```
# Imports
import warnings
warnings.filterwarnings('ignore')


import numpy as np
import pandas as pd


import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error




# Data Understanding


## Reading the dataset (csv file) into Pandas dataframe
housing_file_path = "/Users/killercookie/Documents/GitHub/School code/School-Code/Machine lab/DataSet_LakasArak_labeled.csv"
housing = pd.DataFrame(pd.read_csv(housing_file_path))


## Exploring Dataframe
### Check the head of the dataset
```

```
housing.head()
housing.shape


### The dataset shape shows the number of instances and features in the dataset
num_instances = housing.shape[0] # Number of rows
num_features = housing.shape[1] # Number of columns


housing.info()
housing.describe()


## Features and ground truth tables
### One of the columns contains the prices. In this task, we need to predict the prices based on some information that we have; thus, this column is the ground truth label.
### 'price_created_at' column has the ground truth label that we are going to use in training and testing later.
gt_feature = 'price_created_at'




# Data Preparation


### As we learned earlier, prepare the data for the next steps (e.g. train and test).
### You might use all the available features or part of them. Please justify your choices.
### Attention!!! Do not drop any N/A value.
```

```
# Function to impute missing textual data while preserving distribution
def impute_categorical_with_distribution(df, columns):
for col in columns:
if df[col].isnull().sum() > 0:
# Get the frequency distribution of the non-null values
value_counts = df[col].value_counts(normalize=True)
# Impute missing values by sampling from the observed distribution
imputed_values = np.random.choice(value_counts.index,
size=df[col].isnull().sum(),
p=value_counts.values)
df.loc[df[col].isnull(), col] = imputed_values

return df


# List of categorical/textual columns
categorical_columns = housing.select_dtypes(include=['object']).columns
housing = impute_categorical_with_distribution(housing, categorical_columns)


# Function to impute missing values based on skewness
def impute_numerical_with_distribution(df, columns):
for col in columns:
if df[col].isnull().sum() > 0:
skewness = df[col].skew()
if abs(skewness) < 0.5: # Low skewness, use mean
mean_value = df[col].mean()
noise = np.random.normal(loc=0, scale=df[col].std(), size=df[col].isnull().sum())
df.loc[df[col].isnull(), col] = mean_value + noise
else: # High skewness, use median
median_value = df[col].median()
# Sample from the observed distribution and add a bit of randomness
observed_values = df[col].dropna()
```

```
imputed_values = np.random.choice(observed_values, size=df[col].isnull().sum())
df.loc[df[col].isnull(), col] = imputed_values


return df


# List of numerical columns
numerical_columns = housing.select_dtypes(include=['float64', 'int64']).columns
housing = impute_numerical_with_distribution(housing, numerical_columns)


# File path where the new dataset will be saved
new_file_path = "/Users/killercookie/Documents/GitHub/School code/School-Code/Machine lab/new_dataset.csv"


# Save the modified dataset to a CSV file for easy visibily and accessment
housing.to_csv(new_file_path, index=False)




## Holding out test set for performance evaluation


### 1- We need to decide how much of the data is used for testing.
### In this experiment the data is labeled beforehand, we have 30% of the data for testing purposes.
### 2- How many instances do we have for training and testing?


train_set = housing[housing['split']=='train']
test_set = housing[housing['split']=='test']
```

```
train_set.shape, test_set.shape


### The following is just to assert that the data is complete and none of th einstances was dropped
test_perc = 0.3
train_perc = 1 - test_perc


assert (len(train_set) + len(test_set)) == num_instances
assert (len(train_set)) == int(train_perc*num_instances)
assert (len(test_set)) == (num_instances - len(train_set))
```

Model Selection

```
### After the data preperation/preprocessing step, the list of selected features (as strings) should be saved into a list in the form:
### features = [feature1, feature2, ...]


# Select only numerical features (int64 and float64 types)
numerical_features = housing.select_dtypes(include=['int64', 'float64']).columns


# Exclude the target column (price) and any non-feature columns like 'split'
features = [col for col in numerical_features if col != 'price_created_at']
```

We need to create features and ground truth sets for both train and test splits that we have. Use 'features' and 'gt_feature'.

```
X_train = train_set[features]
y_train = train_set[gt_feature]
```

```
X_test = test_set[features]
y_test = test_set[gt_feature]
```

```
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

For prediction, you should use the following function. As you can see, it is incomplete, please fill the gaps.
The selected model has to learn the features in the data before giving an educated prediction. Why?
We first fit the model using the train data, then use it to predict labels (prices) for the test instances. Why?

```
def model_predict(model, X_train, y_train, X_test):
# fit the model
model.fit(X_train, y_train)
```

```
# make predictions
pred = model.predict(X_test)
return model, pred
```

Using the selected models, You can make the predictions using 'model_predict' function. Please save the returned values so we can check

their performance.
...

```
# Initialize models
model_1 = LinearRegression()
model_2 = RandomForestRegressor()
model_3 = GradientBoostingRegressor()


# Make predictions using the defined model_predict function
model_1, pred_1 = model_predict(model_1, X_train, y_train, X_test)
model_2, pred_2 = model_predict(model_2, X_train, y_train, X_test)
model_3, pred_3 = model_predict(model_3, X_train, y_train, X_test)


# Check predictions (optional)
print(pred_1[:5], pred_2[:5], pred_3[:5])
```

Evaluation

For evaluation we use Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE).
Please complete the following function:

```
def model_evaluate(pred, target):
mae = mean_absolute_error(target, pred)
```

```
mape = np.mean(np.abs((target - pred) / target)) * 100 # MAPE calculation
rmse = np.sqrt(mean_squared_error(target, pred))

return mae, mape, rmse



### Check and compare the performance for all the models. Do you find any interesting observtions(s)? What are your conclusion(s)?
# Evaluate model 1 (Linear Regression)
mae_1, mape_1, rmse_1 = model_evaluate(pred_1, y_test)
print(f"Model 1 (Linear Regression): MAE={mae_1}, MAPE={mape_1}, RMSE={rmse_1}")


# Evaluate model 2 (Random Forest)
mae_2, mape_2, rmse_2 = model_evaluate(pred_2, y_test)
print(f"Model 2 (Random Forest): MAE={mae_2}, MAPE={mape_2}, RMSE={rmse_2}")


# Evaluate model 3 (Gradient Boosting)
mae_3, mape_3, rmse_3 = model_evaluate(pred_3, y_test)
print(f"Model 3 (Gradient Boosting): MAE={mae_3}, MAPE={mape_3}, RMSE={rmse_3}")



# Results:
# Model 1 (Linear Regression): MAE=5.352490791674504, MAPE=29.385819547084473, RMSE=7.404619145740831
# Model 2 (Random Forest): MAE=3.912674221203634, MAPE=21.532896217856997, RMSE=5.946060710633156
# Model 3 (Gradient Boosting): MAE=4.420130247331278, MAPE=24.338369195674282, RMSE=6.3694793132922065


# Model 1 (Linear Regression): MAE=5.348132249379024, MAPE=29.34393550578072, RMSE=7.404487388132059
# Model 2 (Random Forest): MAE=3.8869423648247183, MAPE=21.360714277230905, RMSE=5.918908892399848
# Model 3 (Gradient Boosting): MAE=4.393504220423911, MAPE=24.150770764955077, RMSE=6.33375685754291
```

Model 1 (Linear Regression): MAE=5.356422305362718, MAPE=29.388705395841235, RMSE=7.4067527226305065
Model 2 (Random Forest): MAE=3.889860241066124, MAPE=21.364579499769, RMSE=5.926298361187138
Model 3 (Gradient Boosting): MAE=4.391367562711816, MAPE=24.100257216727986, RMSE=6.333144479104173

Housing Price Forecast findings

I/ Introduction

This is the documentation of the findings gathered from the evaluation of three machine learning models (Linear Regression, Random Forest, and Gradient Boosting) and their accuracy (calculated by the mean absolue error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE)_ trained a dataset of housing information, containing only numerical variables (being postcode, room count, small room count, property area, balcony area, ad view count, active days and number of rooms) and forecasting the housing price of a test dataset.

It is important to note that during Data Preparation, due to the lack of some data in certain areas, some data was fabricated using the mean or median of given datasets such that the distribution curve stays relatively constant.

II/ Results

After running the Linear Regression, Random Forest, and Gradient Boosting models three times, the results of the models is documented as such:

	Test 1			Test 2			Test 3		
	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE
Linear Regression	5.35	29.39	7.40	5.35	29.34	7.40	5.36	29.39	7.41
Random forest	3.91	21.53	5.95	3.89	21.36	5.92	3.89	21.36	5.93
Gradient Boosting	4.42	24.34	6.37	4.39	24.15	6.33	4.39	24.10	6.33

Table 1.0: The results after running each model 3 times

	MAE	MAPE	RMSE
Linear Regression	5.35	29.39	7.40
Random forest	3.89	21.36	5.92
Gradient Boosting	4.39	24.1	6.33

Table 1.1: The average of the erros after running each model 3 times

Key**:**
MAE: Represents the average absolute difference between predicted and actual values. Lower values indicate better performance.
MAPE: Indicates the average percentage error; it's useful to understand the accuracy of predictions in relative terms. A lower MAPE suggests better performance.
RMSE: Penalizes larger errors more than MAE. It's useful for identifying outliers; lower RMSE values indicate better accuracy.

III/ Observations

Random Forest consistently outperformed both Linear Regression and Gradient Boosting in all three runs, with the lowest MAE, MAPE, and RMSE. This suggests that the Random Forest model captures the underlying patterns in the data effectively.
Gradient Boosting had performance metrics between those of Linear Regression and Random Forest, indicating that it provides a good trade-off but does not match the performance of Random Forest.

Gradient Boosting had performance metrics between those of Linear Regression and Random Forest, indicating that it provides a good trade-off but does not match the performance of Random Forest.

The values for each metric across the three runs were relatively stable, especially for the Random Forest model, indicating consistent performance. The small variations suggest that the model is robust to changes in the test set.

IV/ Conclusions

Based on the evaluation metrics, the Random Forest Regressor would be the best choice for predicting housing prices in the dataset compared to Linear Regression and Gradient Boosting. It demonstrates better accuracy and a lower error rate compared to the other models. However, it is important the average accuracy of these are still quite low and unoptimized being only about 70 - 80% making it unreliable for accurate forecasts.