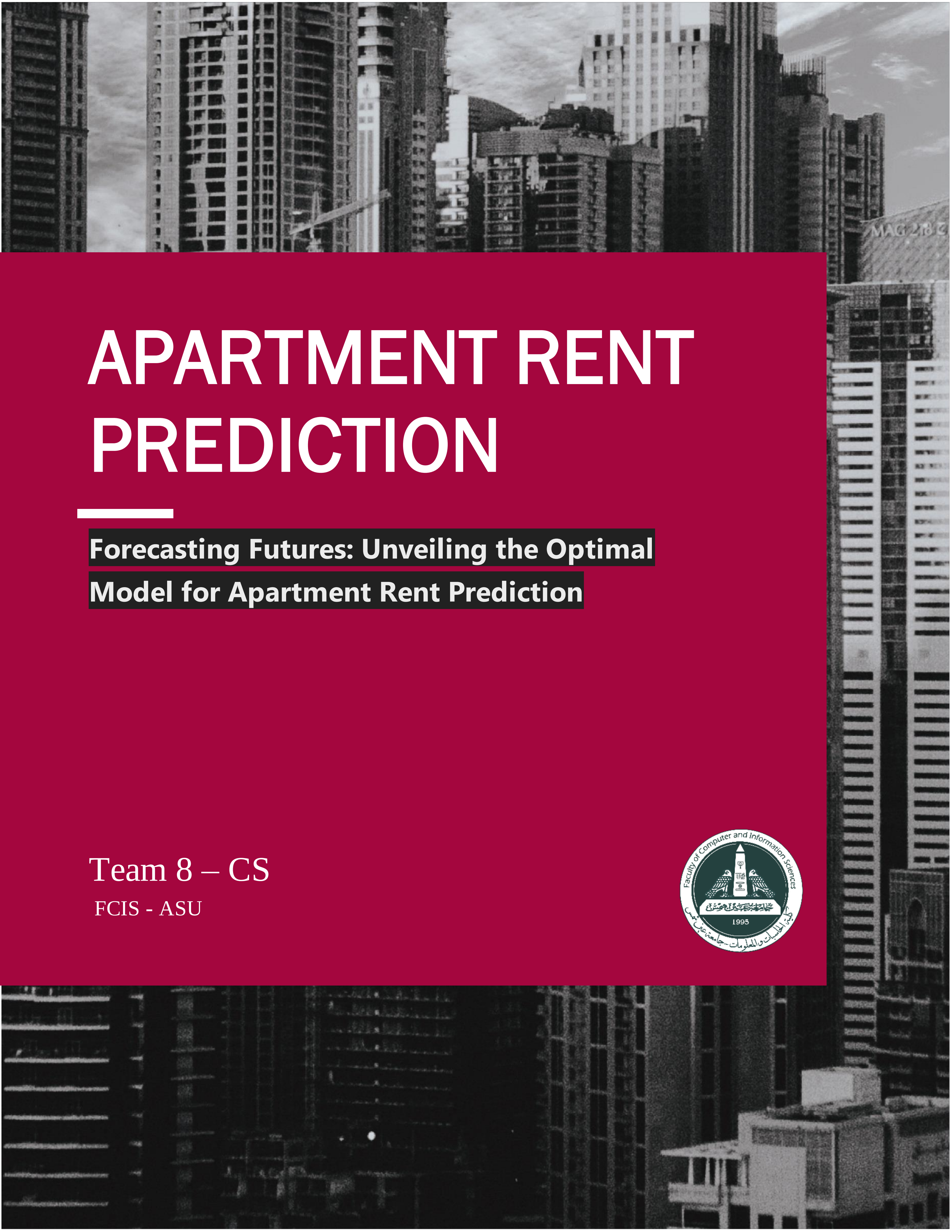


APARTMENT RENT PREDICTION

Forecasting Futures: Unveiling the Optimal Model for Apartment Rent Prediction

Team 8 – CS

FCIS - ASU

Regression Model

1. Exploring Features in Our Dataset

First of all, we have to determine the **shape**, **features**, and **statistical description** of our dataset in order to proceed with our learning pipeline. We've managed to achieve these results using the following lines of code:

```
data = pd.read_csv("data/ApartmentRentPrediction.csv")

# Creating Dataframe
print("Shape of the DataFrame:", data.shape)

data.head()

print("Statistical description of the DataFrame:")
print(data.describe())

print("Columns in the DataFrame:")
print(data.columns)
```

Output:

```
Shape of the DataFrame: (9000, 22)
Statistical description of the DataFrame:
                 id    bathrooms     bedrooms         price   square_feet     latitude    longitude          time
count  9.000000e+03  8970.000000  8993.000000   9000.000000   9000.000000  8993.00000  8993.000000  9.000000e+03
mean   5.623668e+09     1.380769     1.744023   1487.286222    947.138667    37.67689   -94.778612  1.574906e+09
std    7.007402e+07     0.616171     0.942446   1088.561190    668.806214     5.51527    15.769232  3.755142e+06
min    5.508654e+09     1.000000     0.000000    200.000000    106.000000    21.31550  -158.022100  1.568744e+09
25%    5.509250e+09     1.000000     1.000000    950.000000    650.000000    33.66200  -101.858700  1.568781e+09
50%    5.668610e+09     1.000000     2.000000   1275.000000    802.000000    38.75550   -93.707700  1.577358e+09
75%    5.668626e+09     2.000000     2.000000   1695.000000   1100.000000    41.34980   -82.446800  1.577359e+09
max    5.668663e+09     8.500000     9.000000  52500.000000  40000.000000    61.59400   -70.191600  1.577362e+09
Columns in the DataFrame:
Index(['id', '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'],
      dtype='object')
```

As shown above, our dataset contains a total of **9000 rows** & **22 features**, we can also deduce from the statistics, *which only shows numerical features*, that we have quite a few columns with categorical nature.

Dataset Features Description:

- **ID**: Unique identifier for each apartment listing.
- **Category**: Category of the listing on its website (Housing/Rent/X).
- **Title**: Title of the listing on the website.
- **Body**: Description of the listing.
- **Amenities**: Amenities available in the apartment.
- **Bathrooms**: Number of bathrooms in the apartment.
- **Bedrooms**: Number of bedrooms in the apartment.
- **Currency**: Currency used for the price.
- **Fee**: Any additional fees associated with the rental.
- **Photo Attached**: Indicates whether the listing has photos, thumbnails, or none.
- **Allowed Pets**: Indicating what pets are allowed in the apartment.
- **Price**: Rental price.
- **Displayed Price**: Display of the rental price.
- **Price Type**: Time frequency of payment (Weekly or Monthly).
- **Square Feet**: Size of the apartment in square feet.
- **Address**: Address of the apartment.
- **City Name**: City where the apartment is located.
- **State**: State where the apartment is located.
- **Latitude**: Latitude coordinate of the apartment location.
- **Longitude**: Longitude coordinate of the apartment location.
- **Source**: Source website of the listing.
- **Time**: Timestamp of the listing.

id	category	title	body	amenities	bathr	be	curre	fee	has_phot	pets_all	price	price_	price_ty	squ	address	cityname	st	latitude	longitude	source	time
5508832632	housing/rent/apartment	Two BR - $1,194/mo - Apartme	Come experience.	Clubhouse,Fireplace,Gym,I	2.5	2	USD	No	Thumbnail	Cats,Dogs	1194	$1,194	Monthly	800		Cary	NC	35.7585	-78.7783	RentDigs.com	1.57E+09
5664576849	housing/rent/apartment	One BR 640 West Wilson Stree	This unit is locate	Cable or Satellite,Dishwas	1	1	USD	No	Thumbnail	Cats,Dogs	1370	$1,370	Monthly	795	640 West Wilsor	Madison	WI	43.0724	-89.4003	RentLingo	1.58E+09
5668619365	housing/rent/apartment	One BR 2777 Sw Archer Rd	This unit is locate	Basketball,Cable or Satelli	1	1	USD	No	Thumbnail	Cats,Dogs	1009	$1,009	Monthly	560	2777 SW Archer I	Gainesville	FL	29.6533	-82.3656	RentLingo	1.58E+09
5668632604	housing/rent/apartment	One BR 534-542 Park Avenue	This unit is located at 534-542 Park Avenue,		1	1	USD	No	Thumbnail	Cats,Dogs	695	$695	Monthly	600	534-542 Park Ave	Omaha	NE	41.2562	-96.0404	RentLingo	1.58E+09
5668637441	housing/rent/apartment	Three BR 2216 S Palm Grove Av	This unit is located at 2216 S Palm Grove Ave		3	3	USD	No	Thumbnail		3695	$3,695	Monthly	1600	2216 S Palm Gro	Los Angeles	CA	34.0372	-118.2972	RentLingo	1.58E+09
5664597879	housing/rent/apartment	Three BR 680 Bromley Dr.	This unit is located at 680 Bromley Dr., Bator		2	3	USD	No	Thumbnail		1750	$1,750	Monthly	2300	680 Bromley Driv	Baton Rouge	LA	30.4415	-91.1012	RentLingo	1.58E+09
5509268702	housing/rent/apartment	Studio in Lynnwood	We believe that e	Gym,Pool	1	2	USD	No	Yes		2470	$2,470	Monthly	1079		Lynnwood	WA	47.8616	-122.2729	RentDigs.com	1.57E+09
5664594606	housing/rent/apartment	Three BR 1712 Donald Dr	This unit is located at 1712 Donald Dr, Shaw		2	3	USD	No	Thumbnail	Cats,Dogs	1250	$1,250	Monthly	1177	1712 Donald Driv	Shawnee	OK	35.3537	-96.8923	RentLingo	1.58E+09
5668628991	housing/rent/apartment	One BR 2975 Blackburn St Apa	This unit is located at 2975 Blackburn St Apa		1	1	USD	No	Thumbnail	Cats,Dogs	1789	$1,789	Monthly	678	2975 Blackburn S	Dallas	TX	32.8212	-96.7853	RentLingo	1.58E+09
5668609854	housing/rent/apartment	Two BR 290 9th Ave Sw	This unit is locate	Cable or Satellite,Dishwas	1	2	USD	No	Thumbnail		1225	$1,225	Monthly	995	290 9th Avenue S	Forest Lake	MN	45.2764	-92.9901	RentLingo	1.58E+09
5508806497	housing/rent/apartment	Wards Corner, prime location	Bondale Apartme	Wood Floors	1	2	USD	No	Thumbnail	Cats,Dogs	715	$715	Monthly	414		Norfolk	VA	36.9141	-76.2882	RentDigs.com	1.57E+09

2. Preprocessing & Feature Engineering

In order to check if our dataset is healthy & clear of invalid formats, we first make sure it's **complete**, thus we check for **NaN** values. Additionally, we don't want our model to **overfit** the training data and memorize the results, that's why it's essential to deal **duplicated data** as well

The following piece of code demonstrates the process:

```
# Preprocessing and Feature Selection
print("Checking for Missing Values:")
print(data.isna().sum())

print("Checking for Duplicated Data:")
print(data.duplicated().sum())
```

```
Checking for Duplicated Data:
0
```

```
Checking for Missing Values:
id                  0
category            0
title               0
body                0
amenities        3185
bathrooms          30
bedrooms            7
currency            0
fee                 0
has_photo           0
pets_allowed     3751
price               0
price_display       0
price_type          0
square_feet         0
address          2971
cityname           66
state              66
latitude            7
longitude           7
source              0
time                0
dtype: int64
```

According to the shown results, we have data loss ranging from as little as **0.07% - 0.7%** to a stunning **33% - 42%**. On the other hand, no duplicate data was found.

So, to complete our data, we'll have to inject artificial entries so that our model doesn't **underfit** the training set.

Since most of the affected features are categorical, we've chosen to fill them using the **mode** of each one

Note: *We tried experimenting filling with the* ***mean*** *in numerical features,* ***ex. Bathrooms,*** *but we observed less error using the mode.*

The code below shows what have been done:

```python
values_to_choose_from = data['address'].dropna().unique()  # Get unique values from the column, excluding NaNs
random_values = np.random.choice(values_to_choose_from, size=len(data), replace=True)  # Generate random values

# Fill the column with random values
data['address'] = random_values

bedroom_mode = data["bedrooms"].mode()[0]
bathroom_mode = data["bathrooms"].mode()[0]
cityname_mode = data["cityname"].mode()[0]
state_mode = data["state"].mode()[0]
lat_mode = data["latitude"].mode()[0]
long_mode = data["longitude"].mode()[0]
pets_mode = data["pets_allowed"].mode()[0]
amenities_mode = data["amenities"].mode()[0]

print("Most common value in bedrooms:", bedroom_mode)
print("Most common value in bathrooms:", bathroom_mode)
print("Most common value in cityname:", cityname_mode)
print("Most common value in state:", state_mode)
print("Most common value in latitude:", lat_mode)
print("Most common value in longitude:", long_mode)

# Handling missing values

data["amenities"].fillna(amenities_mode, inplace=True)
data["pets_allowed"].fillna(pets_mode, inplace=True)
data["bathrooms"].fillna(bathroom_mode, inplace=True)
data["bedrooms"].fillna(bedroom_mode, inplace=True)
data["cityname"].fillna(cityname_mode, inplace=True)
data["state"].fillna(state_mode, inplace=True)
data["latitude"].fillna(lat_mode, inplace=True)
data["longitude"].fillna(long_mode, inplace=True)

print("Checking for Missing Values after handling:")
print(data.isna().sum())
```

```
Most common value in bedrooms: 1.0
Most common value in bathrooms: 1.0
Most common value in cityname: Austin
Most common value in state: TX
Most common value in latitude: 30.3054
Most common value in longitude: -97.7497
Checking for Missing Values after handling:
id             0
category       0
title          0
body           0
amenities      0
bathrooms      0
bedrooms       0
currency       0
fee            0
has_photo      0
pets_allowed   0
price          0
price_display  0
price_type     0
square_feet    0
address        0
cityname       0
state          0
latitude       0
longitude      0
source         0
time           0
```

