

# ML\_Project\_task1

September 30, 2025

```
[ ]: #videolink:https://drive.google.com/file/d/1BBwLTHphMlOOyuRAtokEHkjZROBEkpAT/  
     ↪view?usp=sharing
```

```
[36]: %matplotlib inline  
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 StandardScaler, OneHotEncoder  
from sklearn.compose import ColumnTransformer  
from sklearn.pipeline import Pipeline  
from sklearn.impute import SimpleImputer  
from sklearn.linear_model import LinearRegression  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score  
from datetime import datetime
```

```
[37]: df = pd.read_excel('Airbnb_data.xlsx', sheet_name='airbnb_data')
```

```
[3]: print("Data Loaded Successfully")  
print(df.head())  
print(df.info())  
print(df.describe())
```

Data Loaded Successfully

	id	log_price	property_type	room_type \
0	6901257	5.010635	Apartment	Entire home/apt
1	6304928	5.129899	Apartment	Entire home/apt
2	7919400	4.976734	Apartment	Entire home/apt
3	13418779	6.620073	House	Entire home/apt
4	3808709	4.744932	Apartment	Entire home/apt

	amenities	accommodates	bathrooms \
0	{"Wireless Internet","Air conditioning",Kitche...	3	1.0
1	{"Wireless Internet","Air conditioning",Kitche...	7	1.0
2	{TV,"Cable TV","Wireless Internet","Air condit...	5	1.0
3	{TV,"Cable TV",Internet,"Wireless Internet",Ki...	4	1.0

```
4 {TV,Internet,"Wireless Internet","Air conditio...      2      1.0
```

```

    bed_type cancellation_policy cleaning_fee ... latitude longitude \
0 Real Bed          strict          True ... 40.696524 -73.991617
1 Real Bed          strict          True ... 40.766115 -73.989040
2 Real Bed      moderate          True ... 40.808110 -73.943756
3 Real Bed      flexible          True ... 37.772004 -122.431619
4 Real Bed      moderate          True ... 38.925627 -77.034596

```

```

                                name      neighbourhood \
0      Beautiful brownstone 1-bedroom  Brooklyn Heights
1  Superb 3BR Apt Located Near Times Square  Hell's Kitchen
2                                The Garden Oasis      Harlem
3      Beautiful Flat in the Heart of SF!      Lower Haight
4      Great studio in midtown DC  Columbia Heights

```

```

    number_of_reviews  review_scores_rating \
0              2              100.0
1              6              93.0
2             10              92.0
3              0              NaN
4              4              40.0

```

```

                                thumbnail_url zipcode bedrooms  beds
0  https://a0.muscache.com/im/pictures/6d7cbbf7-c...  11201      1.0  1.0
1  https://a0.muscache.com/im/pictures/348a55fe-4...  10019      3.0  3.0
2  https://a0.muscache.com/im/pictures/6fae5362-9...  10027      1.0  3.0
3  https://a0.muscache.com/im/pictures/72208dad-9...  94117      2.0  2.0
4                                NaN      20009      0.0  1.0

```

[5 rows x 29 columns]

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 74111 entries, 0 to 74110
```

