Parkinson_Prediction

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import os

# Define the directory path and list all files in the directory
directory_path = r'C:\Users\HP\Desktop\jupyter projects'
files = os.listdir(directory_path)
print(files)

# Define the file path and check if the file exists
file_path = r'C:\Users\HP\Desktop\jupyter
projects\parkinsons.data'
if os.path.isfile(file_path):
    # Load the .data file into a DataFrame
    df = pd.read_csv(file_path, delimiter=',')

    # Display the first few rows of the DataFrame
    print(df.head())
```

```
else:
    print("File not found. Please check the file path.")

# Display basic information about the DataFrame
print(df.info())

# Display statistical summary of the numeric columns
print(df.describe())

# Display the shape of the DataFrame (number of rows and
columns)
print(df.shape)

# Check for missing values in each column
print(df.isnull().sum())

# Display the distribution of the target variable
print(df['status'].value_counts())

# Convert all columns to numeric, coercing errors to NaN
df = df.apply(pd.to_numeric, errors='coerce')

# Ensure 'status' is treated as a categorical variable
```

```
df['status'] = df['status'].astype('category')

# Select only numeric columns for aggregation
numeric_df = df.select_dtypes(include='number')

# Perform groupby operation and calculate the mean for each 'status'
result = numeric_df.groupby(df['status']).mean()
print(result)

# Separate features (X) and target (Y) variables
X = df.drop(columns=['name', 'status'], axis=1)
Y = df['status']
print(X)
print(Y)

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

# Display the shape of the training and testing data
print(X.shape, X_train.shape, X_test.shape)
```

```
# Standardize features by removing the mean and scaling to unit variance
ss = StandardScaler()
X_train = ss.fit_transform(X_train)  # Fit and transform training data
X_test = ss.transform(X_test)        # Transform testing data

# Create and train the SVM model with a linear kernel
model = svm.SVC(kernel="linear")
model.fit(X_train, Y_train)

# Model evaluation: Accuracy on training and testing data
X_train_pred = model.predict(X_train)
train_data_acc = accuracy_score(Y_train, X_train_pred)
print("Accuracy of training data:", train_data_acc)

X_test_pred = model.predict(X_test)
test_data_acc = accuracy_score(Y_test, X_test_pred)
print('Accuracy of testing data:', test_data_acc)

# Define input data with the same number of features as the training data
input_data = [1.0] * X_train.shape[1]  # Replace with actual feature values
```

```
input_data_np = np.asarray(input_data).reshape(1, -1)

# Standardize the input data
s_data = ss.transform(input_data_np)

# Make a prediction with the trained SVM model
pred = model.predict(s_data)
print(pred)

# Interpret the prediction
if pred[0] == 0:
    print("Negative")
else:
    print("Positive")
```