hw2_2

June 30, 2025

0.1

1

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV,␣
    ↪cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
    ↪accuracy_score

plt.style.use('ggplot')
sns.set_palette('viridis')
```

[2]:
```
boston_df = pd.read_csv('BostonHousing.csv')
print("Available features:", list(boston_df.columns))
```

```
Available features: ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MEDV', 'CAT. MEDV']
```

[3]:
```
boston_df.describe()
```

[3]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO       LSTAT  \
```

```
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534   12.653063
std     28.148861    2.105710    8.707259  168.537116    2.164946    7.141062
min      2.900000    1.129600    1.000000  187.000000   12.600000    1.730000
25%     45.025000    2.100175    4.000000  279.000000   17.400000    6.950000
50%     77.500000    3.207450    5.000000  330.000000   19.050000   11.360000
75%     94.075000    5.188425   24.000000  666.000000   20.200000   16.955000
max    100.000000   12.126500   24.000000  711.000000   22.000000   37.970000

             MEDV   CAT. MEDV
count  506.000000  506.000000
mean    22.532806    0.166008
std      9.197104    0.372456
min      5.000000    0.000000
25%     17.025000    0.000000
50%     21.200000    0.000000
75%     25.000000    0.000000
max     50.000000    1.000000
```

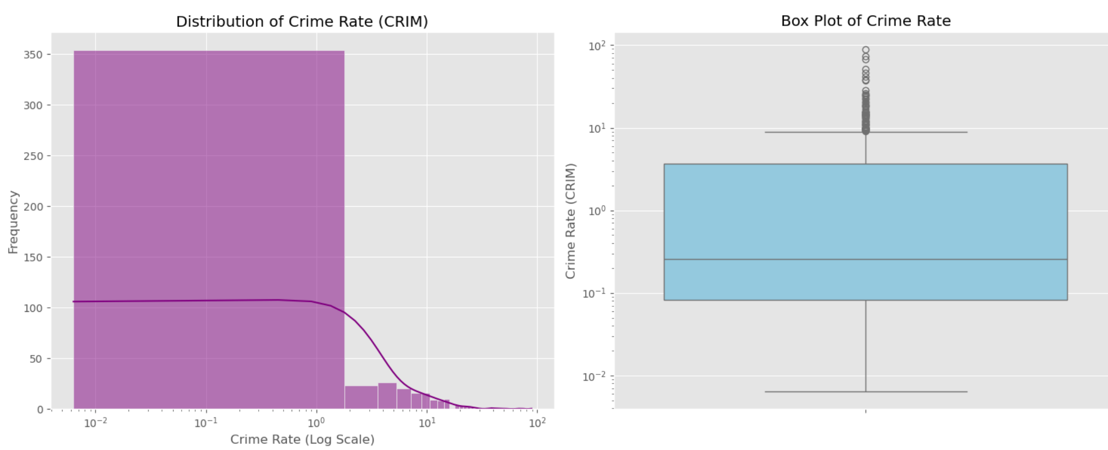
[4]:
```python
print(boston_df.dtypes)
```

```
CRIM          float64
ZN            float64
INDUS         float64
CHAS            int64
NOX           float64
RM            float64
AGE           float64
DIS           float64
RAD             int64
TAX             int64
PTRATIO       float64
LSTAT         float64
MEDV          float64
CAT. MEDV       int64
dtype: object
```

:2 (EDA) (CRIM)

- (CRIM) .
- .
- .
- .
- (percentiles) .

[5]:
```
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.histplot(boston_df['CRIM'], bins=50, kde=True, color='purple')
plt.title('Distribution of Crime Rate (CRIM)')
plt.xlabel('Crime Rate (Log Scale)')
plt.ylabel('Frequency')
plt.xscale('log')

plt.subplot(1, 2, 2)
sns.boxplot(y=boston_df['CRIM'], color='skyblue')
plt.title('Box Plot of Crime Rate')
plt.ylabel('Crime Rate (CRIM)')
plt.yscale('log')

plt.tight_layout()
plt.show()

print("Descriptive Statistics for CRIM:")
print(boston_df['CRIM'].describe())
```




```
Descriptive Statistics for CRIM:
count    506.000000
mean       3.613524
std        8.601545
min        0.006320
25%        0.082045
50%        0.256510
75%        3.677083
max       88.976200
Name: CRIM, dtype: float64
```

:3 (CRIM)

- CRIM) (.
- 33 66 .
-) (33 .

[6]:
```python
p33 = boston_df['CRIM'].quantile(0.33)
p66 = boston_df['CRIM'].quantile(0.66)

boston_df['crime_level'] = pd.cut(
    boston_df['CRIM'],
    bins=[-np.inf, p33, p66, np.inf],
    labels=['Low', 'Moderate', 'High']
)

print(f"Cutoff for Low/Moderate (33rd Percentile): {p33:.4f}")
print(f"Cutoff for Moderate/High (66th Percentile): {p66:.4f}")
print("\nDistribution of samples in each class (confirming balance):")
print(boston_df['crime_level'].value_counts(normalize=True).apply(lambda x:␣
 ↪f"{x:.1%}"))
```

```
Cutoff for Low/Moderate (33rd Percentile): 0.1126
Cutoff for Moderate/High (66th Percentile): 1.0757

Distribution of samples in each class (confirming balance):
crime_level
High        34.0%
Low         33.0%
Moderate    33.0%
Name: proportion, dtype: object
```

:4

4a: (CRIM)

- CRIM .

[7]:
```python
from sklearn.tree import DecisionTreeRegressor


from sklearn.preprocessing import LabelEncoder
df_encoded = boston_df.copy()
label_encoder = LabelEncoder()
boston_df['CRIM_CLASS_ENCODED'] = label_encoder.
 ↪fit_transform(df_encoded['crime_level'])
boston_df  = boston_df.drop(columns=['CRIM', 'crime_level'])
boston_df  = boston_df .rename(columns={'CRIM_CLASS_ENCODED': 'CRIM'})
```

```python
for cls, code in zip(label_encoder.classes_, label_encoder.
    ↪transform(label_encoder.classes_)):
    print(f"{cls} → {code}")

boston_df.sample(n=10)
```

```
High → 0
Low → 1
Moderate → 2
```

[7]:

```
       ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD  TAX  PTRATIO  LSTAT  \
16    0.0   8.14     0  0.538  5.935   29.3  4.4986    4  307     21.0   6.58
111   0.0  10.01     0  0.547  6.715   81.6  2.6775    6  432     17.8  10.16
217   0.0  13.89     0  0.550  6.642   85.1  3.4211    5  276     16.4   9.69
26    0.0   8.14     0  0.538  5.813   90.3  4.6820    4  307     21.0  14.81
434   0.0  18.10     0  0.713  6.208   95.0  2.2222   24  666     20.2  15.17
271  20.0   6.96     0  0.464  6.240   16.3  4.4290    3  223     18.6   6.59
263  20.0   3.97     0  0.647  7.327   94.5  2.0788    5  264     13.0  11.25
402   0.0  18.10     0  0.693  6.404  100.0  1.6390   24  666     20.2  20.31
52   21.0   5.64     0  0.439  6.511   21.1  6.8147    4  243     16.8   5.28
48    0.0   6.91     0  0.448  5.399   95.3  5.8700    3  233     17.9  30.81

     MEDV  CAT. MEDV  CRIM
16   23.1          0     2
111  22.8          0     1
217  28.7          0     1
26   16.6          0     2
434  11.7          0     0
271  25.2          0     2
263  31.0          1     2
402  12.1          0     0
52   25.0          0     1
48   14.4          0     2
```

- .

4b:

- CRIM .
- .

[8]:

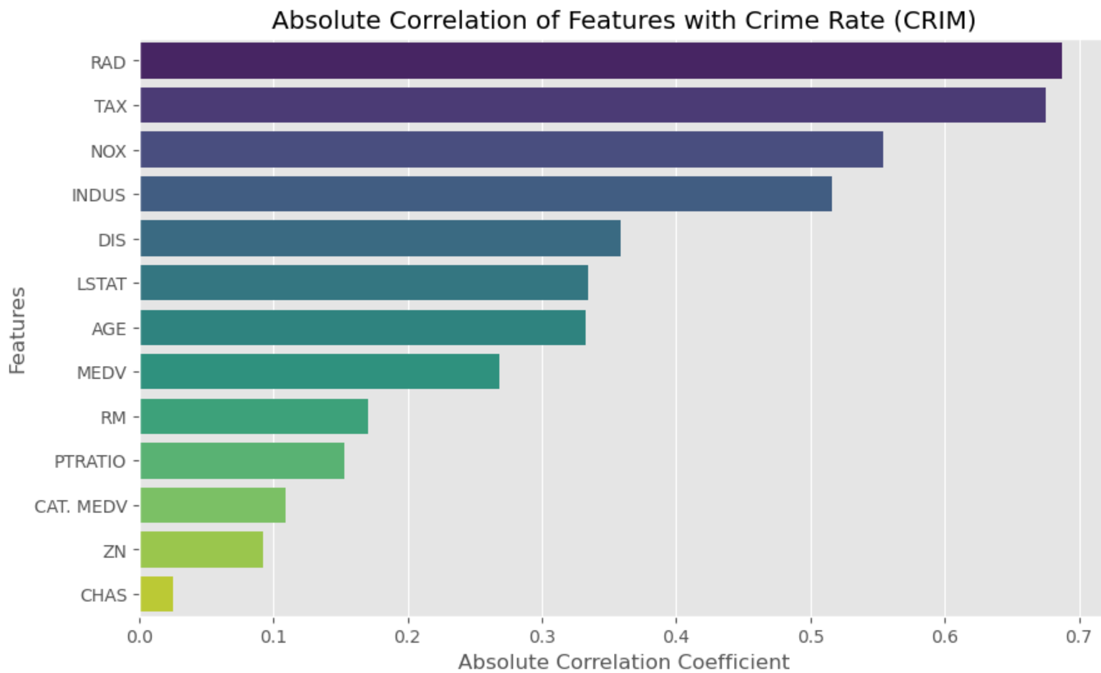
```python
correlation_matrix = boston_df .select_dtypes(include='number').corr()
correlation_with_target = correlation_matrix['CRIM'].abs().
    ↪sort_values(ascending=False)
correlation_with_target = correlation_with_target.drop('CRIM')

print("Absolute correlation of each feature with 'CRIM':")
print(correlation_with_target)
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=correlation_with_target.values, y=correlation_with_target.index,␣
↪palette="viridis")
plt.title('Absolute Correlation of Features with Crime Rate (CRIM)')
plt.xlabel('Absolute Correlation Coefficient')
plt.ylabel('Features')
plt.show()

print("\n--- Feature Importance from a Decision Tree Model ---")

X_for_fs = boston_df.drop('CRIM', axis=1, errors='ignore')
y_for_fs = boston_df['CRIM']

tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_for_fs, y_for_fs)

feature_importances = pd.Series(tree_model.feature_importances_, index=X_for_fs.
↪columns).sort_values(ascending=False)

print("\nFeature importances for predicting 'CRIM' from a Decision Tree:")
print(feature_importances)

plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances.values, y=feature_importances.index,␣
↪palette="plasma")
plt.title('Feature Importance from Decision Tree for Predicting CRIM')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```

