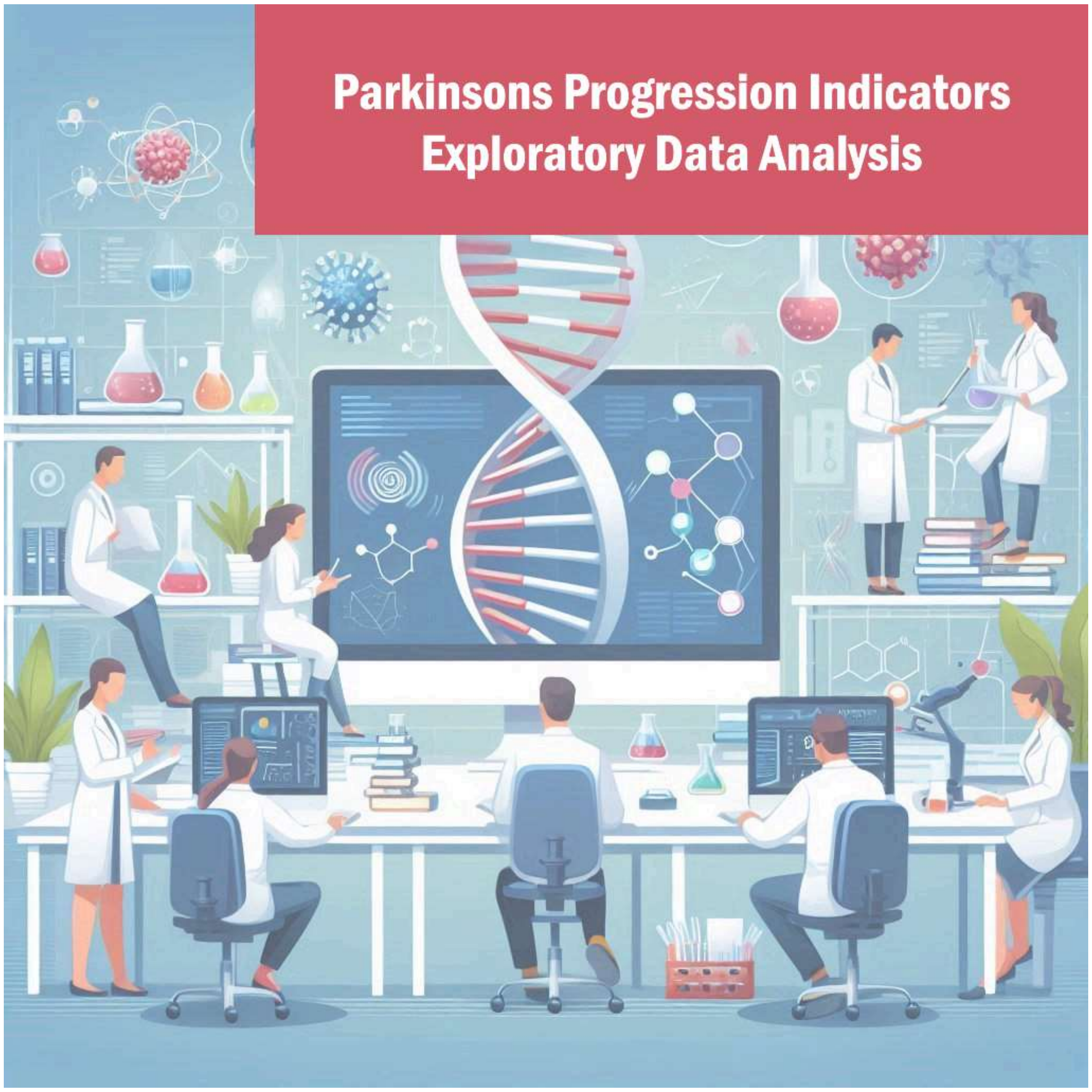


Biomarkers in Neuronal Synuclein Disease

Analysis by Joan Jaylani

Data downloaded with permission from Parkinson's Progression Markers Initiative (PPMI)
https://www.ppmi-info.org/

1. Introduction

Neuronal a-synuclein disease (NSD), refers to a group of neurodegenerative disorders which all have an anomalous accumulation of alpha-synuclein protein, synocleopathies, in neurons. This accumulation affects the neurons in the brain responsible for producing dopamine, a neurotransmitter necessary for coordinated movement. When these neurons become damaged or die, they cause diseases like Parkinson's disease (PD) and Dementia with Lewy Bodies (DLB).

The standard in clinical practice is to diagnose Parkinson's disease by observing symptoms rather than by a blood test or other measurable biological test. Scientists have been researching biomarkers however, including skin, blood, DNA and cerebrospinal fluid to try to find a more definitive test to diagnose NSD.

PPMI is a global observation study that research biomarkers for Parkinson's Disease and related neurological disorders. The PPMI has the largest collection of images, clinical research and biological specimens in the world and has research conducted in the United States, Europe, Israel and Australia.(1) The data is available upon approval to academic and industry researchers.¶

2. Objectives

The motivation for this research project is to join in research done by industry experts to fight against these diseases. Neuronal Synuclein disease is currently incurable, and only barely beginning to be even definitively diagnosed, despite it affecting millions of people throughout the world each year.(2) Family members and friends have been diagnosed with these diseases, and it is the hope of this team to be able to be able to contribute to the care, if not of these specific people, future people who may be able to improve their life qualities and durations as a result of these studies.

The first objective of this research analysis is to analyze biomarkers within tests performed on cerebrospinal fluid to determine if neuronal synuclein disease can be diagnosed with a high likelihood. The second objective is to determine a numerical threshold for testing positive, assuming the first objective is achieved.

The dataset we are using contains numerical values related to the degree of presence or absence of alpha Synuclein Seed Assay (SAA) in cerebral spinal fluid. Our hypothesis is that there is a relationship between SAA and the types of these neurological disorders. We will look at two of the tests performed with the alpha synuclein protein on research subjects to

determine the correlation between those results and the incidence of the disease.

1. Fmax. The first test, the fluorescence value test (Fmax), tests for variances of the SAA protein.

2. TTT: The second test, Time to Threshold (TTT) tests for the time for a substance to reach a predefined threshold. We will look at the values of these tests together and separately to determine the relationships between the tests, the presence of SAA, and the incidence of a positive diagnosis.

3. Importing Libraries

You may need to install ipywidgets with a pip if you have not previously used this extension.

```
In [1]: pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in c:\users\carba\anaconda3\lib\site-packa
ges (8.1.5)
Requirement already satisfied: comm>=0.1.3 in c:\users\carba\anaconda3\lib\site-pack
ages (from ipywidgets) (0.2.1)
Requirement already satisfied: ipython>=6.1.0 in c:\users\carba\anaconda3\lib\site-p
ackages (from ipywidgets) (8.27.0)
Requirement already satisfied: traitlets>=4.3.1 in c:\users\carba\anaconda3\lib\site
-packages (from ipywidgets) (5.14.3)
Requirement already satisfied: widgetsnbextension~=4.0.12 in c:\users\carba\anaconda
3\lib\site-packages (from ipywidgets) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in c:\users\carba\anaconda
3\lib\site-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: decorator in c:\users\carba\anaconda3\lib\site-packag
es (from ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\carba\anaconda3\lib\site-packa
ges (from ipython>=6.1.0->ipywidgets) (0.19.1)
Requirement already satisfied: matplotlib-inline in c:\users\carba\anaconda3\lib\sit
e-packages (from ipython>=6.1.0->ipywidgets) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\carba\anaco
nda3\lib\site-packages (from ipython>=6.1.0->ipywidgets) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in c:\users\carba\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (2.15.1)
Requirement already satisfied: stack-data in c:\users\carba\anaconda3\lib\site-packa
ges (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: colorama in c:\users\carba\anaconda3\lib\site-package
s (from ipython>=6.1.0->ipywidgets) (0.4.6)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\carba\anaconda3\lib\s
ite-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: wcwidth in c:\users\carba\anaconda3\lib\site-packages
(from prompt-toolkit<3.1.0,>=3.0.41->ipython>=6.1.0->ipywidgets) (0.2.5)
Requirement already satisfied: executing in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: asttokens in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (0.2.2)
Requirement already satisfied: six in c:\users\carba\anaconda3\lib\site-packages (fr
om asttokens->stack-data->ipython>=6.1.0->ipywidgets) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install ipywidgets seaborn
```

```
Requirement already satisfied: ipywidgets in c:\users\carba\anaconda3\lib\site-packa
ges (8.1.5)
Requirement already satisfied: seaborn in c:\users\carba\anaconda3\lib\site-packages
(0.13.2)
Requirement already satisfied: comm>=0.1.3 in c:\users\carba\anaconda3\lib\site-pack
ages (from ipywidgets) (0.2.1)
Requirement already satisfied: ipython>=6.1.0 in c:\users\carba\anaconda3\lib\site-p
ackages (from ipywidgets) (8.27.0)
Requirement already satisfied: traitlets>=4.3.1 in c:\users\carba\anaconda3\lib\site
-packages (from ipywidgets) (5.14.3)
Requirement already satisfied: widgetsnbextension~=4.0.12 in c:\users\carba\anaconda
3\lib\site-packages (from ipywidgets) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in c:\users\carba\anaconda
3\lib\site-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\carba\anaconda3\lib
\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\carba\anaconda3\lib\site-pack
ages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\carba\anaconda3\l
ib\site-packages (from seaborn) (3.9.2)
Requirement already satisfied: decorator in c:\users\carba\anaconda3\lib\site-packag
es (from ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\carba\anaconda3\lib\site-packa
ges (from ipython>=6.1.0->ipywidgets) (0.19.1)
Requirement already satisfied: matplotlib-inline in c:\users\carba\anaconda3\lib\sit
e-packages (from ipython>=6.1.0->ipywidgets) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\carba\anaco
nda3\lib\site-packages (from ipython>=6.1.0->ipywidgets) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in c:\users\carba\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (2.15.1)
Requirement already satisfied: stack-data in c:\users\carba\anaconda3\lib\site-packa
ges (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: colorama in c:\users\carba\anaconda3\lib\site-package
s (from ipython>=6.1.0->ipywidgets) (0.4.6)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\carba\anaconda3\lib\site
-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\carba\anaconda3\lib\site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\carba\anaconda3\lib\sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\carba\anaconda3\lib\sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\carba\anaconda3\lib\site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\carba\anaconda3\lib\site-packag
es (from matplotlib!=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\carba\anaconda3\lib\site
-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\carba\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\carba\anaconda3\lib\site-pac
kages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\carba\anaconda3\lib\site-p
ackages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\carba\anaconda3\lib\s
ite-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
```

```
Requirement already satisfied: wcwidth in c:\users\carba\anaconda3\lib\site-packages
(from prompt-toolkit<3.1.0,>=3.0.41->ipython>=6.1.0->ipywidgets) (0.2.5)
Requirement already satisfied: six>=1.5 in c:\users\carba\anaconda3\lib\site-package
s (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Requirement already satisfied: executing in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: asttokens in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in c:\users\carba\anaconda3\lib\site-packag
es (from stack-data->ipython>=6.1.0->ipywidgets) (0.2.2)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\carba\anaconda3\lib\site-pac
kages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\carba\anaconda3\lib\site-pa
ckages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\carba\anaconda3\lib\site-pac
kages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\carba\anaconda3\lib\site-pa
ckages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\carba\anaconda3\lib
\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import seaborn as sns
import scipy.stats as stats
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score,roc_curve, auc, precision_recall_curve,
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.calibration import calibration_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
import statsmodels.api as sm
import ipywidgets as widgets
```

To view this SSA analysis, download the SAA_Biospecimen_Analysis_Results.csv and upload it to the directory of your notebook. The directory can be found using the os code in the cell below.

In [5]:
```python
import os
os.getcwd()
```

Out[5]:
```
'C:\\Users\\carba\\BUAN_660_Statistics'
```

4. Loading Data

In [6]:
```python
# reads file from local folder
# Load a dataset into a Pandas DataFrame
SAA_Biospecimen = pd.read_csv("SAA_Biospecimen_Analysis_Results.csv")
```

5. Data Exploration

In [7]:
```python
# method relays information about the SAA Biospecimen dataframe including total col

SAA_Biospecimen.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1586 entries, 0 to 1585
Data columns (total 48 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   PATNO              1586 non-null   int64  
 1   SEX                1586 non-null   object 
 2   COHORT             1586 non-null   object 
 3   CLINICAL_EVENT     1586 non-null   object 
 4   TYPE               1586 non-null   object 
 5   SAAMethod          1586 non-null   object 
 6   SAA_Status         1586 non-null   object 
 7   SAA_Type           756 non-null    object 
 8   Fmax_24h_Rep1      931 non-null    float64
 9   Fmax_24h_Rep2      932 non-null    float64
 10  Fmax_24h_Rep3      932 non-null    float64
 11  TTT_24h_Rep1       773 non-null    float64
 12  TTT_24h_Rep2       762 non-null    float64
 13  TTT_24h_Rep3       766 non-null    float64
 14  AUC_24h_Rep1       931 non-null    float64
 15  AUC_24h_Rep2       932 non-null    float64
 16  AUC_24h_Rep3       932 non-null    float64
 17  TSmax_24h_Rep1     931 non-null    float64
 18  TSmax_24h_Rep2     932 non-null    float64
 19  TSmax_24h_Rep3     932 non-null    float64
 20  SLOPEMax_24h_Rep1  931 non-null    float64
 21  SLOPEMax_24h_Rep2  932 non-null    float64
 22  SLOPEMax_24h_Rep3  932 non-null    float64
 23  Fmax_150h_Rep1     654 non-null    float64
 24  Fmax_150h_Rep2     654 non-null    float64
 25  Fmax_150h_Rep3     654 non-null    float64
 26  TTT_150h_Rep1      531 non-null    float64
 27  TTT_150h_Rep2      538 non-null    float64
 28  TTT_150h_Rep3      546 non-null    float64
 29  AUC_150h_Rep1      531 non-null    float64
 30  AUC_150h_Rep2      538 non-null    float64
 31  AUC_150h_Rep3      546 non-null    float64
 32  T50_150h_Rep1      528 non-null    float64
 33  T50_150h_Rep2      538 non-null    float64
 34  T50_150h_Rep3      545 non-null    float64
 35  SLOPE_150h_Rep1    531 non-null    float64
 36  SLOPE_150h_Rep2    538 non-null    float64
 37  SLOPE_150h_Rep3    546 non-null    float64
 38  InstrumentRep1     1586 non-null   int64  
 39  InstrumentRep2     1586 non-null   int64  
 40  InstrumentRep3     1586 non-null   int64  
 41  SampleVolRep1      8 non-null      object 
 42  SampleVolRep2      1 non-null      object 
 43  SampleVolRep3      118 non-null    object 
 44  RUNDATE            1586 non-null   object 
 45  PROJECTID          1586 non-null   int64  
 46  PI_NAME            1586 non-null   object 
 47  PI_INSTITUTION     1586 non-null   object 
dtypes: float64(30), int64(5), object(13)
memory usage: 594.9+ KB
```

In [8]:
```
SAA_Biospecimen.head()
```

Out[8]:

	PATNO	SEX	COHORT	CLINICAL_EVENT	TYPE	SAAMethod	SAA_Status	SA
0	245573	Male	PD	BL	Cerebrospinal Fluid	Amprion-24h alpha-synuclein-SAA	Positive	
1	241189	Female	PD	BL	Cerebrospinal Fluid	Amprion-24h alpha-synuclein-SAA	Negative	
2	163324	Male	PD	BL	Cerebrospinal Fluid	Amprion-24h alpha-synuclein-SAA	Positive	
3	250240	Male	PD	BL	Cerebrospinal Fluid	Amprion-24h alpha-synuclein-SAA	Positive	
4	164985	Male	PD	BL	Cerebrospinal Fluid	Amprion-24h alpha-synuclein-SAA	Positive	

5 rows × 48 columns

We learn from the code above that there are no missing results for the index PATNO nor the SAA_Status which indicates whether or not the patient has NSD

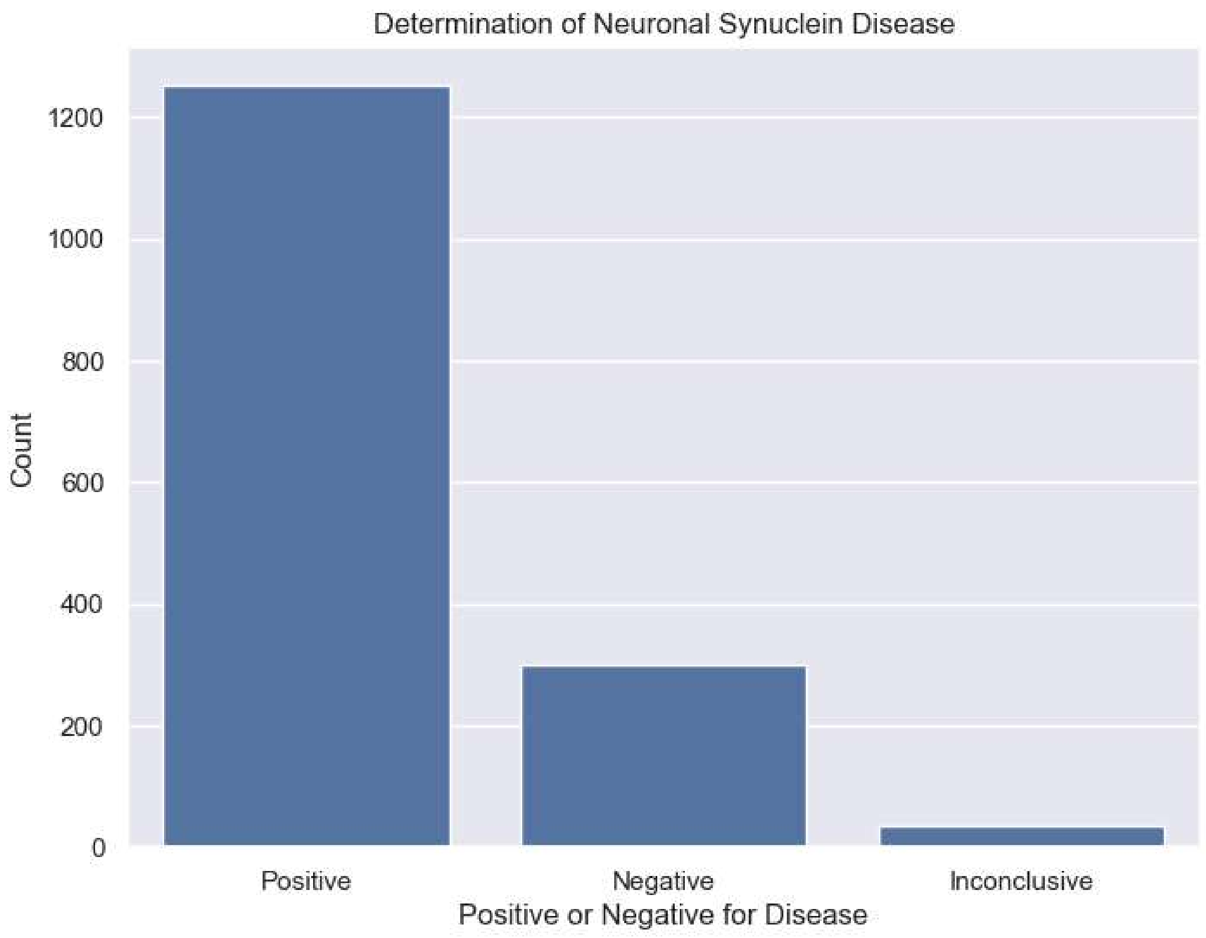
Next we will look at the different values within SAA_Status

