DSC550-T301

Chitramoy Mukherjee

Week-7

Date: 1/24/2023

Exercise - 7.1

Import the housing data as a data frame and ensure that the data is loaded properly.

In [137]:

```
import warnings
warnings.filterwarnings('ignore')

# Required python basic libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from math import sqrt
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.tree import plot_tree

#Required python visualization libraries

# import missingno as msno
import matplotlib
import matplotlib.pyplot as plt

def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)

### Reading the mxmh_survey_results.csv dataset
housing_df = pd.read_csv("C:\\Users\\14024\\OneDrive\\Desktop\\MS-DSC\\DSC

# Check first 5 rows of the dataset
housing_df.head()
```

Out[137]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl

5 rows × 81 columns

Drop the "Id" column and any features that are missing more than 40% of their values.

In [138]:

```
# Drop the "Id" column
housing_df = housing_df.drop("Id", axis=1)

# Calculate the percentage of missing values for each feature
missing_percentage = (housing_df.isnull().sum() / len(housing_df)) * 100

# Drop features with more than 40% missing values
missing_threshold = 40
columns_to_drop = missing_percentage[missing_percentage > missing_threshol
housing_df = housing_df.drop(columns=columns_to_drop)

# Display the modified DataFrame
print(housing_df.head())
```

```
   MSSubClass MSZoning  LotFrontage  LotArea Street LotShape LandContour
\
0          60       RL         65.0     8450   Pave      Reg         Lvl
1          20       RL         80.0     9600   Pave      Reg         Lvl
2          60       RL         68.0    11250   Pave      IR1         Lvl
3          70       RL         60.0     9550   Pave      IR1         Lvl
4          60       RL         84.0    14260   Pave      IR1         Lvl

  Utilities LotConfig LandSlope  ... EnclosedPorch 3SsnPorch ScreenPorch
\
0    AllPub    Inside       Gtl  ...             0         0           0
1    AllPub       FR2       Gtl  ...             0         0           0
2    AllPub    Inside       Gtl  ...             0         0           0
3    AllPub    Corner       Gtl  ...           272         0           0
4    AllPub       FR2       Gtl  ...             0         0           0

  PoolArea MiscVal  MoSold  YrSold  SaleType  SaleCondition SalePrice
0        0       0       2    2008        WD         Normal    208500
1        0       0       5    2007        WD         Normal    181500
2        0       0       9    2008        WD         Normal    223500
3        0       0       2    2006        WD        Abnorml    140000
4        0       0      12    2008        WD         Normal    250000

[5 rows x 74 columns]
```

For numerical columns, fill in any missing data with the median value.

In [139]:
```
# Fill missing values in numerical columns with the median
numerical_columns = housing_df.select_dtypes(include='number').columns
housing_df[numerical_columns] = housing_df[numerical_columns].fillna(housi

# Display the modified data
print("\nData after preprocessing:")
print(housing_df.head())
```

```
Data after preprocessing:
   MSSubClass MSZoning  LotFrontage  LotArea Street LotShape LandContour
\
0          60       RL         65.0     8450   Pave      Reg         Lvl
1          20       RL         80.0     9600   Pave      Reg         Lvl
2          60       RL         68.0    11250   Pave      IR1         Lvl
3          70       RL         60.0     9550   Pave      IR1         Lvl
4          60       RL         84.0    14260   Pave      IR1         Lvl

  Utilities LotConfig LandSlope  ... EnclosedPorch 3SsnPorch ScreenPorch
\
0    AllPub    Inside       Gtl  ...             0         0           0
1    AllPub       FR2       Gtl  ...             0         0           0
2    AllPub    Inside       Gtl  ...             0         0           0
3    AllPub    Corner       Gtl  ...           272         0           0
4    AllPub       FR2       Gtl  ...             0         0           0

  PoolArea MiscVal  MoSold  YrSold  SaleType  SaleCondition SalePrice
0        0       0       2    2008        WD         Normal    208500
1        0       0       5    2007        WD         Normal    181500
2        0       0       9    2008        WD         Normal    223500
3        0       0       2    2006        WD        Abnorml    140000
4        0       0      12    2008        WD         Normal    250000

[5 rows x 74 columns]
```

For categorical columns, fill in any missing data with the most common value (mode).

In [140]:

```
# Fill missing categorical values with the mode
categorical_cols = housing_df.select_dtypes(include='object').columns
housing_df[categorical_cols] = housing_df[categorical_cols].fillna(housing

# Print the modified DataFrame
print(housing_df)
```

```
      MSSubClass MSZoning  LotFrontage  LotArea Street LotShape LandConto
ur  \
0             60       RL         65.0     8450   Pave      Reg         L
vl
1             20       RL         80.0     9600   Pave      Reg         L
vl
2             60       RL         68.0    11250   Pave      IR1         L
vl
3             70       RL         60.0     9550   Pave      IR1         L
vl
4             60       RL         84.0    14260   Pave      IR1         L
vl
...          ...      ...          ...      ...    ...      ...
...
1455          60       RL         62.0     7917   Pave      Reg         L
vl
1456          20       RL         85.0    13175   Pave      Reg         L
vl
1457          70       RL         66.0     9042   Pave      Reg         L
vl
1458          20       RL         68.0     9717   Pave      Reg         L
vl
1459          20       RL         75.0     9937   Pave      Reg         L
vl

     Utilities LotConfig LandSlope  ... EnclosedPorch 3SsnPorch ScreenPor
ch  \
0       AllPub    Inside       Gtl  ...             0         0
0
1       AllPub       FR2       Gtl  ...             0         0
0
2       AllPub    Inside       Gtl  ...             0         0
0
3       AllPub    Corner       Gtl  ...           272         0
0
4       AllPub       FR2       Gtl  ...             0         0
0
...        ...       ...       ...  ...           ...       ...
...
1455    AllPub    Inside       Gtl  ...             0         0
0
1456    AllPub    Inside       Gtl  ...             0         0
0
1457    AllPub    Inside       Gtl  ...             0         0
0
1458    AllPub    Inside       Gtl  ...           112         0
0
1459    AllPub    Inside       Gtl  ...             0         0
0

     PoolArea MiscVal  MoSold  YrSold  SaleType  SaleCondition SalePrice
0           0       0       2    2008        WD         Normal    208500
1           0       0       5    2007        WD         Normal    181500
2           0       0       9    2008        WD         Normal    223500
3           0       0       2    2006        WD        Abnorml    140000
4           0       0      12    2008        WD         Normal    250000
...       ...     ...     ...     ...       ...            ...       ...
```

```
1455        0       0       8    2007        WD         Normal    175000
1456        0       0       2    2010        WD         Normal    210000
1457        0    2500       5    2010        WD         Normal    266500
1458        0       0       4    2010        WD         Normal    142125
1459        0       0       6    2008        WD         Normal    147500

[1460 rows x 74 columns]
```

Convert the categorical columns to dummy variables.