```
Absolute correlation of each feature with 'CRIM':
RAD          0.687384
TAX          0.675175
NOX          0.554123
INDUS        0.515949
DIS          0.358847
LSTAT        0.334646
AGE          0.332410
MEDV         0.268105
RM           0.170890
PTRATIO      0.152755
CAT. MEDV    0.109219
ZN           0.091958
CHAS         0.025257
Name: CRIM, dtype: float64
```

```
C:\Users\Home-PC\AppData\Local\Temp\ipykernel_21596\2760956122.py:9:
```

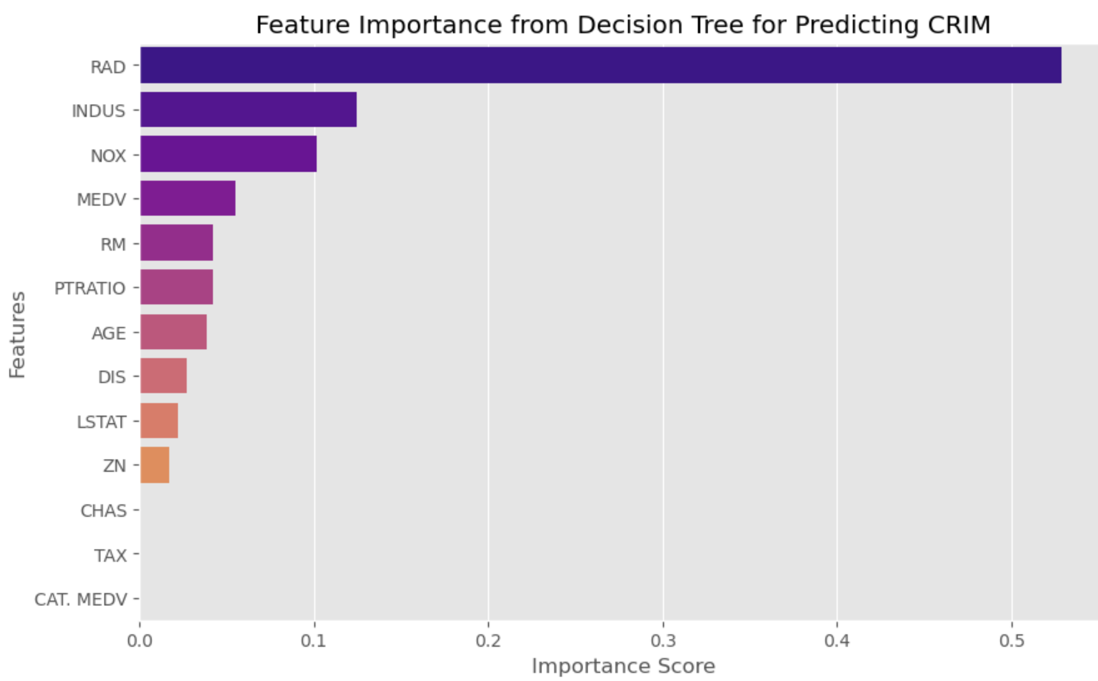
```
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=correlation_with_target.values, y=correlation_with_target.index,
palette="viridis")
```




