In [1]:
```
# Required Libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sc
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
# Specify the file path
file_path = r"D:\CV things\ML projects\parkinsons.data"

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Displaying the first five rows as an example
df
```

Out[2]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.(
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.(
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.(
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.(
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.(
...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.(
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.(
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.(
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.(
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.(

195 rows × 24 columns

In [3]:
```
#Matrix column entries (attributes):
#name - ASCII subject name and recording number
#MDVP:Fo(Hz) - Average vocal fundamental frequency
#MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
#MDVP:Flo(Hz) - Minimum vocal fundamental frequency
#MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several
#measures of variation in fundamental frequency
#MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA -
#NHR,HNR - Two measures of ratio of noise to tonal components in the voice
#status - Health status of the subject (one) - Parkinson's, (zero) - healthy
#RPDE,D2 - Two nonlinear dynamical complexity measures
#DFA - Signal fractal scaling exponent
#spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variati
```

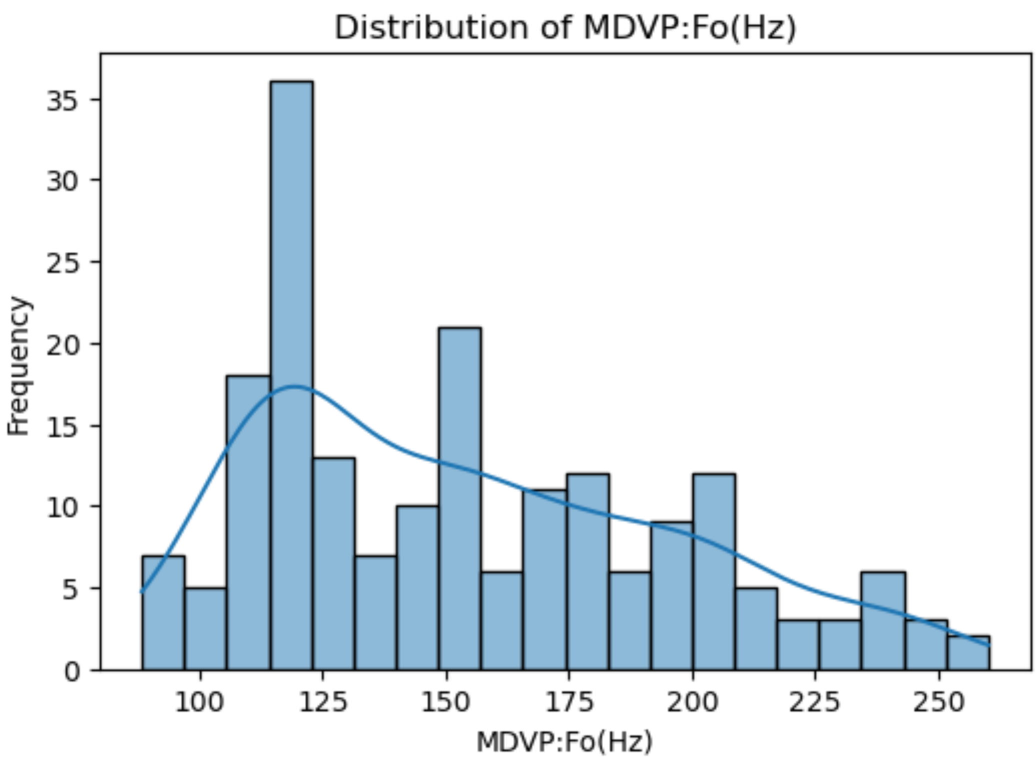
In [4]:
```python
# Drop 'name' column
df.drop(['name'], axis=1, inplace=True)

# Display DataFrame after dropping 'name' column
df.head()
```

Out[4]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP
0	119.992	157.302	74.997	0.00784	0.00007	0.00370
1	122.400	148.650	113.819	0.00968	0.00008	0.00465
2	116.682	131.111	111.555	0.01050	0.00009	0.00544
3	116.676	137.871	111.366	0.00997	0.00009	0.00502
4	116.014	141.781	110.655	0.01284	0.00011	0.00655

5 rows × 23 columns

In [5]:
```python
# Visualize distribution of 'MDVP:Fo(Hz)'
plt.figure(figsize=(6, 4))
sns.histplot(df['MDVP:Fo(Hz)'], bins=20, kde=True)
plt.title('Distribution of MDVP:Fo(Hz)')
plt.xlabel('MDVP:Fo(Hz)')
plt.ylabel('Frequency')
plt.show()
```



In [6]:
```python
# Define X (features) and Y (target)
X = df.drop('status', axis=1)
Y = df['status']

# Split data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_
```

In [7]:
```python
X.head()
```

Out[7]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP
0	119.992	157.302	74.997	0.00784	0.00007	0.00370
1	122.400	148.650	113.819	0.00968	0.00008	0.00465
2	116.682	131.111	111.555	0.01050	0.00009	0.00544
3	116.676	137.871	111.366	0.00997	0.00009	0.00502
4	116.014	141.781	110.655	0.01284	0.00011	0.00655

5 rows × 22 columns

In [8]:

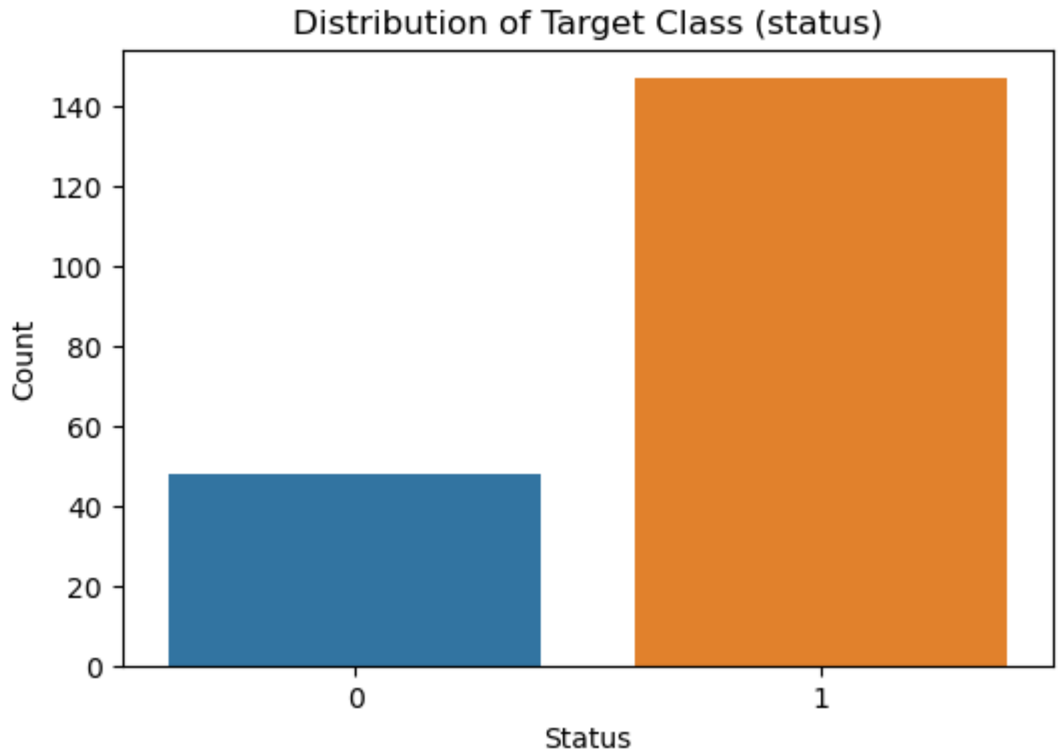
```
Y.head()
```

Out[8]:

```
0    1
1    1
2    1
3    1
4    1
Name: status, dtype: int64
```

In [9]:

```
# Visualize distribution of target class (status)
plt.figure(figsize=(6, 4))
sns.countplot(x='status', data=df)
plt.title('Distribution of Target Class (status)')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```



In [10]:

```python
# Initialize and fit a Random Forest model
RF = RandomForestClassifier(random_state=40)
RF.fit(X_train, Y_train)

# Predict on test data
test_preds_rf = RF.predict(X_test)

# Model evaluation after hyperparameter tuning
print("Model accuracy on test data (Before Hyperparameter Tuning):", accuracy_sc
print("Confusion matrix:\n", confusion_matrix(Y_test, test_preds_rf))
print("Classification Report:\n", classification_report(Y_test, test_preds_rf))
```

```
Model accuracy on test data (Before Hyperparameter Tuning): 0.8974358974358975
Confusion matrix:
 [[ 6  2]
 [ 2 29]]
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.75      0.75         8
           1       0.94      0.94      0.94        31

    accuracy                           0.90        39
   macro avg       0.84      0.84      0.84        39
weighted avg       0.90      0.90      0.90        39
```

In [11]:

```python
# Hyperparameter Tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


RF = RandomForestClassifier(random_state=40)
grid_search = GridSearchCV(estimator=RF, param_grid=param_grid, cv=5, scoring='a
grid_search.fit(X_train, Y_train)
```

Out[11]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=40), n_jobs=-1,
             param_grid={'max_depth': [None, 5, 10, 15],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]},
             scoring='accuracy')
```

In [12]:

```python
# Get the best parameters
best_params = grid_search.best_params_

print("Best parameters found:", best_params)

# Initialize and fit a Random Forest model with optimized hyperparameters
best_RF = RandomForestClassifier(random_state=40, **best_params)
best_RF.fit(X_train, Y_train)

# Predict on test data
test_preds_best_RF = best_RF.predict(X_test)
```

```
Best parameters found: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_s
plit': 2, 'n_estimators': 200}
```

In [13]:
```
# Perform cross-validation
cv_scores = cross_val_score(best_RF, X, Y, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.76923077 0.82051282 0.92307692 0.74358974 0.7435897
4]
Mean CV accuracy: 0.8
```

In [14]:
```
# Model evaluation after hyperparameter tuning
print("Model accuracy on test data (After Hyperparameter Tuning):", accuracy_sco
print("Confusion matrix:\n", confusion_matrix(Y_test, test_preds_rf))
print("Classification Report:\n", classification_report(Y_test, test_preds_rf))
```

```
Model accuracy on test data (After Hyperparameter Tuning): 0.8974358974358975
Confusion matrix:
 [[ 6  2]
 [ 2 29]]
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.75      0.75         8
           1       0.94      0.94      0.94        31

    accuracy                           0.90        39
   macro avg       0.84      0.84      0.84        39
weighted avg       0.90      0.90      0.90        39
```

In [15]:
```
# Feature Importance
feature_importances = best_RF.feature_importances_

# Create DataFrame for feature importance
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': featur
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascen
```

In [16]:
```
# Update table after feature importance calculation
print("\nTable after Feature Importance calculation:")
feature_importance_df
```

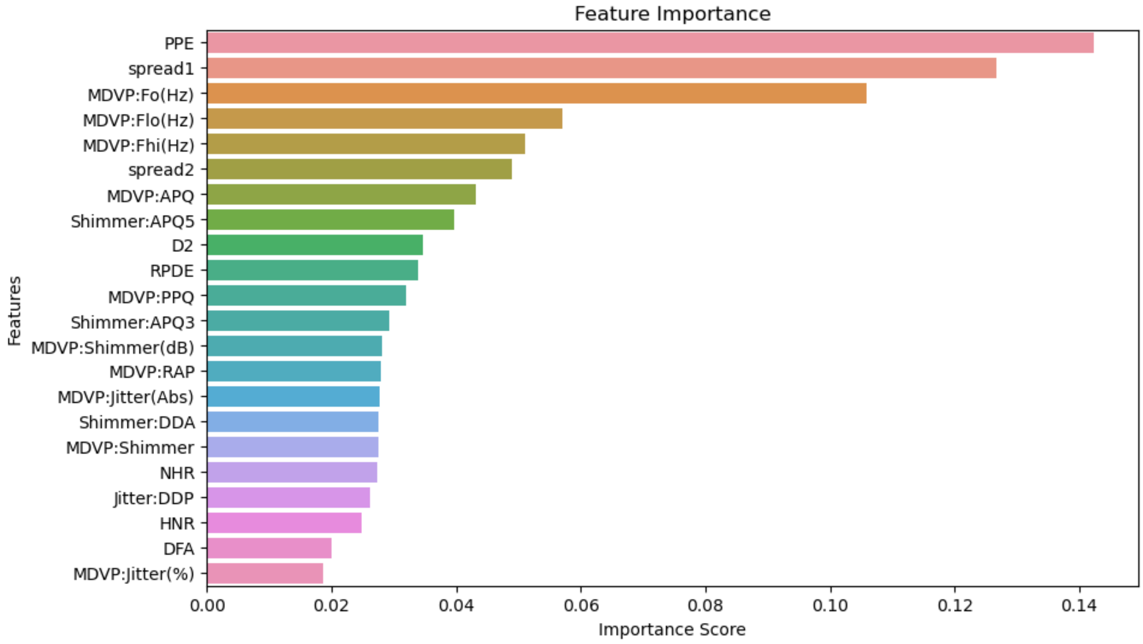
```
Table after Feature Importance calculation:
```

Out[16]:

	Feature	Importance
21	PPE	0.142391
18	spread1	0.126634
0	MDVP:Fo(Hz)	0.105773
2	MDVP:Flo(Hz)	0.056965
1	MDVP:Fhi(Hz)	0.051041
19	spread2	0.048878
12	MDVP:APQ	0.043084
11	Shimmer:APQ5	0.039633
20	D2	0.034715
16	RPDE	0.033922
6	MDVP:PPQ	0.031989
10	Shimmer:APQ3	0.029326
9	MDVP:Shimmer(dB)	0.028092
5	MDVP:RAP	0.027834
4	MDVP:Jitter(Abs)	0.027726
13	Shimmer:DDA	0.027513
8	MDVP:Shimmer	0.027495
14	NHR	0.027349
7	Jitter:DDP	0.026149
15	HNR	0.024834
17	DFA	0.019929
3	MDVP:Jitter(%)	0.018731

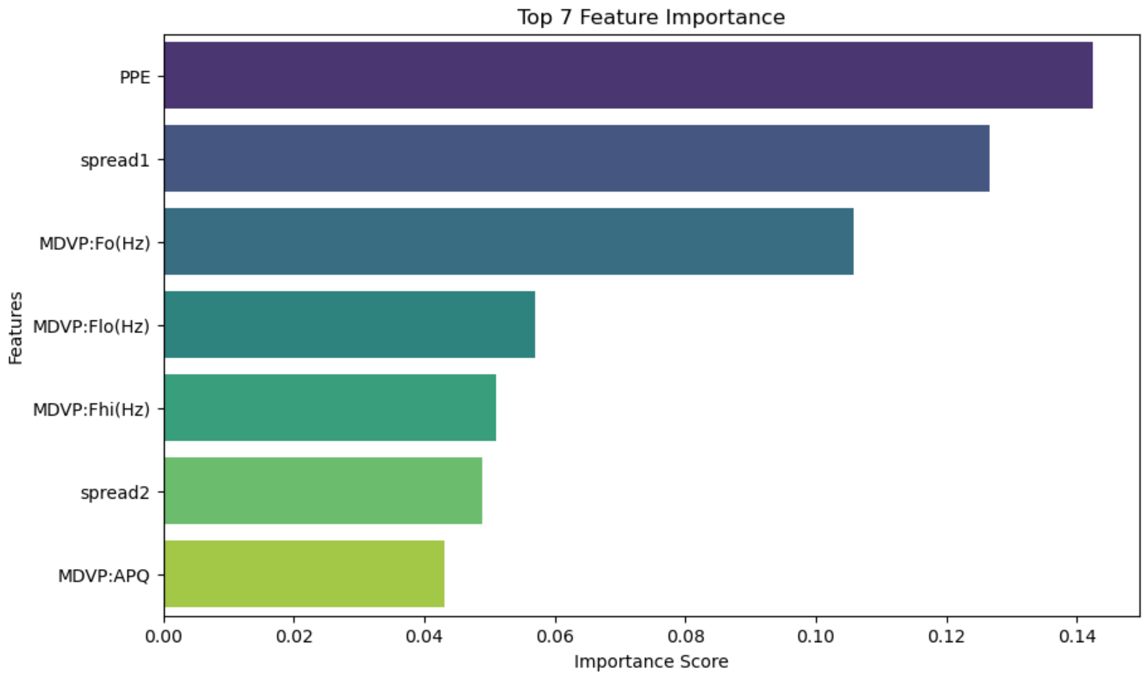
In [17]:

```
# Visualize feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



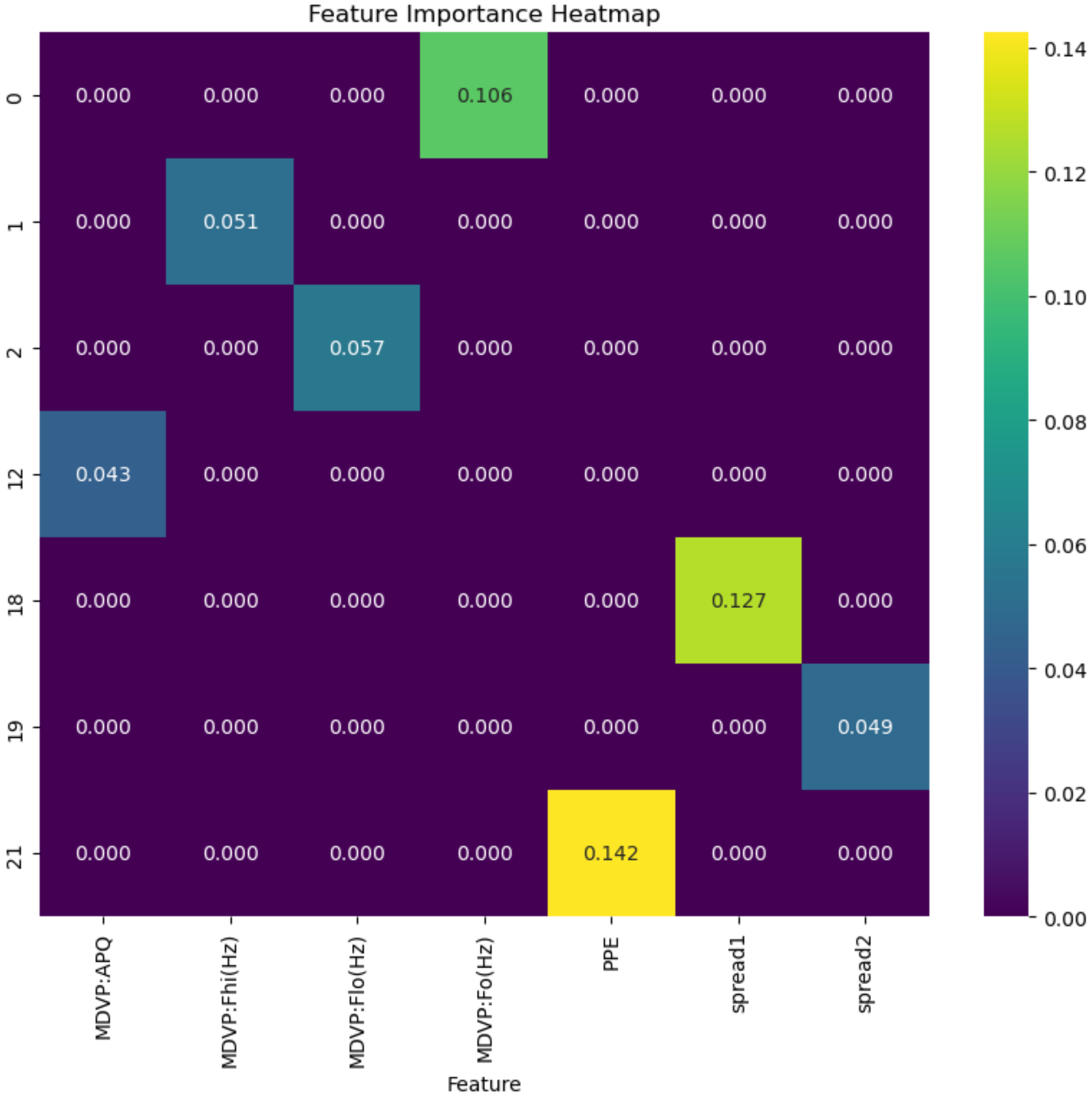
In [18]:
```
# Select top features based on importance scores (e.g., top 7 features)
top_features = feature_importance_df['Feature'][:7].tolist()

# Create new X with selected top features
X_top_features = df[top_features]

# Splitting data with the top selected features
X_train_top, X_test_top, Y_train, Y_test = train_test_split(X_top_features, Y, t
```

In [19]:
```
# Create a new DataFrame for feature importance with the top 7 features
top_features_df = feature_importance_df.head(7)

# Visualize feature importance using bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=top_features_df, palette='viridis'
plt.title('Top 7 Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



In [20]:

```
# Create a pivot table for the feature importance data
pivot_df = top_features_df.pivot(index=None, columns='Feature', values='Importan

# Create the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(pivot_df, cmap='viridis', annot=True, fmt='.3f')
plt.title('Feature Importance Heatmap')
plt.show()
```



In [21]:

```
# Initializing and fitting the Random Forest model with the selected features
best_params = {'n_estimators': 200, 'max_depth': None, 'min_samples_leaf': 1, 'm
best_RF_selected = RandomForestClassifier(random_state=40, **best_params)
best_RF_selected.fit(X_train_top, Y_train)

# Predict on test data with selected features
test_preds_rf_selected = best_RF_selected.predict(X_test_top)
```

In [22]:

```
#Model evaluation with selected features
print("\nModel accuracy on test data (After Feature Selection):", accuracy_score
print("Confusion matrix:\n", confusion_matrix(Y_test, test_preds_rf_selected))
print("Classification Report:\n", classification_report(Y_test, test_preds_rf_se
```

```
Model accuracy on test data (After Feature Selection): 0.9230769230769231
Confusion matrix:
 [[ 6  2]
 [ 1 30]]
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.75      0.80         8
           1       0.94      0.97      0.95        31

    accuracy                           0.92        39
   macro avg       0.90      0.86      0.88        39
weighted avg       0.92      0.92      0.92        39
```

In [23]:
```
acc_before_feature_selection = accuracy_score(Y_test, test_preds_rf)*100
acc_after_feature_selection = accuracy_score(Y_test, test_preds_rf_selected)*100

# Assuming acc_before_feature_selection and acc_after_feature_selection are accu
accuracy_scores = [acc_before_feature_selection, acc_after_feature_selection]
labels = ['Before Selection', 'After Selection']

plt.figure(figsize=(6, 4))
sns.barplot(x=labels, y=accuracy_scores)
plt.title('Model Accuracy Before and After Feature Selection')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0, 100)  # Set the y-axis limits to match accuracy values (0 to 100)
plt.show()
```