In [141]:

```
# Convert categorical columns to dummy variables
housing_df = pd.get_dummies(housing_df, columns=categorical_cols)

# Print the modified DataFrame
print(housing_df)
```

```
      MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBui
lt  \
0             60         65.0     8450            7            5       20
03
1             20         80.0     9600            6            8       19
76
2             60         68.0    11250            7            5       20
01
3             70         60.0     9550            7            5       19
15
4             60         84.0    14260            8            5       20
00
...          ...          ...      ...          ...          ...
...
1455          60         62.0     7917            6            5       19
99
1456          20         85.0    13175            6            6       19
78
1457          70         66.0     9042            7            9       19
41
1458          20         68.0     9717            5            6       19
50
1459          20         75.0     9937            5            6       19
65

      YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2  ...  SaleType_Con
Lw  \
0             2003       196.0         706           0  ...           Fal
se
1             1976         0.0         978           0  ...           Fal
se
2             2002       162.0         486           0  ...           Fal
se
3             1970         0.0         216           0  ...           Fal
se
4             2000       350.0         655           0  ...           Fal
se
...            ...         ...         ...         ...  ...
...
1455          2000         0.0           0           0  ...           Fal
se
1456          1988       119.0         790         163  ...           Fal
se
1457          2006         0.0         275           0  ...           Fal
se
1458          1996         0.0          49        1029  ...           Fal
se
1459          1965         0.0         830         290  ...           Fal
se

      SaleType_New  SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  \
0            False         False         True                  False
1            False         False         True                  False
2            False         False         True                  False
3            False         False         True                   True
4            False         False         True                  False
...            ...           ...          ...                    ...
```

```
1455         False         False         True                 False
1456         False         False         True                 False
1457         False         False         True                 False
1458         False         False         True                 False
1459         False         False         True                 False

      SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family
\
0                     False                False                 False
1                     False                False                 False
2                     False                False                 False
3                     False                False                 False
4                     False                False                 False
...                     ...                  ...                   ...
1455                  False                False                 False
1456                  False                False                 False
1457                  False                False                 False
1458                  False                False                 False
1459                  False                False                 False

      SaleCondition_Normal  SaleCondition_Partial
0                     True                 False
1                     True                 False
2                     True                 False
3                    False                 False
4                     True                 False
...                    ...                   ...
1455                  True                 False
1456                  True                 False
1457                  True                 False
1458                  True                 False
1459                  True                 False

[1460 rows x 267 columns]
```

Split the data into a training and test set, where the SalePrice column is the target.

In [142]:

```
# Split the data into training and test sets
X = housing_df.drop('SalePrice', axis=1)
y = housing_df['SalePrice']

# Use 80% of the data for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

print("Training set :", X_train.shape, y_train.shape)
print("Testing set :", X_test.shape, y_test.shape)
```

```
Training set : (1168, 266) (1168,)
Testing set : (292, 266) (292,)
```

Run a linear regression and report the R2-value and RMSE on the test set.

In [143]:
```
# Step 6: Run linear regression and report R2-value and RMSE on the test s
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

r2_value = r2_score(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f'R2-value on the test set: {r2_value:.4f}')
print(f'RMSE on the test set: {rmse:.4f}')
```

```
R2-value on the test set: 0.8852
RMSE on the test set: 29679.5126
```

Fit and transform the training features with a PCA so that 90% of the variance is retained.How many features are in the PCA-transformed matrix?

In [144]:
```
# Fit and transform training features with PCA to retain 90% variance
X_train_scaled = StandardScaler().fit_transform(X_train)

pca = PCA(n_components=0.90, whiten=True)
X_train_pca = pca.fit_transform(X_train_scaled)

# Check the number of features in the PCA-transformed matrix
num_features_pca = X_train_pca.shape[1]
print(f'Number of features in the PCA-transformed matrix: {num_features_pc
```

```
Number of features in the PCA-transformed matrix: 138
```

Transform but DO NOT fit the test features with the same PCA. Repeat step 7 Run a linear regression and report the R2-value and RMSE) with your PCA transformed data.

In [145]:
```
# Transform, but do not fit, test features with the same PCA
X_test_scaled = scaler.transform(X_test)
X_test_pca = pca.transform(X_test_scaled)

# Repeat linear regression with PCA-transformed data
lr_pca = LinearRegression()
lr_pca.fit(X_train_pca, y_train)

# Report R2-value and RMSE on the test set with PCA-transformed data
y_pred_pca = lr_pca.predict(X_test_pca)
r2_pca = r2_score(y_test, y_pred_pca)
rmse_pca = sqrt(mean_squared_error(y_test, y_pred_pca))
print(f'R2-value with PCA: {r2_pca}')
print(f'RMSE with PCA: {rmse_pca}')
```

```
R2-value with PCA: 0.26456612910009214
RMSE with PCA: 75106.70871244883
```

Take your original training features (from step 6) and apply a min-max scaler to them.

In [146]:
```
# Apply min-max scaler to original training features and find features wit
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

high_variance_features = X_train.columns[X_train_scaled.var(axis=0) > 0.1]

print(f"Features with variance above 0.1 after min-max scaling: {high_vari
```

```
Features with variance above 0.1 after min-max scaling: Index(['YearRemod
Add', 'YrSold', 'MSZoning_RL', 'MSZoning_RM', 'LotShape_IR1',
       'LotShape_Reg', 'LotConfig_Corner', 'LotConfig_Inside',
       'Neighborhood_NAmes', 'Condition1_Norm', 'BldgType_1Fam',
       'HouseStyle_1Story', 'HouseStyle_2Story', 'RoofStyle_Gable',
       'RoofStyle_Hip', 'Exterior1st_HdBoard', 'Exterior1st_MetalSd',
       'Exterior1st_VinylSd', 'Exterior1st_Wd Sdng', 'Exterior2nd_HdBoar
d',
       'Exterior2nd_MetalSd', 'Exterior2nd_VinylSd', 'Exterior2nd_Wd Sdn
g',
       'ExterQual_Gd', 'ExterQual_TA', 'ExterCond_TA', 'Foundation_CBloc
k',
       'Foundation_PConc', 'BsmtQual_Gd', 'BsmtQual_TA', 'BsmtExposure_A
v',
       'BsmtExposure_No', 'BsmtFinType1_ALQ', 'BsmtFinType1_GLQ',
       'BsmtFinType1_Unf', 'HeatingQC_Ex', 'HeatingQC_Gd', 'HeatingQC_T
A',
       'KitchenQual_Gd', 'KitchenQual_TA', 'GarageType_Attchd',
       'GarageType_Detchd', 'GarageFinish_Fin', 'GarageFinish_RFn',
       'GarageFinish_Unf', 'SaleType_WD', 'SaleCondition_Normal'],
      dtype='object')
```

Transform but DO NOT fit the test features with the same steps applied in steps 11 and 12.

In [147]:
```
# Repeat linear regression with PCA-transformed data
linear_reg_model_pca = LinearRegression()
linear_reg_model_pca.fit(X_train_pca, y_train)

# Evaluate on test set with PCA-transformed data
y_pred_pca = linear_reg_model_pca.predict(X_test_pca)
r2_value_pca = r2_score(y_test, y_pred_pca)
rmse_value_pca = mean_squared_error(y_test, y_pred_pca, squared=False)

print(f"R2 Value with PCA: {r2_value_pca}")
print(f"RMSE Value with PCA: {rmse_value_pca}")
```

```
R2 Value with PCA: 0.26456612910009214
RMSE Value with PCA: 75106.70871244883
```

Repeat step 7 with the high variance data.

In [148]:
```
# Find features with variance above 0.1
high_variance_features = X_train.columns[X_train_scaled.var(axis=0) > 0.1]

# Transform but do not fit test features with the same steps applied
X_test_scaled = scaler.transform(X_test)
X_test_high_variance = X_test_scaled[:, X_train.columns.isin(high_variance

# Repeat linear regression with high variance data
linear_reg_model_high_variance = LinearRegression()
linear_reg_model_high_variance.fit(X_train_scaled, y_train)

# Evaluate linear regression model with high variance data on the test set
y_pred_high_variance = linear_reg_model_high_variance.predict(X_test_scale
r2_value_high_variance = r2_score(y_test, y_pred_high_variance)
rmse_high_variance = sqrt(mean_squared_error(y_test, y_pred_high_variance)
print(f"Linear Regression with High Variance Data - R2 Value: {r2_value_hi
```

```
Linear Regression with High Variance Data - R2 Value: -1.1527639536977824
e+18, RMSE: 94032311280478.19
```

