1. Erstelle ein Modell, welches die erwarteten Nachforderungen für einen Betrieb vorhersagt (Erreichbare Modellgüte ist durch den Trainingsdatensatz eingeschränkt, eine durchschnittliche Abweichung von 1000€ ist akzeptabel). Welche weiteren Modelle kämen in Frage?

Erstelltes Modell: Ausprobiert habe ich die Architekturen RandomForestRegressor, Lineare Regression und Ridge. Dazu habe ich ein paar Konfigurationen mittels GridSearchCV gestestet.
Der RandomForestRegressor mit einer maximalen Tiefe von 15 Ebenen erzielte auf einem Validierungsset die besten Ergebnisse. Für das beste Modell habe ich einige Ausreißer im Trainingsset entfernt. Die Ergebnisse konnten an einem Testset leider nicht bestätigt werden. Als Metrik habe ich den Mean Absolute Error hinzugezogen, da in der Aufgabenstellung hierauf verwiesen wurde.

Weitere Modelle: Prinzipiell eignen sich alle Regressionsmodelle zur Prädiktion. Beispiele sind der GradientBoostingRegressor oder ein neuronales Netz.

Besonderheiten des Datensatzes: der Datensatz enthält ausschließlich numerische Werte, und enthält viele Nullwerte in vielen Feature-Spalten und der Target-Spalte (Sparsity). Das macht es für einfache Modelle schwierig, gute Vorhersagen zu treffen, während komplexere Modelle, wie neuronale Netze, zum Overfitting neigen und damit nicht generalisierbar sind. Bei der linearen Regression und ihren Verwandten besteht zudem die Schwierigkeit, dass der unterliegende Datengenerierungsprozess wahrscheinlich nicht linear ist (auf den ersten Blick). Ein aufwendiges Feature Engineering könnte Vorhersagen verbessern.

Weitere Ideen, um die Ergebnisse zu verbessern:

- Die PCA verdichtet den Datensatz und sorgt daher für weniger Nullen. Hier habe ich sie nicht vernwendet, auch, weil die Features keine hohen Korrelationen aufweisen
- Ein mehrstufiges Modell könnte ausprobiert werden, vor allem um der Sparseness im Target zu begegnen. Beispielsweise könnte im ersten Schritt vorhergesagt werden, ob überhaupt eine Nachzahlung stattfindet (Klassifizierung), und erst im zweiten, wie hoch die Nachforderung ausfällt.

2. Was sind die stärksten Prädiktoren?

Es gibt verschiedene Ansätze, die Prädiktoren zu interpretieren. Da mein bestes Modell der Scikit-Learn RandomForestRegressor war, habe ich das eingebaute Attribut feature_importances_ genutzt. Laut diesem Attribut hat feature_49 mit Abstand die höchste Wichtigkeit. Danach folgen feature_0 und feature_47.

Die Feature Importance des RandomForests misst, wie viel ein Feature dazu beigetragen hat, die Impurity zu reduzieren. Allerdings hat der RandomForest einige stochastische Elemente und eine iterative Herangehensweise. Dadurch sind die Modelle sehr variabel, wovon auch die Feature Importance betroffen ist.

Alternative Ansätze sind modellagnostische Interpretationsmethoden. Die meisten testen, wie sich Vorhersagen verändern, wenn einzelne Features variiert werden. Das setzt voraus, dass Features möglichst kleine Korrelationskoeffizienten aufweisen, was hier der Fall ist. Beispiele hierfür sind die Permutation Importance oder Shapley Values.

3. Wie wird ein solches Modell evaluiert? Wie kann gemessen werden, wie gut das Modell in der Praxis funktioniert?

In der Trainingsphase werden sowohl Validierungs- und Testset zurückgehalten. Das Validierungsset dient der Modellselektion, das Testset kommt erst am Ende zum Einsatz. Anhand des Testsets wird überprüft, ob die Ergebnisse bestätigt werden können. Wenn nicht, dass ein “Selektionsoverfitting” stattgefunden hat, oder dass Data Leakage vorliegt, also Information in die Validierungsdaten geflossen ist, die hier nicht hätte sein dürfen.

In meinem Fall war die Modellgüte im Testset sehr viel schlechter als im Validierungsset. Ich bin mir recht sicher, dass Data Leakage nicht das Problem war, und auch nicht eine zu fokussierte Modellselektion. Meinem Gefühl nach liegt das Problem an der Sparseness. Möglicherweise war hier auch das Entfernen von Ausreißern keine gute Strategie.

In der Produktionsphase wird die Modellgüte im Rahmen des Monitorings konstant beobachtet.

4. Nenne 5 Aspekte, die für die Produktionalisierung zu berücksichtigen sind.

- **Experiment Tracking**: schon während der Trainingsphase werden Modellergebnisse strukturiert nachverfolgt. So ist für alle Teammitglieder die Historie einsehbar. Außerdem bieten Experiment Tracking Tools wie MLFlow Schnittstellen zum Produktionsprozess. Data Scientists können die besten Modelle dem DevOps-Team zur Produktionalisierung freigeben.

- **CI/CD, Testing**: Wird bei jedem Push in das Versionskontrollsystem eine Testpipeline aktiviert, wird die Wahrscheinlichkeit erhöht, dass nur funktionierender Code in die Produktion eingehen kann. Erst nach Durchlaufen der hinterlegten Tests ist eine Überführung auf die Staging und Production-Ebenen möglich.

- **Monitoring**: ein Modell in der Produktionsphase muss laufend überprüft werden, um sicherzustellen, dass die Modellgüte nicht abfällt. Hierzu sind Tests mit aktuellen Daten notwendig. In dem vorliegenden Setting ist es vermutlich nötig, regelmäßig manuelle Annotierungen vorzunehmen. Das Monitoring kann sich auch auf die Funktionsfähigkeit der Datenpipeline beziehen.

- **Deployment:** hier wird ein System kreiert oder modifiziert, in dem die Prüfenden auf die Vorhersagen zugreifen können. Es wird also eine Schnittstelle zwischen Modell und Applikation geschaffen. Idealerweise werden hierzu bereits bestehende Systeme genutzt, um Datentransfer (und damit Arbeitsaufwand, Datenfehler und Datenschutzmängel) zu vermeiden.

- **Datenqualität/Datenintegration:** Data Engineers stellen sicher, dass die Daten die zur Vorhersage genutzt werden, gewisse Anforderungen erfüllen, damit das Modell überhaupt Vorhersagen treffen kann. Beispielsweise sollten numerische Spalten keine String-Eingaben erlauben. Datentransformationen können teilweise über Preprocessing-Pipelines abgedeckt werden.

5. Beschreibe wie sichergestellt werden kann, dass mehrere Kollegen gleichzeitig an der gleichen Code-Basis effizient arbeiten können.

- Ein **Versionskontrollsystem** wie git (GitLab, GitHub, Bitbucket etc.) sorgt dafür, dass ein Team immer an der aktuellsten Version arbeitet, in Branches abseits der Produktionspipeline arbeiten kann und auch gleichzeitig stattfindende Änderungen integriert werden. Außerdem ist hiermit der Zugriff auf ältere Versionen möglich, ohne dass das Anlegen von Duplikaten mit verschiedenen Versionen nötig wäre.
- Eine **einheitliche Umgebung** ist nötig, um Versionskonflikte (der Packages) zu vermeiden und Ergebnisse reproduzierbar zu machen. Eine Möglichkeit hierfür ist das Arbeiten in *Docker-Containern*. Ein Docker-Image dient als einheitliche Vorlage. Alternativ kann auch eine *virtuelle Maschine* genutzt werden. Für kleinere, überschaubare Projekte würde unter Umständen das Arbeiten mit *Environments* ausreichen. In Python gibt es beispielsweise das Anaconda Environment, oder virtualenv und venv. Die virtuellen Environments bieten weniger Abgrenzung zum restlichen Betriebssystem als Docker.