```
Data columns (total 29 columns):
```

```

#      Column              Non-Null Count  Dtype
---  -
0      id              74111 non-null  int64
1      log_price       74111 non-null  float64
2      property_type   74111 non-null  object
3      room_type       74111 non-null  object
4      amenities       74111 non-null  object
5      accommodates     74111 non-null  int64
6      bathrooms       73911 non-null  float64
7      bed_type        74111 non-null  object
8      cancellation_policy  74111 non-null  object
9      cleaning_fee     74111 non-null  bool
10     city            74111 non-null  object
11     description     74105 non-null  object

```

12	first_review	58247	non-null	object
13	host_has_profile_pic	73923	non-null	object
14	host_identity_verified	73923	non-null	object
15	host_response_rate	55812	non-null	float64
16	host_since	73923	non-null	object
17	instant_bookable	74111	non-null	object
18	last_review	58284	non-null	object
19	latitude	74111	non-null	float64
20	longitude	74111	non-null	float64
21	name	74101	non-null	object
22	neighbourhood	67239	non-null	object
23	number_of_reviews	74111	non-null	int64
24	review_scores_rating	57389	non-null	float64
25	thumbnail_url	65895	non-null	object
26	zipcode	73143	non-null	object
27	bedrooms	74020	non-null	float64
28	beds	73980	non-null	float64

dtypes: bool(1), float64(8), int64(3), object(17)

memory usage: 15.9+ MB

None

	id	log_price	accommodates	bathrooms \
count	7.411100e+04	74111.000000	74111.000000	73911.000000
mean	1.126662e+07	4.782069	3.155146	1.235263
std	6.081735e+06	0.717394	2.153589	0.582044
min	3.440000e+02	0.000000	1.000000	0.000000
25%	6.261964e+06	4.317488	2.000000	1.000000
50%	1.225415e+07	4.709530	2.000000	1.000000
75%	1.640226e+07	5.220356	4.000000	1.000000
max	2.123090e+07	7.600402	16.000000	8.000000

	host_response_rate	latitude	longitude	number_of_reviews \
count	55812.000000	74111.000000	74111.000000	74111.000000
mean	0.943520	38.445958	-92.397525	20.900568
std	0.163418	3.080167	21.705322	37.828641
min	0.000000	33.338905	-122.511500	0.000000
25%	1.000000	34.127908	-118.342374	1.000000
50%	1.000000	40.662138	-76.996965	6.000000
75%	1.000000	40.746096	-73.954660	23.000000
max	1.000000	42.390437	-70.985047	605.000000

	review_scores_rating	bedrooms	beds
count	57389.000000	74020.000000	73980.000000
mean	94.067365	1.265793	1.710868
std	7.836556	0.852143	1.254142
min	20.000000	0.000000	0.000000
25%	92.000000	1.000000	1.000000
50%	96.000000	1.000000	1.000000
75%	100.000000	1.000000	2.000000

```
max          100.000000    10.000000    18.000000
```

```
[4]: print("Cleaning Fee Values:\n", df['cleaning_fee'].value_counts(dropna=False))
      print("Cleaning Fee Missing Count:", df['cleaning_fee'].isnull().sum())
```

Cleaning Fee Values:

```
cleaning_fee
```

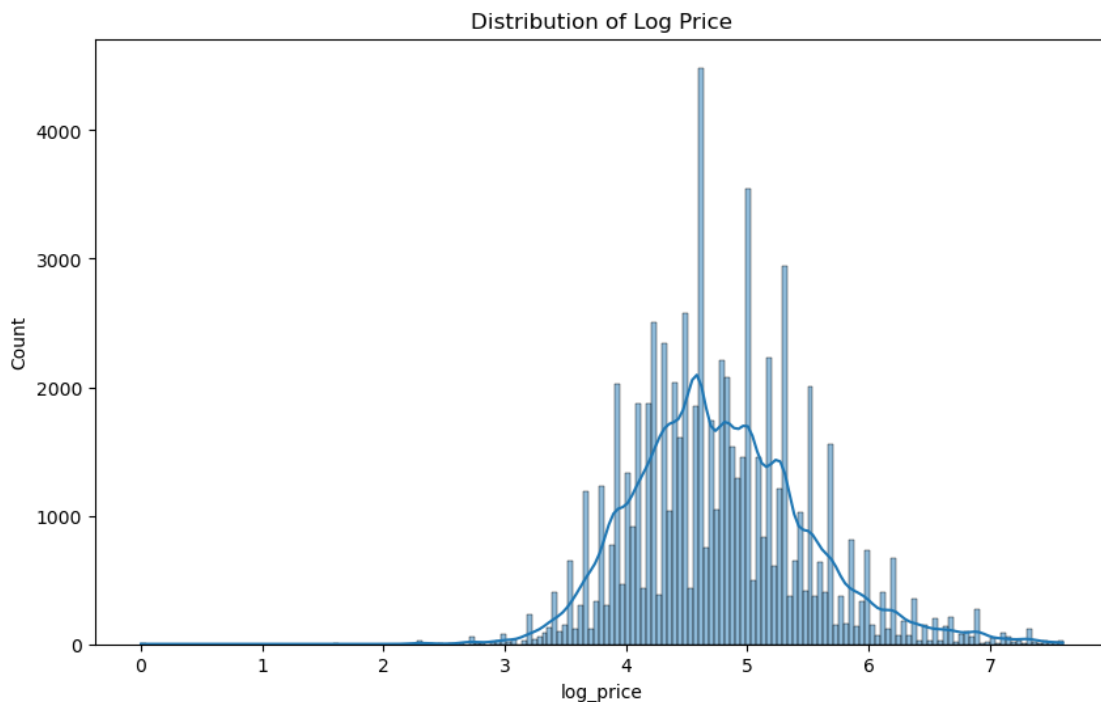
```
True      54403
```

```
False     19708
```

```
Name: count, dtype: int64
```

Cleaning Fee Missing Count: 0

```
[5]: fig, ax = plt.subplots(figsize=(10, 6))
      sns.histplot(df['log_price'], kde=True, ax=ax)
      ax.set_title('Distribution of Log Price')
      plt.show()
```



```
[6]: missing = df.isnull().sum()
      print("Missing Values:\n", missing[missing > 0])
```

Missing Values:

```
bathrooms          200
```

```
description          6
```

```
first_review       15864
```

```
host_has_profile_pic  188
```

```
host_identity_verified 188
```

```

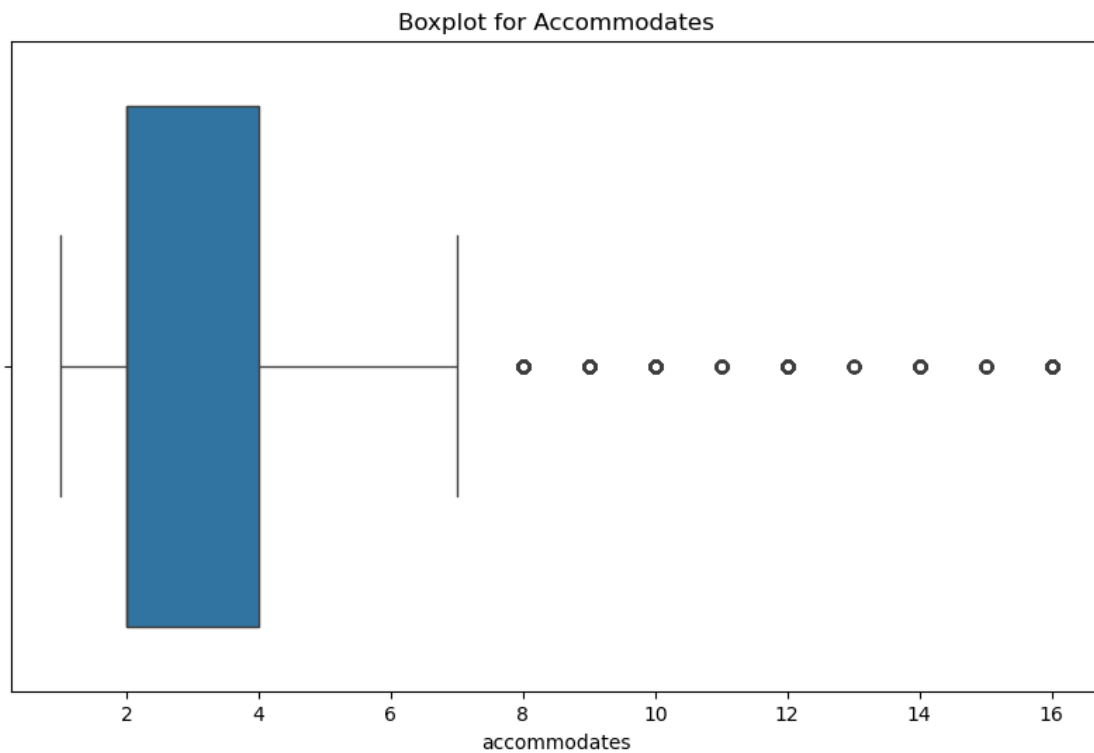
host_response_rate    18299
host_since            188
last_review           15827
name                  10
neighbourhood         6872
review_scores_rating   16722
thumbnail_url         8216
zipcode               968
bedrooms              91
beds                  131
dtype: int64

```