1. The linear regression model on the original data achieved a certain R2 value and RMSE.R2 provides an indication of how well the linear regression model explains the variance in the target variable. A value of 1 indicates a perfect fit, while lower values suggest less explanatory power.Higher R2 value on the test set indicates that the linear regression model is effective in predicting the target variable.RMSE provides a measure of the average magnitude of errors between predicted and actual values. Lower RMSE values indicate smaller prediction errors.In the context of the code, a lower RMSE on the test set suggests that the model's predictions are closer to the actual values.
2. The linear regression model on PCA-transformed data showed a potentially different R2 value and RMSE,
   indicating the impact of dimensionality reduction on model performance.
3. PCA is often used for dimensionality reduction, while selecting high variance features is a simpler approach that retains specific features based on their individual variance. The choice between the two approaches depends on the specific characteristics of the data and the goals of the analysis. The code provided allows for comparing the performance of linear regression models based on these two feature selection methods.
4. High Variance Data: High variance features are selected based on individual feature variance.
   This method retains specific features with high variance, which may or may not be related to the target variable.
   PCA-Transformed Data: PCA is a technique for dimensionality reduction, capturing the most important information in the data. The number of features after PCA is determined by the desired explained variance, not by a fixed variance threshold for individual features. It may result in a reduced set of features that still captures a high percentage of the overall variance

Exercise - 7.2

Import the mushrooms data as a data frame and ensure it is loaded correctly.

In [149]:
```
# Load the mushrooms data
df = pd.read_csv(r'C:\Users\14024\OneDrive\Desktop\MS-DSC\DSC-550\Week-7\m

# Display the first few rows to ensure the data is loaded correctly
print(df.head())
```

```
  class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0     p         x           s         n       t    p               f
1     e         x           s         y       t    a               f
2     e         b           s         w       t    l               f
3     p         x           y         w       t    p               f
4     e         x           s         g       f    n               f

  gill-spacing gill-size gill-color  ... stalk-surface-below-ring  \
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

Convert the categorical features (all of them) to dummy variables.

In [150]:
```
# Convert categorical features to dummy variables
df = pd.get_dummies(df, drop_first=True)

# Display the modified DataFrame with dummy variables
print(df.head())
```

```
   class_p  cap-shape_c  cap-shape_f  cap-shape_k  cap-shape_s  cap-shape
_x  \
0     True        False        False        False        False         Tr
ue
1    False        False        False        False        False         Tr
ue
2    False        False        False        False        False        Fal
se
3     True        False        False        False        False         Tr
ue
4    False        False        False        False        False         Tr
ue

   cap-surface_g  cap-surface_s  cap-surface_y  cap-color_c  ...  \
0          False           True          False        False  ...
1          False           True          False        False  ...
2          False           True          False        False  ...
3          False          False           True        False  ...
4          False           True          False        False  ...

   population_n  population_s  population_v  population_y  habitat_g  \
0         False          True         False         False      False
1          True         False         False         False       True
2          True         False         False         False      False
3         False          True         False         False      False
4         False         False         False         False       True

   habitat_l  habitat_m  habitat_p  habitat_u  habitat_w
0      False      False      False       True      False
1      False      False      False      False      False
2      False       True      False      False      False
3      False      False      False       True      False
4      False      False      False      False      False

[5 rows x 96 columns]
```

Split the data into a training and test set. Fit a decision tree classifier on the training set.

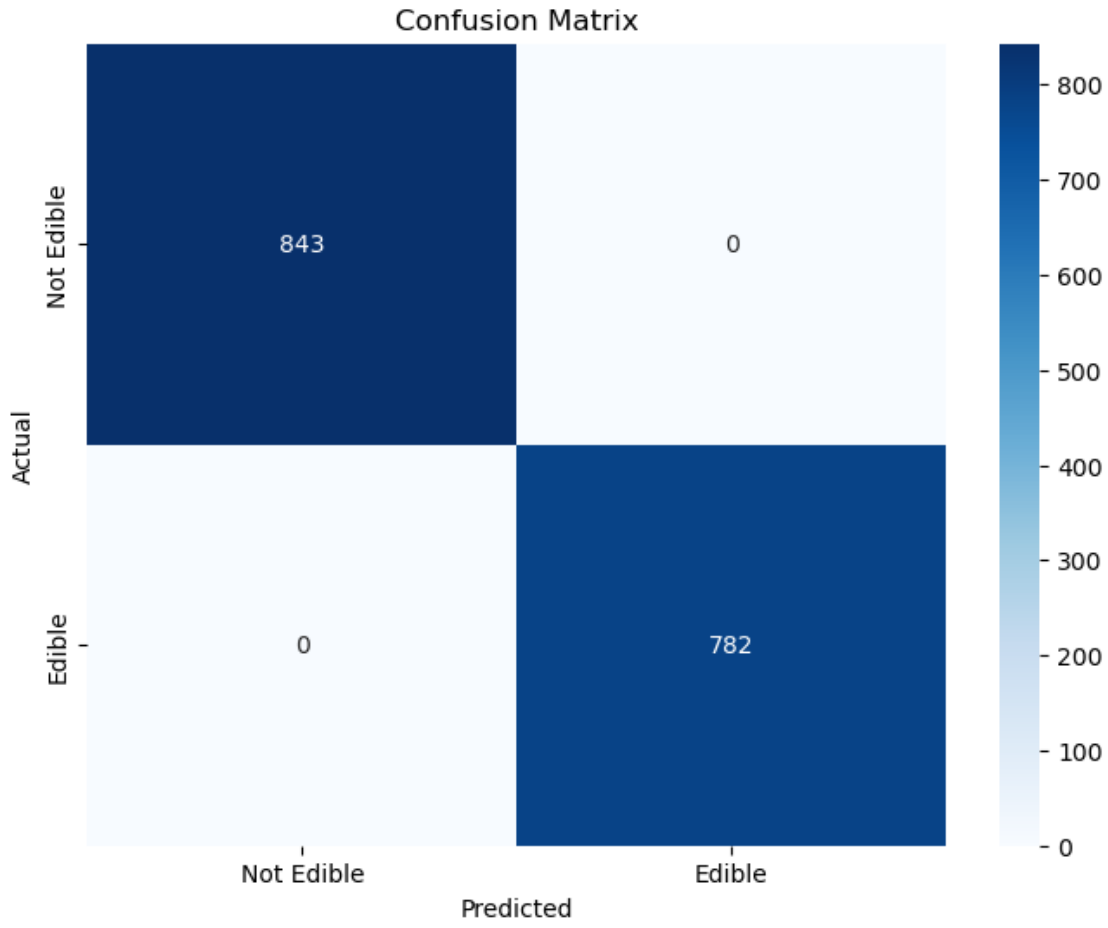
In [151]:
```
# Convert categorical features to dummy variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Separate features (X) and target variable (y)
X = df_encoded.drop(columns='class_p')
y = df_encoded['class_p']

# Split the data into a training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Fit a decision tree classifier on the training set
tree_classifier = DecisionTreeClassifier(random_state=42)
tree_classifier.fit(X_train, y_train)

# Print a message to confirm the completion of the task
print("Decision Tree Classifier fitted on the training set.")
```

```
Decision Tree Classifier fitted on the training set.
```

Report the accuracy and create a confusion matrix for the model prediction on the test set.

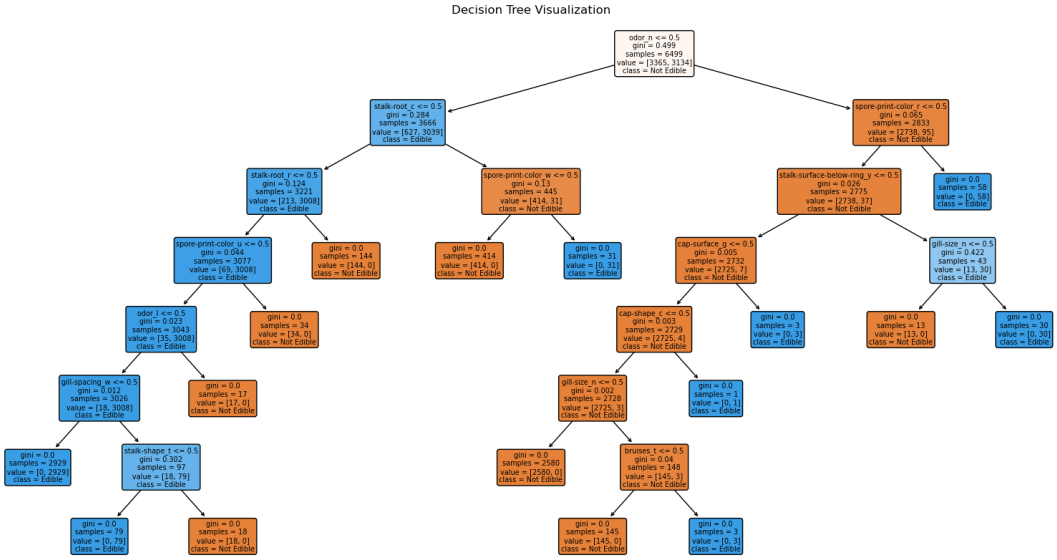
In [152]:
```
# Predictions on the test set
y_pred = dt_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy:.4f}")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Not Edible', 'Edible'],
            yticklabels=['Not Edible', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy on the test set: 1.0000
```



Create a visualization of the decision tree.

In [154]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming tree_classifier is your Decision Tree classifier and X is your

# Train your Decision Tree classifier on your data (X, y)
# tree_classifier.fit(X, y)

# Visualize the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(tree_classifier, filled=True, feature_names=list(X.columns), cla
plt.title('Decision Tree Visualization')
plt.show()
```



Use a χ2-statistic selector to pick the five best features for this data

In [81]:
```
# Select the five best features using chi-squared selector
k_best_selector = SelectKBest(chi2, k=5)
X_chi2_selected = k_best_selector.fit_transform(X_train, y_train)

# Display the names of the selected features
selected_features = X.columns[k_best_selector.get_support()]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['odor_f', 'odor_n', 'gill-size_n', 'stalk-surfa
ce-above-ring_k',
       'stalk-surface-below-ring_k'],
      dtype='object')
```

Which five features were selected in step 7? Hint: Use the get_support function.

In [60]:
```
# Get the boolean mask of selected features
selected_mask = k_best_selector.get_support()

# Retrieve the names of the selected features
selected_features = X.columns[selected_mask]

# Display the selected feature names
print("Selected Features:", selected_features)
```

