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

from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Set plotting style
sns.set(style="whitegrid")

# 3. DATA LOADING & INITIAL EXPLORATION

# Read the CSV file (update the path as needed)
df = pd.read_csv("/content/drive/MyDrive/Machine Learning/COMP1816_Housing_Dataset_Regression.csv")

# Check for null values in each column and overall
null_counts = df.isnull().sum()
print("Null values per column:\n", null_counts)

total_nulls = df.isnull().sum().sum()
print(f"\nTotal number of null values: {total_nulls}")

# Display rows with any null values
print("\nRows with null values:")
print(df[df.isnull().any(axis=1)])

# Drop rows with null values and display dataframe info
df.dropna(inplace=True)
df.info()


# 4. DATA SPLITTING

# Separate features and target variable
X = df.drop(['median_house_value'], axis=1)
y = df['median_house_value']

# Split data into train and test sets (using last 190 samples for testing)
X_train, X_test = X.iloc[:-190], X.iloc[-190:]
y_train, y_test = y.iloc[:-190], y.iloc[-190:]

# Combine X and y for training set for easier preprocessing
train_data = pd.concat([X_train, y_train], axis=1)


# 5. EXPLORATORY DATA ANALYSIS (EDA)

# Plot histogram for training data
plt.figure(figsize=(15, 9))
train_data.hist(bins=50, figsize=(15, 9))
plt.tight_layout()
plt.show()

# Calculate and plot correlation heatmap for numerical features
numerical_features = train_data.select_dtypes(include=['number'])
correlation_matrix = numerical_features.corr()

plt.figure(figsize=(15, 9))
sns.heatmap(correlation_matrix, annot=True, cmap='YlGnBu')
plt.title("Correlation Heatmap (Training Data)")
plt.show()

# Scatter plot for geographic distribution colored by median house value
plt.figure(figsize=(15, 9))
sns.scatterplot(x='longitude', y='latitude', hue='median_house_value', data=train_data, palette='coolwarm')
plt.title("Geographic Scatter Plot")
plt.show()

# 6. DATA PREPROCESSING & FEATURE ENGINEERING

# --- Preprocessing for Training Data ---
# Log-transform selected numerical features (add 1 to avoid log(0))
for col in ['total_rooms', 'total_bedrooms', 'population', 'households']:
    train_data[col] = np.log(train_data[col] + 1)
```

```
# Check the distribution of 'ocean_proximity'
print("\nOcean Proximity Value Counts (Training):")
print(train_data['ocean_proximity'].value_counts())

# Convert categorical feature 'ocean_proximity' to dummy variables
train_data = train_data.join(pd.get_dummies(train_data.ocean_proximity)).drop(['ocean_proximity'], axis=1)

# Create new features
train_data['bedroom_ratio'] = train_data['total_bedrooms'] / train_data['total_rooms']
train_data['household_rooms'] = train_data['total_rooms'] / train_data['households']

# Plot updated correlation heatmap
plt.figure(figsize=(15, 9))
sns.heatmap(train_data.corr(), annot=True, cmap='YlGnBu')
plt.title("Correlation Heatmap with Engineered Features (Training Data)")
plt.show()


# 7. PREPARE DATA FOR MODELING

# Separate features and target from training data
X_train_proc = train_data.drop(['median_house_value'], axis=1)
y_train_proc = train_data['median_house_value']

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_proc)

# 8. TRAIN A LINEAR REGRESSION MODEL

reg = LinearRegression()
reg.fit(X_train_scaled, y_train_proc)


# 9. PREPROCESS THE TEST DATA

# Combine test features and target for preprocessing
test_data = X_test.join(y_test)

# Apply log transformation to numerical features
for col in ['total_rooms', 'total_bedrooms', 'population', 'households']:
    test_data[col] = np.log(test_data[col] + 1)

# Convert categorical 'ocean_proximity' to dummy variables
test_data = test_data.join(pd.get_dummies(test_data.ocean_proximity)).drop(['ocean_proximity'], axis=1)

# Create new features for test data
test_data['bedroom_ratio'] = test_data['total_bedrooms'] / test_data['total_rooms']
test_data['household_rooms'] = test_data['total_rooms'] / test_data['households']

# Ensure the test data has the same feature columns as training data
X_test_proc = test_data.reindex(columns=X_train_proc.columns, fill_value=0)
y_test_proc = test_data['median_house_value']

# Scale test features using the same scaler as training data
X_test_scaled = scaler.transform(X_test_proc)


# 10. EVALUATE THE LINEAR REGRESSION MODEL

y_pred_lr = reg.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test_proc, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test_proc, y_pred_lr)

print("\nLinear Regression Evaluation:")
print(f"Mean Squared Error (MSE): {mse_lr:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_lr:.2f}")
print(f"R-squared (R²): {r2_lr:.2f}")


# 11. TRAIN A RANDOM FOREST MODEL (Directly on Test Data for Demo)

forest = RandomForestRegressor()
forest.fit(X_test_scaled, y_test_proc)


# 12. GRID SEARCH FOR RANDOM FOREST HYPERPARAMETERS

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': [2, 4, 6, 8],
    'min_samples_split': [2, 4],
```

```
    'max_depth': [None, 4, 8]
}

grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=5,
                           scoring="neg_mean_squared_error", return_train_score=True)
grid_search.fit(X_train_scaled, y_train_proc)

best_rf = grid_search.best_estimator_
print("\nGrid Search Best Estimator (Random Forest):")
print(best_rf)
print(f"Test Score (R²): {best_rf.score(X_test_scaled, y_test_proc):.2f}")


# 13. MODEL COMPARISON: DECISION TREE, RANDOM FOREST, GRADIENT BOOSTING

# Define models
models = {
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, random_state=42)
}

results = {}

# Evaluate each model
for name, model in models.items():
    model.fit(X_train_scaled, y_train_proc)
    y_pred = model.predict(X_test_scaled)
    results[name] = {
        "MSE": mean_squared_error(y_test_proc, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test_proc, y_pred)),
        "R²": r2_score(y_test_proc, y_pred)
    }

print("\nModel Comparison:")
print("{:<20} {:<10} {:<10} {:<10}".format('Model', 'MSE', 'RMSE', 'R²'))
for model_name, metrics in results.items():
    print("{:<20} {:<10.2f} {:<10.2f} {:<10.2f}".format(
        model_name, metrics['MSE'], metrics['RMSE'], metrics['R²']
    ))

# Print Linear Regression results for reference
print("\nLinear Regression Reference:")
print("{:<20} {:<10.2f} {:<10.2f} {:<10.2f}".format(
    "Linear Regression", mse_lr, rmse_lr, r2_lr
))


# 14. FINAL HYPERPARAMETER TUNING & EVALUATION WITH RANDOM FOREST

# Definining a simpler parameter grid for final tuning
param_grid_final = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5]
}

grid_search_final = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid_final,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

grid_search_final.fit(X_train_scaled, y_train_proc)
best_model = grid_search_final.best_estimator_

print("\nFinal Grid Search Best Parameters:")
print(grid_search_final.best_params_)
print(f"Best RMSE (Training CV): {np.sqrt(-grid_search_final.best_score_):.2f}")

# Feature Importance
importances = best_model.feature_importances_
feature_names = X_train_proc.columns
sorted_idx = np.argsort(importances)[::-1]

plt.figure(figsize=(12, 6))
plt.bar(range(len(feature_names)), importances[sorted_idx], align='center')
plt.xticks(range(len(feature_names)), feature_names[sorted_idx], rotation=90)
plt.title("Random Forest Feature Importances")
plt.xlabel("Features")
plt.ylabel("Importance Score")
```

```
plt.tight_layout()
plt.show()

#Residual Analysis
y_pred_best = best_model.predict(X_test_scaled)
residuals = y_test_proc - y_pred_best

plt.figure(figsize=(10, 6))
plt.scatter(y_pred_best, residuals, alpha=0.5)
plt.hlines(y=0, xmin=y_pred_best.min(), xmax=y_pred_best.max(), colors='red', linestyles='--')
plt.title("Residual Plot for Best Random Forest Model")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals (Actual - Predicted)")
plt.grid(True)
plt.show()

#Final Evaluation
mse_final = mean_squared_error(y_test_proc, y_pred_best)
rmse_final = np.sqrt(mse_final)
r2_final = best_model.score(X_test_scaled, y_test_proc)

print("\nFinal Evaluation Metrics (Best Random Forest):")
print(f"MSE: {mse_final:.2f}")
print(f"RMSE: {rmse_final:.2f}")
print(f"R² Score: {r2_final:.2f}")
```

```
Null values per column:
 No.                  0
longitude            0
latitude             0
housing_median_age   0
total_rooms          0
total_bedrooms      12
population           0
households           0
median_income        0
median_house_value   0
ocean_proximity      7
dtype: int64

Total number of null values: 19

Rows with null values:
     No.  longitude  latitude  housing_median_age  total_rooms  \
72    73    -122.08     37.88                  26         2947
93    94    -119.80     36.75                  46         2625
98    99    -119.82     36.81                  25         3305
168  169    -118.28     34.25                  29         2559
236  237    -118.38     34.05                  49          702
548  549    -117.87     33.83                  27         2287
585  586    -121.26     38.74                  22         7173
595  596    -121.04     39.00                  21         4059
603   94    -119.80     36.75                  46         2625
608   99    -119.82     36.81                  25         3305
621  622    -116.21     33.75                  22          894
740  741    -117.02     32.66                  19          771
786  787    -122.45     37.77                  52         2602
792  793    -122.50     37.75                  45         1620
821  822    -122.39     37.59                  32         4497
893  894    -121.94     36.97                  31         1738
981  982    -119.06     34.24                  21         7436
988  989    -118.69     34.18                  11         1177
992  993    -121.52     38.57                  43         2360

     total_bedrooms  population  households  median_income  \
72              NaN         825         626         2.9330
93            593.0        1368         551         1.5273
98              NaN        1149         500         5.0698
168             NaN        1886         769         2.6036
236             NaN         458         187         4.8958
548             NaN        1140         351         5.6163
585          1314.0        3526        1316         3.3941
595           730.0        1874         693         4.8051
603           593.0        1368         551         1.5273
608             NaN        1149         500         5.0698
621             NaN         830         202         3.0673
740             NaN         376         108         6.6272
786             NaN        1330         647         3.5435
792             NaN         941         328         4.3859
821             NaN        1846         715         6.1323
893           422.0         746         355         2.5172
981           984.0        2982         988         7.6775
988             NaN         415         119        10.0472
992           471.0        1041         452         2.8900

     median_house_value ocean_proximity
72                85000        NEAR BAY
93                59000             NaN
98               150900          INLAND
168              162100       <1H OCEAN
236              333600       <1H OCEAN
548              231000       <1H OCEAN
585              135900             NaN
595              174300             NaN
603               59000             NaN
608              150900          INLAND
621               68200          INLAND
740              273600      NEAR OCEAN
786              278600        NEAR BAY
792              270200      NEAR OCEAN
821              500001      NEAR OCEAN
893              330800             NaN
981              391200             NaN
988              500001       <1H OCEAN
992               86200             NaN
<class 'pandas.core.frame.DataFrame'>
Index: 981 entries, 0 to 999
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   No.                 981 non-null    int64
```