1

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Load dataset (Replace this with your dataset loading code)
# Assuming you have a DataFrame with columns: bedrooms, bathrooms, square_footage, location, etc.
# X represents features, y_price represents rental price label, and y_area represents area label
# Replace "features.csv" and "labels.csv" with your dataset file paths
X = pd.read_csv("features.csv")
y_price = pd.read_csv("price_labels.csv")
y_area = pd.read_csv("area_labels.csv")

# Split the dataset into training and testing sets for each label
X_train_price, X_test_price, y_train_price, y_test_price = train_test_split(X, y_price, test_size=0.2, random_state=42)
X_train_area, X_test_area, y_train_area, y_test_area = train_test_split(X, y_area, test_size=0.2, random_state=42)

# Model training and evaluation for rental price prediction
# Linear Regression model
lr_price = LinearRegression()
lr_price.fit(X_train_price, y_train_price)
lr_price_predictions = lr_price.predict(X_test_price)
lr_price_mse = mean_squared_error(y_test_price, lr_price_predictions)

# Random Forest Regression model
rf_price = RandomForestRegressor()
```

```
rf_price.fit(X_train_price, y_train_price.values.ravel())

rf_price_predictions = rf_price.predict(X_test_price)

rf_price_mse = mean_squared_error(y_test_price, rf_price_predictions)


print("Mean Squared Error for Rental Price Prediction:")

print("Linear Regression:", lr_price_mse)

print("Random Forest Regression:", rf_price_mse)


# Model training and evaluation for area prediction

# Linear Regression model

lr_area = LinearRegression()

lr_area.fit(X_train_area, y_train_area)

lr_area_predictions = lr_area.predict(X_test_area)

lr_area_mse = mean_squared_error(y_test_area, lr_area_predictions)


# Random Forest Regression model

rf_area = RandomForestRegressor()

rf_area.fit(X_train_area, y_train_area.values.ravel())

rf_area_predictions = rf_area.predict(X_test_area)

rf_area_mse = mean_squared_error(y_test_area, rf_area_predictions)


print("\nMean Squared Error for Area Prediction:")

print("Linear Regression:", lr_area_mse)

print("Random Forest Regression:", rf_area_mse)
```

2

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.linear_model import Ridge, Lasso

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, make_scorer

import matplotlib.pyplot as plt


# Load dataset (Replace this with your dataset loading code)

# Assuming you have a DataFrame with columns: bedrooms, bathrooms, square_footage, location, etc.

# X represents features, y_price represents rental price label, and y_area represents area label

# Replace "features.csv" and "labels.csv" with your dataset file paths

X = pd.read_csv("features.csv")

y_price = pd.read_csv("price_labels.csv")

y_area = pd.read_csv("area_labels.csv")


# Split the dataset into training and testing sets for each label

X_train_price, X_test_price, y_train_price, y_test_price = train_test_split(X, y_price, test_size=0.2, random_state=42)

X_train_area, X_test_area, y_train_area, y_test_area = train_test_split(X, y_area, test_size=0.2, random_state=42)


# Define a custom scorer for GridSearchCV

mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)


# Create a pipeline for scaling and model

pipeline = Pipeline([

    ('scaler', StandardScaler()),
    ('regressor', Ridge())  # Change this to the model you want to use
])

# Define hyperparameters to tune
param_grid = {
    'regressor__alpha': [0.1, 1.0, 10.0],
    # Add more hyperparameters as needed for the chosen model
}

# Hyperparameter tuning for rental price prediction
grid_search_price = GridSearchCV(pipeline, param_grid, cv=5, scoring=mse_scorer)
grid_search_price.fit(X_train_price, y_train_price.values.ravel())

best_model_price = grid_search_price.best_estimator_

# Hyperparameter tuning for area prediction
grid_search_area = GridSearchCV(pipeline, param_grid, cv=5, scoring=mse_scorer)
grid_search_area.fit(X_train_area, y_train_area.values.ravel())

best_model_area = grid_search_area.best_estimator_

# Evaluate models
def evaluate_model(model, X_test, y_test):
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    return mse

mse_price = evaluate_model(best_model_price, X_test_price, y_test_price)
mse_area = evaluate_model(best_model_area, X_test_area, y_test_area)

print("Mean Squared Error for Rental Price Prediction:", mse_price)
print("Mean Squared Error for Area Prediction:", mse_area)


# Plot learning curves to diagnose bias/variance issues
def plot_learning_curve(model, X_train, y_train, title):
    train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=5,
scoring=mse_scorer)
    train_scores_mean = -np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = -np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure()
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Mean Squared Error")
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training error")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation error")

    plt.legend(loc="best")
    return plt
```

```
plot_learning_curve(best_model_price, X_train_price, y_train_price.values.ravel(), "Learning Curve - Rental Price Prediction")

plot_learning_curve(best_model_area, X_train_area, y_train_area.values.ravel(), "Learning Curve - Area Prediction")


plt.show()
```

3

```
import joblib


# Save the best model to a file

joblib.dump(best_model_price, 'best_model_price.pkl')

joblib.dump(best_model_area, 'best_model_area.pkl')


# Convey the basis for choosing the best model

print("\nBest Model Selection:")

print("For Rental Price Prediction, the best model chosen is:", best_model_price.named_steps['regressor'].__class__.__name__)

print("Based on hyperparameter tuning and evaluation results.")

print("\nFor Area Prediction, the best model chosen is:", best_model_area.named_steps['regressor'].__class__.__name__)

print("Based on hyperparameter tuning and evaluation results.")
```