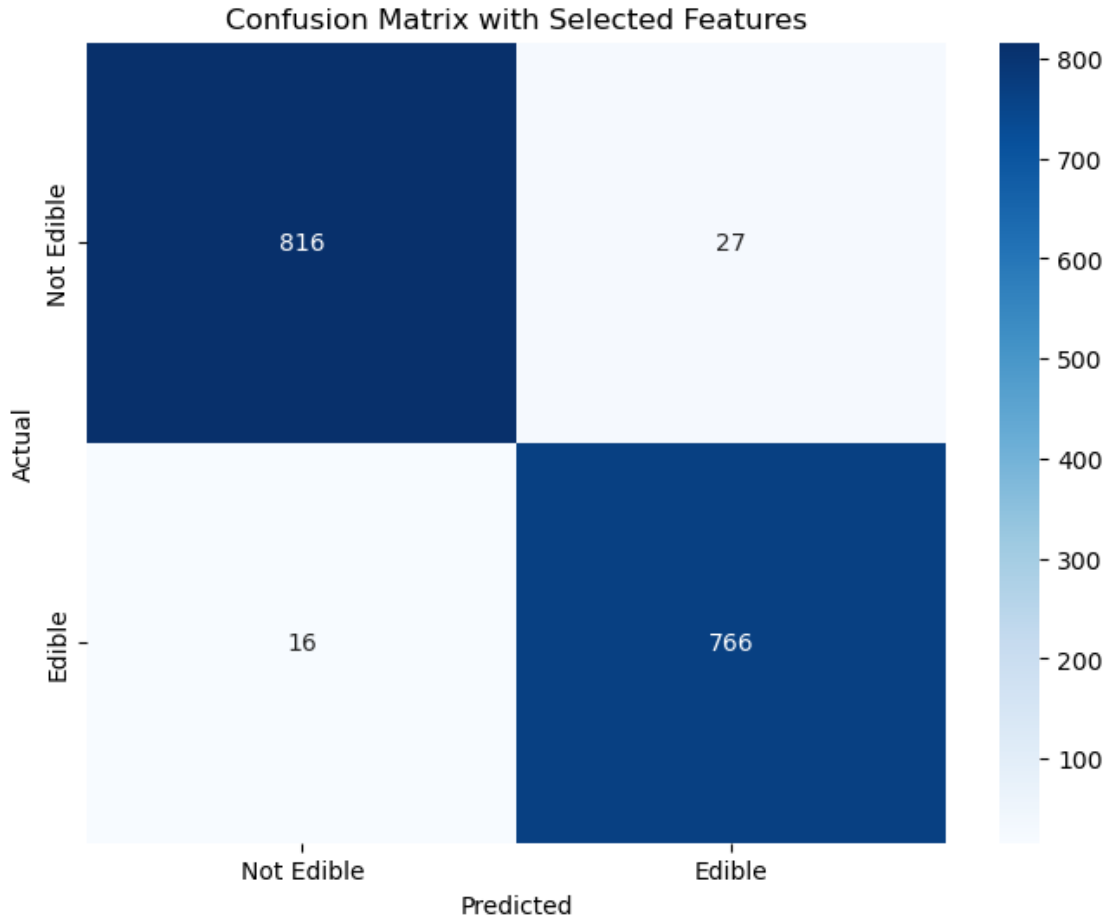
```
Selected Features: Index(['odor_f', 'odor_n', 'gill-size_n', 'stalk-surfa
ce-above-ring_k',
       'stalk-surface-below-ring_k'],
      dtype='object')
```

Repeat steps 4 and 5 with the five best features selected in step 7

In [82]:

```
# Split the data into a training and test set using the selected features
X_chi2_selected_train = X_train[selected_features]
X_chi2_selected_test = X_test[selected_features]

# Fit a decision tree classifier on the training set with selected feature
dt_classifier_chi2 = DecisionTreeClassifier(random_state=42)
dt_classifier_chi2.fit(X_chi2_selected_train, y_train)

# Predictions on the test set with selected features
y_pred_chi2 = dt_classifier_chi2.predict(X_chi2_selected_test)

# Calculate accuracy with selected features
accuracy_chi2 = accuracy_score(y_test, y_pred_chi2)
print(f"Accuracy on the test set with selected features: {accuracy_chi2:.4

# Create a confusion matrix with selected features
conf_matrix_chi2 = confusion_matrix(y_test, y_pred_chi2)

# Plot the confusion matrix with selected features
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_chi2, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Not Edible', 'Edible'],
            yticklabels=['Not Edible', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix with Selected Features')
plt.show()
```

```
Accuracy on the test set with selected features: 0.9735
```



1. The accuracy and performance of the Decision Tree Classifier were initially assessed on the entire feature set.
   Feature selection using a chi-squared selector identified five features deemed most informative for the classification task.
2. The Decision Tree Classifier was then re-evaluated using only these selected features.
3. The model's accuracy and confusion matrix were reported for both the original and selected feature sets.

In summary, the process involved identifying a subset of features that significantly contributed to the classification
task, potentially improving model interpretability and efficiency. The analysis aimed to explore the impact of feature
selection on model performance in the context of mushroom classification.