Project Data Job
Finale Report on Data Exploration, Visualization, Preprocessing, Modelling and Modell Results

Ayele T. Sedeke, Frankfurt

Christopher Peter Giesbrecht, Bielefeld

Jan Simon, Hamburg

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

Content

1 Data Exploration an Visualisation

1.1 Subsection 1

Chrsistopher?

1.2 Subsection 2

Chrsistopher?

1.3 Subesection 3

Chrsistopher?

2 Data Preprocessing and Modelling

2.1 Preliminary Remarks

The objective of this section is to document the relevant steps for data preprocessing and modelling as part of the project Data Job-. This project is part of the Data Analyst Course from cohort *B2C | DA Inter | Continu | JAN25 | English*.

The code is derived from [link], which is the last updated notebook before building our app in streamlit [link].

For some code snippets we have used ChatGPT or Gemini as a Co-Pilot. However, we would like to emphasize that the overall structure and the decision we have made as a project group where entirely ours.

2.2 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
```

2.3 Data Structure

2.3.1 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')
```

2.3.2 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 general *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 *Data Science Roles* as a target *y_S* as follows:

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

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)
```

2.4 Data Preprocessing

2.4.1 Preparing encoders

From the data exploration, we know that we work purely on categorical data for both the target and the features. We therefore established a well-prepared encoding process., where we choose

- LabelEncoding *lab* for the targets *y_L* and *y_S*
- OneHotEncoding *ohe* for features without intrinsic order
- OrdinalEncoding *ord* for features with intrinsic order

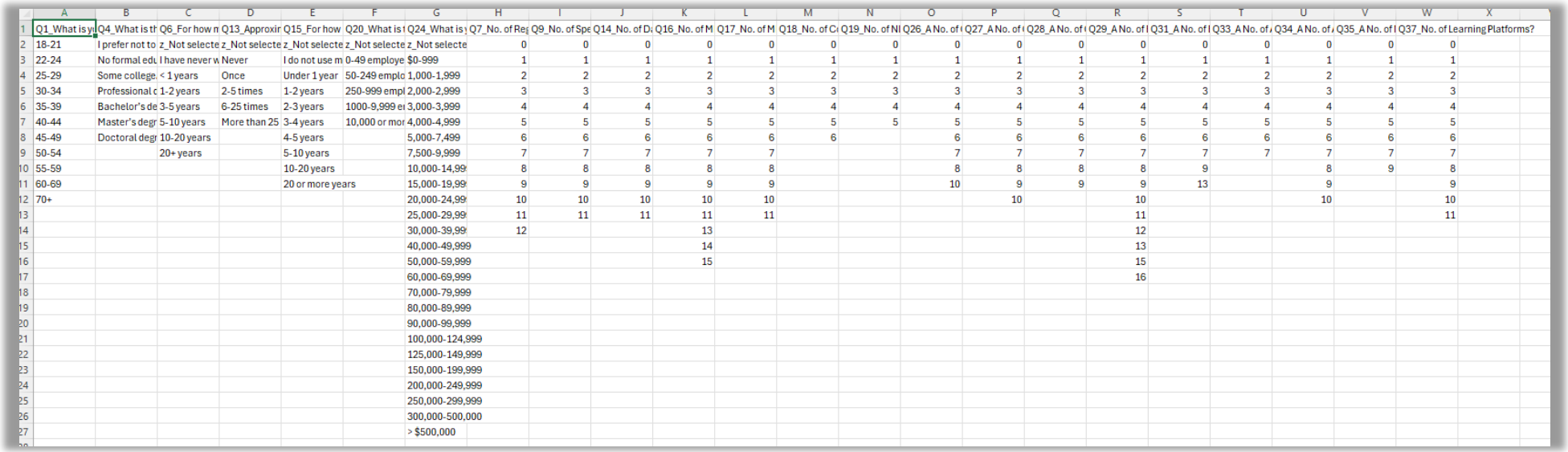
Assigning the encoder type to each column by a dictionary seems time consuming and invites for mistakes. Hence, in a first step, we build up a table *encoder_assigment.csv* in UTF-8 format.

encoder_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
```

