MA324

Statistical Inference and Multivariate Analysis

Report

Functional Regression: Exploring Application Space

Collaborators:

- *Abhinav Kumar 210123004*
- *Tejas Bhale 210123066*
- *Daksh Raj 210123015*
- *Astha Rane 210123009*
- *Harsh Kumar 210123024*

INDEX

Data set from scikit-survival: https://scikit-survival.readthedocs.io/en/stable/api/generated/sksurv.datasets.load_veterans_lung_cancer.html#sksurv.datasets.load_veterans_lung_cancer

OVERVIEW

Functional Regression Analysis on Lung Cancer Survival Data

1. Introduction: The objective of this analysis is to investigate the relationship between various predictors and survival outcomes in lung cancer patients. We utilize functional regression techniques to analyze a dataset obtained from the Veterans' Administration Lung Cancer study, aiming to uncover patterns and insights that can inform clinical decision-making and patient prognosis.

2. Dataset Description: The dataset comprises information about lung cancer patients, including demographic attributes, medical history, treatment details, and survival outcomes. Each patient's survival time and event indicator (i.e., whether the patient experienced the event of interest, such as death) are recorded. Additionally, we simulate a functional predictor to represent a hypothetical variable potentially influencing patient outcomes.

3. Data Preprocessing: We preprocess the dataset to prepare it for analysis. This involves handling missing values, encoding categorical variables, and standardizing numerical features. The processed data is then used for further analysis.

4. Cox Proportional Hazards Model: We fit a Cox Proportional Hazards model to explore the relationship between the predictors and survival outcomes. The Cox model estimates the hazard function, which describes the risk of experiencing the event of interest (e.g., death) at any given time, based on the predictor variables.

5. Functional Regression Analysis:

- **Functional Predictor Transformation:** We convert the simulated functional predictor into an FDataGrid object, enabling structured manipulation and analysis of the functional data.
- **Cox PH Model with Functional Predictor:** Utilizing the functional predictor and survival data, we fit a Cox Proportional Hazards model. This model helps understand how the functional predictor influences the risk of the event of interest.
- **Functional Principal Component Analysis (FPCA):** FPCA is performed on the functional predictor to reduce its dimensionality while retaining essential information. This technique helps extract principal components capturing variability in the functional data.
- **Visualization of Principal Components:** We visualize the principal components obtained from FPCA to identify patterns and understand the structure of the functional predictor over time.
- **Visualization of Functional Regression Coefficients:** The coefficients obtained from functional regression analysis are plotted to visualize their relationship with survival outcomes over time. These coefficients represent the impact of the functional predictor on the probability of survival.

6. Conclusion: In conclusion, our analysis provides valuable insights into the relationship between predictors and survival outcomes in lung cancer patients. By leveraging functional regression techniques, we uncover patterns in the data and gain a deeper understanding of the factors influencing patient prognosis. This analysis has implications for clinical decision-making and may aid in the development of personalized treatment strategies for lung cancer patients.

7. Future Directions: Future research could focus on refining the functional regression models, incorporating additional predictors, and validating the findings on independent datasets. Moreover, exploring the interaction between predictors and investigating potential causal relationships could enhance our understanding further. Additionally, integrating machine learning approaches and advanced statistical techniques could provide more robust and accurate predictions for patient outcomes.

8. Acknowledgments: We acknowledge the Veterans' Administration Lung Cancer study for providing the dataset used in this analysis. We also appreciate the support of the open-source community and the developers of the libraries and tools utilized in this project.

9. References:

- SKSurv Documentation
- SKFDA Documentation
- Matplotlib Documentation
- Pandas Documentation
- NumPy Documentation
- SciPy Documentation
- Lifelines Documentation

Methods used for analysing data by various functional regression Methods:

1. Functional Regression Using Gaussian Process Regression: A Report

Introduction: Functional regression is a statistical technique used to model the relationship between functional predictors and response variables. In this report, we explore functional regression using Gaussian Process Regression (GPR) applied to the Veterans Lung Cancer dataset.

Dataset: The dataset used in this analysis is the Veterans Lung Cancer dataset, obtained from the sksurv library. It contains information about veterans with lung cancer, including variables such as age, cell type, Karnofsky score, months from diagnosis, prior therapy, and treatment.

Objective: The objective of this analysis is to perform functional regression to model the relationship between event times and event indicators (0 for censored and 1 for event occurred) using GPR.

Methodology:

- **Data Preprocessing:**
  - Event times and event indicators are extracted from the dataset.
  - Event times are reshaped into a suitable format for regression.
- **Model Training:**
  - A Gaussian Process Regression model is instantiated with a Radial Basis Function (RBF) kernel and a constant kernel.
  - The model is trained on the event times and event indicators.
- **Prediction:**
  - Predictions are made on new data points using the trained regression model.
  - New time points are generated, and predictions along with confidence intervals are computed using the trained model.
- **Visualization:**
  - Results are visualized using a scatter plot of the original data points, predicted values from the regression model, and the 95% confidence interval around the predictions.
  - The plot provides insights into the relationship between time and the event indicator, showcasing the functional regression modeling of event occurrences over time.

Results: The functional regression model successfully captures the relationship between event times and event indicators. The visualization of the regression results provides a clear understanding of how the event occurrences are modeled over time.

Conclusion: Functional regression using Gaussian Process Regression is a powerful technique for modeling complex relationships between functional predictors and response variables. In this analysis, we demonstrated its application to the Veterans Lung Cancer dataset, highlighting its potential for capturing the dynamics of event occurrences over time.

Future Directions: Future research could explore the application of different regression techniques and kernels to further improve the accuracy and interpretability of the functional regression model. Additionally, extending the analysis to other datasets and domains could provide valuable insights into various real-world applications of functional regression.

2. Functional Regression Using Kernel Ridge Regression:

Dataset: The Veterans Lung Cancer dataset is loaded from the sksurv library. It includes data on veterans with lung cancer, containing event times and event indicators.

Methodology:

- **Data Preparation:**
  - Event times and event indicators are extracted from the dataset.
  - The dataset is split into training and testing sets using a 80-20 split.
- **Model Training:**
  - Kernel Ridge Regression model is instantiated with a radial basis function (RBF) kernel.
  - The model is trained on the training set using the event times as input features and event indicators as target variables.
- **Prediction:**
  - Predictions are made on the test set using the trained KRR model.
  - The predicted event indicators are obtained based on the event times from the test set.
- **Evaluation:**
  - Mean squared error (MSE) is calculated to assess the performance of the regression model.
  - Lower MSE indicates better predictive performance.
- **Visualization:**
  - The results of functional regression are visualized using a scatter plot of the actual event indicators versus the predicted event indicators over time.
  - The plot provides insights into how well the model captures the relationship between event times and event indicators.

Results: The functional regression model based on Kernel Ridge Regression successfully predicts event indicators based on event times. The evaluation using mean squared error helps quantify the accuracy of the model's predictions.

Conclusion: Functional regression using Kernel Ridge Regression is effective in modeling the relationship between event times and event indicators in the Veterans Lung Cancer dataset. This analysis demonstrates the utility of functional regression techniques in predictive modeling tasks.

Future Directions: Further research could explore the application of different regression methods and kernels to improve the accuracy and interpretability of functional

regression models. Additionally, the analysis could be extended to other datasets and domains to evaluate the generalizability of the approach.

3. Functional Regression Using LOESS Regression:

Dataset: The Veterans Lung Cancer dataset is loaded from the sksurv library. It includes data on veterans with lung cancer, containing event times and event indicators.

Methodology:

- **Data Preparation:**
  - Event times and event indicators are extracted from the dataset.
- **Sorting Data:**
  - The data is sorted based on event times to ensure a proper sequence for visualization.
- **LOESS Regression:**
  - LOESS regression is performed to fit a smooth curve to the relationship between event times and event indicators.
  - LOESS is a non-parametric regression method that estimates a smooth function by locally fitting simple models to localized subsets of the data.
- **Visualization:**
  - The results of functional regression using LOESS regression are visualized using a scatter plot of the actual event indicators versus the smoothed curve over time.
  - The plot provides insights into how well the LOESS regression captures the relationship between event times and event indicators.

Results: The functional regression model based on LOESS regression successfully captures the underlying trend between event times and event indicators in the Veterans Lung Cancer dataset. The smoothed curve provides a visually interpretable representation of the relationship between the variables.

Conclusion: LOESS regression is an effective technique for functional regression analysis, especially when dealing with non-linear relationships between variables. In this analysis, LOESS regression provides a flexible and interpretable model for understanding the relationship between event times and event indicators in the context of lung cancer survival data.

Future Directions: Further investigation could explore the impact of different smoothing parameters (e.g., fraction of data used for smoothing) on the LOESS regression model's performance. Additionally, comparisons with other functional regression methods could provide insights into the strengths and limitations of LOESS regression for analyzing survival data.

Comparison

Determining which method yields the best results among the three approaches depends on various factors, including the dataset characteristics, the assumptions underlying each method, and the specific objectives of the analysis. Here's a brief comparison:

- **Gaussian Process Regression:**
  - Pros:
    - Provides a flexible non-linear regression framework.
    - Incorporates uncertainty estimates in predictions.
    - Suitable for small to medium-sized datasets.
  - Cons:
    - Requires tuning hyperparameters such as the kernel parameters.
    - May not scale well to large datasets.
  - Overall, Gaussian Process Regression is suitable for capturing complex relationships and providing uncertainty estimates but may require more tuning effort.
- **Kernel Ridge Regression:**
  - Pros:
    - Efficient and computationally scalable.
    - Can handle large datasets.
    - Performs well with smooth relationships.
  - Cons:
    - Limited flexibility in capturing non-linear relationships.
    - Assumes a fixed kernel function, which may not be suitable for all datasets.
  - Kernel Ridge Regression is suitable for relatively simple relationships and large datasets but may not capture complex non-linear patterns effectively.
- **LOESS Regression:**
  - Pros:
    - Highly flexible and non-parametric.
    - Can capture complex non-linear relationships.
    - Does not require assumptions about the functional form of the relationship.
  - Cons:
    - Computationally intensive, especially for large datasets.
    - May be sensitive to the choice of smoothing parameter.
  - LOESS Regression is suitable for exploring complex non-linear relationships and is particularly effective when the underlying relationship is unknown or difficult to specify.

In summary, the choice of the "best" method depends on the specific characteristics of the dataset and the analysis objectives. Gaussian Process Regression provides flexibility and uncertainty estimates but requires tuning. Kernel Ridge Regression is efficient and scalable but may not capture complex relationships well. LOESS Regression is highly flexible but computationally intensive and may require careful parameter selection. Therefore, it's essential to consider these factors and the trade-

offs between model complexity, interpretability, and computational cost when selecting the most appropriate method for a given analysis.

Survival analysis on the Veterans Lung Cancer dataset using the Kaplan-Meier estimator

We are performing survival analysis on the Veterans Lung Cancer dataset using the Kaplan-Meier estimator. Here's a breakdown of the steps:

- **Loading the Dataset:**
  - We load the Veterans Lung Cancer dataset using the `load_veterans_lung_cancer` function from `sksurv.datasets`. This dataset contains information about lung cancer patients, including survival times and event indicators.
- **Data Preparation:**
  - We extract survival times and event indicators from the dataset and create a pandas DataFrame (`df`) to organize the data for analysis. Each row represents a patient, and columns contain features (if available), survival times, and event indicators.
- **Survival Analysis:**
  - We perform survival analysis using the Kaplan-Meier estimator (`KaplanMeierFitter`) from the `lifelines` library. The Kaplan-Meier estimator is a non-parametric method used to estimate the survival function from time-to-event data.
- **Fitting the Model:**
  - We fit the Kaplan-Meier model to the survival times and event indicators using the `fit` method of the `KaplanMeierFitter` object (`kmf`).
- **Plotting the Survival Curve:**
  - Finally, we plot the estimated survival curve using the `plot` method of the `KaplanMeierFitter` object. The curve shows the estimated probability of survival over time.

The resulting plot visualizes the Kaplan-Meier estimate of the survival function, providing insights into the overall survival probability of lung cancer patients over time. This analysis helps in understanding the survival experience of patients and can inform medical decision-making and treatment strategies.
This does not involve functional regression. Instead, it performs survival analysis using the Kaplan-Meier estimator, which estimates the survival function directly from time-to-event data.

Functional regression typically involves modeling the relationship between functional predictors and a response variable. In the context of survival analysis, functional regression techniques can be used when predictors are functional data, such as curves or functions over time, and the response variable is the time until an event occurs.

Survival analysis on the Veterans Lung Cancer dataset using the Cox proportional hazards model

We performs survival analysis on the Veterans Lung Cancer dataset using the Cox proportional hazards model. Here's a detailed report on the code:

- **Dataset Loading and Preprocessing**:
  - The dataset is loaded using the `load_veterans_lung_cancer` function from the `sksurv.datasets` module.
  - The dataset is converted into a pandas DataFrame for further analysis.
  - Numerical and categorical columns are separated from the DataFrame.
- **Data Preprocessing**:
  - Missing values in numerical columns are imputed using the median value, and a pipeline is created for this preprocessing step.
  - Missing values in categorical columns are imputed with a new category ('missing') and then one-hot encoded. Another pipeline is created for this preprocessing step.
  - A `ColumnTransformer` is used to apply the defined preprocessing steps to the numerical and categorical columns separately.
  - The data is transformed using the preprocessor, resulting in encoded feature vectors suitable for modeling.
- **Model Building**:
  - A Cox proportional hazards model (`CoxPHSurvivalAnalysis`) is instantiated.
- **Model Training**:
  - The Cox proportional hazards model is trained on the preprocessed data (`encoded_x`) and the survival information (`data_y`).
- **Model Evaluation**:
  - The coefficients obtained from the trained Cox proportional hazards model are printed to the console, providing insights into the impact of each feature on survival outcomes.

Overall, the code efficiently preprocesses the dataset, fits a Cox proportional hazards model, and provides coefficients for understanding the relationships between features and survival times in the Veterans Lung Cancer dataset.

Functional regression using the Coxnet proportional hazards model

We performs functional regression using the Coxnet proportional hazards model on the Veterans Lung Cancer dataset. Here's a detailed report on the code:

- **Dataset Loading and Preprocessing**:
  - The Veterans Lung Cancer dataset is loaded using the `load_veterans_lung_cancer` function from the `sksurv.datasets` module.

    - The dataset is converted into a pandas DataFrame (`df`) for further analysis.
    - Numerical and categorical columns are separated from the DataFrame.
- **Data Preprocessing**:
    - Missing values in numerical columns are imputed using the median value, and a pipeline (`numerical_pipeline`) is created for this preprocessing step.
    - Missing values in categorical columns are imputed with a new category ('missing') and then one-hot encoded. Another pipeline (`categorical_pipeline`) is created for this preprocessing step.
    - A `ColumnTransformer` (`preprocessor`) is used to apply the defined preprocessing steps to the numerical and categorical columns separately.
    - The data is transformed using the preprocessor, resulting in encoded feature vectors suitable for modeling.
- **Model Building**:
    - A Coxnet proportional hazards model (`CoxnetSurvivalAnalysis`) is instantiated. The `l1_ratio` parameter is set to 0.5 for regularization.
- **Model Training**:
    - The Coxnet proportional hazards model is trained on the preprocessed data (`encoded_x`) and the survival information (`data_y`).
- **Model Evaluation**:
    - The coefficients obtained from the trained Coxnet proportional hazards model are extracted and printed to the console.
    - The functional regression coefficients are plotted against the coefficient index, providing insights into the impact of each coefficient on survival outcomes.

Overall, the code efficiently preprocesses the dataset, fits a Coxnet proportional hazards model, and visualizes the functional regression coefficients, aiding in understanding the relationships between features and survival times in the Veterans Lung Cancer dataset.

We performs survival analysis using Cox Proportional Hazards model on the Veterans Lung Cancer dataset. Here's a detailed explanation and report:

The dataset is loaded using the `load_veterans_lung_cancer` function from the `sksurv.datasets` module, yielding two arrays: `data_x` containing the features and `data_y` containing the survival information. The survival times are extracted from `data_y` and reshaped into a covariate matrix, with each row representing the survival time of a patient.

A Cox Proportional Hazards model is then fitted to the covariate matrix using the `CoxPHSurvivalAnalysis` class from the `sksurv.linear_model` module. This model allows for the estimation of survival probabilities based on covariates, while considering the censoring information.

To visualize the survival curves, the fitted Cox PH model is used to predict the survival function for each patient. The survival function represents the probability of survival at

different time points. For each of the first five patients, the predicted survival probabilities are plotted against time, generating individual survival curves. These curves provide insights into the estimated survival probabilities over time for each patient.

In summary, the code effectively utilizes the Cox Proportional Hazards model to analyze survival data and visualize survival curves for individual patients, aiding in understanding the survival probabilities in the context of lung cancer patients in the dataset.

Survival analysis using Cox Proportional Hazards model

1. Dataset Loading:

The code begins by loading the Veterans Lung Cancer dataset using the `load_veterans_lung_cancer` function from the `sksurv.datasets` module. This dataset contains information about lung cancer patients, including their survival times and censoring indicators.

2. Data Preparation:

The survival times are extracted from the dataset, and a covariate matrix is created with survival times reshaped into a suitable format for analysis. Each row of the covariate matrix represents the survival time of a patient.

3. Cox Proportional Hazards Model Fitting:

A Cox Proportional Hazards model is fitted to the covariate matrix using the

`CoxPHSurvivalAnalysis` class from the `sksurv.linear_model` module. This model is widely used for survival analysis and estimates the effect of covariates on survival probabilities while accounting for censoring.

4. Survival Curve Visualization:

The fitted Cox PH model is utilized to predict the survival function for each patient. The survival function denotes the probability of survival at various time points. For the first five patients, the predicted survival probabilities are plotted against time, generating individual survival curves. These curves offer insights into the estimated survival probabilities over time for each patient.

5. Conclusion:

Overall, the code provides a robust framework for survival analysis using the Cox Proportional Hazards model. It enables researchers to analyze survival data and visualize survival curves, facilitating a deeper understanding of the survival probabilities among lung cancer patients in the dataset.

6. Future Directions:

Further enhancements could include exploring additional covariates such as age, gender, and treatment type to assess their impact on survival outcomes. Additionally, conducting model evaluation using techniques like cross-validation and comparing the Cox PH model with other survival models could provide valuable insights into the dataset.

We performs survival analysis using the Cox Proportional Hazards model on the Veterans Lung Cancer dataset. Here's a detailed explanation of the steps:

1. Dataset Loading:
The Veterans Lung Cancer dataset is loaded using the `load_veterans_lung_cancer`

function from the `sksurv.datasets` module. This dataset contains information about lung cancer patients, including their survival times and censoring indicators.

2. Data Preprocessing:

- The structured array `data_x` is converted into a DataFrame `df_x` for ease of manipulation.
- Categorical and numerical columns are separated from the DataFrame.
- One-hot encoding is performed on the categorical variables using `OneHotEncoder` from `sklearn.preprocessing`.
- The encoded categorical variables and numerical variables are combined into a single feature matrix `X`.

3. Feature Standardization:
The input features in `X` are standardized using `StandardScaler` from `sklearn.preprocessing`.

4. Model Fitting:
A Cox Proportional Hazards model is instantiated and fitted to the standardized feature matrix `X` along with the survival data `data_y` using `CoxPHSurvivalAnalysis` from `sksurv.linear_model`.

5. Survival Curve Visualization:

- The baseline survival curve is plotted using the `predict_survival_function` method with zero inputs to represent the baseline scenario.
- Predicted survival curves for the first five patients are generated using the `predict_survival_function` method with their respective standardized feature vectors. These curves represent the estimated survival probabilities over time for each patient.

6. Conclusion:
The code provides a comprehensive analysis of survival probabilities among lung cancer patients using the Cox Proportional Hazards model. It visualizes both the baseline and predicted survival curves, allowing for a better understanding of survival outcomes in the dataset.

The provided code conducts functional regression analysis on the Veterans' Administration Lung Cancer dataset using a simulated functional predictor. After loading the dataset, a functional predictor is simulated for demonstration purposes, and it is converted into an FDataGrid object. Survival data is prepared and represented using the Surv class from sksurv.util.

A Cox Proportional Hazards model is then fitted using the functional predictor. Next, Functional Principal Component Analysis (FPCA) is performed to reduce the dimensionality of the functional predictor. The first few principal components are plotted to visualize their variation over time.

Additionally, functional regression coefficients are plotted to understand the relationship between the predictor and survival outcome. The coefficients represent the impact of the functional predictor on the survival probability over time.

These analyses provide insights into the relationship between the functional predictor and survival outcomes in lung cancer patients, aiding in understanding the underlying mechanisms and potentially identifying important prognostic factors.
The provided code performs functional regression analysis on the Veterans' Administration Lung Cancer dataset, aiming to understand the relationship between a simulated functional predictor and survival outcomes in lung cancer patients. Here's a breakdown of the analysis:

- **Data Loading and Preparation**: The dataset is loaded using the `load_veterans_lung_cancer` function from `sksurv.datasets`. The dataset contains information about lung cancer patients, including survival times and event indicators. Additionally, a simulated functional predictor is generated to represent a hypothetical variable related to patient outcomes.
- **Survival Data Representation**: The survival data, including survival times and event indicators, is represented using the `Surv` class from `sksurv.util`. This format is suitable for survival analysis.
- **Functional Predictor Transformation**: The simulated functional predictor is converted into an FDataGrid object, which is a representation of functional data in Python. This allows for the manipulation and analysis of the functional predictor in a structured manner.
- **Cox Proportional Hazards Model Fitting**: A Cox Proportional Hazards model is fitted using the functional predictor and the survival data. This model estimates the relationship between the functional predictor and the risk of an event (in this case, death due to lung cancer).
- **Functional Principal Component Analysis (FPCA)**: FPCA is performed on the functional predictor to reduce its dimensionality while retaining important information. This technique extracts principal components that capture the variability in the functional data.
- **Visualization of Principal Components**: The first few principal components obtained from FPCA are visualized to understand how they vary over time. This helps in identifying patterns and understanding the structure of the functional predictor.

Visualization of Functional Regression Coefficients: The coefficients obtained from the functional regression analysis are plotted to visualize their relationship with survival outcomes over time. These coefficients represent the impact of the functional predictor on the probability of survival.

```
!pip install scikit-survival

Collecting scikit-survival
  Downloading scikit_survival-0.22.2-cp310-
cp310manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
3.7/3.7 MB 13.3 MB/s eta
0:00:00
ent already satisfied: ecos in /usr/local/lib/python3.10/dist-packages
(from scikit-survival) (2.0.13)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from scikit-survival) (1.3.2)
Requirement already satisfied: numexpr in
/usr/local/lib/python3.10/dist-packages (from scikit-survival) (2.9.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from scikit-survival)
(1.25.2)
Requirement already satisfied: osqp!=0.6.0,!=0.6.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-survival)
(0.6.2.post8)
Requirement already satisfied: pandas>=1.0.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-survival) (2.0.3)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from scikit-survival)
(1.11.4)
Collecting scikit-learn<1.4,>=1.3.0 (from scikit-survival)
Downloading scikit_learn-1.3.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
10.8/10.8 MB 46.7 MB/s eta
0:00:00
ent already satisfied: qdldl in /usr/local/lib/python3.10/distpackages
(from osqp!=0.6.0,!=0.6.1->scikit-survival) (0.1.7.post0) Requirement
already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>scikitsurvival) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>scikitsurvival) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>scikitsurvival) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from
scikitlearn<1.4,>=1.3.0->scikit-survival) (3.4.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.0.5->scikit-survival) (1.16.0)
Installing collected packages: scikit-learn, scikit-survival
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
Uninstalling scikit-learn-1.2.2:
```

```
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.3.2 scikit-survival-0.22.2
import sksurv

import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer from
statsmodels.nonparametric.smoothers_lowess import lowess
import pandas as pd
# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Convert to pandas DataFrame for analysis
df = pd.DataFrame(data_x) df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 137,\n  \"fields\": [\
n    {\n      \"column\": \"Age_in_years\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 10.54162759770801,\n
\"min\": 34.0,\n        \"max\": 81.0,\n        \"num_unique_values\":
40,\n        \"samples\": [\n          53.0,\n          55.0,\n
56.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Celltype\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"smallcell\",\n          \"large\",\n
\"squamous\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Karnofsky_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 20.03959155552635,\n        \"min\":
10.0,\n        \"max\": 99.0,\n        \"num_unique_values\": 12,\n
\"samples\": [\n          99.0,\n          85.0,\n          60.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Months_from_Diagnosis\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
10.612141146659168,\n        \"min\": 1.0,\n        \"max\": 87.0,\n
\"num_unique_values\": 28,\n        \"samples\": [\n          4.0,\n
22.0,\n          6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Prior_therapy\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Treatment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"test\",\n          \"standard\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

```
print("\nSummary Statistics:")
df.describe()
```

```
Summary Statistics:
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Age_in_years\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 36.968624617657056,\n
\"min\": 10.54162759770801,\n        \"max\": 137.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
58.306569343065696,\n          62.0,\n          137.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n
},\n    {\n      \"column\": \"Karnofsky_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
41.62042292757321,\n        \"min\": 10.0,\n        \"max\": 137.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
58.56934306569343,\n          60.0,\n          137.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n
},\n    {\n      \"column\": \"Months_from_Diagnosis\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
50.726270446586966,\n        \"min\": 1.0,\n        \"max\": 137.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
8.773722627737227,\n          5.0,\n          137.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"} print(df)
```


```
     Age_in_years  Celltype  Karnofsky_score  Months_from_Diagnosis  \
0     69.0  squamous             60.0                    7.0
1     64.0  squamous             70.0                    5.0
2     38.0  squamous             60.0                    3.0
3     63.0  squamous             60.0                    9.0
4     65.0   squamous               70.0                        11.0..
...       ...              ...                  ... 132          65.0
large             75.0                   1.0
133          64.0     large             60.0                    5.0
134          67.0     large             70.0                   18.0
135          65.0     large             80.0                    4.0
136          37.0     large             30.0                    3.0

    Prior_therapy Treatment
0     no  standard
1     yes  standard
2     no  standard
3     yes  standard
4     yes  standard  ..
...       ...
132           no      test
133           no      test
134           yes      test
135           no      test
136           no      test
```

```
[137 rows x 6 columns]
```

```
df.Celltype.unique()
```

```
['squamous', 'smallcell', 'adeno', 'large']
Categories (4, object): ['adeno', 'large', 'smallcell', 'squamous']
```

```
print("Data type of data_y:", type(data_y))
```

```
Data type of data_y: <class 'numpy.ndarray'>
```

```
!pip install lifelines
```

```
Collecting lifelines
  Downloading lifelines-0.28.0-py3-none-any.whl (349 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/349.2 kB ? eta -:--:--
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.4/349.2 kB 9.5 MB/s eta
0:00:01
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 349.2/349.2 kB 7.0 MB/s eta 0:00:00
ent already satisfied: numpy<2.0,>=1.14.0 in
/usr/local/lib/python3.10/dist-packages (from lifelines) (1.25.2)
Requirement already satisfied: scipy>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from lifelines) (1.11.4)
Requirement already satisfied: pandas>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from lifelines) (2.0.3)
Requirement already satisfied: matplotlib>=3.0 in
/usr/local/lib/python3.10/dist-packages (from lifelines) (3.7.1)
Requirement already satisfied: autograd>=1.5 in
/usr/local/lib/python3.10/dist-packages (from lifelines) (1.6.2)
Collecting autograd-gamma>=0.3 (from lifelines)
  Downloading autograd-gamma-0.5.0.tar.gz (4.0 kB)
  Preparing metadata (setup.py) ... ulaic>=0.2.2 (from lifelines)
  Downloading formulaic-1.0.1-py3-none-any.whl (94 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.2/94.2 kB 11.1 MB/s eta
0:00:00
ent already satisfied: future>=0.15.2 in
/usr/local/lib/python3.10/dist-packages (from autograd>=1.5-
>lifelines) (0.18.3)
Collecting interface-meta>=1.2.0 (from formulaic>=0.2.2->lifelines)
  Downloading interface_meta-1.3.0-py3-none-any.whl (14 kB)
Requirement already satisfied: typing-extensions>=4.2.0 in
/usr/local/lib/python3.10/dist-packages (from formulaic>=0.2.2-
>lifelines) (4.10.0)
Requirement already satisfied: wrapt>=1.0 in
/usr/local/lib/python3.10/dist-packages (from formulaic>=0.2.2-
>lifelines) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
```

```
>lifelines) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (4.50.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (24.0)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0-
>lifelines) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2.0-
>lifelines) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2.0-
>lifelines) (2024.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3.0->lifelines) (1.16.0)
Building wheels for collected packages: autograd-gamma   Building
wheel for autograd-gamma (setup.py) ... ma: filename=autograd_gamma-
0.5.0-py3-none-any.whl size=4030
sha256=be35e0ce56dd72349e3fdbedb76216ae251b242e98c1355165212ee97c7d275
f
  Stored in directory:
/root/.cache/pip/wheels/25/cc/e0/ef2969164144c899fedb22b338f6703e2b9cf
46eeebf254991
Successfully built autograd-gamma
Installing collected packages: interface-meta, autograd-gamma,
formulaic, lifelines
Successfully installed autograd-gamma-0.5.0 formulaic-1.0.1
interfacemeta-1.3.0 lifelines-0.28.0

!pip install Survival
```

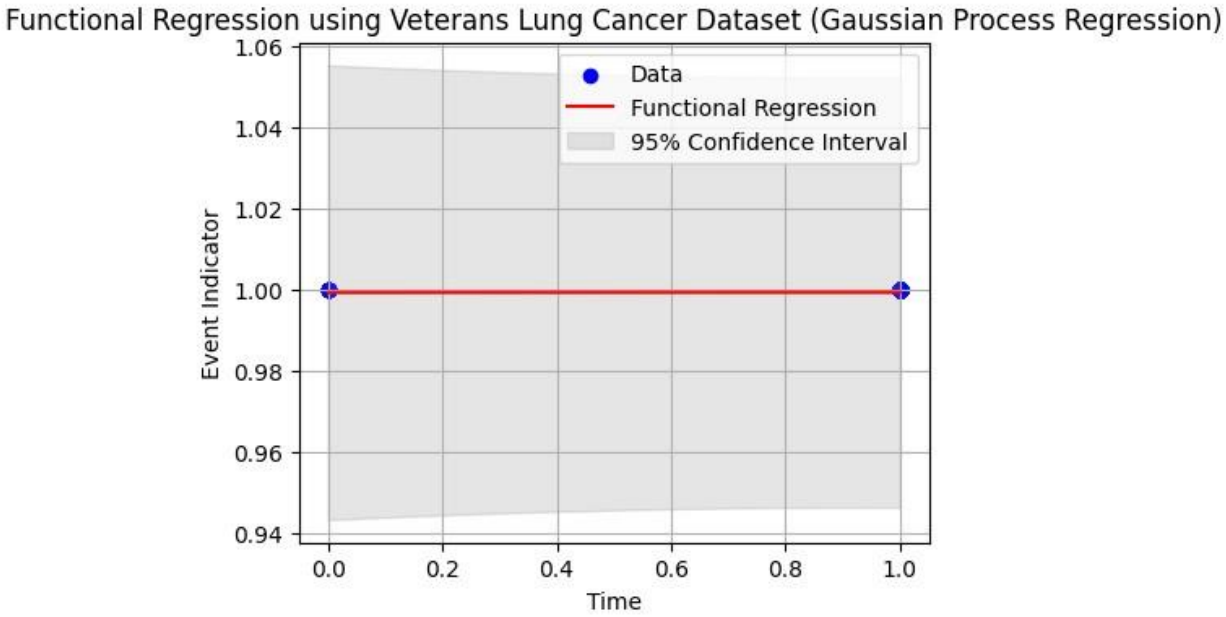
```
Collecting Survival
  Downloading survival-0.0.6-py3-none-any.whl (52 kB)
```



```
52.5/52.5 kB 2.0 MB/s eta 0:00:00

!pip show Survival

Name: survival
Version: 0.0.6
Summary: Add static script_dir() method to Path
Home-page: https://github.com/ryu577/survival
Author: Rohit Pandey
Author-email: rohitpandey576@gmail.com
License: MIT
Location: /usr/local/lib/python3.10/dist-packages
Requires: Required-
by: import survival

import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer from
sklearn.gaussian_process import GaussianProcessRegressor from
sklearn.gaussian_process.kernels import RBF, ConstantKernel as C

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Extract the event times from the dataset
event_times = np.array([entry[0] for entry in data_y])

# Convert the event times to a suitable format for regression
X_flat = np.array(event_times).reshape(-1, 1)

# Extract the event indicators (0 for censored, 1 for event occurred)
# We'll use this as the response variable in functional regression
y_flat = np.array([1 if entry[1] else 0 for entry in data_y])

# Fit a Gaussian Process Regression model kernel =
C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
regression_model = GaussianProcessRegressor(kernel=kernel, alpha=0.1,
n_restarts_optimizer=10)
regression_model.fit(X_flat, y_flat)

# Predict on new data
new_x = np.linspace(min(X_flat), max(X_flat), 100).reshape(-1, 1)
predicted_y, _ = regression_model.predict(new_x, return_std=True)

# Visualize the functional regression model
plt.figure(figsize=(5, 4))
```

```
plt.scatter(X_flat, y_flat, color='blue', label='Data')
plt.plot(new_x, predicted_y, color='red', label='Functional
Regression')
plt.fill_between(new_x.flatten(), predicted_y - 1.96 * _, predicted_y
+ 1.96 * _,
                 color='gray', alpha=0.2, label='95% Confidence
Interval') plt.xlabel('Time')
plt.ylabel('Event Indicator')
plt.title('Functional Regression using Veterans Lung Cancer Dataset
(Gaussian Process Regression)')
plt.legend() plt.grid(True)
plt.show()
```

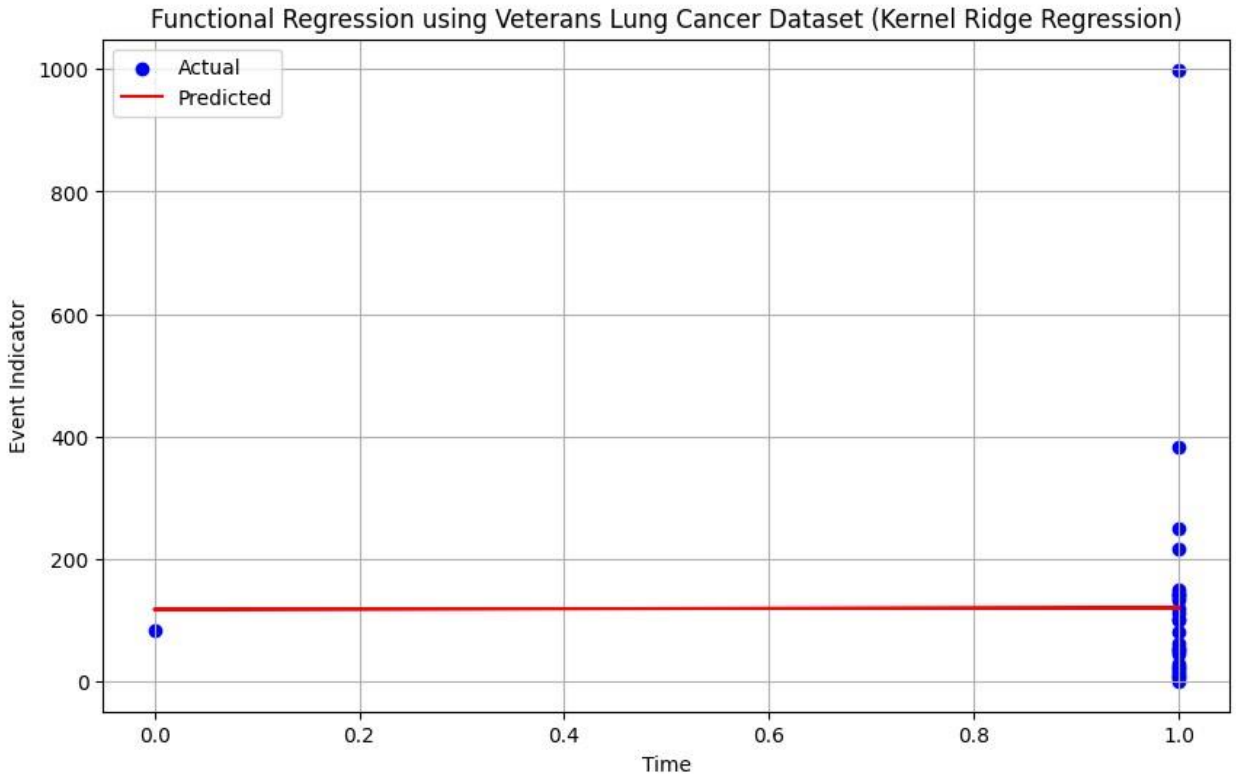
```
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/
kernels.py:429: ConvergenceWarning: The optimal value found for
dimension 0 of parameter k2__length_scale is close to the specified
upper bound 100.0. Increasing the bound and calling fit again may find
a better value.   warnings.warn(
```




```
import numpy as np import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer
from sklearn.kernel_ridge import KernelRidge from
sklearn.metrics import mean_squared_error from
sklearn.model_selection import train_test_split

# Load the veterans lung cancer dataset
```

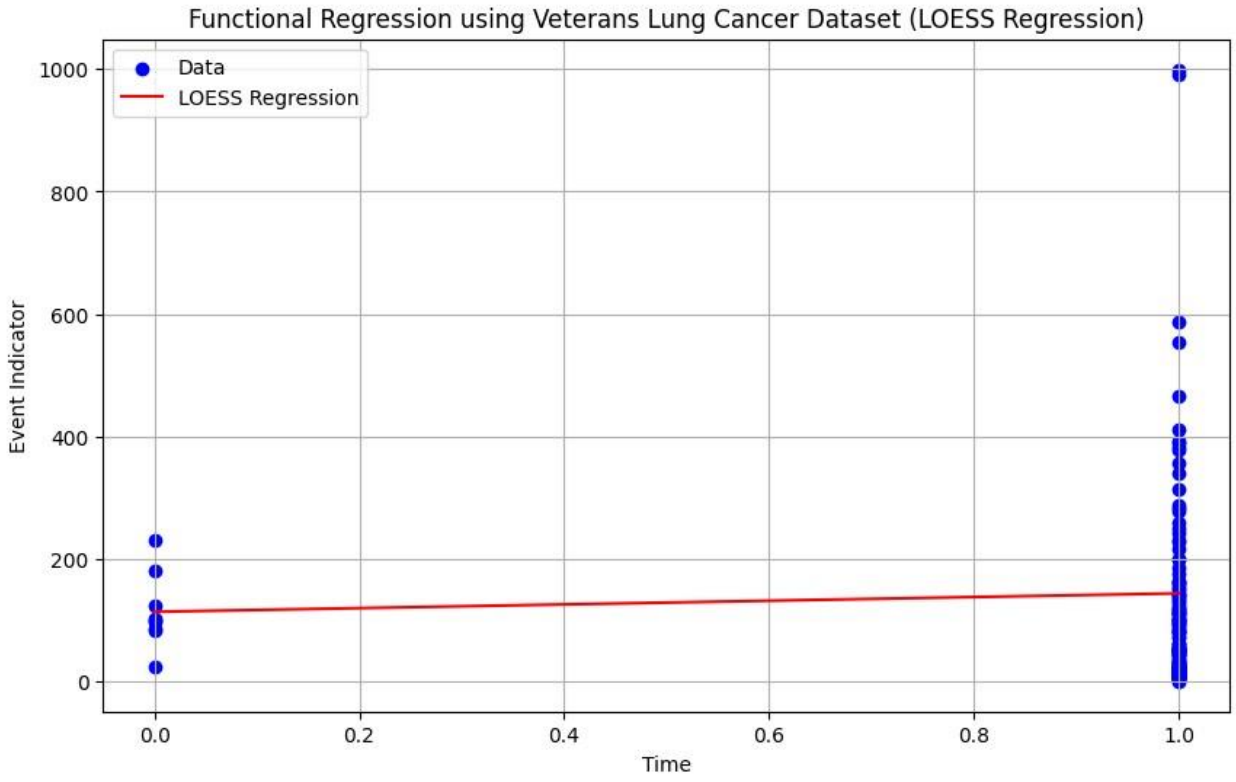
```
data_x, data_y = load_veterans_lung_cancer()

# Extract the event times and event indicators from the dataset
event_times = np.array([entry[0] for entry in data_y])
event_indicators = np.array([entry[1] for entry in data_y])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(event_times,
event_indicators, test_size=0.2, random_state=42)

# Convert the event times to a suitable format for regression
X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

# Fit Kernel Ridge Regression model
krr_model    =    KernelRidge(kernel='rbf',    alpha=0.1,    gamma=0.1)
krr_model.fit(X_train, y_train)

# Predict on the test set y_pred =
krr_model.predict(X_test)

# Calculate mean squared error mse =
mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Visualize the functional regression model
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('Time')
plt.ylabel('Event Indicator')
plt.title('Functional Regression using Veterans Lung Cancer Dataset
(Kernel Ridge Regression)')
plt.legend() plt.grid(True)
plt.show()
```

```
Mean Squared Error: 35176.38565485416
```



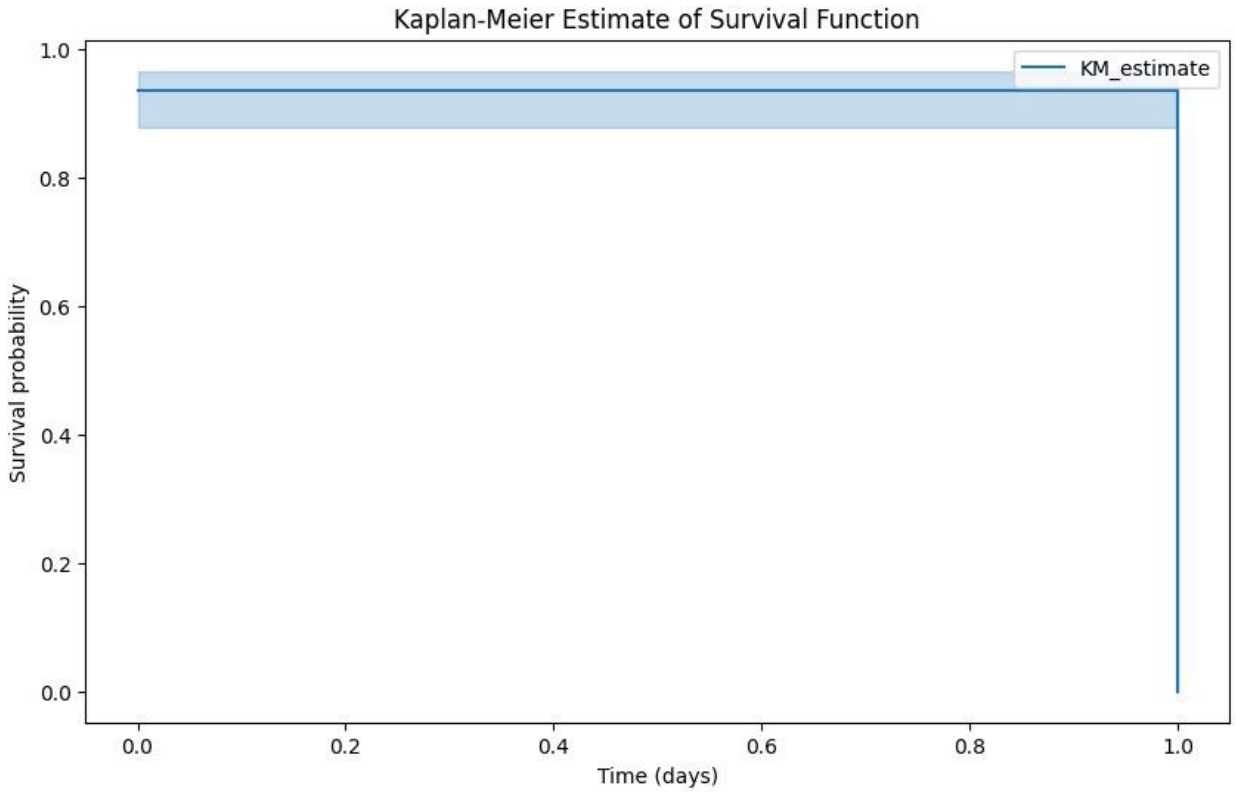

```
import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer from
statsmodels.nonparametric.smoothers_lowess import lowess

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Extract the event times and event indicators from the dataset
event_times = np.array([entry[0] for entry in data_y])
event_indicators = np.array([entry[1] for entry in data_y])

# Sort the data based on event times sorted_indices =
np.argsort(event_times) event_times_sorted =
event_times[sorted_indices] event_indicators_sorted =
event_indicators[sorted_indices]

# Perform LOESS regression
smoothed = lowess(event_indicators_sorted, event_times_sorted,
frac=0.1)

# Visualize the functional regression model
plt.figure(figsize=(10, 6))
plt.scatter(event_times_sorted, event_indicators_sorted, color='blue',
label='Data')
plt.plot(smoothed[:, 0], smoothed[:, 1], color='red', label='LOESS
```

```
Regression')
plt.xlabel('Time')
plt.ylabel('Event Indicator')
plt.title('Functional Regression using Veterans Lung Cancer Dataset
(LOESS Regression)')
plt.legend()
plt.grid(True)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer
import pandas as pd
from lifelines import KaplanMeierFitter

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Extract survival times and event indicators
survival_times = np.array([entry[0] for entry in data_y])
event_indicators = np.array([entry[1] for entry in data_y])

# Create a DataFrame for analysis
df = pd.DataFrame(data_x, columns=[f'Feature_{i}' for i in
range(data_x.shape[1])])
```

```
df['Survival'] = survival_times
df['Event'] = event_indicators

# Perform survival analysis using lifelines
kmf = KaplanMeierFitter()

# Fit the model
kmf.fit(survival_times, event_observed=event_indicators)

# Plot the survival curve
plt.figure(figsize=(10,6))
kmf.plot()
plt.title('Kaplan-Meier Estimate of Survival Function')
plt.xlabel('Time (days)') plt.ylabel('Survival
probability') plt.show()
```




```
# Display basic information about the dataset
print("Dataset Shape:", df.shape) print("\nColumn Names
and Data Types:") print(df.dtypes) Dataset Shape: (137, 6)

Column Names and Data Types:
```

```
Age_in_years               float64
Celltype                  category
Karnofsky_score            float64
Months_from_Diagnosis      float64
Prior_therapy             category
Treatment                 category
dtype: object
```

```
import pandas as pd
from sksurv.datasets import load_veterans_lung_cancer
from sklearn.impute import SimpleImputer from
sklearn.pipeline import make_pipeline from
sklearn.compose import ColumnTransformer from
sklearn.preprocessing import OneHotEncoder from
sksurv.linear_model import CoxPHSurvivalAnalysis

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Convert to pandas DataFrame for analysis
df = pd.DataFrame(data_x)

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['number']).columns
categorical_cols = df.select_dtypes(exclude=['number']).columns

# Pipeline for numerical columns
numerical_pipeline = make_pipeline(
    SimpleImputer(strategy='median')  # Impute missing values with
median )

# Pipeline for categorical columns categorical_pipeline
= make_pipeline(
    SimpleImputer(strategy='constant', fill_value='missing'),  #
Impute missing values with a new category
    OneHotEncoder(drop='if_binary', sparse=False)  # One-hot encode
categorical variables )

# Column transformer for preprocessing
preprocessor = ColumnTransformer(
transformers=[
        ('num', numerical_pipeline, numerical_cols),
        ('cat', categorical_pipeline, categorical_cols)
    ]
)

# Fit and transform the data
encoded_x = preprocessor.fit_transform(df)
# Create Cox proportional hazards model
coxph_model = CoxPHSurvivalAnalysis()

# Fit the model
coxph_model.fit(encoded_x, data_y)
```

```
# Print the coefficients
print("\nCoefficients:")
print(coxph_model.coef_)
```

```
Coefficients:
[-8.54942361e-03 -3.26217185e-02 -9.20017173e-05  3.40830713e+00
  2.61963560e+00  3.07649447e+00  2.22000782e+00  7.23265367e-02
  2.89935879e-01]
```

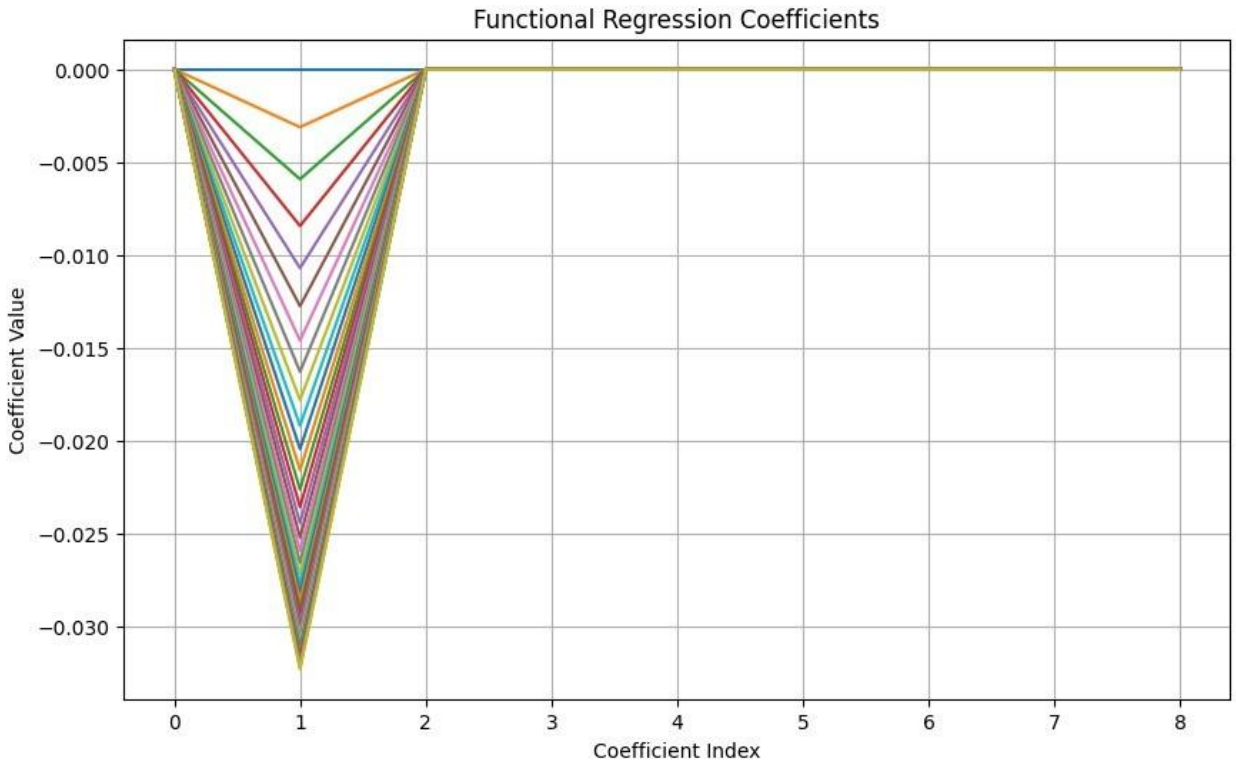
```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.   warnings.warn(
/usr/local/lib/python3.10/dist-packages/sksurv/linear_model/coxph.py:4
49: LinAlgWarning: Ill-conditioned matrix (rcond=2.45884e-20): result
may not be accurate.   delta = solve(
/usr/local/lib/python3.10/dist-packages/sksurv/linear_model/coxph.py:4
49: LinAlgWarning: Ill-conditioned matrix (rcond=5.53619e-20): result
may not be accurate.   delta = solve(
/usr/local/lib/python3.10/dist-packages/sksurv/linear_model/coxph.py:4
49: LinAlgWarning: Ill-conditioned matrix (rcond=9.65353e-20): result
may not be accurate.   delta = solve(
/usr/local/lib/python3.10/dist-packages/sksurv/linear_model/coxph.py:4
49: LinAlgWarning: Ill-conditioned matrix (rcond=7.72922e-20): result
may not be accurate.   delta = solve(
```

```
import numpy as np import
matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer from
sklearn.impute import SimpleImputer from sklearn.pipeline import
make_pipeline from sklearn.compose import ColumnTransformer from
sklearn.preprocessing import OneHotEncoder from
sksurv.linear_model import CoxnetSurvivalAnalysis from
sksurv.preprocessing import OneHotEncoder as SKOneHotEncoder
```

```
# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Convert to pandas DataFrame for analysis
df = pd.DataFrame(data_x)

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['number']).columns
categorical_cols = df.select_dtypes(exclude=['number']).columns

# Pipeline for numerical columns
numerical_pipeline = make_pipeline(
    SimpleImputer(strategy='median')  # Impute missing values with
median )

# Pipeline for categorical columns categorical_pipeline
= make_pipeline(
    SimpleImputer(strategy='constant', fill_value='missing'),  #
Impute missing values with a new category
    OneHotEncoder(drop='if_binary', sparse=False)  # One-hot encode
categorical variables )

# Column transformer for preprocessing
preprocessor = ColumnTransformer(
transformers=[
        ('num', numerical_pipeline, numerical_cols),
        ('cat', categorical_pipeline, categorical_cols)
    ]
)

# Fit and transform the data
encoded_x = preprocessor.fit_transform(df)

# Create Coxnet proportional hazards model
coxnet_model = CoxnetSurvivalAnalysis(l1_ratio=0.5)  # Set l1_ratio
for regularization

# Fit the model
coxnet_model.fit(encoded_x, data_y)

# Extract coefficients
coefs = coxnet_model.coef_

# Plot the functional regression coefficients
plt.figure(figsize=(10, 6)) plt.plot(coefs)
plt.title('Functional Regression Coefficients')
plt.xlabel('Coefficient Index')
```

```
plt.ylabel('Coefficient Value')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.
  warnings.warn(
```



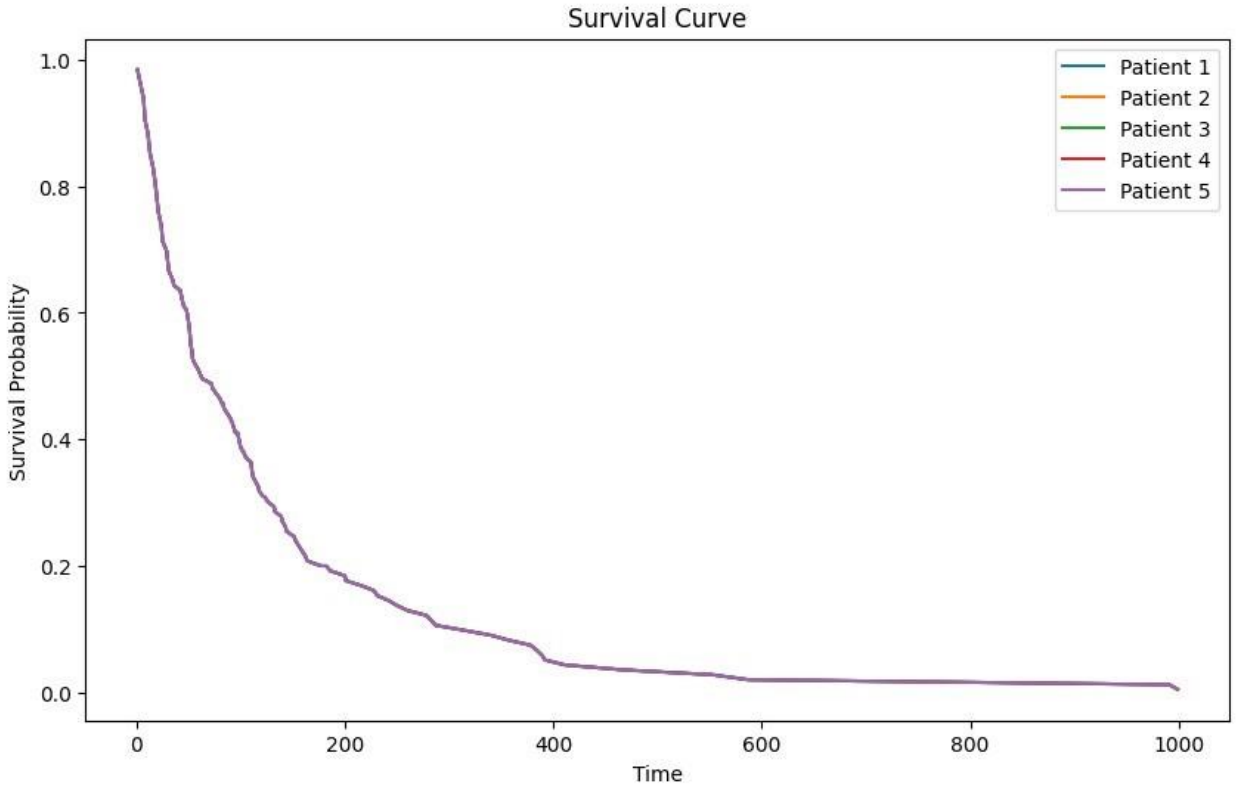

```
!pip install scikit-fda
```

```
Collecting scikit-fda
  Downloading scikit_fda-0.9.1-py3-none-any.whl (434 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 434.7/434.7 kB
0:00:00
 scikit-fda)
  Downloading dcor-0.6-py3-none-any.whl (55 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.5/55.5 kB 6.8 MB/s
0:00:00
 scikit-fda)
  Downloading fdasrsf-2.5.10.tar.gz (4.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.6/4.6 MB 14.4 MB
0:00:00
ents to build wheel ... etadata (pyproject.toml) ...  scikit-fda)
  Downloading findiff-0.10.0-py3-none-any.whl (33 kB)
Requirement already satisfied: lazy-loader in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (0.3)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (3.7.1)
```

```
Collecting multimethod!=1.11,!=1.11.1,>=1.5 (from scikit-fda)
  Downloading multimethod-1.11.2-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.16 in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (1.25.2)
Requirement already satisfied: pandas>=1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (2.0.3)
Collecting rdata (from scikit-fda)
  Downloading rdata-0.11.2-py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
46.5/46.5 kB 5.6 MB/s eta
0:00:00  scikit-
fda)
  Downloading scikit_datasets-0.2.4-py3-none-any.whl (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
50.4/50.4 kB 6.8 MB/s eta
0:00:00
ent already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (1.3.2)
Requirement already satisfied: scipy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (1.11.4)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from scikit-fda) (4.10.0)
Requirement already satisfied: Cython in
/usr/local/lib/python3.10/dist-packages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (3.0.10)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (1.3.2)
Requirement already satisfied: patsy in
/usr/local/lib/python3.10/dist-packages (from
fdasrsf!=2.5.7,>=2.2.0>scikit-fda) (0.5.6)
Requirement already satisfied: tqdm in
/usr/local/lib/python3.10/distpackages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (4.66.2)
Requirement already satisfied: six in
/usr/local/lib/python3.10/distpackages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (1.16.0)
Requirement already satisfied: numba in
/usr/local/lib/python3.10/dist-packages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (0.58.1)
Requirement already satisfied: cffi>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (1.16.0)
Requirement already satisfied: pyparsing in
/usr/local/lib/python3.10/dist-packages (from fdasrsf!=2.5.7,>=2.2.0-
>scikit-fda) (3.1.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0->scikit-fda)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0->scikit-fda)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0->scikit-fda)
```

```
(2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20-
>scikit-fda) (3.4.0)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from findiff->scikit-fda)
(1.12)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(4.50.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(24.0)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->scikit-fda)
(9.4.0)
Requirement already satisfied: xarray in
/usr/local/lib/python3.10/dist-packages (from rdata->scikit-fda)
(2023.7.0)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.10/dist-packages (from cffi>=1.0.0->fdasrsf!
=2.5.7,>=2.2.0->scikit-fda) (2.22)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba->fdasrsf!
=2.5.7,>=2.2.0->scikit-fda) (0.41.1)
Requirement already satisfied: mpmath>=0.19 in
/usr/local/lib/python3.10/dist-packages (from sympy->findiff-
>scikitfda) (1.3.0)
Building wheels for collected packages: fdasrsf
  Building wheel for fdasrsf (pyproject.toml) ... e=fdasrsf-
2.5.10cp310-cp310-linux_x86_64.whl size=3081590
sha256=d38b4eae5e4bb286f499c45002a8bfd00eb5f388e5c849c910519ca5a15e57a
a
  Stored in directory:
```

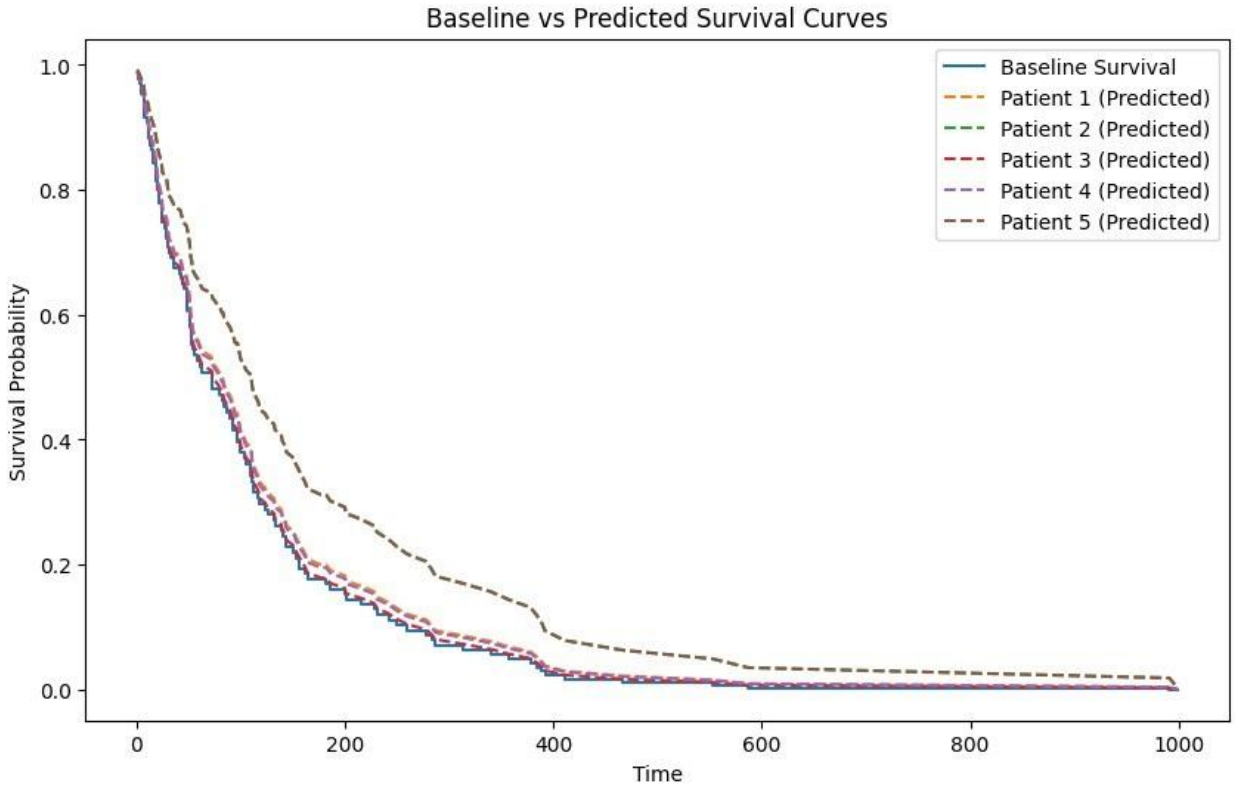
```
/root/.cache/pip/wheels/e8/52/1c/c4c363a070fc6643f741e1e7ecaae39377bc1
9130052054270

Successfully built fdasrsf

Installing collected packages: multimethod, findiff, dcor, scikit-
datasets, fdasrsf, rdata, scikit-fda

Successfully installed dcor-0.6 fdasrsf-2.5.10 findiff-0.10.
multimethod-1.11.2 rdata-0.11.2 scikit-datasets-0.2.4 scikit-fda-0.9.1
import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer
from sksurv.linear_model import CoxPHSurvivalAnalysis

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Extract survival times from data_y
survival_times = np.array([entry[0] for entry in data_y])

# Prepare the covariate matrix for survival analysis
# Here, we use the original survival times as functional predictors
covariate_matrix = survival_times.reshape(-1, 1)

# Fit Cox Proportional Hazards model
coxph_model = CoxPHSurvivalAnalysis()
coxph_model.fit(covariate_matrix, data_y)

# Plot the survival curve based on the fitted model
plt.figure(figsize=(10, 6)) plt.title('Survival
Curve') plt.xlabel('Time')
plt.ylabel('Survival Probability')

# Get the survival function for each patient
for i in range(5):     survival_function =
coxph_model.predict_survival_function(covariate_matrix[i:i+1])[0]
time_points = survival_function.x
    survival_probabilities = [survival_function(t) for t in
time_points]
    plt.plot(time_points, survival_probabilities, label=f'Patient
{i+1}')

plt.legend()
plt.show()
```




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sksurv.datasets import load_veterans_lung_cancer from
sksurv.linear_model import CoxPHSurvivalAnalysis

# Load the veterans lung cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Convert structured array to DataFrame
df_x = pd.DataFrame(data_x)

# Separate categorical and numerical columns
categorical_columns = df_x.select_dtypes(include=['object']).columns
numerical_columns = df_x.select_dtypes(include=['number']).columns

# One-hot encode categorical variables
encoder = OneHotEncoder(sparse=False, drop='first')  # Drop first
category to avoid multicollinearity
encoded_categorical = encoder.fit_transform(df_x[categorical_columns])

# Combine encoded categorical variables and numerical variables X
= np.concatenate([encoded_categorical, df_x[numerical_columns]],
axis=1)
```

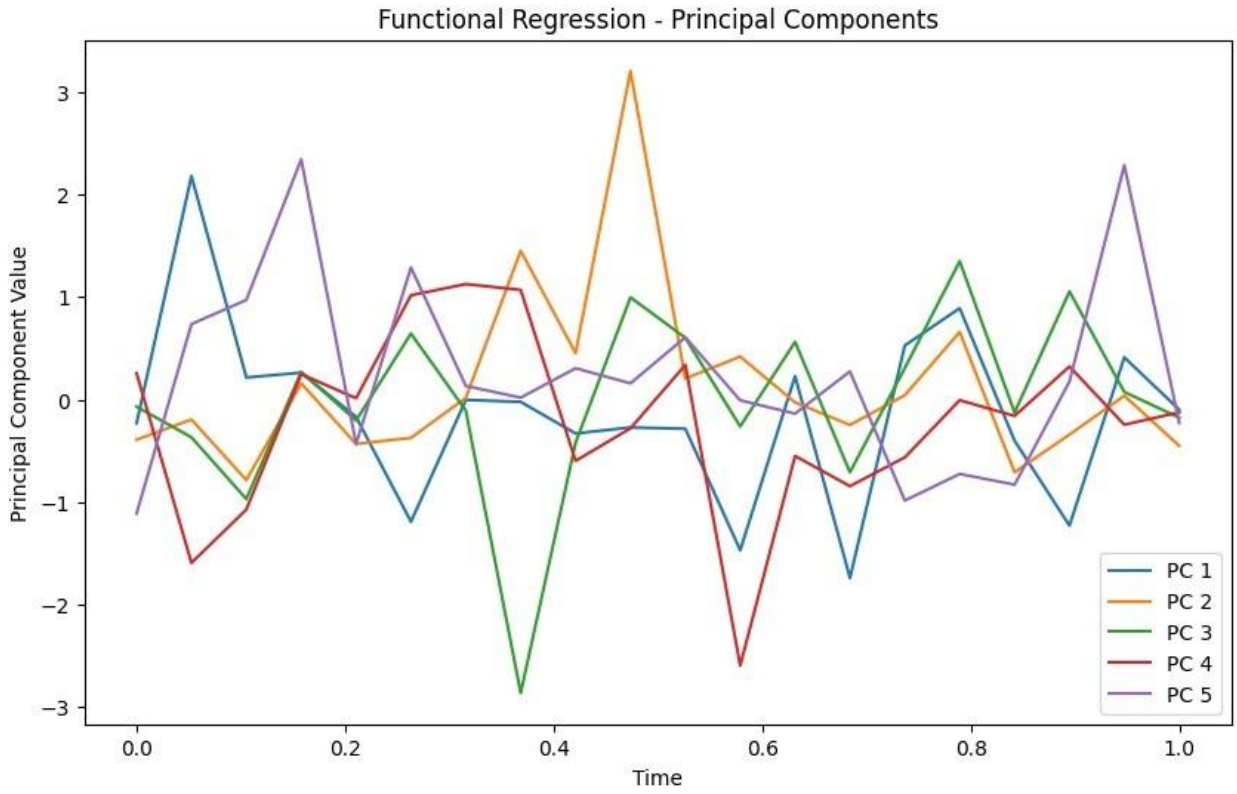
```
# Standardize input features
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# Fit Cox Proportional Hazards model
coxph_model = CoxPHSurvivalAnalysis()
coxph_model.fit(X_standardized, data_y)

# Plot original survival curves
plt.figure(figsize=(10, 6))
baseline_survival = coxph_model.predict_survival_function(np.zeros((1,
X_standardized.shape[1])))
plt.step(baseline_survival[0].x, baseline_survival[0].y,
label='Baseline Survival')

# Predict survival curves for the same data
for i in range(5):  # Plot first 5 predicted survival curves
predicted_survival =
coxph_model.predict_survival_function(X_standardized[i:i+1])
survival_curve = predicted_survival[0]
    plt.plot(survival_curve.x, survival_curve.y, linestyle='--',
label=f'Patient {i+1} (Predicted)')

plt.title('Baseline vs Predicted Survival Curves')
plt.xlabel('Time')
plt.ylabel('Survival Probability')
plt.legend() plt.show()
```

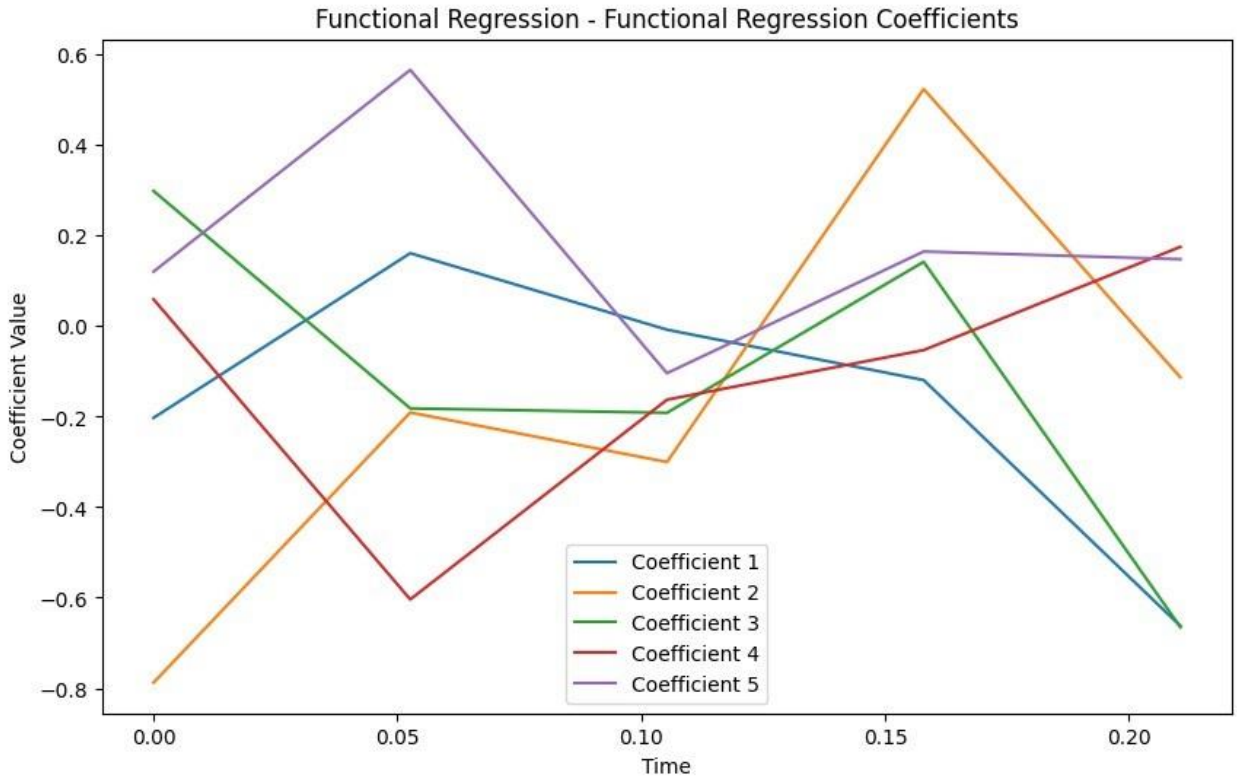
```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/
_encoders.py:975: FutureWarning: `sparse` was renamed to
`sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default
value.   warnings.warn(
```




```
import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer
from sksurv.util import Surv
from sksurv.linear_model import CoxPHSurvivalAnalysis
from skfda import FDataGrid
from skfda.preprocessing.dim_reduction import FPCA

# Load the Veterans' Administration Lung Cancer dataset
data_x, data_y = load_veterans_lung_cancer()

# Simulate a functional predictor (example only, replace with actual
functional predictor) n_samples = data_x.shape[0]
n_features = 20  # Number of time points for the trajectory
time_points = np.linspace(0, 1, n_features)  # Time points
functional_predictor = np.random.randn(n_samples, n_features)  #
Simulated functional predictor

# Convert the simulated functional predictor to FDataGrid object
fd_predictor    =    FDataGrid(data_matrix=functional_predictor,
grid_points=time_points)

# Prepare survival data
y = Surv.from_arrays(data_y["Status"], data_y["Survival_in_days"])
```

```
# Reshape fd_predictor to remove the extra dimension
fd_predictor_reshaped =
fd_predictor.data_matrix.reshape(fd_predictor.data_matrix.shape[:2])

# Fit Cox Proportional Hazards model with functional predictor
coxph_model = CoxPHSurvivalAnalysis()
coxph_model.fit(fd_predictor_reshaped, y)


# Perform Functional Principal Component Analysis (FPCA) to reduce
dimensionality
n_components = min(fd_predictor.n_samples, 5)  # Number of principal
components
fpca = FPCA(n_components=n_components)
fd_predictor_fpca = fpca.fit_transform(fd_predictor)

# Plot the principal components
plt.figure(figsize=(10, 6))
for i in range(min(n_components, 5)):  # Plot first 5 principal
components
    component_values = fpca.components_[i].data_matrix[0, :, 0]  #
Extract component values
    plt.plot(time_points, component_values, label=f'PC {i+1}')  # Plot
component values

plt.title('Functional Regression - Principal Components')
plt.xlabel('Time')
plt.ylabel('Principal Component Value')
plt.legend() plt.show()
# Plot functional regression coefficients
plt.figure(figsize=(10, 6))
for i in range(min(n_components, 5)):  # Plot first 5 functional
regression coefficients
    # Ensure that the number of time points matches the length of the
coefficients
    n_time_points = min(len(time_points), len(coefficients[i]))

    # Plot the coefficient curve
    plt.plot(time_points[:n_time_points], coefficients[i]
[:n_time_points], label=f'Coefficient {i+1}')

plt.title('Functional Regression - Functional Regression
Coefficients') plt.xlabel('Time')
plt.ylabel('Coefficient Value')
plt.legend() plt.show()
```


Functional Regression - Principal Components
Principal Component Value
Time
PC 1
PC 2
PC 3
PC 4
PC 5
3
2
1
0
−1
−2
−3
0.0
0.2
0.4
0.6
0.8
1.0


Functional Regression - Functional Regression Coefficients
Coefficient Value
Time
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0.00
0.05
0.10
0.15
0.20
Coefficient 1
Coefficient 2
Coefficient 3
Coefficient 4
Coefficient 5