In [9]:
```
distinct_values = SAA_Biospecimen['SAA_Status'].unique()

# Display the distinct values
print(distinct_values)

distinct_count = SAA_Biospecimen['SAA_Status'].nunique()
print(f"Number of distinct values: {distinct_count}")

value_counts = SAA_Biospecimen['SAA_Status'].value_counts()

# Display the counts for each distinct value in 'SAA_Status'
print(value_counts)
```

```
['Positive' 'Negative' 'Inconclusive']
Number of distinct values: 3
SAA_Status
Positive        1252
Negative         300
Inconclusive      34
Name: count, dtype: int64
```

In [10]:

```
# Check the distribution of the target variable (SAA_Status)
plt.figure(figsize=(8, 6))
sns.set(style='darkgrid')
sns.countplot(x='SAA_Status', data=SAA_Biospecimen)
plt.title('Determination of Neuronal Synuclein Disease')
plt.xlabel('Positive or Negative for Disease')
plt.ylabel('Count')
plt.show()

# Check the Cancellation Rate
## Count total bookings
total_participants = len(SAA_Biospecimen)

# tpt on SAA_Type (including all rows)
total_participants = len(SAA_Biospecimen)

# Count the number of 'Positive' in the SAA_Status column (this is the cancellation
number_of_positives = SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] == 'Positive'].

# Calculate the cancellation rate as the percentage of 'Positive' statuses
positive_rate = (number_of_positives / total_participants) * 100

# Output the results
print(f'Total Study Participants: {total_participants}')
print(f'Number Participants of Positive for Neuronal Synuclein Disease: {number_of_
print(f'Positive Rate: {positive_rate:.2f}%')
```



```
Total Study Participants: 1586
Number Participants of Positive for Neuronal Synuclein Disease: 1252
Positive Rate: 78.94%
```

Because the distribution is so skewed, it needs to be adjusted since imbalances can make it difficult to correctly make predictions. AS a result, we want to balance the distribution.

We can balance the distribution by three ways: Under-sampling, over-sampling, or SMOTE. Under-sampling can leave out important data if the sample size ends up being too small, Over-sampling increases the data size which would be acceptable in this sitation, but it can also overfit the model because data is duplicated. SMOTE reduces overfitting and balances classes better. SMOTE also uses more computation, but in this sitation, those additional resources are not significant.

We will use SMOTE in a further step when we are ready to perform the Logistic Regression.

We want to link variables that predict either a positive or negative result for Neuronal Synuclein Disease. As a result, we will drop value "Inconclusive" since that will not help our analysis and may make the results less clear

In [11]:
```
SAA_Biospecimen = SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] != 'Inconclusive']

# Assuming df is your DataFrame
distinct_values = SAA_Biospecimen['SAA_Status'].unique()
```

```
# Display the distinct values
print(distinct_values)

distinct_count = SAA_Biospecimen['SAA_Status'].nunique()
print(f"Number of distinct values: {distinct_count}")

value_counts = SAA_Biospecimen['SAA_Status'].value_counts()

# Display the counts for each distinct value in 'SAA_Status'
print(value_counts)
```

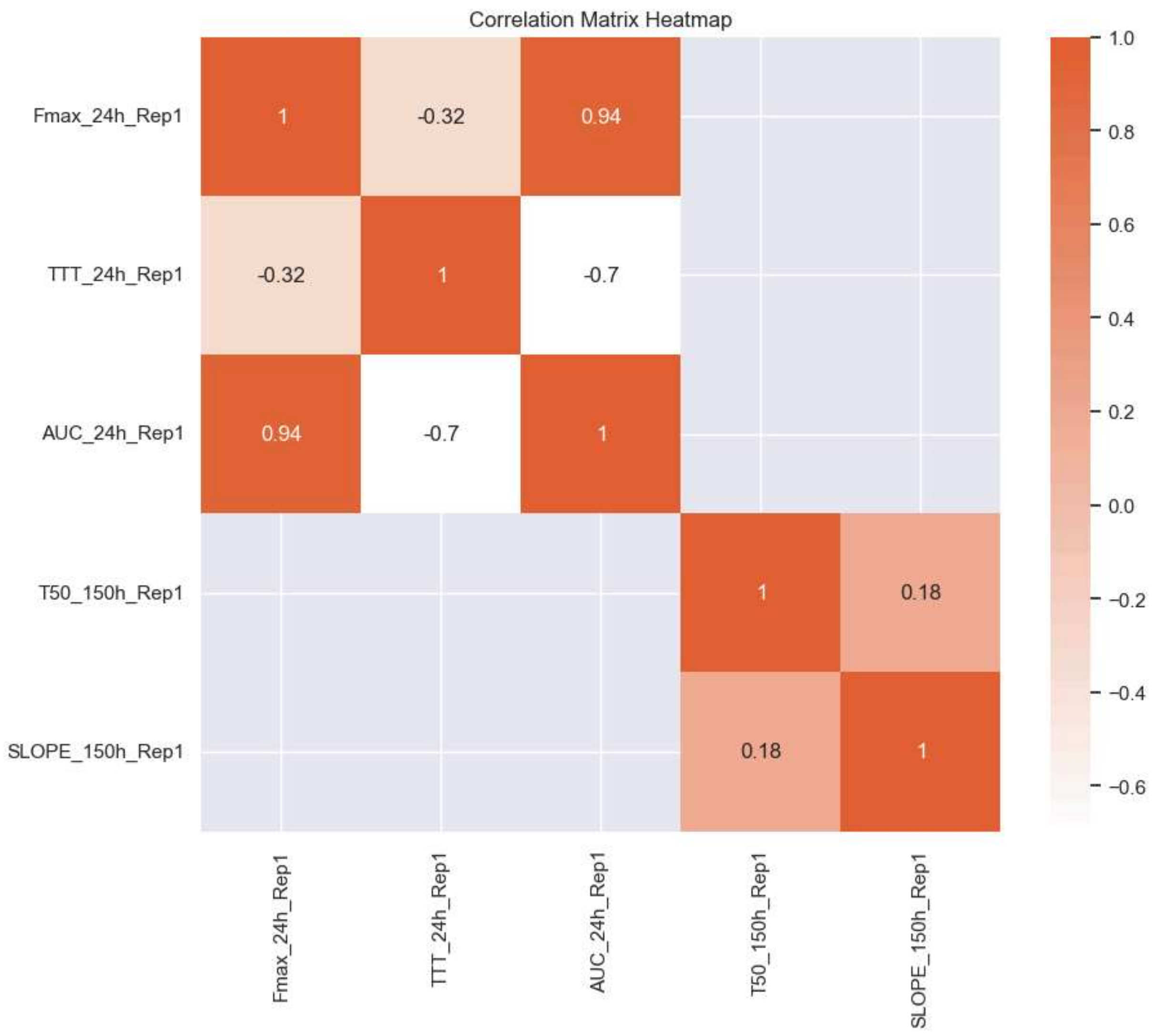
```
['Positive' 'Negative']
Number of distinct values: 2
SAA_Status
Positive    1252
Negative     300
Name: count, dtype: int64
```

6. Reviewing Correlation

All of the columns that begin with Fmax, TTT, AUC, T50, SLOPE have multiple tests that are the same except with different time values. Additionally, because off of these tests are looking at various parts of the protein, alpha-synuclein, many of them are highly correlated. We will create a correlation matrix to compare the correlation of each type of test.

In [12]:
```
# Select the columns for the correlation matrix
columns = ['Fmax_24h_Rep1', 'TTT_24h_Rep1', 'AUC_24h_Rep1',
           'T50_150h_Rep1', 'SLOPE_150h_Rep1']

# Create the correlation matrix for the selected columns
correlation_matrix = SAA_Biospecimen[columns].corr()

# Set the figure size
plt.figure(figsize=(10, 8))

# Create a colormap that transitions from white to #E16032
custom_colors = ['#FFFFFF', '#E16032']
cmap = LinearSegmentedColormap.from_list('custom_cmap', custom_colors, N=100)

# Create the heatmap
sns.heatmap(correlation_matrix, cmap=cmap, annot=True)

# Set the title of the plot
plt.title('Correlation Matrix Heatmap')
```

Out[12]:
```
Text(0.5, 1.0, 'Correlation Matrix Heatmap')
```



As we expected, we can see that there is correlation between these tests. Also, we can see that the first three tests, Fmax_24h_Rep1, TTT_24h_Rep1, AUC_24h_Rep1 have no rows that overlap with T50_150h_Rep1, SLOPE_150h_Rep1. We will pick just two with a moderate amount of correlation as well as data overlap to continue the analysis and drop the remaining columns.

Additionally, we will drop all categorical values not useful to this analysis.

Using values with high correlation can result in multicollinearity, and overfitting which can lead to incorrect results and make conclusions difficult or at worst, erroneous.

7. Data Preprocessing

We will only want to keep a few of the most related variables. Accordingly, we will use just a slice of the data for the prediction algorithms.

In [13]:
```
# Creating a new dataframe

# List of columns we want to keep
selected_columns = [
'PATNO',
'SAA_Status',
'Fmax_24h_Rep1',
'TTT_24h_Rep1']

# Create a new DataFrame by selecting only the columns you need
Select_Biospecimen = SAA_Biospecimen[selected_columns]

# Check the new dataset
print(Select_Biospecimen.head())
```

```
    PATNO SAA_Status  Fmax_24h_Rep1  TTT_24h_Rep1
0  245573   Positive       153626.0         10.70
1  241189   Negative          600.0           NaN
2  163324   Positive        75742.0         18.07
3  250240   Positive       110430.0         15.39
4  164985   Positive       135020.0         11.74
```

In [14]:
```
# Remove rows where 'Fmax_24h_Rep1','TTT_24h_Rep1' and 'SAA_Status' have NaN values

Ultra_Select_Bio = Select_Biospecimen.dropna(subset=['SAA_Status','Fmax_24h_Rep1',

# Check the resulting DataFrame
print(Ultra_Select_Bio.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 766 entries, 0 to 1585
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   PATNO          766 non-null    int64  
 1   SAA_Status     766 non-null    object 
 2   Fmax_24h_Rep1  766 non-null    float64
 3   TTT_24h_Rep1   766 non-null    float64
dtypes: float64(2), int64(1), object(1)
memory usage: 29.9+ KB
None
```

In [15]:
```
# Checking on the number of values remaining in the dataframe after cleaning

# Assuming df is your DataFrame
distinct_values = Ultra_Select_Bio['SAA_Status'].unique()

# Display the distinct values
print(distinct_values)

distinct_count = Ultra_Select_Bio['SAA_Status'].nunique()
print(f"Number of distinct values: {distinct_count}")

value_counts = Ultra_Select_Bio['SAA_Status'].value_counts()
```

```
# Display the counts for each distinct value in 'SAA_Status'
print(value_counts)
```

```
['Positive' 'Negative']
Number of distinct values: 2
SAA_Status
Positive    745
Negative     21
Name: count, dtype: int64
```

In [16]:
```
Log_Reg_SAA = Ultra_Select_Bio

# Check the resulting DataFrame
print(Log_Reg_SAA.info())
```

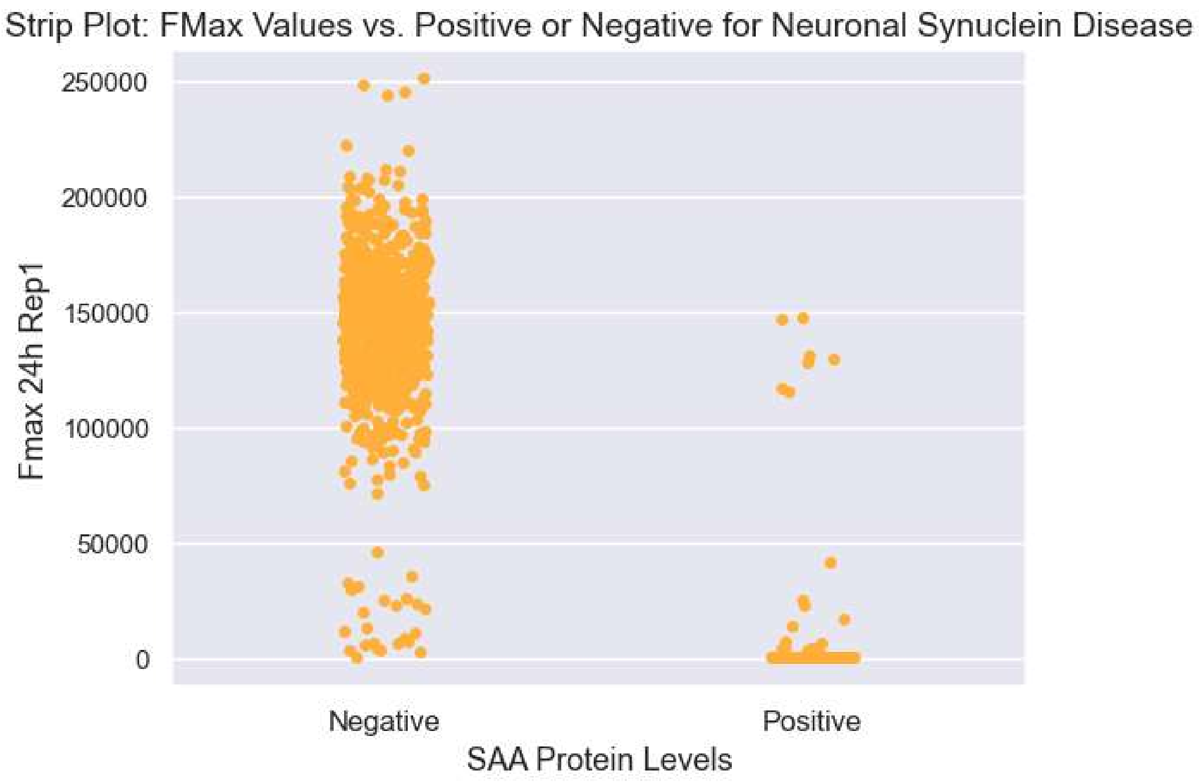
```
<class 'pandas.core.frame.DataFrame'>
Index: 766 entries, 0 to 1585
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   PATNO          766 non-null    int64  
 1   SAA_Status     766 non-null    object 
 2   Fmax_24h_Rep1  766 non-null    float64
 3   TTT_24h_Rep1   766 non-null    float64
dtypes: float64(2), int64(1), object(1)
memory usage: 29.9+ KB
None
```

In [ ]:

8. Reviewing Variables Used for Prediction

Distribution of Maximum Fluorescence in RFU after 24 hours (FMax 24h) Variable

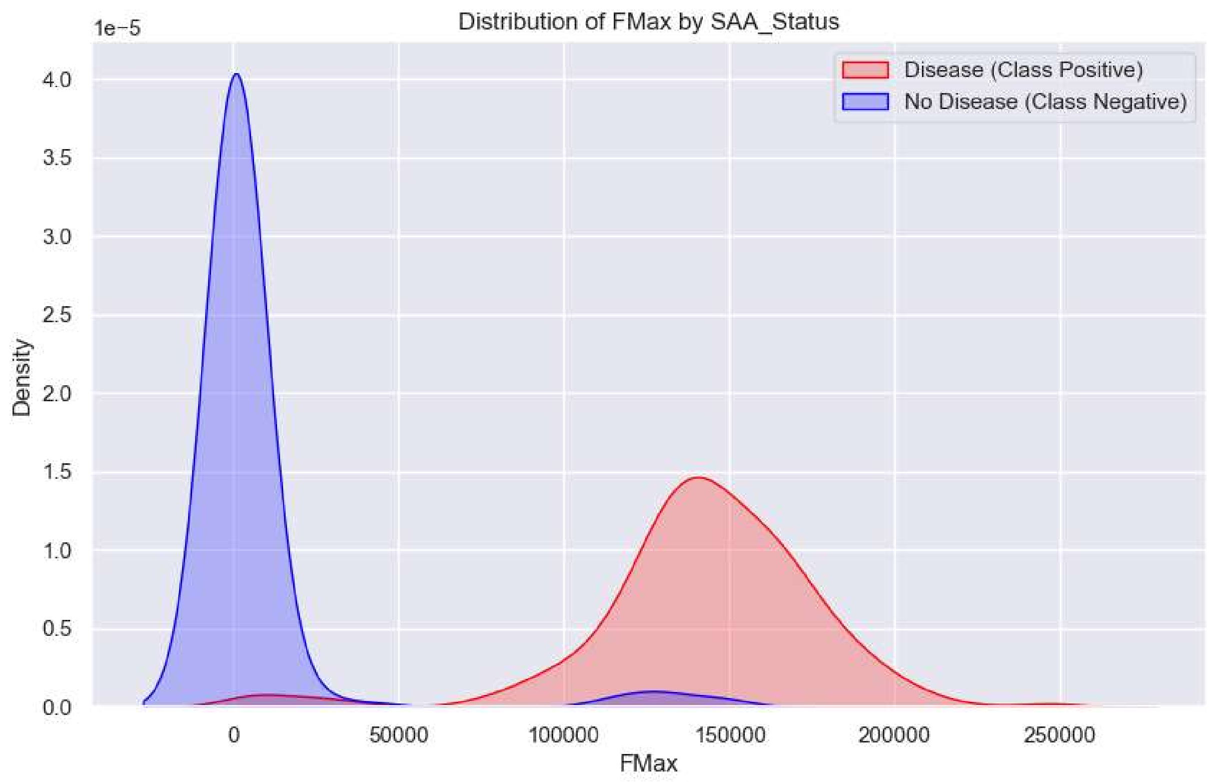
In [17]:
```
# Create a strip plot with filtered data
sns.stripplot(x='SAA_Status', y='Fmax_24h_Rep1', data=SAA_Biospecimen.dropna(subset

# Add labels and title
plt.xlabel('SAA Protein Levels', fontsize=13)
plt.ylabel('Fmax 24h Rep1', fontsize=13)
plt.title('Strip Plot: FMax Values vs. Positive or Negative for Neuronal Synuclein

# Set custom labels for the x-axis
plt.xticks(ticks=[0, 1], labels=['Negative', 'Positive'], fontsize=12)

plt.show()
```



Kernel Density Estimate plot and summary statistics for FMax 24h as tested on SSA Protein

In [18]:
```
# Convert 'SAA_Status' to numeric: 'Positive' -> 1, 'Negative' -> 0
SAA_Biospecimen['SAA_Status'] = SAA_Biospecimen['SAA_Status'].map({'Positive': 1, '
```

In [19]:
```
plt.figure(figsize=(10, 6))
sns.kdeplot(SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] == 1]['Fmax_24h_Rep1'], l
sns.kdeplot(SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] == 0]['Fmax_24h_Rep1'], l
plt.title('Distribution of FMax by SAA_Status')
plt.xlabel('FMax')
plt.ylabel('Density')
plt.legend()
plt.show()
```

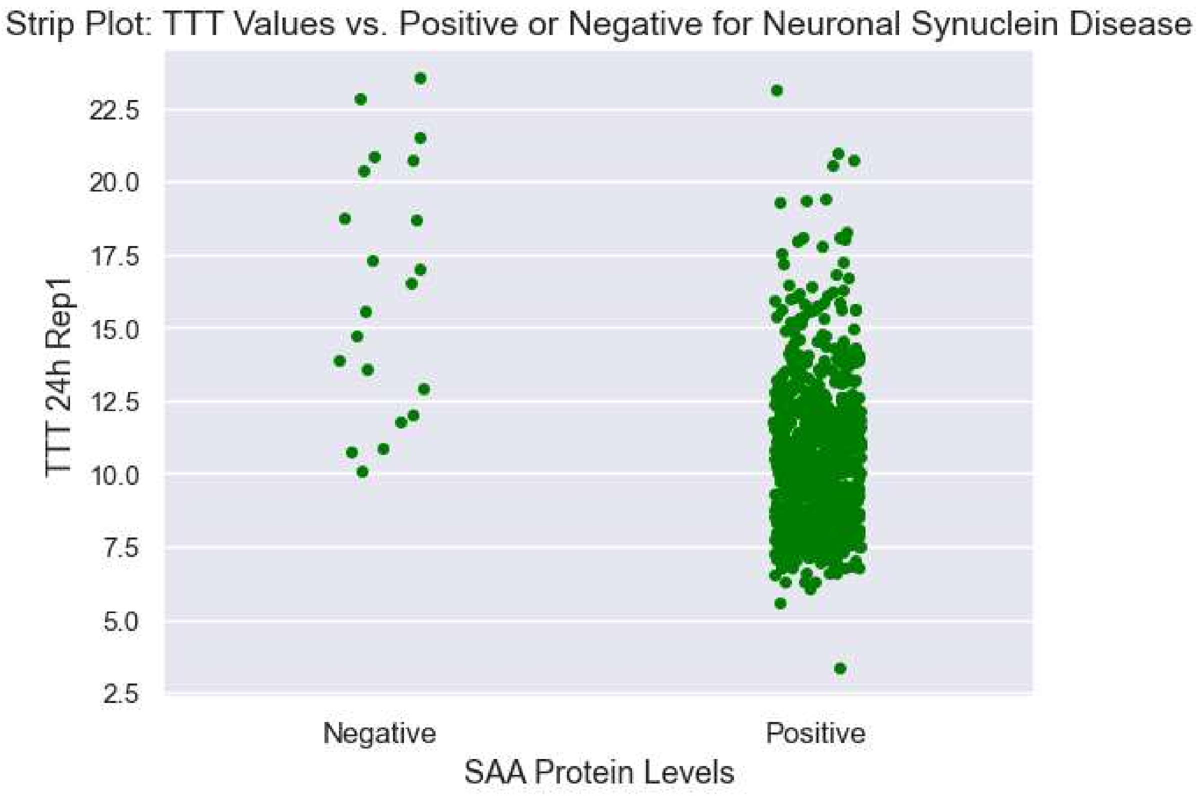


In [20]:
```
Fmax_24h_Rep1 = SAA_Biospecimen['Fmax_24h_Rep1'].dropna().describe()
print(Fmax_24h_Rep1)
```

```
count       923.000000
mean     116520.590466
std       62984.315363
min         445.000000
25%      101162.500000
50%      137018.000000
75%      158904.500000
max      250877.000000
Name: Fmax_24h_Rep1, dtype: float64
```

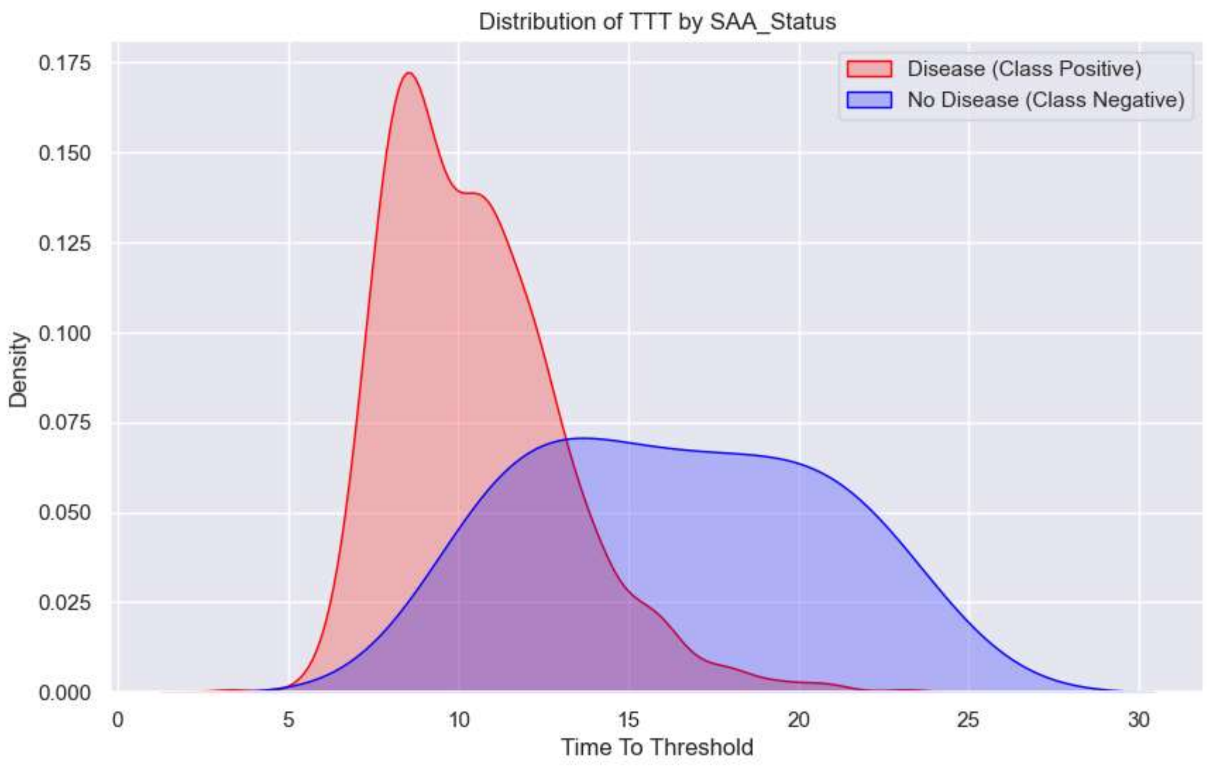
Distribution of Time To Threshold in 24 hours (TTT 24h) Variable

In [21]:
```
# Create a strip plot with filtered data
sns.stripplot(x='SAA_Status', y='TTT_24h_Rep1', data=SAA_Biospecimen.dropna(subset=

# Add labels and title
plt.xlabel('SAA Protein Levels', fontsize=13)
plt.ylabel('TTT 24h Rep1', fontsize=13)
plt.title('Strip Plot: TTT Values vs. Positive or Negative for Neuronal Synuclein D

# Set custom labels for the x-axis
plt.xticks(ticks=[0, 1], labels=['Negative', 'Positive'], fontsize=12)

plt.show()
```



Kernel Density Estimate plot and summary statistics for TTT 24h as tested on SSA Protein

In [22]:
```
plt.figure(figsize=(10, 6))
sns.kdeplot(SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] == 1]['TTT_24h_Rep1'], la
sns.kdeplot(SAA_Biospecimen[SAA_Biospecimen['SAA_Status'] == 0]['TTT_24h_Rep1'], la
plt.title('Distribution of TTT by SAA_Status')
plt.xlabel('Time To Threshold')
plt.ylabel('Density')
plt.legend()
plt.show()
```



In [23]:
```
# Cell 14: Summary Statistics for Time To Threshold in 24 hours (TTT 24h)
# Calculate and display summary statistics for TTT_24h_Rep1
TTT_24h_Rep1_stats = SAA_Biospecimen['TTT_24h_Rep1'].dropna().describe()
print(TTT_24h_Rep1_stats)
```

```
count    766.000000
mean      10.641606
std        2.819982
min        3.330000
25%        8.520000
50%       10.270000
75%       12.082500
max       23.540000
Name: TTT_24h_Rep1, dtype: float64
```

9. Logistic Regression Model

In [24]:
```
# Selecting features and target variable
X = Log_Reg_SAA.drop(columns=['SAA_Status'])
y = Log_Reg_SAA['SAA_Status']

# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, ran
# Use SMOTE to generate synthetic samples for the minority class in the training se
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Train your predictive model (e.g., Logistic Regression) on the SMOTE-balanced dat
model_name = "Logistic Regression"
model = LogisticRegression()
model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probability estimates for the p

# Classification Report
print(f"{model_name} Classification Report:")
print(classification_report(y_test, y_pred, digits=3))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

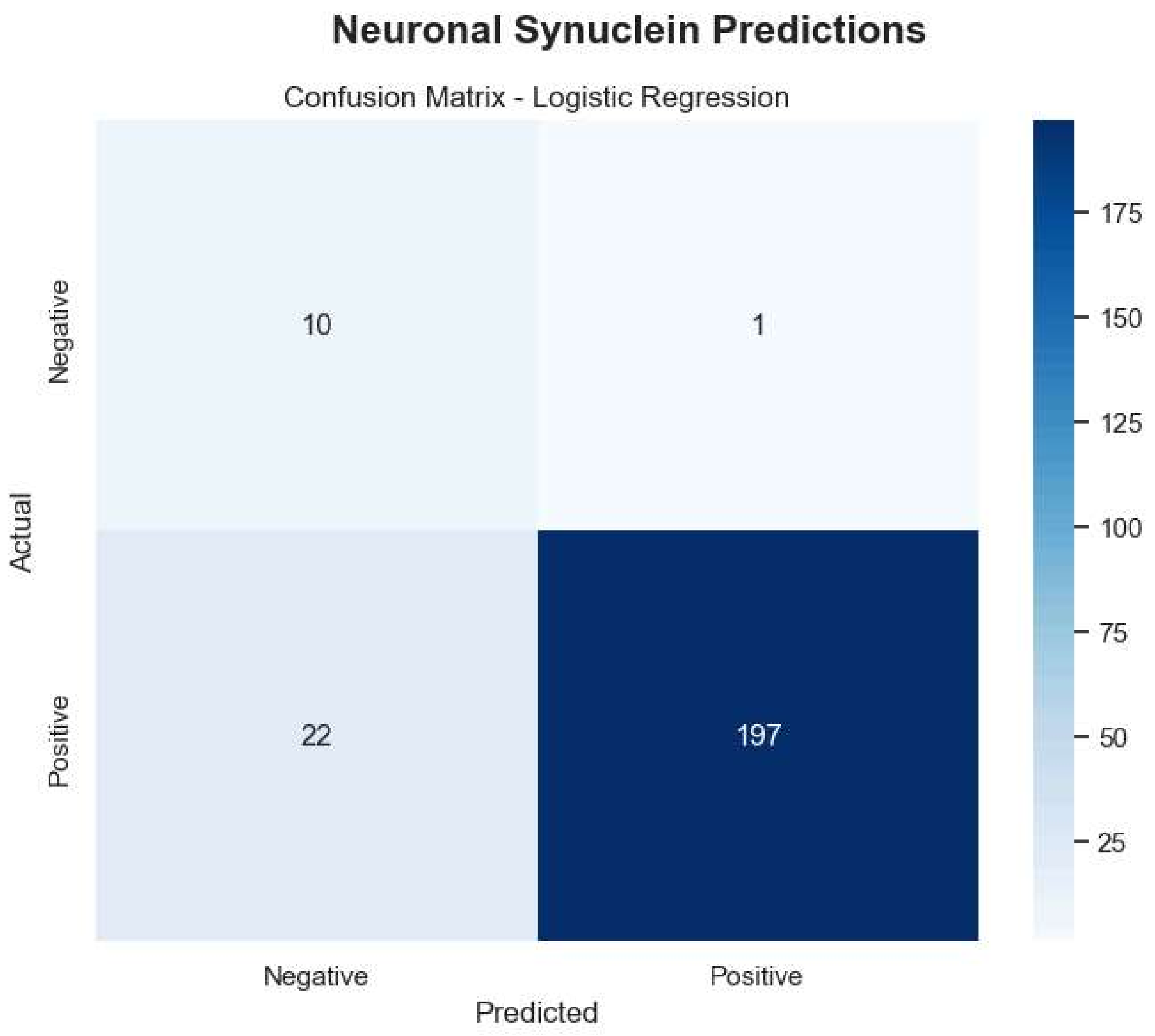
```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

