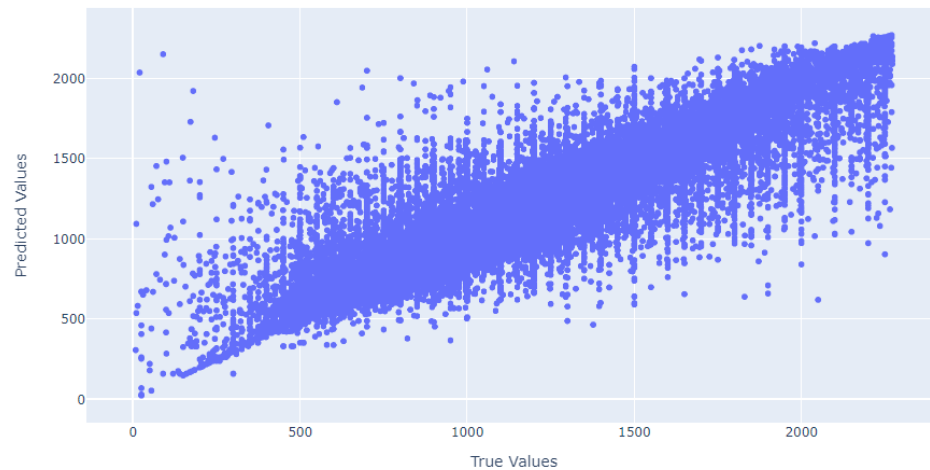
```
In [22]: pred_graph_ran = pd.DataFrame({
    "True Values": y_test,
    "Predicted Values": y_pred,
    "State": X_test["state"].map(state_mapping.set_index("encoded_state")["state"]) # Assuming "state" is the column name in
})
pred_graph_ran
```

Out[22]:

	True Values	Predicted Values	State
220842	1214	1192.111667	pa
253104	785	807.848333	tx
100939	649	655.673333	ks
252601	615	613.807251	tx
326050	750	768.781907	ar
...
267496	650	650.000000	tx
73309	675	675.000000	il
2615	1468	1490.988333	ca
39466	775	775.000000	fl
100110	575	575.000000	ks

102249 rows x 3 columns

```
In [23]: fig=px.scatter(pred_graph_ran, x="True Values", y="Predicted Values")
fig.show()
```



```
In [21]: models={"LR":[lr_r2, lr_MSE, lr_RMSE],
    "XGB":[xgb_r2, xgb_MSE, xgb_RMSE],
    "GBR":[gbr_r2, gbr_MSE, gbr_RMSE],
    "RAN":[ran_r2, ran_MSE, ran_RMSE],}
models=pd.DataFrame(models)
models=models.rename(index={0:"R^2", 1:"MSE", 2:"RMSE"})
models
```

Out[21]:

	LR	XGB	GBR	RAN
R^2	0.236733	0.813321	0.619766	0.902279
MSE	110576.509409	27044.711885	55085.563050	14157.106064
RMSE	332.530464	164.452765	234.703138	118.983638

Από τον τελευταίο πίνακα, ο οποίος έχει όλα τα δεδομένα από όλες τις παλινδρομήσεις, σύμφωνα με τις μετρικές R^2 , MSE και RMSE που παρέχονται, το μοντέλο που θα ακολουθήσουμε είναι το Random Forest Regressor(RAN). Ο λόγος είναι ότι το Random Forest Regressor έχει το υψηλότερο $R^2(0.902279)$, το χαμηλότερο MSE(14157.106064) και το χαμηλότερο RMSE(118.983638) σε σύγκριση με τα άλλα μοντέλα. Αυτό υποδεικνύει ότι το μοντέλο Random Forest Regressor έχει την καλύτερη ικανότητα πρόβλεψης της μεταβλητής εξαρτώμενης σε σχέση με τα άλλα μοντέλα.

