Predicting Income Levels Using the Adult Census Dataset

In [1]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import altair as alt
alt.data_transformers.enable("vegafusion")
```

Out[1]:
```
DataTransformerRegistry.enable('vegafusion')
```

Summary

In this project, we examine whether demographic and employment-related factors can be used to predict if an individual's annual income exceeds $50K, using the Adult Census Income dataset. After cleaning the data and preparing the features, we compared several classification models, including logistic regression, SVM, and random forest. Logistic regression provided the strongest baseline performance, and after further tuning, the final model reached an accuracy of 0.80 and a weighted F1 score of 0.83 on the test set. While the model performs well on the majority class, it continues to face challenges in identifying higher-income individuals due to class imbalance. Overall, our findings suggest that income prediction is feasible, though additional techniques—such as resampling or more advanced models—may help improve performance on the minority class.

Introduction

Income inequality has reached a record high in Canada within the recent years (Statistics Canada, 2025). Increase in Canada's wealth gap has been shown to perpetuate poverty cycles, affect economic mobility, and limit socioeconomic opportunities (Connoly and Haeck, 2024). Understanding the individual factors associated with income level could help governments and policy makers better address these issues.

In this project, we seek to build a machine learning model capable of accurately predicting whether an individual's income exceeds 50,000. To do such classification, our model will consider factors such as age, education level, occupation, marital status, and gender just to name a few. We believe an income classification model to be valuable because of its ability to identify broad patterns, and highlight features that contribute most to high income status. These insights could also shed light on structural inequalities, economic trends, and educational impacts. Although a human could perform this task reasonably accurately, a

machine learning model enables scalability, and limits the biases present in human decision making. Therefore, our goal is to build a model that uses relevant features to predict income status, and evaluate how it performs on such tasks.

Methods

Data

The data set used in this project is the Adult census income dataset. It was originally extracted from the 1994 U.S. Census database by Barry Becker. In 1996, the dataset was donated to the UCI Machine Learning Repository with the collaboration of Ronny Kohavi (Becker and Kohavi 1996), the dataset can be found here: https://archive.ics.uci.edu/dataset/2/adult. The dataset contains 48,842 rows and 14 columns. We split the data into a 40% training set and a 60% test set. Each row in the data set represents an individual's personal and demographic information, including the income class label (whether their annual income exceeded $50K or not) and attributes such as age, education level, occupation, and hours per week.

To examine whether each predictor may help discriminate between the two income groups (<=50K vs. >50K), we first explored the numeric and categorical features separately.

In [2]:
```
adult = fetch_ucirepo(id=2)
X = adult.data.features
y = adult.data.targets
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife
...	...	...	...	...	...	...	...	...
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband

48842 rows × 14 columns

In [4]:
```
whole_df = pd.concat([X,y],axis=1)
train_df, test_df = train_test_split(whole_df, random_state=123, test_size=0.6)
train_df = train_df.replace("?", np.nan)
test_df = test_df.replace("?", np.nan)
```

In [5]:
```
train_df.head()
```

Out[5]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship
4044	19	Private	246652	Some-college	10	Never-married	Sales	Own-child
34649	20	NaN	323309	HS-grad	9	Never-married	NaN	Own-child
48375	51	Self-emp-inc	318351	HS-grad	9	Divorced	Adm-clerical	Not-in-family
37576	18	Self-emp-not-inc	161245	12th	8	Never-married	Farming-fishing	Own-child
8211	73	Private	301210	1st-4th	2	Married-civ-spouse	Transport-moving	Husband

Summary of the Training Data

By checking the summary table below, we observe that there are several missing values in features such as workclass, occupation, and native-country. These missing values will be handled during data preprocessing using an appropriate imputation strategy.

Additionally, the target variable income contains inconsistencies in spelling (e.g., '<50K' vs. '<50K.'), so we cleaned these values to ensure a consistent label representation.

In [6]:
```
summary = pd.DataFrame({
    "unique value count": train_df.nunique(),
    "null count": train_df.isnull().sum()
}).T
summary
```

Out[6]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship
unique value count	72	8	14821	16	16	7	14	6
null count	0	1151	0	0	0	0	1153	0

In [7]:
```
train_df["income"].unique()
```

Out[7]:
```
array(['<=50K', '<=50K.', '>50K', '>50K.'], dtype=object)
```

In [26]:
```
train_df["income"] = train_df["income"].str.replace('.', '')
```

Numeric Features

In [9]:
```
train_df.describe(include="number").round(2)
```

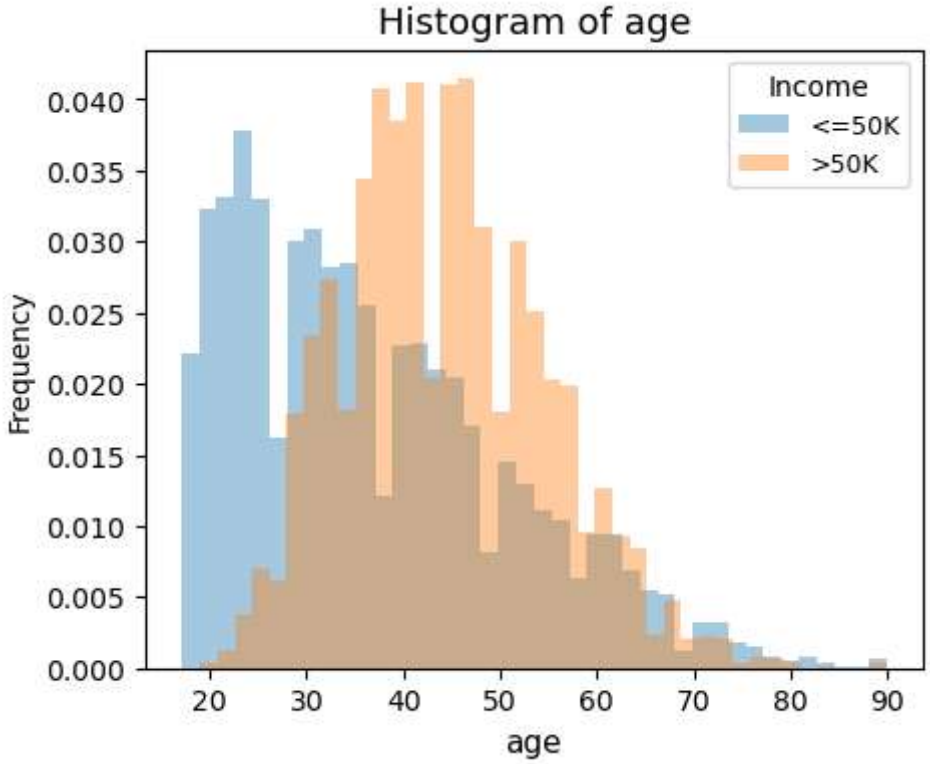
Out[9]:

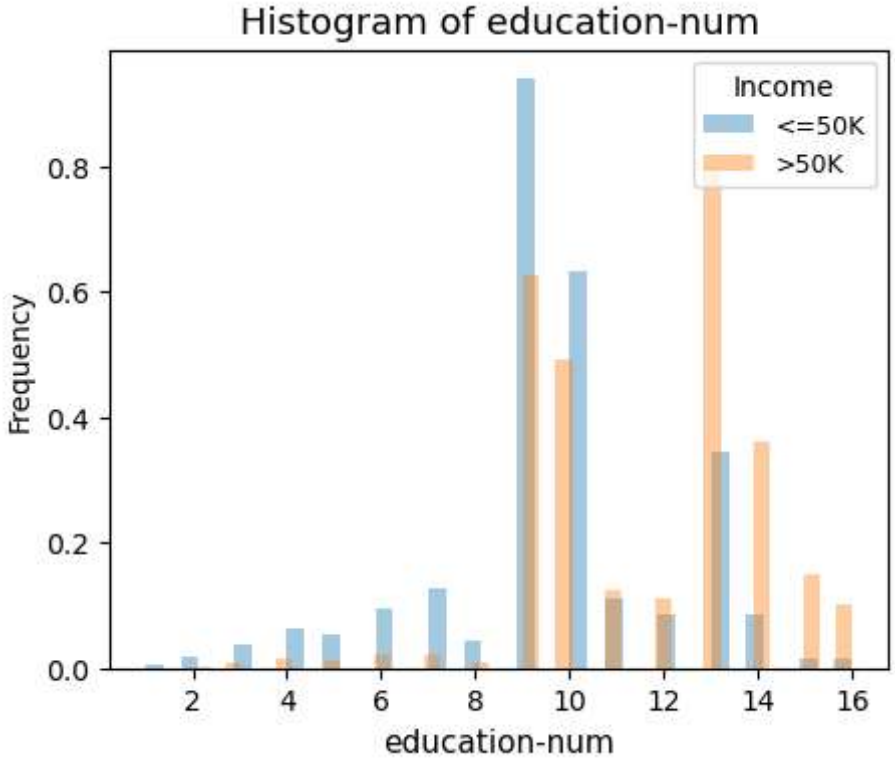
	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	19536.00	19536.00	19536.00	19536.00	19536.00	19536.00
mean	38.67	190134.46	10.08	1067.70	91.99	40.38
std	13.78	106274.65	2.59	7461.83	413.19	12.38
min	17.00	13769.00	1.00	0.00	0.00	1.00
25%	28.00	117496.00	9.00	0.00	0.00	40.00
50%	37.00	178312.00	10.00	0.00	0.00	40.00
75%	48.00	238589.25	12.25	0.00	0.00	45.00
max	90.00	1490400.00	16.00	99999.00	4356.00	99.00