    Negative      0.312     0.909     0.465        11
    Positive      0.995     0.900     0.945       219

    accuracy                          0.900       230
   macro avg      0.654     0.904     0.705       230
weighted avg      0.962     0.900     0.922       230

Confusion Matrix:
[[ 10   1]
 [ 22 197]]
```

In [25]:
```
# Convert y_test to numeric
y_test_numeric = y_test.map({'Positive': 1, 'Negative': 0})

# Calculate Precision, Recall, ROC-AUC, Precision-Recall AUC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba)
roc_auc = auc(fpr, tpr)
precision, recall, _ = precision_recall_curve(y_test_numeric, y_pred_proba)
average_precision = average_precision_score(y_test_numeric, y_pred_proba)

# Display the results
print(f"ROC-AUC: {roc_auc:.3f}")
print(f"Average Precision (AP): {average_precision:.3f}")
```

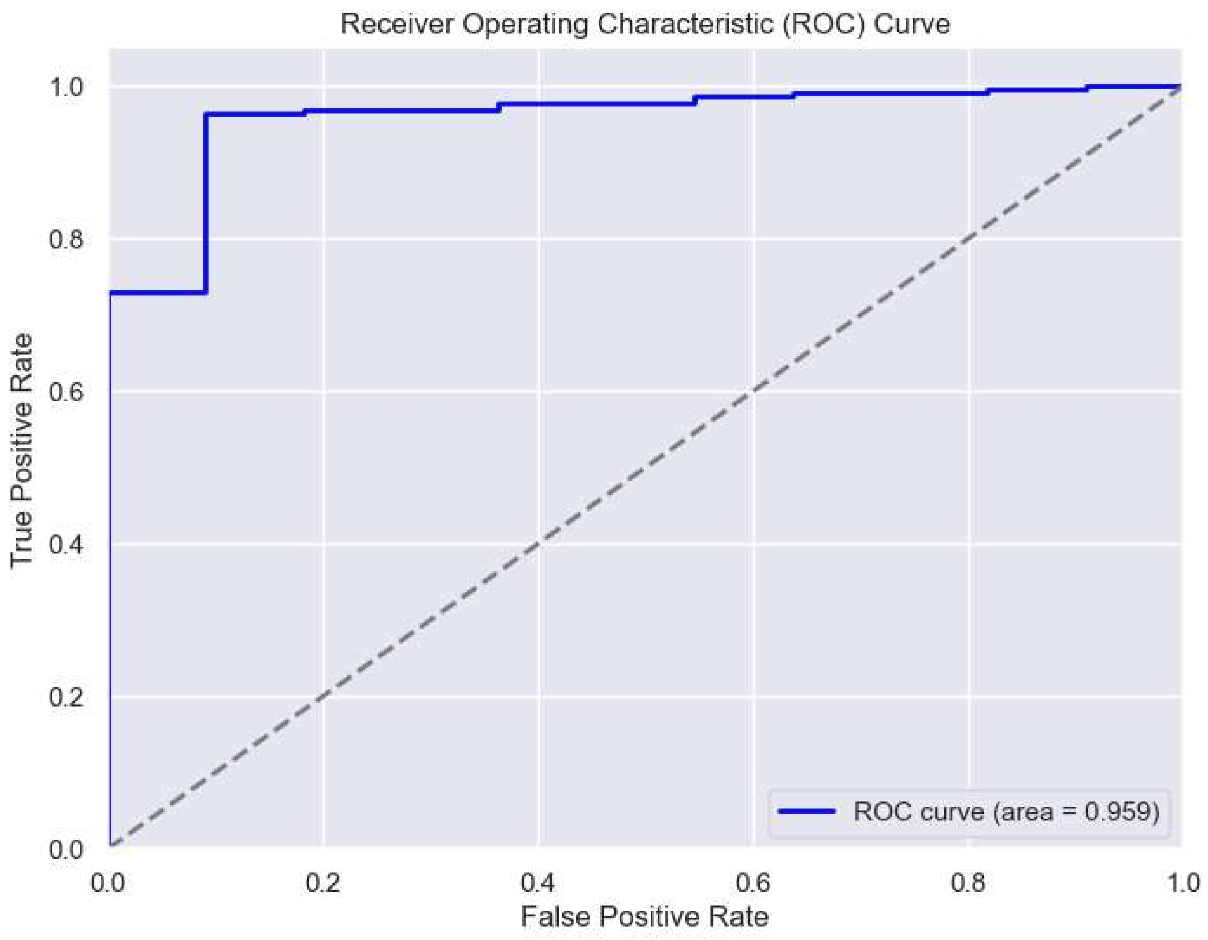
```
ROC-AUC: 0.959
Average Precision (AP): 0.998
```

In [26]:
```
# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative'
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
# Add the main title and the subtitle
plt.suptitle('Neuronal Synuclein Predictions', fontsize=16, fontweight='bold')
plt.title('Confusion Matrix - Logistic Regression', fontsize=12)
plt.show()
```

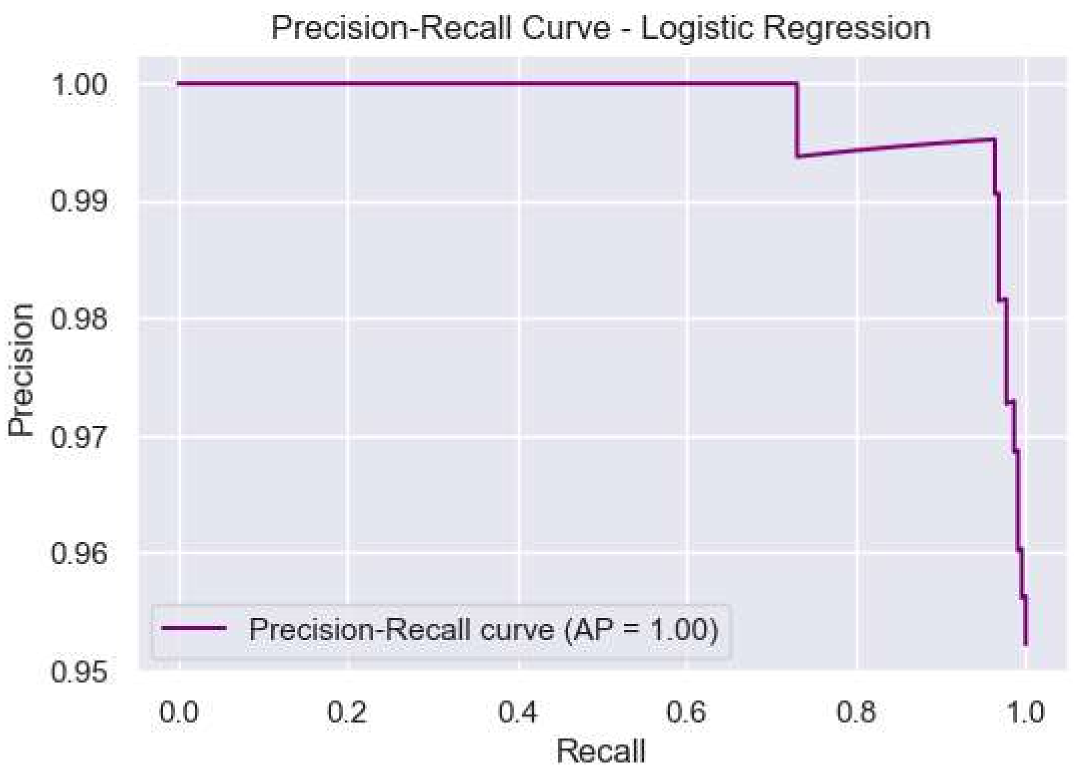


In [27]:
```
# Plotting the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Display the results
print(f"{model_name} Classification Report:")
print(f"ROC-AUC: {roc_auc:.3f}")
print(f"Average Precision (AP): {average_precision:.3f}")
```



```
Logistic Regression Classification Report:
ROC-AUC: 0.959
Average Precision (AP): 0.998
```

In [28]:
```
# Plotting Precision-Recall Curve
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='purple', label=f'Precision-Recall curve (AP = {a
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve - Logistic Regression")
plt.legend(loc="lower left")
plt.show()

# Displaying evaluation metrics
model_performance = {
    "Confusion Matrix": conf_matrix,
    "ROC AUC": roc_auc,
    "Average Precision (AP)": average_precision
}

print(model_performance)
```



```
{'Confusion Matrix': array([[ 10,   1],
       [ 22, 197]], dtype=int64), 'ROC AUC': 0.958904109589041, 'Average Precision
(AP)': 0.9977657268650819}
```

10. Interpretation of the Results of Logistic Regression

Key Metrics:

Precision:

Class 0 (Negative): 0.312 indicates that only 31.2% of the model's predictions for "negative" cases are correct.

Class 1 (Positive): 0.995 means that when the model predicts someone is positive for neuronal synuclein disease, it is correct 99.5% of the time.

Recall:

Class 0 (Negative): 0.909 indicates the model captures 90.9% of true negatives.

Class 1 (Positive): 0.900 means the model correctly identifies 90% of those who actually have the disease.

F1-Score:

Combines precision and recall: Class 0 (Negative): 0.465 indicates weaker performance for

detecting true negatives.
Class 1 (Positive): 0.945 suggests strong performance for detecting true positives.

Accuracy:
Overall, the model correctly classifies 90% of cases.
ROC-AUC (0.959):
This high value indicates excellent discrimination between positive and negative cases.
verage Precision (0.998):
This is the area under the precision-recall curve. A value of 0.591 suggests moderate precision across recall levels, particularly for the minority class (Class 2).

Confusion Matrix:
True Negatives (TN): 10 – Correctly predicted negative cases (Class 0).
False Negatives (FN): 1 – Predicted negative (Class 0) but actually positive (Class 1).
True Positives (TP): 197 – Correctly predicted positive cases (Class 1).
False Positives (FP): 22 – Predicted positive (Class 1) but actually negative (Class 0).
Strengths:
The model excels at identifying positive cases (Class 1), with high precision and recall for this class. Overall accuracy and ROC-AUC are very high, indicating the model is generally effective.

Weaknesses:
The model struggles with the minority class (Class 0, negative cases), as seen in its low precision (31.2%) and F1-score (0.465). This issue arises because the dataset is imbalanced (Class 1 has 219 cases vs. 11 for Class 2). SMOTE helped improve recall for Class 2 (0.909), but precision remains low.

Conclusion:

The logistic regression model is highly effective at identifying positive cases, with a very high precision and recall for the positive class. However, the precision for the negative class is quite low, indicating that when the model predicts a negative result, it is often incorrect. The recall for the negative class is high, meaning it correctly identifies most of the actual negatives.

Overall, the model has high accuracy and performs well on average, but the imbalance in precision between the classes is something to be aware of. This could be due to the imbalance in the number of positive and negative cases.

11. Evaluation of Fmax Threshold to Determine Disease Positivity

As the levels of alpha-synuclein measured increase, the chances of being positive for the disease also increase.

Here we determine the lowest threshold value likely to determine a positive test result.

In [29]:
```
# List of percentiles to evaluate
percentiles = np.arange(0, 1.01, 0.01)  # From 0% to 100% in 1% increments

# Initialize variables to track the best threshold and performance
best_percentile = None
best_threshold = None
best_auc = 0

# Create a copy of the DataFrame to avoid modifying the original
df_copy = Ultra_Select_Bio.copy()

# Convert SAA_Status to numeric
df_copy['SAA_Status'] = df_copy['SAA_Status'].map({'Positive': 1, 'Negative': 0})

# Loop through percentiles
for percentile in percentiles:
    threshold = df_copy['Fmax_24h_Rep1'].quantile(percentile)

    # Use .loc to set Predicted_SAA_Status
    df_copy.loc[:, 'Predicted_SAA_Status'] = (df_copy['Fmax_24h_Rep1'] > threshold)

    # Calculate AUC and performance metrics
    auc_score = roc_auc_score(df_copy['SAA_Status'], df_copy['Predicted_SAA_Status'

    # Update the best threshold if performance improves
    if auc_score > best_auc:
        best_auc = auc_score
        best_percentile = percentile
        best_threshold = threshold

# Print the results
print(f"Best Percentile: {best_percentile * 100:.2f}%")
print(f"Best Threshold (FMax): {best_threshold:.3f}")
print(f"Best ROC-AUC: {best_auc:.3f}")

# Check classification report for the best threshold
df_copy.loc[:, 'Predicted_SAA_Status'] = (df_copy['Fmax_24h_Rep1'] > best_threshold
print("\nClassification Report:")
print(classification_report(df_copy['SAA_Status'], df_copy['Predicted_SAA_Status'],
Best Percentile: 5.00%
Best Threshold (FMax): 76181.750
Best ROC-AUC: 0.817

Classification Report:
              precision    recall  f1-score   support

           0      0.359     0.667     0.467        21
           1      0.990     0.966     0.978       745

    accuracy                          0.958       766
   macro avg      0.675     0.817     0.722       766
weighted avg      0.973     0.958     0.964       766
```

In [35]:
```
from ipywidgets import interact, FloatSlider

# Reuse the same DataFrame and best_threshold from the first block
def plot_kde(threshold):
    plt.figure(figsize=(10, 6))
    sns.kdeplot(df_copy['Fmax_24h_Rep1'][df_copy['SAA_Status'] == 0], fill=True, la
    sns.kdeplot(df_copy['Fmax_24h_Rep1'][df_copy['SAA_Status'] == 1], fill=True, la
    plt.axvline(threshold, color='r', linestyle='--', label=f'Threshold: {threshold

    # Calculate probabilities
    positive_probability = (df_copy['Fmax_24h_Rep1'] > threshold).mean() * 100
    negative_probability = (df_copy['Fmax_24h_Rep1'] <= threshold).mean() * 100

    plt.title(f'KDE Plot of Fmax_24h_Rep1 by SAA Status\nProbability of Positive Re
    plt.xlabel('Fmax_24h_Rep1')
    plt.ylabel('Density')
    plt.legend()
    plt.show()

# Create the interactive widget with best_threshold as the initial value
interact(plot_kde, threshold=FloatSlider(value=best_threshold, min=df_copy['Fmax_24
```

```
interactive(children=(FloatSlider(value=3124.0, description='threshold', max=250877.
0, min=3124.0), Output()),…
```

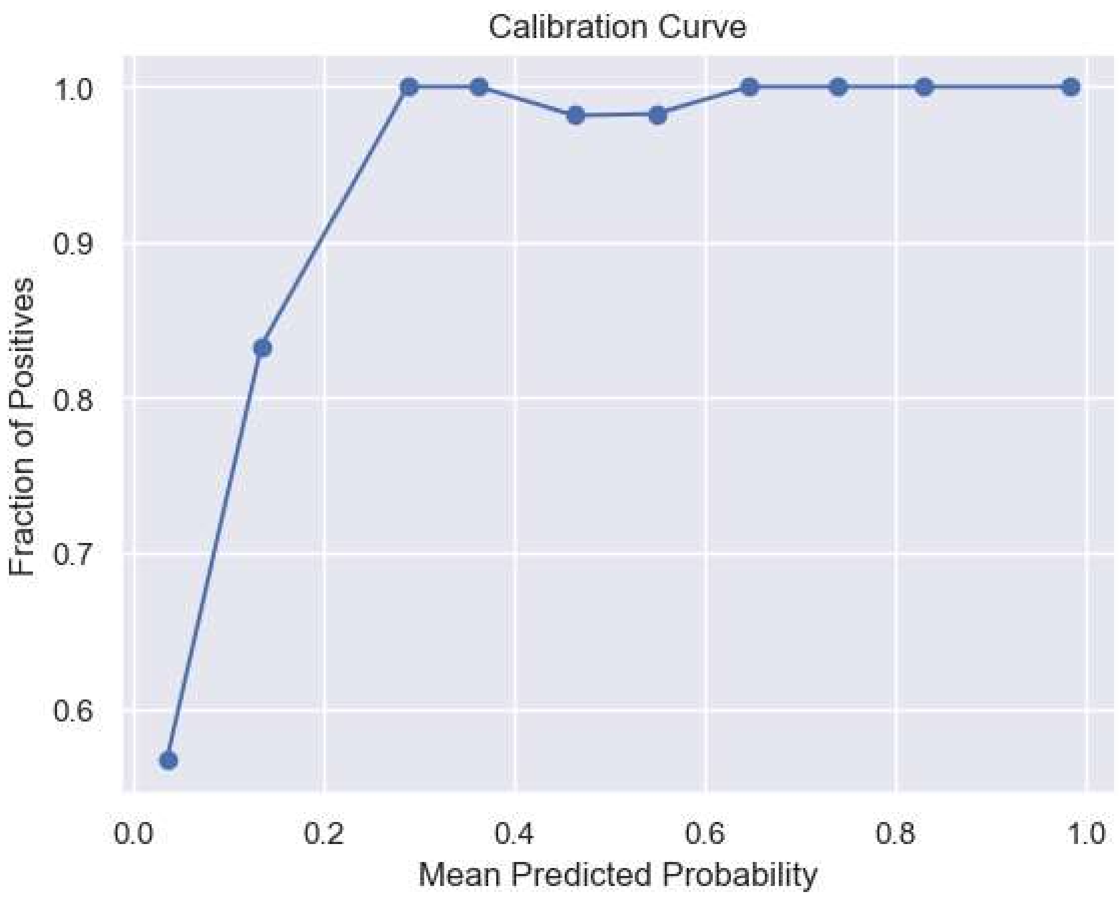
Out[35]:
```
<function __main__.plot_kde(threshold)>
```

In [31]:
```
# Displaying evaluation metrics
model_performance = {
    "Confusion Matrix": conf_matrix,
    "ROC AUC": round(roc_auc, 3),
    "Average Precision (AP)": round(average_precision, 3)
}

print(model_performance)
```

```
{'Confusion Matrix': array([[ 10,   1],
       [ 22, 197]], dtype=int64), 'ROC AUC': 0.959, 'Average Precision (AP)': 0.998}
```

In [32]:
```
# Calibration curve
scaler = MinMaxScaler()
df_copy['Fmax_24h_Rep1_scaled'] = scaler.fit_transform(df_copy[['Fmax_24h_Rep1']])
```

```
prob_true, prob_pred = calibration_curve(df_copy['SAA_Status'], df_copy['Fmax_24h_R

# Plot the calibration curve
plt.plot(prob_pred, prob_true, marker='o')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Curve')
plt.show()

# Displaying evaluation metrics
model_performance = {
    "Confusion Matrix": conf_matrix,
    "ROC AUC": round(roc_auc, 3),
    "Average Precision (AP)": round(average_precision, 3)
}
print(model_performance)
```



```
{'Confusion Matrix': array([[ 10,   1],
       [ 22, 197]], dtype=int64), 'ROC AUC': 0.959, 'Average Precision (AP)': 0.998}
```

Evaluation of the FMax Variable Threshold

This classification report evaluates a model with a threshold set at the 5th percentile (FMax = 76181.75) for predicting NSD status. Key insights include:

Class 0 (Negative): Low precision (35.9%) indicates many false positives.
High recall (66.7%) shows the model captures most true negatives.

F1-score (46.7%) reflects poor overall performance for negatives.

Class 1 (Positive): Very high precision (99%) and recall (96.6%) demonstrate excellent detection of positives.
F1-score (97.8%) confirms strong performance for this class.
Overall Metrics: Accuracy (95.8%) and ROC-AUC (0.817) indicate good overall model performance. Macro average shows imbalance, with weaker performance for Class 0. Weighted average highlights the dominance of Class 1 in the dataset.
Overall, this means that this variable has a strong positive relationship to determining positivity for NDS. Test results over an FMax value of 76181.75 give an accuracy level of almost 96%. This accuracy level is somewhat misleading however because the total count of negative results is much fewer. The model predicts most true negatives but also tends to give quite a few false positives as well.

The conclusion for this variable by itself is that it may be useful for an early screening, but using this value alone is not sufficient for a definitive diagnosis.

12. Evaluation of Time to Threshold to Determine Disease Positivity

In [33]:
```
# List of percentiles to evaluate
percentiles = np.arange(0, 1.01, 0.01)  # From 0% to 100% in 1% increments

# Initialize variables to track the best threshold and performance
best_percentile = None
best_threshold = None
best_auc = 0

# Create a copy of the DataFrame to avoid modifying the original
df_copy2 = Ultra_Select_Bio.copy()

# Convert SAA_Status to numeric
df_copy2['SAA_Status'] = df_copy2['SAA_Status'].map({'Positive': 1, 'Negative': 0})

# Loop through percentiles
for percentile in percentiles:
    threshold = df_copy2['TTT_24h_Rep1'].quantile(percentile)

    # Use .loc to set Predicted_SAA_Status
    df_copy2.loc[:, 'Predicted_SAA_Status'] = (df_copy2['TTT_24h_Rep1'] > threshold

    # Calculate AUC and performance metrics
    auc_score = roc_auc_score(df_copy2['SAA_Status'], df_copy2['Predicted_SAA_Statu

    # Update the best threshold if performance improves
    if auc_score > best_auc:
        best_auc = auc_score
        best_percentile = percentile
        best_threshold = threshold

# Print the results
print(f"Best Percentile: {best_percentile * 100:.2f}%")
print(f"Best Threshold (TTT): {best_threshold:.3f}")
print(f"Best ROC-AUC: {best_auc:.3f}")

# Check classification report for the best threshold
df_copy2.loc[:, 'Predicted_SAA_Status'] = (df_copy2['TTT_24h_Rep1'] > best_threshol
print("\nClassification Report:")
print(classification_report(df_copy2['SAA_Status'], df_copy2['Predicted_SAA_Status'
```

```
Best Percentile: 100.00%
Best Threshold (TTT): 23.540
Best ROC-AUC: 0.500

Classification Report:
              precision    recall  f1-score   support

           0      0.027     1.000     0.053        21
           1      0.000     0.000     0.000       745

    accuracy                          0.027       766
   macro avg      0.014     0.500     0.027       766
weighted avg      0.001     0.027     0.001       766
```

In [34]:

```
from ipywidgets import interact, FloatSlider
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Reuse the same DataFrame and best_threshold from the first block
def plot_kde(threshold):
    plt.figure(figsize=(10, 6))
    sns.kdeplot(df_copy2['TTT_24h_Rep1'][df_copy2['SAA_Status'] == 0], fill=True, l
    sns.kdeplot(df_copy2['TTT_24h_Rep1'][df_copy2['SAA_Status'] == 1], fill=True, l
    plt.axvline(threshold, color='r', linestyle='--', label=f'Threshold: {threshold

    # Calculate probabilities
    positive_probability = (df_copy2['TTT_24h_Rep1'] > threshold).mean() * 100
    negative_probability = (df_copy2['TTT_24h_Rep1'] <= threshold).mean() * 100

    plt.title(f'KDE Plot of TTT_24h_Rep1 by SAA Status\nProbability of Positive Res
    plt.xlabel('TTT_24h_Rep1')
    plt.ylabel('Density')
    plt.legend()
    plt.show()

# Create the interactive widget with best_threshold as the initial value
interact(plot_kde, threshold=FloatSlider(value=best_threshold, min=df_copy2['TTT_24
```

```
interactive(children=(FloatSlider(value=23.54, description='threshold', max=23.54, m
in=3.33), Output()), _dom…
```

```
Out[34]: <function __main__.plot_kde(threshold)>
```

Evaluation of TTT Variable

If we consider the kernel density graph shown above, we can see that although there are definite negative correlations between positivity for NSD, the range of values overlaps almost completely until the threshold of 23.54. Below this threshold number, the accuracy level is less than 3%, and has a ROC of .5, meaning the same accuracy as if we flipped a coin. We can not use the TTT variable alone to give us much information except that if the TTT Value is greater than 23.54 the chance of a positive result is less than 1%. This is not very useful however, because this number represents only a small amount of the overall values given.

Using this variable alone is not a useful evaluation method.

13. Conclusion

Our first objective was to determine if we could predict incidence of neuronal synuclein disease (NSD) using logistic regression and the FMax and TTT variables. The logistic regression model performs exceptionally well in identifying positive cases of neuronal synuclein disease (Class 1), with high precision, recall, and overall accuracy (90%). However, it struggles with detecting negative cases (Class 0), reflected in low precision and F1-score. The model's strengths make it suitable for applications prioritizing correct identification of positive cases, but it has limitations in handling imbalanced data which should be addressed to improve negative case predictions. It will be difficult to improve upon this model with the current data, due to the significant imbalance in the testing results from participants in the study.

Our second objective was to further analyze if a prediction of NSD was possible, what would be the threshold to determine a positive or negative result. Looking at the FMax variable, we can predict at just a 5% threshold, the incidence of a correct positive diagnosis is 99%. However, the incidence of false positives is also high with a 35.9% precision level meaning nearly 36% of the people who would show as positive are actually negative. This would potentially be acceptable for a very early, first pass screening test, however, could not provide a definite diagnosis. Looking at just the Time to Threshold variable gave us inconclusive results because the prediction accuracy was only at about 50% for most of the data.
Ultimately, improving either the Logistic Regression model or the Threshold Model will be challenging without including more control subjects in the study. However,

increasing control numbers is particularly difficult with cerebrospinal fluid due to the invasive nature of this medical test. Nevertheless, it is possible to see conclusive evidence that the presence of alpha-synuclein protein in cerebrospinal fluid serves as a strong indicator of the disease.

14. References

1 Data used in the preparation of this article was obtained on 2024-09-18 from the Parkinson’s Progression Markers Initiative (PPMI) database (www.ppmi-info.org/access-dataspecimens/download-data), RRID:SCR_006431. For up-to-date information on the study, visit www.ppmi-info.org. PPMI – a public-private partnership – is funded by the Michael J. Fox Foundation for Parkinson’s Research, and funding partners; including the Michael J. Fox Foundation for Parkinson’s Research and funding partners, including 4D Pharma, Abbvie, AcureX, Allergan, Amathus Therapeutics, Aligning Science Across Parkinson's, AskBio, Avid Radiopharmaceuticals, BIAL, BioArctic, Biogen, Biohaven, BioLegend, BlueRock Therapeutics, Bristol-Myers Squibb, Calico Labs, Capsida Biotherapeutics, Celgene, Cerevel Therapeutics, Coave Therapeutics, DaCapo Brainscience, Denali, Edmond J. Safra Foundation, Eli Lilly, Gain Therapeutics, GE HealthCare, Genentech, GSK, Golub Capital, Handl Therapeutics, Insitro, Jazz Pharmaceuticals, Johnson & Johnson Innovative Medicine, Lundbeck, Merck, Meso Scale Discovery, Mission Therapeutics, Neurocrine Biosciences, Neuron23, Neuropore, Pfizer, Piramal, Prevail Therapeutics, Roche, Sanofi, Servier, Sun Pharma Advanced Research Company, Takeda, Teva, UCB, Vanqua Bio, Verily, Voyager v. 25MAR2024 Therapeutics, the Weston Family Foundation and Yumanity Therapeutics. https://www.ppmi-info.org/

2 Prevalence & Incidence | Parkinson's Foundation https://www.parkinson.org

```
In [ ]:
```