For *LabelEncoding*, labels are assigned in alphabetical order. For *OneHotEncoding*, new columns are generated automatically. For ordinal encoding, things are a bit more complicated, as we have to define the intrinsic order for each column that was assigned to *ord*. Taking that into consideration, we generate as second file *unique_with_rank.csv* in UTF-8 format which has the following structure:

unique_with_rank.csv:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	Q1_What is yo	Q4_What is th	Q6_For how m	Q13_Approxir	Q15_For how	Q20_What is t	Q24_What is y	Q7_No. of Re	Q9_No. of Spe	Q14_No. of Da	Q16_No. of M	Q17_No. of M	Q18_No. of Cl	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?
2	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	
3	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	
4	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	
5	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	
6	35-39	Bachelor's de	3-5 years	6-25 times	2-3 years	1000-9,999 e	3,000-3,999	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
7	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	
8	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	
9	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	
10	55-59				10-20 years		10,000-14,99	8	8	8	8	8			8	8	8	8	9		8	9	8	
11	60-69				20 or more years		15,000-19,99	9	9	9	9	9			10	9	9	9	13		9		9	
12	70+						20,000-24,99	10	10	10	10	10				10		10			10		10	
13							25,000-29,99	11	11	11	11	11						11					11	
14							30,000-39,99	12			13							12						
15							40,000-49,999				14							13						
16							50,000-59,999				15							15						
17							60,000-69,999											16						
18							70,000-79,999																	
19							80,000-89,999																	
20							90,000-99,999																	
21							100,000-124,999																	
22							125,000-149,999																	
23							150,000-199,999																	
24							200,000-249,999																	
25							250,000-299,999																	
26							300,000-500,000																	
27							> $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())
```

Now we are prepared for date preprocessing by applying distinct types of encoders.

2.4.2 Encoding for target variables

For simplicity, we do not use *lab_columns* which makes the list redundant. We instantiate and apply the LabelEncoder. However, 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
y_train_L = pd.Series(y_train_L)
y_test_L = pd.Series(y_test_L)
y_train_L = lab_L.fit_transform(y_train_L)
y_test_L = lab_L.transform(y_test_L)

y_train_S = pd.Series(y_train_S)
y_test_S = pd.Series(y_test_S)
y_train_S = lab_S.fit_transform(y_train_S)
y_test_S = lab_S.transform(y_test_S)
```

2.4.3 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)
])
```

2.5 Modelling

2.5.1 Model Choice

We choose three models with for two data sets related to targets *y_L* (career path) and *y_S* (Data Science Roles).

Overview applied Machine Learning Models

Target / Model	Career Path *y_L*	Data Science Roles *y_S*

RandomForest *rfc*		
HistGradientBooster *hgb*		
XGBoost *xgb*		

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, where single models are combined to one stronger model.

Random Forest 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.

2.5.2 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 GradientBooster

```
# HistGradientBoosting pipeline:
# - learning_rate=0.05: shrinkage rate for each tree
```

[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

```
# - 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
    ))
])
```

2.5.3 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 works 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

[4] For results of initial codes, compare notebook RoleRecommender v0.61.ipynb

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.

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
}


# 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
```

2.5.4 Results

We choose classification report for evaluating model performance.

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.

Results for Data Set y_S

Model	Class	Precision	Recall	F1-Score	Support
RandomForest	Data Analyst	0.5593	0.5482	0.5537	301

Model	Class	Precision	Recall	F1-Score	Support
	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), 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, we choose the XGBoost model for both data sets y_L and y_S.

2.5.5 Model Choice for streamlit App

RandomizedSearchCV for XGBoost has shown the following recommendations for y_L and y_S:

```
Best parameters for S (XGB):
{'xgb__reg_lambda': 1.0, 'xgb__n_estimators': 200, 'xgb__min_child_weight': 10,
'xgb__max_depth': 3, 'xgb__learning_rate': 0.1}

Best parameters for L:(XGB) {'xgb__reg_lambda': 2.0, 'xgb__n_estimators': 300,
'xgb__min_child_weight': 20, 'xgb__max_depth': 6, 'xgb__learning_rate': 0.1}
```

We define these parameters as *best_param_L* or *best_param_S*.

```
# Define the optimal hyperparameters found for Dataset L through Randomized Search
best_params_L = {
```

```
    'xgb__reg_lambda': 2.0,
    'xgb__n_estimators': 300,
    'xgb__min_child_weight': 20,
    'xgb__max_depth': 6,
    'xgb__learning_rate': 0.1
}

# Define the optimal hyperparameters found for Dataset S through Randomized Search
best_params_S = {
    'xgb__reg_lambda': 1.0,         # L2 regularization term
    'xgb__n_estimators': 200,       # Number of boosting rounds (trees)
    'xgb__min_child_weight': 10,    # Minimum sum of instance weight (hessian)
needed in a child
    'xgb__max_depth': 3,            # Maximum depth of a tree
    'xgb__learning_rate': 0.1       # Step size shrinkage to prevent overfitting
```

This model will be the basis for our streamlit app. One can see that if the results are printed out by the classification report, they are identical to the results presented in the tables above.

3 } Evaluating Features with SHAP

3.1 Code

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 are the things that matter?

