System-Dokumentation

Inhalt

1 Einführung

Ziel der Anwendung ist es, dem Anwender eine KI-gestützte Insolvenzprognose zu ermöglichen, die so konfigurierbar und erklärbar wie möglich ist. Sie bietet eine Reihe von Funktionen, die in diesem Rahmen notwendig sind. Dazu gehören Daten-Upload, Daten-Übersicht, Daten-Bereinigung, Feature-Engineering, Daten-Segmentierung, Modell-Setup und -Optimierung sowie Modell-Evaluierung.

2 Framework

Die Wahl von Python als Programmiersprache für die Implementierung des Dashboards wurde aus mehreren Gründen getroffen. Python ist bekannt für seine Vielseitigkeit und seine umfangreiche Standardbibliothek, die eine breite Palette von Anwendungsfällen abdeckt. Darüber hinaus hat sich Python aufgrund der verfügbaren Bibliotheken für maschinelles Lernen und Datenanalyse als häufig eingesetzte Programmiersprache für KI etabliert.

In der vorliegenden Anwendung wurde außerdem das Web-Framework Django speziell wegen seiner robusten und flexiblen Architektur ausgewählt. Die in Django integrierte SQLite-Datenbank ist für die Entwicklung der Anwendung praktisch, da direkt mit der Datenpersistenz begonnen werden kann. Darüber hinaus bietet Django ein integriertes Admin-Panel, das eine intuitive Benutzeroberfläche für die Administration der Anwendung bereitstellt. Diese Features erleichtern nicht nur die Datenmanipulation und -verwaltung, sondern tragen auch zur Steigerung der Benutzerfreundlichkeit und zur Verkürzung der Entwicklungszeit bei. In diesem Zusammenhang erweist sich Django als besonders geeignet, da es ein Gleichgewicht zwischen Funktionalität und Benutzerfreundlichkeit schafft und gleichzeitig die Anforderungen an Sicherheit und Datenintegrität erfüllt.

3 Module

3.1 James

Das JAMes-Projekt, entwickelt unter Verwendung des Django-Frameworks Version 4.0.6, stellt eine Webanwendung dar, die verschiedene Module für spezifische Funktionalitäten integriert. Im technischen Aufbau folgt es den Django-Konventionen.

Beginnend mit der settings.py-Datei, die als das Herzstück der Anwendung dient, werden hier grundlegende Einstellungen vorgenommen. Zu den wichtigsten gehören die Liste der installierten Apps (INSTALLED_APPS), welche die unterschiedlichen Module repräsentieren, und die Middleware-Konfiguration (MIDDLEWARE), die den Ablauf der Request- und Response-Verarbeitung steuert. Die Datenbank ist mit SQLite konfiguriert, wie in der DATABASES-Einstellung ersichtlich ist.

Die URL-Routen werden in der urls.py-Datei definiert. Diese Datei leitet URL-Anfragen an die entsprechenden Views weiter, die in den verschiedenen Modulen implementiert sind. Die Verwendung von include() für die URLs ermöglicht es, dass jedes Modul seine eigenen URL-Routen haben kann, was die Modularität und die Wartbarkeit des Systems erhöht.

Für die Authentifizierung und Autorisierung verwendet JAMes die eingebaute auth-Applikation von Django, ergänzt durch ein benutzerdefiniertes usermanagement-Modul. Die Authentifizierung erfolgt über Formulare und Passwort-Validatoren, die in AUTH_PASSWORD_VALIDATORS definiert sind.

Die Middleware ist so konfiguriert, dass sie verschiedene sicherheits- und funktionsrelevante Aufgaben ausführt, darunter CSRF-Schutz und Session-Management. Diese Middleware-Schichten werden in der Reihenfolge ausgeführt, in der sie in der MIDDLEWARE-Liste angegeben sind.

Die settings.py des JAMes-Moduls enthält spezifische Einstellungen für Internationalisierung, darunter LANGUAGE_CODE auf 'en-us' und TIME_ZONE auf 'Europe/Berlin'. Diese Konfigurationen erleichtern die Lokalisierung der Anwendung für verschiedene Sprachen und Zeitzonen.

Die statischen und medialen Dateien werden im System über die in der settings.py-Datei festgelegten STATIC_URL, MEDIA_ROOT und MEDIA_URL verwaltet. Diese Einstellungen sind für das korrekte Laden von CSS, JavaScript und hochgeladenen Medien erforderlich.

3.2 Users

Das "Users"-Modul sorgt für die Schaffung einer einheitlichen und benutzerfreundlichen Erfahrung. Es implementiert die grundlegenden strukturellen Elemente und Navigationsmechanismen, die in der gesamten Anwendung wiederkehrend sind. Im Zentrum dieses Moduls stehen verschiedene HTML-Templates, die jeweils spezifische Aufgaben erfüllen und in den anderen Modulen wiederverwendet werden.

Das base.html-Template ist das Grundgerüst der Anwendung und bietet eine strukturierte Umgebung für die Darstellung der Inhalte. Es enthält die grundlegenden HTML-Elemente wie Header und Footer sowie den Import von Stylesheets und JavaScript-Bibliotheken. Dieses Grundtemplate dient als universelle Vorlage, auf der die anderen Templates der Anwendung aufbauen.

Ein weiteres Schlüsselelement ist das navbar.html-Template, das die Navigationsleiste der Anwendung darstellt. Es ist so konzipiert, dass es in alle anderen Module und Templates der Anwendung integriert werden kann. Durch die Verwendung des {% extend %}-Tags von Django können andere Templates diese Navigationsleiste problemlos erweitern. Dies stellt sicher, dass die Navigationsleiste über alle Module hinweg konsistent ist, was wiederum die Benutzererfahrung verbessert.

Die Templates homepage.html und about.html dienen als Einstiegspunkte in die Anwendung und bieten allgemeine Informationen. Sie sind in der Regel die ersten Anlaufstellen für neue oder nicht angemeldete Benutzer.

Das Users-Modul stellt somit eine zentrale Rolle in der Architektur der Anwendung dar und trägt zur Einheitlichkeit und Benutzerfreundlichkeit der gesamten Applikation bei.

3.3 Usermanagement

Das Modul Usermanagement stellt Funktionen zur Verwaltung der persönlichen Benutzerdaten zur Verfügung. Technisch gesehen sind zwei Hauptansichten definiert: personal_dashboard und personal_upload, beide mit dem Dekorator @login_required versehen, was bedeutet, dass nur authentifizierte Benutzer Zugriff auf diese Funktionen haben.

In der Ansicht personal_dashboard wird eine Abfrage auf das CSVFile-Modell ausgeführt, um alle gespeicherten CSV-Dateien sowie Modelle des Typs XGModel zu erhalten. Diese Daten werden dann im Kontext der Template-Rendering-Funktion übergeben, so dass sie im dazugehörigen HTML-Template dargestellt werden können.

Die Ansicht personal_upload implementiert die Upload-Funktionalität für CSV-Dateien. Die Methode prüft, ob es sich um eine HTTP POST-Anfrage handelt. In diesem Fall werden die entsprechenden Datei- und Benutzerinformationen extrahiert und ein neues CSVFile-Objekt wird über das Django ORM erstellt. Wenn Dateien erfolgreich hochgeladen werden, wird eine HTTP-Weiterleitung zur data_overview-Ansicht aus den nächsten Modul ausgelöst.

Das zugrunde liegende Datenmodell CSVFile speichert Informationen über die hochgeladenen CSV-Dateien, einschließlich der zugehörigen Benutzer und des Zeitpunkts des Uploads. Das Modell verwendet das Django-eigene models.Model als Basisklasse und definiert verschiedene Felder wie superuser, user, uploaded_at und file zur Speicherung relevanter Daten.

Die Authentifizierungsmechanismen für das Einloggen und Ausloggen sind ebenfalls in diesem Modul integriert, wobei Django's eingebaute Authentifizierungsfunktionen und -Dekoratoren verwendet werden.

3.4 Process Documentation

Das Modul Process Documentation dient in erster Linie als Informationsressource innerhalb der Anwendung und bietet dem Benutzer eine schrittweise Anleitung für die Modellentwicklung. Technisch betrachtet besteht aus einer einzigen Ansicht und einem zugehörigen HTML-Template.

Die Ansicht process_documentation ist in der Datei views.py definiert und verwendet Djangos render-Methode, um das Template process_documentation.html darzustellen. Diese Ansicht erfordert keine Zustandsinformationen und ist daher direkt zugänglich.

Der Hauptinhalt des Templates ist eine geordnete Liste, die die verschiedenen Schritte des Modellentwicklungsprozesses beschreibt. Die Beschreibung führt direkt zum Modellerstellungsprozess.

3.5 Data Overview

Das Modul Data Overview hat die Aufgabe, dem Benutzer eine detaillierte Übersicht über die hochgeladenen Datensätze zu bieten. Technisch setzt es auf drei Hauptansichten: data_overview, results und results_csv.

Die Ansicht data_overview dient als Einstiegspunkt. Sie rendert lediglich ein HTML-Template, das ein Formular enthält, in dem der Benutzername eingegeben werden kann.

Die results-Ansicht verarbeitet die durch das Formular übermittelten Informationen. Sie filtert die Datensätze im CSVFile-Modell anhand des übergebenen Benutzernamens. Die gefilterten Datensätze werden dann an das zugehörige HTML-Template weitergeleitet, wo sie in einer Tabelle dargestellt werden.

In der dritten Ansicht, results_csv, werden mehrere Operationen ausgeführt. Zunächst werden der Dateiname und der aktuelle Benutzer aus den GET-Parametern extrahiert. Dann wird die entsprechende CSV-Datei geladen und mittels der Bibliothek Pandas analysiert. Verschiedene statistische Kennzahlen und Datenqualitätsmetriken werden berechnet und dann an das HTML-Template weitergegeben, wo sie in unterschiedlichen Tabellenformaten dargestellt werden.

Die Integration von Pandas ermöglicht hier eine effiziente Datenanalyse direkt innerhalb der Django-Anwendung. Außerdem wird durch die Verwendung des cache_page-Dekorators die Belastung der Datenbank minimiert, da die aufwendigen Berechnungen nur einmal alle 15 Minuten durchgeführt werden. Dies ermöglicht es dem Benutzer, ohne lange Wartezeiten im Modellerstellungsprozess vor und zurück zu navigieren.

3.6 Feature Selection

Das Modul "Feature Selection" setzt auf der Basis eines bereits gereinigten Datensatzes auf. Der Prozess des Feature-Engineerings und der Feature-Auswahl wird in mehreren Django-Ansichten umgesetzt.

Beginnend mit der Ansicht datacleaning, dient diese lediglich als Einstiegspunkt und weist den Benutzer darauf hin, dass ein Cleaning-Prozess stattfinden wird. Das eigentliche Cleaning erfolgt in der cleaned_data-Ansicht, die auf der cleaning-Funktion aus der cleaning.py-Datei aufbaut. Diese Funktion bereitet den Datensatz auf mehreren Ebenen vor. Sie fügt beispielsweise neue Variablen wie *Anlagenintensitaet* und *Umlaufintensitaet* hinzu und bereinigt den Datensatz von negativen Werten

und fehlenden Beobachtungen durch die Methoden remove_missing und remove_negative_values aus der functions.py-Datei.

