Predicting academic success of undergraduate students

by Catherine Meng, Jenson Chang, Jingyuan Wang, Siddarth Subrahmanian 2024/11/23

In [1]:
```python
import pandas as pd
import altair as alt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, chi2_contingency, pearsonr
from sklearn import set_config
from itertools import combinations
set_config(display='text')
import pandera as pa
import os
```

In [2]:
```python
chi2_validation = {
    'Admission grade': np.float64(1436.5038834914599),
    'Age at enrollment': np.float64(547.2674289070418),
    'GDP': np.float64(51.965049926087076)}

pearson_validation = {
    'Admission grade_Age at enrollment': -0.02991535667400831,
    'Admission grade_GDP': -0.019519481896555686,
    'Age at enrollment_GDP': -0.06467811172086718}
```

Summary

In this analysis, we attempt to build a classification model using the k-nearest neighbors algorithm to predict student dropout and academic success based on information available at enrollment (including academic path, demographics, and socio-economic factors). Our final classifier performed consistently on unseen test data, achieving a cross-validation training score of 0.71, with a similar test score. Although the model's accuracy is moderate, it performs consistently. Given that the data was collected from a single institution, a larger dataset may be necessary to generalize predictions to other institutions or countries. We believe this model is close to supporting dropout prediction for the institution from which the data was collected, though further research to improve performance and better understand characteristics of incorrectly predicted students would still be beneficial.

Introduction

Higher education institutions worldwide face the ongoing challenge of academic dropout and student failure, which affect not only individual students' futures but also the institution's reputation and resources. The ability to predict and anticipate students' potential difficulties is valuable not only for supporting individual students in achieving their goals but also for institutions aiming to implement strategies that support and guide students who may be at risk of academic failure or dropout.

The goal of this analysis is to help reduce academic dropout and failure in higher education by applying machine learning techniques to identify at-risk students early in their academic journey, enabling institutions to implement targeted support strategies.

Methods

Data

The data set is created by Mónica Vieira Martins, Jorge Machado, Luís Baptista and Valentim Realinho at the Instituto Politécnico de Portalegre (M.V.Martins, D. Tolledo, J. Machado, L. M.T. Baptista, V.Realinho. 2021). It is sourced from UC Irvine's Machine Learning Repository and can be found here. The data contains demographic, enrollment and academic (1st and 2nd semesters) information on the students. Each row in the data set represents a student record. Using these data, a model would be built to predict the academic outcome of the student. There are 36 columns in total.

Analysis

The k-nearest neighbors (k-nn) algorithm was used to build a classification model to predict whether a student is at risk of dropping out. All variables included in the original data set, with the exception of the Course, Nacionality, Gender, Unemployment rate, Inflation rate, GDP, Previous qualification, Mother qualification Mother occupation, Father qualification, Father occupation columns were used to fit the model. Data was split with 80% being partitioned into the training set and 20% being partitioned into the test set. The hyperparameter K was chosen using 5-fold cross validation. All numeric features were standardized just prior to model fitting. We leave the categorical features as they are because they all have integer data type.

Results & Discussion

To look at whether each of the predictors might be useful to predict the tumour class, we plotted the distributions of each predictor from the training data set and coloured the distribution by class (Dropout: blue, Enrolled: orange, and Graduate: red).

In [3]:
```python
# check the file format
def check_csv(file_path):
    """
    Check if the given file is a CSV file by its extension.

    Args:
    file_path (str): Path to the file.

    Returns:
    bool: True if the file is a CSV file, False otherwise.
    """
    # Check if file extension is .csv
    if file_path.endswith(".csv"):
        return True

    # Try to read the file using pandas (this will raise an error if it's not a CSV file)
    try:
        pd.read_csv(file_path)
        return True
    except Exception:
        return False

file_path = "../data/raw/data.csv"
if check_csv(file_path):
    print(f"{file_path} is a CSV file.")
else:
    print(f"{file_path} is not a CSV file.")
```

```
../data/raw/data.csv is a CSV file.
```

In [4]:
```python
# Import data
df = pd.read_csv('../data/raw/data.csv', delimiter = ';')

# Remove extra '\t' from the column name
df.rename(columns = {"Daytime/evening attendance\t" : "Daytime/evening attendance"}, inplace = True)\

# Remove ' from column name to prevent issues with Altair plots
df.columns = df.columns.str.replace("'s", "", regex=False)
```

In [5]:
```python
# validate data before split
schema = pa.DataFrameSchema(
    {
        "Marital status": pa.Column(int, pa.Check.isin([1, 2, 3, 4, 5, 6]),
                                    nullable=True),
        "Application mode": pa.Column(int, pa.Check.isin(
            [1, 2, 5, 7, 10, 15, 16, 17, 18, 26,
             27, 39, 42, 43, 44, 51, 53, 57])),
        "Application order": pa.Column(int, pa.Check.isin(
            [0, 1, 2, 3, 4, 5, 6, 9])),
        "Course": pa.Column(int, pa.Check.isin(
            [33, 171, 8014, 9003, 9070, 9085, 9119, 9130, 9147, 9238,
             9254, 9500, 9556, 9670, 9773, 9853, 9991]), nullable=True),
        "Daytime/evening attendance": pa.Column(int, pa.Check.isin(
            [0, 1]), nullable=True),
        "Previous qualification": pa.Column(int, pa.Check.isin(
            [1, 2, 3, 4, 5, 6, 9, 10, 12, 14, 15, 19, 38, 39, 40, 42, 43])),
        "Previous qualification (grade)": pa.Column(float, pa.Check.between(
            0, 200)),
        "Nacionality": pa.Column(int, pa.Check.isin(
            [1, 2, 6, 11, 13, 14, 17, 21, 22, 24, 25, 26, 32, 41, 62,
             100, 101, 103, 105, 108, 109]), nullable=True),
        "Mother qualification": pa.Column(int, pa.Check.isin(
            [1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 14, 18, 19, 22, 26, 27, 29, 30,
             34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]), nullable=True),
        "Father qualification": pa.Column(int, pa.Check.isin(
            [1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 18, 19, 20, 22, 25,
             26, 27,29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
             43, 44]), nullable=True),
        "Mother occupation": pa.Column(int, pa.Check.isin(
            [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 90, 99, 122, 123, 125, 131,
             132, 134, 141, 143, 144, 151, 152, 153, 171, 173, 175, 191,
             192, 193, 194]), nullable=True),
        "Father occupation": pa.Column(int, pa.Check.isin(
            [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 90, 99, 101, 102, 103,
             112, 114, 121, 122, 123, 124, 131, 132, 134, 135, 141, 143,
             144, 151, 152, 153, 154, 161, 163, 171, 172, 174, 175,
             181, 182, 183, 192, 193, 194, 195]), nullable=True),
        "Admission grade": pa.Column(float, pa.Check.between(0, 200),
                                     nullable=True),
        "Displaced": pa.Column(int, pa.Check.isin([0, 1]), nullable=True),
        "Educational special needs": pa.Column(int, pa.Check.isin([0, 1]),
                                               nullable=True),
        "Debtor": pa.Column(int, pa.Check.isin([0, 1]), nullable=True),
        "Tuition fees up to date": pa.Column(int, pa.Check.isin([0, 1]),
                                             nullable=True),
        "Gender": pa.Column(int, pa.Check.isin([0, 1]), nullable=True),
        "Scholarship holder": pa.Column(int, pa.Check.isin([0, 1]),
                                        nullable=True),
        "Age at enrollment": pa.Column(int, pa.Check.between(15, 100),
                                       nullable=True),
        "International": pa.Column(int, pa.Check.isin([0, 1]),
                                   nullable=True),
        "Curricular units 1st sem (credited)": pa.Column(int,
                                                         nullable=True),
        "Curricular units 1st sem (enrolled)": pa.Column(int,
                                                         nullable=True),
        "Curricular units 1st sem (evaluations)": pa.Column(int,
                                                            nullable=True),
        "Curricular units 1st sem (approved)": pa.Column(int,
                                                         nullable=True),
        "Curricular units 1st sem (grade)": pa.Column(
            float, pa.Check.between(0, 20), nullable=True),
        "Curricular units 1st sem (without evaluations)": pa.Column(
            int, nullable=True),
        "Curricular units 2nd sem (credited)": pa.Column(
            int, nullable=True),
        "Curricular units 2nd sem (enrolled)": pa.Column(
            int, nullable=True),
        "Curricular units 2nd sem (evaluations)": pa.Column(
            int, nullable=True),
        "Curricular units 2nd sem (approved)": pa.Column(int, nullable=True),
        "Curricular units 2nd sem (grade)": pa.Column(
            float, pa.Check.between(0, 20), nullable=True),
        "Curricular units 2nd sem (without evaluations)": pa.Column(
            int, nullable=True),
        "Unemployment rate": pa.Column(float, nullable=True),
        "Inflation rate": pa.Column(float, nullable=True),
        "GDP": pa.Column(float, nullable=True),
        "Target": pa.Column(str, pa.Check.isin(
            ['Dropout', 'Enrolled', 'Graduate']))
    },
    checks=[
        pa.Check(lambda df: ~df.duplicated().any(),
                 error="Duplicate rows found."),
        pa.Check(lambda df: ~(df.isna().all(axis=1)).any(),
                 error="Empty rows found.")
    ]
)

schema.validate(df, lazy=True)
```

