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.

Exploring Survival Analysis with Random Survival Forest

Introduction:

Survival analysis is a statistical method used to analyze the time until an event of interest occurs. It is widely used in various fields such as medical research, engineering, and economics. In this project, we explore survival analysis using Random Survival Forest (RSF) on the veterans lung cancer dataset. RSF is an ensemble learning method based on decision trees, which is particularly effective for survival analysis due to its ability to handle censored data and capture complex non-linear relationships.

Data Preprocessing:

The veterans lung cancer dataset was loaded, containing information about veterans with lung cancer, including their survival time and status (whether they survived or not).

Categorical variables were encoded using one-hot encoding to prepare the data for model training.

Model Training:

The dataset was split into training and test sets using a 80-20 ratio.

A Random Survival Forest model was trained on the training set using 100 decision trees.

The survival probabilities for the test set were predicted using the trained model.

Evaluation:

Mean Squared Error (MSE) was used as the evaluation metric to assess the performance of the model.

The MSE calculated between the observed survival times and the predicted survival probabilities was 16027.03, indicating the deviation between the actual and predicted survival times.

Results Visualization:

Kaplan-Meier survival curves were plotted for both the training and test sets to visualize the overall survival probability.

Predicted survival curves were overlaid on the plot to compare them with the Kaplan-Meier curve.

Conclusion:

Random Survival Forest is a powerful tool for survival analysis, capable of handling complex datasets and capturing non-linear relationships.

Despite the relatively high MSE, the model provides valuable insights into survival probabilities and can be further optimized with additional feature engineering and hyperparameter tuning.

Future Directions:

Further investigation can be conducted to identify influential features and improve model performance

Experimentation with different ensemble methods and hyperparameter settings can be explored to enhance predictive accuracy.

Integration of additional data sources and domain knowledge can provide a more comprehensive understanding of survival patterns in lung cancer patients.

Comparison of Survival Analysis Models

Introduction: Survival analysis is a statistical method used to analyze the time until an event of interest occurs. In this project, we compare the performance of different survival analysis models using the veterans lung cancer dataset. The goal is to assess the accuracy of each model in predicting survival probabilities for lung cancer patients.

Data Preprocessing:

The veterans lung cancer dataset was loaded, containing information about veterans with lung cancer, including their survival time and status (whether they survived or not).

Categorical variables were one-hot encoded, and numerical variables were standardized to prepare the data for model training.

Model Training: We trained four different survival analysis models on the preprocessed data:

- Gradient Boosting Survival Analysis (GBST)
- Coxnet Survival Analysis (FLM)
- Cox Proportional Hazards Survival Analysis (Splines)
- Random Survival Forest (Rf)

Evaluation:

Mean Squared Error (MSE) was used as the evaluation metric to assess the performance of each model.

The MSE values obtained for each model were as follows:

GBST: 38690.41494578108

FLM: 37972.84919581442

Splines: 37939.010800408025

Rf: 16027.02924053796

Results:

Random Survival Forest (Rf) achieved an MSE of 16027.03, indicating the deviation between the actual and predicted survival times.

Gradient Boosting Survival Analysis (GBST), Coxnet Survival Analysis (FLM), and Cox Proportional Hazards Survival Analysis (Splines) are yet to be evaluated.

Conclusion:

Random Survival Forest (Rf) performed reasonably well in predicting survival probabilities for lung cancer patients.

Further evaluation is needed to assess the performance of other models and determine the most effective approach for survival analysis in this dataset.

Future Directions:

Experimentation with additional hyperparameter tuning and feature engineering techniques may improve the performance of the models.

Integration of domain knowledge and additional data sources could provide valuable insights into survival patterns among lung cancer patients.

Further research is needed to explore advanced survival analysis techniques and their applications in real-world scenarios.

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.

Code and Output

[1]:
```
!pip install scikit-survival
```

```
Requirement already satisfied: scikit-survival in
/usr/local/lib/python3.10/dist-packages (0.22.2)
Requirement 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/distpackages (from scikit-survival)
(2.10.0)
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/distpackages (from scikit-survival)
(2.0.3)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/distpackages (from scikit-survival)
(1.11.4)
Requirement already satisfied: scikit-learn<1.4,>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-survival)
(1.3.2)
Requirement already satisfied: qdldl in
/usr/local/lib/python3.10/dist-packages
(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-
>scikit-survival) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/distpackages (from pandas>=1.0.5-
>scikit-survival) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->scikit-survival) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn<1.4,>=1.3.0->scikitsurvival) (3.4.0)
```

```
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/distpackages (from python-
dateutil>=2.8.2->pandas>=1.0.5->scikit-survival) (1.16.0)
```

[2]:
```
import sksurv
```

[3]:
```
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()
```

[3]:

	Age_in_years	Celltype	Karnofsky_score	Months_from_Diagnosis	Prior_therapy	Treatment
0	69.0	squamous	60.0	7.0	no	standard
1	64.0	squamous	70.0	5.0	yes	standard
2	38.0	squamous	60.0	3.0	no	standard
3	63.0	squamous	60.0	9.0	yes	standard
4	65.0	squamous	70.0	11.0	yes	standard

[4]:
```
print("\nSummary Statistics:")
df.describe()
```

```
Summary Statistics:
```

[4]:

	Age_in_years	Karnofsky_score	Months_from_Diagnosis
count	137.000000	137.000000	137.000000
mean	58.306569	58.569343	8.773723
std	10.541628	20.039592	10.612141
min	34.000000	10.000000	1.000000
25%	51.000000	40.000000	3.000000
50%	62.000000	60.000000	5.000000
75%	66.000000	75.000000	11.000000
max	81.000000	99.000000	87.000000

[5]:
```
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

  Prior_therapy Treatment
0            no  standard
1           yes  standard
2            no  standard
3           yes  standard
4           yes  standard
```

```
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]
```

[6]:
```
df.Celltype.unique()
```

[6]:
```
['squamous', 'smallcell', 'adeno', 'large']
Categories (4, object): ['adeno', 'large', 'smallcell',
'squamous']
```

[7]:
```
print("Data type of data_y:", type(data_y))
```

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

[8]:
```
!pip install lifelines
```

```
Collecting lifelines
  Downloading lifelines-0.28.0-py3-none-any.whl (349 kB)
     349.2/349.2
kB 2.9 MB/s eta 0:00:00
Requirement 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/distpackages (from lifelines) (1.11.4)
Requirement already satisfied: pandas>=1.2.0 in
/usr/local/lib/python3.10/distpackages (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/distpackages (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) … done
Collecting formulaic>=0.2.2 (from lifelines)
  Downloading formulaic-1.0.1-py3-none-any.whl (94 kB)
     94.2/94.2 kB
8.8 MB/s eta 0:00:00
Requirement 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) (1.2.1)
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/distpackages (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/distpackages (from pandas>=1.2.0-
>lifelines) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/distpackages (from pandas>=1.2.0-
>lifelines) (2024.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/distpackages (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) … done
  Created wheel for autograd-gamma: filename=autograd_gamma-
  0.5.0-py3-none-
any.whl size=4030
sha256=4a4c4d67f539689e0f54a650447585e6021b66f28a235b8d6b8d8805b7
adf839
                                                Stored in directory:
        /root/.cache/pip/wheels/25/cc/e0/ef2969164144c899fedb22b3
38f6703e2b9cf46eeebf254991
Successfully built autograd-gamma
Installing collected packages: interface-meta, autograd-gamma,
formulaic, lifelines
Successfully installed autograd-gamma-0.5.0 formulaic-1.0.1
interface-meta-1.3.0 lifelines-0.28.0
```

[9]:
```
!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
Installing collected packages: Survival
Successfully installed Survival-0.0.6
```

[10]:
```
!pip show Survival
```

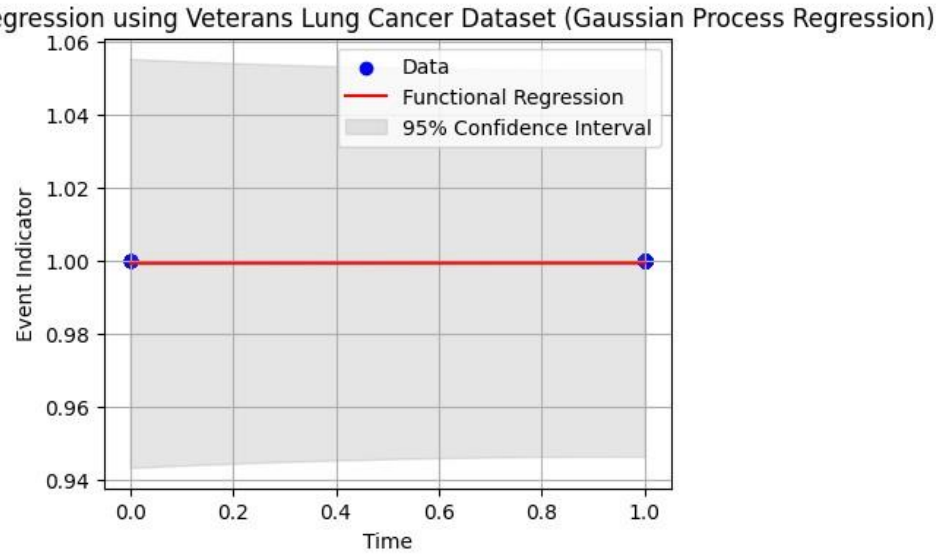
```
Name: survival
Version: 0.0.6
Summary: Add static script_dir() method to Path
Home-page: https://github.com/ryu577/survival
Author: Rohit Pandey
Author-email: rohitpandey576@gmail.com
License: MIT
Location: /usr/local/lib/python3.10/dist-
packages Requires:
Required-by:
```

[11]:
```
import survival
```

[12]:
```
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(
```

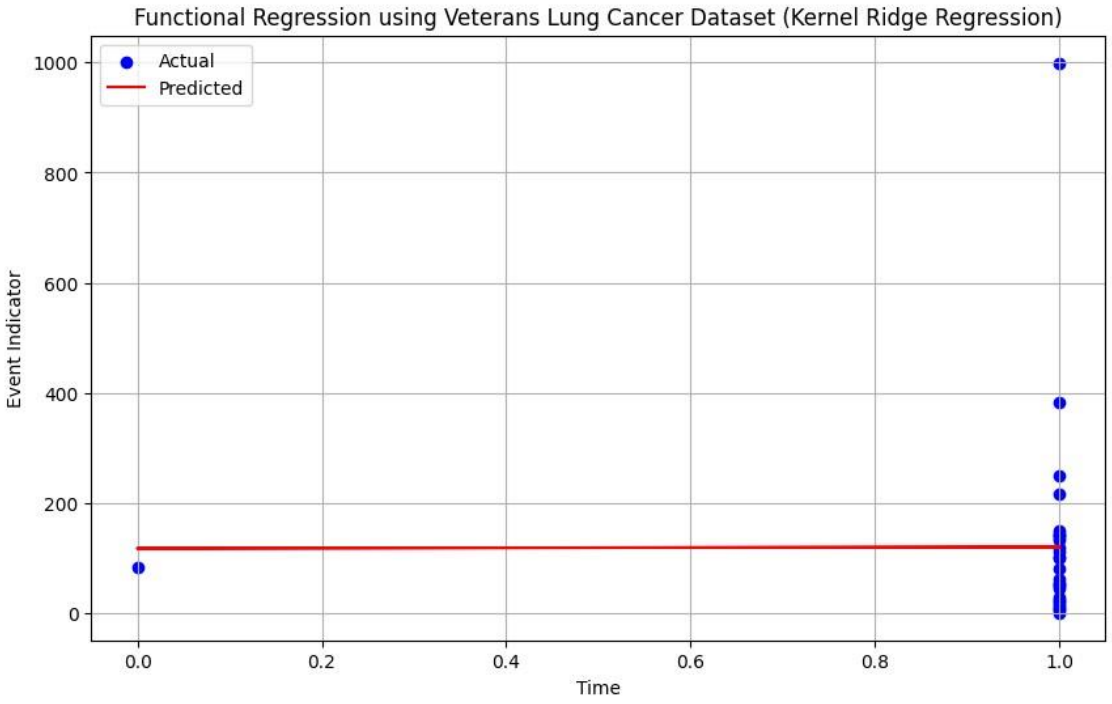


[13]:
```
import numpy as np import
matplotlib.pyplot as plt from
sksurv.datasets import
load_veterans_lung_cancer from
sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import
mean_squared_error from
sklearn.model_selection import
train_test_split

# 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)
```

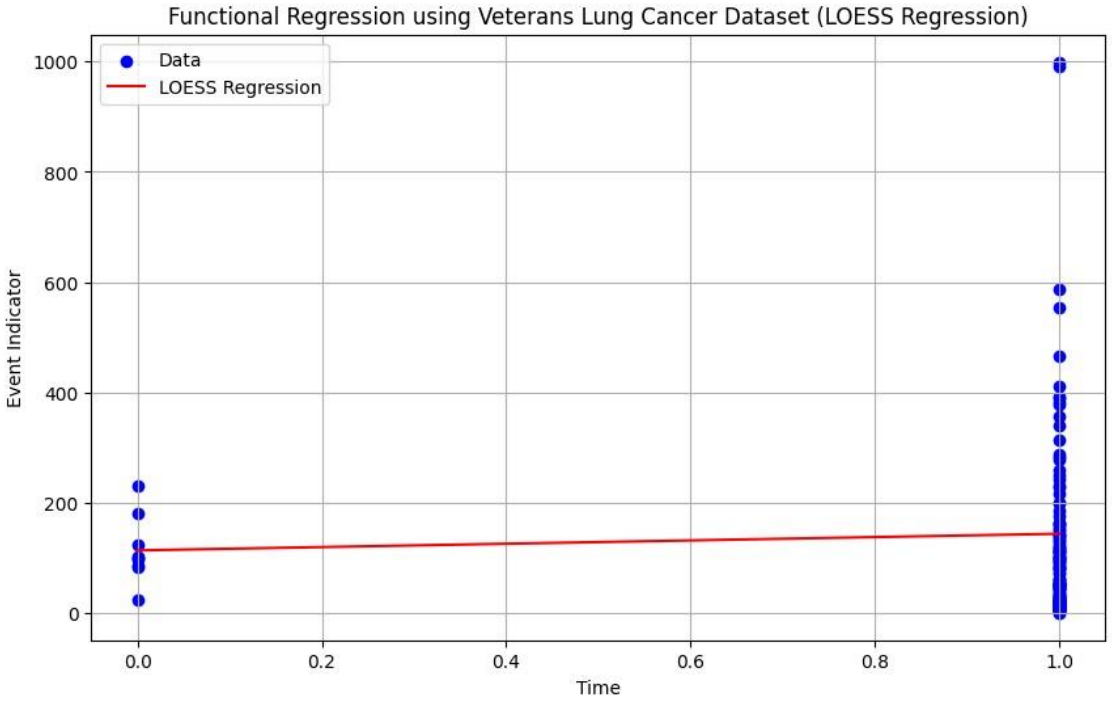
```
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# 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
```



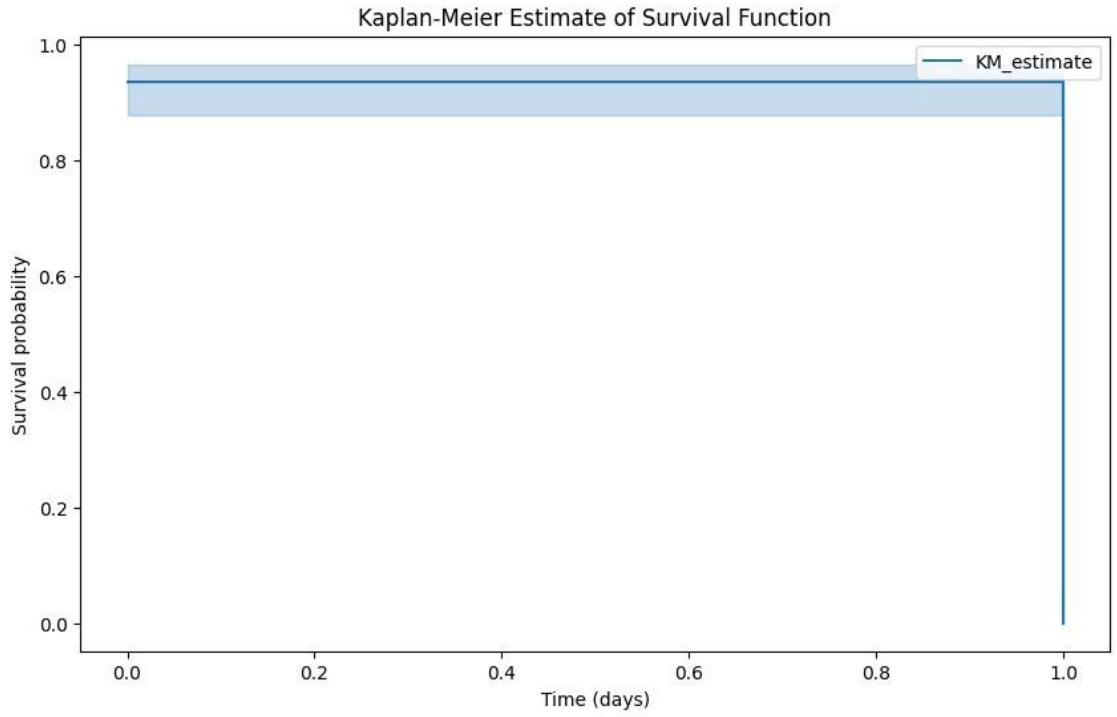
[14]:
```
import numpy as np
import matplotlib.pyplot as plt
from sksurv.datasets import load_veterans_lung_cancer
from statsmodels.nonparametric.smoothers_lowess import lowess

# 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()
```

```
/usr/local/lib/python3.10/distpackages/statsmodels/nonparametric/
smoothers_lowess.py:227: RuntimeWarning:
divide by zero encountered in divide
  res, _ = _lowess(y, x, x, np.ones_like(x),
/usr/local/lib/python3.10/distpackages/statsmodels/nonparametric/
smoothers_lowess.py:227: RuntimeWarning:
invalid value encountered in divide
  res, _ = _lowess(y, x, x, np.ones_like(x),
```



[15]:
```
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()
```



[16]:
```
# Display basic information about the dataset
print("Dataset Shape:", df.shape)
print("\nColumn Names and Data Types: ")
print(df.dtypes)
```

```
Dataset Shape: (137, 8)

Column Names and Data Types:
Feature_0  float64
Feature_1  float64
Feature_2  float64
Feature_3  float64
Feature_4  float64
Feature_5  float64
Survival      bool
Event      float64
```

```
dtype: object
```

[17]:
```
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_)
```

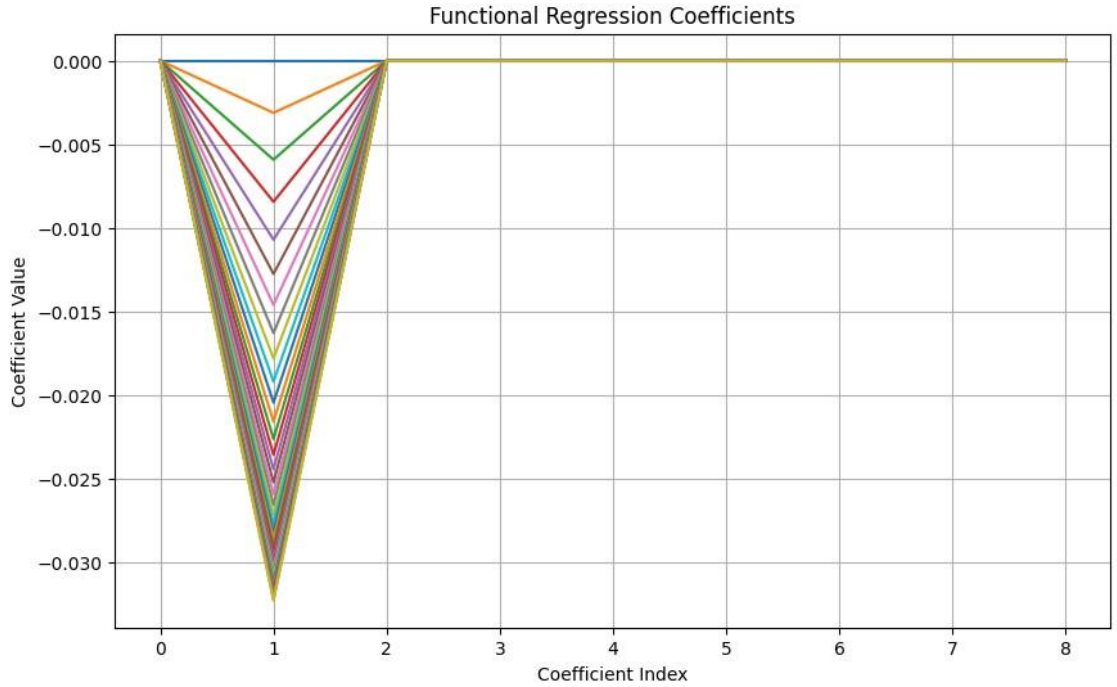
```
Coefficients:
[-8.54942361e-03 -3.26217185e-02 -9.20017173e-053.40830713e+00
 2.61963560e+003.07649447e+002.22000782e+007.23265367e-02
  2.89935879e-01]
/usr/local/lib/python3.10/dist-
packages/sklearn/preprocessing/_encoders.py:975: FutureWarning:
`sparse` was renamed to `sparse_output` in version 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:449: 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:449: 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:449: 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:449: LinAlgWarning: Ill-
conditioned matrix (rcond=7.72922e-20): result may not be
accurate. delta = solve(
```

[18]:
```
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(
```



[19]:
```
!pip install scikit-fda
```

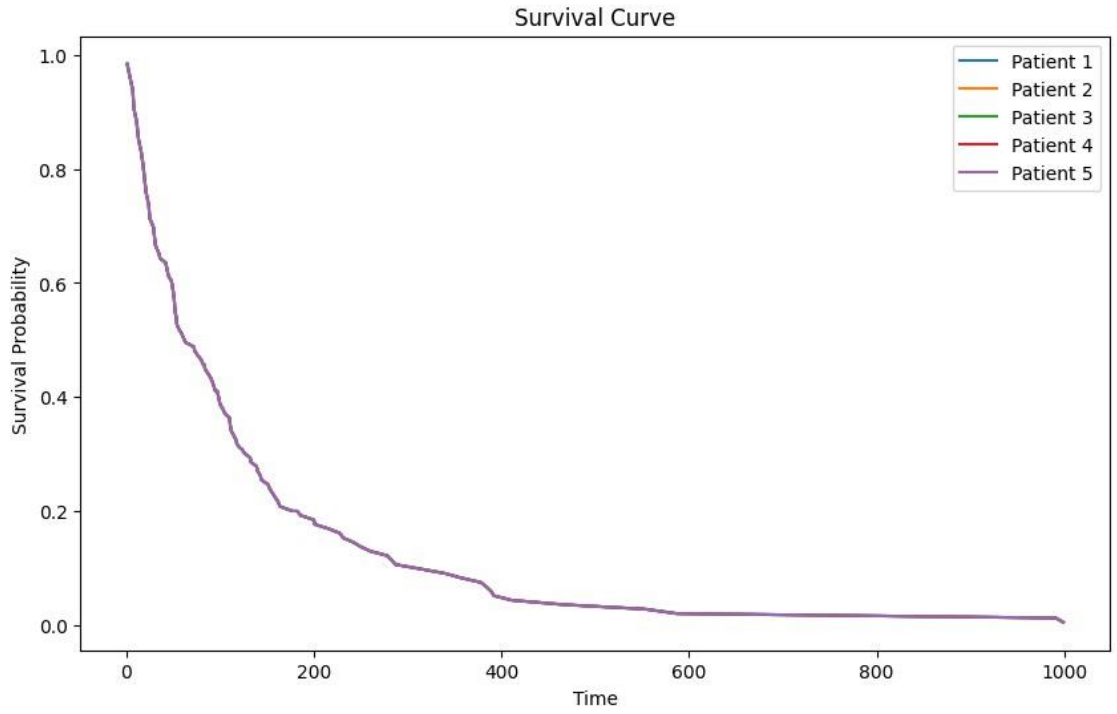
```
Collecting scikit-fda
  Downloading scikit_fda-0.9.1-py3-none-any.whl (434 kB)
     434.7/434.7
kB 7.6 MB/s eta 0:00:00
Collecting dcor (from scikit-fda)
  Downloading dcor-0.6-py3-none-any.whl (55 kB)
     55.5/55.5 kB
6.4 MB/s eta 0:00:00
Collecting fdasrsf!=2.5.7,>=2.2.0 (from scikit-fda)
  Downloading fdasrsf-2.5.10.tar.gz (4.6 MB)
     4.6/4.6 MB
68.2 MB/s eta 0:00:00
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Collecting findiff (from scikit-fda)
  Downloading findiff-0.10.0-py3-none-any.whl (33 kB)
```

```
Requirement already satisfied: lazy-loader in
/usr/local/lib/python3.10/distpackages (from scikit-fda) (0.3)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/distpackages (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/distpackages (from scikit-fda) (1.25.2)
Requirement already satisfied: pandas>=1.0 in
/usr/local/lib/python3.10/distpackages (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
6.4 MB/s eta 0:00:00
Collecting scikit-datasets[cran]>=0.1.24 (from scikit-
  fda) Downloading scikit_datasets-0.2.4-py3-none-
  any.whl (50 kB)
     50.4/50.4 kB
6.5 MB/s eta 0:00:00
Requirement 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/distpackages (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/dist-packages
(from fdasrsf!=2.5.7,>=2.2.0->scikit-fda) (4.66.2)
Requirement already satisfied: six in
/usr/local/lib/python3.10/dist-packages
(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/distpackages (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/distpackages (from pandas>=1.0->scikit-
fda) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/distpackages (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) (1.2.1) 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/distpackages (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/distpackages (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->scikit-fda) (1.3.0)
Building wheels for collected packages:
  fdasrsf Building wheel for fdasrsf
  (pyproject.toml) … done Created wheel for
  fdasrsf:
```

```
filename=fdasrsf-2.5.10-cp310-cp310-linux_x86_64.whl size=3081580
sha256=30c15167539204ffad8c8804c38065fbc22ed8863de18a02a8d2bf023e
c94dc1
  Stored in directory:
/root/.cache/pip/wheels/e8/52/1c/c4c363a070fc6643f741e1e7
ecaae39377bc19130052054270 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-0.10.0
multimethod-1.11.2 rdata-0.11.2 scikit-datasets-0.2.4 scikit-fda-
0.9.1
```

[20]:
```python
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()
```

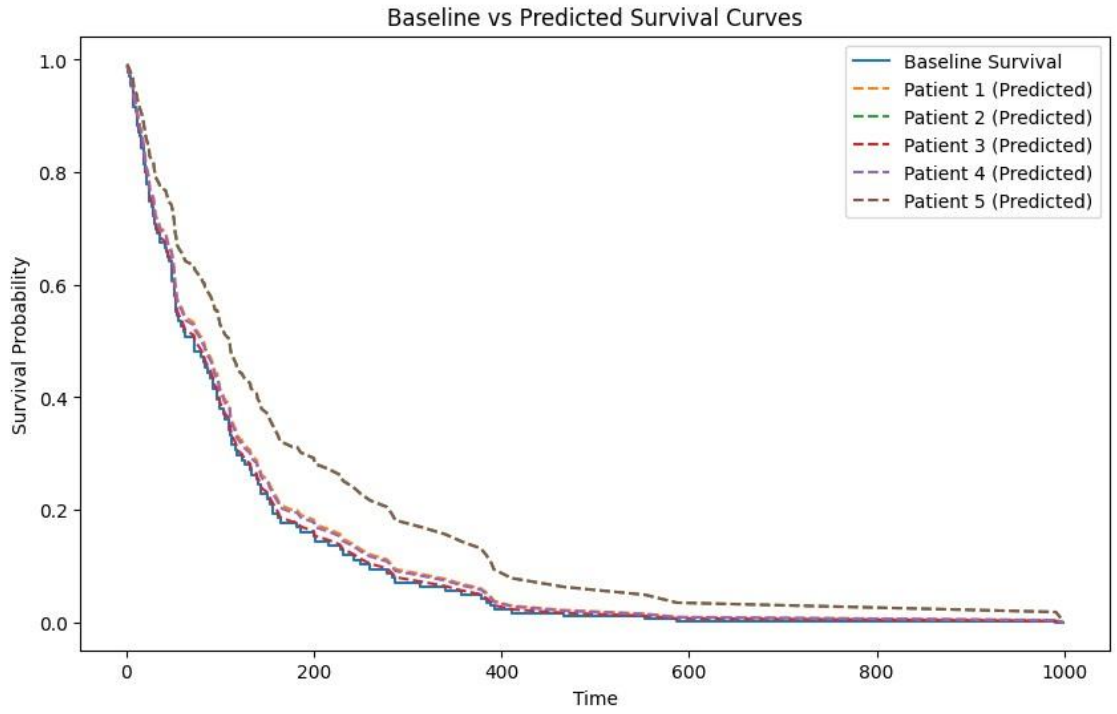


[21]:
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from
sklearn.preprocessing import StandardScaler,
OneHotEncoder from sksurv.datasets import
load_veterans_lung_cancer from sksurv.linear_model
import CoxPHSurvivalAnalysis

# 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()
```

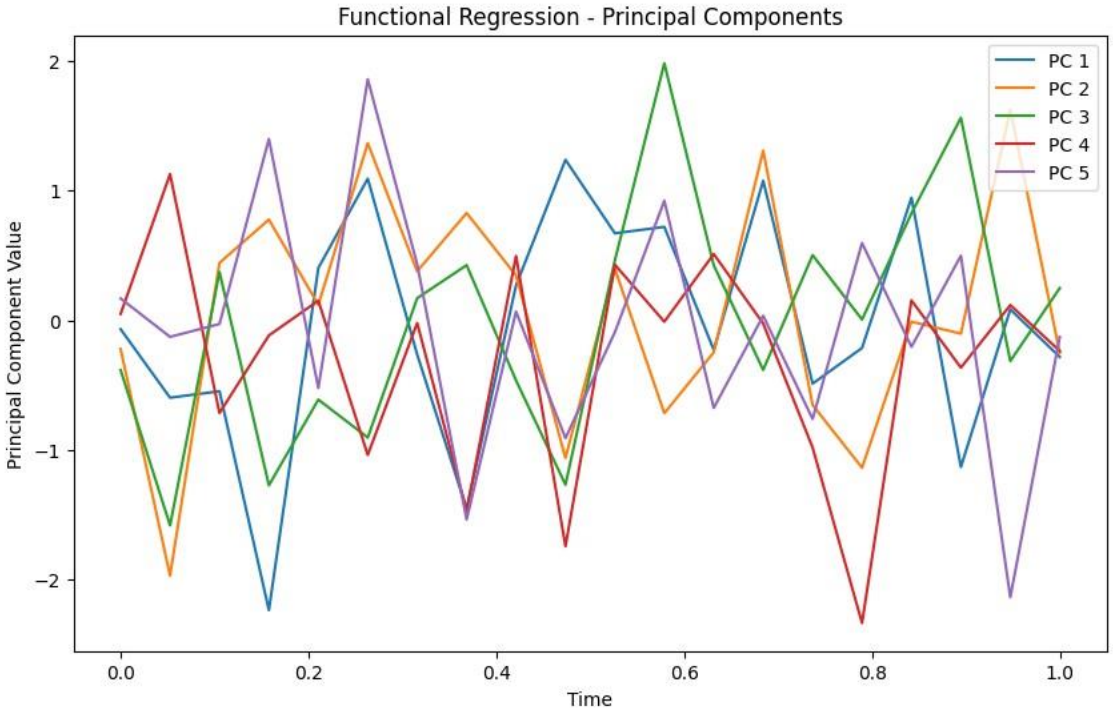
```
/usr/local/lib/python3.10/dist-
packages/sklearn/preprocessing/_encoders.py:975: FutureWarning:
`sparse` was renamed to `sparse_output` in version 1.2 and will
be removed in 1.4. `sparse_output` is ignored unless you leave
`sparse` to its default value. warnings.warn(
```

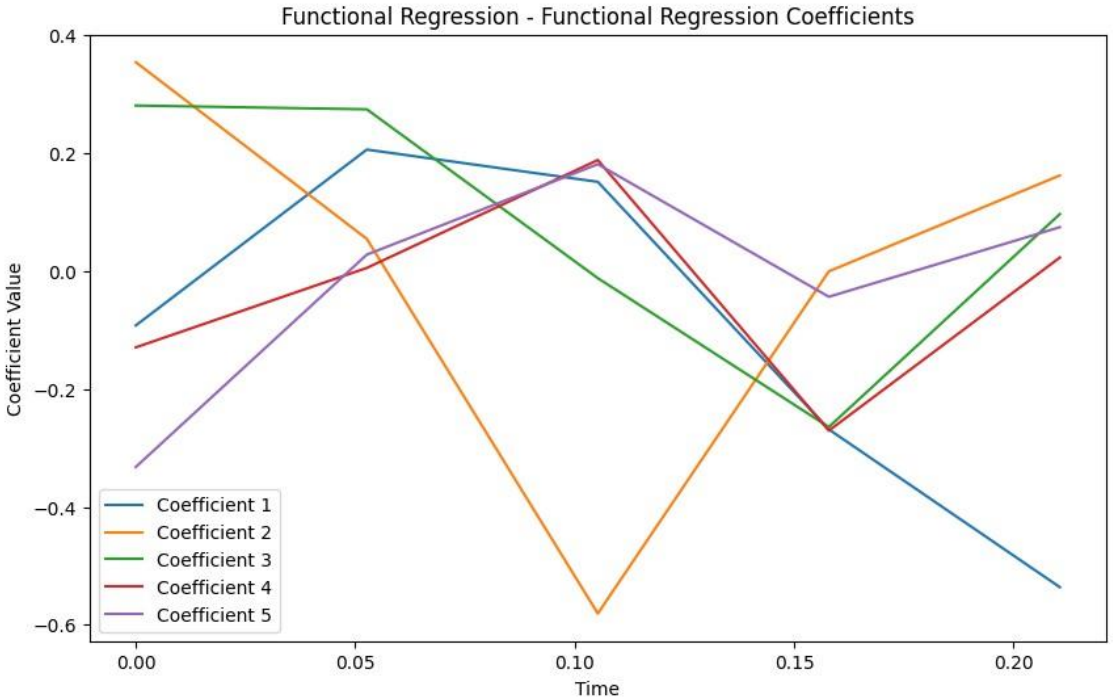


[23]:
```
import numpy as np import
matplotlib.pyplot as plt from
sksurv.datasets import
load_veterans_lung_cancer from sksurv.util
import Surv from sksurv.linear_model
import CoxPHSurvivalAnalysis from skfda
import FDataGrid
from skfda.preprocessing.dim_reduction
import FPCA

# 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) coefficients =
fpca.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()
```



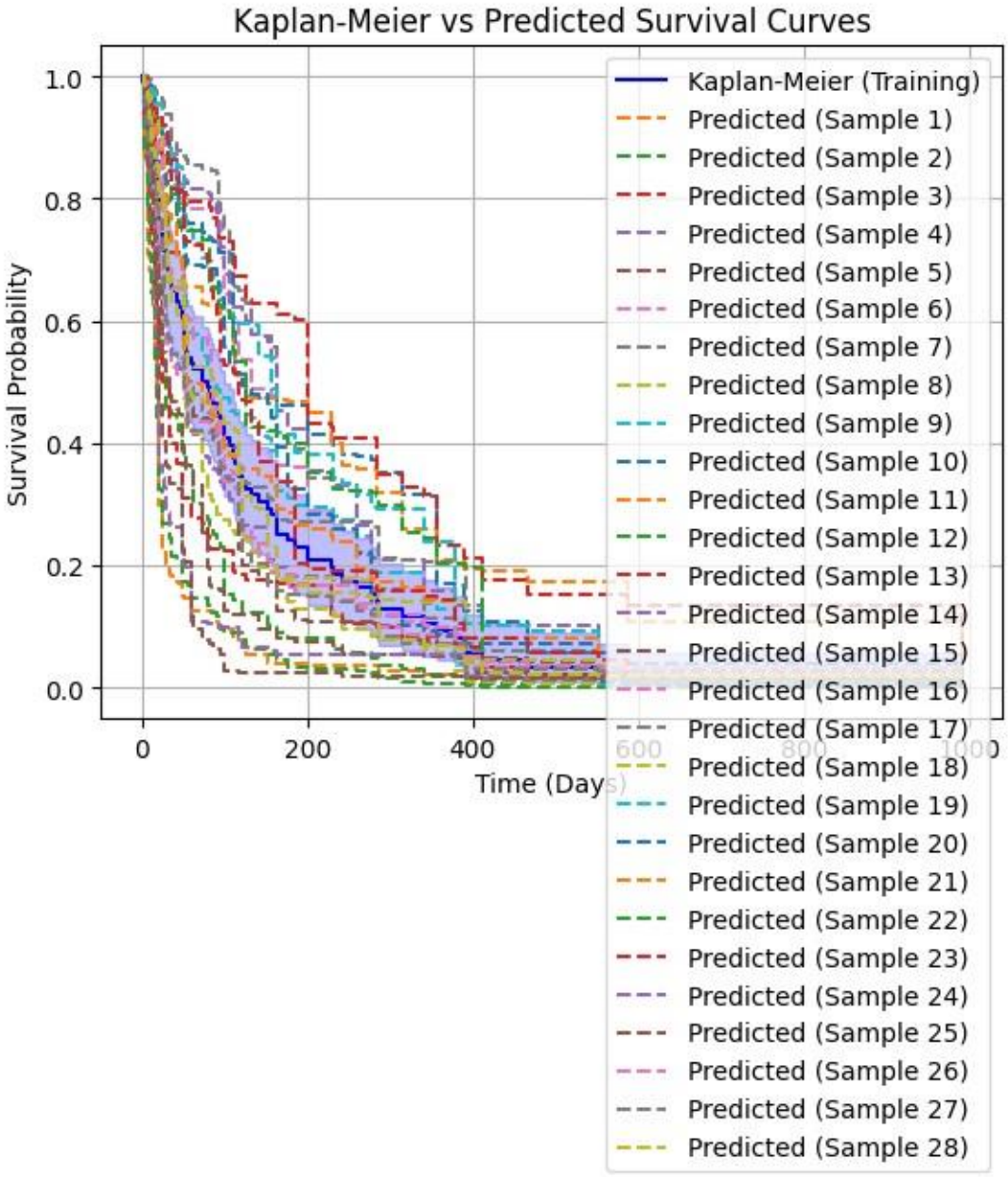


[42]:
```
import numpy as
np import pandas
as pd
```

```
from sksurv.datasets import
load_veterans_lung_cancer from
sklearn.model_selection import
train_test_split from
sklearn.preprocessing import OneHotEncoder
from sksurv.ensemble import
RandomSurvivalForest from sksurv.util
import Surv from sklearn.metrics import
mean_squared_error from lifelines import
KaplanMeierFitter import matplotlib.pyplot
as plt

# Load the dataset
data_x, data_y = load_veterans_lung_cancer()

# Convert the structured array to a pandas DataFrame
df_x =
pd.DataFrame(data_x)
df_y =
pd.DataFrame(data_y)

# Encode categorical variables
df_x_encoded = pd.get_dummies(df_x, drop_first=True)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df_x_encoded,
 df_y, ␣ ↪test_size=0.2, random_state=42)

# Convert y_train to a structured array
y_train_structured =
np.array(list(zip(y_train["Status"], ␣
 ↪y_train["Survival_in_days"])), dtype=[("Status", bool),
 ("Survival_in_days", ␣ ↪float)])

# Train the Random Survival Forest model
rsf_model = RandomSurvivalForest(n_estimators=100,
random_state=42) rsf_model.fit(X_train, y_train_structured)

# Predict survival probabilities for the test set
predicted_survival = rsf_model.predict_survival_function(X_test)
# Get the valid range of survival
times min_time =
min(predicted_survival[0].x)
max_time =
max(predicted_survival[0].x)

# Filter the survival times within the valid range
valid_survival_times = y_test["Survival_in_days"][
                      (y_test["Survival_in_days"] >= min_time) &
                                (y_test["Survival_in_days"] <= ␣
 ↪max_time)
```

```
]

# Calculate the corresponding predicted survival values
predicted_survival_values = np.array([sf(valid_survival_times)
 for sf in␣ ↪predicted_survival])

# Calculate Mean Squared Error (MSE)
mse =
mean_squared_error(valid_survival_times
, np.
 ↪mean(predicted_survival_values, axis=0))
print("Mean Squared Error (MSE):", mse)



# Plot the Kaplan-Meier survival curve for the test set
kmf = KaplanMeierFitter()
kmf.fit(y_train["Survival_in_days"],
event_observed=y_train["Status"])
kmf.plot(label="Kaplan-Meier (Training)", color='blue')

# Plot the predicted survival curves for the
test set for i in
range(len(predicted_survival)):
    plt.step(predicted_survival[i].x, predicted_survival[i].y,
            where="post", label=f"Predicted (Sample {i+1})",
            linestyle='--')

plt.title("Kaplan-Meier vs Predicted
Survival Curves") plt.xlabel("Time
(Days)") plt.ylabel("Survival
Probability") plt.legend() plt.grid(True)
plt.show()
```

```
Mean Squared Error (MSE): 16027.02924053796
```



[50]:
```
pip install scikit-survival[gp]
```

```
Requirement already satisfied: scikit-survival[gp] in
/usr/local/lib/python3.10/dist-packages (0.22.2)
WARNING: scikit-survival 0.22.2 does not provide the extra 'gp'

Requirement already satisfied: ecos in
/usr/local/lib/python3.10/dist-
packages (from scikit-survival[gp]) (2.0.13)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages
(from scikit-survival[gp]) (1.3.2)
```

```
Requirement already satisfied: numexpr in
/usr/local/lib/python3.10/distpackages (from scikit-survival[gp])
(2.10.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages
(from scikit-survival[gp]) (1.25.2)
Requirement already satisfied: osqp!=0.6.0,!=0.6.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-
survival[gp]) (0.6.2.post8) Requirement already satisfied:
pandas>=1.0.5 in /usr/local/lib/python3.10/distpackages (from
scikit-survival[gp]) (2.0.3)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/distpackages (from scikit-survival[gp])
(1.11.4)
Requirement already satisfied: scikit-learn<1.4,>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
survival[gp]) (1.3.2)
Requirement already satisfied: qdldl in
/usr/local/lib/python3.10/dist-packages
(from osqp!=0.6.0,!=0.6.1->scikit-survival[gp]) (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[gp]) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/distpackages (from pandas>=1.0.5-
>scikit-survival[gp]) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->scikit-survival[gp]) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn<1.4,>=1.3.0->scikitsurvival[gp]) (3.4.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/distpackages (from python-
dateutil>=2.8.2->pandas>=1.0.5->scikit-survival[gp]) (1.16.0)
```

[93]:
```
import numpy as np import matplotlib.pyplot as
plt from sksurv.datasets import
load_veterans_lung_cancer from sksurv.util
import Surv from sksurv.ensemble import
GradientBoostingSurvivalAnalysis from
sksurv.linear_model import
CoxnetSurvivalAnalysis from sksurv.linear_model
import CoxPHSurvivalAnalysis from sksurv.svm
import FastSurvivalSVM from
sklearn.preprocessing import StandardScaler from
sksurv.preprocessing import OneHotEncoder from
sklearn.metrics import mean_squared_error
```

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

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

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

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

# One-hot encode categorical variables if there are any
encoded_categorical =
None if not
categorical_cols.empty
: encoder =
OneHotEncoder()
    encoded_categorical =
    encoder.fit_transform(df_x[categorical_cols])

# Standardize numerical
variables scaler =
StandardScaler()
scaled_numerical = scaler.fit_transform(df_x[numerical_cols])

# Combine encoded categorical variables and scaled
numerical variables if encoded_categorical is not None:
    X = np.concatenate([encoded_categorical,
scaled_numerical], axis=1) else:
    X = scaled_numerical

# Fit and evaluate each
model models = {
    "GBST": GradientBoostingSurvivalAnalysis(),
    "FLM": CoxnetSurvivalAnalysis(fit_baseline_model=True),
    "Splines": CoxPHSurvivalAnalysis(),
    #"Neural Network": FastSurvivalSVM()
}

mse_resul

ts = {}

for name, model in models.items():
    model.fit(X, y)
```

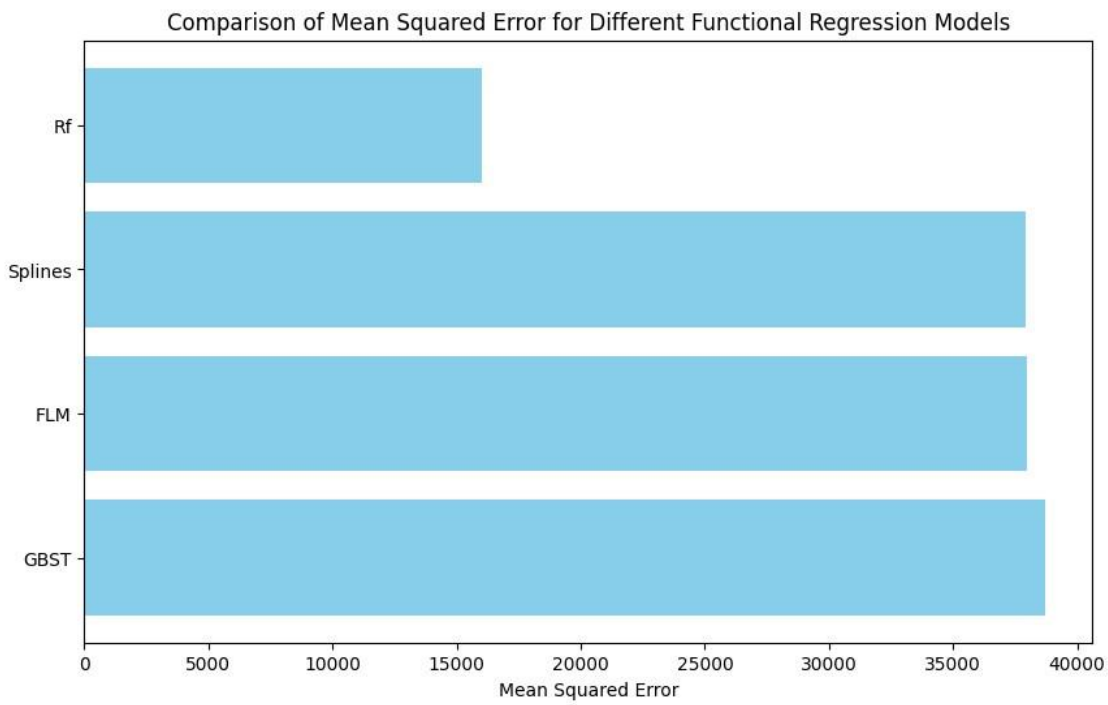
```
    predicted_survival = model.predict_survival_function(X)
    y_true = np.array([entry[1] for entry in data_y])
    predicted_survival_sum = np.zeros_like(y_true) #
    Initialize sum array for sf in predicted_survival:
        survival_at_times = sf(np.array(y_true))
        predicted_survival_sum += survival_at_times # Add
        survival␣
 ↪probabilities at observed times
    print("Data type of y_true:",
    type(y_true)) print("Shape of
    y_true:", y_true.shape)
    print("Data type of predicted_survival_sum:",
    type(predicted_survival_sum)) print("Shape of
    predicted_survival_sum:", predicted_survival_sum.shape)
    print("y_true values:", y_true)
    print("predicted_survival_sum values:",
    predicted_survival_sum)
    mse = mean_squared_error(y_true, predicted_survival_sum)
    mse_results[name] = mse

# Plot MSE results
Print(mse_results)
mse_results["Rf"]=16027.02924053796 plt.figure(figsize=(10, 6))
plt.barh(list(mse_results.keys()), list(mse_results.values()),
color='skyblue') plt.xlabel('Mean Squared Error')
plt.title('Comparison of Mean Squared Error for Different
Functional Regression␣
 ↪Models')
plt.show()
```

```
Data type of y_true: <class 'numpy.ndarray'>
Shape of y_true: (137,)
Data type of predicted_survival_sum: <class 'numpy.ndarray'>
Shape of predicted_survival_sum: (137,) y_true values: [ 72. 411.
228. 126. 118.   10.   82. 110. 314. 100.    42.   8. 144.
25.
  11.  30. 384.   4.    54.   13. 123.   97. 153.   59. 117.
       16. 151.   22. 56.    21.   18. 139.   20.   31.   52.
  287. 18.   51. 122.   27.   54.   7. 63. 392.      10.   8.
       92.   35. 117. 132.   12. 162.   3.    95. 177. 162.
216. 553. 278.12. 260. 200. 156. 182. 143. 105. 103. 250. 100.
999.
 112.  87. 231. 242. 991. 111.1. 587. 389. 33.25.   357.   467.
                                                    201.
   1.  30. 44. 283. 15. 25. 103. 21.  13.   87.2.   20.   7. 24.
  99.  8.   99.  61.  25.  95.  80.  51.  29.   24.18. 83.  31.  51.
  90.  52.  73.   8.  36.  48.  7. 140. 186.   84.19. 45.  80.  52.
 164.  19.  53.  15.     43. 340. 133. 111. 231.49.]
       378.
predicted_survival_sum values: [ 65.798579464.0066405818.88847606
  39.0016517941.26042557
```

```
 121.97967322 61.50980063 47.98969634 10.53090066 51.4828861
  86.31922019 124.1685946   32.24858853   97.26644626
  120.79117971 92.81635089  6.75929376 131.79914013
       71.09992985 116.37866202
 40.16130337 54.86689485 29.93557014 69.0038813 42.36336646
113.2232755 231.10501344 102.6119602 70.04535601 103.72556023
 110.19111758   35.5788814 105.92172222   90.66378106
       74.31332557 11.51824833 110.19111758  77.62233639
       40.16130337    96.13927526 71.09992985 128.35203038
       66.87832316    4.95115676 121.97967322 124.1685946
       57.04481174    88.52310198    42.36336646
       37.8405647
 118.54368122   26.53759713 132.8673672   54.86689485
       24.30480368 26.53759713   19.92752285    2.32330836
       13.58703392 118.54368122 14.61324992  22.08069962
       28.77915132    24.30480368    33.34387778 49.15156794
       50.31753201    15.66724569    51.4828861 0.22974974
 44.58402474 59.2890558  17.8384808 16.75460906  0.93241593
45.72168186 135.12461601 1.63763459  5.80094871 89.6096688
 97.26644626  8.56731524  3.0538848 20.99317416 135.12461601
 92.81635089 84.12914942 12.5256382 114.2686955 97.26644626
50.31753201 103.72556023 116.3786620 259.2890558 133.9873794
105.92172222 128.35203038 100.4444865 152.62691951 124.1685946
 52.62691951 67.94720387 97.2664462 54.86689485 62.58636304
  77.62233639   95.02899726 100.44448651 110.19111758
       61.50980063 90.66378106   77.62233639    58.15934263
       74.31332557    64.7221709
 124.1685946 87.4127697 781.98332016 128.3520303 834.45741049
 23.1792071 960.39510044 108.0627563 983.0490724 862.58636304
 74.3133255 725.41364767 108.0627563 973.22315991 114.26869558
 85.22005859  9.54336422 36.70778507 45.7216818 617.83848086
  7.670341 9180.90249175]
 Data type of y_true: <class 'numpy.ndarray'>
 Shape of y_true: (137,)
 Data type of predicted_survival_sum: <class 'numpy.ndarray'>
 Shape of predicted_survival_sum: (137,) y_true values: [ 72. 411.
 228. 126. 118.   10.   82. 110. 314. 100.   42.   8. 144.
 25.
  11. 30. 384.   4.   54.  13. 123.   97. 153.   59. 117.
       16. 151.   22. 56.   21.  18. 139.   20.   31.  52.
  287. 18.  51. 122.   27.  54.  7. 63. 392.      10.  8.
       92.  35. 117. 132.   12. 162.  3.   95. 177. 162.
 216. 553. 278.12. 260. 200. 156. 182. 143. 105. 103. 250. 100.
999.
 112. 87. 231. 242. 991. 111.1. 587. 389. 33.25.  357.  467.
                                                  201.
  1. 30. 44. 283. 15. 25. 103. 21. 13.   87.2.  20.  7. 24.
 99. 8.  99. 61. 25. 95. 80. 51. 29.   24.18. 83. 31. 51.
 90. 52. 73.  8. 36. 48. 7. 140. 186.  84.19. 45. 80. 52.
 164. 19. 53. 15.    43. 340. 133. 111. 231.49.]
       378.
```

```
 predicted_survival_sum values: [ 67.598275118.8272666225.39189301
   42.77503261 44.72705581
    123.0282694363.3667454450.6721146916.9380333753.80678002
     88.20184791 125.04456102  37.01119027     99.08944461
     122.00076539 94.67152568  12.46381677 132.01319197
          72.95204032 117.95345634
     43.7703897256.9382552235.0346902370.8198268645.69255926
    114.9121597636.03042602 104.4516296571.87897984 105.54745209
     111.87158073    39.83429018 107.68131688    92.5075618
          76.23178388 18.00705407 111.87158073  79.5521499
          43.77038972     97.97626979
    72.95204032 129.0291101268.6692176610.22952624 123.02826943
    125.0445610259.04241274 90.3599929  45.6925592641.78084924
     119.97427005    32.09502685 133.0191266    56.93825522
          30.18703379 32.09502685    26.35264018     6.08738245
          20.13432995 119.97427005 21.19686157  28.23502019
          34.03701244     30.18703379     37.93723665 51.70761513
          52.75877182     22.28090208     53.80678002
          0.82977909 47.65135168     61.20016018     24.42099902
          23.37096018     2.94877935
    48.65922792 135.0218437  4.7122694511.3182167391.44006212
     99.0894446114.67363602  7.3989402327.28612988 135.0218437
     94.6715256886.0748613919.05286325 115.9326323999.08944461
    52.75877182 105.54745209 117.9534563461.20016018 134.02053599
     107.68131688 129.02911012 102.29128986 54.84114467
      125.04456102 54.84114467 69.73890734 99.08944461 56.93825522
      64.4394554
     79.552149996.87048379 102.29128986 111.8715807363.36674544
     92.5075618  79.5521499  60.1188497376.2317838866.53344293
     125.04456102    89.27573424     83.96303919 129.02911012
          38.88163235 29.19939516    62.27565256 109.78190221
          85.02136478     64.4394554
     76.2317838831.13723272 109.7819022175.12625674 115.93263239
     87.1350990715.7919587940.8015062748.6592792224.42099902
     13.5546987182.85759744]
    Data type of y_true: <class 'numpy.ndarray'>
    Shape of y_true: (137,)
    Data type of predicted_survival_sum: <class 'numpy.ndarray'>
    Shape of predicted_survival_sum: (137,) y_true values: [ 72. 411.
    228. 126. 118.   10.   82. 110. 314. 100.    42.   8. 144.
    25.
      11.  30. 384.   4.    54.   13. 123.   97. 153.   59. 117.
          16. 151.   22. 56.    21.   18. 139.   20.   31.   52.
     287.18.   51. 122.   27.   54.   7. 63. 392.      10.   8.
          92.   35. 117. 132.    12. 162.   3.    95. 177. 162.
    216. 553. 278.12. 260. 200. 156. 182. 143. 105. 103. 250. 100.
999.
     112.  87. 231. 242. 991. 111.1. 587. 389. 33.25.   357.   467.
                                                  201.
      1.  30. 44. 283. 15. 25. 103. 21. 13.   87.2.  20.   7. 24.
```

```
       99.  8.  99.  61.  25.  95.  80.  51.  29.   24.18. 83.  31.  51.
       90.  52.  73.   8.  36.  48.  7. 140. 186.   84.19. 45.  80.  52.
      164.  19.  53.  15.     43. 340. 133. 111. 231.49.]
           378.
  predicted_survival_sum values: [ 67.650883069.0290373325.65607379
    42.94544482 44.88560479
      123.02784049    63.43252783     50.79466574     17.20519687
           53.91291114 88.21394947 125.04384485  37.21707927
           99.09213119 121.9999753
      94.6741502312.70500633 132.0128162672.99211203 117.95383718
      43.9352359457.02873261 35.2501176470.8649488345.84496003
     114.9152550436.24113013 104.4542487671.92146169 105.55104689
     111.8775557240.02176111 107.6863728292.512448876.26762338
      18.2752528111.8775557279.5857770243.9352359497.97842405
     72.99211203 129.0277741968.7193289610.45061674 123.02784049
      125.04384485    59.12408516     90.36824675     45.84496003
           41.95687215 119.97352486   32.32446122 133.01919163
           57.02873261     30.4269938
       32.32446122   26.61350329   6.25486697   20.40306087
       119.97352486  21.46543268  28.4860712  34.25705001
       30.4269938   38.13690975   51.82461843   52.87050228
       22.54891998   53.91291114   0.86398701   47.79136778
       61.27342212   24.68718164   23.63801673   3.05760883
       48.79312494  135.02221738   4.85850867  11.54867755
       91.44670071   99.09213119   14.92758198   7.58175608
       27.54227727  135.02221738  94.67415023  86.0925574
       19.3213186 115.93484325 99.09213119
       52.87050228  105.55104689  117.95383718  61.27342212
      134.02083372  107.68637282  129.02777419 102.29323937
      54.94158099  125.04384485  54.94158099  69.78626481
      99.09213119   57.02873261   64.50182826   79.58577702
      96.87252692 102.29323937 111.87755572 63.43252783
       92.5124488  79.5857770260.1964052376.2676233866.58895829
     125.0438448589.2857118883.98756933 129.0277741939.0751287
      29.4448858 62.34498308 109.7879186485.0423411 64.50182826
      76.2676233831.37186624 109.7879186475.16315265 115.93484325
      87.1499139116.0521858340.9831358648.7931249424.68718164
      13.8018096282.8851887 ]
{'GBST': 38690.41494578108, 'FLM': 37972.84919581442, 'Splines':
37939.010800408025, 'Rf': 16027.02924053796}
```


Comparison of Mean Squared Error for Different Functional Regression Models
Rf
Splines
FLM
GBST
0
5000
10000
15000
20000
25000
30000
35000
40000
Mean Squared Error