Nach dem Reinigungsprozess folgt die featureselection-Ansicht, die das Herzstück des Moduls darstellt. Hier wird dem Benutzer ein Formular präsentiert, das es ihm ermöglicht, spezifische Features für die weitere Analyse auszuwählen. Bei der Formularübermittlung wird die Validität der Eingaben geprüft. So wird zum Beispiel sichergestellt, dass mindestens drei Features ausgewählt wurden. Nach der Validierung wird die Funktion feature_calc aufgerufen. Diese Funktion nimmt die ausgewählten Features und berechnet die zugehörigen Metriken für den Datensatz.

Ein neuer CSV-Dateiname wird erstellt, und die Datei mit den berechneten Features wird im Modell Feature_Table gespeichert. Es erfolgt dann eine Weiterleitung zur featurestat-Ansicht.

In der featurestat-Ansicht werden schließlich statistische Analysen der ausgewählten Features durchgeführt. Die Ansicht verwendet Pandas, um verschiedene statistische Metriken zu berechnen und sie in einem ansprechenden Format darzustellen.

Es ist wichtig zu bemerken, dass auch hier Caching-Strategien angewendet werden. Die cleaned_data und featurestat Ansichten sind mit einem Cache-Dekorator versehen, der die Daten für 15 Minuten speichert, um die Performance zu verbessern und die Serverlast zu reduzieren.

3.7 Group Preprocessing

Das Modul "Group Preprocessing" dient der Segmentierung eines Datensatzes in verschiedene Gruppen basierend auf festgelegten Grenzwerten. Das Modul ist in der views.py-Datei implementiert und verwendet Formulare und Modelle, die in den Dateien forms.py und models.py definiert sind.

Im Backend erfolgt der Prozess in mehreren Schritten, beginnend mit der Funktion group_processing(request). Diese Funktion sammelt Informationen wie den Dateinamen und den aktuellen Benutzer und rendert die anfängliche HTML-Seite group_processing.html.

Die Funktion presegmentation(request) ist der nächste Schritt im Prozess. Sie nimmt die Anzahl der zu bildenden Gruppen als Eingabe und leitet den Benutzer zur Seite segmentation weiter, wo die eigentliche Segmentierung stattfindet.

Die Hauptlogik der Segmentierung ist in der Funktion segmentation(request) enthalten. Diese Funktion stellt ein Formular bereit, das mit der Klasse GroupForm aus forms.py verknüpft ist. Benutzer können die Anzahl der Gruppen, die Gruppennamen und die unteren und oberen Grenzwerte für jede Gruppe festlegen. Dabei werden mehrere Validierungsregeln angewendet, beispielsweise ob die untere Grenze kleiner als die obere Grenze ist oder ob die Grenzwerte einer Gruppe mit denen einer anderen Gruppe für die gleiche Datei überschneiden. Wenn das Formular gültig ist, werden die Gruppendaten in der Datenbank gespeichert.

Das Modell Group in der Datei models.py dient zur Speicherung der Gruppeninformationen. Jede Instanz des Modells enthält Felder für den Dateinamen, den Gruppennamen, die Anzahl der Gruppen, die Gruppenauswahl und die unteren und oberen Grenzwerte.

Die Datenvalidierung wird auch auf der Ebene des Formulars in forms.py durchgeführt. Die Methode clean_group_choice sorgt dafür, dass keine doppelten Gruppenauswahlen für die gleiche Datei existieren können.

Zusätzlich bietet das Modul die Möglichkeit, bestehende Gruppen zu löschen. Dies wird durch die Funktion delete_group(request) ermöglicht, die die entsprechende Gruppen-ID als GET-Parameter annimmt und die zugehörige Gruppeninstanz aus der Datenbank entfernt.

3.8 Group Representation

Das Modul "Group Representation" ist für die Darstellung und Speicherung der segmentierten Gruppen von Datensätzen zuständig. Die Kernfunktionalität ist in der Datei views.py implementiert, und das zugehörige Datenmodell GroupSplit wird in models.py definiert.

Die Funktion group_representation(request) ist der zentrale Bestandteil dieses Moduls. Zunächst werden alle Gruppeninformationen für die aktuelle Datei aus der Datenbank abgerufen, die zuvor im "Group Preprocessing"-Modul festgelegt wurden. Anschließend wird überprüft, ob bereits segmentierte Gruppen in der Datenbank vorhanden sind, um redundante Berechnungen zu vermeiden.

Falls keine segmentierten Gruppen vorhanden sind, wird der ursprüngliche DataFrame geladen und auf der Grundlage der definierten Gruppenkriterien segmentiert. Für jede Gruppe wird ein neuer DataFrame erstellt und gespeichert. Dabei werden sowohl die Gruppennamen als auch die Dateinamen zur Identifikation verwendet. Jeder dieser DataFrames wird dann in eine CSV-Datei umgewandelt und als GroupSplit-Objekt in der Datenbank gespeichert.

Die GroupSplit-Modelle enthalten verschiedene Attribute, darunter den Zeitpunkt des Uploads, den Benutzernamen, den Dateinamen und den Gruppennamen. Zusätzliche Felder wie ai_method, train_test_validation_split, sampling_technique und bayesian_optimization können optional zur späteren Verwendung in künstlicher Intelligenz- oder maschinellem Lernszenarien befüllt werden.

Nachdem die segmentierten Gruppen gespeichert sind, werden diese für die Darstellung im Frontend aufbereitet. Dabei werden verschiedene statistische Metriken berechnet, wie die Anzahl der einzigartigen Firmen, Jahre und Insolvenzfälle, und als HTML-Tabellen gerendert. Diese statistischen Darstellungen werden dann an das Frontend übergeben und in der group_representation.html angezeigt.

Zusätzlich zur Kernfunktionalität nutzt das Modul Caching-Mechanismen, um die Ausführungsgeschwindigkeit zu verbessern. Durch den Einsatz von @cache_page(15 * 60) wird das Ergebnis der group_representation(request)-Funktion für 15 Minuten im Cache gespeichert, um zukünftige Anfragen zu beschleunigen.

3.9 AI Selection

Das Modul "AI Selection" ist ein integraler Bestandteil des Systems und fungiert als Schnittstelle zwischen der Datenaufbereitung und der eigentlichen Anwendung von maschinellem Lernen.

Die Funktion ai_selection(request) ist ein Kernbestandteil des AI-Moduls und dient der Auswahl und Konfiguration von KI-Methoden für verschiedene Datengruppen. Die Funktion wird durch einen HTTP-Request ausgelöst und erhält wichtige Parameter wie den Dateinamen, den aktuellen Benutzer und die Anzahl der Datengruppen.

Zunächst wird geprüft, ob die Anzahl der Gruppen gleich null ist. Ist dies der Fall, hat der Benutzer die Daten in Kapitel 3.7 nicht in Gruppen segmentiert. Danach werden spezielle Feature-Tabellen für den aktuellen Benutzer und die Datei abgerufen. Handelt es sich bei der Anfrage um eine POST-Anfrage, werden KI-Methoden, Gruppenaufteilungen und weitere Parameter vom Benutzer über das Formular empfangen. Anschließend werden die Trainings-, Test- und Validierungsdatensätze generiert, entweder mit oder ohne Bayesianische Optimierung, abhängig von der Benutzerauswahl. Diese Datensätze werden dann in einem Modell namens FinalData gespeichert, das alle notwendigen Informationen für spätere KI-Analysen enthält.

Falls die Anzahl der Gruppen nicht null ist, wird ein ähnlicher Prozess durchgeführt, jedoch werden diesmal die Gruppeninformationen aus der GroupSplit-Tabelle abgerufen. Diese Informationen

werden aktualisiert, um die ausgewählten KI-Methoden und Parameter zu reflektieren. Danach wird für jede Gruppe ein Trainings-, Test- und ggf. Validierungsdatensatz erstellt und in der FinalData-Tabelle gespeichert.

Die Funktionen train_test_val_split und get_train_test_val_split sind Bestandteil der Datenaufbereitung und -teilung in der Anwendung. Sie nehmen jeweils eine Datei und ein Split-Verhältnis als Argumente und führen eine Reihe von Transformationen und Filterungen auf den Daten durch, um sie für maschinelles Lernen vorzubereiten.

In der Funktion train_test_val_split wird zunächst der Datensatz in ein Pandas-DataFrame geladen. Es werden spezifische Spalten entfernt und die Daten werden nach Unternehmen und Jahr sortiert. Anschließend wird der Datensatz in zwei Gruppen geteilt: Unternehmen, die Insolvenz angemeldet haben, und solche, die das nicht getan haben. Für Unternehmen mit Insolvenzen wird nur der Datensatz bis zur Insolvenz behalten. Nach der Generierung dieser Datensätze werden diese zu einem einzigen DataFrame konsolidiert. Dabei erfolgt eine Filterung auf Basis einer einheitlichen Anzahl von Perioden pro Unternehmen, die sich am Mittelwert der Periodenlängen aller Unternehmen orientiert.

get_train_test_val_split führt eine ähnliche Reihe von Schritten durch, fügt jedoch eine Oversampling-Technik hinzu, falls eine solche Methode im Argument sampling_method angegeben ist. Dies hilft, das Ungleichgewicht in der Zielvariable auszugleichen.

Nach der Datenvorbereitung werden die Daten in Trainings-, Test- und optionalen Validierungsdatensätzen aufgeteilt. Diese Aufteilung berücksichtigt die Stratifikation der Zielvariable, um sicherzustellen, dass alle Klassen gleichmäßig verteilt sind. Fehlermeldungen werden generiert, wenn bestimmte Kriterien nicht erfüllt sind, z.B. wenn die Zielvariable nicht mehr als eine Klasse hat oder wenn die Trainingsdaten nicht genügend Fälle von Insolvenzen enthalten.

Die Funktion group_stats generiert Statistiken für Datensatzgruppen, basierend auf mehreren Parametern wie file, current_user und number_of_groups, die über den HTTP-Request erhalten werden. Die Daten werden aus der Datenbank gefiltert und in Pandas-DataFrames geladen, woraufhin deskriptive Statistiken generiert und als HTML-Tabellen gerendert werden. Dank des Cache-Mechanismus mit einer Dauer von 15 Minuten wird die Performance der Funktion verbessert.

Die Funktion xg_param ist für das Training des XGBoost-Modells verantwortlich. Sie erhält die gleichen Parameter wie group_stats sowie zusätzlich die verwendete KI-Methode. Die Daten für Trainings- und Testsets werden aus der Datenbank abgerufen und in Pandas-DataFrames konvertiert. Es gibt zwei Hauptpfade für das Training: entweder durch Bayesianische Optimierung oder durch manuell ausgewählte Hyperparameter. Abhängig von der Methode werden die Modelle trainiert und die relevanten Informationen in der Datenbank gespeichert. Zuletzt werden die trainierten Modelle als Pickle-Dateien gespeichert.

3.10 AI Explainability

Das Modul für KI-Erklärbarkeit integriert Analysemethoden zur Interpretation von KI-Modellen. Das Herzstück dieses Moduls ist die Funktion xgexplain, die in der Datei views.py definiert ist. Diese Funktion ist dafür verantwortlich, bereits trainierte Modelle zu laden, Vorhersagen zu treffen, verschiedene Metriken zu berechnen und Erklärbarkeitsfunktionen wie SHAP (Shapley Additive Explanations) und XGBoost Feature Importance einzusetzen.

Beim Aufruf dieser Funktion werden zunächst alle notwendigen Parameter und Dateien aus dem Request-Objekt extrahiert. Insbesondere wird das Modell, das von Interesse ist, aus der Datenbank geladen. Die Dateipfade der Testdatensätze werden ebenfalls extrahiert, und pandas DataFrames werden für die weitere Verarbeitung erstellt. Das Modell selbst wird mit der Hilfe der pickle

Bibliothek geladen, die in der Hilfsfunktion load_model definiert ist. Anschließend werden Vorhersagen für den Testdatensatz gemacht.

Die nächsten Schritte umfassen die Berechnung verschiedener Metriken wie Genauigkeit, F1-Score, Präzision und Sensitivität. Hier werden Funktionen aus der sklearn.metrics Bibliothek genutzt. Zusätzlich wird eine Konfusionsmatrix erstellt, wobei die confusion_matrix Funktion von sklearn und die heatmap Funktion von seaborn für die grafische Darstellung verwendet werden.

Für den Erklärbarkeitsaspekt des Moduls wird die SHAP-Bibliothek eingesetzt. Ein SHAP-Explainer-Objekt wird mit dem geladenen Modell initialisiert. Dieses Objekt wird dann verwendet, um die SHAP-Werte für den Testdatensatz zu berechnen. Verschiedene SHAP-Plots werden erstellt und gespeichert, einschließlich eines "summary plots" und eines "dependence plots".

Schließlich werden all diese Informationen in ein PDF-Dokument eingebettet, das dann als Bericht in der Datenbank gespeichert wird. Dafür wird die ReportLab-Bibliothek verwendet, um den PDF-Inhalt dynamisch zu generieren.

3.11 Bankrupt Company Search

Die primäre Funktion des Moduls "Bankrupt Company Search" ist die Durchführung einer granularen Suche nach insolventen Unternehmen, wobei erweiterte KI-Modelle und Datenfilterungsfunktionen eingesetzt werden.

Im Hintergrund arbeitet die Anwendung mit verschiedenen Python-Bibliotheken, einschließlich Pandas für Datenmanipulation und Pickle für die Modellserialisierung. Der Hauptkontrollfluss beginnt mit dem search_company-View, der durch die Django-Annotation @cache_page für 15 Minuten gecached wird. Dieser View nimmt mehrere GET-Parameter entgegen, darunter company_id, file, current_user und ai_method, um die Suche zu verfeinern.

Die Anwendung lädt zuerst die finalen Daten und XGBoost-Modelle aus der Datenbank. Diese Daten werden nach dem Benutzer und der Datei gefiltert und dann nach dem Hochladezeitpunkt sortiert. Sobald die Daten und Modelle geladen sind, werden sie mit Pandas und Pickle deserialisiert. Das XGBoost-Modell wird verwendet, um Vorhersagen für die Testdatensätze zu treffen. Diese Vorhersagen werden dann wieder in den DataFrame eingefügt. Dabei wird auch die Spalte Group_Name berücksichtigt, die angibt, zu welcher Gruppe die Daten gehören.

Für die Paginierung der Suchergebnisse wird der Paginator von Django verwendet. Dies ermöglicht eine effiziente Darstellung der Daten, indem nur eine Untermenge der Ergebnisse geladen und angezeigt wird. Der Paginator ist so eingestellt, dass er 100 Zeilen pro Seite anzeigt.

Zusätzlich zu search_company gibt es mehrere Filterfunktionen wie filter_company, filter_bankrupt, filter_predicted_bankrupt und filter_bankrupt_and_predicted. Diese Funktionen ermöglichen eine weitere Feinabstimmung der Suche. Sie sind ähnlich aufgebaut wie search_company, verwenden jedoch JSON-Responses anstelle von HTML-Templates, um die gefilterten Daten zurückzugeben.

Schließlich bietet das Modul eine Exportfunktion namens save_predictions_to_csv, die es ermöglicht, die Suchergebnisse in eine CSV-Datei zu exportieren. Diese Funktion verwendet den HttpResponse-Mechanismus von Django, um eine CSV-Datei zu generieren und sie dem Benutzer zum Download zur Verfügung zu stellen.