Import libraries

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor

# For tidy display in notebooks
pd.set_option('display.max_columns', 200)
pd.set_option('display.width', 200)
sns.set_style("whitegrid")
```

Load Dataset

```
# Load Airbnb dataset
df = pd.read_csv("Airbnb_data - airbnb_data.csv")

# Quick overview
print("Dataset Shape:", df.shape)
print("\nFirst 5 rows:\n", df.head())
print("\nMissing values:\n", df.isnull().sum())
```

```
Dataset Shape: (74111, 29)

First 5 rows:
          id  log_price property_type        room_type
amenities  accommodates  bathrooms  bed_type cancellation_policy
cleaning_fee city  \
0   6901257   5.010635     Apartment  Entire home/apt  {"Wireless
Internet","Air conditioning",Kitche...             3        1.0  Real Bed
strict          True  NYC
1   6304928   5.129899     Apartment  Entire home/apt  {"Wireless
Internet","Air conditioning",Kitche...             7        1.0  Real Bed
strict          True  NYC
2   7919400   4.976734     Apartment  Entire home/apt {TV,"Cable
TV","Wireless Internet","Air condit...             5        1.0  Real Bed
moderate          True  NYC
3  13418779   6.620073         House  Entire home/apt  {TV,"Cable
TV",Internet,"Wireless Internet",Ki...             4        1.0  Real Bed
flexible          True   SF
```

```
4   3808709   4.744932     Apartment  Entire home/apt  {TV,Internet,"Wireless
Internet","Air conditio...             2        1.0  Real Bed
moderate          True   DC

                                        description first_review
host_has_profile_pic host_identity_verified host_response_rate  host_since
instant_bookable last_review   latitude   longitude  \
0  Beautiful, sunlit brownstone 1-bedroom in the ...   18-06-2016
t                    t                NaN  26-03-2012                 f  18-
07-2016  40.696524  -73.991617
1  Enjoy travelling during your stay in Manhattan...   05-08-2017
t                    f               100%  19-06-2017                 t  23-
09-2017  40.766115  -73.989040
2  The Oasis comes complete with a full backyard ...   30-04-2017
t                    t               100%  25-10-2016                 t  14-
09-2017  40.808110  -73.943756
3  This light-filled home-away-from-home is super...          NaN
t                    t                NaN  19-04-2015                 f
NaN  37.772004 -122.431619
4  Cool, cozy, and comfortable studio located in ...   12-05-2015
t                    t               100%  01-03-2015                 t  22-
01-2017  38.925627  -77.034596

                                        name     neighbourhood
number_of_reviews  review_scores_rating
thumbnail_url zipcode  bedrooms  beds
0            Beautiful brownstone 1-bedroom  Brooklyn Heights
2                 100.0  https://a0.muscache.com/im/pictures/6d7cbbf7-c...
11201       1.0   1.0
1  Superb 3BR Apt Located Near Times Square    Hell's Kitchen
6                  93.0  https://a0.muscache.com/im/pictures/348a55fe-4...
10019       3.0   3.0
2                         The Garden Oasis            Harlem
10                   92.0  https://a0.muscache.com/im/pictures/6fae5362-9...
10027       1.0   3.0
3        Beautiful Flat in the Heart of SF!      Lower Haight
0                    NaN  https://a0.muscache.com/im/pictures/72208dad-9...
94117       2.0   2.0
4                Great studio in midtown DC  Columbia Heights
4                   40.0                                                NaN
20009       0.0   1.0

Missing values:
 id                            0
log_price                     0
property_type                 0
room_type                     0
amenities                     0
accommodates                  0
bathrooms                   200
```

```
bed_type                      0
cancellation_policy           0
cleaning_fee                  0
city                          0
description                   0
first_review              15864
host_has_profile_pic        188
host_identity_verified      188
host_response_rate        18299
host_since                  188
instant_bookable              0
last_review               15827
latitude                      0
longitude                     0
name                          0
neighbourhood              6872
number_of_reviews             0
review_scores_rating      16722
thumbnail_url              8216
zipcode                     968
bedrooms                     91
beds                        131
dtype: int64
```

Handle Missing values

```
# handle missing values
missing_pct = df.isnull().mean().sort_values(ascending=False)
print("Columns with highest missing %:\n", missing_pct.head(20))

# Drop columns with > 30% missing
cols_to_drop = missing_pct[missing_pct > 0.30].index.tolist()
print("Dropping columns (>30% missing):", cols_to_drop)
df = df.drop(columns=cols_to_drop)

# Fill numeric columns with median; categorical with mode
num_cols = df.select_dtypes(include=np.number).columns.tolist()
cat_cols = df.select_dtypes(include='object').columns.tolist()

# Keep 'amenities' as string column
if 'amenities' in cat_cols:
    cat_cols.remove('amenities')

# Impute numeric
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Impute categorical
for c in cat_cols:
    if df[c].isnull().any():
```

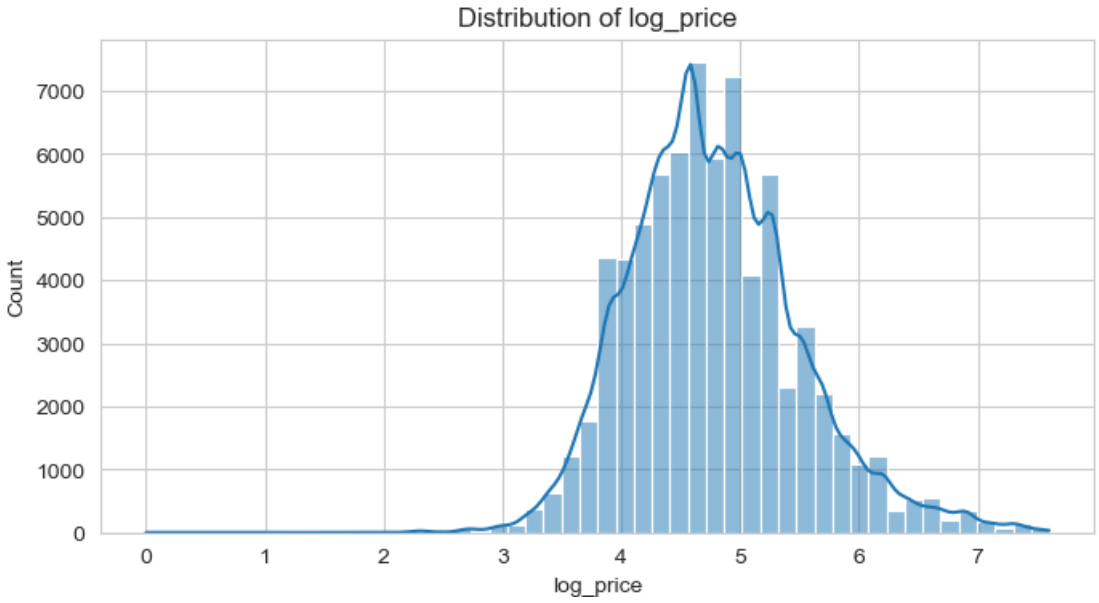
```
        mode = df[c].mode()
        fill_val = mode.iloc[0] if (not mode.empty) else 'missing'
        df[c] = df[c].fillna(fill_val)

# amenities - replace NaN with empty string
if 'amenities' in df.columns:
    df['amenities'] = df['amenities'].fillna('')
```

```
Columns with highest missing %:
 host_response_rate        0.246913
review_scores_rating      0.225635
first_review              0.214057
last_review               0.213558
thumbnail_url             0.110861
neighbourhood             0.092726
zipcode                   0.013061
bathrooms                 0.002699
host_identity_verified    0.002537
host_has_profile_pic      0.002537
host_since                0.002537
beds                      0.001768
bedrooms                  0.001228
description               0.000000
city                      0.000000
cleaning_fee              0.000000
cancellation_policy       0.000000
accommodates              0.000000
bed_type                  0.000000
room_type                 0.000000
dtype: float64
Dropping columns (>30% missing): []
```

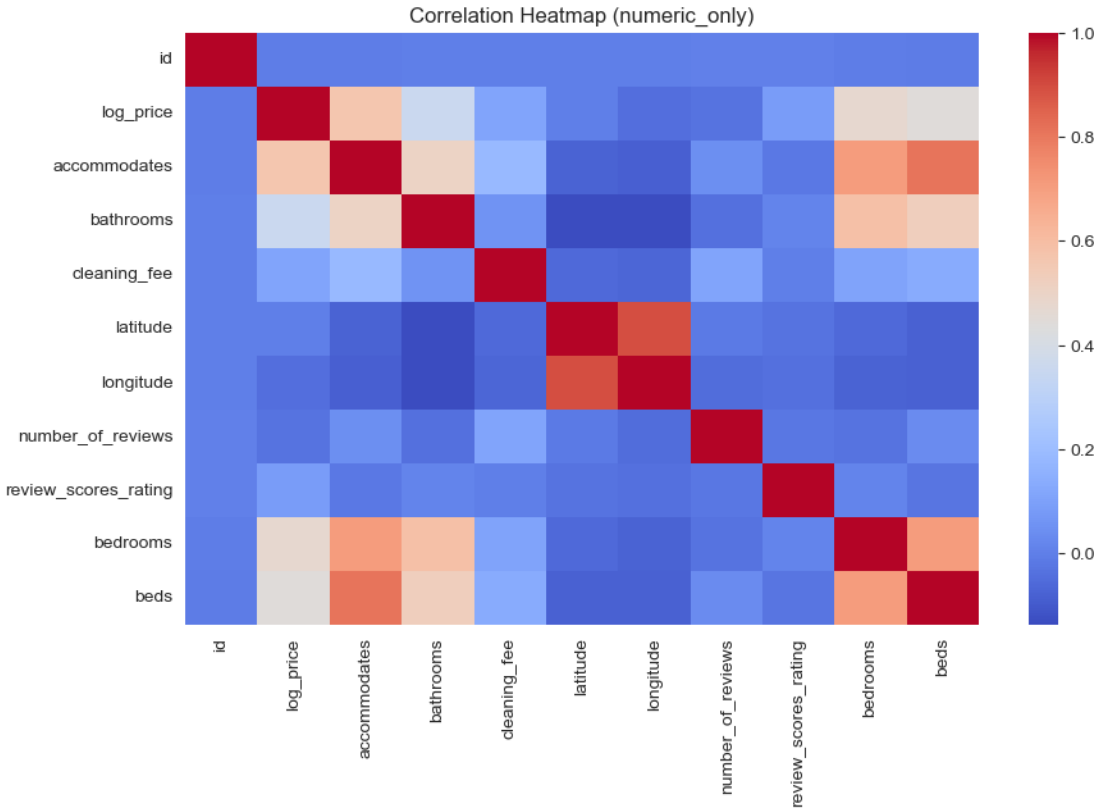
Explanatory Data Analysis

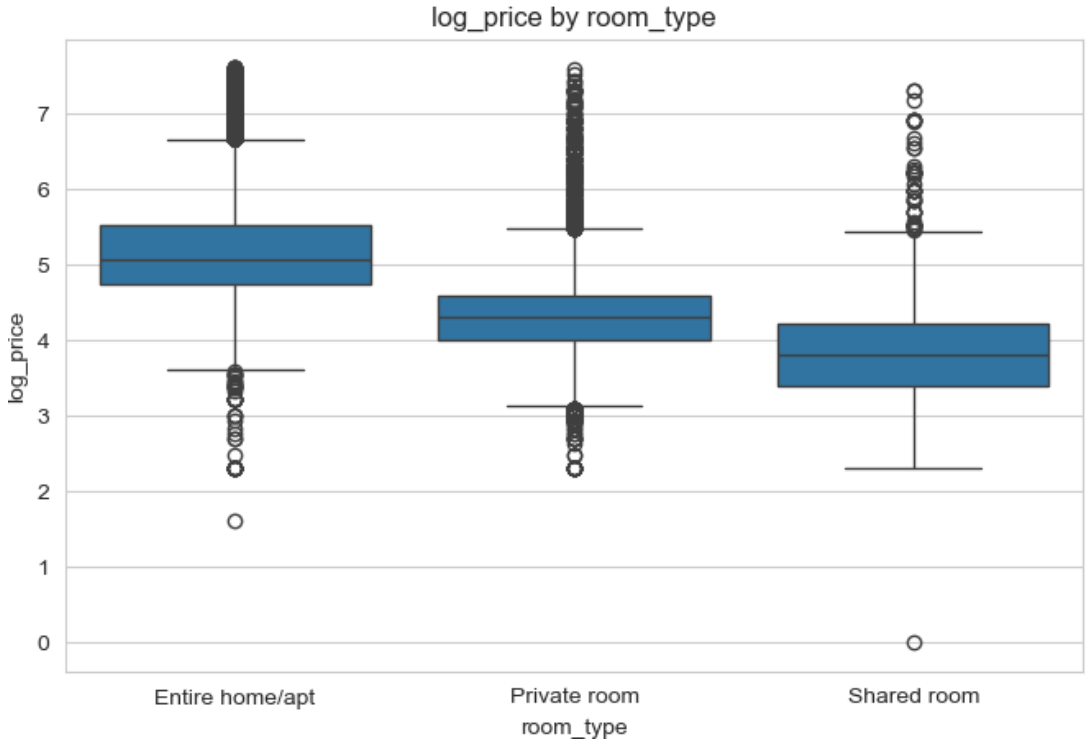
```
# Cell 7: Exploratory Data Analysis (REPLACE)
# Distribution of target (log_price)
if 'log_price' in df.columns:
    plt.figure(figsize=(8,4))
    sns.histplot(df['log_price'], bins=50, kde=True)
    plt.title("Distribution of log_price")
    plt.show()

    # If you want to see original prices (exp of log1p)
    if 'price' in df.columns:
        plt.figure(figsize=(8,4))
        sns.histplot(np.expm1(df['log_price']), bins=50, kde=True)
        plt.title("Distribution of price (exp of log1p)")
        plt.show()

    # Outlier detection via IQR in log space
    Q1 = df['log_price'].quantile(0.25)
```

```
    Q3 = df['log_price'].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    print(f"log_price IQR outlier bounds: lower={lower:.2f},
upper={upper:.2f}")
    outliers = df[(df['log_price'] < lower) | (df['log_price'] > upper)]
    print("Number of outlier listings (IQR rule):", len(outliers))

# Correlation heatmap (numeric features)
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(numeric_only=True), annot=False, cmap="coolwarm")
plt.title("Correlation Heatmap (numeric_only)")
plt.show()

# Example boxplot: price by room type (if present)
if 'room_type' in df.columns and 'log_price' in df.columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(x='room_type', y='log_price', data=df)
    plt.title("log_price by room_type")
    plt.show()
```




```
log_price IQR outlier bounds: lower=2.96, upper=6.57
Number of outlier listings (IQR rule): 1532
```


Correlation Heatmap (numeric_only)
id
log_price
accommodates
bathrooms
cleaning_fee
latitude
longitude
number_of_reviews
review_scores_rating
bedrooms
beds
1.0
0.8
0.6
0.4
0.2
0.0


log_price by room_type
log_price
7
6
5
4
3
2
1
0
Entire home/apt
Private room
Shared room
room_type

Feature Engineering

```
# 1) Number of amenities (robust)
if 'amenities' in df.columns:
    df['num_amenities'] = df['amenities'].apply(
        lambda x: len([a.strip() for a in str(x).split(',') if a.strip() !=
''])
    )
else:
    df['num_amenities'] = 0

# 2) Host response rate -> numeric (if exists)
if 'host_response_rate' in df.columns:
    df['host_response_rate'] = (
        df['host_response_rate'].astype(str)
        .str.replace('%', '', regex=False)
        .replace('', np.nan)
    ).astype(float)

# 3) Host activity metric: number of listings per host
(host_total_listings_count)
if 'host_id' in df.columns:
    host_counts =
df.groupby('host_id').size().rename('host_total_listings_count')
    df = df.merge(host_counts, left_on='host_id', right_index=True,
how='left')

