Project Data Job
Final Report

Data Exploration, Visualization, Preprocessing, Modelling and Feature Importance

Ayele T. Sedeke, Frankfurt

Christopher Peter Giesbrecht, Bielefeld

Jan Simon, Hamburg

Cohort: B2C | DA Inter | Continu | JAN25 | English

Content

0 Executive Abstract

We developed a **role recommender system** to help students identify career paths that best align with their skills and aspirations.

The project is based on the **2020 Kaggle Data Science and Machine Learning Survey** with 20,036 valid responses from 171 countries. The **RoleRecommender App** was implemented on **Streamlit**.

This report focuses on **data exploration, preprocessing, modelling, and validation**. From Question 5: “Q5_Select the title most similar to your current role (or most recent title if retired):”we derived two target variables (broad vs. specific career goals).

[Christopher: Hier bitte Dein Input. Ich kann meinen Teil noch kürzen]

We applied **pipeline techniques** with **OrdinalEncoding** for categorical features and trained **Random Forest, HistGradientBoosting,** and **XGBoost** models. Random Forest achieved the highest F1-score for the "Data Analyst" class, while HistGradientBoosting and XGBoost showed stronger overall performance across all classes.

We chose XGBoost as the **core engine** of our app because it showed the best overall **performance** among the tested models. It predicts the broad career path “Data Science” and the specific paths “Data Scientist” and “Software Engineer” **reliably.**

Key improvement areas:

(1) **Broader feature selectio**n, optional by using unsupervised methods, to get more explanatory power on Tech and Data Analyst career paths.
(2) **Refinement of target class** definitions for better balance of classes.
(3) Extended **hyperparameter tuning** for better model performance.
(4) **Alternative algorithms** that might better capture the characteristics of low performing classes
(5) **More exact coding** on feature importance, format selection and inverse label transformation.

However, the app and its notebooks **are stable and ready for use**. We hope they are offering valuable insights for the data science community.

1 Overview on Coding steps

The following steps are implemented in the notebook `RoleRecommender_Models_v1.0.ipynb`.

Table 1: Description of Coding steps

Coding steps	Description
1. Data Preparation	- Import and merge raw survey data. - Combine question IDs and texts as column headers - Handle missing values ("z_not_answered") - Encode multiple-choice answers as boolean data.
2. Feature & Target Engineering	- Select relevant questions - Aggregate multiple-choice selections - Create target variables y_L (Broad Career Focus) and y_S (Specific Career Focus)
3. Data Preprocessing	- Assign encoders per feature/target (encoder_assignment.csv) - Apply OneHotEncoder for nominal features and Ordinal Encoding for ordered categories (unique_with_rank.csv) - Build preprocessing pipelines (preprocessor_L and preprocessor_S).
4. Train-Test Split	- Partition the dataset into training and test sets.
5. Modeling & Evaluation	- Train RandomForest, HistGradientBoosting, and XGBoost pipelines - Perform hyperparameter tuning using RandomizedSearch - Evaluate performance with classification reports
6. Evaluating Feature Importance	- Apply SHAP-Method - Visualize results

The next sections show results of data exploration as well as the corresponding coding steps and commented code snippets.

2 Data Exploration and Visualisation

2.1 Subsection 1

Christopher?

2.2 Subsection 2

Christopher?

2.3 Subesection 3

Christopher?

3 Feature and Target Engineering

3.1 Import of Modules and Classes

```
#Import of relevant libraries, classes or methods (in order of appearance)

## For imports in Colab
from google.colab import files

## Standard libraries
import pandas as pd
import numpy as np

## Data pre processing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer

## For pipelines
from sklearn.pipeline import Pipeline
from sklearn.base import clone

## Machine Leaning models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from xgboost import XGBClassifier

## Tuning hyperparameters
from sklearn.model_selection import RandomizedSearchCV

## Evaluate performance and feature importance
from sklearn.metrics import classification_report
import shap
```

3.2 Data Structure

3.2.1 Columns

We start with DataFrame `df_short`, which consists of the following columns:

```
Index(['Q1_What is your age (# years)?',

       'Q4_What is the highest level of formal education that you have attained or plan to attain
within the next 2 years?',

       'Q5_Select the title most similar to your current role',

       'Q6_For how many years have you been writing code and/or programming?',

       'Q8_What programming language would you recommend an aspiring data scientist to learn first?',

       'Q13_Approximately how many times have you used a TPU (tensor processing unit)?',

       'Q15_For how many years have you used machine learning methods?',

       'Q20_What is the size of the company where you are employed?',

       'Q22_Does your current employer incorporate machine learning methods into their business?',

       'Q24_What is your current yearly compensation (approximate $USD)?',

       'Q30_Which of the following big data products (relational database, data warehouse, data lake,
or similar) do you use most often?',

       'Q32_Which of the following business intelligence tools do you use most often?',
       'Q7_No. of Regular used programming languages?',

       'Q9_No. of Specialized IDE?', 'Q14_No. of DataViz Libs or Tools?',

       'Q16_No. of ML Framworks?', 'Q17_No. of ML algorithms?',

       'Q18_No. of Computer Vsion methods?', 'Q19_No. of NLP methods?',

       'Q26_A No. of Current Cloud platforms?',

       'Q27_A No. of Current Cloud Products?',

       'Q28_A No. of Current ML products?', 'Q29_A No. of Big Data Tools?',

       'Q31_A No. of BI Tools used?', 'Q33_A No. of Automted ML Tools?',

       'Q34_A No. of Auto ML Tools?', 'Q35_A No. of ML Experiment Management?',

       'Q37_No. of Learning Platforms?'],

      dtype='object')
```

3.2.2 Target definition

The overall target column is `'Q5_Select the title most similar to your current role'`

During the exploration phase it turned out that concentrating on pure Data Science roles could be problematic as data could be biased because of small sample numbers per class. Hence, we define a second target `y_L` that consists of a Broad_career path:

```
     ds_roles = [
        'Data Scientist',
        'Data Analyst',
        'Machine Learning Engineer',
        'Data Engineer',
        'Research Scientist'
    ]
    tech_roles = [
        'Software Engineer',
        'DBA/Database Engineer'
    ]
```

On the other hand, we will define Specifc career roles `y_S` as follows:

```
    selected_roles_S = [
          'Data Scientist',
          'Software Engineer',
          'Data Analyst'
        ]
```

4 Train-Test Split

After applying several transforming steps, we define the following training- and test sets with `train_test_split` method:

```
X_train_L, X_test_L, y_train_L, y_test_L =
                     train_test_split(X_L, y_L,
                                      test_size=0.2,
                                      random_state=42)

X_train_S, X_test_S, y_train_S, y_test_S =
                     train_test_split(X_S, y_S,
                                      test_size=0.2,
                                      random_state=42)
```

5 Data Preprocessing

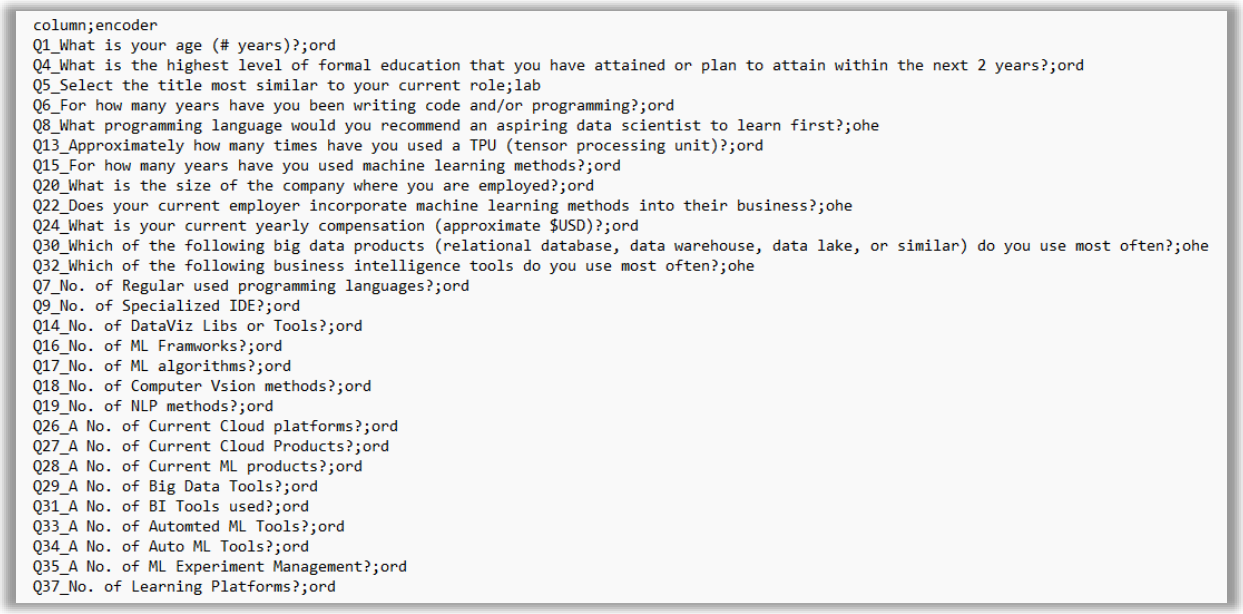
5.1 Endoding Strategy

Based on the data exploration, both the target variables and the features consist exclusively of categorical data. To ensure a robust and reproducible preprocessing pipeline, we implemented a structured encoding strategy comprising the following components:

- Label Encoding for the target variables `y_L` and `y_S`. Labels are automatically assigned in alphabetical order.
- One-Hot Encoding for features without intrinsic order. New binary indicator columns are generated automatically for each category.
- Ordinal Encoding for features with a defined intrinsic order, requiring a custom ranking of categories.

Assigning encoder types manually to each column via dictionaries would be both time-consuming and error prone. To streamline this process, we created the configuration file `encoder_assignment.csv` (UTF-8 format), which defines the encoding method for every feature in a centralized and structured manner.

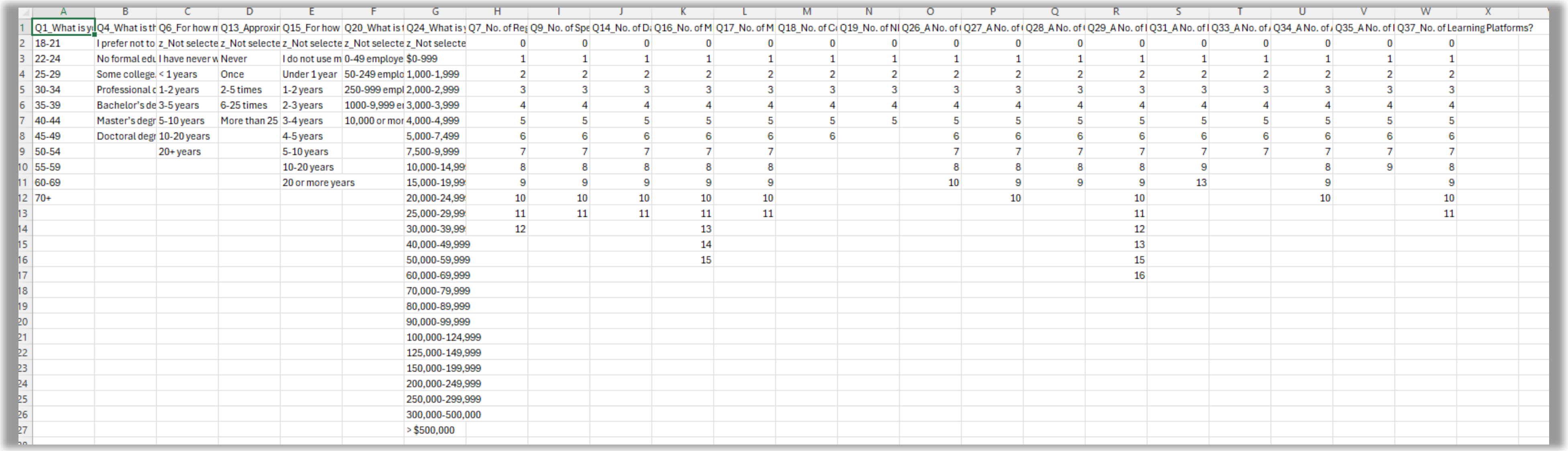
Table 2: encode_assigment.csv



```
column;encoder
Q1_What is your age (# years)?;ord
Q4_What is the highest level of formal education that you have attained or plan to attain within the next 2 years?;ord
Q5_Select the title most similar to your current role;lab
Q6_For how many years have you been writing code and/or programming?;ord
Q8_What programming language would you recommend an aspiring data scientist to learn first?;ohe
Q13_Approximately how many times have you used a TPU (tensor processing unit)?;ord
Q15_For how many years have you used machine learning methods?;ord
Q20_What is the size of the company where you are employed?;ord
Q22_Does your current employer incorporate machine learning methods into their business?;ohe
Q24_What is your current yearly compensation (approximate $USD)?;ord
Q30_Which of the following big data products (relational database, data warehouse, data lake, or similar) do you use most often?;ohe
Q32_Which of the following business intelligence tools do you use most often?;ohe
Q7_No. of Regular used programming languages?;ord
Q9_No. of Specialized IDE?;ord
Q14_No. of DataViz Libs or Tools?;ord
Q16_No. of ML Framworks?;ord
Q17_No. of ML algorithms?;ord
Q18_No. of Computer Vsion methods?;ord
Q19_No. of NLP methods?;ord
Q26_A No. of Current Cloud platforms?;ord
Q27_A No. of Current Cloud Products?;ord
Q28_A No. of Current ML products?;ord
Q29_A No. of Big Data Tools?;ord
Q31_A No. of BI Tools used?;ord
Q33_A No. of Automted ML Tools?;ord
Q34_A No. of Auto ML Tools?;ord
Q35_A No. of ML Experiment Management?;ord
Q37_No. of Learning Platforms?;ord
```

While Label and OneHotEncodings are straightforward, OrdinalEncoding requires explicit definition of category order for each relevant feature. To address this, we introduced a second configuration file, `unique_with_rank.csv` (UTF-8 format). This file stores the intrinsic order of all ordinal features in a tabular structure, mapping each feature to its categories and their associated rank. This approach ensures consistency, reproducibility, and full transparency of the encoding process.

Table 3: unique_with_rank.csv



Q1_What is y	Q4_What is th	Q6_For how m	Q13_Approxir	Q15_For how	Q20_What is t	Q24_What is y	Q7_No. of Reg	Q9_No. of Spe	Q14_No. of Da	Q16_No. of M	Q17_No. of M	Q18_No. of Co	Q19_No. of NI	Q26_A No. of	Q27_A No. of	Q28_A No. of	Q29_A No. of	Q31_A No. of	Q33_A No. of	Q34_A No. of	Q35_A No. of	Q37_No. of Learning Platforms?
18-21	I prefer not to	z_Not selecte	z_Not selecte	z_Not selecte	z_Not selecte	z_Not selecte	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22-24	No formal edu	I have never w	Never	I do not use m	0-49 employe	$0-999	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
25-29	Some college.	< 1 years	Once	Under 1 year	50-249 emplo	1,000-1,999	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
30-34	Professional c	1-2 years	2-5 times	1-2 years	250-999 empl	2,000-2,999	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
35-39	Bachelor's de	3-5 years	6-25 times	2-3 years	1000-9,999 ei	3,000-3,999	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
40-44	Master's degr	5-10 years	More than 25	3-4 years	10,000 or mor	4,000-4,999	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
45-49	Doctoral degr	10-20 years		4-5 years		5,000-7,499	6	6	6	6	6	6		6	6	6	6	6	6	6	6	6
50-54		20+ years		5-10 years		7,500-9,999	7	7	7	7	7			7	7	7	7	7	7	7	7	7
55-59				10-20 years		10,000-14,99	8	8	8	8	8			8	8	8	8	9		8	9	8
60-69				20 or more years		15,000-19,99	9	9	9	9	9			10	9	9	9	13		9		9
70+						20,000-24,99	10	10	10	10	10				10		10			10		10
						25,000-29,99	11	11	11	11	11						11					11
						30,000-39,99	12			13							12					
						40,000-49,999				14							13					
						50,000-59,999				15							15					
						60,000-69,999											16					
						70,000-79,999																
						80,000-89,999																
						90,000-99,999																
						100,000-124,999																
						125,000-149,999																
						150,000-199,999																
						200,000-249,999																
						250,000-299,999																
						300,000-500,000																
						> $500,000																

We generate list of column names for each encoder type:

```
## Assign columns to encoder type:

columns_to_keep = [col for col in encoder_assignment["column"] if col in df_short.columns]
lab_columns  = encoder_assignment.query("encoder == 'lab'")["column"].tolist()
ohe_columns  = encoder_assignment.query("encoder == 'ohe'")["column"].tolist()
ord_columns  = encoder_assignment.query("encoder == 'ord'")["column"].tolist()
```

This is a pre-requisite for building the `ColumnTransformer` pipeline (see next section).

In addition, for `ord_columns`, we create the list categories, which we then will set as a parameter for the `OrdinalEncoder`. Handling NaNs and numbers requires some coding here, but helps us to reach the goal:

```
def clean_float_strings(val):
    # Werte, die echte NaN darstellen sollen
    if val in ['nan', 'NaN', '']:
        return np.nan
    try:
        f = float(val)
        if f.is_integer():
            return str(int(f))  #  '0.0' → '0'
        return str(f)           #  '2.5' bleibt '2.5'
    except:
        return val             # Text remains text

unique_with_rank_cl = unique_with_rank.applymap(clean_float_strings)

### Create finale list of ordered categories for OrdinalEncoder:

unique_dict = {}

for col in unique_with_rank_cl.columns:
    cats = unique_with_rank_cl[col].dropna().unique().tolist()
# Extract all values without NaNa and remove duplicates
    cats = [cat for cat in cats if cat != 'nan']

# Remove nan as string
    unique_dict[col] = cats

### Final list for Ordninal Encoder

categories = list(unique_dict.values())
```

In fact, json-format would have been the better choice here because of less trouble with handling NaNs. However, we are prepared for data preprocessing by applying distinct types of encoders.

5.2 Encoding for target variables

For simplicity, we do not use `lab_columns` which makes the list redundant. We instantiate and apply the `LabelEncoder`. After having applied `fit_transform` the training set and transform on the test set, we re-generate the targets as Series.

```
# Apply Label Encoder for target

lab_L = LabelEncoder()
lab_S = LabelEncoder()

# Transform into Series and apply fit_transform/ transform
```

5.3 Feature encoding with pipelines for ColumnTransformer

Thanks to our preparation, we now use the following code to transform the feature columns with `OneHotEncoder` and `OrdinalEncoder`. Both are defined as a `pipeline`, then the `ColumnTransformer` merges into one pipeline, which we integrate in the next section into our Machine Leaning models.

```
# Pipeline for one-hot encoding: drops the first category to avoid multicollinearity,
# outputs a dense array, and ignores unknown categories.

ohe_transformer = Pipeline([
    ('ohe', OneHotEncoder(drop='first', sparse_output=False, handle_unknown='ignore'))
])

# Pipeline for ordinal encoding uses specified 'categories',
# assigns -1 to unknown values.

ord_transformer = Pipeline([
    ('ord', OrdinalEncoder(categories=categories, handle_unknown='use_encoded_value', unknown_value=-
1))
])

# ColumnTransformer applies different transformers to specified columns:
# 'ohe_transformer' to 'ohe_columns', and 'ord_transformer' to #'ord_columns'.


preprocessor = ColumnTransformer([
    ('ohe', ohe_transformer, ohe_columns),
    ('ord', ord_transformer, ord_columns)
])
```

6 Modelling

6.1 Model Choice

We choose three models with for two data sets related to targets `y_L` and `y_S`.

Table 4: Description of Coding steps

Target / Model	Broad career path y_L	Specific career path y_S
RandomForest	pipe_rfc_L	pipe_rfc_S
HistGradientBooster	pipe_hgb_L	pipe_hgb_SL
XGBoost	pipe_xgb_L	pipe_xgb_S

We have chosen these models because they are suitable for categorical data like in our case. The models were recommended either in our course (`RandomForest`[1], `XGBoost`[2]) or by our tutor Alia (`HistGradientBooster`[3]). All models represent ensemble models[whlink], where single models are combined to one stronger model.

[1] See https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[2] https://xgboost.readthedocs.io/en/stable/python/python_api/sklearn.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

RandomForest represents a bagging model, where decision trees are trained on randomly chosen sub data from the training set.

HistGradientBooster and XGBoost, on the other hand, train the model in a sequence, where each sub model corrects the error for the previous sub-model.

6.2 Initial Setup

Basically, the initial default parameter setup for a model ML model can be derived from the relevant documentations or by using

```
print(RandomForestClassifier().get_params())
print(XGBClassifier().get_params())
print(HistGradientBoostingClassifier().get_params())
```

In our case, for simplicity and time pressure, we have asked ChatGPT or Gemini.

