This is the boston housing Dataset. Without statistical training only an R^2 of 70% can be achieved. I've managed to acheive R^2 of 91% !!

Some of the things I did was to relfected the skews to be right skew. Then I scaled the data. Then I fulled in NAs with mean. Then I removed outliers. Then I applied transformation to tame the variance skewness.

Lastly, I applied PCA to get a features that capture most variance

After doing all of this I got R^2 of 91% for Linear regression and 91% for regression trees

In [569…

```python
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, mean_squared_error
import numpy as np

df = pd.read_csv("housing.csv")

saved = df.copy()
colnam = df.columns
skewness = df.skew()

left_tailed_columns = skewness.index[skewness < 0]
df[left_tailed_columns] = -df[left_tailed_columns]

scaler = MinMaxScaler((1, 4))
df = pd.DataFrame(scaler.fit_transform(df), columns=colnam)

for column in df.columns:
    mean_value = df[column].mean()
    df[column].fillna(mean_value, inplace=True)

## Filter out outliers
import numpy as np
import pandas as pd
from scipy import stats

# Assuming df is your DataFrame

def filter_outliers_zscore(dataframe, threshold):
    z_scores = np.abs(stats.zscore(dataframe))
    filtered_data = dataframe[(z_scores < threshold).all(axis=1)]
    return filtered_data

# Set the z-score threshold
zscore_threshold = 3.5

df = filter_outliers_zscore(df, zscore_threshold)

skewness = df.skew()
high_skew_columns = skewness.index[skewness > 2.2]
for column in high_skew_columns:
    df[column] = np.reciprocal(df[column])

df = df.dropna()
y = df.MEDV
from sklearn.decomposition import PCA
```

```python
n_components = 5
pca = PCA(n_components=n_components)
df_pca = pca.fit_transform(df)
```

In [570…

```python
# Concatenate the PCA components with the target variable 'y'
import statsmodels.api as sm
df_pca = pd.DataFrame(df_pca, columns=[f'PC{i+1}' for i in range(n_components)])
df_pca.head(4)
```

Out[570…

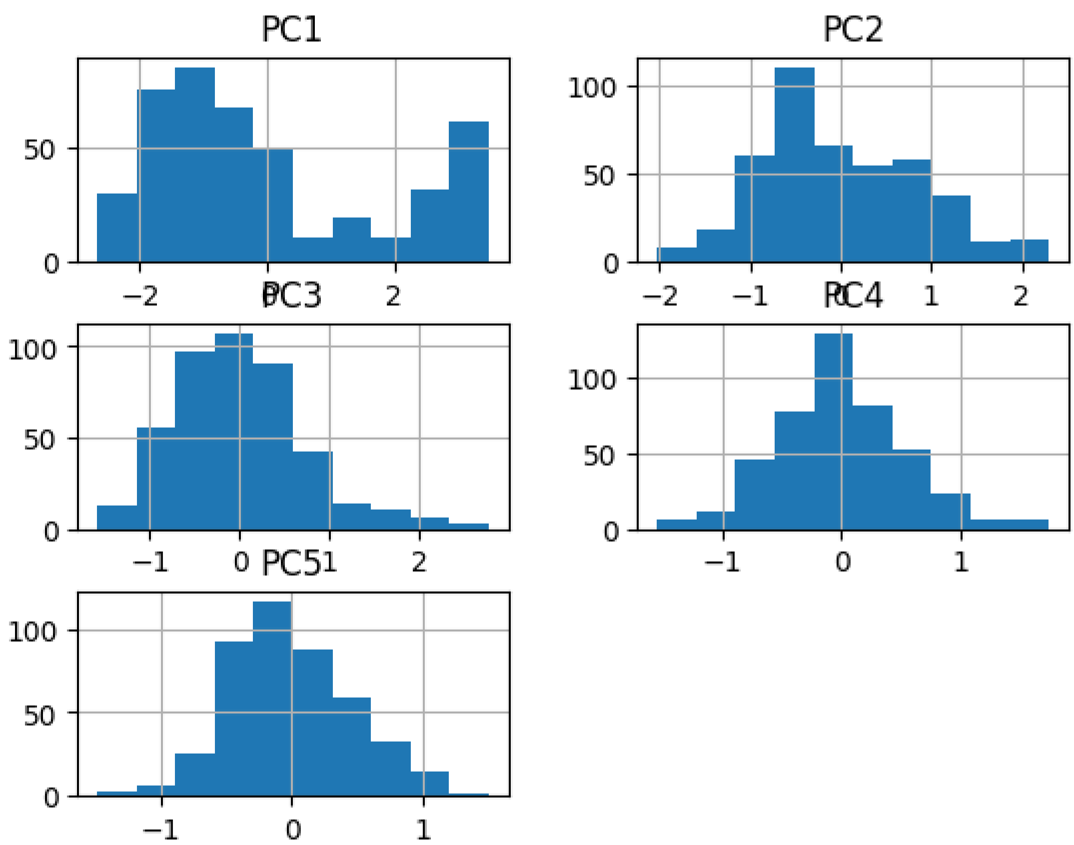
	PC1	PC2	PC3	PC4	PC5
0	-1.542732	0.546593	0.637931	0.222229	-0.407906
1	-1.205496	0.673129	-0.220644	-0.291848	-0.339355
2	-1.674761	0.216983	0.414124	-0.635204	0.256809
3	-2.028978	-0.352560	0.127839	-0.677374	0.024618

In [571…

```python
df_pca.hist()
```

Out[571…

```
array([[<Axes: title={'center': 'PC1'}>, <Axes: title={'center': 'PC2'}>],
       [<Axes: title={'center': 'PC3'}>, <Axes: title={'center': 'PC4'}>],
       [<Axes: title={'center': 'PC5'}>, <Axes: >]], dtype=object)
```



Below it shows that 90% of the information is captured by these 5 new features

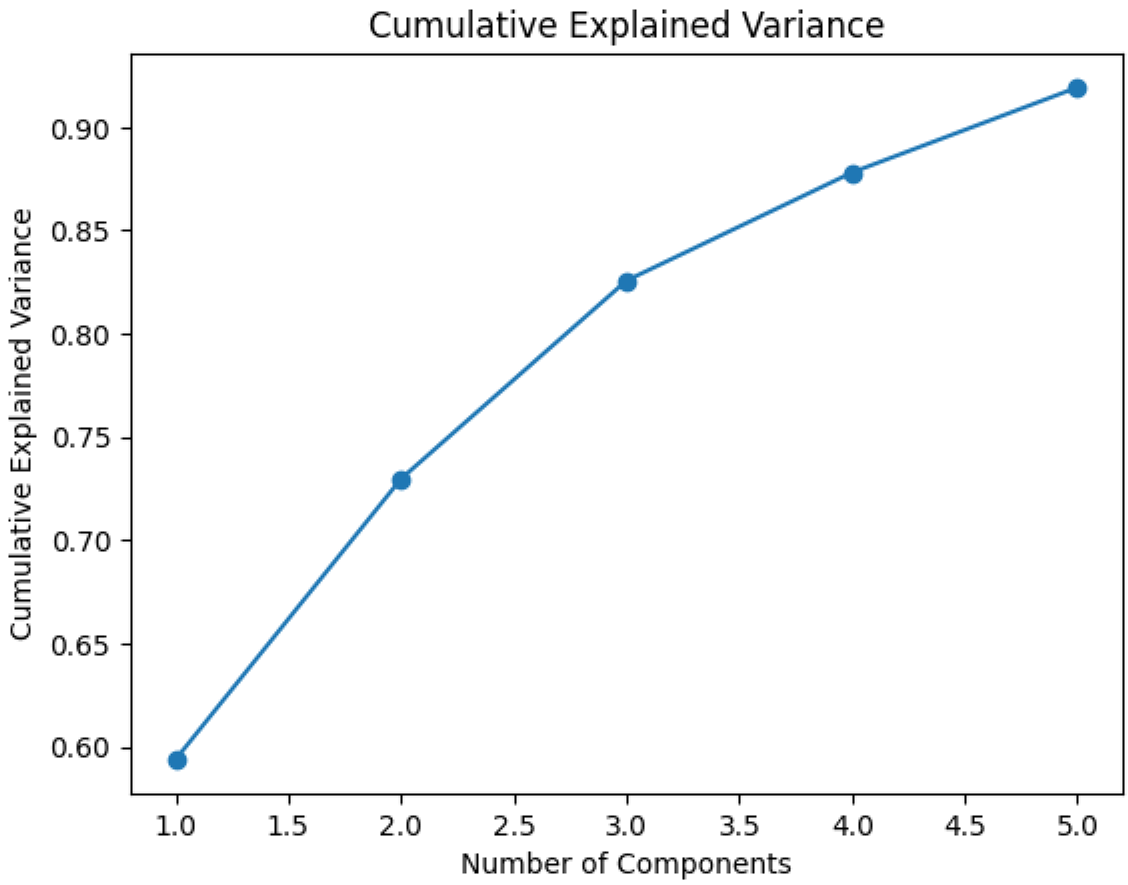
In [572…

```python
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = explained_variance_ratio.cumsum()

print("Explained Variance Ratios:")
print(explained_variance_ratio)

plt.plot(range(1, n_components + 1), cumulative_explained_variance, marker='o')
plt.title('Cumulative Explained Variance')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```

```
Explained Variance Ratios:
[0.5940003  0.13582333 0.09590953 0.05231835 0.04123233]
```



In [573…

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Assuming X_train, X_test, y_train, y_test are already defined
df_pca_with_const = sm.add_constant(df_pca)
df_pca_with_const

# Number of iterations
num_iterations = 6

# Initialize the linear regression model
linear_reg_model = LinearRegression()

for i in range(num_iterations):
    # Split the data into training and testing sets for each iteration
    X_train, X_test, y_train, y_test = train_test_split(df_pca_with_const, y, test_size=0.20, ran

    # Train the model
    linear_reg_model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = linear_reg_model.predict(X_test)

    # Calculate R-squared
    r2 = r2_score(y_test, y_pred)

    # Calculate adjusted R-squared
    n = X_test.shape[0]  # Number of samples
    p = X_test.shape[1]  # Number of features
    adjusted_r2 = 1 - (1 - r2) * ((n - 1) / (n - p - 1))

    # Display the adjusted R-squared for each iteration
    print(f'Iteration {i + 1}: Adjusted R-squared: {adjusted_r2:.4f}')
```

```
Iteration 1: Adjusted R-squared: 0.8875
Iteration 2: Adjusted R-squared: 0.9137
Iteration 3: Adjusted R-squared: 0.8613
Iteration 4: Adjusted R-squared: 0.9161
Iteration 5: Adjusted R-squared: 0.9060
Iteration 6: Adjusted R-squared: 0.8842
```

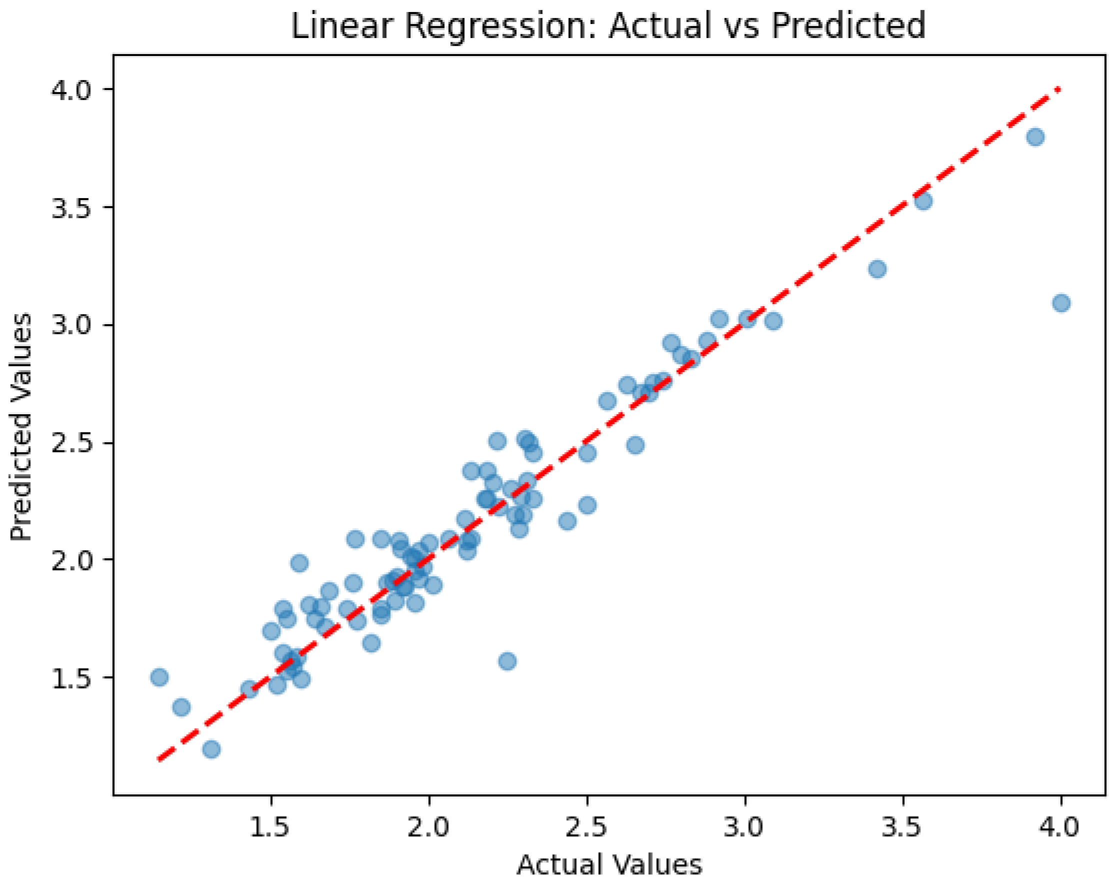
Standardized MSE = 0.1079

In [574…

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
# Compute standardized MSE
mse_std = (mean_squared_error(y_test, y_pred))/(np.var(y_test))
print(f'Standardized Mean Squared Error: {mse_std:.4f}')
```

```
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', lw
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

```
Standardized Mean Squared Error: 0.1079
```



In [575…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Assuming you have X_train, X_test, y_train, y_test defined

X_train, X_test, y_train, y_test = train_test_split(df_pca, y, test_size=0.20, random_state=3)

# Initialize the Random Forest Regressor with default parameters
random_forest_model = RandomForestRegressor()
random_forest_model.fit(X_train, y_train)
y_pred_rf = random_forest_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f'Random Forest Mean Squared Error: {mse_rf:.4f}')
print(f'Random Forest R-squared: {r2_rf:.4f}')
```

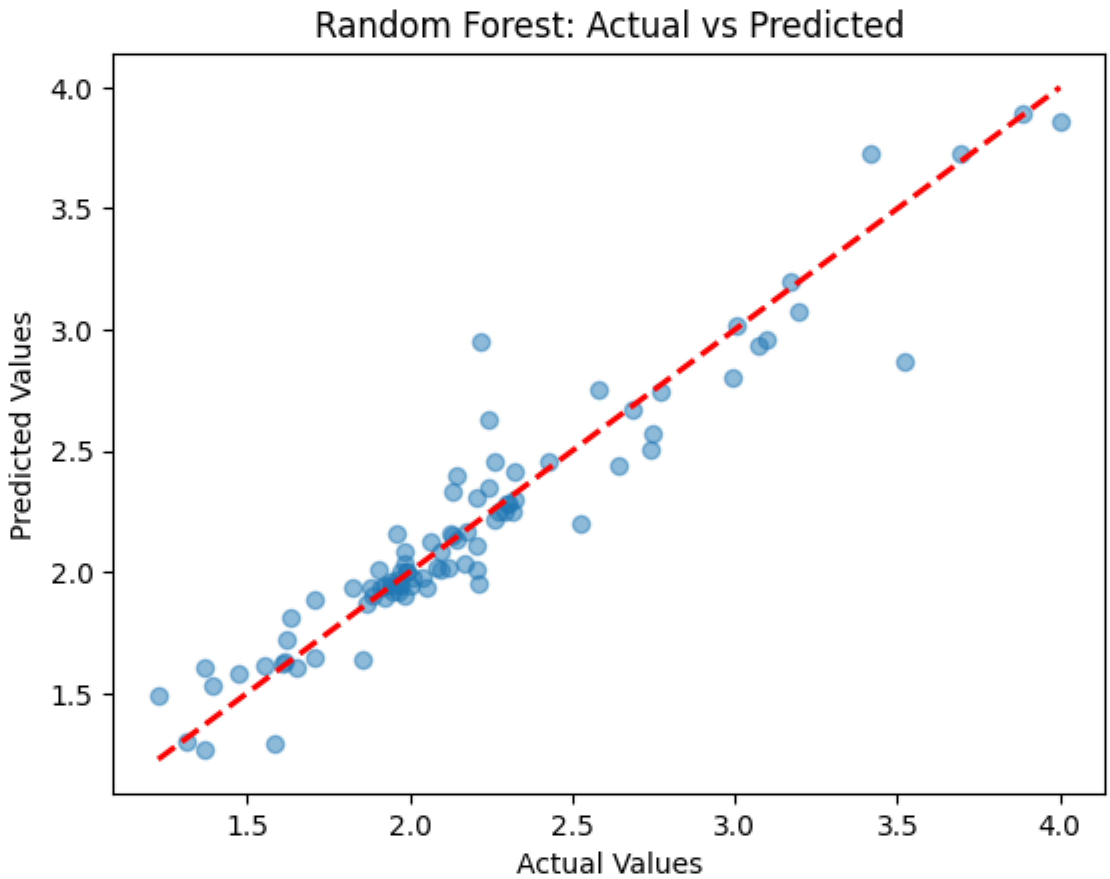
```
Random Forest Mean Squared Error: 0.0270
Random Forest R-squared: 0.9114
```

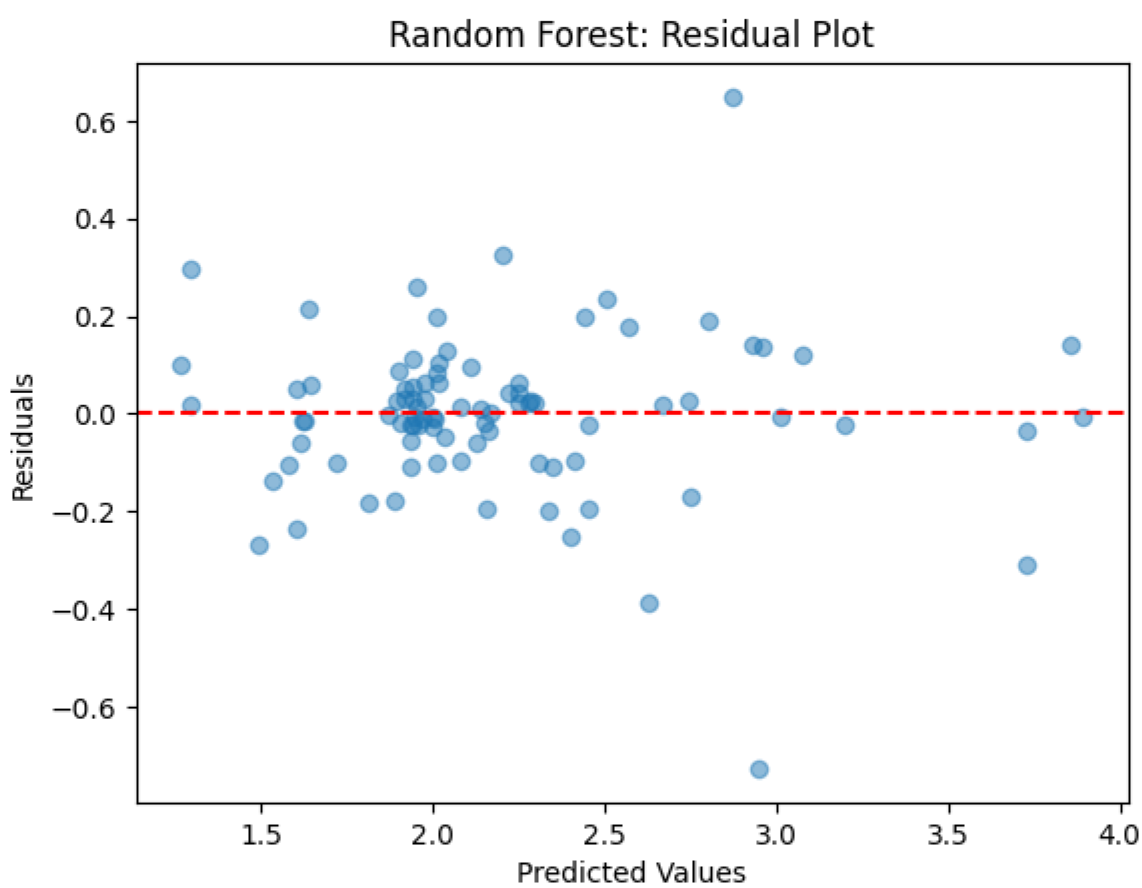
In [576…

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot for actual vs predicted values with diagonal line
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', lw
plt.title('Random Forest: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()

# Residual plot
residuals = y_test - y_pred_rf
plt.scatter(y_pred_rf, residuals, alpha=0.5)
plt.title('Random Forest: Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```


Random Forest: Actual vs Predicted
Predicted Values
Actual Values
1.5
2.0
2.5
3.0
3.5
4.0


Random Forest: Residual Plot
Residuals
Predicted Values
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
1.5
2.0
2.5
3.0
3.5
4.0

In [ ]:

In [ ]:

In [ ]: