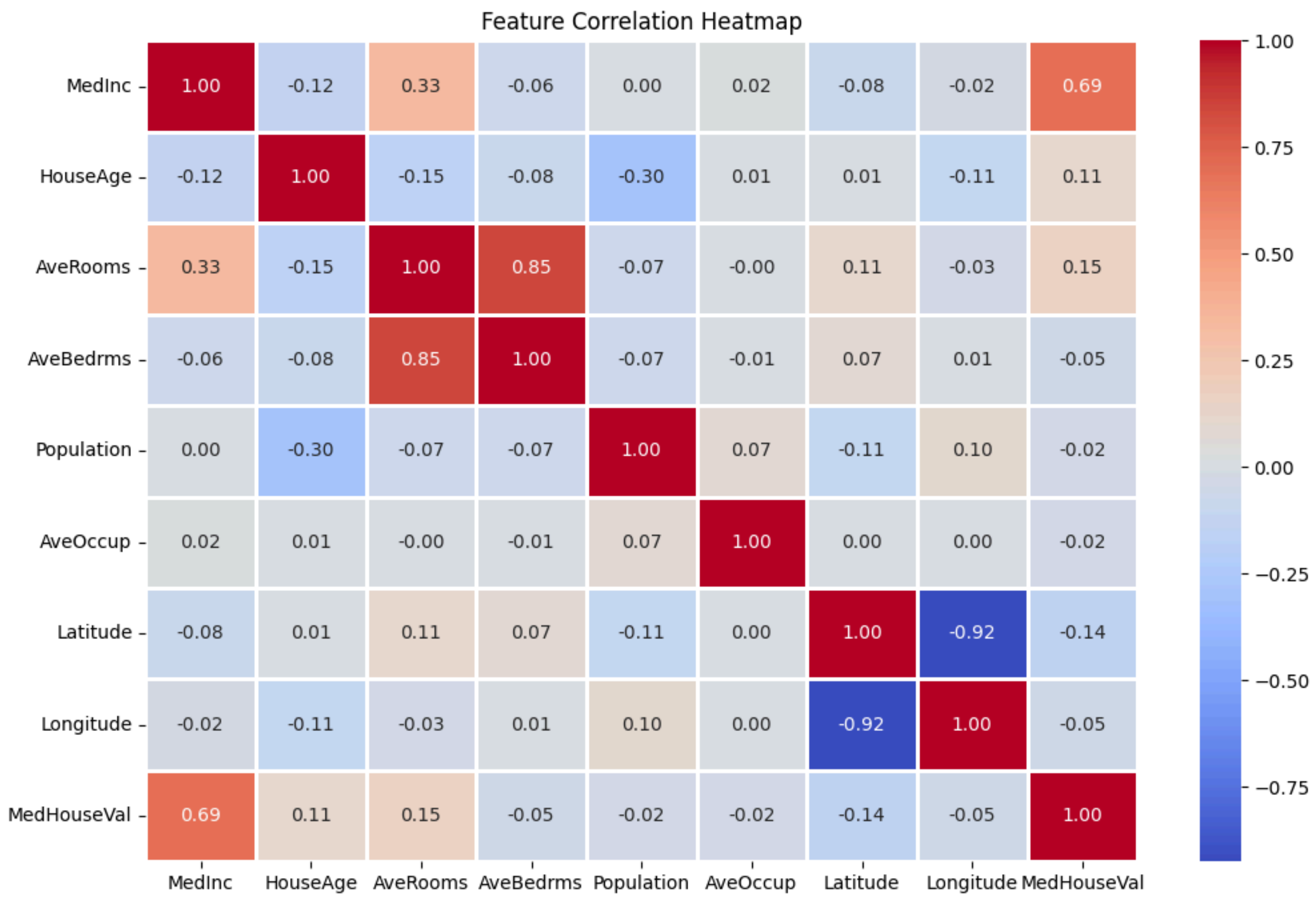
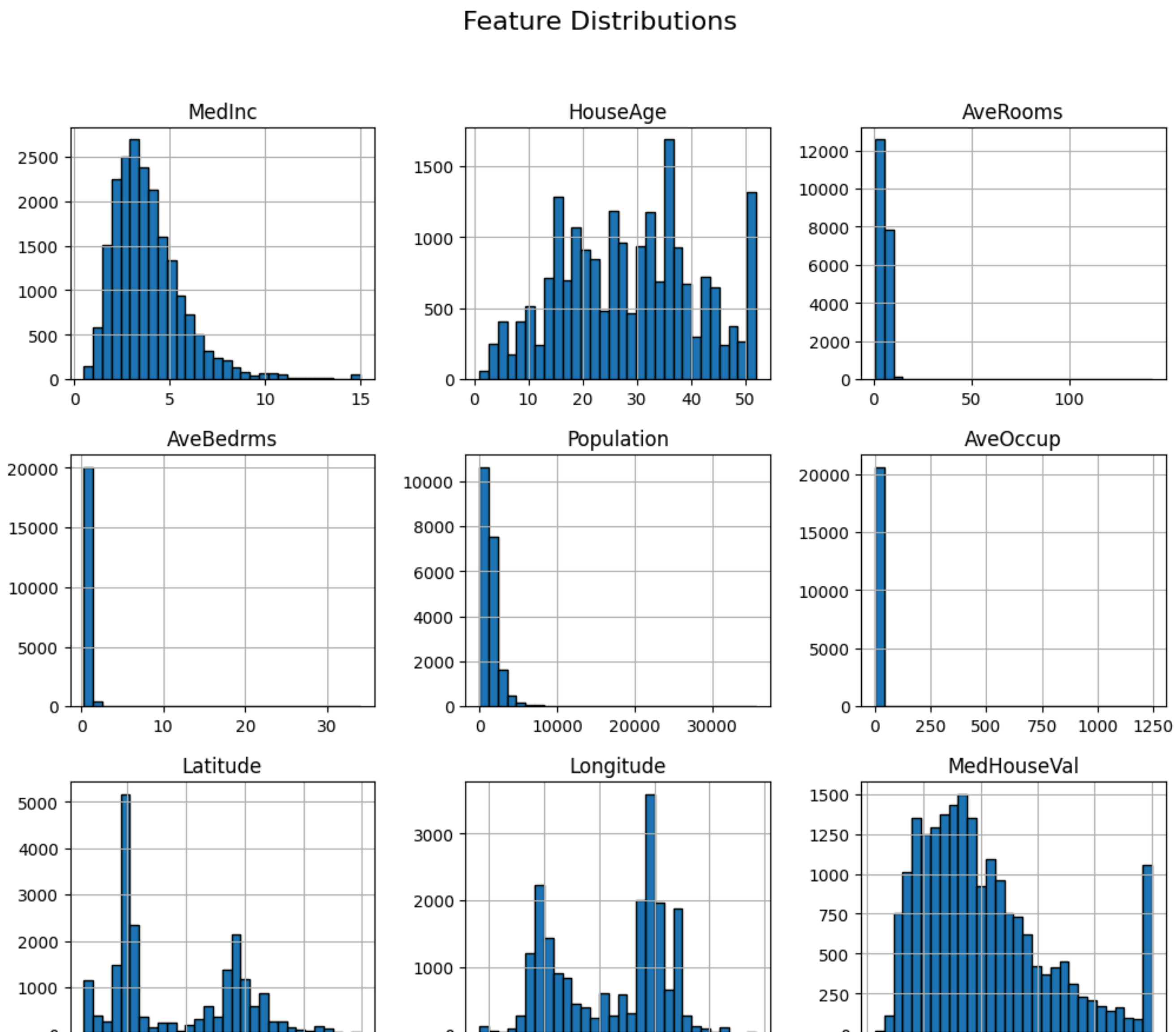
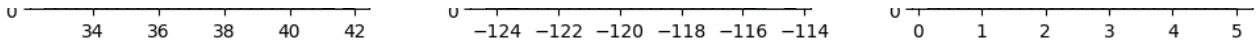
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, mean_absolute_percentage_error, median_absolute_error
from sklearn.datasets import fetch_california_housing

#  We will load real-world dataset (California Housing Dataset)
data = fetch_california_housing(as_frame=True)
df = data.frame

# Handle missing values
df.dropna(inplace=True)

# Exploratory Data Analysis (EDA)
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1)
plt.title("Feature Correlation Heatmap")
plt.show()

plt.figure(figsize=(12, 6))
df.hist(figsize=(12, 10), bins=30, edgecolor='black')
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()

# Defining features and target
X = df.drop(columns=['MedHouseVal'])
y = df['MedHouseVal'] * 100000  # Convert to similar scale as house prices

# Encoding categorical variables (No categorical variables in this dataset, so only scaling needed)
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), X.columns)
])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model 1: Linear Regression Pipeline
lr_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])
lr_pipeline.fit(X_train, y_train)
lr_preds = lr_pipeline.predict(X_test)

# Model 2: Random Forest Regressor with Hyperparameter Tuning
param_grid_rf = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [None, 10, 20],
    'model__min_samples_split': [2, 5, 10]
}

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

grid_search_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=3, n_jobs=-1)
grid_search_rf.fit(X_train, y_train)
rf_preds = grid_search_rf.best_estimator_.predict(X_test)

# Model 3: Decision Tree Regressor
dt_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', DecisionTreeRegressor(random_state=42))
])
dt_pipeline.fit(X_train, y_train)
dt_preds = dt_pipeline.predict(X_test)

# Model 4: Neural Network (MLP Regressor)
nn_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=1000, random_state=42))
```

```
])
nn_pipeline.fit(X_train, y_train)
nn_preds = nn_pipeline.predict(X_test)

# Model Evaluation
def evaluate_model(name, y_true, y_pred):
    print(f"{name} Performance:")
    print(f"MAE: {mean_absolute_error(y_true, y_pred)}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_true, y_pred))}")
    print(f"R² Score: {r2_score(y_true, y_pred)}")
    print(f"MAPE: {mean_absolute_percentage_error(y_true, y_pred)}")
    print(f"Median AE: {median_absolute_error(y_true, y_pred)}\n")

evaluate_model("Linear Regression", y_test, lr_preds)
evaluate_model("Random Forest", y_test, rf_preds)
evaluate_model("Decision Tree", y_test, dt_preds)
evaluate_model("Neural Network", y_test, nn_preds)

# Best Parameters for Random Forest
print("Best Parameters for Random Forest:", grid_search_rf.best_params_)

# Model Performance Summary
summary = """
Based on the provided metrics, the Random Forest model appears to be the best performing model overall.
It has the lowest MAE, RMSE, and MAPE, and the highest R² score.
This indicates that it makes the most accurate predictions with the least amount of error, and it explains a larger proportion of the va

While the Neural Network has the lowest Median AE, Random Forest outperforms it in most metrics, including R-squared, which is a key inc
 Therefore, in this case, Random Forest would likely be the preferred choice for predicting house prices.
"""
print(summary)
```



<Figure size 1200x600 with 0 Axes>






```
Linear Regression Performance:
MAE: 53320.013049565656
RMSE: 74558.13830127762
R² Score: 0.575787706032451
MAPE: 0.319521874136152
Median AE: 41023.30008496222

Random Forest Performance:
MAE: 32666.261956556853
RMSE: 50365.40771114935
R² Score: 0.8064211756763489
MAPE: 0.18848927137352098
Median AE: 20037.999999999996

Decision Tree Performance:
MAE: 45463.60658914729
RMSE: 70641.87845094097
R² Score: 0.6191818681083447
MAPE: 0.24909150018637158
Median AE: 25700.0

Neural Network Performance:
MAE: 41000.23863931191
RMSE: 59001.55371819165
R² Score: 0.7343437507705559
MAPE: 0.24059355439784808
Median AE: 29383.08741765647

Best Parameters for Random Forest: {'model__max_depth': None, 'model__min_samples_split': 2, 'model__n_estimators': 300}

Based on the provided metrics, the Random Forest model appears to be the best performing model overall.
It has the lowest MAE, RMSE, and MAPE, and the highest R² score.
This indicates that it makes the most accurate predictions with the least amount of error, and it explains a larger proportion of

While the Neural Network has the lowest Median AE, Random Forest outperforms it in most metrics, including R-squared, which is a k
 Therefore, in this case, Random Forest would likely be the preferred choice for predicting house prices.

/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optir
  warnings.warn(
```

Start coding or generate with AI.

```
import pickle

# Save Linear Regression model
filename = 'linear_regression_model.pkl'
pickle.dump(lr_pipeline, open(filename, 'wb'))

# Save Random Forest model
filename = 'random_forest_model.pkl'
pickle.dump(grid_search_rf.best_estimator_, open(filename, 'wb'))

# Save Decision Tree model
filename = 'decision_tree_model.pkl'
pickle.dump(dt_pipeline, open(filename, 'wb'))

# Save Neural Network model
filename = 'neural_network_model.pkl'
pickle.dump(nn_pipeline, open(filename, 'wb'))
```

```
import pickle

# Load Linear Regression model
filename = 'linear_regression_model.pkl'
loaded_lr_model = pickle.load(open(filename, 'rb'))

# ... (Similarly for other models)
```

Start coding or generate with AI.

```
!pip install Flask==2.3.3
```

```
Collecting Flask==2.3.3
  Downloading flask-2.3.3-py3-none-any.whl.metadata (3.6 kB)
Requirement already satisfied: Werkzeug>=2.3.7 in /usr/local/lib/python3.11/dist-packages (from Flask==2.3.3) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask==2.3.3) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask==2.3.3) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from Flask==2.3.3) (8.1.8)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages (from Flask==2.3.3) (1.9.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->Flask==2.3.3) (3.0.2
Downloading flask-2.3.3-py3-none-any.whl (96 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.1/96.1 kB 3.8 MB/s eta 0:00:00
Installing collected packages: Flask
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.0
    Uninstalling Flask-3.1.0:
      Successfully uninstalled Flask-3.1.0
Successfully installed Flask-2.3.3
```

```
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)

# Load your saved models
linear_regression_model = pickle.load(open('linear_regression_model.pkl', 'rb'))
random_forest_model = pickle.load(open('random_forest_model.pkl', 'rb'))
decision_tree_model = pickle.load(open('decision_tree_model.pkl', 'rb'))
neural_network_model = pickle.load(open('neural_network_model.pkl', 'rb'))

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()  # Get data from the request

    # Extract features from the data
    features = [
        data['square_feet'],
        data['bedrooms'],
        data['bathrooms'],
        data['location_score'],
        data['year_built'],
        data['garage_size'],
        data['distance_to_city_center'],
        data['neighborhood']
    ]

    # Make predictions using your loaded models
    linear_regression_prediction = linear_regression_model.predict([features])[0]
    random_forest_prediction = random_forest_model.predict([features])[0]
    decision_tree_prediction = decision_tree_model.predict([features])[0]
    neural_network_prediction = neural_network_model.predict([features])[0]
```