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

SHAP calculates the contribution of each individual feature to a model's prediction — for each instance. From our perspective, this is what we need.

Look on the code for calculating SAHP values for y_L and y_S:

```
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
```

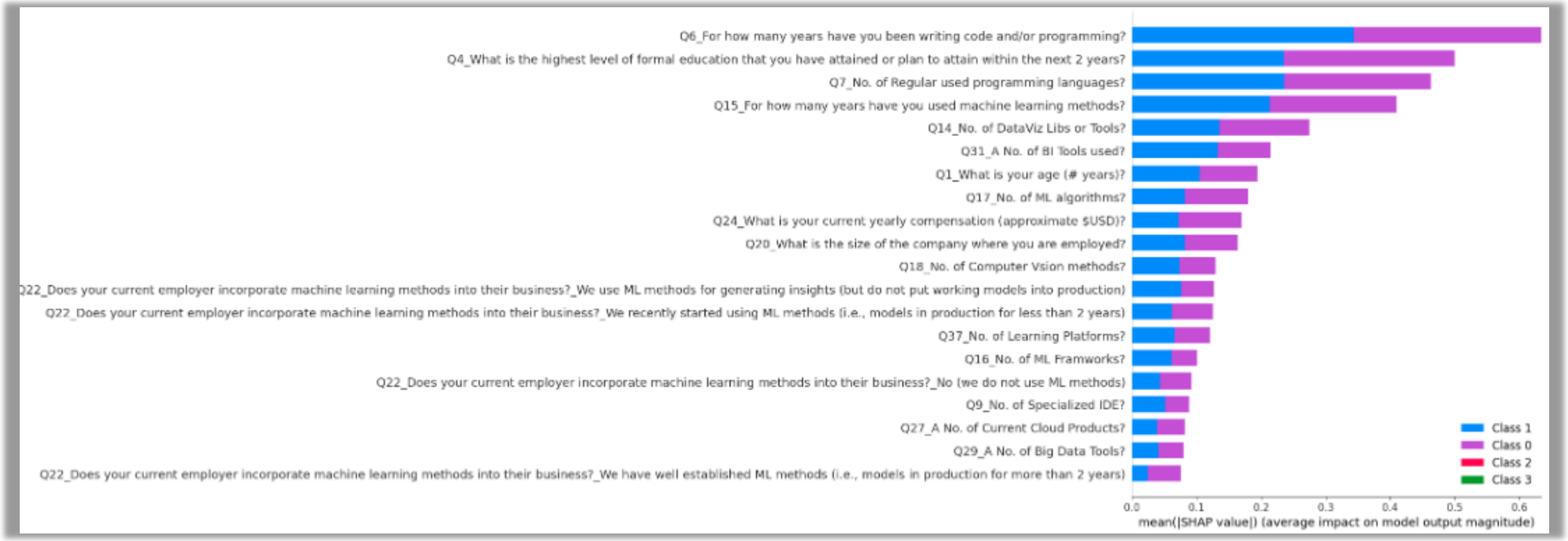
[5] See https://shap.readthedocs.io/en/latest/ for official SHAP Documentation.

```
    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")
```

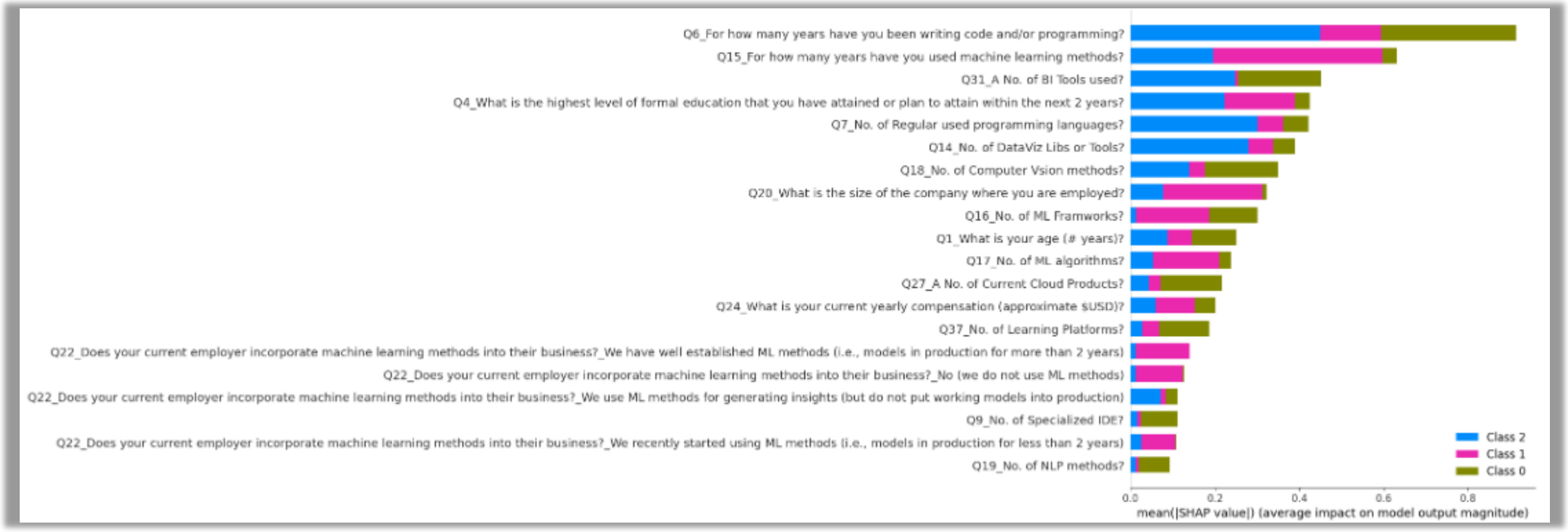
Now look on the next page to see the results:

