P^3

PRATYUSH
PARAS
PARTH
AKSHAT

Pravartak Datathon

HOUSERATE FORECASTER

Problem Statement:

In the real estate industry, determining the appropriate rental price for a property is crucial for property owners, tenants, and property management companies. Accurate rent predictions can help landlords set competitive prices, tenants make informed rental decisions, and property management companies optimize their portfolio management.
The goal of this project is to develop a data-driven model that predicts the rental price of residential properties based on relevant features. By analyzing historical rental data and property attributes, the model aims to provide accurate and reliable rent predictions.

	lease_type	gym	lift	swimming_po	negotiable	furnishing	parking	property_size	property_age	bathroom	facing	cup_board	floor
4471	FAMILY	1	1	1	0	SEMI_FURNIS	BOTH	1250	25	2	E	2	
2797	ANYONE	0	1	0	1	SEMI_FURNIS	BOTH	1400	4	2	NE	2	
1214	FAMILY	0	1	0	0	SEMI_FURNIS	BOTH	1350	6	3	E	3	
3369	FAMILY	0	0	0	1	SEMI_FURNIS	TWO_WHEELI	600	3	1	E	1	
4586	FAMILY	0	0	0	1	SEMI_FURNIS	BOTH	1500	15	3	E	4	
3141	FAMILY	1	1	1	1	SEMI_FURNIS	BOTH	1080	0	2	E	1	
6621	ANYONE	1	1	1	1	FULLY_FURN	BOTH	1895	5	3	NE	5	
6379	ANYONE	0	0	0	1	SEMI_FURNIS	BOTH	1000	10	2	S	2	
2407	ANYONE	0	0	0	0	SEMI_FURNIS	NONE	900	10	2	S	2	
3033	ANYONE	1	1	0	1	SEMI_FURNIS	BOTH	1290	4	2	E	4	
5016	FAMILY	0	0	0	1	SEMI_FURNIS	BOTH	1200	2	2	E	4	
9562	ANYONE	1	1	0	1	FULLY_FURN	FOUR_WHEEI	1300	8	2	N	2	
1461	ANYONE	0	1	0	1	SEMI_FURNIS	BOTH	930	7	2	E	2	
3703	FAMILY	1	1	1	0	SEMI_FURNIS	FOUR_WHEEI	1418	4	2	E	2	
3747	BACHELOR	0	0	0	1	SEMI_FURNIS	TWO_WHEELI	600	10	1	E	1	
3136	FAMILY	0	0	0	1	NOT_FURNISI	TWO_WHEELI	1200	2	1	E	0	
0653	FAMILY	0	0	0	0	SEMI_FURNIS	TWO_WHEELI	1200	3	2	E	2	
4088	ANYONE	0	0	0	0	SEMI_FURNIS	TWO_WHEELI	1200	10	2	E	0	
3521	FAMILY	0	0	0	0	SEMI_FURNIS	BOTH	1020	0	2	SE	2	
2669	ANYONE	0	1	0	1	SEMI_FURNIS	BOTH	1084	10	2	E	2	
3616	FAMILY	1	1	0	1	SEMI_FURNIS	FOUR_WHEEI	1285	3	2	W	2	
5702	ANYONE	0	0	0	1	SEMI_FURNIS	NONE	600	1	1	E	2	
9061	ANYONE	0	0	0	1	NOT_FURNISI	BOTH	800	1	1	W	0	
6845	FAMILY	1	1	1	1	SEMI_FURNIS	BOTH	1610	8	2	E	6	
7345	FAMILY	0	0	0	1	SEMI_FURNIS	TWO_WHEELI	1100	10	2	N	2	
0454	ANYONE	0	0	0	1	SEMI_FURNIS	BOTH	1200	10	2	E	2	
7931	ANYONE	0	1	0	0	SEMI_FURNIS	BOTH	800	3	2	E	1	
7264	FAMILY	0	0	0	1	SEMI_FURNIS	NONE	800	12	1	E	2	
2315	FAMILY	1	1	1	1	SEMI_FURNIS	BOTH	1484	1	2	N	1	
3776	FAMILY	0	0	0	1	SEMI_FURNIS	NONE	500	10	1	N	2	
2144	ANYONE	0	1	0	0	SEMI_FURNIS	BOTH	1200	5	2	W	2	
4814	ANYONE	0	1	0	0	FULLY_FURN	BOTH	1300	8	2	S	2	
6828	BACHELOR	0	0	0	1	SEMI_FURNIS	TWO_WHEELI	250	10	1	E	1	

Tools Used:

Following tools were used for this project

```
import pandas as pd
import numpy as np
import json
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
import seaborn as sns
import matplotlib. pyplot as plt
```

```
[236] df= pd.read_excel("/content/House_Rent_Train.xlsx")
```

```
[237] #df['lease_type']= np.nan_to_num(df['lease_type'], copy=False, nan=-1)
```

```
[238] df= df.dropna()
```

DATA PREPROCESSING

```
categorical= [ 'type', 'lease_type', 'facing', 'water_supply', 'building_type', 'furnishing', 'parking']
```

Label Encoding

One hot Encoding

Why Label Encoding?

```
# Initialize the OneHotEncoder
encoder = LabelEncoder()

# Fit and transform the encoder on the categorical columns
df['type'] = encoder.fit_transform(df['type'])

df['type']
```

```
df['lease_type']= encoder.fit_transform(df['lease_type'])
```

```
df['facing']= encoder.fit_transform(df['facing'])
```

```
df['water_supply']= encoder.fit_transform(df['water_supply'])
df['building_type']= encoder.fit_transform(df['building_type'])
df['furnishing']= encoder.fit_transform(df['furnishing'])
df['parking']= encoder.fit_transform(df['parking'])
```

Handling Dictionary input:

```
# spliting the data into columns of their key value and converting into the int values from bool
amenities_dicts = [json.loads(entry) for entry in df["amenities"]]
data = pd.DataFrame(amenities_dicts)
mean_value= data.mean()
data= data.fillna(mean_value)
data= data.astype(int)
```

```
# droping the values which are added twice
df=df.drop('GYM', axis=1)
df=df.drop('LIFT', axis=1)
```

Do we need Locality?

```
# adding the splitted data to df and removing amenities from data
df = df.join(data)
df = df.drop("amenities", axis = 1)
# removing locality due to highly related to longitude and latitude
df= df.drop('locality', axis=1)
```

Feature Engineering

```
# defining new factor which is more co-related to rent values
df['basic']= df['bathroom']*df['property_size']*df['lift']
df['mid']= df['PARK']*df['gym']*df['INTERNET']
df['location']=df['longitude']*df['latitude']*df['property_size']
```

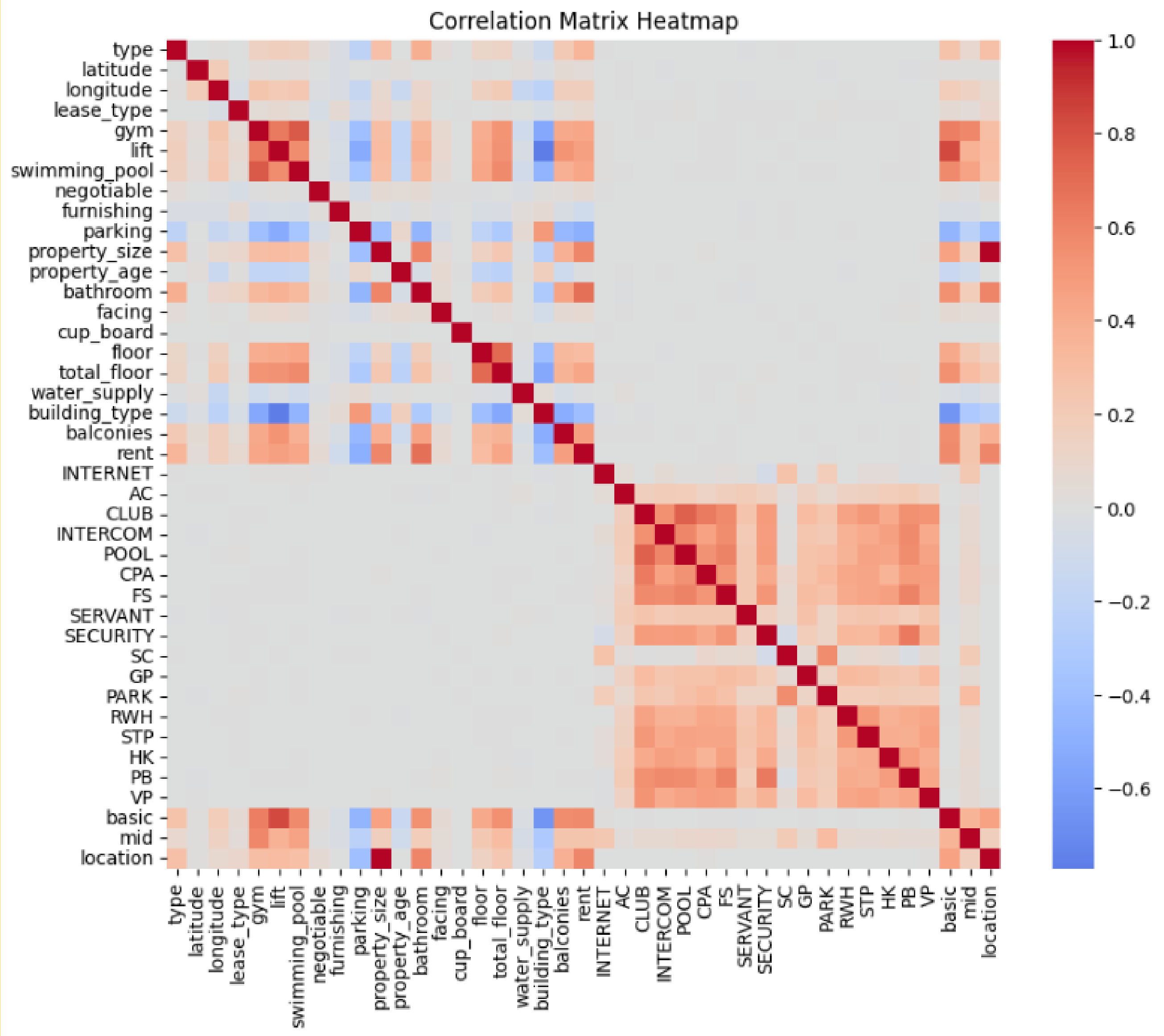
To Evaluate Features:

```
# finding co-relation between the rent and the factors
corr_matrix= df.corr()
corr_matrix['rent'].sort_values(ascending= False)
```