Out[5]:

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Previous qualification (grade)	Nacionality	Mother qualification	Father qualification	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Curricular units 2nd sem (evaluations)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Curricular units 2nd sem (without evaluations)	Unemployment rate	Inflation rate	GDP	Target
0	1	17	5	171	1	1	122.0	1	19	12	...	0	0	0	0	0.000000	0	10.8	1.4	1.74	Dropout
1	1	15	1	9254	1	1	160.0	1	1	3	...	0	6	6	6	13.666667	0	13.9	-0.3	0.79	Graduate
2	1	1	5	9070	1	1	122.0	1	37	37	...	0	6	0	0	0.000000	0	10.8	1.4	1.74	Dropout
3	1	17	2	9773	1	1	122.0	1	38	37	...	0	6	10	5	12.400000	0	9.4	-0.8	-3.12	Graduate
4	2	39	1	8014	0	1	100.0	1	37	38	...	0	6	6	6	13.000000	0	13.9	-0.3	0.79	Graduate
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4419	1	1	6	9773	1	1	125.0	1	1	1	...	0	6	8	5	12.666667	0	15.5	2.8	-4.06	Graduate
4420	1	1	2	9773	1	1	120.0	105	1	1	...	0	6	6	2	11.000000	0	11.1	0.6	2.02	Dropout
4421	1	1	1	9500	1	1	154.0	1	37	37	...	0	8	9	1	13.500000	0	13.9	-0.3	0.79	Dropout
4422	1	1	1	9147	1	1	180.0	1	37	37	...	0	5	6	5	12.000000	0	9.4	-0.8	-3.12	Graduate
4423	1	10	1	9773	1	1	152.0	22	38	37	...	0	6	6	6	13.000000	0	12.7	3.7	-1.70	Graduate

4424 rows × 37 columns

In [6]:
```python
#Check Target/response variable follows expected distribution
# Calculate observed frequencies in the 'Target' column
observed_frequencies = df['Target'].value_counts()
print("Observed Frequencies:\n", observed_frequencies)

# Calculate total number of students (observations)
total_students = len(df)
# Define the expected frequencies for a uniform distribution
expected_frequencies = [total_students / len(observed_frequencies)] * len(observed_frequencies)
print("\nExpected Frequencies (Uniform Distribution):", expected_frequencies)

from scipy.stats import chisquare

# Perform the Chi-Square goodness-of-fit test
chi2_stat, p_value = chisquare(observed_frequencies, expected_frequencies)

print(f"\nChi2 Stat: {chi2_stat}")
print(f"P-Value: {p_value}")
```

```
Observed Frequencies:
 Target
Graduate    2209
Dropout     1421
Enrolled     794
Name: count, dtype: int64

Expected Frequencies (Uniform Distribution): [1474.6666666666667, 1474.6666666666667, 1474.6666666666667]

Chi2 Stat: 681.8033453887884
P-Value: 8.877377277253749e-149
```

In [7]:
```python
# Checking for anomalous correlations between target variable and a subset of features variables
chi2_results = {}
features = ['Admission grade', 'Age at enrollment', 'GDP']

# Perform chi-square test for each feature
for feature in features:
    # Create contingency table
    contingency_table = pd.crosstab(df[feature], df['Target'])

    # Perform chi-square test
    chi2_results[feature] = chi2_contingency(contingency_table).statistic

# Check if the test statistic is approximately equal to pre-calculated values
for feature in chi2_results:
    np.testing.assert_almost_equal(chi2_results[feature], chi2_validation[feature])
```

In [8]:
```python
# Checking for anomalous correlations between features
pearson_results = {}
feature_pairs = list(combinations(features, 2))

# Perform Pearson test on each feature pairs
for f1, f2 in feature_pairs:
    pearson_results[f1+'_'+f2] = pearsonr(df[f1], df[f2]).statistic

# Check if the test statistic is approximately equal to pre-calculated values
for pair in pearson_results:
    np.testing.assert_almost_equal(pearson_results[pair], pearson_validation[pair])
```

In [9]:
```python
train, test = train_test_split(df, train_size = 0.8, random_state = 123)
```

In [10]:
```python
train.nunique()
```

Out[10]:
```
Marital status                                      6
Application mode                                   17
Application order                                   7
Course                                             17
Daytime/evening attendance                          2
Previous qualification                             17
Previous qualification (grade)                     97
Nacionality                                        21
Mother qualification                               29
Father qualification                               30
Mother occupation                                  31
Father occupation                                  44
Admission grade                                   585
Displaced                                           2
Educational special needs                           2
Debtor                                              2
Tuition fees up to date                             2
Gender                                              2
Scholarship holder                                  2
Age at enrollment                                  46
International                                       2
Curricular units 1st sem (credited)                21
Curricular units 1st sem (enrolled)                22
Curricular units 1st sem (evaluations)             34
Curricular units 1st sem (approved)                22
Curricular units 1st sem (grade)                  693
Curricular units 1st sem (without evaluations)     11
Curricular units 2nd sem (credited)                19
Curricular units 2nd sem (enrolled)                21
Curricular units 2nd sem (evaluations)             29
Curricular units 2nd sem (approved)                20
Curricular units 2nd sem (grade)                  685
Curricular units 2nd sem (without evaluations)     10
Unemployment rate                                  10
Inflation rate                                      9
GDP                                                10
Target                                              3
dtype: int64
```

In [11]:
```python
# Group feature types based on feature description from source data
categorical_features = ["Application order", "Course", "Nacionality", "Gender",
                        "Marital status", "Application mode", "Daytime/evening attendance",
                        "Previous qualification", "Mother qualification",  "Mother occupation",
                        "Father qualification", "Father occupation", "Displaced",
                        "Educational special needs", "Debtor", "Tuition fees up to date",
                        "Scholarship holder", "International"]

numeric_features = ["Previous qualification (grade)", "Admission grade", "Age at enrollment",
                    "Curricular units 1st sem (credited)", "Curricular units 1st sem (enrolled)",
                    "Curricular units 1st sem (evaluations)", "Curricular units 1st sem (approved)",
                    "Curricular units 1st sem (grade)", "Curricular units 1st sem (without evaluations)",
                    "Curricular units 2nd sem (credited)", "Curricular units 2nd sem (enrolled)",
                    "Curricular units 2nd sem (evaluations)", "Curricular units 2nd sem (approved)",
                    "Curricular units 2nd sem (grade)", "Curricular units 2nd sem (without evaluations)",
                    "Unemployment rate", "Inflation rate", "GDP"]
```