*The **address** feature was handled by filling the set with random values as shown above.*

Now, for the next step, which is label encoding our categorical features using the below function:

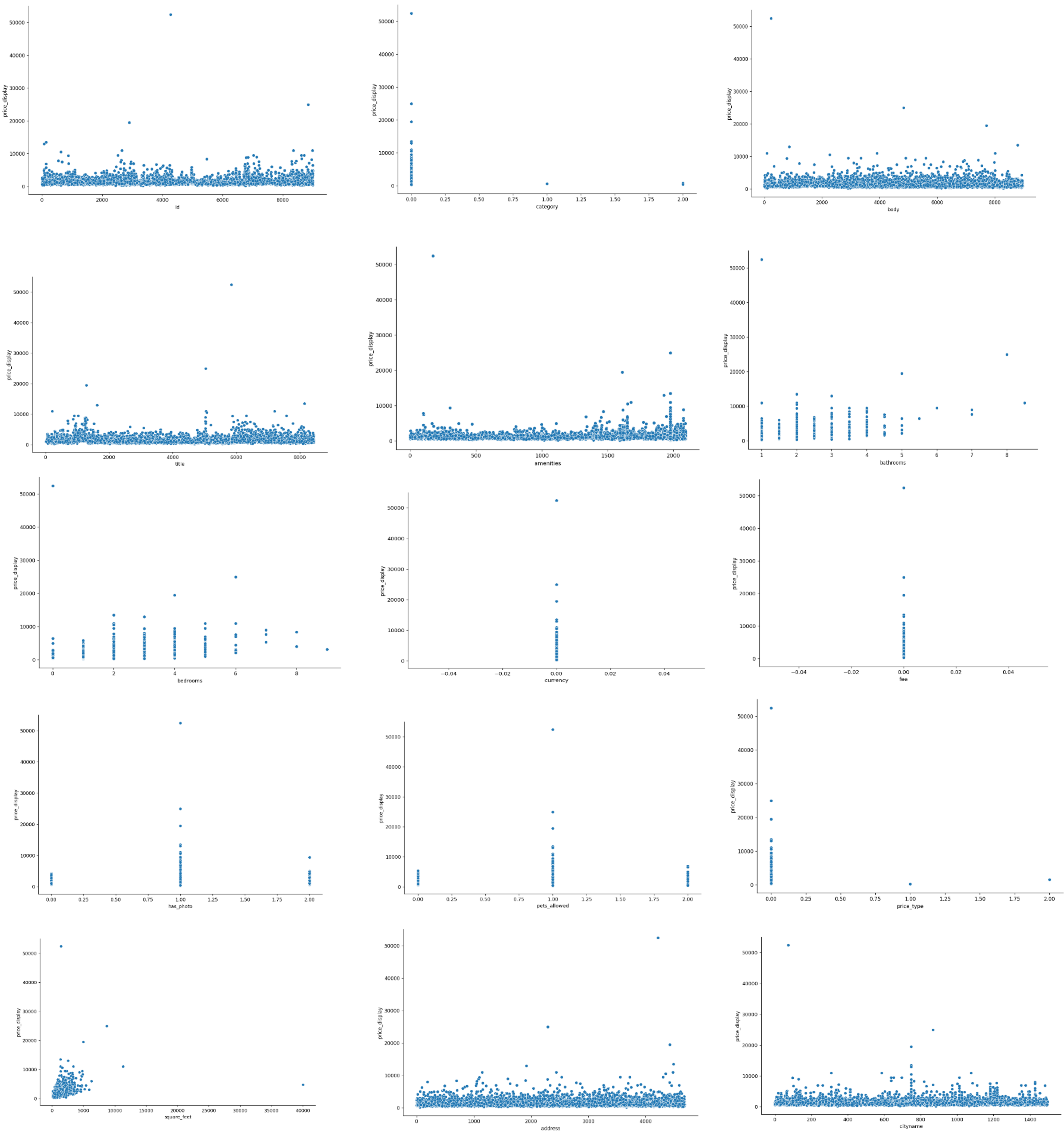
```python
# Encoding
def encode_categorical(data, columns):
    label_encoder = LabelEncoder()
    for column in columns:
        data[column] = label_encoder.fit_transform(data[column])
    return data

categorical_columns = ['amenities', 'cityname', 'state', 'address', 'category','id', 'title', 'body', 'source', 'time','currency',
                       'has_photo','price_type', 'pets_allowed', 'fee']
data_encoded = encode_categorical(data, categorical_columns)

# Displaying the encoded DataFrame
print("Encoded DataFrame:")
print(data.head())
```

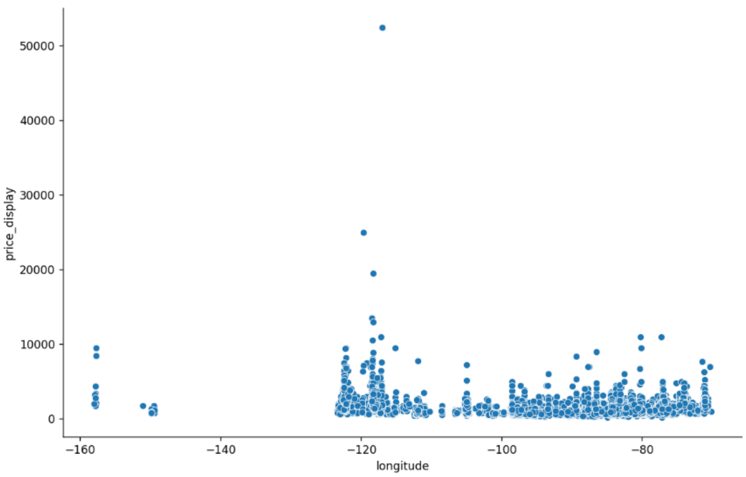
Encoded DataFrame:

