Problem statement

Problem Description

To predict rental price of units in Dallas, Texas.

Dataset: [2]
Features: category, title, body, amenities, bathrooms, bedrooms, currency, fee, has_photo, pets_allowed, price, price_display, price_type, square_feet, address, cityname, state, latitude, longitude, source, time
99826 records

Steps

Preprocess

- Explore data, decide what features to use to predict price

Cleaning

- For each selected features, handle nan, and convert to proper dtype to be fed to models (categorical: encode; numerical)

Please see detailed preprocess & cleaning steps in next section (data cleaning&pre-processing)

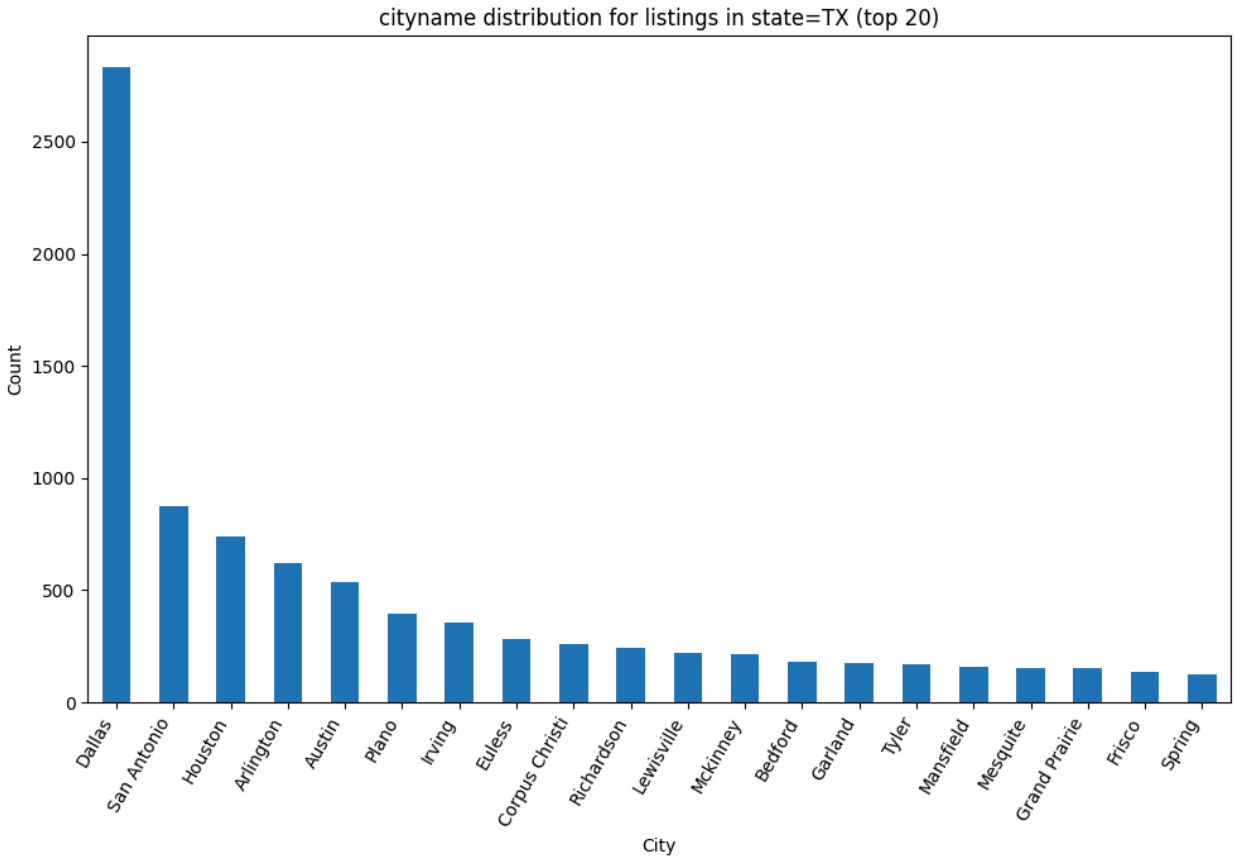
Model approach

- Linear and non-linear
- perform hyper param tuning for both, with the same cv (split the training data into 5 disjoint folds; each param combo is trained on 4 folds and validated on the 5th, rotating through all folds), and same score/criteria (mse loss)

Explore data, 5 plots/ explanation

Cityname distribution for listings in state=TX (top 20)

- Note data where state=TX, city=Dallas alone exceeds 2500 rows, which suffices the assignment requirement of greater than 1k data instances



Distribution (category, fee, currency, price_type, state) for data with state=”TX”

- Note for column category, fee, price_type, and currency, all data of state TX (thus of city Dallas which belongs to TX) have same value, can drop these 4 columns when analyze Dallas data

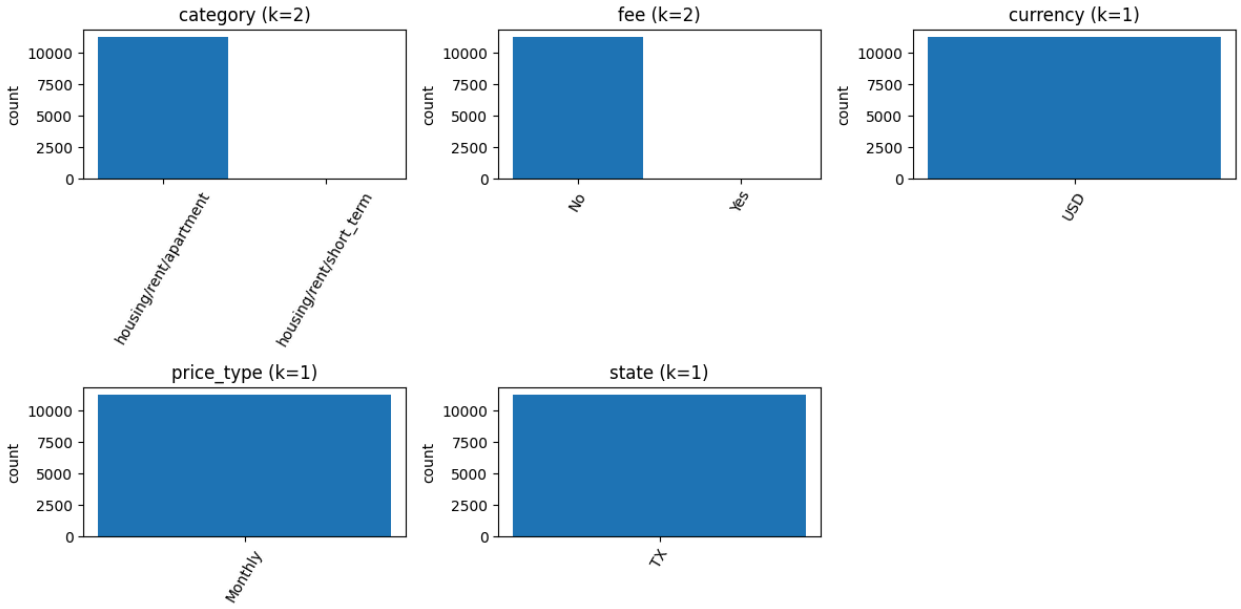


Table of Missing Count for selected features

- Note among the 7 selected features (with price being the target), bathrooms and bedrooms have very few nan values, so can drop rows safely with nan values for these 2 features.
- While for feature amenities and pets_allowed, there are quite a few nan values, so cannot just drop nans, need to figure out a way to address such values.

amenities	546
bathrooms	1
bedrooms	3
price	0
square_feet	0
has_photo	0
pets_allowed	1873

dtype: int64

Table of unique values for feature 'has_photo'& 'pets_allowed'

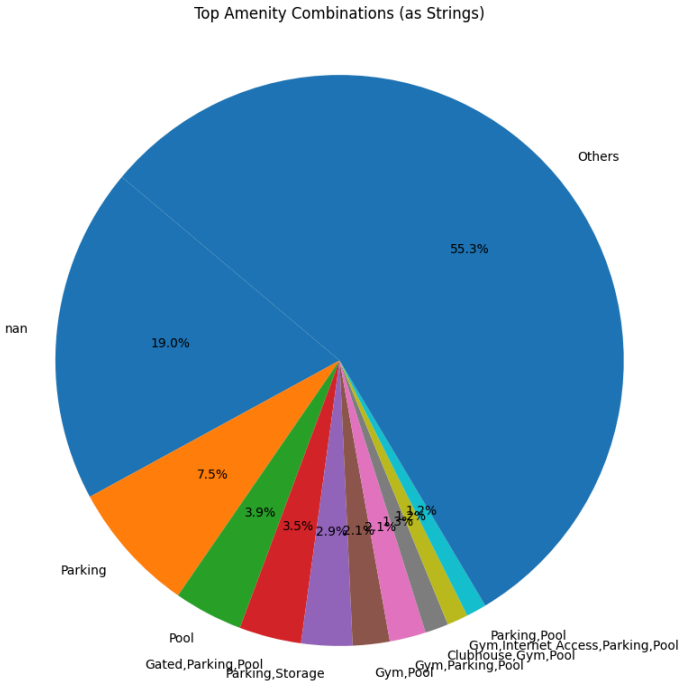
- Note has_photos has 3 values, so can be encoded by turning these col to 3 binary cols, and only 1 col among the 3 takes 1 to indicate the original value, the others take 0.
- Note pets_allowed has 4 values, among which, “Cats, Dogs”, “Cats”, and “Dogs” are similar, meaning “allowing pets”, while value “nan” means “not allowing dog”. So can encode this col using a binary col, where 1 means allowing, 0 means not allowing.

“has_photos”
1. Yes: 1762
2. Thumbnail: 732
3. No: 368

“pets_allowed”
1. nan: 1873
2. Cats,Dogs: 965
3. Cats: 22
4. Dogs: 2

Piechart of Top Amenity Combinations

Note col amenities has a list of items as its value, and has many unique values. To clean/encode the feature, can parse each value, and gather the most mentioned items (say top 20 items), and turn each item into a column, use 0 and 1 to indicate if the item is provided.



Data cleaning/pre-processing: what is done and why

Filter to focus on only Dallas, Texas data.

- Because Dallas has the most data records among all citynames (more than 2500, while the cityname in second palace has around 1000 records), which is sufficient for the assignment requirement, even after removing a few records with nan values

Drop columns:

“category”: b/c all Dallas data has same value for category:housing/rent/apartment
“title”: b/c this column is long text, hard to process
“body”: , b/c also long text, hard to process
“currency”:b/c all Dallas data has same value: “USD”
“fee”: b/c all Dallas data has same value: “No”
“price_display”: redundant, will use feature “price”
“price_type”: b/c all Dallas data has same value: “Monthly”
“address”: irrelevant of the task
“cityname”: b/c all Dallas data has same value: “Dallas”
“state”: b/c all Dallas data has same value: “Texas”
“latitude”:irrelevant of the task
“longitude”: irrelevant of the task
“source”: irrelevant of the task
 “time”: irrelevant of the task

Keep the rest features and clean each as following:

‘square_feet’
- convert dtype from `object` to numerical

‘bathrooms’
- Drop rows with nan values (only 1 rows)
- convert dtype from `object` to numerical

‘bedrooms’
- Drop rows with nan values (only 3 rows)
- convert dtype from `object` to numerical

'amenities'
- Since value is a list of items, turn each value into individual items and combine all unique items into a py list (only keeps items appear more than `threshold` times), then encode with `MultiLabelBinarizer`, which creates a binary column for each unique item, which takes 0 or 1 to indicate if that item is provided.

‘Has_photo’
- Originally has 3 unique values: “Yes”, “No”, “Thumnail”, no nan
- Convert to 3 one-hot features: has_photo_Yes, has_photo_Thumbnail, has_photo_No

‘Pets_allowed’

- Originally has 4 unique values: nan, “Cats, Dogs”, “Cats”, “Dogs”
- Convert to 1 binary features: “pets_allowed_allow”; for all non-nan value, take value1; o/w: 0

Split into train and test

Apply log to target, then split train 80%, test 20%, with seed 42

```
Xtr, Xte, ytr, yte = train_test_split(X, np.log1p(y), test_size=0.2, random_state=42)
```

Models, compare, explain, which one to pick

Tried 2 models, ElasticNet (a linear model that combines both L1 and L2 penalties), and HGBR (a tree-based ensemble algorithm, that is able to capture nonlinear relations between features)

```
# linear baseline
lin = Pipeline([
    ('sc', StandardScaler()), # feature scaling
    ('enet', ElasticNet(alpha=0.1, # overall regularization strength (↑alpha
                        l1_ratio=0.5, # 0=ridge (all L2), 1=lasso (all L1);
                        max_iter=1000))
])
```

EN gives MSE 17464.49

```
gbr = HistGradientBoostingRegressor(
    learning_rate=0.05,
    max_depth=None, # max depth of each tree; None = no explicit depth cap
    max_leaf_nodes=31,# cap on leaves per tree; more leaves = more flexible
    l2_regularization=0.0,# L2 penalty on leaf values; >0 shrinks/smooths predictions
    max_iter=200,
    random_state=42
)
```

HBG gives MSE 344.13

MSE is mean-square-error computed between true price and predicted price on a test set. A smaller MSE indicates a better estimate of rental price. So pick HGBR over ElasticNet.

HyperParam tuning

Perform param grids search for both models, using the same train+validation folds, and scorer (MSE).

Linear model

tuning the following parameters,

```
# search grid on train
param_grid = {
    'enet__alpha': np.logspace(-3, 1, 10),
    'enet__l1_ratio': [0.1, 0.5, 0.9],
}
```

And the best param combo with params:

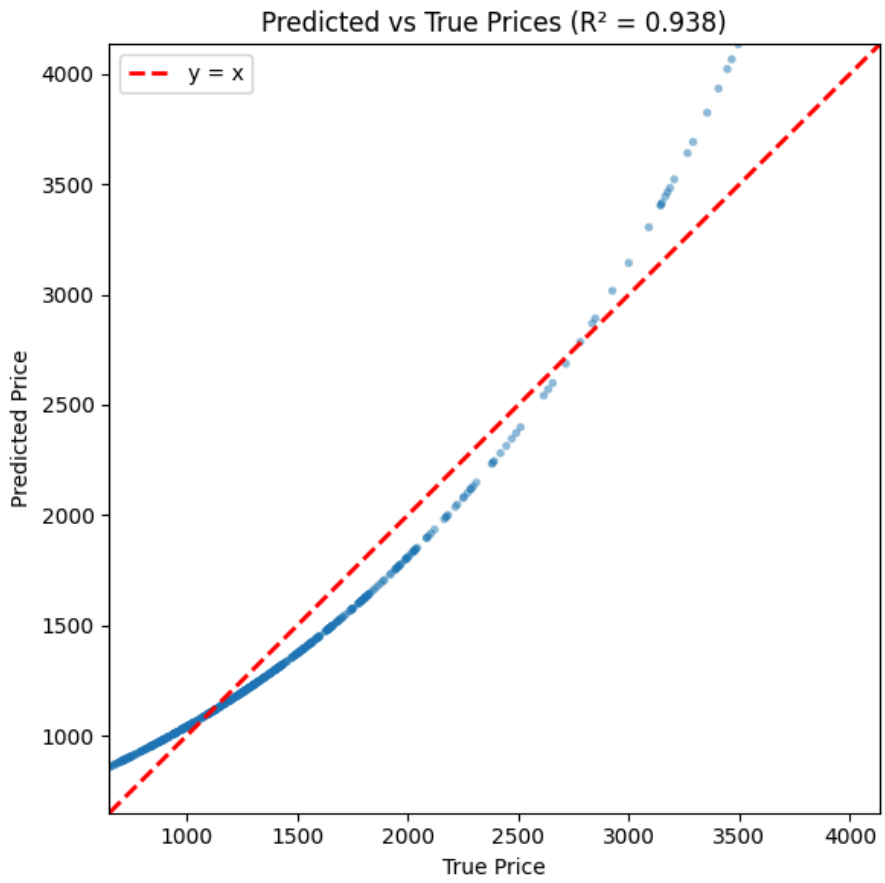
```
Best params: {'enet__alpha': np.float64(0.05994842503189409), 'enet__l1_ratio': 0.9}
```

With metrics:

```
Best CV MSE (price units): 18398.60

Test — MSE: 17650.95 |  MAE: 109.08 | R²: 0.938
```

And R^2 plot (predicted vs true for test data) as follows:



Non-linear model

Tuning the following parameters:

```
# param grid
param_distributions = {
    "learning_rate": [0.02, 0.05, 0.1],
    "max_depth": [None, 3, 5, 7],
    "max_leaf_nodes": [15, 31, 63, ],
    "l2_regularization": [0.0, 1e-3, 1e-2, 1e-1],
}
```

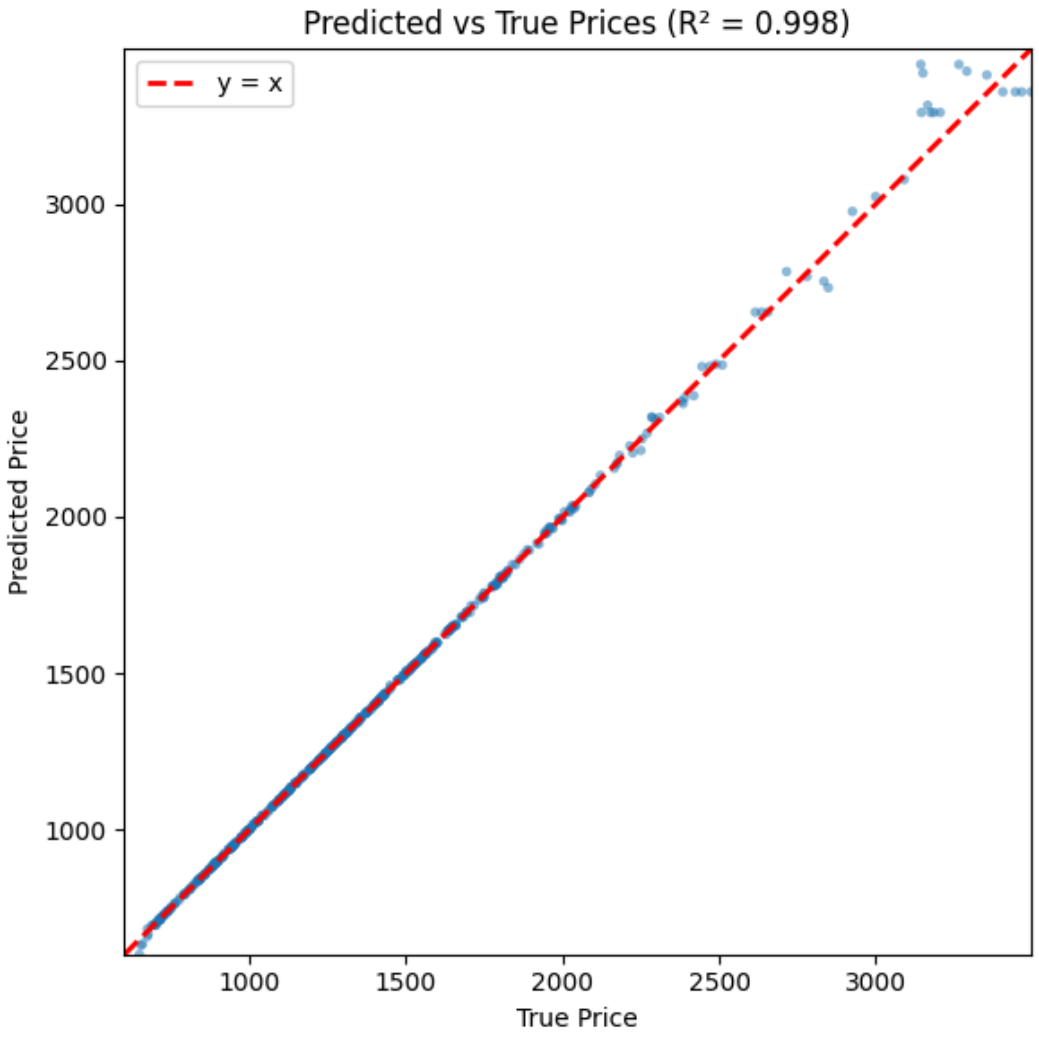
Which gives best model with parameters:

```
Best params: {'max_leaf_nodes': 31, 'max_depth': None, 'learning_rate': 0.1, 'l2_regularization': 0.01}
```

evaluation metrics:

```
Best CV MSE (price units): 739.72
Test MSE: 666.60
```

And R^2 plot (predicted vs true for test data):



Finding, conclusions, future research

Using features such as amenities, bathrooms, bedrooms, price, square_feet, has_photo, and pets_allowed, and after proper preprocessing and encoding, the proposed Histogram-based Gradient Boosting (HGB) model achieves highly accurate rental price predictions for Dallas, with an R^2 score of 0.98 on the test set—approaching perfect performance.

For future improvements, incorporating geolocation data—specifically latitude and longitude, which were omitted due to processing complexity—could further enhance model performance. A promising direction would be to engineer spatial features or use clustering-based location encodings to capture regional pricing patterns more effectively.

Reference

[2] Apartment for Rent Classified [Dataset]. (2019). UCI Machine Learning Repository. https://doi.org/10.24432/C5X623.