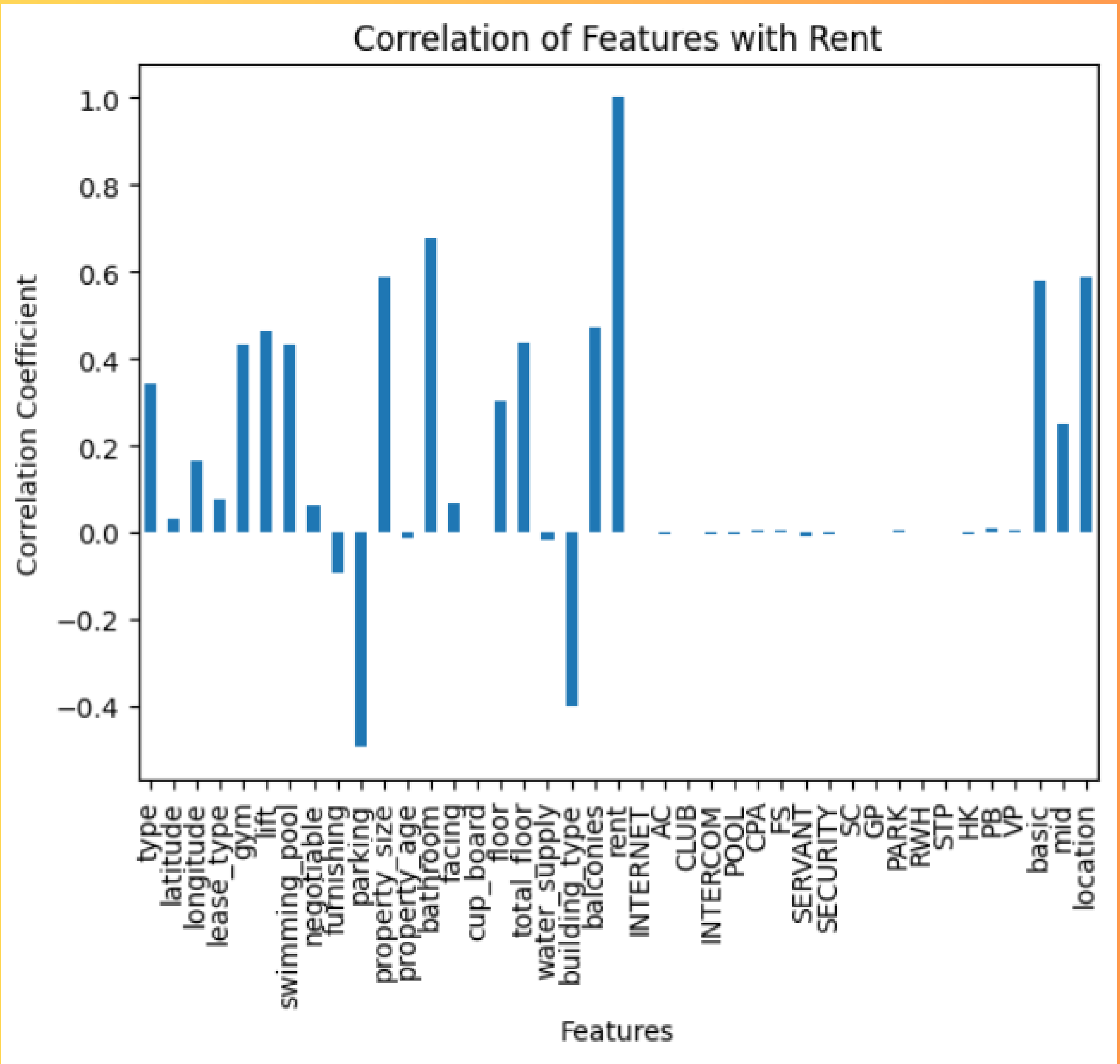
```
rent             1.000000
bathroom         0.677367
location         0.587464
property_size    0.587153
basic            0.577313
balconies        0.472521
lift             0.461769
total_floor      0.436403
swimming_pool    0.434065
gym              0.431096
type             0.344673
floor            0.305168
mid              0.250324
longitude        0.163305
lease_type       0.076220
facing           0.067047
negotiable       0.062798
latitude         0.033017
PB               0.007522
PARK             0.006613
CPA              0.005609
FS               0.004560
VP               0.002943
INTERNET         0.002732
SC               0.002297
CLUB             0.002234
RWH              0.002011
GP               0.001169
```

VISUALIZATION

```python
# finding correlation matrix
correlation_matrix = df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, cmap="coolwarm", center=0)
plt.title("Correlation Matrix Heatmap")
plt.show()
```



```
correlation_with_rent = df.corrwith(df['rent'])

# Print the correlation coefficients
print(correlation_with_rent)

correlation_with_rent.plot(kind='bar')
plt.title("Correlation of Features with Rent")
plt.xlabel("Features")
plt.ylabel("Correlation Coefficient")
plt.show()
```



Stratified Shuffle split

```
[ ] #value counts
    #type              lease_type        gym
    #2    11607        1    10113        0  15767
    #3     4368        2     9622        1  4610
    #1     4140        3      584
    #4      262        4       58
```

```
[ ] y= df['rent']
```

```
[ ] X= df.drop('rent',axis=1)
```

```
# Initialise and Prepare data
X_train, X_cv, y_train, y_cv = train_test_split(X, y, test_size=0.2, stratify=df['type'], random_state=42)
```

MODEL

Why XGBoost?

```
model = xgb.XGBRegressor(n_estimators=1000, learning_rate=0.25, random_state=42)
model.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.25, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

Hyper Parameter Optimization

```python
#Random Grid Search
from sklearn.model_selection import RandomizedSearchCV
```

```python
param_grid = {
    'n_estimators': np.arange(50, 501, 50),
    'learning_rate': [0.01, 0.1, 0.2, 0.3, 0.4],
    'max_depth': np.arange(3, 11),
    'min_child_weight': np.arange(1, 11),
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2, 0.3, 0.4],
    'reg_alpha': [0, 0.1, 0.5, 1.0],
   # 'reg_lambda': [0, 0.1, 0.5, 1.0]
}
```

```python
xgb_regressor = xgb.XGBRegressor(random_state=42, objective='reg:squarederror')
random_search = RandomizedSearchCV(
    xgb_regressor,
    param_distributions=param_grid,
    n_iter=20,  # Number of random combinations to try
    scoring='neg_mean_squared_error',  # Evaluation metric
    cv=5,  # Number of cross-validation folds
    verbose=1,
    n_jobs=-1,  # Use all available CPU cores
    random_state=42
)

random_search.fit(X_train, y_train)
```

```
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'subsample': 0.9, 'reg_alpha': 1.0, 'n_estimators': 450, 'min_child_weight':
Best Score: -12454422.590882456
```

RMSE using best params

```
[180] model = xgb.XGBRegressor(n_estimators=450, max_depth=4, learning_rate=0.1,reg_alpha=1.0, colsample_byt
      model.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.6, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.3, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=6, missing=nan, monotone_constraints=None,
             n_estimators=450, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

```
[270] # Calculate Root Mean Squared Error
      y_pred = model.predict(X_cv)
      mse = mean_squared_error(y_cv, y_pred)
      rmse= mse**0.5
      print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 3776.1769760881343
```

Test Data

PreProcessing

```
encoder = LabelEncoder()

# Fit and transform the encoder on the categorical columns
test['type'] = encoder.fit_transform(test['type'])
test['lease_type']= encoder.fit_transform(test['lease_type'])
test['facing']= encoder.fit_transform(test['facing'])
test['water_supply']= encoder.fit_transform(test['water_supply'])
test['building_type']= encoder.fit_transform(test['building_type'])
test['furnishing']= encoder.fit_transform(test['furnishing'])
test['parking']= encoder.fit_transform(test['parking'])
```

```
amenities_dicts = [json.loads(entry) for entry in test["amenities"]]
data = pd.DataFrame(amenities_dicts)
mean_value= data.mean()
data= data.fillna(mean_value)
data= data.astype(int)
```

PREDICTION

```
target = model.predict(test)
```

```
rents = pd.DataFrame({'id': id, 'rent': target})
```

```
rents.to_csv('rents.csv', index= False)
```

THANK YOU!!