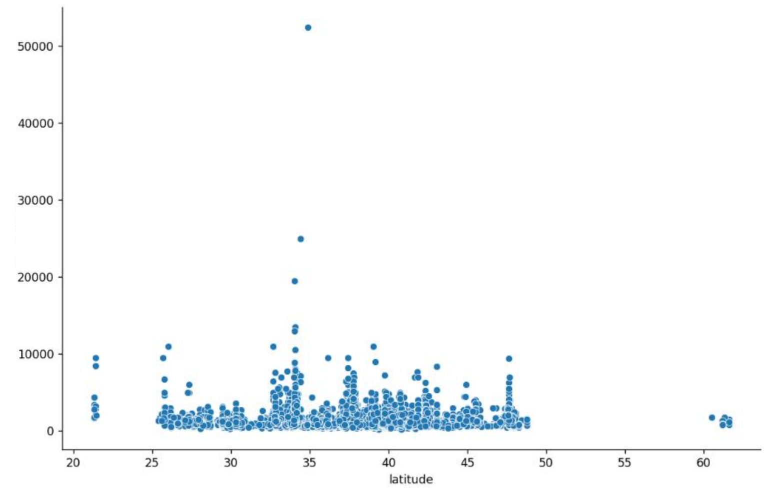
	id	category	title	body	amenities	bathrooms	bedrooms	currency	fee	...	price_type	square_feet	address	cityname	state	latitude	longitude	source	time
0	501	0	6851	332	1277	2.5	2.0	0	0	...	0	800	2363	221	26	35.7585	-78.7783	7	489
1	3552	0	3927	7039	815	1.0	1.0	0	0	...	0	795	778	763	47	43.0724	-89.4003	8	3247
2	5886	0	2999	4989	464	1.0	1.0	0	0	...	0	560	3683	481	9	29.6533	-82.3656	8	4458
3	7745	0	3733	6603	1977	1.0	1.0	0	0	...	0	600	3141	982	28	41.2562	-96.0404	8	5276
4	8429	0	6265	4474	1977	3.0	3.0	0	0	...	0	1600	3917	747	4	34.0372	-118.2972	8	5567

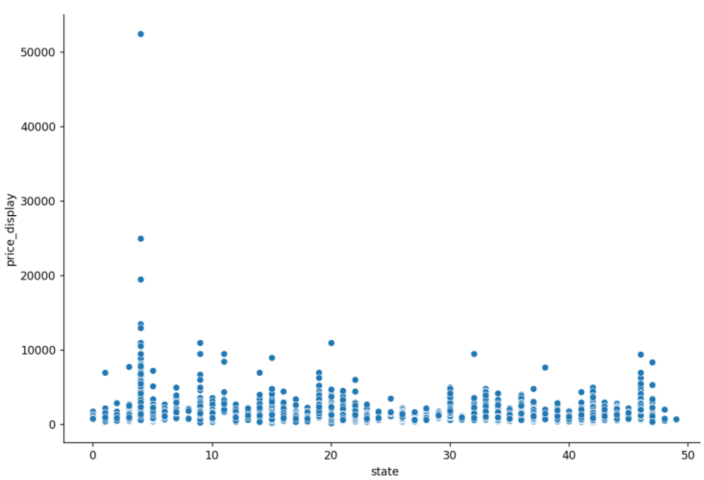
Currently, our data is clean, and complete, which means it’s ready to be **plotted**!

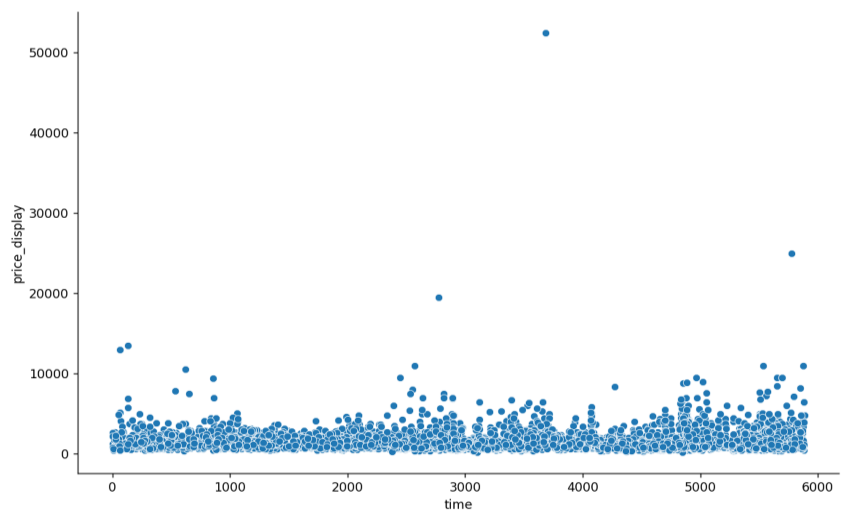
Here’s a first look at the plots against our target feature:

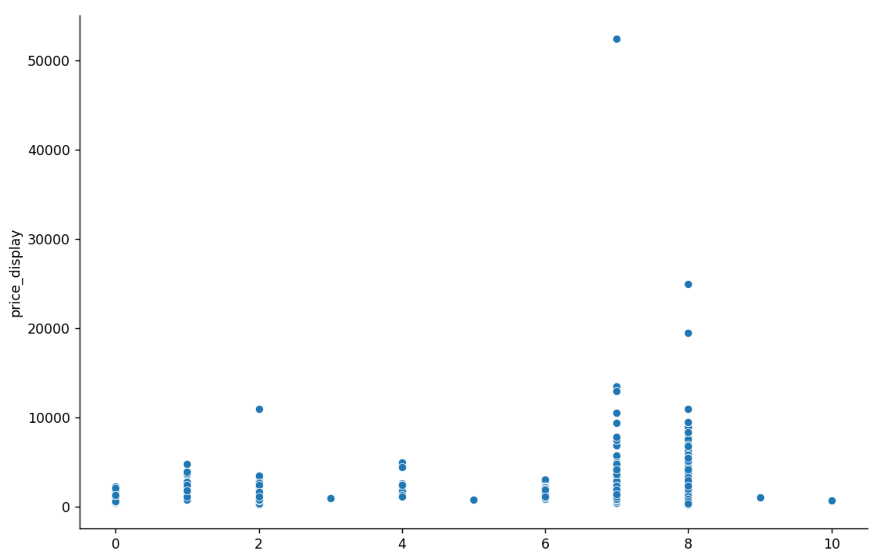



50000
40000
30000
20000
10000
0
price_display
−160
−140
−120
−100
−80
longitude


50000
40000
30000
20000
10000
0
20
25
30
35
40
45
50
55
60
latitude


50000
40000
30000
20000
10000
0
price_display
0
10
20
30
40
50
state


50000
40000
30000
20000
10000
0
price_display
0
1000
2000
3000
4000
5000
6000
time


50000
40000
30000
20000
10000
0
price_display
0
2
4
6
8
10
source

As we can see, the data is kind of **broad**, as some contain only 1 value, and while others are affected by **outliers**.

We'll solve the outliers issue using **Z-Score** measure, as well is **IQR** to statistically get rid of any **noise** that would cause **bias** in our model.

Discussed in the code below:

```
# Calculate Z-scores for each column
data = data.astype(float)  # Convert to float
z_scores = np.abs(stats.zscore(data))

# Set threshold for identifying outliers (e.g., Z-score > 3)
threshold = 3

# Find indices of outliers
outlier_indices = np.where(z_scores > threshold)

# Remove outliers from DataFrame
data_cleaned = data.drop(outlier_indices[0])

data = data_cleaned.apply(pd.to_numeric, errors='coerce').dropna()

Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

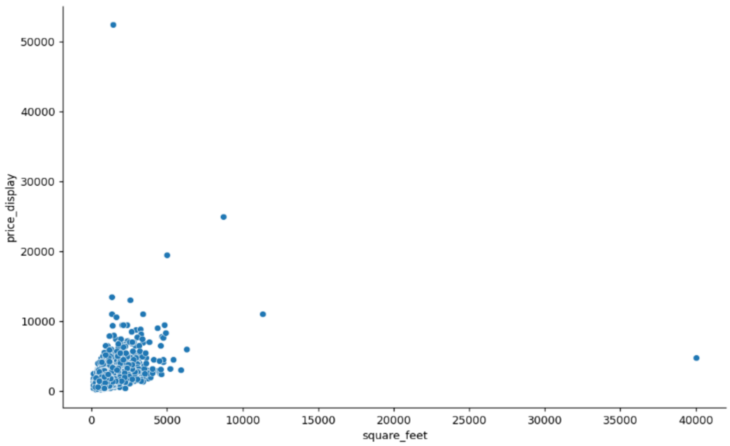
*Any value with Z > 3 OR > Q3+(1.5 * IQR) OR < Q1 – (1.5 * IQR) has been discarded*

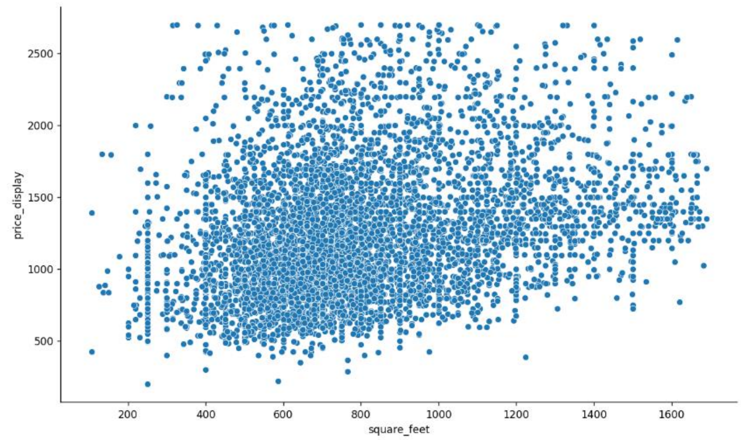
Here's a comparison of the target feature before & after:

```
count     9000.000000
mean      1487.286222
std       1088.561190
min        200.000000
25%        950.000000
50%       1275.000000
75%       1695.000000
max      52500.000000
Name: price_display, dtype: float64
```

```
count    5976.000000
mean     1247.540161
std       464.438785
min       200.000000
25%       899.000000
50%      1175.000000
75%      1500.000000
max      2700.000000
Name: price_display, dtype: float64
```

<u>Here’s a comparison between the Square Feet feature plots before & after</u>



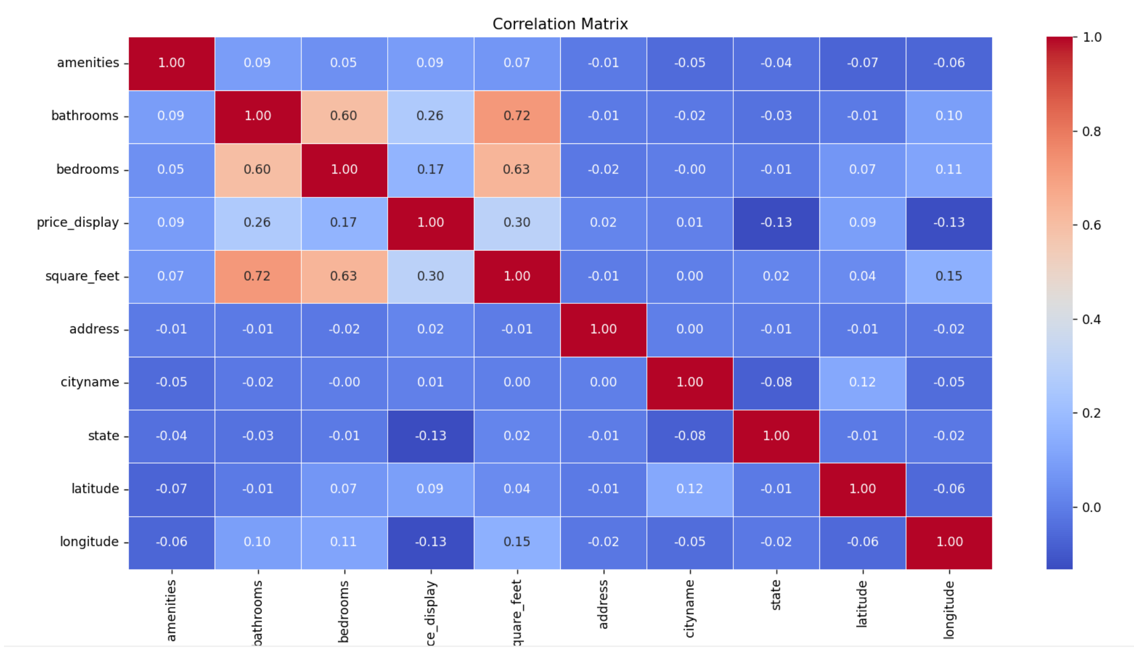


As for the columns with only a single value, we’ll have to drop them as they serve no purpose in our training process.

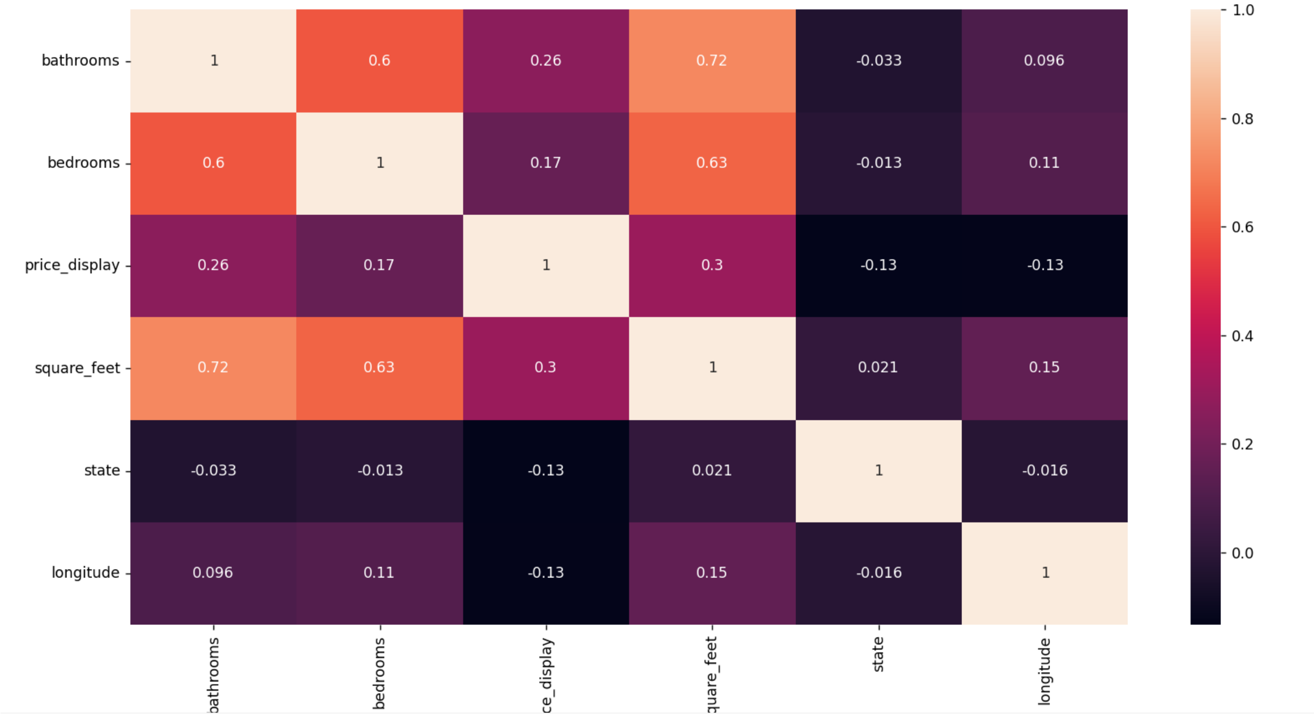
```
columns_to_drop = ['category','id','price', 'title', 'body', 'source', 'time','currency', 'fee','has_photo','price_type', 'pets_allowed']
data = data.drop(columns=columns_to_drop)
```

We’re now ready for the **Feature Selection** phase. We’ve used 2 methods:

- **Correlation**
- **ANOVA**



Based on the provided **Correlation Matrix,** we can deduce the following:



Features with correlation coefficient >±0.1

```
# ANOVA for categorical features
anova_results = f_classif(data_encoded[categorical_columns], data['price_display'])
anova_p_values = pd.Series(anova_results[1], index=categorical_columns)

# Select significant categorical features based on p-value threshold
significant_categorical_features = anova_p_values[anova_p_values < 0.05].index.tolist()

print("Significant categorical features based on ANOVA p-values:", significant_categorical_features)
```

```
Significant categorical features based on ANOVA p-values: ['amenities', 'cityname', 'state', 'address']
```

Features with ANOVA values < 0.05

Therefore, our final **selected features** for our model training are the following:

- **Bathrooms**
- **Bedrooms**
- **Square Feet**
- **State**
- **Longitude**
- **City Name**
- **Amenities**
- **Address**

Now we're clear to proceed to the next step in our pipeline, **Model Training.**

3. Model Training & Evaluation

We start our process by dividing our data into training, testing & validation sets. We'll be using k-fold cross validation in later stages, but for now let's settle on **80% - 20%** train-test distribution

```
# data splitting
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True, random_state=10)
```

Cross Validation coefficient (K) was also set to **8 Folds**

```
scores = cross_val_score(poly_model1, X_train_poly_model_1, y_train, scoring='neg_mean_squared_error', cv=8)
```

As for our model, we've seen that the selected features are **multiple,** thus it'd be logical to use multiple regression models.

<u>Models used in the training phase:</u>

- **Multiple Linear Regression**
- **Multiple Polynomial Regression**
- **Multiple Polynomial Regression with K-Fold Cross Validation**

<u>Starting off with the linear model, here's the supplied code:</u>

```
# linear model
linear_reg = linear_model.LinearRegression()
linear_reg.fit(x_train, y_train)

# model testing
y_train_prediction = linear_reg.predict(x_train)
y_predict = linear_reg.predict(x_test)
```

Given this R2 Score & MSE, we can deduce that this model isn't performing well.

Let's try the **Polynomial Model.**

```
linear model
Mean Square Error for testing 190300.6390524467
Mean Square Error for training 183064.20787404975
r2 score: 0.1650954195630745
```

Polynomial Model Code & Results:

```
# polynomial model
poly_features = PolynomialFeatures(degree=3)
X_train_poly = poly_features.fit_transform(x_train)

poly_model = linear_model.LinearRegression()
poly_model.fit(X_train_poly, y_train)

# model testing
prediction = poly_model.predict(poly_features.fit_transform(x_test))
prediction1 = poly_model.predict(poly_features.fit_transform(x_train))

print("polynomial model")
print('Mean Square Error for testing', metrics.mean_squared_error(y_test, prediction))
print('Mean Square Error for training', metrics.mean_squared_error(y_train, prediction1))
print("r2 score:", r2_score(y_test, prediction))
```

```
polynomial model
Mean Square Error for testing 122561.48506248063
Mean Square Error for training 112140.09308372051
r2 score: 0.46228690679480267
```

This time, it's way better than the linear model, even the model complexity isn't high, which pushes away any suspicion of over fitting. R2 score is also coming up pretty nice almost hitting **0.5**

Polynomial Model with Cross Validation Code & Results:

For this model we used 2 different degrees:

- 2nd degree denoted as **model 1**
- 3rd dergree denoted as **model 2**

```python
# poly model with cross validation
print('\ncross validation')
model_1_poly_features = PolynomialFeatures(degree=2)
# transforms the existing features to higher degree features.
X_train_poly_model_1 = model_1_poly_features.fit_transform(x_train)

# fit the transformed features to Linear Regression
poly_model1 = linear_model.LinearRegression()
scores = cross_val_score(poly_model1, X_train_poly_model_1, y_train, scoring='neg_mean_squared_error', cv=8)
model_1_score = abs(scores.mean())

poly_model1.fit(X_train_poly_model_1, y_train)
print("model 1 cross validation score is " + str(model_1_score))

model_2_poly_features = PolynomialFeatures(degree=3)
# transforms the existing features to higher degree features.
X_train_poly_model_2 = model_2_poly_features.fit_transform(x_train)

# fit the transformed features to Linear Regression
poly_model2 = linear_model.LinearRegression()
scores = cross_val_score(poly_model2, X_train_poly_model_2, y_train, scoring='neg_mean_squared_error', cv=8)
model_2_score = abs(scores.mean())
poly_model2.fit(X_train_poly_model_2, y_train)

print("model 2 cross validation score is " + str(model_2_score))

# predicting on test data-set
prediction = poly_model1.predict(model_1_poly_features.fit_transform(x_test))
print('\nModel 1 Test Mean Square Error', metrics.mean_squared_error(y_test, prediction))
print("r2 score:", r2_score(y_test, prediction))

# predicting on test data-set
prediction = poly_model2.predict(model_2_poly_features.fit_transform(x_test))
print('Model 2 Test Mean Square Error', metrics.mean_squared_error(y_test, prediction))
print("r2 score:", r2_score(y_test, prediction))
```

```
cross validation
model 1 cross validation score is 129438.22845359438
model 2 cross validation score is 121211.21945224862

Model 1 Test Mean Square Error 131313.15597955053
r2 score: 0.4238907659750939
Model 2 Test Mean Square Error 122561.48506248063
r2 score: 0.46228690679480267
```

In Conclusion:

To conclude our process, we observe that **Polynomial Model in 3rd degree** with **8-Fold Cross Validation** retains the best R2 Score and MSE

Classification Model

For our second phase of this project, our data has changed shape has changed as a new column has been added "Rent Category", and "Price" column has been removed. Our target is to classify our test data into one of 3 classes.

1. Preprocessing & Feature Engineering

Since we're dealing with mostly the same data, there will be no additional steps than what we discussed previously. But we'll do a certain strategy to improve our model accuracy. We'll be splitting our data into train/test before preprocessing, store the encoding & null filling values achieved from the training set, and then use them on our test set. Therefore, we try to avoid any form of interference of our test data in our model training.

```python
ordinalE = OrdinalEncoder()
data['RentCategory'] = ordinalE.fit_transform(data[['RentCategory']])
Y = data['RentCategory']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, shuffle=True, random_state=10)

# print(X_train.isna().sum())

bedroom_mean = X_train['bedrooms'].mean()
bathroom_mean = X_train['bathrooms'].mean()
cityname_mode = X_train['cityname'].mode()[0]
state_mode = X_train['state'].mode()[0]
lat_mean = X_train['latitude'].mean()
long_mean = X_train['longitude'].mean()
pets_mode = X_train['pets_allowed'].mode()[0]
amenities_mode = X_train["amenities"].mode()[0]

X_train["amenities"].fillna(amenities_mode, inplace=True)
X_train['bathrooms'].fillna(bathroom_mean, inplace=True)
X_train['bedrooms'].fillna(bedroom_mean, inplace=True)
X_train['pets_allowed'].fillna(pets_mode, inplace=True)
X_train['cityname'].fillna(cityname_mode, inplace=True)
X_train['state'].fillna(state_mode, inplace=True)
X_train['latitude'].fillna(lat_mean, inplace=True)
X_train['longitude'].fillna(long_mean, inplace=True)
```

```python
X_test["amenities"].fillna(amenities_mode, inplace=True)
X_test['bathrooms'].fillna(bathroom_mean, inplace=True)
X_test['bedrooms'].fillna(bedroom_mean, inplace=True)
X_test['pets_allowed'].fillna(pets_mode, inplace=True)
X_test['cityname'].fillna(cityname_mode, inplace=True)
X_test['state'].fillna(state_mode, inplace=True)
X_test['latitude'].fillna(lat_mean, inplace=True)
X_test['longitude'].fillna(long_mean, inplace=True)


# Fill missing address with city and state name
X_test['address'] = X_test.apply(lambda row: f"{row['cityname']}, {row['state']}" if pd.isnull(row['address']) else row

# print(X_test.isna().sum())

X_test[categorical_columns] = ordinal_Encoder.transform(X_test[categorical_columns])
```

```python
# Fill missing address with city and state name
X_train['address'] = X_train.apply(lambda row: f"{row['cityname']}, {row['state']}" if pd.isnull(row['address']) else row['address'], axis=1)
```

Notice how we've also adopted a new approach by injecting the address NaN with artificial data

As for Feature Selection, we've decided to adhere with filter methods once again, as wrapper methods expressed a great challenge to achieve high accuracy with. However, since our target variable this time is categorical, we've settled on **Chi Squared** for our categorical features, as well as **ANOVA** for our numerical ones. The following code represents the process.

Hyperparameters including ANOVA **p-value = 0.05** & Chi Squared **k = 4** have been tuned to match the highest results

```
# ANOVA for numerical features
anova_results = f_classif(train_Data[numerical_columns], train_Data['RentCategory'])
anova_p_values = pd.Series(anova_results[1], index=numerical_columns)

significant_numerical_features = anova_p_values[anova_p_values < 0.05].index.tolist()

# print("Significant numerical features based on ANOVA p-values:", significant_numerical_features)

cate = train_Data[categorical_columns]

# Apply Chi-squared test
chi2_selector = SelectKBest(chi2, k=4)
chi2_selector.fit(cate, train_Data['RentCategory'])

# Get the scores for each feature
chi_scores = pd.DataFrame({
    'Feature': categorical_columns,
    'Score': chi2_selector.scores_
}).sort_values(by='Score', ascending=False)

# print(chi_scores)

best_features_indices = chi2_selector.get_support(indices=True)
best_features_names = [cate.columns[i] for i in best_features_indices]