In [12]:
```python
# Plot numerical features
alt.Chart(train).mark_rect().encode(
    x = alt.X(alt.repeat()).bin(maxbins=30),
    y = alt.Y('Target', title=None),
    color = alt.Color('count()').legend(orient="top")
).properties(
    width = 180
).repeat(
    numeric_features,
    columns = 2
)
```

Out[12]:

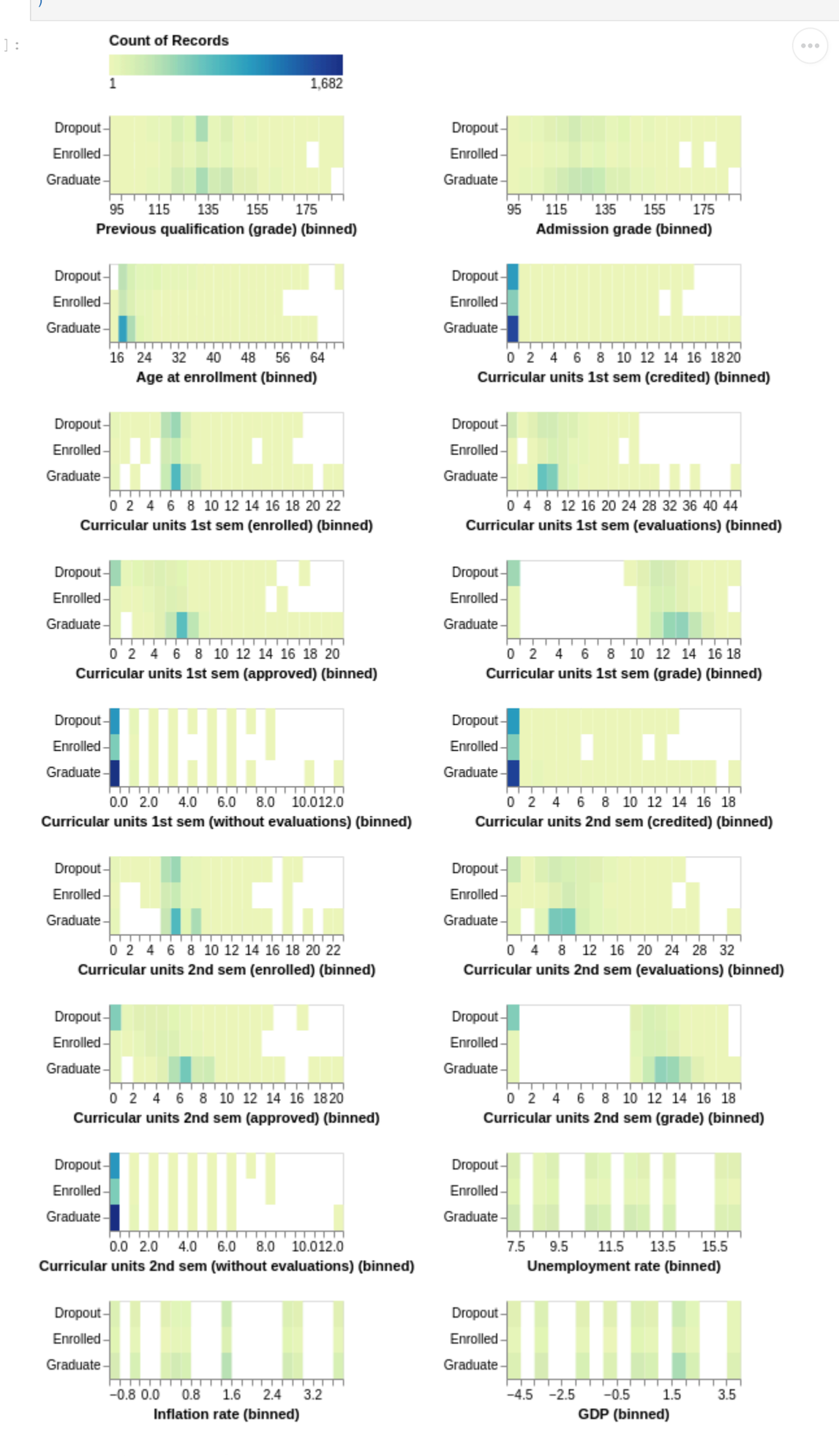

Figure 1. Distribution of Numerical Variable per Academic Outcome

In [13]:
```python
# Plot categorical and boolean features
alt.Chart(train).mark_bar().encode(
    x = alt.X(alt.repeat()).type('quantitative'),
    y = 'count()',
    column = alt.Column('Target', title = None),
    color = alt.Color('Target', legend = None)
).properties(
    width = 120,
    height = 80
).resolve_scale(
    y = 'independent'
).repeat(
    categorical_features,
    columns = 1
)
```

Out[13]:

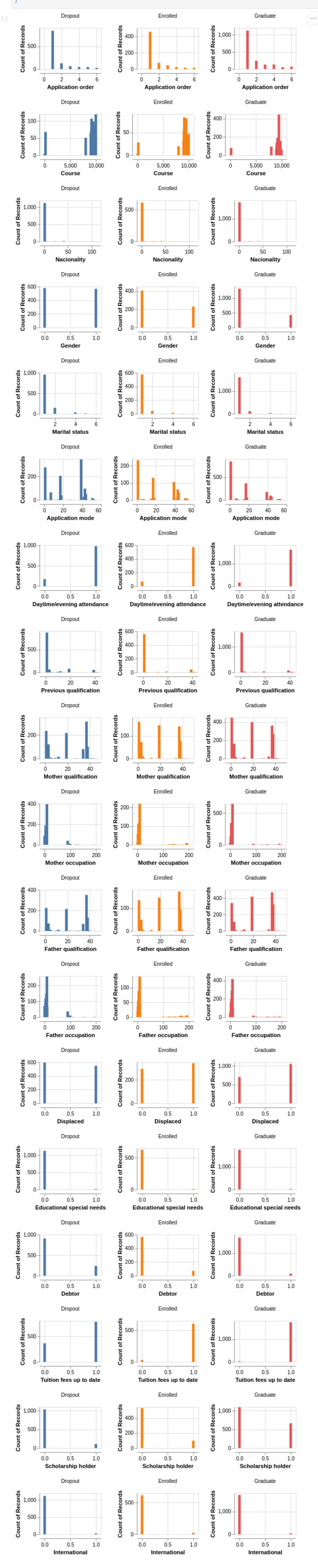

Figure 2. Distribution of Categorical Variable per Academic Outcome

In [14]:
```python
# Features to drop from model
drop_features = ["Course", "Nacionality", "Gender", "Unemployment rate",
                 "Inflation rate", "GDP", "Previous qualification", "Mother qualification",
                 "Mother occupation", "Father qualification", "Father occupation"]
```

Reason for dropping:

- `Course` only captures 17 different courses and does not generalize the real world well.
- `Nacionality` and `Gender` are removed to avoid racial and gender bias
- In Figure 1, although `Unemployment rate`, `Inflation rate` and `GDP` are continous values, they each have less than 10 unique values out of 3000+ rows. This doesn't provide enough range to generalize the problem.
- In Figure 2, `Previous qualification`, `Mother qualification`, `Mother occupation`, `Father qualification` and `Father occupation` have cluster patterns but it's unclear what the pattern represents since the ranking of education levels are arbitrary. E.g. "5 - Higher Education - Doctorate" is ranked higher than "1 - Secondary Education" but lower than "10 - 11th Year of Schooling - Not Completed". The source data website provides description on each ranking.