```
--- Feature Importance from a Decision Tree Model ---

Feature importances for predicting 'CRIM' from a Decision Tree:
RAD          0.528559
INDUS        0.124630
NOX          0.101702
MEDV         0.055105
RM           0.042590
PTRATIO      0.042006
AGE          0.038546
DIS          0.027473
LSTAT        0.022299
ZN           0.017091
CHAS         0.000000
TAX          0.000000
CAT. MEDV    0.000000
```

```
dtype: float64
```

```
C:\Users\Home-PC\AppData\Local\Temp\ipykernel_21596\2760956122.py:29:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=feature_importances.values, y=feature_importances.index,
palette="plasma")
```



4c: (Mutual Information)

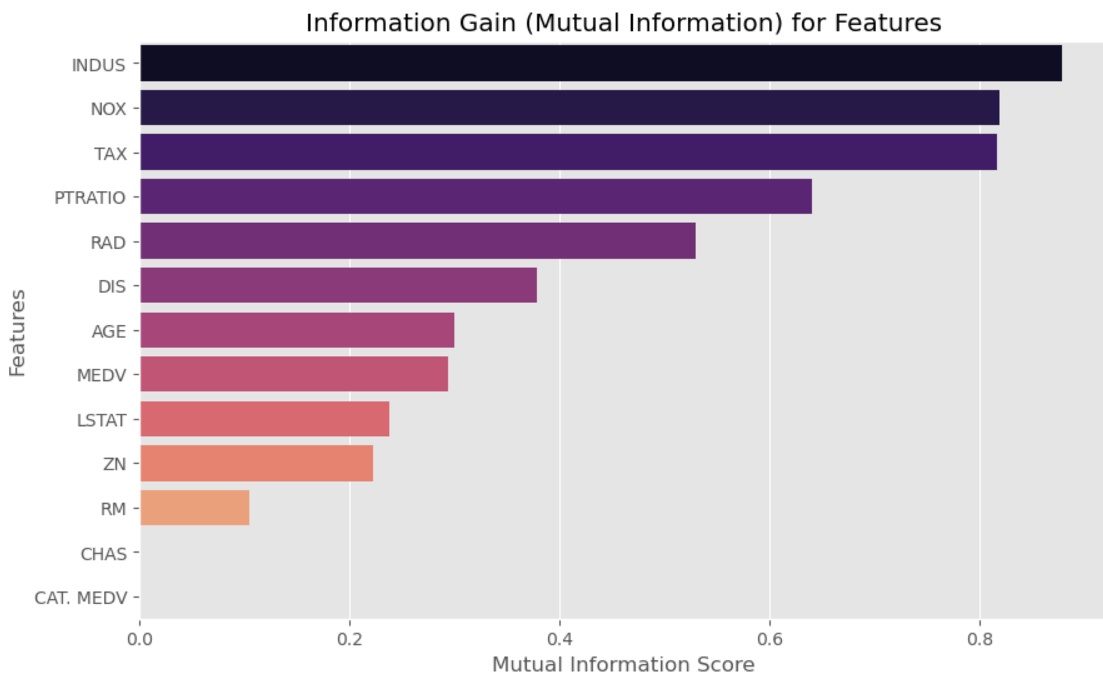
- (crime_level) .
- .
- `mutual_info_classif` .
- .
- .

[9]:
```python
from sklearn.feature_selection import mutual_info_classif

# Calculate mutual information
info_gain = mutual_info_classif(X_for_fs, y_for_fs, random_state=42)
```

```
info_gain_series = pd.Series(info_gain, index=X_for_fs.columns).
    ↪sort_values(ascending=False)
print(info_gain_series)

# Visualize
plt.figure(figsize=(10, 6))
sns.barplot(x=info_gain_series.values, y=info_gain_series.index,␣
    ↪palette="magma")
plt.title('Information Gain (Mutual Information) for Features')
plt.xlabel('Mutual Information Score')
plt.ylabel('Features')
plt.show()
```

```
INDUS        0.877920
NOX          0.818217
TAX          0.816464
PTRATIO      0.640076
RAD          0.529583
DIS          0.378668
AGE          0.299666
MEDV         0.293350
LSTAT        0.237985
ZN           0.222072
RM           0.104689
CHAS         0.000000
CAT. MEDV    0.000000
dtype: float64
```

```
C:\Users\Home-PC\AppData\Local\Temp\ipykernel_21596\27001401.py:11:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=info_gain_series.values, y=info_gain_series.index,
palette="magma")
```



) (

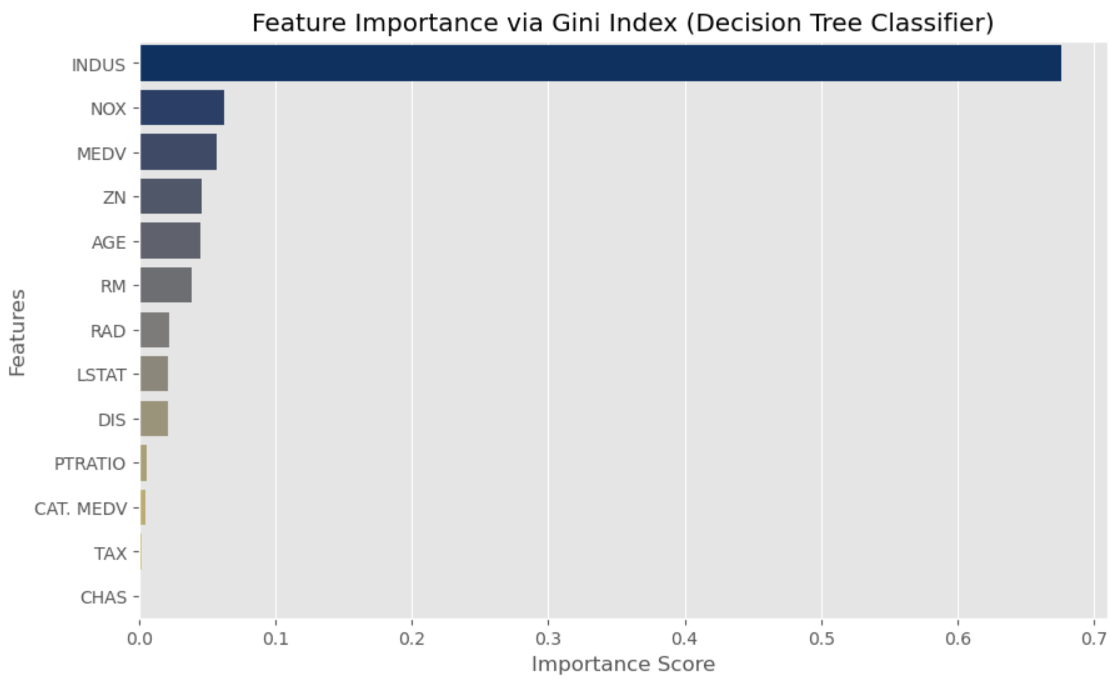
- .
- .
- .

[10]:
```python
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='gini', random_state=42)
clf.fit(X_for_fs, y_for_fs)

gini_importances = pd.Series(clf.feature_importances_, index=X_for_fs.columns).
    ↪sort_values(ascending=False)
print(gini_importances)

plt.figure(figsize=(10, 6))
sns.barplot(x=gini_importances.values, y=gini_importances.index,␣
    ↪palette="cividis")
plt.title('Feature Importance via Gini Index (Decision Tree Classifier)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()

top_info_gain = set(info_gain_series.head(4).index)
top_gini = set(gini_importances.head(4).index)
```

```python
#
final_features_union = list(set( top_info_gain  | top_gini)   )

print(f"Selected Features: {final_features_union}")
```

```
INDUS        0.676157
NOX          0.062450
MEDV         0.056440
ZN           0.045569
AGE          0.044798
RM           0.038723
RAD          0.021764
LSTAT        0.021318
DIS          0.020838
PTRATIO      0.005718
CAT. MEDV    0.004744
TAX          0.001482
CHAS         0.000000
dtype: float64
```

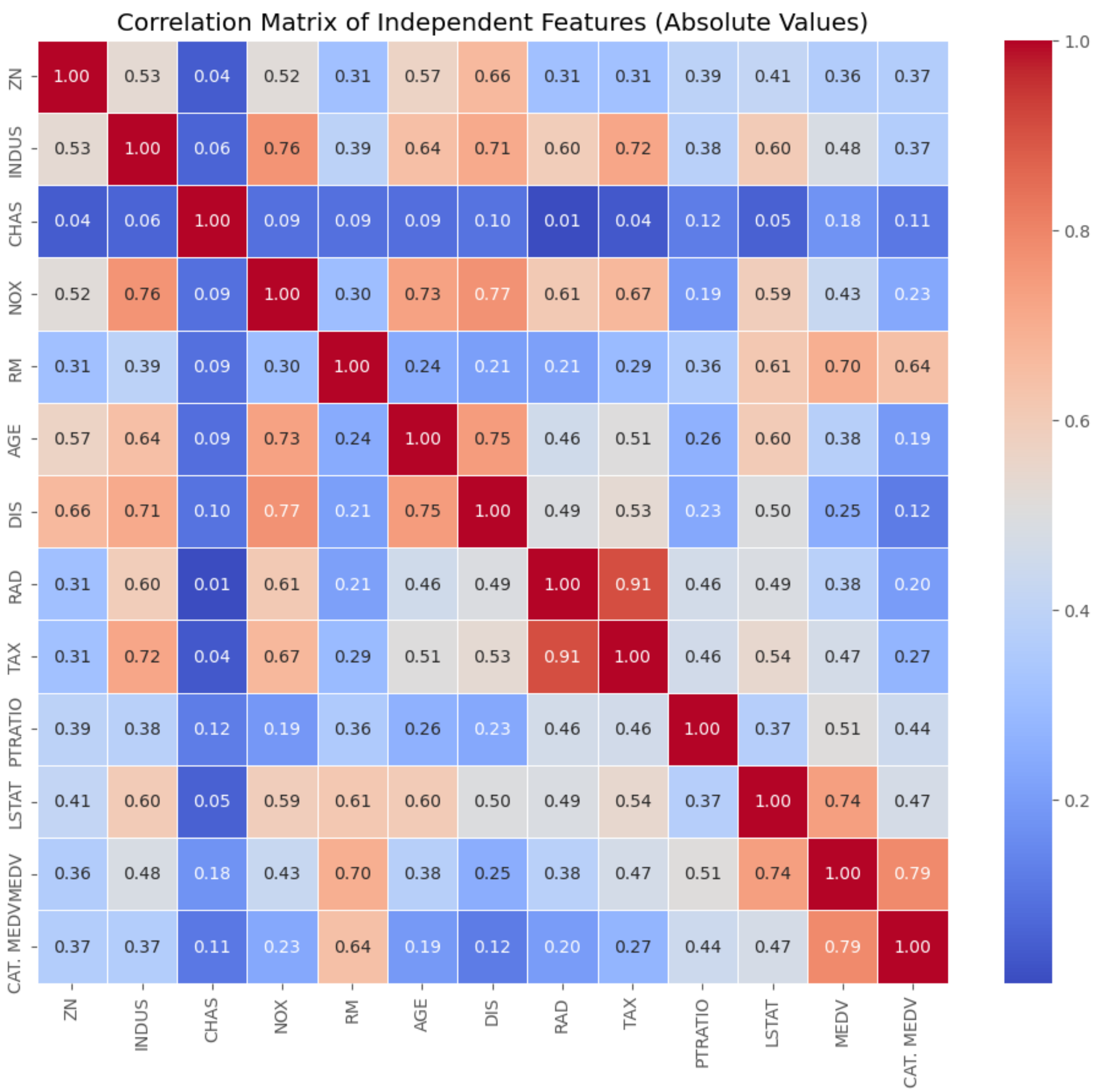
```
C:\Users\Home-PC\AppData\Local\Temp\ipykernel_21596\1848739465.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=gini_importances.values, y=gini_importances.index,
palette="cividis")
```




```
Selected Features: ['NOX', 'INDUS', 'TAX', 'MEDV', 'ZN', 'PTRATIO']
```

- CRIM .
- CRIM) MEDV (.
-) (.
- (heatmap) .

[11]:
```
independent_features = boston_df.drop(columns=['CRIM'], errors='ignore')

corr_matrix = independent_features.corr().abs()

plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix of Independent Features (Absolute Values)")
plt.show()
```



3d:

- .
- X .
- CRIM y .
- .
- X y .

[12]:
```
final_selected_features = ['INDUS', 'DIS', 'PTRATIO', 'LSTAT', 'RAD']

X = boston_df[final_selected_features]
y = boston_df['CRIM']
```

```python
print("\nCorrelation matrix for the final selected features:")
final_features_corr = boston_df[final_selected_features].corr().abs()
print(final_features_corr)

print(f"\n \n \n \nFinal X (features) created with shape: {X.shape}")
print(f"Final y (target) created with shape: {y.shape}")


print("Checking the target variable (y) before splitting the data\n")
print(f"Data type of variable y: {y.dtype}")

print("\nNumber of samples in each class:")
print(y.value_counts())
```

```
Correlation matrix for the final selected features:
            INDUS       DIS   PTRATIO     LSTAT       RAD
INDUS    1.000000  0.708027  0.383248  0.603800  0.595129
DIS      0.708027  1.000000  0.232471  0.496996  0.494588
PTRATIO  0.383248  0.232471  1.000000  0.374044  0.464741
LSTAT    0.603800  0.496996  0.374044  1.000000  0.488676
RAD      0.595129  0.494588  0.464741  0.488676  1.000000




Final X (features) created with shape: (506, 5)
Final y (target) created with shape: (506,)
Checking the target variable (y) before splitting the data

Data type of variable y: int32

Number of samples in each class:
CRIM
0    172
1    167
2    167
Name: count, dtype: int64
```

:5

1. :

- /
- ( )

.2 .

.3 KNN .

[13]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_full_scaled = scaler.fit_transform(X_train_full)
X_test_scaled = scaler.transform(X_test)


print(f"Data split into a full training set of {X_train_full.shape[0]} samples␣
↪and a test set of {X_test.shape[0]} samples.")
```

```
Data split into a full training set of 404 samples and a test set of 102
samples.
```

:6 k) KNN) k:

1. **Holdout**:
   - .
   - k .