```

[7]: fig, ax = plt.subplots(figsize=(10, 6))
     sns.boxplot(x=df['accommodates'], ax=ax)
     ax.set_title('Boxplot for Accommodates')
     plt.show()

```



```

[8]: # Data Preprocessing

df['bathrooms'] = df['bathrooms'].fillna(df['bathrooms'].median())
df['bedrooms'] = df['bedrooms'].fillna(df['bedrooms'].median())
df['beds'] = df['beds'].fillna(df['beds'].median())

```

```
df['review_scores_rating'] = df['review_scores_rating'].
    ↪fillna(df['review_scores_rating'].median())
df['number_of_reviews'] = df['number_of_reviews'].fillna(0)
```

```
[9]: # Handle host_response_rate: Convert to numeric, handle potential percentages
df['host_response_rate'] = pd.to_numeric(df['host_response_rate'],
    ↪errors='coerce')
if df['host_response_rate'].max() > 1:
    df['host_response_rate'] = df['host_response_rate'] / 100
df['host_response_rate'] = df['host_response_rate'].
    ↪fillna(df['host_response_rate'].median())
```

```
[10]: # Convert booleans to 0/1
bool_cols = ['cleaning_fee', 'host_has_profile_pic', 'host_identity_verified',
    ↪'instant_bookable']
for col in bool_cols:
    df[col] = df[col].map({'true': 1, 't': 1, 'f': 0, 'false': 0, np.nan: 0})
```

```
[11]: # Verify cleaning_fee after mapping
print("Cleaning Fee After Mapping:\n", df['cleaning_fee'].
    ↪value_counts(dropna=False))
```

```
Cleaning Fee After Mapping:
cleaning_fee
NaN      74111
Name: count, dtype: int64
```

```
[12]: # Feature Engineering
# Amenities count
df['amenities'] = df['amenities'].fillna('')
df['amenities_count'] = df['amenities'].apply(lambda x: len(x.strip('{}').
    ↪split(','))) if x else 0)
```

```
[13]: # Host tenure (days since host_since)
df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce',
    ↪dayfirst=True)
df['host_tenure_days'] = (datetime.now() - df['host_since']).dt.days
df['host_tenure_days'] = df['host_tenure_days'].fillna(df['host_tenure_days'].
    ↪median())
```

```
[14]: # Neighborhood popularity (average reviews per neighborhood)
neigh_reviews = df.groupby('neighbourhood')['number_of_reviews'].mean().
    ↪to_dict()
df['neigh_popularity'] = df['neighbourhood'].map(neigh_reviews).fillna(0)
```

```
[15]: # Drop unnecessary columns
```

```
drop_cols = ['id', 'description', 'first_review', 'last_review', 'name',
             ↪ 'thumbnail_url', 'zipcode', 'latitude', 'longitude', 'amenities',
             ↪ 'host_since']
df = df.drop(columns=drop_cols)
```

```
[16]: # Handle outliers
df['accommodates'] = np.clip(df['accommodates'], df['accommodates'].quantile(0.
    ↪ 01), df['accommodates'].quantile(0.99))
```

```
[17]: # Prepare features and target
X = df.drop('log_price', axis=1)
y = df['log_price']
```

```
[18]: # Identify categorical and numerical columns
cat_cols = ['property_type', 'room_type', 'bed_type', 'cancellation_policy',
    ↪ 'city', 'neighbourhood']
```

```
[19]: # Ensure cleaning_fee is treated as numerical (binary 0/1)
num_cols = [col for col in X.columns if col not in cat_cols]
```

```
[20]: # Check if cleaning_fee has valid values; if entirely NaN, remove from num_cols
    ↪ or drop
if df['cleaning_fee'].isnull().all() or df['cleaning_fee'].nunique() == 1:
    print("Warning: cleaning_fee has no valid values or is constant. Removing
    ↪ from numerical columns.")
    num_cols = [col for col in num_cols if col != 'cleaning_fee']
    if 'cleaning_fee' in X.columns:
        X = X.drop(columns=['cleaning_fee'])
```

Warning: cleaning\_fee has no valid values or is constant. Removing from numerical columns.

```
[21]: # Data Splitting: 70% train, 15% val, 15% test
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.
    ↪ 15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full,
    ↪ test_size=0.1765, random_state=42) # 0.1765 ~ 15/85
```

```
[22]: # Step 2: Model Development
# Preprocessing Pipeline
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
```

```

        ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False)) #_
        ↪For newer scikit-learn
    ])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_cols),
        ('cat', cat_transformer, cat_cols)
    ])

```

[23]: *# Model Pipeline - Start with Linear Regression*

```

lr_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])

```

[24]: *# Train Linear Regression*

```

lr_pipeline.fit(X_train, y_train)

```

```

[24]: Pipeline(steps=[('preprocessor',
                        ColumnTransformer(transformers=[('num',
                                                         Pipeline(steps=[('imputer',
                                                                              SimpleImputer(strategy='median')),
                                                                              ('scaler',
                                                                              StandardScaler()))]),
                                                         ('cat',
                                                         Pipeline(steps=[('imputer',
                                                                              SimpleImputer(fill_value='missing',
                                                                              strategy='constant')),
                                                                              ('onehot',
                                                                              OneHotEncoder(handle_unknown='ignore',
                                                                              sparse_output=False))])),
                        ['accommodates', 'bathrooms',
                        'host_has_profile_pic',
                        'host_identity_verified',
                        'host_response_rate',
                        'instant_bookable',
                        'number_of_reviews',
                        'review_scores_rating',
                        'bedrooms', 'beds',
                        'amenities_count',
                        'host_tenure_days',
                        'neigh_popularity']),
                        ('cat',
                        Pipeline(steps=[('imputer',
                                                                              SimpleImputer(fill_value='missing',
                                                                              strategy='constant')),
                                                                              ('onehot',
                                                                              OneHotEncoder(handle_unknown='ignore',
                                                                              sparse_output=False))])),
                        ['property_type', 'room_type',
                        'bed_type',
                        'cancellation_policy',
                        'city',

```



```

('neighbourhood']]])),
('model', LinearRegression())])

```

```

[25]: # Try Random Forest with simplified GridSearch for tuning
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(random_state=42))
])

# Simplified parameter grid to reduce computation
param_grid = {
    'model__n_estimators': [50], # Single value to avoid multiple iterations
    'model__max_depth': [10]     # Single value
}

grid_search = GridSearchCV(rf_pipeline, param_grid, cv=3,
    scoring='neg_mean_squared_error', verbose=2)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

print("Best Params:", grid_search.best_params_)

```

Fitting 3 folds for each of 1 candidates, totalling 3 fits

```

[CV] END ...model__max_depth=10, model__n_estimators=50; total time= 1.8min
[CV] END ...model__max_depth=10, model__n_estimators=50; total time= 1.8min
[CV] END ...model__max_depth=10, model__n_estimators=50; total time= 1.7min
Best Params: {'model__max_depth': 10, 'model__n_estimators': 50}

```

```

[26]: # Step 3: Model Evaluation
# Predict on validation set
y_val_pred = best_model.predict(X_val)

```

```

[27]: # Metrics on log_price
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))
mae_val = mean_absolute_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

```

```

[28]: print(f"Validation RMSE: {rmse_val}")
print(f"Validation MAE: {mae_val}")
print(f"Validation R²: {r2_val}")

```

```

Validation RMSE: 0.4432755123387756
Validation MAE: 0.33154709041466957
Validation R²: 0.6225090862746364

```

```

[29]: # Predict on test set
y_test_pred = best_model.predict(X_test)

```

```

rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
mae_test = mean_absolute_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print(f"Test RMSE: {rmse_test}")
print(f"Test MAE: {mae_test}")
print(f"Test R2: {r2_test}")

```

Test RMSE: 0.44513135352732563

Test MAE: 0.33129523205833655

Test R<sup>2</sup>: 0.6144829487412282

```

[30]: # Optional: Metrics on actual price (exp)
y_test_actual = np.exp(y_test)
y_test_pred_actual = np.exp(y_test_pred)

rmse_actual = np.sqrt(mean_squared_error(y_test_actual, y_test_pred_actual))
mae_actual = mean_absolute_error(y_test_actual, y_test_pred_actual)
print(f"Test RMSE (Actual Price): {rmse_actual}")
print(f"Test MAE (Actual Price): {mae_actual}")

```

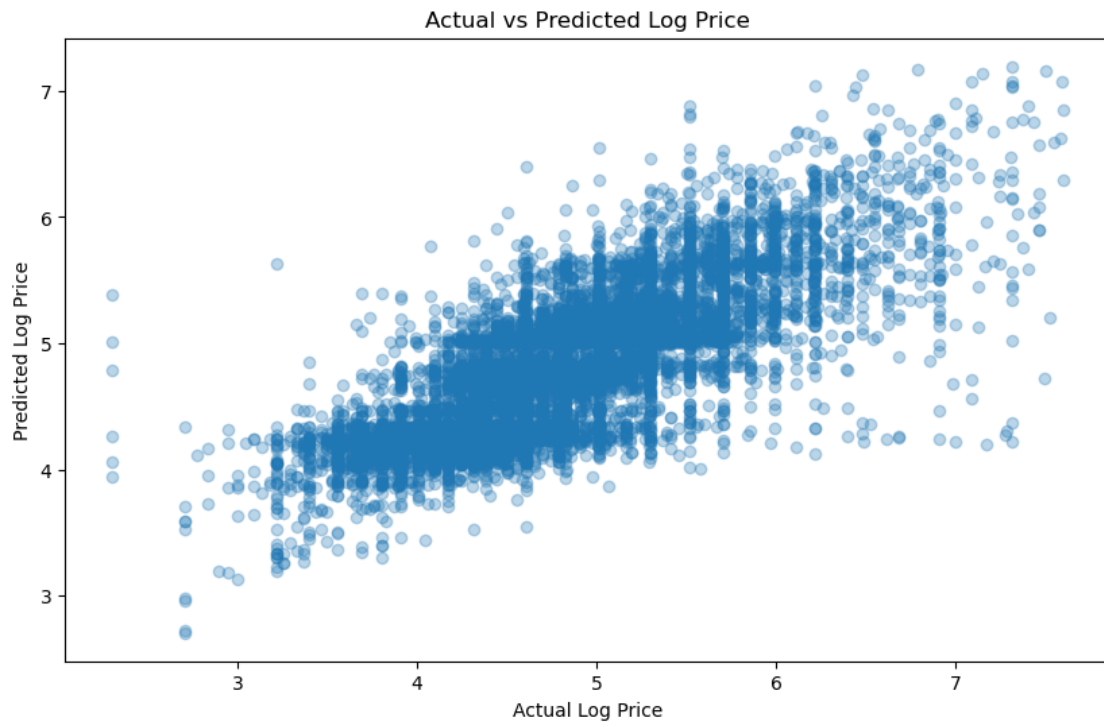
Test RMSE (Actual Price): 125.43673038572977

Test MAE (Actual Price): 57.553218288705295

```

[31]: # Visualizations: Actual vs Predicted
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(y_test, y_test_pred, alpha=0.3)
ax.set_xlabel('Actual Log Price')
ax.set_ylabel('Predicted Log Price')
ax.set_title('Actual vs Predicted Log Price')
plt.show()

```



[ ]: