Data Analysis

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Load dataset
df = pd.read_csv('/content/drive/MyDrive/car_evaluation.csv')
df.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=['unacc'])
y = df['unacc']

# Scale for KNN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['unacc'] = le.fit_transform(df['unacc'])
df.head(100)
```

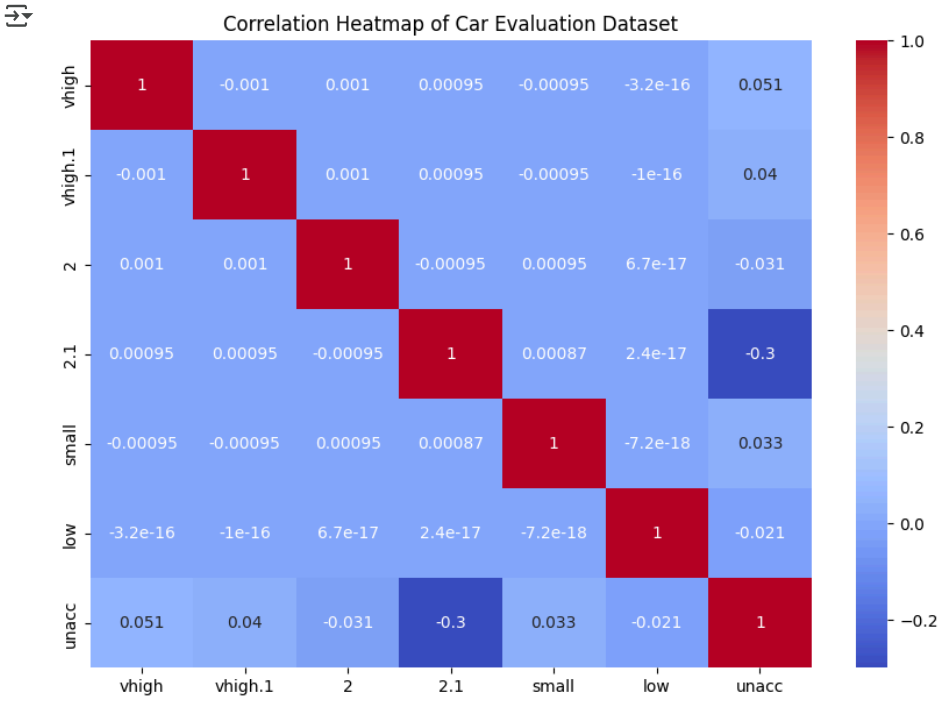
	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	2
1	vhigh	vhigh	2	2	small	high	2
2	vhigh	vhigh	2	2	med	low	2
3	vhigh	vhigh	2	2	med	med	2
4	vhigh	vhigh	2	2	med	high	2
...	...	...	...	...	...	...	...
95	vhigh	vhigh	5more	4	big	low	2
96	vhigh	vhigh	5more	4	big	med	2
97	vhigh	vhigh	5more	4	big	high	2
98	vhigh	vhigh	5more	more	small	low	2
99	vhigh	vhigh	5more	more	small	med	2

100 rows × 7 columns

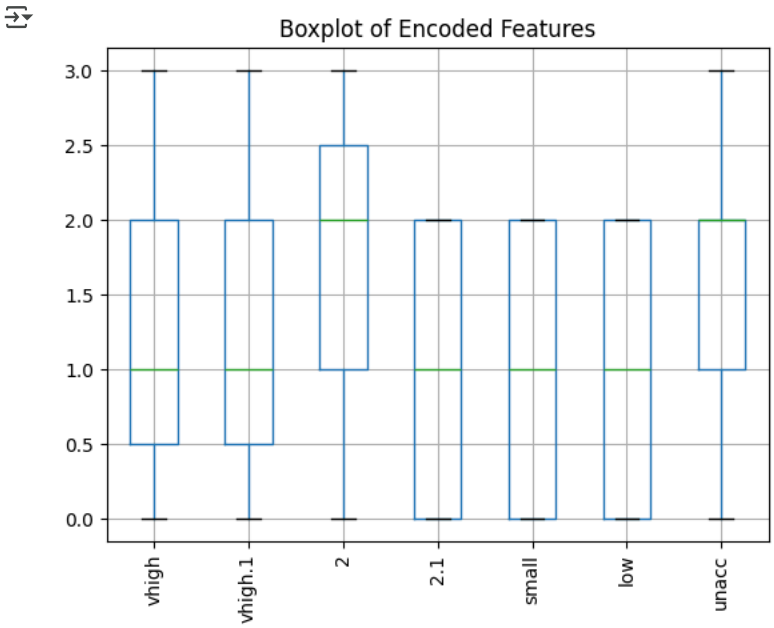
Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Encode categorical features
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])
```

```
# Visualizations
plt.figure(figsize=(10, 7))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Car Evaluation Dataset')
plt.show()
```

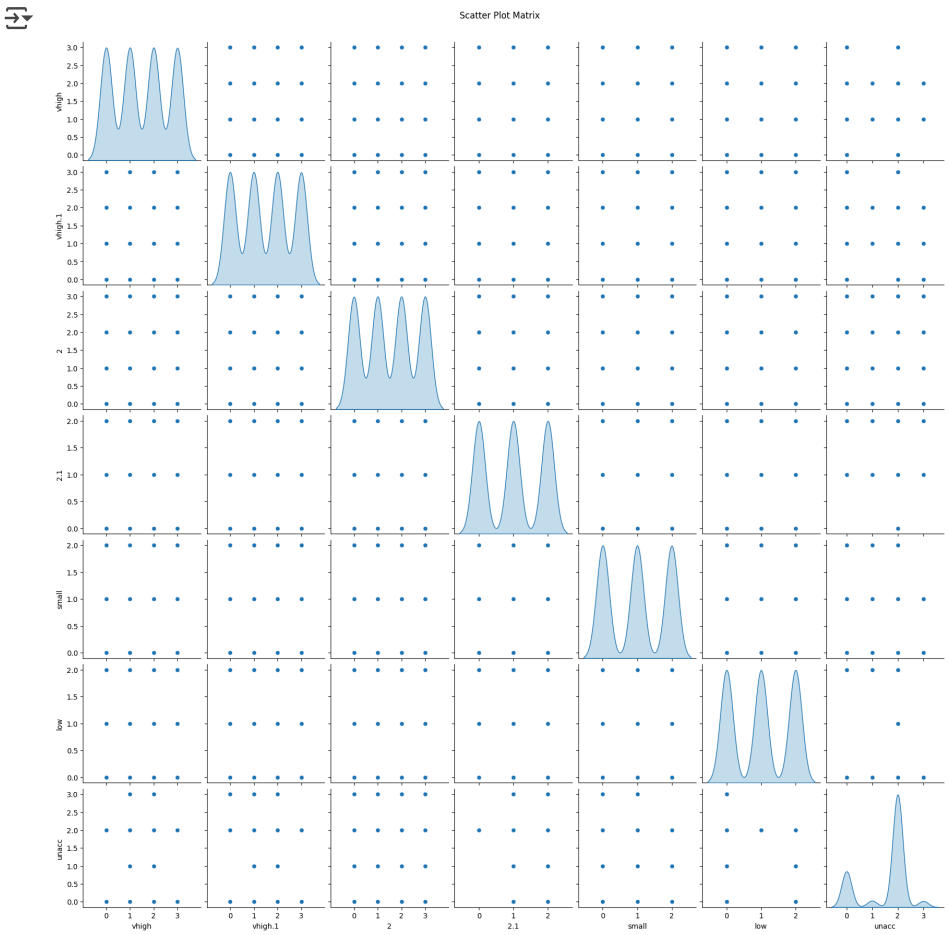



```
df.boxplot(rot=90)
plt.title('Boxplot of Encoded Features')
plt.show()
```




```
sns.pairplot(df, diag_kind='kde')
plt.suptitle('Scatter Plot Matrix', y=1.02)
```

```
plt.suptitle('Scatter Plot Matrix', y=1.02)
plt.show()
```



Algorithm Implementation: Decision Tree Implementation

```
# Features and target
X = df.drop(columns=['unacc'])
y = df['unacc']
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
#Decision Tree
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
```

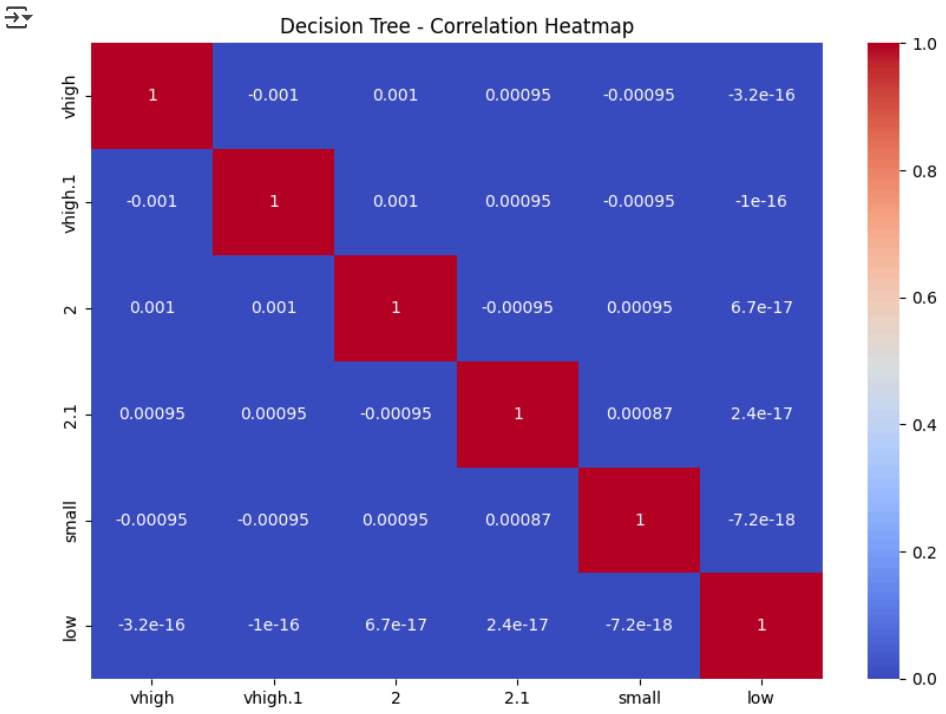
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

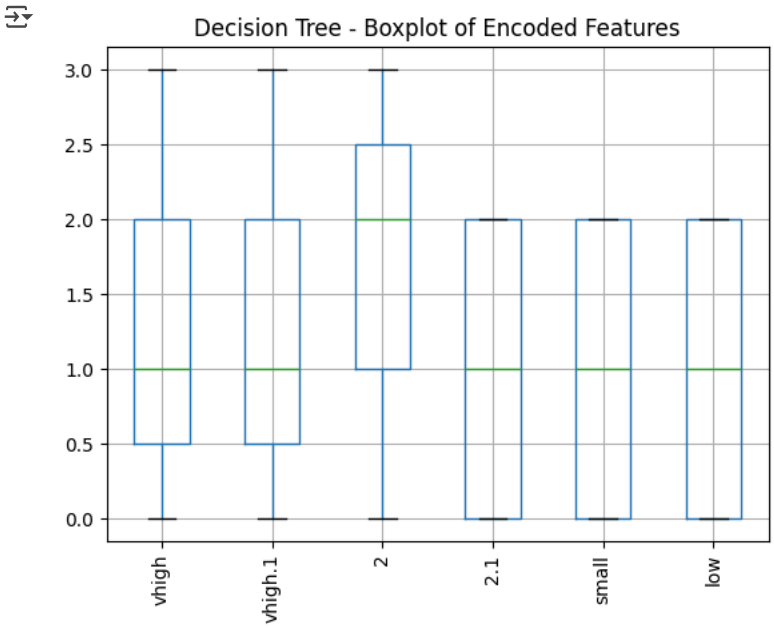
```
# Evaluate Decision Tree
print("Decision Tree Results")
print(classification_report(y_test, dt_pred))
print(confusion_matrix(y_test, dt_pred))
print('Decision Tree Accuracy:', accuracy_score(y_test, dt_pred))
```

```
Decision Tree Results
                precision    recall  f1-score   support

             0       0.98      0.95      0.97       105
             1       0.76      0.81      0.79        16
             2       1.00      1.00      1.00       290
             3       0.86      0.90      0.88        21

      accuracy                           0.98       432
     macro avg       0.90      0.92      0.91       432
  weighted avg       0.98      0.98      0.98       432

[[100   2   1   2]
 [  2  13   0   1]
 [  0   0 290   0]
 [  0   2   0  19]]
Decision Tree Accuracy: 0.9768518518518519
```

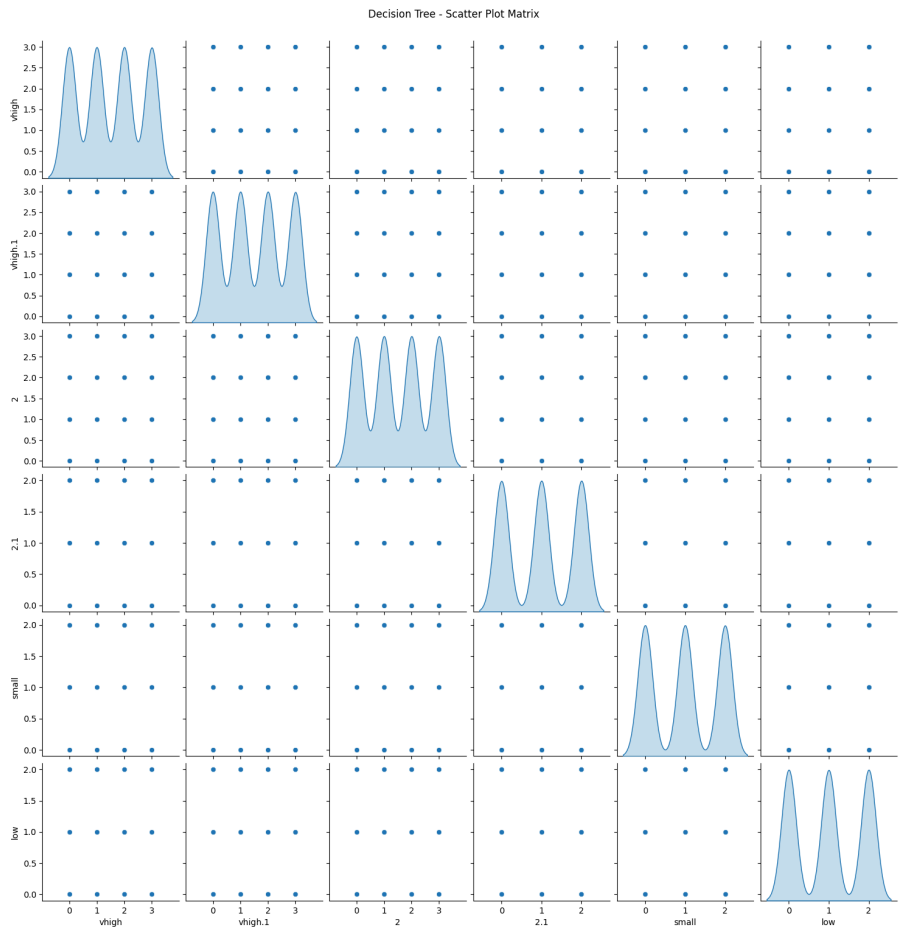
```
# Visualizations specific to Decision Tree
plt.figure(figsize=(10, 7))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm')
plt.title('Decision Tree - Correlation Heatmap')
plt.show()
```




```
X.boxplot(rot=90)
plt.title('Decision Tree - Boxplot of Encoded Features')
plt.show()
```



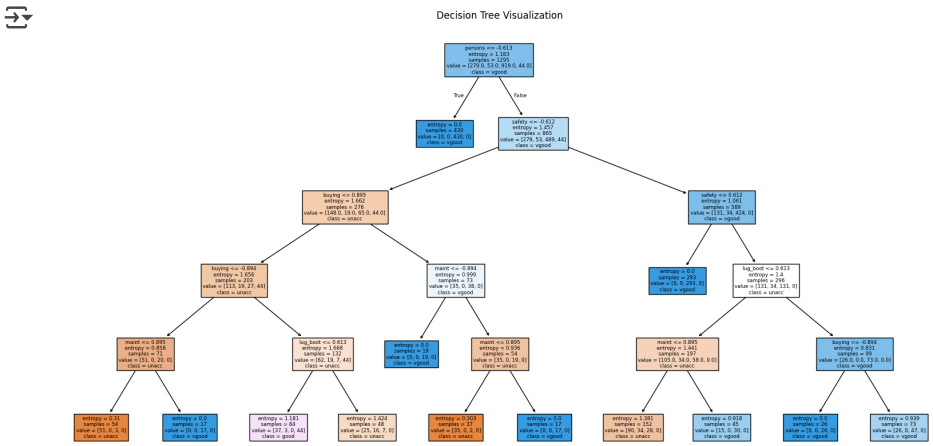

```
sns.pairplot(df.drop(columns=['unacc']), diag_kind='kde')
plt.suptitle('Decision Tree - Scatter Plot Matrix', y=1.02)
plt.show()
```




```
# Load the dataset
columns = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
# Decision Tree Visualizations
df = pd.read_csv('/content/drive/MyDrive/car_evaluation.csv')
dt = DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=42)
dt.fit(X_train, y_train)

from sklearn.tree import plot_tree
plt.figure(figsize=(20, 10))
plot_tree(dt, feature_names=columns[:-1], class_names=df['unacc'].unique(), filled=True)
plt.title('Decision Tree Visualization')
```

```
plt.show()
```



Dataset Exploration

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

```
print("Dataset shape:", df.shape)
print("\nFirst 5 rows:")
print(df.head())
print("\nValue counts for each feature:")
for col in df.columns:
    print(f"\n{col}:")
    print(df[col].value_counts())
```

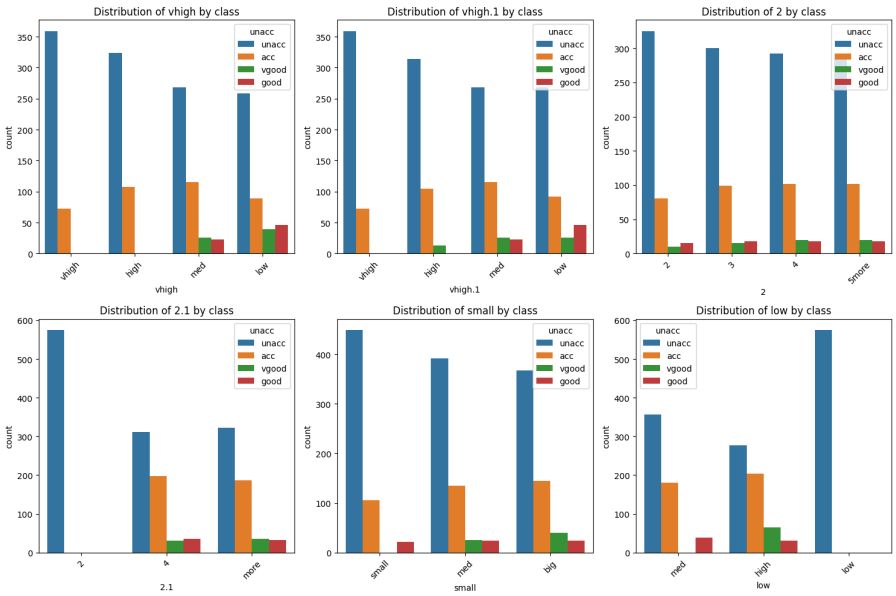
```
2  vhigh   vhigh  2   2    med   low  unacc
3  vhigh   vhigh  2   2    med   med  unacc
4  vhigh   vhigh  2   2    med  high  unacc

Value counts for each feature:

vhigh:
vhigh
high     432
med      432
low      432
vhigh    431
Name: count, dtype: int64

vhigh.1:
vhigh.1
high     432
med      432
low      432
vhigh    431
```

```
5more    432
2        431
Name: count, dtype: int64

2.1:
2.1
4       576
more    576
2       575
Name: count, dtype: int64

small:
small
med      576
big      576
small    575
Name: count, dtype: int64

low:
low
med     576
high    576
low     575
Name: count, dtype: int64

unacc:
unacc
unacc    1209
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

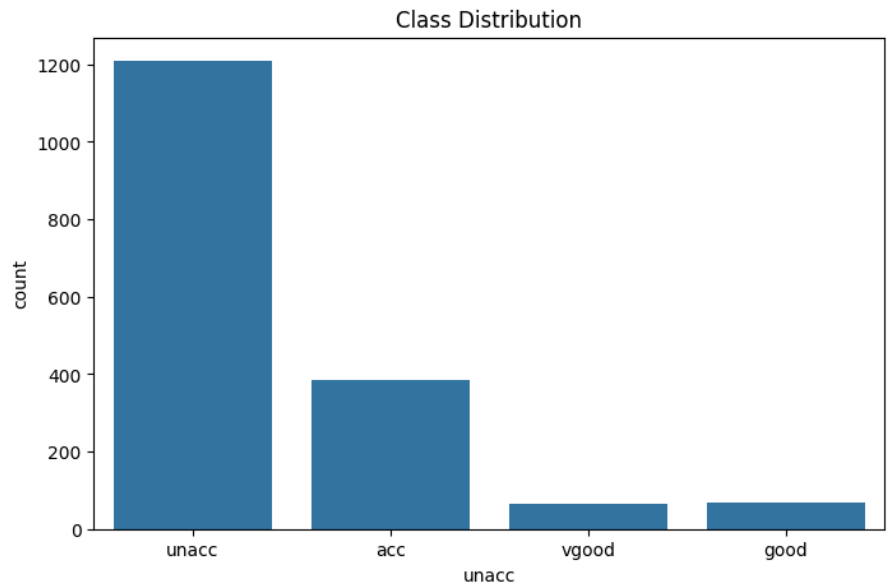
```
# Visualizations
plt.figure(figsize=(15, 10))
```

```
<Figure size 1500x1000 with 0 Axes>
<Figure size 1500x1000 with 0 Axes>
```

```
# Bar plots for each feature
plt.figure(figsize=(15, 10))
for i, col in enumerate(df.columns[:-1]):
    plt.subplot(2, 3, i+1)
    sns.countplot(data=df, x=col, hue='unacc')
    plt.title(f'Distribution of {col} by class')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
# Class distribution
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='unacc')
plt.title('Class Distribution')
plt.show()
```



K-Nearest Neighbors

```
# Correlation heatmap
encoded_df = df.copy()
le = LabelEncoder()
for col in encoded_df.columns:
    encoded_df[col] = le.fit_transform(encoded_df[col])
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)
```

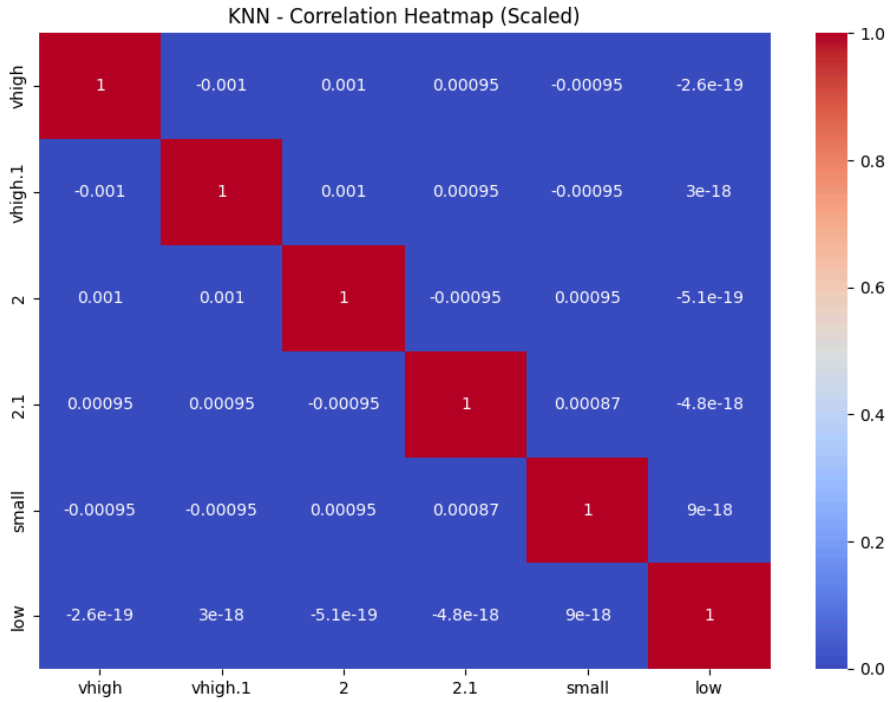
```
# Train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto')
knn_model.fit(X_train, y_train)

# Predict on test set
y_pred_knn = knn_model.predict(X_test)
```

```
# Evaluation
print("K-Nearest Neighbors Results")
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
print('KNN Accuracy:', accuracy_score(y_test, y_pred_knn))
```

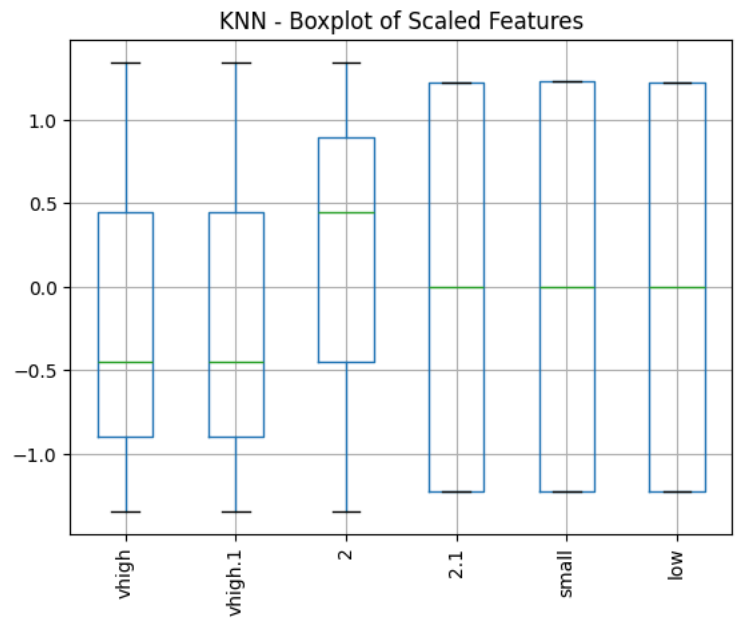
```
K-Nearest Neighbors Results
                  precision    recall  f1-score   support

               0       0.80      0.88      0.84       105
               1       0.67      0.12      0.21        16
               2       0.95      0.99      0.97       290
               3       1.00      0.67      0.80        21

        accuracy                           0.91       432
       macro avg       0.85      0.66      0.70       432
    weighted avg       0.91      0.91      0.90       432

    [[ 92   1  12   0]
     [ 12   2   2   0]
     [  4   0 286   0]
     [  7   0   0  14]]
    KNN Accuracy: 0.9120370370370371
```

```
# Visualizations specific to K-Nearest Neighbors
plt.figure(figsize=(10, 7))
sns.heatmap(pd.DataFrame(X_scaled, columns=X.columns).corr(), annot=True, cmap='coolwarm')
plt.title('KNN - Correlation Heatmap (Scaled)')
```

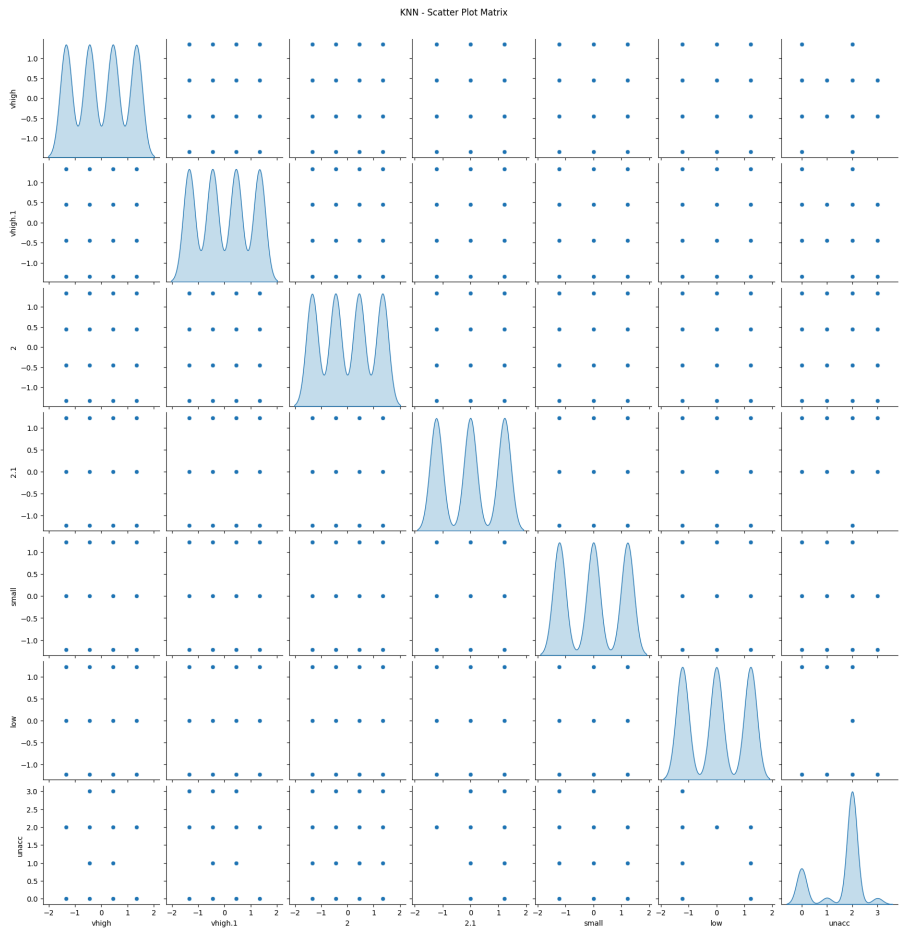
```
Text(0.5, 1.0, 'KNN - Correlation Heatmap (Scaled)')
```




```
pd.DataFrame(X_scaled, columns=X.columns).boxplot(rot=90)
plt.title('KNN - Boxplot of Scaled Features')
plt.show()
```



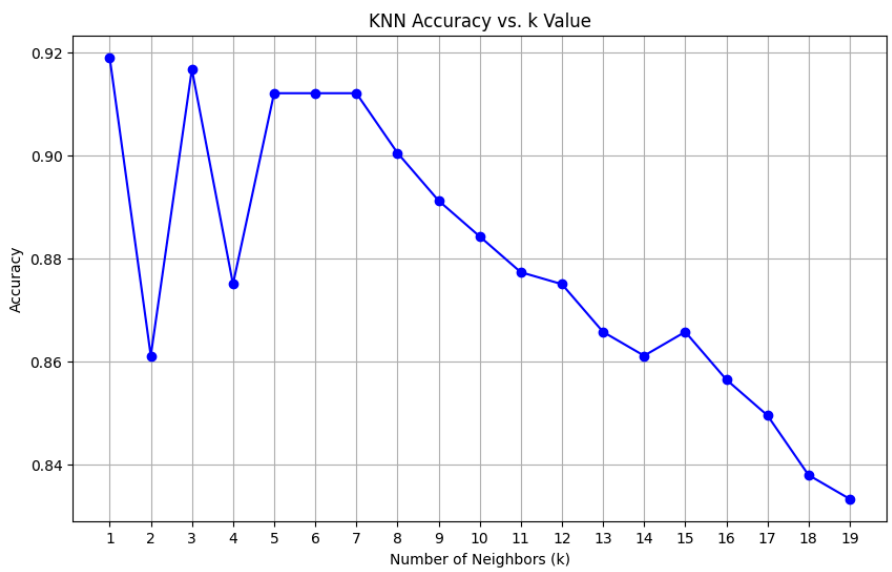

```
sns.pairplot(pd.concat([pd.DataFrame(X_scaled, columns=X.columns), y.reset_index(drop=True)], axis=1), diag_kind='kde')
plt.suptitle('KNN - Scatter Plot Matrix', y=1.02)
plt.show()
```



ptimal K Value for KNN

```
#Finding Optimal K Value
k_values = range(1, 20)
accuracies = []

for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred_k = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracies.append(acc)
```

```
# Plotting k vs accuracy
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='blue')
plt.title('KNN Accuracy vs. k Value')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```



Performance Comparison

```
# Performance comparison
# Calculate Accuracy
dt_accuracy = accuracy_score(y_test, dt_pred)
knn_accuracy = accuracy_score(y_test, y_pred_knn)

print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
print(f"KNN Accuracy: {knn_accuracy:.2f}")
# Classification Reports
dt_report = classification_report(y_test, dt_pred, output_dict=True)
knn_report = classification_report(y_test, y_pred_knn, output_dict=True)

# Extract F1 Scores
dt_f1 = dt_report['macro avg']['f1-score']
knn_f1 = knn_report['macro avg']['f1-score']
# Summary Table
print("\nAlgorithm Comparison:")
print(f"{'Metric':<15}{'Decision Tree':<15}{'KNN':<15}")
print(f"{'Accuracy':<15}{dt_accuracy:<15.4f}{knn_accuracy:<15.4f}")
print(f"{'F1-score (macro)':<15}{dt_f1:<15.4f}{knn_f1:<15.4f}")
```

```
Decision Tree Accuracy: 0.98
KNN Accuracy: 0.91
```