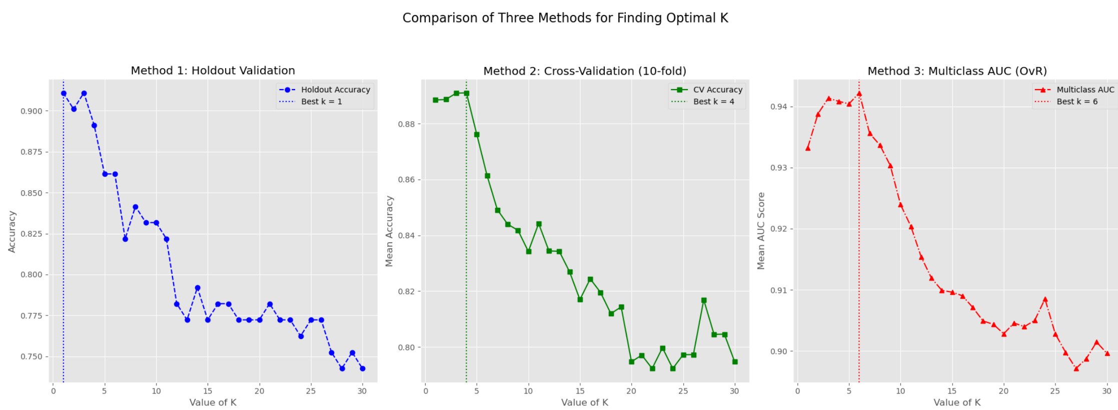
.2 **(Cross-Validation)**:
   - 10-fold CV .
   - .

.3 **AUC (One-vs-Rest)**:
   - .
   - AUC .