# 4) Neighborhood popularity: number of listings per neighbourhood
nb_cols = [c for c in
['neighbourhood_cleansed','neighborhood','neighbourhood'] if c in df.columns]
if nb_cols:
    nb_col = nb_cols[0]
    nb_counts =
df[nb_col].value_counts().rename_axis(nb_col).reset_index(name='neighborhood_
popularity')
    df = df.merge(nb_counts, on=nb_col, how='left')
else:
    # create placeholder
    df['neighborhood_popularity'] = 0

# 5) Host seniority (days since host_since)
if 'host_since' in df.columns:
    df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
    df['host_age_days'] = (pd.Timestamp('today') -
df['host_since']).dt.days.fillna(0)
```

```
# Confirm feature creation
print("New features (sample):", df[['num_amenities'] + ([
'host_total_listings_count'] if 'host_total_listings_count' in df.columns
else [])][:5])
```

```
New features (sample):    num_amenities
0                  9
1                 15
2                 19
3                 15
4                 12
```

Split features and target

```
# We'll split the dataframe (not preprocessed X) so that preprocessing is fit
on train only.

# define target
y_col = 'log_price'
if y_col not in df.columns:
    raise KeyError("log_price not found. Make sure price/log_price is present
(see load cell).")

# first split: test set (20% final)
train_val_df, test_df = train_test_split(df, test_size=0.20, random_state=42)

# second split: from train_val split validation (0.25 of train_val -> 0.2
overall)
train_df, val_df = train_test_split(train_val_df, test_size=0.25,
random_state=42)

print("Shapes -> train, val, test:", train_df.shape, val_df.shape,
test_df.shape)

# Identify numeric and categorical features (choose columns we engineered)
exclude_cols = [y_col, 'price']  # exclude price and target
all_numeric = train_df.select_dtypes(include=np.number).columns.tolist()
numeric_features = [c for c in all_numeric if c not in exclude_cols]

all_categorical = train_df.select_dtypes(include='object').columns.tolist()
# keep a sane set of categorical features: small-cardinality ones commonly
used
categorical_candidates = ['room_type', 'property_type',
'neighbourhood_cleansed', 'neighborhood', 'host_verifications']
categorical_features = [c for c in categorical_candidates if c in
train_df.columns]
```

```
print("Numeric features sample:", numeric_features[:10])
print("Categorical features to encode:", categorical_features)
```

```
Shapes -> train, val, test: (44466, 33) (14822, 33) (14823, 33)
Numeric features sample: ['id', 'accommodates', 'bathrooms',
'host_response_rate', 'latitude', 'longitude', 'number_of_reviews',
'review_scores_rating', 'bedrooms', 'beds']
Categorical features to encode: ['room_type', 'property_type']
```

Train models using Pipeline + ColumnTransformer (FIXED)

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline

# Preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False),
categorical_features)
    ],
    remainder='drop'  # drop any other columns not in lists
)

# Model pipelines
lr_pipeline = Pipeline([('preprocessor', preprocessor),
                        ('model', LinearRegression())])

rf_pipeline = Pipeline([('preprocessor', preprocessor),
                        ('model', RandomForestRegressor(random_state=42,
n_jobs=-1))])

xgb_pipeline = Pipeline([('preprocessor', preprocessor),
                         ('model', XGBRegressor(objective='reg:squarederror',
random_state=42, n_jobs=1))])

# Prepare X/y for pipeline fitting (pandas DataFrames)
X_train = train_df[numeric_features + categorical_features]
y_train = train_df[y_col]
X_val = val_df[numeric_features + categorical_features]
y_val = val_df[y_col]
X_test = test_df[numeric_features + categorical_features]
y_test = test_df[y_col]

# Fit and evaluate on validation
def eval_on_val(pipeline, name):
    pipeline.fit(X_train, y_train)
    y_pred_val = pipeline.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, y_pred_val))
```

```
    mae = mean_absolute_error(y_val, y_pred_val)
    r2 = r2_score(y_val, y_pred_val)
    print(f"{name} on VAL → RMSE(log): {rmse:.4f}, MAE(log): {mae:.4f}, R2:
{r2:.4f}")
    return pipeline, (y_pred_val, rmse, mae, r2)

print("Training & evaluating Linear Regression...")
lr_pipeline, lr_val_stats = eval_on_val(lr_pipeline, "LinearRegression")

print("Training & evaluating Random Forest...")
rf_pipeline, rf_val_stats = eval_on_val(rf_pipeline, "RandomForest")

print("Training & evaluating XGBoost...")
xgb_pipeline, xgb_val_stats = eval_on_val(xgb_pipeline, "XGBoost")
```

```
Training & evaluating Linear Regression...
LinearRegression on VAL → RMSE(log): 0.4928, MAE(log): 0.3694, R2: 0.5345
Training & evaluating Random Forest...
RandomForest on VAL → RMSE(log): 0.3983, MAE(log): 0.2865, R2: 0.6959
Training & evaluating XGBoost...
XGBoost on VAL → RMSE(log): 0.3941, MAE(log): 0.2857, R2: 0.7022
```

Model Evaluation on TEST set

```
# Choose the final model

pipelines_val_stats = {
    'LinearRegression': lr_val_stats,
    'RandomForest': rf_val_stats,
    'XGBoost': xgb_val_stats
}

best_name = min(pipelines_val_stats.keys(), key=lambda k:
pipelines_val_stats[k][1])  # minimize RMSE
best_pipeline = {'LinearRegression': lr_pipeline, 'RandomForest':
rf_pipeline, 'XGBoost': xgb_pipeline}[best_name]
print("Best model on validation:", best_name)

# Predict on test set
y_pred_test_log = best_pipeline.predict(X_test)

# Metrics in log space
rmse_log = np.sqrt(mean_squared_error(y_test, y_pred_test_log))
mae_log = mean_absolute_error(y_test, y_pred_test_log)
r2v = r2_score(y_test, y_pred_test_log)
print(f"Test set (log space) → RMSE: {rmse_log:.4f}, MAE: {mae_log:.4f}, R²:
{r2v:.4f}")

# Back-transform to original prices (we used log1p above -> use expm1)
if 'price' in df.columns:
```

```
    y_test_price = np.expm1(y_test)
    y_pred_test_price = np.expm1(y_pred_test_log)
    rmse_price = np.sqrt(mean_squared_error(y_test_price, y_pred_test_price))
    mae_price = mean_absolute_error(y_test_price, y_pred_test_price)
    print(f"Test set (original price) → RMSE: {rmse_price:.2f}, MAE:
{mae_price:.2f}")
else:
    print("Original 'price' not present -> skipping original-scale metrics.")
```

```
Best model on validation: XGBoost
Test set (log space) → RMSE: 0.3931, MAE: 0.2847, R²: 0.6992
Original 'price' not present -> skipping original-scale metrics.
```

GridSearch on pipeline

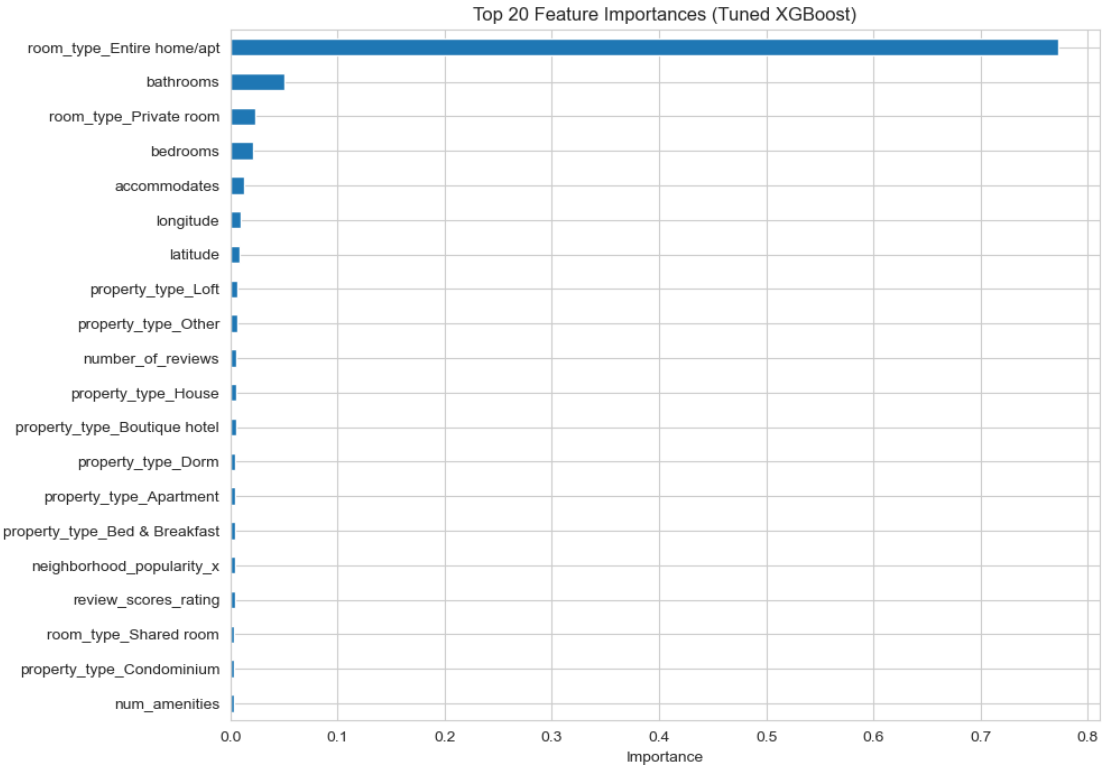
```
#Define parameter grid for tuning  cell 20
param_grid = {
    'model__n_estimators': [200, 400],
    'model__max_depth': [3, 5, 7],
    'model__learning_rate': [0.05, 0.1],
    'model__subsample': [0.8, 1.0]
}

# 🔧 GridSearchCV with 3-fold cross validation
grid = GridSearchCV(
    estimator=xgb_pipeline,       # from Cell 18
    param_grid=param_grid,
    cv=3,
    scoring='r2',
    n_jobs=-1,
    verbose=1
)

# Fit grid search on training data
grid.fit(X_train, y_train)

print("Best Params (XGB GridSearch):", grid.best_params_)

# ☑️ Evaluating best model on validation & test sets
best_xgb = grid.best_estimator_

# Predict & evaluate
y_val_pred = best_xgb.predict(X_val)
y_test_pred = best_xgb.predict(X_test)

val_rmse = np.sqrt(mean_squared_error(y_val, y_val_pred))
val_r2 = r2_score(y_val, y_val_pred)

test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
```

```
test_r2 = r2_score(y_test, y_test_pred)

print(f"Tuned XGB - VAL RMSE: {val_rmse:.4f}")
print(f"Tuned XGB - VAL R2: {val_r2:.4f}")
print(f"Tuned XGB - TEST RMSE: {test_rmse:.4f}")
print(f"Tuned XGB - TEST R2: {test_r2:.4f}")
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Best Params (XGB GridSearch): {'model__learning_rate': 0.05,
'model__max_depth': 7, 'model__n_estimators': 400, 'model__subsample': 0.8}
Tuned XGB - VAL RMSE: 0.3867
Tuned XGB - VAL R2: 0.7133
Tuned XGB - TEST RMSE: 0.3854
Tuned XGB - TEST R2: 0.7109
```

Feature Importance Visualization using tuned XGB

```
model_pipeline = best_xgb  # Use tuned XGB model from GridSearchCV

# Extract fitted preprocessor and model
preprocessor = model_pipeline.named_steps['preprocessor']
model = model_pipeline.named_steps['model']

# Build feature names: numeric + one-hot encoded categorical names
num_feats = numeric_features
cat_transformer = preprocessor.named_transformers_['cat']
cat_ohe_names =
list(cat_transformer.get_feature_names_out(categorical_features)) if
hasattr(cat_transformer, "get_feature_names_out") else []
feature_names = list(num_feats) + cat_ohe_names

# Get feature importances
importances = pd.Series(model.feature_importances_, index=feature_names)
importances = importances.sort_values(ascending=True).tail(20)

# Plot
plt.figure(figsize=(10, 8))
importances.plot(kind='barh')
plt.title("Top 20 Feature Importances (Tuned XGBoost)")
plt.xlabel("Importance")
plt.show()
```



VIDEO LINK : https://drive.google.com/file/d/174haZ34IVSNan004we-P6wuf1mvDkel1/view?usp=drivesdk