Τελικά, υπολογίσαμε την ποσοστιαία διαφορά μεταξύ της πραγματικής τιμής και της προβλεπόμενης.

```
In [24]: mean_predicted_values = pred_graph_ran.groupby("State").mean()
mean_predicted_values["Percentage Difference"] = ((mean_predicted_values["Predicted Values"] - mean_predicted_values["True Values"]) / mean_predicted_values["True Values"]) * 100
mean_predicted_values = mean_predicted_values.sort_values(by="Percentage Difference", ascending=False)
mean_predicted_values
```

Out[24]:

	True Values	Predicted Values	Percentage Difference
State			
mo	714.893443	752.474338	5.256853
wv	1030.940594	1066.885376	3.486601
wy	882.631579	904.886051	2.521377
nh	1529.367257	1548.574080	1.255867
oh	857.030864	865.527325	0.991383
ma	1506.730056	1520.239671	0.896618
nd	920.321897	928.144969	0.850036
ia	909.406689	916.434176	0.772755
ar	832.871046	839.225442	0.762951
pa	1153.967693	1161.781681	0.677141

Εστιάζοντας στα τρία πρώτα και τα τρία τελευταία, παρατηρούμε τα εξής:

1. Missouri (MO): Η προβλεπόμενη τιμή είναι 5.26% υψηλότερη από την πραγματική τιμή.
2. West Virginia (WV): Η προβλεπόμενη τιμή είναι 3.49% υψηλότερη από την πραγματική τιμή.
3. Wyoming (WY): Η προβλεπόμενη τιμή είναι 2.52% υψηλότερη από την πραγματική τιμή.

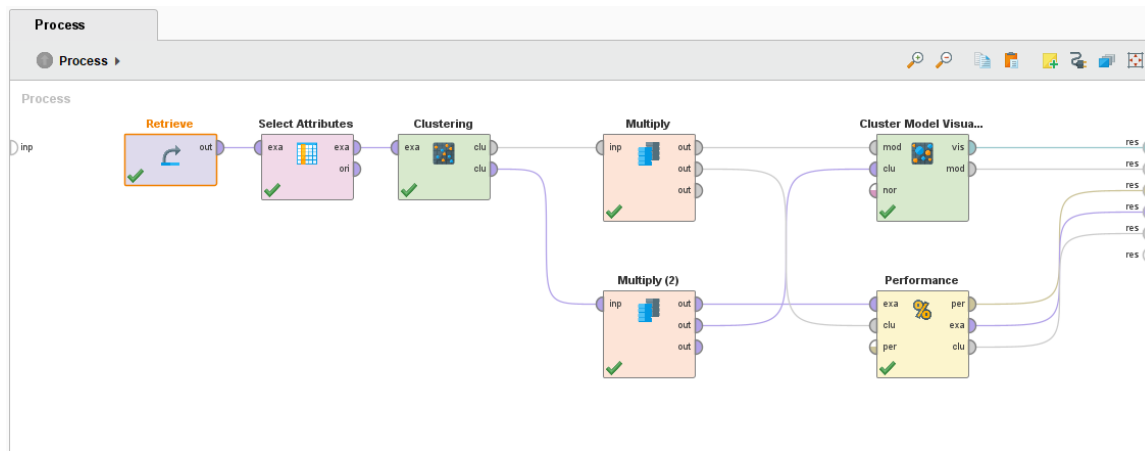
Και τα τρία πρώτα παραδείγματα δείχνουν υψηλότερη πρόβλεψη από την πραγματική τιμή, πράγμα που μπορεί να είναι σημαντικό για έναν επενδυτή ή ιδιοκτήτη κατοικίας, καθώς μπορεί να επηρεάσει τις αποφάσεις τους σχετικά με την ενοικίαση του χώρου του.

1. Nebraska (NE): Η προβλεπόμενη τιμή είναι 0.36% χαμηλότερη από την πραγματική τιμή.
2. Wisconsin (WI): Η προβλεπόμενη τιμή είναι 0.41% χαμηλότερη από την πραγματική τιμή.
3. Alaska (AK): Η προβλεπόμενη τιμή είναι 0.71% χαμηλότερη από την πραγματική τιμή.

Τα τρία τελευταία παραδείγματα δείχνουν ελαφρά υποεκτίμηση από το μοντέλο, οπότε ένας επενδυτής ή ιδιοκτήτης κατοικίας θα μπορούσε να λάβει υπόψη αυτήν την πληροφορία όταν καθορίζει την τιμή που είναι διατεθειμένος να πληρώσει για την απόκτηση μίας κατοικίας ή να δεχτεί για ένα ακίνητο που του ανήκει στις συγκεκριμένες πολιτείες.

2. Συσταδοποίηση

Για την μέθοδο της συσταδοποίησης επιλέξαμε το open-source εργαλείο RapidMiner και δημιουργήσαμε την παρακάτω διαδικασία(process).



Ας εξηγήσουμε, λοιπόν, τον τρόπο με τον οποίο εργαστήκαμε στην συγκεκριμένη διαδικασία. Αρχικά, μέσω του Jupyter Notebook, δημιουργήσαμε ένα καινούργιο αρχείο μετατρέποντας όλες τις παροχές σε δυαδικές τιμές.

```
In [2]: df['parking_options'] = df['parking_options'].apply(lambda x: 0 if x in ['Unknown', 'no parking'] else 1)
df['laundry_options'] = df['laundry_options'].apply(lambda x: 0 if x in ['Unknown', 'no laundry on site'] else 1)
df.to_csv('amenities.csv', index=False)
```

Έτσι, ανακτούμε τα δεδομένα μας και τα φορτώνουμε στην συγκεκριμένη διαδικασία. Ύστερα, επιλέγουμε τις κατάλληλες μεταβλητές, μέσω του Operator Select Attributes, που θα μας δώσουν πληροφορία για τις καταχωρήσεις. Οι μεταβλητές αυτές, όπως φαίνεται και παρακάτω, εστιάζουν στις παροχές των καταχωρήσεων.

Parameters

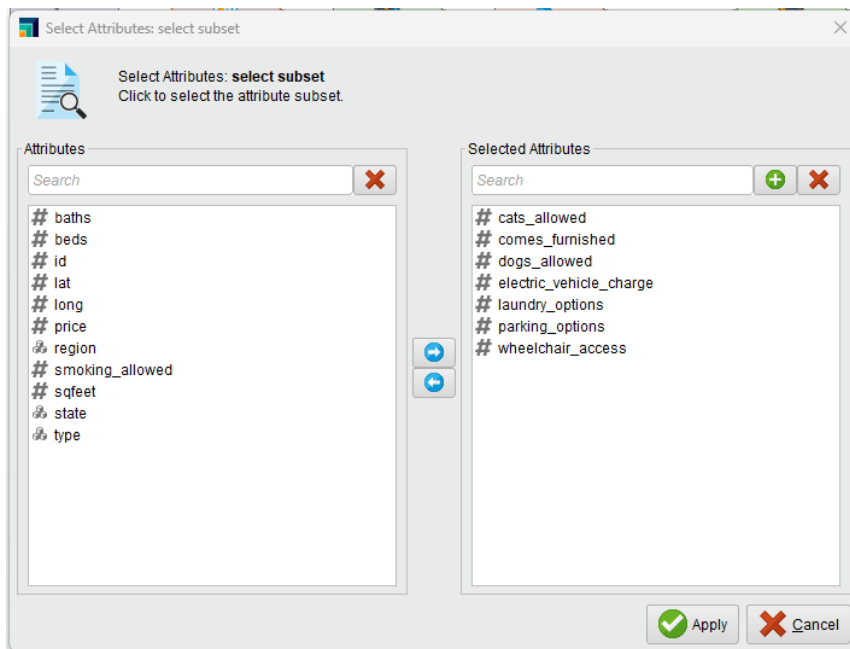
Select Attributes

type
include attributes

attribute filter type
a subset

select subset
Select Attributes...

☐ also apply to special attributes (id, label..)



Στην συνέχεια, πραγματοποιούμε ένα k-Means Clustering(συσταδοποίηση), και περνάμε τα αποτελέσματα αυτά στους Operators: Performance και Cluster Model Visualizer. Τελικά, επικεντρωνόμαστε στα παρακάτω αποτελέσματα:

Cluster Model

```
Cluster 0: 43122 items
Cluster 1: 138990 items
Cluster 2: 7343 items
Cluster 3: 56759 items
Cluster 4: 24856 items
Cluster 5: 3790 items
Cluster 6: 19375 items
Cluster 7: 9193 items
Cluster 8: 1557 items
Cluster 9: 31399 items
Cluster 10: 4444 items
Total number of items: 340828
```

Cluster	cats_allowed	dogs_allowed	wheelchair_access	electric_vehicle_charge	comes_furnished	laundry_options	parking_options
Cluster 0	0.016	0.008	0.001	0.001	0.004	0	0.063
Cluster 1	0.986	1	0	0.008	0.019	1	1
Cluster 2	0.179	0.032	0.039	0.004	0.042	1	0
Cluster 3	0.999	1	0.062	0.004	0.015	1	0
Cluster 4	0.999	1.000	0.043	0.006	0.029	0	0
Cluster 5	0.005	0.096	0	0.007	1	0.891	0.996
Cluster 6	0.988	1.000	1	0.082	0.198	0.993	0.933
Cluster 7	1	0	0.049	0.005	0.018	0.961	1
Cluster 8	0.013	0.039	1	0.017	0.321	0.960	0.981
Cluster 9	0	0	0	0.004	0	1	1
Cluster 10	0.925	1	0.025	0.002	0.061	0	1

Όπως παρατηρούμε, οι συστάδες(clusters) με τα περισσότερα items είναι το cluster 0, το cluster 1, το cluster 3, το cluster 4 και το cluster 9. Αναλύοντας τα ποσοστά εμφάνισης τα οποία παρουσιάζονται παραπάνω, καταλήγουμε στα εξής:

1. **Barren Listings**(Cluster 0): Σε αυτή την συστάδα ανήκουν οι καταχωρήσεις που δεν προσφέρουν καμία παροχή. Το ποσοστό των συγκεκριμένων καταχωρήσεων ανέρχεται στο 12.5%.
2. **Pet-Friendly Haven with Laundry & Parking Amenities** (Cluster 1): Σε αυτή την συστάδα ανήκουν όλοι οι χώροι που παρέχουν την άδεια κατοχής σκύλου και γάτας, καθώς και πλυντήριο ρούχων και θέση πάρκινγκ. Ποσοστό εμφάνισης: 40.5%.
3. **Pet-Friendly Nest with Laundry Option & NO Parking** (Cluster 3): Η ακριβώς ίδια κατηγορία με προηγούμενως, απλά στην συγκεκριμένη δεν συμπεριλαμβάνεται η θέση πάρκινγκ. Ποσοστό εμφάνισης: 16.5%.
4. **Pet Heaven** (Cluster 4): Η συστάδα ενθυλακώνει καταχωρήσεις που συμπεριλαμβάνουν σε ποσοστό 100% την δυνατότητα κατοχής γάτας και σκύλου. Ποσοστό εμφάνισης: 7%.
5. **Convenient Laundry & Parking Hub** (Cluster 9): Η συστάδα περιλαμβάνει τις παροχές του πλυντηρίου και της θέσης πάρκινγκ. Το ποσοστό εμφάνισης είναι 9.1%.

Για να αναλύσουμε και την επιχειρηματική αξία, την οποία μας προσφέρουν οι παραπάνω συστάδες, καταλήγουμε στις ακόλουθες πληροφορίες για κάποιον επενδυτή ή κάτοχο ακινήτου:

- **Ανάλυση Παροχών:** Είναι αρκετά εύκολο να διαπιστωθούν οι παροχές εκείνες που εμφανίζονται στις περισσότερες συστάδες. Για παράδειγμα, φαίνεται ότι η παροχή πλυντηρίου εμφανίζεται σε πολλά clusters και συνδυάζεται ποικιλοτρόπως με άλλες παροχές.
- **Ζήτηση για Κατοικίδια:** Μια ακόμα παρατήρηση είναι αυτή της ζήτησης χώρων στους οποίους επιτρέπονται τα κατοικίδια ζώα. Φαίνεται, από τα αποτελέσματά μας, ότι η συγκεκριμένη παροχή εμφανίζεται στις περισσότερες συστάδες με μεγάλα ποσοστά εμφάνισης.
- **Συνολική Αξία:** Παρατηρούμε ότι αρκετές παροχές δεν συμπεριλαμβάνονται στις κύριες συστάδες. Οπότε, προκειμένου να αυξήσει την ζήτηση και την τιμή του ακινήτου του, κάποιος ιδιοκτήτης θα μπορούσε να συμπεριλάβει και άλλες παροχές, όπως να βελτιώσει την προσβασιμότητα για άτομα σε αναπηρικά αμαξίδια ή να εξοπλίσει τον χώρο με κάποιο φορτιστή για ηλεκτρικά αυτοκίνητα.
- **Ανάλυση Ανταγωνισμού:** Συνυφασμένος με την συνολική αξία του ακινήτου είναι ο ανταγωνισμός. Ένας ιδιοκτήτης μπορεί να παρατηρήσει τις σχέσεις μεταξύ των παροχών και τις ελλείψεις που υπάρχουν στην αγορά, δημιουργώντας νέες επενδυτικές ευκαιρίες.