Initial set up for `RandomForest`:

```
# RandomForest pipeline:
# - max_depth=50: limit tree depth to prevent overfitting
# - random_state=42 ensures reproducible results
# - n_estimators=100: number of trees in the forest
# - class_weight='balanced': adjusts for class imbalance

pipe_rfc = Pipeline([
    ('preprocessing', preprocessor),
    ('rfc', RandomForestClassifier(
        max_depth=50,
        random_state=42,
        n_estimators=100,
        class_weight='balanced'
    ))
])
```

Initial set up for `HistGradientBooster`:

```
# HistGradientBoosting pipeline:
# - learning_rate=0.05: shrinkage rate for each tree
# - max_iter=200: number of boosting iterations (trees)
# - max_depth=6: maximum tree depth
# - min_samples_leaf=20: minimum samples per leaf (controls overfitting)
# - l2_regularization=1.0: L2 penalty to reduce model complexity
# - random_state=42 ensures reproducibility

pipe_hgb = Pipeline([
    ('preprocessing', preprocessor),
    ('hgb', HistGradientBoostingClassifier(
        learning_rate=0.05,
        max_iter=200,
        max_depth=6,
        min_samples_leaf=20,
        l2_regularization=1.0,
        random_state=42
    ))
])
```

Initial set up for `XGBoost`:

```
# XGBoost pipeline:
# - learning_rate=0.05: step size shrinkage
# - n_estimators=200: number of boosting rounds
# - max_depth=6: maximum tree depth
# - reg_lambda=1.0: L2 regularization
# - min_child_weight=20: min. sum of instance weight in a leaf
# - objective='multi:softmax': multi-class classification (direct class output)
# - num_class=4: number of target classes
# - eval_metric='mlogloss': multiclass log loss as evaluation metric
# - random_state=42 ensures reproducibility

pipe_xgb = Pipeline([
    ('preprocessing', preprocessor),
    ('xgb', XGBClassifier(
        learning_rate=0.05,
        n_estimators=200,
        max_depth=6,
        reg_lambda=1.0,
        min_child_weight=20,
        objective='multi:softmax',
        num_class=4,
        eval_metric='mlogloss',
        random_state=42
    ))
])
```

6.3 Tuning Hyperparameters

Based on the initial ML models[4], we tune hyperparamters. We use `RandomizedSearchCV`, as we can handle big parameter space with small computing capacity. The latter is even more important, as our team worked on Google Colab which is slower than locally installed consoles. As an alternative, `GridSearchCV` is known to us from the course. `GridSeachCV` has the advantage of explicitly testing each combination in the defined parameter space. However, this leads to the disadvantage of long computing time. Above all, it turned out that in our case there was a tendency for overfitting. Hence, `RandomizedsearchCV` was the right choice from our perspective.

Whatever the optimization methods are, two basic questions must be answered:

1. What is the optimization target?
   We choose `f1-macro`, because our data exploration showed the especially in `y_S` we see small groups leading unbalanced data.

2. How do we define the parameter space?
   We have to find the right balance between the number of parameters and the concrete arguments defined for each parameter. The more combination is possible, the longer we must wait for results due to computer power restrictions.

Please note that during the project phase, our emphasize was more on clear coding rather than finding the last percentages by parameter tuning.

`RandomForerest`: Parameter space and tuning by `RandomizedSearchCV`

```
# Define Parameter Space for RandomForestClassifier:

param_dist_rfc = {
    'rfc__n_estimators': [100, 200, 300],           # No. of trres
    'rfc__max_depth': [10, 30, 50, None],           # Max. Depth with none as no limit
    'rfc__min_samples_split': [2, 5, 10],           # Split of Trre
    'rfc__min_samples_leaf': [1, 2, 4],             # Minuum amout of leafs
    'rfc__max_features': ['sqrt', 'log2', None],    # Features for split
    'rfc__bootstrap': [True, False]                 # Bootstrapping on/off
```

[4] For results of initial codes, compare notebook RoleRecommender v1.0 models.ipynb

```
}


# RandomizedSearchCV für Set S
random_search_rfc_S = RandomizedSearchCV(
    clone(pipe_rfc),
    param_distributions=param_dist_rfc,
    n_iter=10,
    cv=3,
    scoring='f1_macro',
    random_state=42,
    n_jobs=-1,
    verbose=1
)

# RandomizedSearchCV für Set L
random_search_rfc_L = RandomizedSearchCV(
    clone(pipe_rfc),
    param_distributions=param_dist_rfc,
    n_iter=10,
    cv=3,
    scoring='f1_macro',
    random_state=42,
    n_jobs=-1,
    verbose=1
)
```

`HistGradientBooster`: Parameter space and tuning by `RandomizedSearchCV`

```
# Define Paramter Space for Gradient Booster:
param_dist_hgb = {
    'hgb__learning_rate': [0.01, 0.05, 0.1],          # shrinkage step size
    'hgb__max_iter': [100, 200, 300],                  # number of boosting iterations
    'hgb__max_depth': [3, 6, 9],                       # maximum depth of individual trees
    'hgb__min_samples_leaf': [10, 20, 30],             # minimum samples per leaf
    'hgb__l2_regularization': [0.0, 1.0, 10.0]         # L2 penalty for regularization
}


from sklearn.model_selection import RandomizedSearchCV
from sklearn.base import clone

# Randomized search for dataset S
random_search_hgb_S = RandomizedSearchCV(
    clone(pipe_hgb),                     # avoid shared state
    param_distributions=param_dist_hgb,  # defined param space
    n_iter=10,                           # number of combinations
    cv=3,                                # 3-fold CV
    scoring='f1_macro',                  # macro F1 score
    random_state=42,
    n_jobs=-1,                           # use all CPUs
    verbose=1
)

# Randomized search for dataset L
random_search_hgb_L = RandomizedSearchCV(
    clone(pipe_hgb),
    param_distributions=param_dist_hgb,
    n_iter=10,
    cv=3,
    scoring='f1_macro',
    random_state=42,
    n_jobs=-1,
    verbose=1
)
```

`XGBoost`: Parameter space and tuning by RandomizedSearchCV

```
# Define parameter grid for RandomizedSearchCV
param_dist = {
    'xgb__learning_rate': [0.01, 0.05, 0.1],
    'xgb__n_estimators': [100, 200, 300],
    'xgb__max_depth': [3, 6, 9],
    'xgb__reg_lambda': [0.5, 1.0, 2.0],
    'xgb__min_child_weight': [1, 10, 20]
}

# Create RandomizedSearchCV objects for each dataset
random_search_S = RandomizedSearchCV(
    clone(pipe_xgb),          # clone to avoid shared state
    param_distributions=param_dist,
    n_iter=10,                # number of parameter settings sampled
    cv=3,                     # 3-fold cross-validation
    scoring='f1_macro',
    random_state=42,
    n_jobs=-1,                # use all CPU cores
    verbose=1
)

random_search_L = RandomizedSearchCV(
    clone(pipe_xgb),
    param_distributions=param_dist,
    n_iter=10,
    cv=3,
    scoring='f1_macro',
    random_state=42,
    n_jobs=-1,
    verbose=1
```

6.4 Results

We choose classification report for evaluating model performance.

Table 5: Results for Data Set y_L

Model	Class	Precision	Recall	F1-Score	Support
RandomForest	DS	0.8656	0.8612	0.8634	1369
	Tech	0.5540	0.5632	0.5586	419
HistGradientBoost	DS	0.8446	0.9131	0.8775	1369
	Tech	0.6136	0.4511	0.5199	419
XGBoost	DS	0.8448	0.9145	0.8783	1369
	Tech	0.6176	0.4511	0.5214	419

All three models perform well on the dominant class "DS" (F1 ≈ 0.86–0.88), with XGBoost and HistGradientBoosting slightly outperforming RandomForest.

Performance on the minority class "Tech" is notably lower across all models, with XGBoost showing the best overall balance.

The class imbalance (1369 vs. 419) leads to a significant drop in recall and F1-score for "Tech", especially in the boosting models, where recall drops to 0.45.

Table 6: Results for Data Set y_S

Model	Class	Precision	Recall	F1-Score	Support
RandomForest	Data Analyst	0.5593	0.5482	0.5537	301
	Data Scientist	0.6934	0.7197	0.7063	528
	Software Engineer	0.6850	0.6608	0.6727	395
HistGradientBoost	Data Analyst	0.5756	0.4551	0.5083	301
	Data Scientist	0.6911	0.7500	0.7193	528
	Software Engineer	0.6828	0.7139	0.6980	395
XGBoost	Data Analyst	0.5907	0.4651	0.5204	301
	Data Scientist	0.6850	0.7538	0.7178	528
	Software Engineer	0.6872	0.7063	0.6966	395

Among the three models, all perform best on the "Data Scientist" and "Software Engineer" classes, with F1-scores above 0.69.

The "Data Analyst" class consistently shows lower performance across models (F1 ≈ 0.51–0.55), from our perspective due to fewer samples (301 vs. 528/395) or more complex class boundaries.

While RandomForest shows the highest F1-score for 'Data Analyst', HistGradientBoost and XGBoost deliver better overall balance across all classes, with slightly stronger performance on the majority classes.

For simplicity and code maintenance, **we decided to concentrate on XGBoost regarding Feature Importance.** In the next section, the other models will not be investigated any more.

7 Evaluating Features with SHAP

7.1 Why Feature Importance?

Now, as we have received an adequate balanced and optimized Machine Learning Model, we come close to our original business question: What are crucial factors for a future role in the Tech industry? What are relevant factors for a role for a more tech oriented or more Data Science oriented role? And if I had the choice becoming a Data Analyst, Data Scientist or Software engineer, what factors matter?

In our course, we have applied the `feature_importances` method which easily applicable to DataFrame objects. However, while doing some research in the libraries, we found that the `SHAP` method[5] (Shapley Additive Explanations) has more explanation power and can deliver more sophisticated graphical analytics, which might be important for later app configuration.[6]

`SHAP` calculates the contribution of each individual feature to a model's prediction — for each instance. This is what we need.

7.2 Note on Differences Between App and ML Models

There is a small but important difference between the `XGBoost` model presented here and the model used in the RoleRecommender App.

The App initializes the models as follows:

```
# Determine the number of unique classes in each target set
num_classes_S = len(np.unique(y_train_S))
num_classes_L = len(np.unique(y_train_L))
```

In contrast, as described in Section 6.2, both `xgb_L` and `xgb_S` in this report were instantiated with `# - num_class=4: number of target classes`

Why does this matter? The broad career path is represented by binary classes (0 = Data Science, 1 = Tech). By defining `num_class=4`, XGBoost incorrectly assumes a multiclass problem, resulting in SHAP values that appear for four classes instead of two. This can be confusing when interpreting the SHAP rankings for `xgb_L`, as we are in fact dealing with a binary classification problem.

Another difference is that in the RoleRecommender App, features that were one-hot encoded are excluded from SHAP analysis to improve user experience, since the OneHotEncoder generates verbose and less interpretable feature names.

Consequently, the SHAP values reported here differ from those displayed in the App.

7.3 Code

Look on the code for calculating `SHAP` values for `y_L` and `y_S`:

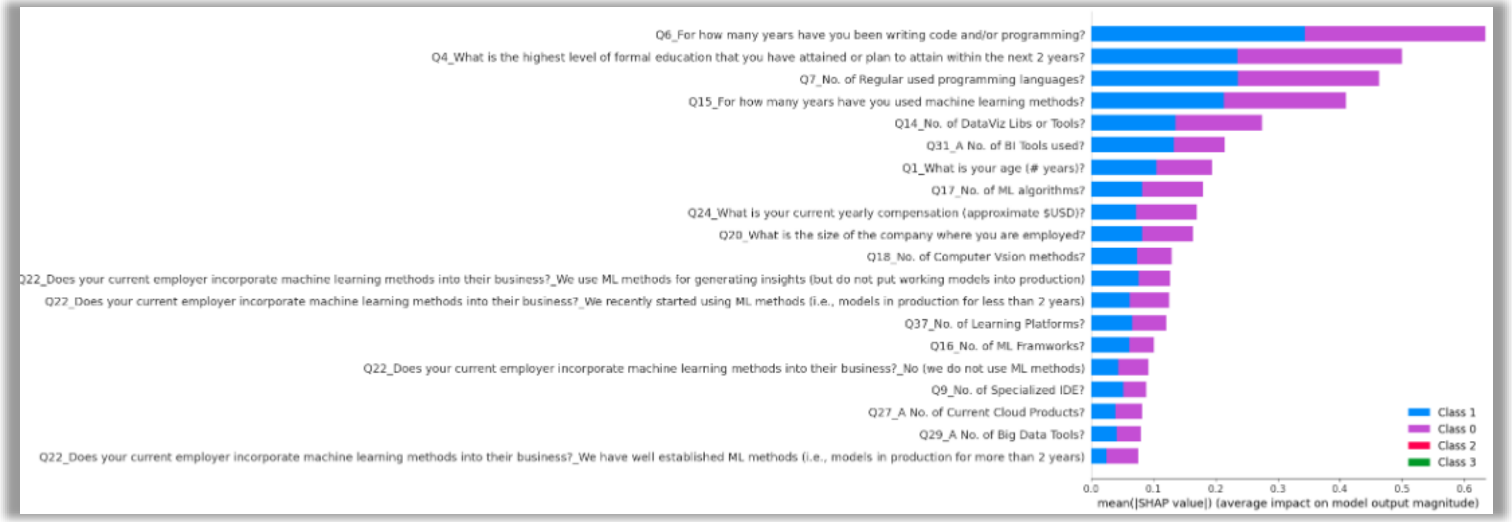
[6] See https://shap.readthedocs.io/en/latest/ for official SHAP Documentation.

```
def shap_analysis_from_pipeline(pipe, X_train, filename_csv):
    # Extract model and preprocessor from pipeline
    model = pipe.named_steps['xgb']
    preprocessor = pipe.named_steps['preprocessing']

    # Sample and preprocess data
    X_sample = X_train.sample(100, random_state=42)
    X_transformed = preprocessor.transform(X_sample)

    # Get feature names
    feature_names = get_feature_names(preprocessor)

    # Initialize SHAP explainer and calculate SHAP values
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_transformed)  # shape: (samples, features, classes)

    # Get class names
    class_names = model.classes_

    # Rearrange SHAP values for class-wise iteration
    shap_values_classwise = np.transpose(shap_values, (2, 0, 1))  # (classes, samples, features)

    # Collect mean absolute SHAP values per class
    shap_list = []
    for i, class_name in enumerate(class_names):
        df_shap = pd.DataFrame(shap_values_classwise[i], columns=feature_names)
        shap_mean = df_shap.abs().mean().sort_values(ascending=False)
        shap_list.append(shap_mean.rename(f"Mean |SHAP| ({class_name})"))

    # Combine results into one DataFrame
    df_shap_all = pd.concat(shap_list, axis=1)

    # Display the combined table
    display(df_shap_all)

    # Save to CSV
    df_shap_all.to_csv(filename_csv)

    # Plot SHAP summary bar chart
    shap.summary_plot(shap_values, X_transformed, feature_names=feature_names, plot_type="bar")

# Call for für model L
# shap_analysis_from_pipeline(pipe_xgb_L, X_train_L, "shap_feature_importance_all_classes_L.csv")

# Call for für model L
# shap_analysis_from_pipeline(pipe_xgb_S, X_train_S, "shap_feature_importance_all_classes_S.csv")
```

Now look on the next page to see the results:

7.4 Results

Plot 1: Feature importance for y_L and y_S (bar plot)



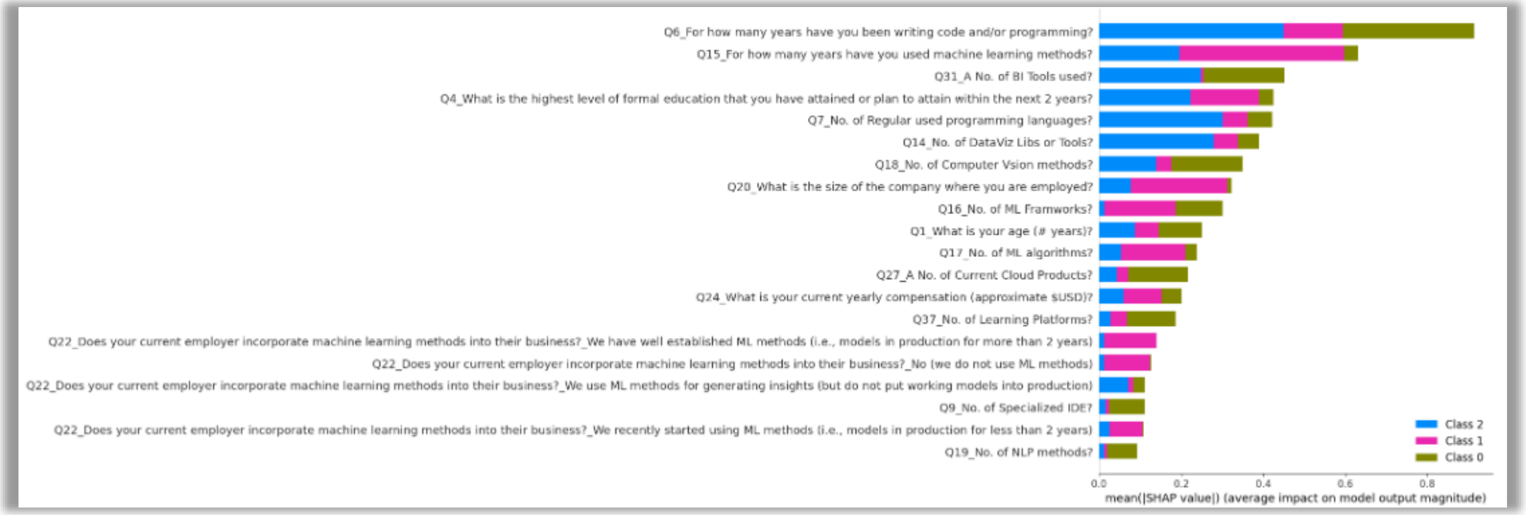


Table 7: Feature importance for `y_L` (Table, Top 10)

Feature	**Mean |SHAP| (1)**
Q6_For how many years have you been writing code and/or programming?	0.3437433
Q4_What is the highest level of formal education that you have attained or plan to attain within the next 2 years?	0.2344554
Q7_No. of Regular used programming languages?	0.23549913
Q15_For how many years have you used machine learning methods?	0.21285132
Q14_No. of DataViz Libs or Tools?	0.13546537
Q17_No. of ML algorithms?	0.08096919
Q24_What is your current yearly compensation (approximate $USD)?	0.07249658
Q1_What is your age (# years)?	0.10509999
Q20_What is the size of the company where you are employed?	0.081667855
Q31_A No. of BI Tools used?	0.13249616

Table 8: Feature importance for `y_S` (Table, Top 10)

Feature	**Mean |SHAP| (0)**	**Mean |SHAP| (1)**	**Mean |SHAP| (2)**
Q6_For how many years have you been writing code and/or programming?	0.31980723	0.14511815	0.44924602
Q31_A No. of BI Tools used?	0.19679905	0.006944795	0.24706441
Q18_No. of Computer Vsion methods?	0.17369808	0.03740157	0.13855328
Q27_A No. of Current Cloud Products?	0.14556897	0.028403513	0.041869167
Q37_No. of Learning Platforms?	0.12014642	0.040044785	0.026903678
Q16_No. of ML Framworks?	0.11571947	0.17306806	0.012887654
Q1_What is your age (# years)?	0.1056089	0.056996033	0.08750332
Q9_No. of Specialized IDE?	0.08626564	0.008121383	0.015968625
Q19_No. of NLP methods?	0.072917655	0.006585701	0.012172943
Q7_No. of Regular used programming languages?	0.060197808	0.059852432	0.3013157

For further investigation, we recommend explore the results in* `RoleRecommender_Models_v1.0.ipynb` *or directly in the App*[7]*.

[7] https://rolerecommender-app-5tbdfgf6jgrgxpvbtx2bgx.streamlit.app/Your_personal_RoleRecommender

8 Concluding Remarks

We developed a **role recommender** system that guides students towards career paths aligned with their skills and aspirations. The system is based on the 2022 Kaggle survey, containing over **20,000 responses** and more than **300 questions**, including optional multiple-choice items. Technically, we addressed a binary and multiclass **classification problem** with categorical data. Key steps included selecting **three ensemble models** (scikit-learn and xgboost), transforming multiple-choice responses into boolean values, and interpreting NaNs as unanswered questions. We also focused on careful **feature selection**, **hyperparameter tuning**, and **feature importance** analysis to ensure robust and comparable models. Finally, the app design emphasized user experience while minimizing client-side computational requirements.

[Christopher, Dein Input]

We chose XGBoost as the core engine due to its strong overall performance. It predicts the broad career path "Data Science" and the specific paths "Data Scientist" and "Software Engineer" **reliably**, while predictions for "Tech" and "Data Analyst" remain close to random chance. Despite these weaknesses, XGBoost was the best fit for the RoleRecommender. We recommend addressing limitations by expanding the dataset, optimizing features for "Tech" and "Data Analyst," and evaluating alternative algorithms that better capture these classes.

Coding: Our code contains **redundancies**, as the two notebooks share many identical code lines, making maintenance more difficult. As noted in Section 7.2, a **suboptimal parameter** definition in XGBoost could introduce errors. There is also room for improvement in handling **long question texts** and their mapping to question numbers—critical for the app's **user experience**. Regular expressions (REGEX) could have enhanced clarity and readability.

However, the app and its notebooks **are stable and ready for use**. We hope they are offering valuable insights for the data science community.

Appendix: Relevant Links to Github and Streamlit

id	Feature	Link
1	RoleRecommender App, hosted on Streamlit Cluad	https://w9v3uossaz4zuzqyfzufvp.streamlit.app/
2	Github respsitory with focus on Machine Learning and generating relevant data for RoleRecommender App	https://github.com/jan399/final_report
3	Data sets feeding the Notebooks	https://github.com/jan399/final_report/tree/main/data
4	Export data for feeding the App	https://github.com/jan399/final_report/tree/main/notebooks/downloads_for_streamlit
5	Github respsitory for RoleRecommender App	https://github.com/jan399/my_app
6	Modules for App pages	https://github.com/jan399/my_app/tree/main/pages
7	Modules for App support functions	https://github.com/jan399/my_app/tree/main/utils
8	Data Feeding the App, including the Download Center	https://github.com/jan399/my_app/tree/main/data