[14]:
```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.preprocessing import LabelBinarizer
import numpy as np
import matplotlib.pyplot as plt

k_range = range(1, 31)
plt.figure(figsize=(20, 7))
plt.suptitle('Comparison of Three Methods for Finding Optimal K', fontsize=16,␣
↪y=1.02)

# Method 1: Holdout Validation
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full_scaled, y_train_full, test_size=0.25, random_state=42,␣
↪stratify=y_train_full
)
holdout_scores = []
```

```
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    holdout_scores.append(accuracy_score(y_val, knn.predict(X_val)))

best_k_holdout = k_range[np.argmax(holdout_scores)]
ax1 = plt.subplot(1, 3, 1)
ax1.plot(k_range, holdout_scores, marker='o', linestyle='--', color='blue',␣
↪label='Holdout Accuracy')
ax1.axvline(best_k_holdout, color='blue', linestyle=':', label=f'Best k =␣
↪{best_k_holdout}')
ax1.set_title('Method 1: Holdout Validation')
ax1.set_xlabel('Value of K')
ax1.set_ylabel('Accuracy')
ax1.legend()
ax1.grid(True)

# Method 2: Cross-Validation
cv_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_full_scaled, y_train_full, cv=10,␣
↪scoring='accuracy')
    cv_scores.append(scores.mean())

best_k_cv = k_range[np.argmax(cv_scores)]
ax2 = plt.subplot(1, 3, 2)
ax2.plot(k_range, cv_scores, marker='s', linestyle='-', color='green',␣
↪label='CV Accuracy')
ax2.axvline(best_k_cv, color='green', linestyle=':', label=f'Best k =␣
↪{best_k_cv}')
ax2.set_title('Method 2: Cross-Validation (10-fold)')
ax2.set_xlabel('Value of K')
ax2.set_ylabel('Mean Accuracy')
ax2.legend()
ax2.grid(True)

# Method 3: Multiclass AUC
lb = LabelBinarizer()
y_val_binarized = lb.fit_transform(y_val)
roc_auc_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_proba = knn.predict_proba(X_val)
    roc_auc_scores.append(roc_auc_score(y_val_binarized, y_pred_proba,␣
↪multi_class='ovr', average='macro'))
```

```
best_k_roc = k_range[np.argmax(roc_auc_scores)]
ax3 = plt.subplot(1, 3, 3)
ax3.plot(k_range, roc_auc_scores, marker='^', linestyle='-.', color='red',␣
 ↪label='Multiclass AUC')
ax3.axvline(best_k_roc, color='red', linestyle=':', label=f'Best k =␣
 ↪{best_k_roc}')
ax3.set_title('Method 3: Multiclass AUC (OvR)')
ax3.set_xlabel('Value of K')
ax3.set_ylabel('Mean AUC Score')
ax3.legend()
ax3.grid(True)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



:

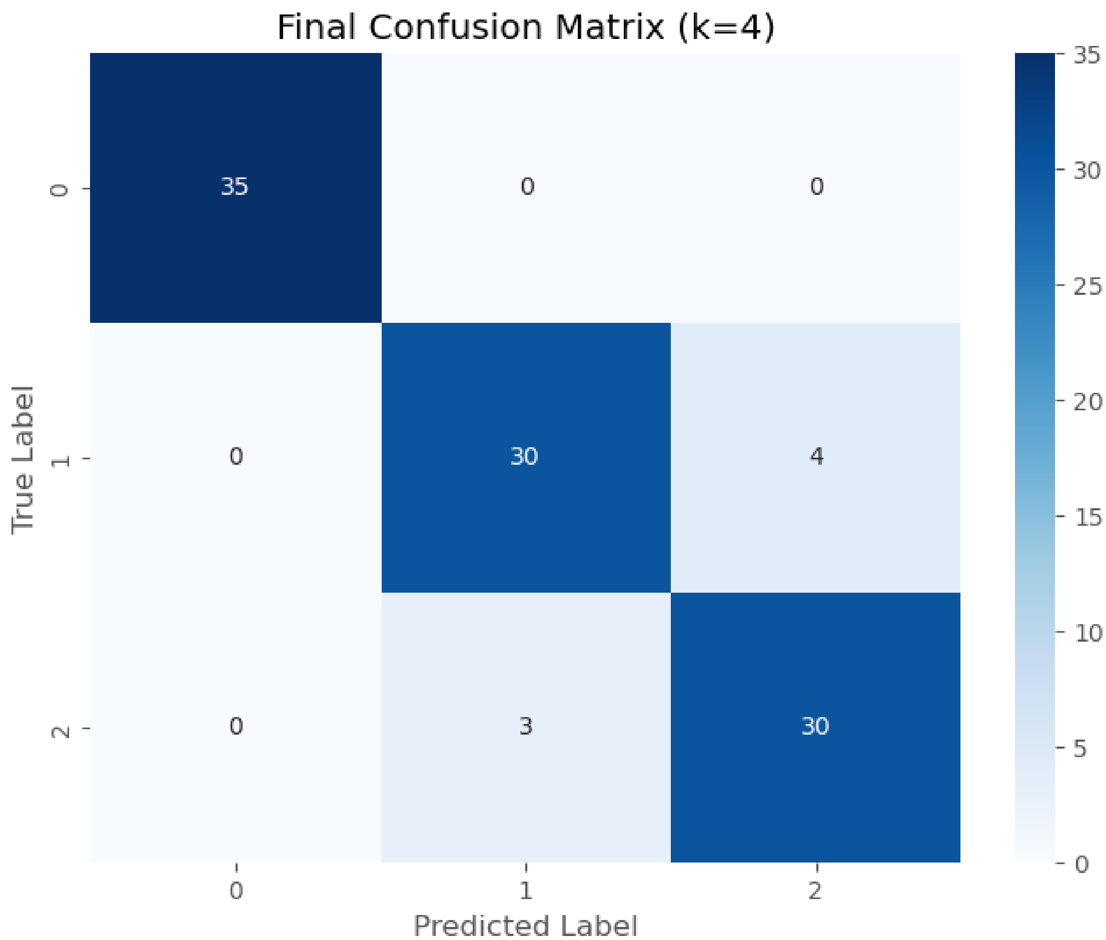
- KNN k .
- .
- (20) :
  - (Accuracy)
  - (Classification Report)
  - (Confusion Matrix) .

[15]:
```
optimal_k = 4
```

[16]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(f"Building final model with the optimal k = {optimal_k}...")

# Build KNN model with the optimized k value
final_model = KNeighborsClassifier(n_neighbors=optimal_k)
```

```
# Train the model on the full standardized training data
final_model.fit(X_train_full_scaled, y_train_full)

# Make predictions on the standardized test data
y_pred = final_model.predict(X_test_scaled)

# Calculate accuracy
final_accuracy = accuracy_score(y_test, y_pred)
print(f"\nFinal Model Accuracy (with k={optimal_k}): {final_accuracy:.2%}\n")

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Low', 'Moderate',␣
↪'High']))

# Confusion matrix
print("\nConfusion Matrix:")
labels_order = sorted(y.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels_order)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=labels_order, yticklabels=labels_order)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'Final Confusion Matrix (k={optimal_k})')
plt.show()
```

```
Building final model with the optimal k = 4…

Final Model Accuracy (with k=4): 93.14%

Classification Report:
              precision    recall  f1-score   support

         Low       1.00      1.00      1.00        35
    Moderate       0.91      0.88      0.90        34
        High       0.88      0.91      0.90        33

    accuracy                           0.93       102
   macro avg       0.93      0.93      0.93       102
weighted avg       0.93      0.93      0.93       102


Confusion Matrix:
```



- **:** [‘INDUS’, ‘DIS’, ‘PTRATIO’, ‘LSTAT’, ‘RAD’]
- **k:** 4
- **:** 93.14

.1 **:** ” “ ” “ ” “ .
.2 **:**) (.
.3 **k:** k=4 .

- **93.14** .
- .

[17]:
```python
print(f"- Selected Features: {final_selected_features}")
print(f"- Optimal k: {optimal_k}")
print(f"- Test Accuracy: {final_accuracy:.2%}")
```

```
- Selected Features: ['INDUS', 'DIS', 'PTRATIO', 'LSTAT', 'RAD']
- Optimal k: 4
- Test Accuracy: 93.14%
```