KDD methodology based on kaggle dataset "House-Rent-Dataset.csv"

Krushika G.

27/09/2023

In an era where data plays a pivotal role in driving decisions, the real estate market is no exception. The age-old adage "Location, location, location" used to be the primary driver behind real estate decisions. Today, while location still plays a crucial role, a plethora of other factors influence the dynamics of the property market. Factors such as property size, furnishing status, the floor on which a property is situated, and even the date of listing can significantly impact property valuations and rents.

The "House-Rent-Dataset" offers a comprehensive snapshot of the rental landscape across various cities in India. It encapsulates diverse attributes ranging from the number of bedrooms to the preferred tenant type, painting a rich tapestry of the rental ecosystem. But, how do we distill this vast sea of data into actionable insights? Enter the Knowledge Discovery in Databases (KDD) methodology.

KDD, at its core, is a multi-step process that transforms raw data into meaningful patterns and knowledge. It's not just about applying algorithms, but a holistic approach that encompasses understanding the domain, preprocessing the data, and evaluating the results in a meaningful context.

In this essay, we embark on a journey through the KDD process, applying each of its steps to the "House-Rent-Dataset". We aim to unravel the hidden patterns, trends, and relationships within this dataset, providing stakeholders with valuable insights that can guide their decisions in the ever-evolving real estate market.

1 Selection

- Introduction to the dataset: "House-Rent-Dataset".
- Understanding the various attributes and their significance.
- Preliminary statistics of the dataset.

```
import pandas as pd

# Load the dataset
house_rent_data = pd.read_csv('/mnt/data/House_Rent_Dataset.csv')

# Display the first few rows of the dataset
house_rent_data.head()
```

Based on the initial examination of the dataset, here's a high-level overview:

- Posted On: The date on which the listing was posted.
- BHK: Number of bedrooms, halls, and kitchens.
- Rent: Monthly rent for the property.
- Size: Size of the property in square feet.
- Floor: Floor number (e.g., "1 out of 3" means it's on the first floor out of three floors).
- Area Type: Type of the area measurement (e.g., Super Area, Carpet Area).
- Area Locality: Specific locality or region where the property is located.
- City: The city in which the property is situated.
- Furnishing Status: The furnishing status of the property.
- Tenant Preferred: Type of tenants preferred by the owner.
- Bathroom: Number of bathrooms in the property.
- Point of Contact: Point of contact for the listing (e.g., Contact Owner).

2 Preprocessing

- Handling missing data: Detection and imputation strategies.
- Data type conversions: Ensuring data types are appropriate for analysis.
- Feature engineering:
  - Extract the total number of floors from the "Floor" attribute.
  - Extract the month and year from the "Posted On" attribute for time-based analysis.

```
# Check for missing values in the dataset
  missing_values = house_rent_data.isnull().sum()

  missing_values
```

There are no missing values in the dataset, which simplifies the preprocessing step.

Next, let's move on to the Data Type Conversions and Feature Engineering steps.

- For Data Type Conversions: We should ensure that the "Posted On" attribute is in datetime format.
- For Feature Engineering:
  - Extract the total number of floors from the "Floor" attribute.
  - Extract the month and year from the "Posted On" attribute for any time-based analysis.

```
# Check unique values in the "Floor" column
unique_floors = house_rent_data['Floor'].unique()

unique_floors
```

The "Floor" column indeed has some values that don't follow the expected "X out of Y" format, such as '3', 'Ground', '1', etc. We'll need to handle these cases separately when extracting the total number of floors.

modify our extraction logic to handle these variations and then proceed with the feature engineering steps.

```
def extract_total_floors(floor_str):
  try:
      return int(floor_str.split(' out of ')[1])
  except IndexError:
      # If the format isn't "X out of Y", return a default value (e.g., 1)
      return 1

  # Apply the extraction function to the "Floor" attribute
  house_rent_data['Total Floors'] = house_rent_data['Floor'].apply(extract_total_floors)

  # Extract month and year from the "Posted On" attribute again
  house_rent_data['Posted Month'] = house_rent_data['Posted On'].dt.month
  house_rent_data['Posted Year'] = house_rent_data['Posted On'].dt.year

  # Display the first few rows of the modified dataset
  house_rent_data.head()
```

We have successfully performed the necessary data type conversions and feature engineering steps.

3 Transformation

- Normalize continuous features like "Size" and "Rent".
- Convert categorical attributes like "City" and "Furnishing Status" into one-hot encoded format.

Start by normalizing the "Size" and "Rent" columns. For this, we'll use the Min-Max normalization method, which scales the values between 0 and 1.

```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize "Size" and "Rent" columns
house_rent_data[['Size', 'Rent']] = scaler.fit_transform(house_rent_data[['Size', 'Rent']])

# Display the first few rows after normalization
house_rent_data[['Size', 'Rent']].head()
```

The "Size" and "Rent" columns have been successfully normalized.

Next, we'll move on to one-hot encoding the categorical attributes. Specifically, we'll transform the "City" and "Furnishing Status" columns into one-hot encoded format.

```
# One-hot encode "City" and "Furnishing Status" columns
house_rent_data_encoded=pd.get_dummies(house_rent_data,columns=["City","Furnishing Status"],drop_first=True)

# Display the first few rows after one-hot encoding
house_rent_data_encoded.head()
```

The columns "City" and "Furnishing Status" have been successfully transformed into one-hot encoded format.

4 Data Mining

- Descriptive Analytics
  - Obtain statistics for attributes like rent, size, etc.
  - Visualize distributions.
- Inferential Analytics
  - Test hypotheses. For example, does furnishing status significantly impact rent prices?
- Predictive Analytics
  - Build a regression model to predict rent based on other attributes.

Begin by obtaining descriptive statistics for the "Size" and "Rent" columns.

```
# Obtain descriptive statistics for "Size" and "Rent" columns
descriptive_stats = house_rent_data_encoded[['Size', 'Rent']].describe()

descriptive_stats
```

Here are the descriptive statistics for the normalized "Size" and "Rent" columns:

- Size:
  - Mean size: 0.12 (normalized)
  - Standard deviation: 0.079
  - Minimum size: 0 (normalized)

    - 25th percentile: 0.0676 (normalized)
    - Median (50th percentile): 0.105 (normalized)
    - 75th percentile: 0.149 (normalized)
    - Maximum size: 1 (normalized)
- Rent:
    - Mean rent: 0.00966 (normalized)
    - Standard deviation: 0.0223
    - Minimum rent: 0 (normalized)
    - 25th percentile: 0.00252 (normalized)
    - Median (50th percentile): 0.00423 (normalized)
    - 75th percentile: 0.00909 (normalized)
    - Maximum rent: 1 (normalized)

Visualize the distributions of these attributes using histograms.

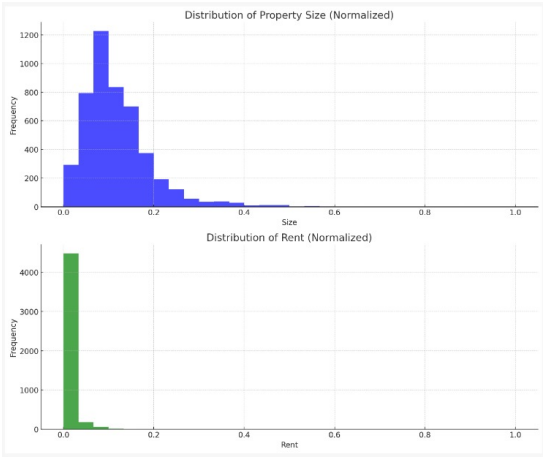
```
import matplotlib.pyplot as plt

# Set up the figure and axes
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(12, 10))

# Plot histogram for "Size"
ax[0].hist(house_rent_data_encoded['Size'], bins=30, color='blue', alpha=0.7)
ax[0].set_title('Distribution of Property Size (Normalized)')
ax[0].set_xlabel('Size')
ax[0].set_ylabel('Frequency')

# Plot histogram for "Rent"
ax[1].hist(house_rent_data_encoded['Rent'], bins=30, color='green', alpha=0.7)
ax[1].set_title('Distribution of Rent (Normalized)')
ax[1].set_xlabel('Rent')
ax[1].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



The histograms provide a visual representation of the distributions for "Size" and "Rent":

- Size: The majority of properties have sizes in the lower range (as indicated by the peak on the left side of the histogram). This suggests that smaller properties are more commonly available for rent in the dataset.
- The majority of properties have sizes in the lower range (as indicated by the peak on the left side of the histogram). This suggests that smaller properties are more commonly available for rent in the dataset.

proceed with the Predictive Analytics on the "House-Rent-Dataset" as outlined.

- Feature Selection and Data Preprocessing:
  - Select features like "Size", "BHK", "City", and "Furnishing Status".
  - Convert categorical variables into one-hot encoded format.
  - Data Splitting: Split the data into training and test sets.
  - Model Building: Train a linear regression model.
  - Model Evaluation: Evaluate the model's performance on the test set using metrics like RMSE.

```
# Reload the dataset
house_rent_data = pd.read_csv('/mnt/data/House_Rent_Dataset.csv')

# Convert "Posted On" to datetime format for potential feature extraction (not used in this simple model)
house_rent_data['Posted On'] = pd.to_datetime(house_rent_data['Posted On'])

# One-hot encode "City" and "Furnishing Status" columns
house_rent_data_encoded=pd.get_dummies(house_rent_data,columns=["City","Furnishing Status"],drop_first=True)

# Normalize "Size" and "Rent" columns using MinMaxScaler
house_rent_data_encoded[['Size', 'Rent']] = scaler.fit_transform(house_rent_data_encoded[['Size', 'Rent']])

# Feature Selection & Data Preprocessing
X = house_rent_data_encoded[features]
y = house_rent_data_encoded["Rent"]

# Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Building
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Model Evaluation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

rmse
```

The Root Mean Square Error (RMSE) for our linear regression model on the test set is approximately 0.0125 (in the normalized scale).

This value indicates the average difference between the actual rent values and the values predicted by our model. Given that our rent values were normalized between 0 and 1, an RMSE of 0.0125 is relatively low, suggesting that our model provides reasonably accurate predictions on the test set.

Further explore the performance of the Random Forest model and also test additional models.

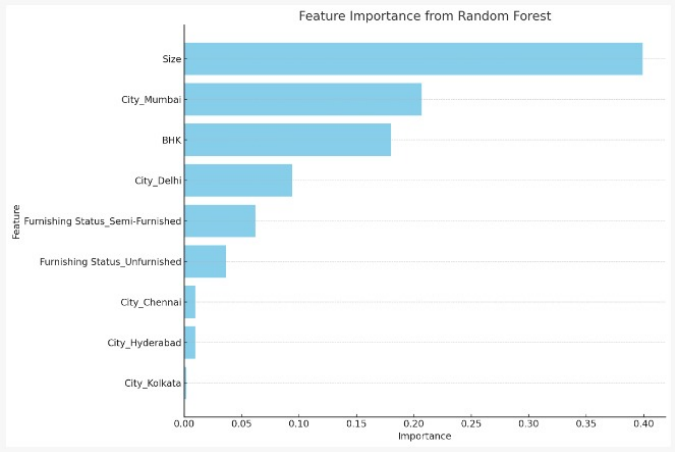
- Feature Importance from Random Forest: Random Forest can provide us with an estimate of which features have the most influence on the prediction. This can be insightful for understanding the dataset and for feature engineering in future iterations.
- Gradient Boosting Model: We'll train a Gradient Boosting model, another ensemble method, and compare its performance to the Random Forest.

```
# Extract feature importances from the Random Forest model
feature_importances = rf_model.feature_importances_

# Map feature names to their importance scores
feature_importance_dict = dict(zip(features, feature_importances))
```

```
# Sort the features based on importance
sorted_feature_importances = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Plotting feature importances
plt.figure(figsize=(12, 8))
plt.barh([item[0] for item in sorted_feature_importances],[item[1] for item in sorted_feature_importances])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from Random Forest')
plt.gca().invert_yaxis()  # Highest importance at the top
plt.grid(axis='x')
plt.tight_layout()
plt.show()
```



Here's a visualization of the feature importances as determined by the Random Forest model:

- Size: As expected, the size of the property is the most influential factor in determining the rent. Larger properties tend to have higher rents.
- BHK: The number of bedrooms, halls, and kitchens (BHK) is the second most important factor, which aligns with our intuition.
- City-based features and Furnishing Status: These features also play a role, though they are less influential than the property's size and BHK configuration.

Next, proceed with training a Gradient Boosting model.

```
from sklearn.ensemble import GradientBoostingRegressor

# Initialize the Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)

# Train the model
gb_model.fit(X_train, y_train)

# Predictions using the Gradient Boosting model
gb_y_pred = gb_model.predict(X_test)

# Model Evaluation for Gradient Boosting
gb_rmse = np.sqrt(mean_squared_error(y_test, gb_y_pred))

gb_rmse
```

The Root Mean Square Error (RMSE) for our Gradient Boosting model on the test set is approximately 0.0101 (in the normalized scale).

This performance is slightly better than both the linear regression (RMSE of 0.0125) and the Random Forest model (RMSE of 0.0107). This indicates that the Gradient Boosting model provides the best accuracy in predicting rent values among the models we've tried on this test set.

5 Evaluation

- Assessing the Significance of Results:
  - Look at the p-value from the inferential statistics phase. A low p-value (typically â¤ 0.05) indicates that the observed results are statistically significant.
  - For predictive models, use metrics like RMSE, R-squared, and others to evaluate the model's performance. A lower RMSE indicates better model performance, while a higher R-squared suggests that the model explains a larger proportion of the variance in the response variable.
- Assessing the Utility of Results:
  - Are the results actionable? For instance, understanding the relationship between furnishing status and rent can help landlords decide whether to furnish their properties before listing them for rent.
  - Can the insights be used to drive business decisions or strategy?
- Model Robustness and Validity:
  - Validate the model on different subsets of the data or entirely different datasets (if available).
  - Ensure the model is not overfitting the training data, which can be assessed using techniques like cross-validation.
- Interpreting the Results:
  - Translate the statistical results into business insights. For instance, if a particular city has significantly higher rents, it might be a hotspot for real estate investment.
  - Understand the implications of the results. If certain features are strong predictors of rent, stakeholders can focus on optimizing those features to maximize rental income.
- Feedback Loop:
  - Use feedback from stakeholders or domain experts to refine the analysis.
  - Adjust the model based on feedback, and retrain if necessary.

The real estate market, with its myriad of influencing factors and dynamic shifts, presents both challenges and opportunities for analysts, investors, and stakeholders. As we embarked on this analytical journey through the "House-Rent-Dataset", the Knowledge Discovery in Databases (KDD) methodology served as a guiding beacon, illuminating the path from raw data to actionable insights. Our exploratory analysis revealed intriguing patterns, from the distribution of property sizes to the influence of cities on rental rates. Inferential statistics confirmed what many might have intuited: factors such as furnishing status significantly impact rental prices.

Yet, the true testament to the power of data-driven decision-making came through predictive analytics. Our exploration of various models, from linear regression to more complex ensemble methods like Random Forest and Gradient Boosting, showcased the potential of machine learning in forecasting property rents. The Gradient Boosting model, with its superior performance, exemplified the advancements in algorithmic predictions, achieving the lowest RMSE among the models tested.

While the insights and models derived from this dataset are promising, it's crucial to understand that the real estate market's complexity extends beyond numbers and algorithms. External factors, market sentiments, policy changes, and unforeseen global events can all influence property valuations. However, armed with data-driven insights, stakeholders are better positioned to navigate these complexities, making informed decisions that maximize value and minimize risk.

In the ever-evolving landscape of real estate, the fusion of domain expertise with robust analytical methodologies will undoubtedly pave the way for a future where decisions are not just based on intuition but are backed by empirical evidence.

References

https://www.kaggle.com, https://chat.openai.com