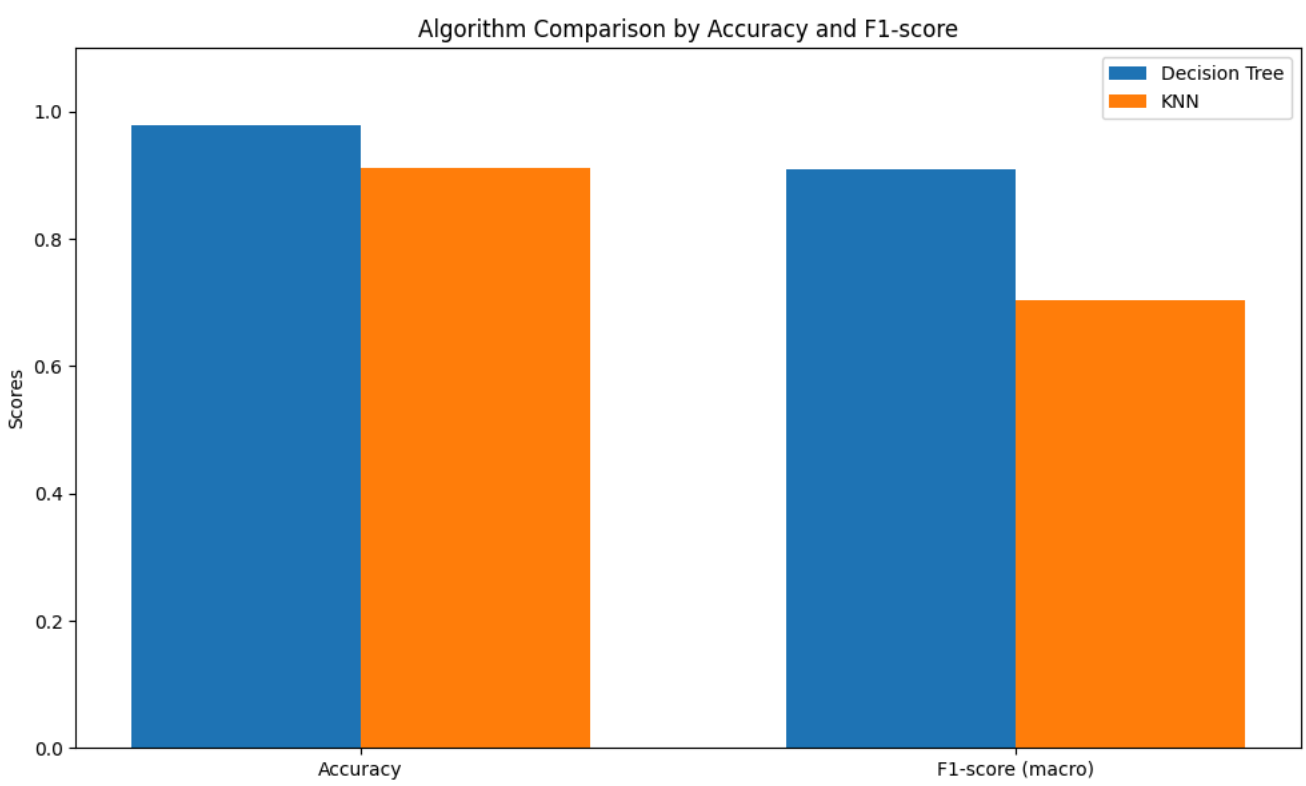
```
Algorithm Comparison:
Metric         Decision Tree  KNN
Accuracy       0.9769         0.9120
F1-score (macro)0.9090         0.7041
```

Visualization of Algorithm Comparison

```
# Bar Chart Comparison
import numpy as np
import matplotlib.pyplot as plt

metrics = ['Accuracy', 'F1-score (macro)']
dt_scores = [dt_accuracy, dt_f1]
knn_scores = [knn_accuracy, knn_f1]

x = np.arange(len(metrics))  # [0, 1]
width = 0.35
fig, ax = plt.subplots(figsize=(10, 6))

# Bar plots
rects1 = ax.bar(x - width/2, dt_scores, width, label='Decision Tree')
rects2 = ax.bar(x + width/2, knn_scores, width, label='KNN')

# Labels and formatting
ax.set_ylabel('Scores')
ax.set_title('Algorithm Comparison by Accuracy and F1-score')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.set_ylim(0, 1.1)  # Ensure full score visibility
ax.legend()
fig.tight_layout()
plt.show()
```

Visualization of Algorithm Comparison

```
# Bar Chart Comparison
import numpy as np
import matplotlib.pyplot as plt

metrics = ['Accuracy', 'F1-score (macro)']
dt_scores = [dt_accuracy, dt_f1]
knn_scores = [knn_accuracy, knn_f1]

x = np.arange(len(metrics))  # [0, 1]
width = 0.35
fig, ax = plt.subplots(figsize=(10, 6))

# Bar plots
rects1 = ax.bar(x - width/2, dt_scores, width, label='Decision Tree')
rects2 = ax.bar(x + width/2, knn_scores, width, label='KNN')

# Labels and formatting
ax.set_ylabel('Scores')
ax.set_title('Algorithm Comparison by Accuracy and F1-score')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.set_ylim(0, 1.1)  # Ensure full score visibility
ax.legend()
fig.tight_layout()
plt.show()
```



```
# ROC Curve Comparison (Binary Example)
from sklearn.metrics import roc_curve, auc
import numpy as np

# Convert to binary: class 0 vs others
binary_y = (y == 0).astype(int)
X_train_bin, X_test_bin, y_train_bin, y_test_bin = train_test_split(X, binary_y, test_size=0.25, random_state=42)
X_train_sbin, X_test_sbin = train_test_split(X_scaled, test_size=0.25, random_state=42)

# DT
dt_model_bin = DecisionTreeClassifier()
dt_model_bin.fit(X_train_bin, y_train_bin)
y_score_dt = dt_model_bin.predict_proba(X_test_bin)[:, 1]
fpr_dt, tpr_dt, _ = roc_curve(y_test_bin, y_score_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# KNN
knn_model_bin = KNeighborsClassifier(n_neighbors=5)
knn_model_bin.fit(X_train_sbin, y_train_bin)
y_score_knn = knn_model_bin.predict_proba(X_test_sbin)[:, 1]
fpr_knn, tpr_knn, _ = roc_curve(y_test_bin, y_score_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)

plt.figure(figsize=(10, 6))
plt.plot(fpr_dt, tpr_dt, label=f'DT (AUC = {roc_auc_dt:.2f})', color='blue')
plt.plot(fpr_knn, tpr_knn, label=f'kNN (AUC = {roc_auc_knn:.2f})', color='green')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```