In [2]:
```
import pandas as pd

#read in dataset
df = pd.read_csv('C:/Users/rachr/Rowey-DATA1030-Project/wpbc.data')
df.columns = ['ID number', 'outcome', 'time', 'radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1', 'compactne
#print(df)
```

In [3]:
```
#feature engineering
df['fractal_dimension_avg'] = df[['fractal_dimension1', 'fractal_dimension2', 'fractal_dimension3']].mean(axis=1)
df['symmetry_avg'] = df[['symmetry1', 'symmetry2', 'symmetry3']].mean(axis=1)
df['concave_points_avg'] = df[['concave_points1', 'concave_points2', 'concave_points3']].mean(axis=1)
df['concavity_avg'] = df[['concavity1', 'concavity2', 'concavity3']].mean(axis=1)
df['compactness_avg'] = df[['compactness1', 'compactness2', 'compactness3']].mean(axis=1)
df['smoothness_avg'] = df[['smoothness1', 'smoothness2', 'smoothness3']].mean(axis=1)
df['area_avg'] = df[['area1', 'area2', 'area3']].mean(axis=1)
df['perimeter_avg'] = df[['perimeter1', 'perimeter2', 'perimeter3']].mean(axis=1)
df['texture_avg'] = df[['texture1', 'texture2', 'texture3']].mean(axis=1)
df['radius_avg'] = df[['radius1', 'radius2', 'radius3']].mean(axis=1)
```

In [5]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import OrdinalEncoder, StandardScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

#dataset
y = df['outcome']
X = df[['time', 'radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1', 'compactness1', 'concavity1', 'concave_p
        'symmetry1', 'fractal_dimension1', 'radius2', 'texture2', 'perimeter2', 'area2', 'smoothness2', 'compactness2
        'symmetry2', 'fractal_dimension2', 'radius3', 'texture3', 'perimeter3', 'area3', 'smoothness3', 'compactness3
        'symmetry3', 'fractal_dimension3', 'tumor_size', 'lymph_node_status', 'compactness_avg', 'radius_avg', 'textu
        'smoothness_avg', 'concavity_avg', 'concave_points_avg', 'symmetry_avg', 'fractal_dimension_avg']]

#convert target variable values [N, R] to numerical data [0, 1]
df['outcome'] = df['outcome'].replace({'N': 0, 'R': 1}).astype(int)

#define categorical and numerical features
categorical_features = ['lymph_node_status']
```

```
numerical_features = X.columns.difference(categorical_features)

#categorical pipeline
ord_cats = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '13', '14', '15', '16', '17', '18', '20',
categorical_pipeline = Pipeline([
    ('impute', SimpleImputer(strategy='constant', fill_value='?')),
    ('ordinal', OrdinalEncoder(categories=[ord_cats]))
])

#preprocess all features
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', categorical_pipeline, categorical_features),
        ('num', StandardScaler(), numerical_features)
    ]
)
```

In [6]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import fbeta_score, accuracy_score
import numpy as np
import pandas as pd

#define pipeline
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=10000, random_state=42))
])

#define parameter grid
param_grid = {
    'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100], #regularization strength
    'classifier__penalty': ['l1', 'l2'], #regularization type
    'classifier__solver': ['liblinear', 'saga'], #solvers for l1 and l2 penalties
}

#random states
random_states = [0, 44, 123, 2024, 5678]

#store results for each random state
best_params_list = []
```

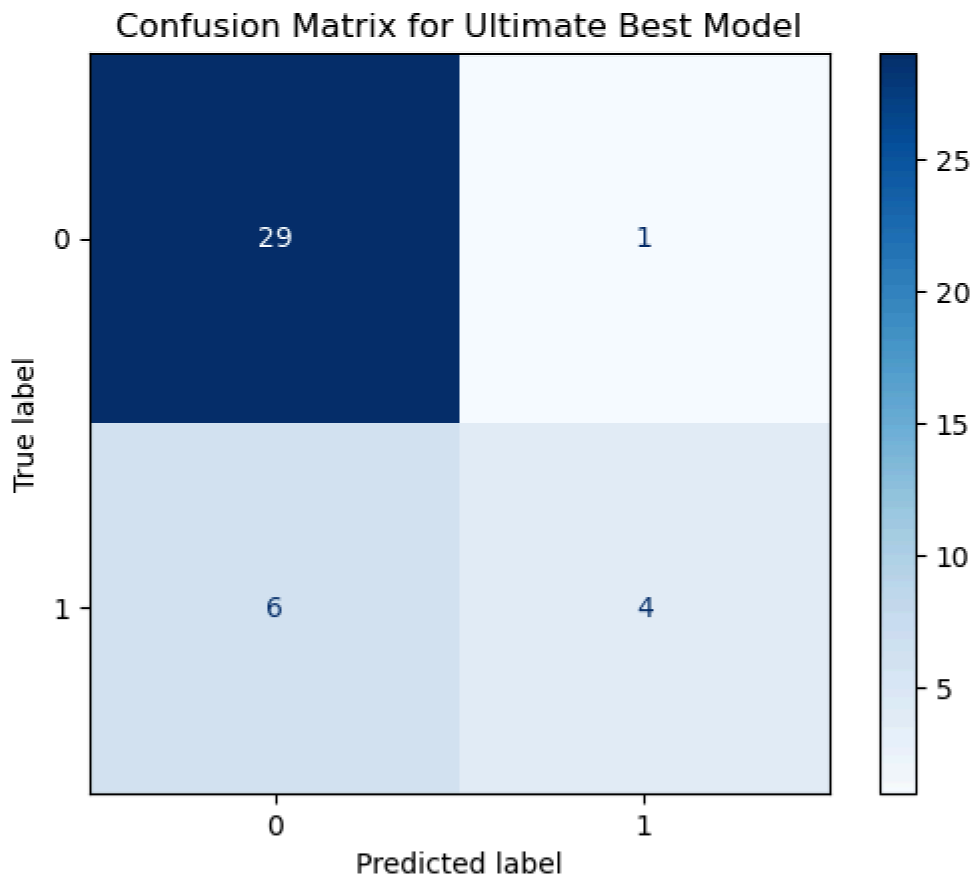
```python
best_cv_scores = []
best_test_scores = []
best_models = []
best_test_fscores = []

for state in random_states:
    print(f"\nRandom State: {state}")

    #stratified train-test split
    X_other, X_test, y_other, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=state
    )
    print('Test data balance:', np.unique(y_test, return_counts=True))

    #stratified K-Fold Cross-Validation
    kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=state)


    #GridSearchCV for hyperparameter tuning
    grid_search = GridSearchCV(
        estimator=model_pipeline,
        param_grid=param_grid,
        cv=kf,
        scoring='accuracy',
        n_jobs=-1,
    )

    #fit GridSearchCV on training data (X_other, y_other)
    grid_search.fit(X_other, y_other)

    #evaluate on test set
    best_model = grid_search.best_estimator_
    y_test_pred = best_model.predict(X_test)

    #accuracy and f-scores
    test_score = accuracy_score(y_test, y_test_pred)
    fscore = fbeta_score(y_test, y_test_pred, beta=1, average='binary')

    # Store best parameters, scores, and models
    best_params_list.append(grid_search.best_params_)
    best_cv_scores.append(grid_search.best_score_)
    best_test_scores.append(test_score)
    best_test_fscores.append(fscore)
    best_models.append(best_model)

    print(f"  Best Parameters for this state: {grid_search.best_params_}")
    print(f"  Best CV Score for this state: {grid_search.best_score_:.4f}")
    print(f"  Test Score: {test_score:.4f}")
    print(f"  Test F-Score: {fscore:.4f}")

#select best model based on test score
ultimate_best_idx = np.argmax(best_cv_scores)
ultimate_best_params = best_params_list[ultimate_best_idx]
ultimate_best_model = best_models[ultimate_best_idx]

#print best parameters
print("\nUltimate Best Model Across Random States:")
print(f"  Best Parameters: {ultimate_best_params}")
print(f"  Best CV Score: {best_cv_scores[ultimate_best_idx]:.4f}")
```

```
Random State: 0
Test data balance: (array([0, 1]), array([30, 10], dtype=int64))
  Best Parameters for this state: {'classifier__C': 10, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinea
r'}
  Best CV Score for this state: 0.7897
  Test Score: 0.8250
  Test F-Score: 0.6316

Random State: 44
Test data balance: (array([0, 1]), array([30, 10], dtype=int64))
  Best Parameters for this state: {'classifier__C': 1, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinea
r'}
  Best CV Score for this state: 0.7895
  Test Score: 0.7500
  Test F-Score: 0.5455

Random State: 123
Test data balance: (array([0, 1]), array([30, 10], dtype=int64))
  Best Parameters for this state: {'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'saga'}
  Best CV Score for this state: 0.7893
  Test Score: 0.8250
  Test F-Score: 0.5882

Random State: 2024
Test data balance: (array([0, 1]), array([30, 10], dtype=int64))
  Best Parameters for this state: {'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'saga'}
  Best CV Score for this state: 0.7829
  Test Score: 0.8000
  Test F-Score: 0.6364

Random State: 5678
Test data balance: (array([0, 1]), array([30, 10], dtype=int64))
  Best Parameters for this state: {'classifier__C': 1, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
  Best CV Score for this state: 0.7903
  Test Score: 0.8250
  Test F-Score: 0.5333

Ultimate Best Model Across Random States:
  Best Parameters: {'classifier__C': 1, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
  Best CV Score: 0.7903
```

In [7]:
```
#test scores summary (accuracy and fbeta scores)
mean_test_score = np.mean(best_test_scores)
std_test_score = np.std(best_test_scores)
mean_fscore = np.mean(best_test_fscores)
std_fscore = np.std(best_test_fscores)

print("\nTest Score Summary Across Random States:")
print(f"Mean Test Score: {mean_test_score:.4f}")
print(f"Standard Deviation of Test Scores: {std_test_score:.4f}")
print(f"Mean F-Score: {mean_fscore:.4f}")
print(f"Standard Deviation of F-Score: {std_fscore:.4f}")
```

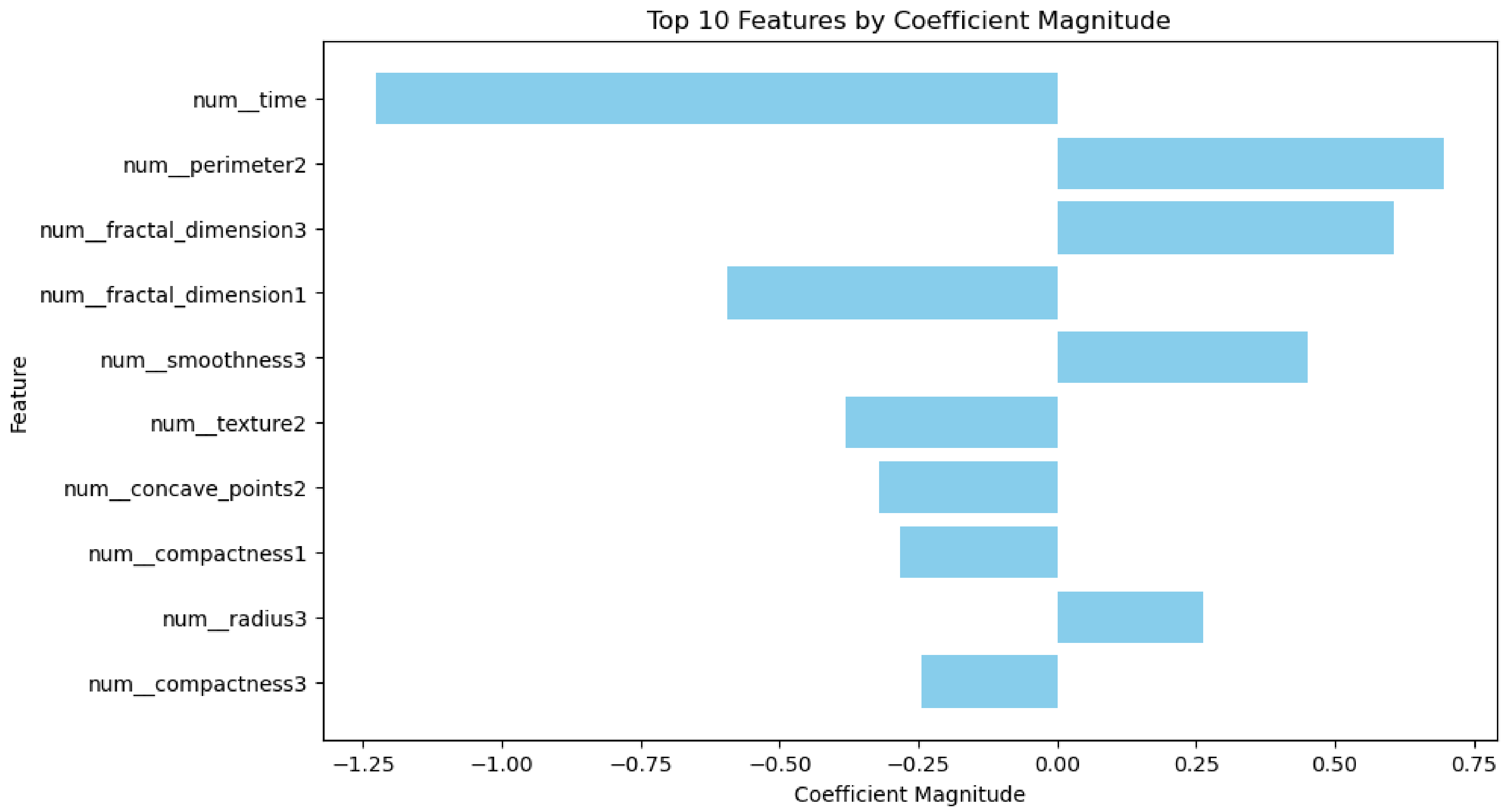
```
Test Score Summary Across Random States:
Mean Test Score: 0.8050
Standard Deviation of Test Scores: 0.0292
Mean F-Score: 0.5870
Standard Deviation of F-Score: 0.0425
```

In [16]:
```
#confusion matrix for best model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

y_test_pred = ultimate_best_model.predict(X_test)
cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=ultimate_best_model.classes_)
disp.plot(cmap='Blues')
disp.ax_.set_title("Confusion Matrix for Ultimate Best Model")
plt.savefig('cm.png', dpi=300)
```



In [19]:
```
#save info to separate csv file

ultimate_best_predictions = ultimate_best_model.predict(X_test)

#create dictionary
best_model_info = {
    "Random State": random_states[ultimate_best_idx],
    "Best Parameters": ultimate_best_params,
    "Best CV Score": best_cv_scores[ultimate_best_idx],
    "Best Test Accuracy": best_test_scores[ultimate_best_idx],
    "Best Test F-Score": best_test_fscores[ultimate_best_idx],
    "Mean Test Score": mean_test_score,
    "Std Test Score": std_test_score,
    "Mean F-Score": mean_fscore,
    "Std F-Score": std_fscore,
    "Test Predictions": list(ultimate_best_predictions)  # Save predictions as a list
}

#convert dictionary to pd dataframe
best_model_df = pd.DataFrame([best_model_info])

#save to csv
best_model_df.to_csv("model_info_preds.csv", index=False)
```

In [11]:
```
#global feature importance (coefficients)
import numpy as np
import matplotlib.pyplot as plt

logreg_model = ultimate_best_model.named_steps['classifier']
feature_names = ultimate_best_model.named_steps['preprocessor'].get_feature_names_out()
coefficients = logreg_model.coef_[0]
feature_importance = list(zip(feature_names, coefficients))
sorted_features = sorted(feature_importance, key=lambda x: abs(x[1]), reverse=True)
next_top_features = sorted_features[:10]
next_top_feature_names, next_top_coefficients = zip(*next_top_features)

#plot
plt.figure(figsize=(10, 6))
plt.barh(next_top_feature_names, next_top_coefficients, color='skyblue')
plt.xlabel('Coefficient Magnitude')
plt.ylabel('Feature')
plt.title('Top 10 Features by Coefficient Magnitude')
plt.gca().invert_yaxis()
plt.show()
plt.savefig('coefficient_mag_GI.png', dpi=300)
```

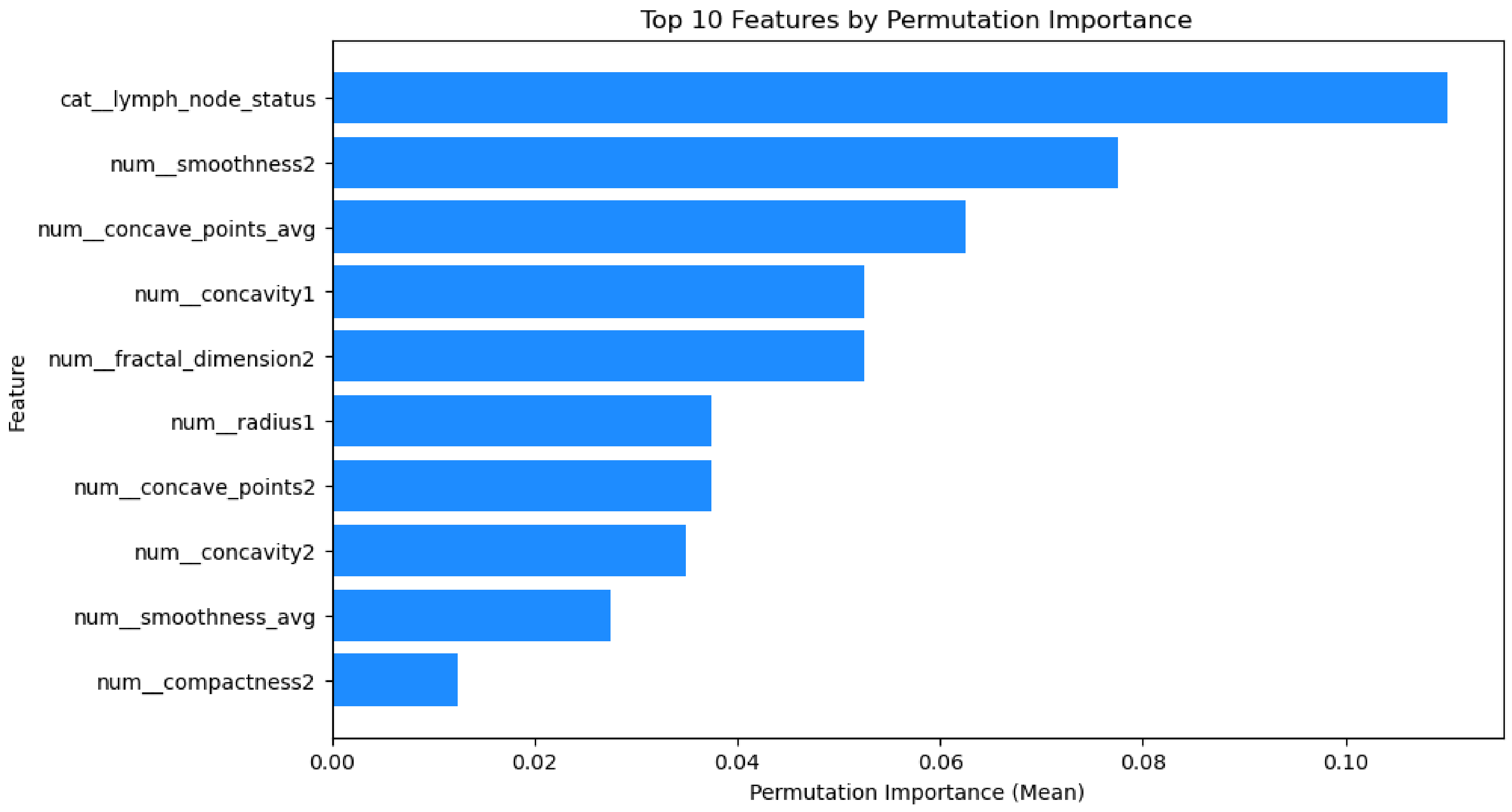


```
<Figure size 640x480 with 0 Axes>
```

In [12]:

```
#global feature importance (permutation)
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

perm_importance = permutation_importance(
    ultimate_best_model, X_test, y_test, scoring='accuracy', n_repeats=10, random_state=42
)
feature_names = ultimate_best_model.named_steps['preprocessor'].get_feature_names_out()
perm_importance_mean = perm_importance.importances_mean
perm_feature_importance = list(zip(feature_names, perm_importance_mean))
sorted_perm_importance = sorted(perm_feature_importance, key=lambda x: x[1], reverse=True)
top_perm_features = sorted_perm_importance[:10]
```

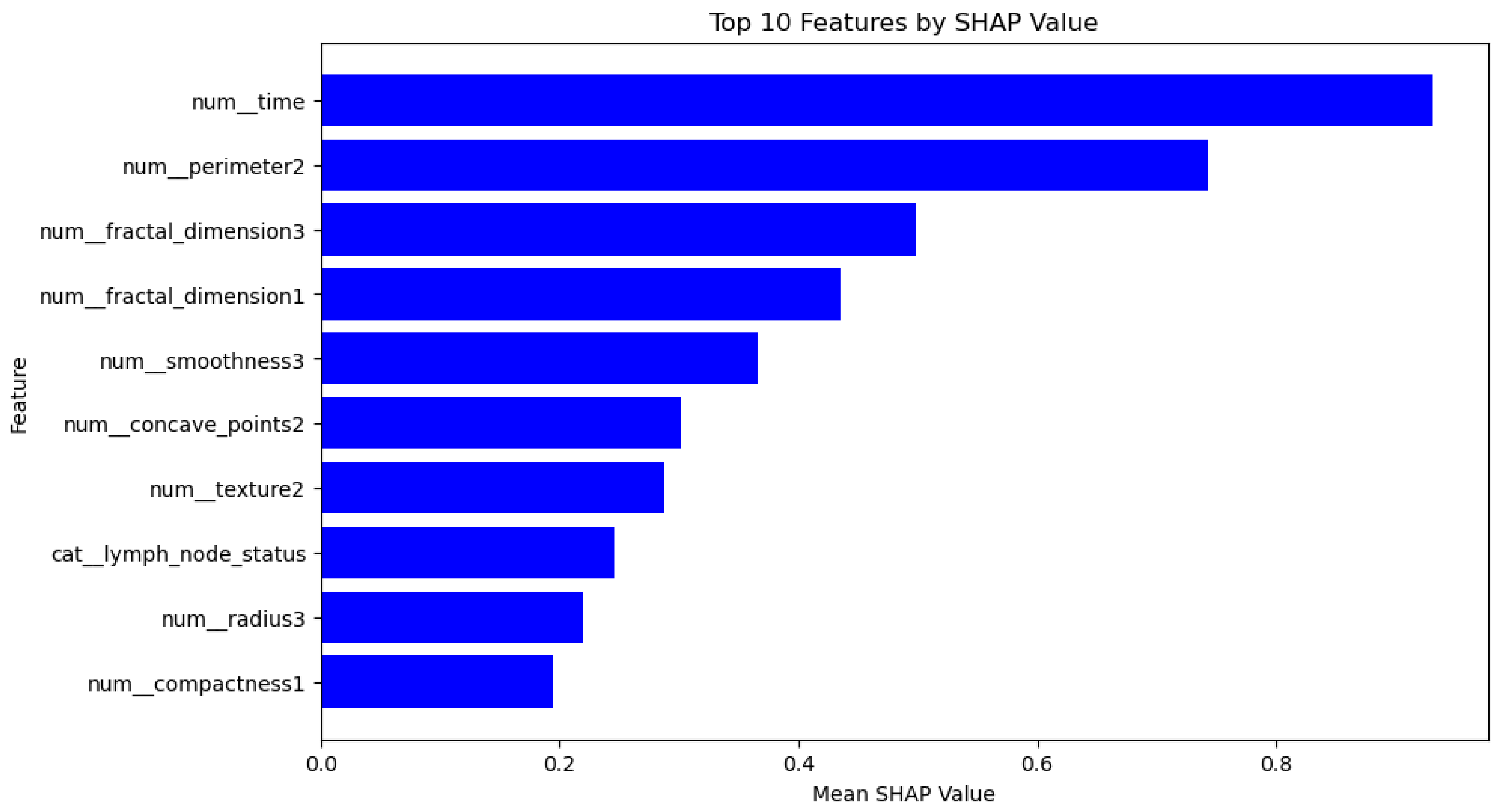
```
top_perm_feature_names, top_perm_importances = zip(*top_perm_features)

#plot
plt.figure(figsize=(10, 6))
plt.barh(top_perm_feature_names, top_perm_importances, color='dodgerblue')
plt.xlabel('Permutation Importance (Mean)')
plt.ylabel('Feature')
plt.title('Top 10 Features by Permutation Importance')
plt.gca().invert_yaxis()
plt.show()
plt.savefig('permutation_GI.png', dpi=300)
```



```
<Figure size 640x480 with 0 Axes>
```

In [13]:

```
#global feature importance (SHAP)
import shap
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

best_model = ultimate_best_model
X_test_transformed = best_model.named_steps['preprocessor'].transform(X_test)
feature_names = best_model.named_steps['preprocessor'].get_feature_names_out()
explainer = shap.Explainer(best_model.named_steps['classifier'], X_test_transformed)
shap_values_test = explainer(X_test_transformed)

#mean SHAP values for global
mean_abs_shap = np.abs(shap_values_test.values).mean(axis=0)
feature_importance = pd.DataFrame({
    "Feature": feature_names,
    "Mean SHAP Value": mean_abs_shap
}).sort_values(by="Mean SHAP Value", ascending=False)

top_features = feature_importance.iloc[:10]  # Select the first 10 rows

#plot
plt.figure(figsize=(10, 6))
colors = ['blue']  # Alternating colors
plt.barh(top_features["Feature"], top_features["Mean SHAP Value"], color=colors)
plt.xlabel("Mean SHAP Value")
plt.ylabel("Feature")
plt.title("Top 10 Features by SHAP Value")
plt.gca().invert_yaxis()  # Most important feature on top
plt.show()
plt.savefig('SHAP_GI.png', dpi=300)
```

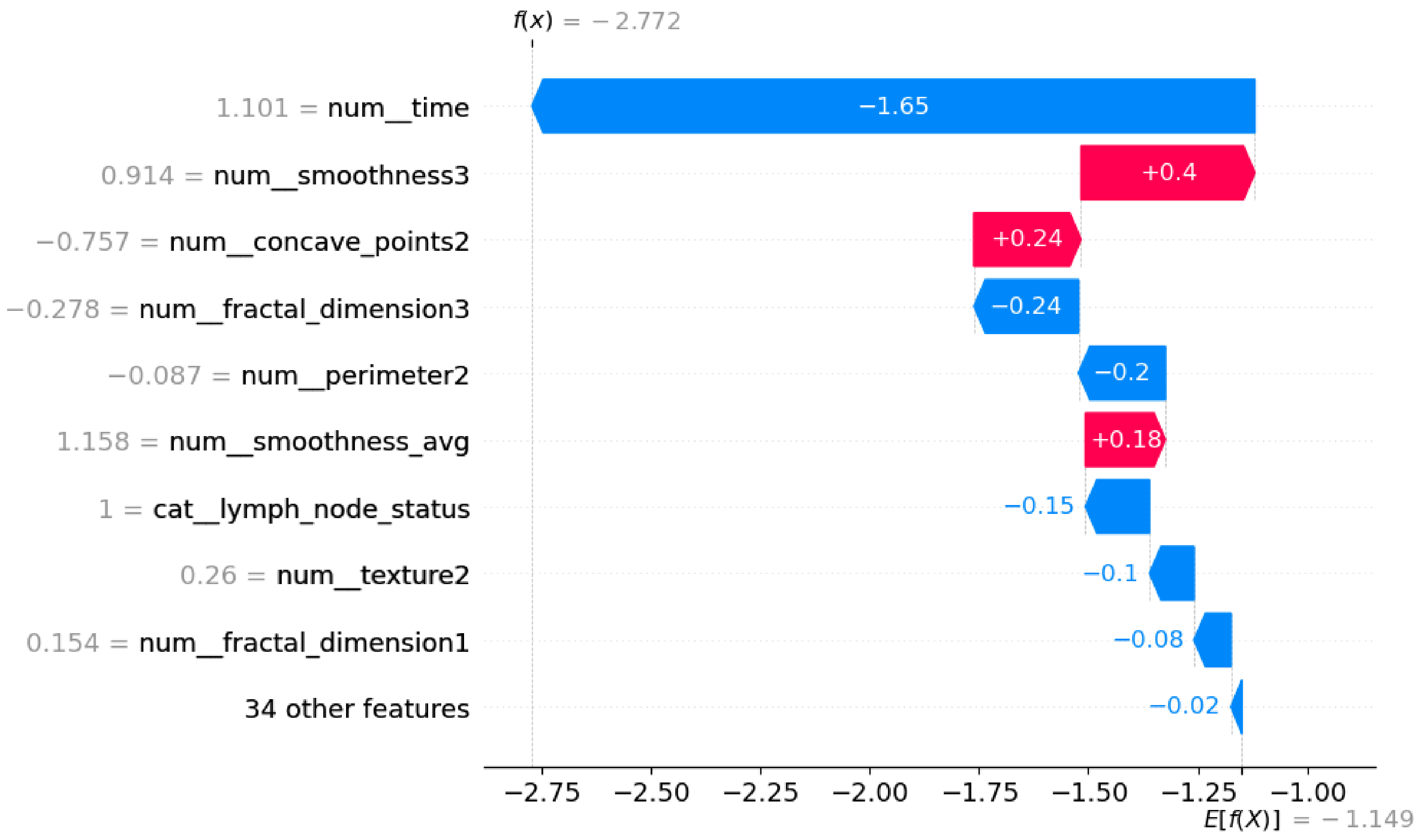


```
<Figure size 640x480 with 0 Axes>
```

In [15]:
```
#local feature importance (SHAP)
import shap
import matplotlib.pyplot as plt
import pandas as pd

best_model = ultimate_best_model
X_test_transformed = best_model.named_steps['preprocessor'].transform(X_test)
feature_names = best_model.named_steps['preprocessor'].get_feature_names_out()
X_test_transformed_df = pd.DataFrame(X_test_transformed, columns=feature_names)
explainer = shap.Explainer(best_model.named_steps['classifier'], X_test_transformed_df)
shap_values_test = explainer(X_test_transformed_df)
```

```
print("\nLocal Interpretability: Waterfall Plot for a Test Sample")
sample_idx = 0
shap.waterfall_plot(shap_values_test[sample_idx])
plt.show()
plt.savefig('SHAP_LI.png', dpi=300)
```

```
Local Interpretability: Waterfall Plot for a Test Sample
```



```
<Figure size 640x480 with 0 Axes>
```

In [ ]: