Data Science Project

Property Rent Calculator

Introduction

The project objectives were to work on a complete Data Science project, with all the steps needed:

- **Collect data** from scratch, from real estate listing websites.
- Use **data cleansing** and **data manipulation** techniques
- Use clean data to **train a model.**
- Using the model to **predict property rent values**
- **Deploy** this model to be accessible as a web application.

The aim was to predict rent values from Brasília/DF, Brazil. Data was collected from a website that lists real estate specifically from this city, as it was the website with the larger list length found.

Features were collected from the collected data, using data cleanse and manipulation techniques. Then a regressor algorithm was used to learn from this data, to then be able to predict values. At last, the model was deployed on Heroku cloud platform.

Data Collection

To start, a Selenium Python script was made, to be used to collect real estate data from the web. Selenium is primarily used for automated testing on web applications. Using it involves some knowledge of HTML, as it needs to know the data paths on the website.

The website was then inspected, using the native browser inspection tool, and the paths needed were collected. Using these paths, and automating a loop to repeat the search for every page, data was collected one page at a time. A random wait time was used between pages, to prevent the script from being blocked. Try and except were also included in the script, as some properties won't have some data fields (for example the number of suites info), so the script would not stop because of an error when not finding it.

```
# funcion to collect property info
def get_property( i ):
    try:
        title = driver.find_element(By.XPATH, property_path + i + property_title_path)
        all_title.append(title.text)
    except:
        all_title.append('')
    try:
        subtile = driver.find_element(By.XPATH, property_path + i + property_subtitle_path)
        all_subtitle.append(subtile.text)
    except:
        all_subtitle.append('')
```

```
for j  in range( 2 , number_pages + 1):
    # wait time between pages
    time.sleep( wait_time )
    # going to each page
    driver.get( url + str(j) )
    # repeating first page process
    property_list = driver.find_elements(By.CLASS_NAME, class_to_search )
    number_elements = len(property_list)
    print('number of element in page {} is {}'.format(j, number_elements) )
    for i in range( 1 , number_elements + 1):
        get_property( str(i) )
```

The collected data was organized in a Pandas data frame, and then it was exported as a CSV file. The script and the exported data are available in the folder related to this step, on the project GitHub repository.

Data Cleanse

In this second step, the data was analyzed using a Python Jupyter Notebook. Pandas and NumPy libraries were used. To start, columns were renamed for easier visualization. The number of null values on each column was checked, and also the number of unique values.

The “Type and Size” field was used to collect the property type, which would become a string column. The size was extracted also, to become an int column. After extracting these columns, the original column was dropped. The bedrooms, suites, and parking columns were also treated, extracting only the digits from them. The neighborhood name was extracted from the title column, which is a very important feature for this model. After this, the title was dropped, and also some more columns would not be needed anymore.

3298	1.800	58	31.0	Apartamento	2	1	NaN	Avenida Parque Águas Claras, NORTE, AGUAS CLARAS	NaN	Excelente apartamento no Ed. Portal do Parque,...	Avenida Parque Águas Claras, NORTE, AGUAS CLARAS
2294	3.900	120	32.0	Apartamento	3	1	NaN	SQN 108 Bloco F, ASA NORTE, BRASILIA	NaN	IMÓVEL EM PROCESSO DE DESOCUPAÇÃO PREVISÃO PAR...	SQN 108 Bloco F, ASA NORTE

Extracting the neighborhood name.

```
df['neighborhood'] = df['neighborhood'].str.extract(r'([^,]*)$', expand=False)
```

```
df.sample(5)
```

	rent	size_m2	rent_per_m2	property_type	bedrooms	suite	parking	title	subtitle	description	neighborhood
3086	2.300	62	37.0	Apartamento	2	NaN	NaN	QRSW 6 Bloco B 3, SUDOESTE, BRASILIA	NaN	IMÓVEL EM PROCESSO DE DESOCUPAÇÃO PREVISÃO PAR...	SUDOESTE
1883	1.590	45	35.0	Apartamento	1	NaN	NaN	AOS 01, OCTOGONAL, BRASILIA	NaN	OCTOGONAL - Apartamento com 1/4 com 45,00 m², ...	OCTOGONAL

Then a new CSV was exported from this Notebook. The notebook used, and the exported CSV, are available in the folder related to this step, on the project GitHub repository.

Data Analysis

In this third step, the clean data was further analyzed, to later be used to train a machine learning algorithm. A new Jupyter Notebook was used on this step, reading the new CSV created in the last step. Besides Pandas and NumPy, Scikit-learn is the machine learning library used here. The notebook used in this step also contains the model training (next step after data analysis).

The column's data types were checked. The rent value column was transformed into an int column. Some strange values rows were checked, for example, 0 and 1 m² properties. These rows were dropped, as also rows with extremely high values.

```
In [7]: df.rent.value_counts()
```

```
Out[7]: 1.100     171
        1.200     146
        1.000     143
        800       128
        1.500     126
                 ...
        12.800      1
        2.510       1
        775         1
        4.999       1
        520         1
        Name: rent, Length: 325, dtype: int64
```

```
In [8]: df['rent'] = df['rent'].str.replace('.', '', regex=False)
```

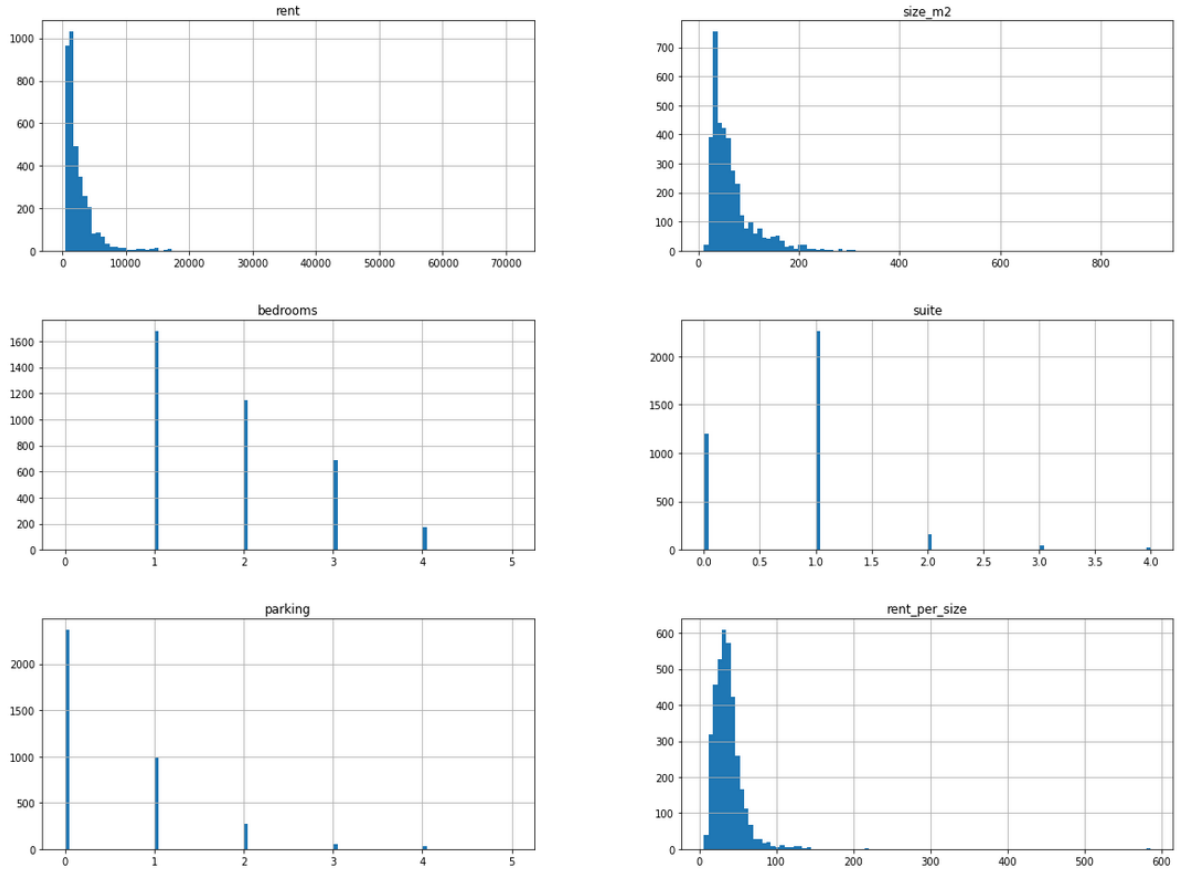
```
In [9]: df.rent.value_counts()
```

```
Out[9]: 1100     171
        1200     146
        1000     143
        800      128
        1500     126
                ...
        12800      1
        2510       1
        775        1
        4999       1
        520        1
        Name: rent, Length: 325, dtype: int64
```

```
In [10]: df['rent'] = df.rent.astype('int32')
```

A new column "rent per size" was created, to check for any rows with strange rent values, or some more properties with too high sizes. Also, the bedrooms, suites, and parking columns were treated. Some rows with strange numbers on these columns were dropped.

Using some graph visualizations, the histogram of numerical columns was checked. It is an important step, as the scaler choice that will be made later involves knowing how our data is distributed.

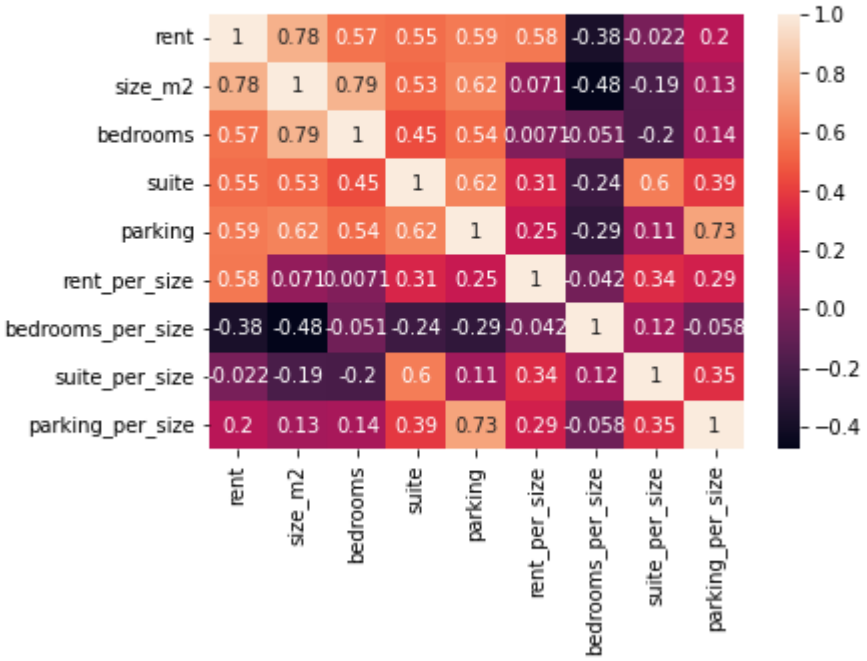


Also using the “rent per size” column to help, the neighborhood column was treated. As we had some neighborhoods with very few rows, they were grouped into some tiers, with similar rent per size values. In the end, 6 data tiers were created. Only one of them still remains with a low number of rows (14 rows, from 3700 total), but unfortunately, they can’t be grouped with other neighborhoods of similar rent per size values, because they are neighborhoods with very distinct characteristics from the others.

Data was then divided for training and testing. As we don’t have a large number of rows in total, an 80/20 split was chosen. Some combined columns were created, “bedrooms per size”, “suites per size”, and “parking per size”. The correlation between all the numerical columns was then checked, using the help of a Seaborn heatmap visualization.

```
sns.heatmap(df_strat_train.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93c16d0f10>
```



Then the “rent per size” column was dropped, and the “rent” column was separated from the rest. It would now become our “y” column. The other columns would be our features.

Our categorical columns were transformed into boolean columns, using the Pandas “get dummies” method. Also, the numerical columns were scaled, using the Robust Scaler class, as it would work better with our data, as the histograms showed left skewed data. A boxplot visualization was used to compare data before and after scaling. The test data was also scaled, using the same scaler fitted with the training data.

Model Training

Finally, using the prepared data, model training was started. As a reference, 3 types of models were tested straight out of the box, using default parameters:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor

Trey were trained using the x and y training sets, and their prediction error was measured using the “mean squared error”. They were also used to predict the y from the x test set, and the error was again compared.

As expected, the linear regression model did not perform well, as we have very different characteristics from the distinct neighborhoods.

The decision tree showed to learn well the training set, show a very low error when predicting training data. But when predicting the test data, it performs worse than the linear model. This shows us it has learned too well the training data and is not generalizing well, being overfitted.

```
tree_reg = DecisionTreeRegressor()
```

```
tree_reg.fit(x_train, y_train)
```

```
DecisionTreeRegressor()
```

```
np.sqrt(mean_squared_error(y_train.values,
                           tree_reg.predict(x_train)))
```

```
286.7492765148164
```

```
np.sqrt(mean_squared_error(y_test.values,
                           tree_reg.predict(x_test)))
```

```
1069.698634072441
```

The random forest model performs better than the two previous models, being just slightly overfitted, as it had lower training data error than on test data.

Then the models were tested again using cross-validation. This proved to be very effective, as now all models had a higher error when predicting training data, which shows us that they are not memorizing the training data. And also they show now better results with the test data, showing that they are generalizing better now.

```
scores = cross_val_score(tree_reg,
                         x_train,
                         y_train,
                         scoring='neg_mean_squared_error',
                         cv=5)
```

```
tree_rmse_scores = np.sqrt(-scores)
```

```
np.mean(tree_rmse_scores)
```

```
1528.8172517377348
```

```
np.std(tree_rmse_scores)
```

```
778.8652330387204
```

After these initial tests, the Random Forest model was further explored, with the help of the Grid Search function. This allows us to test all the combinations of parameters we want, training and testing the model multiple times, while also using cross-validation. The function then returns to us the best combination of parameters found, also being able to show the results of each combination tested.

```
grid_search.best_params_
```

```
{'max_depth': 12, 'max_features': 10, 'n_estimators': 80}
```

```
np.sqrt(-grid_search.best_score_)
```

```
1587.3825102050653
```

This fine-tuned trained model was then extracted and saved to a file. Then it can be later loaded and used. The extracted model, the extracted scaler, and the notebook used to analyze the data and train the model, are available in the folder related to this step, on the project GitHub repository.

```
from joblib import dump, load

dump(rf_reg_v3, 'property-rent-calc.joblib')
```

```
['property-rent-calc.joblib']
```

```
rf_test = load('property-rent-calc.joblib')
```

```
np.sqrt(mean_squared_error(y_train,
                           rf_test.predict(x_train)))
```

```
719.855020866176
```

```
np.sqrt(mean_squared_error(y_test,
                           rf_test.predict(x_test)))
```

```
796.9233471123395
```

Model Deployment

The last step of this project is related to deploying the model, in order for it be available to be accessed as a web application, using the Streamlit framework.

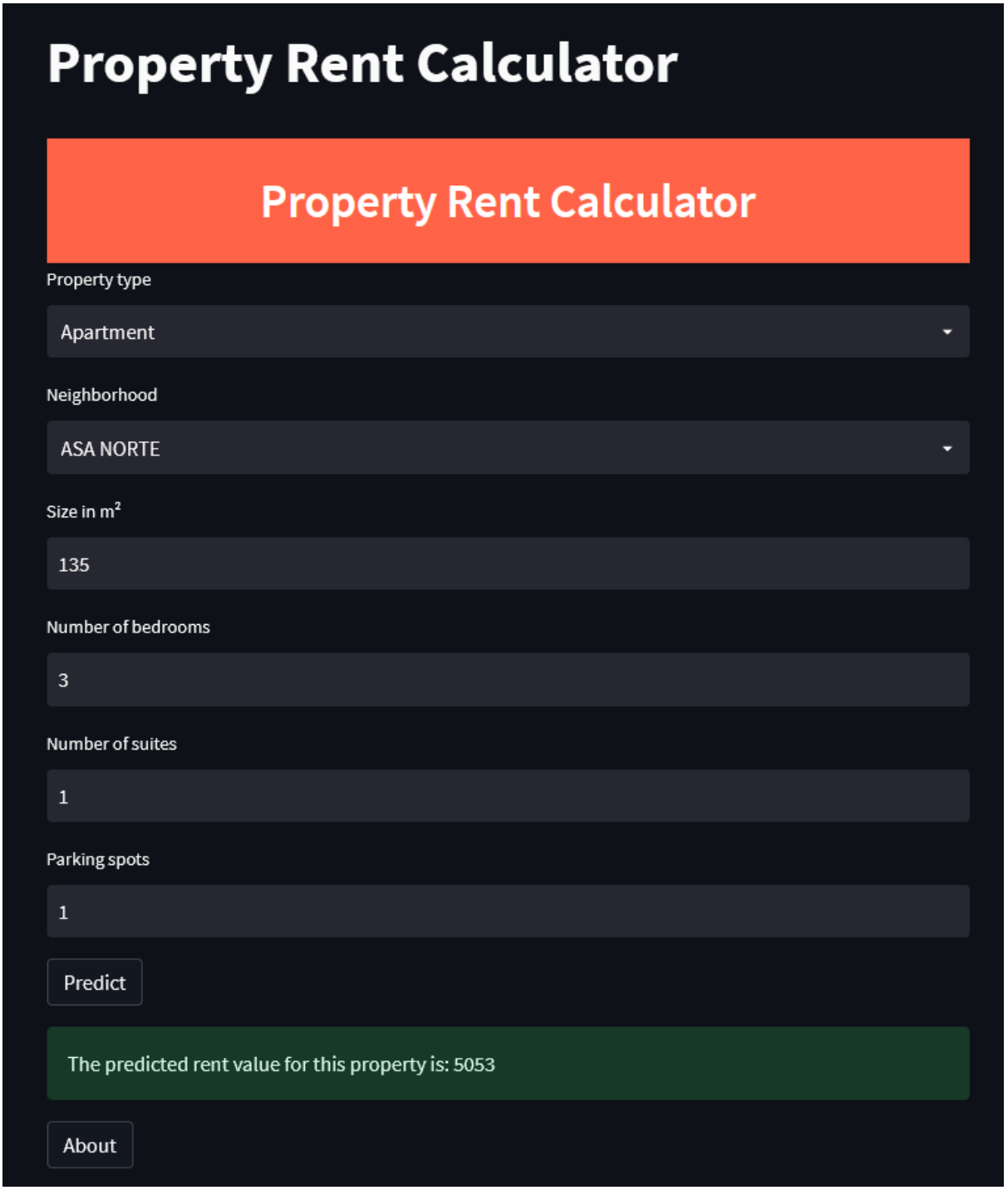
The Python script used here imports the trained model and the scaler from the previous step. The scaler will be used on new imputed data, to prepare the new data, and then the model is used to predict the rent value from this new property.

A pipeline is created here, in order to prepare the new data in the same way it was done for training and testing data. New combined columns are created, and property type and neighborhood are again informed on boolean columns. And, at last, numerical columns are scaled using the same scaler fitted with the training data. After the new data goes through all these steps, the model predicts rent value and returns on the screen interface.

Selection boxes were used for property types and neighborhoods, to prevent inserting data that the model is not prepared to work. Size in m², bedrooms, suites, and parking are imputed as numbers.

Further work could be done on this project, by hosting this Streamlit Script on a cloud virtual machine, so this calculator would be available via the web. But for learning purposes, this project will end up here.

Here we can see what the interface made with Streamlit looks like, running it locally:



Conclusions

This project shows a walkthrough of how a data science project can be made from scratch, showing a real-life application of data science concepts. It shows just the tip of the iceberg, and every step done here can be further worked in order to achieve better results.

The feature extraction from the original data could be improved, by working on the description field from each property. The address on the title could also be worked to give more data for the model to learn. The neighborhood tier segmentation could also easily be improved. More regression models could be tested. These are just examples of how this project would be further developed if it were a real application. Hosting the framework on a cloud virtual machine would be the next step to be done.