In [15]:
```python
X_train = train.drop(columns=['Target'])
y_train = train['Target']
X_test = test.drop(columns=['Target'])
y_test = test['Target']

# Make preprocessor
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    ('drop', drop_features)
)
```

In [16]:
```python
# Build the pipeline, use knn to train the model

my_pipeline = make_pipeline(
    preprocessor,
    KNeighborsClassifier(n_neighbors=5)
)

my_pipeline.fit(X_train, y_train)
```

Out[16]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('standardscaler',
                                                  StandardScaler(),
                                                  ['Previous qualification '
                                                   '(grade)',
                                                   'Admission grade',
                                                   'Age at enrollment',
                                                   'Curricular units 1st sem '
                                                   '(credited)',
                                                   'Curricular units 1st sem '
                                                   '(enrolled)',
                                                   'Curricular units 1st sem '
                                                   '(evaluations)',
                                                   'Curricular units 1st sem '
                                                   '(approved)',
                                                   'Curricular units 1st sem '
                                                   '(grade...
                                                   'Curricular units 2nd sem '
                                                   '(grade)',
                                                   'Curricular units 2nd sem '
                                                   '(without evaluations)',
                                                   'Unemployment rate',
                                                   'Inflation rate', 'GDP']),
                                                 ('drop', 'drop',
                                                  ['Course', 'Nacionality',
                                                   'Gender',
                                                   'Unemployment rate',
                                                   'Inflation rate', 'GDP',
                                                   'Previous qualification',
                                                   'Mother qualification',
                                                   'Mother occupation',
                                                   'Father qualification',
                                                   'Father occupation'])])),
                ('kneighborsclassifier', KNeighborsClassifier())])
```

In [17]:
```python
# Use RandomizedSearchCV to tune hyperparameters

param_distributions = {
    'kneighborsclassifier__n_neighbors': randint(1, 30)
}

random_search = RandomizedSearchCV(
    estimator=my_pipeline,
    param_distributions=param_distributions,
    n_iter=50,
    cv=5,
    scoring='accuracy',
    random_state=42,
    n_jobs=-1
)

random_search.fit(X_train, y_train)

print("Best Parameters:", random_search.best_params_)
print("Best CV Accuracy:", random_search.best_score_)
```

```
Best Parameters: {'kneighborsclassifier__n_neighbors': 17}
Best CV Accuracy: 0.7143280671892855
```

We utilized the KNN to train the dataset and employed RandomizedSearchCV to fine-tune the hyperparameters. Based on the results, the optimal hyperparameter value is k=12, achieving a best cross-validation score of 0.71. Using this value, we retrained the model and evaluated its performance on the test set, obtaining a final test score of 0.71.

In [18]:
```python
my_pipeline_best = make_pipeline(
    preprocessor,
    KNeighborsClassifier(n_neighbors=12)
)

my_pipeline_best.fit(X_train, y_train)

test_score = my_pipeline_best.score(X_test, y_test)
print(f"Test accuracy: {test_score}")
```

```
Test accuracy: 0.7129943502824859
```

References

1. Martins, M. V., D. Tolledo, J. Machado, L. M. T. Baptista, and V. Realinho. "Early Prediction of Student's Performance in Higher Education: A Case Study." In Trends and Applications in Information Systems and Technologies, vol. 1, Advances in Intelligent Systems and Computing series. Springer, 2021. https://archive.ics.uci.edu/dataset/697/predict+students+dropout+and+academic+success. DOI: 10.1007/978-3-030-72657-7_16.
2. Scikit-learn Developers. "Scikit-learn: Machine Learning in Python – API Reference Documentation." Accessed November 20, 2024. https://scikit-learn.org/dev/api/index.html.
3. Vega-Altair Developers. "Altair User Guide." Accessed November 20, 2024. https://altair-viz.github.io/user_guide/data.html#.
4. Timbers, T., J. Ostblom, and M. Lee. "Predicting Breast Cancer from Digitized Images of Breast Mass." Accessed November 21, 2024. https://github.com/ttimbers/breast-cancer-predictor/blob/0.0.1/notebooks/breast_cancer_predictor_report.ipynb.