3.2 Results

Feature importance for y_L (bar plot)



Feature importance for y_S (bar_plot)



Feature importance for y_L (Table, Top 10)

	Mean |SHAP| (0)	Mean |SHAP| (1)
Q6_For how many years have you been writing code and/or programming?	0.29051375	0.3437433
Q4_What is the highest level of formal education that you have attained or plan to attain within the next 2 years?	0.26510268	0.2344554
Q7_No. of Regular used programming languages?	0.22693248	0.23549913
Q15_For how many years have you used machine learning methods?	0.19703034	0.21285132
Q14_No. of DataViz Libs or Tools?	0.13973098	0.13546537
Q17_No. of ML algorithms?	0.098400295	0.08096919
Q24_What is your current yearly compensation (approximate $USD)?	0.096739635	0.07249658
Q1_What is your age (# years)?	0.088588655	0.10509999
Q20_What is the size of the company where you are employed?	0.081823036	0.081667855
Q31_A No. of BI Tools used?	0.081815355	0.13249616

Feature importance for y_S (Table, Top 10)

	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

3.3 Summary

For Career path “Data Science” vs. “Tech”

Feature	DS (0)	Tech (1)	Meaning
Q6 – Years of programming	0.29	0.34	High experience in both, slightly more in Tech roles.
Q4 – Highest education level	0.27	0.23	More relevant for DS; advanced degrees are common.
Q7 – No. of programming languages	0.23	0.24	Both classes use many languages → weak class separation.
Q15 – Years using ML methods	0.20	0.21	ML experience equally present in both.
Q14 – No. of DataViz tools	0.14	0.14	Visualization tools common in both roles.

Interpretation:

Obviously, for both a Tech and a Data Science career education and experience matter. However, feature values are similar across both classes, suggesting that class boundaries are weak.

There is a slight advantage for DS via education and ML use, but not enough to cleanly separate Tech roles.

Overall., this matches with the model performanceasu shown in the classification report: Models predict Data Science well, but struggle with Tech Professionals, reflected in low recall and F1. This might be the result of a pre-selection of questions that were carried out with a strong Data Science focus.

For Data Science and Tech Roles

Feature	DA (0)	DS (1)	SE (2)	Meaning
Q6 – Years of programming	0.32	0.15	**0.45**	Strongly distinguishes SE; higher experience linked to engineering roles.
Q31 – No. of BI tools used	0.20	0.01	0.25	Common for DA and SE, not typical for DS.
Q18 – No. of CV methods used	0.17	0.04	0.14	DS and SE use more computer vision tools.

Feature	DA (0)	DS (1)	SE (2)	Meaning
Q27 – No. of cloud products	0.15	0.03	0.04	More relevant for DA; moderate tech exposure.
Q37 – No. of learning platforms	0.12	0.04	0.03	Higher for DA, indicating self-guided learning background.

Interpretation:

Software Engineers show strong signals through technical tools and skills (programming & BI tools). Data Scientists stand out with ML & CV/NLP focus, though not strongly represented in the Top 5.

Data Analysts lean more toward BI & cloud tools, but overlap with others reduces separability.

Overall., this matches with the model performance shown in the classification report: DA has lowest F1-score due to overlapping patterns; SE and DS are well-separated and better classified.

4 Concluding remarks

Overall approach

Data Visualization

Compostion of Data Sets

Model Selction

Model training

Handling Imbalcens

Evaluating with SHAP

Our own competencies

Appendix

Transforming the original data set

Functions

Complete results of classification reports from RoleRecommender v0.61.ipynb