Y_train = train_Data['RentCategory']
X_train = train_Data[list(best_features_names) + list(significant_numerical_features)]
# X_train.head()
```

We'll be saving the values of those features in our test script in order for our model to efficiently predict the results.

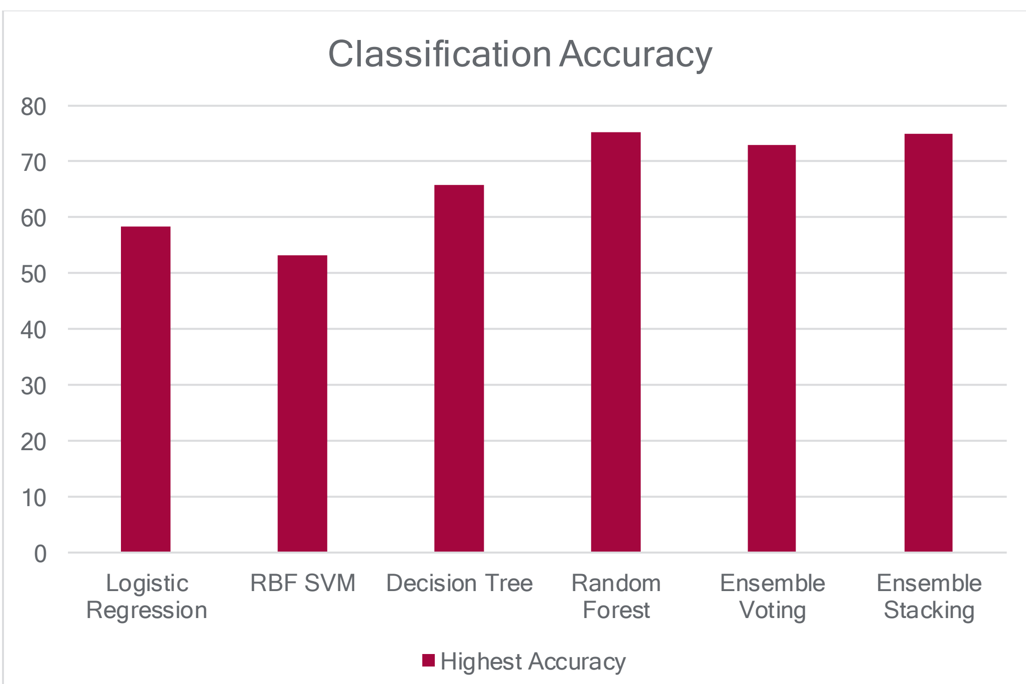
	amenities	cityname	state	address	bathrooms	bedrooms	square_feet	latitude	longitude
5250	1675.0	57.0	42.0	4810.0	1.0	3.0	516.0	30.3054	-97.7497
5018	1700.0	49.0	10.0	1468.0	1.0	1.0	675.0	33.8077	-84.3753
5453	1017.0	1120.0	20.0	5317.0	1.0	1.0	812.0	39.0650	-76.9815

Snapshot of training dataset with the selected features just before training our models

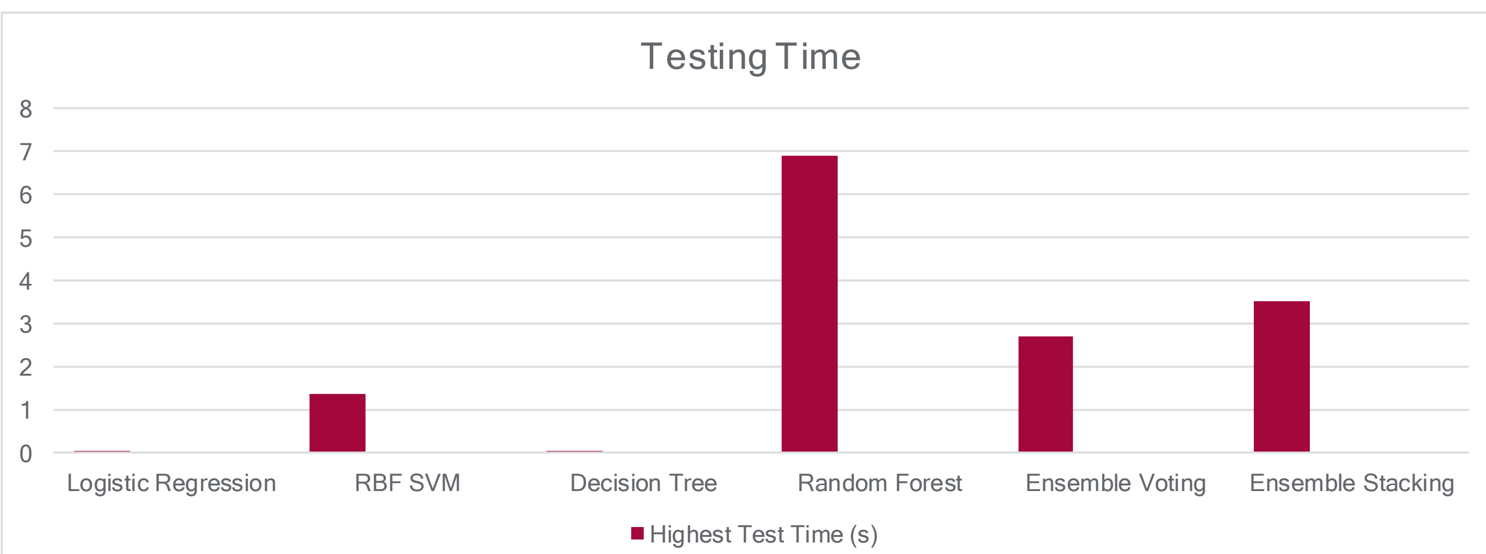
2. Model Training & Evaluation

We've trained 4 different models: Multinomial Logistic Regression, RBF SVM, Decision Tree, and Random Forest. We've also experimented with Ensemble Learning techniques involving voting, as well as stacking. Below is the classification accuracy and training & testing time bar charts

Algorithm	Highest Accuracy
Logistic Regression	58.3
RBF SVM	53.1
Decision Tree	65.8
Random Forest	75.3
Ensemble Voting	73
Ensemble Stacking	75



Algorithm	Highest Train Time (s)	Highest Test Time (s)
Logistic Regression	3.057	0.001
RBF SVM	40	1.36
Decision Tree	2.5	0.002
Random Forest	292.3	6.9
Ensemble Voting	16.8	2.7
Ensemble Stacking	74.4	3.5



3. Hyperparameter Tuning

Logistic Regression

The logistic regression model is trained with different solvers (lbfgs, saga, newton-cg) and as well different iteration settings. (1000, 100K, 1M, 5M)

We've also decided to manually try out the different combinations as instructed in the document. Further models will utilize Grid Searching

The highest accuracy achieved is around 58% with newton-cg solver and 1k iterations.

Random Forest

Initially trained with default hyperparameters, achieving an accuracy of approximately 75%.

Hyperparameter tuning using GridSearchCV improves accuracy to around 77%.

Best parameters found include n_estimators: 5000, max_depth: None, min_samples_leaf: 1, and min_samples_split: 5.

Support Vector Machine (SVM)

The initial SVM model achieves an accuracy of approximately 53%.

Hyperparameter tuning using GridSearchCV does not significantly improve accuracy.

Decision Tree

Initially trained with default hyperparameters, achieving an accuracy of around 61%.

Hyperparameter tuning using GridSearchCV improves accuracy to approximately 67%.

Best parameters found include max_depth: 10, min_samples_leaf: 1, and min_samples_split: 2.

Comparison of Models

Random Forest achieves the highest accuracy among all classifiers, followed by the Stacking Classifier.

Logistic Regression and SVM show comparatively lower accuracies.

Ensemble methods, such as Voting Classifier and Stacking Classifier, improve accuracy compared to individual classifiers.

Conclusion

The code reflects the iterative nature of machine learning model development. It begins with data preprocessing steps to ensure data quality and consistency. Then, feature selection techniques are employed to identify the most relevant features for predictive modeling. Subsequently, various classifiers are trained and evaluated, providing insights into their individual performance.

Hyperparameter tuning emerges as a critical step in enhancing model performance. By systematically searching through a range of hyperparameter values, the code identifies configurations that maximize model accuracy. This process highlights the importance of fine-tuning model parameters to achieve optimal results.

Furthermore, the inclusion of ensemble methods, such as the Voting Classifier and Stacking Classifier, underscores the effectiveness of combining multiple models to improve predictive performance. These ensemble techniques leverage the strengths of individual classifiers, leading to enhanced accuracy compared to standalone models.

Thank You,

- **Khaled Ayman Salah – 2021170174**
- **Jana Karim Saleh - 2021170141**
- **Mohammad Hani Mohammad – 2021170491**
- **Amera Abdelaziz Saber - 2021170092**
- **Yosif Sayed Mahmoud – 2021170633**
- **Mostafa Saeed Abdelfadeel – 2021170522**