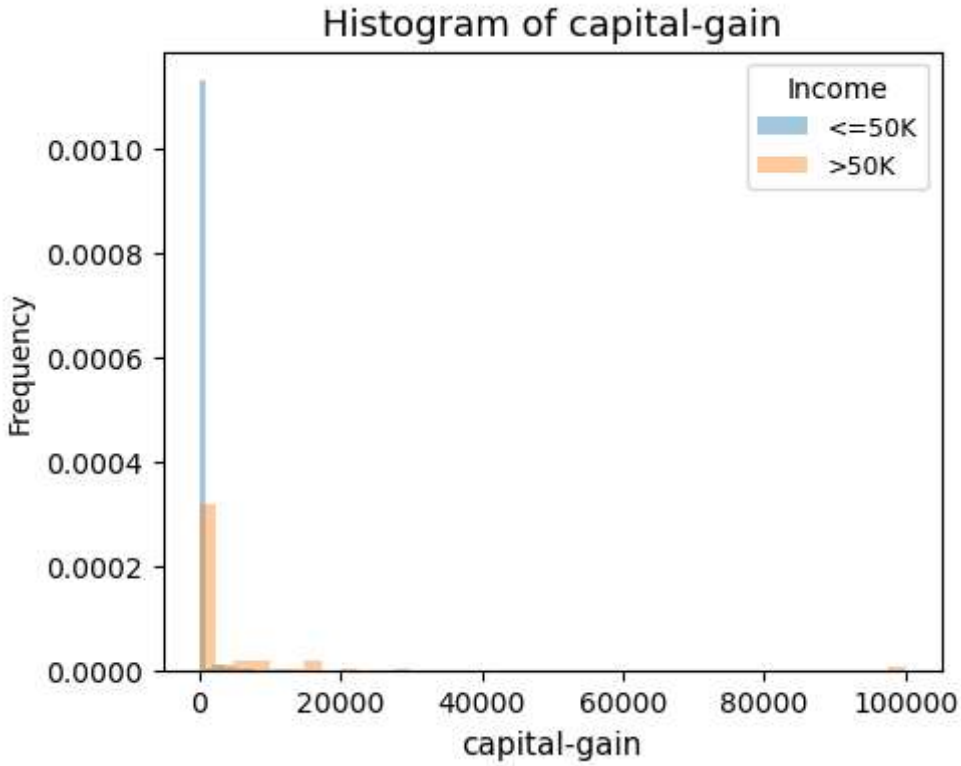
According to the summary table, the numeric features vary widely in both range and scale. For example, hours-per-week ranges from 1 to 99, whereas capital-gain ranges from 0 to 99,999. This large difference in magnitude makes it necessary to apply a StandardScaler to normalize the features so that models relying on distance can compare them fairly.

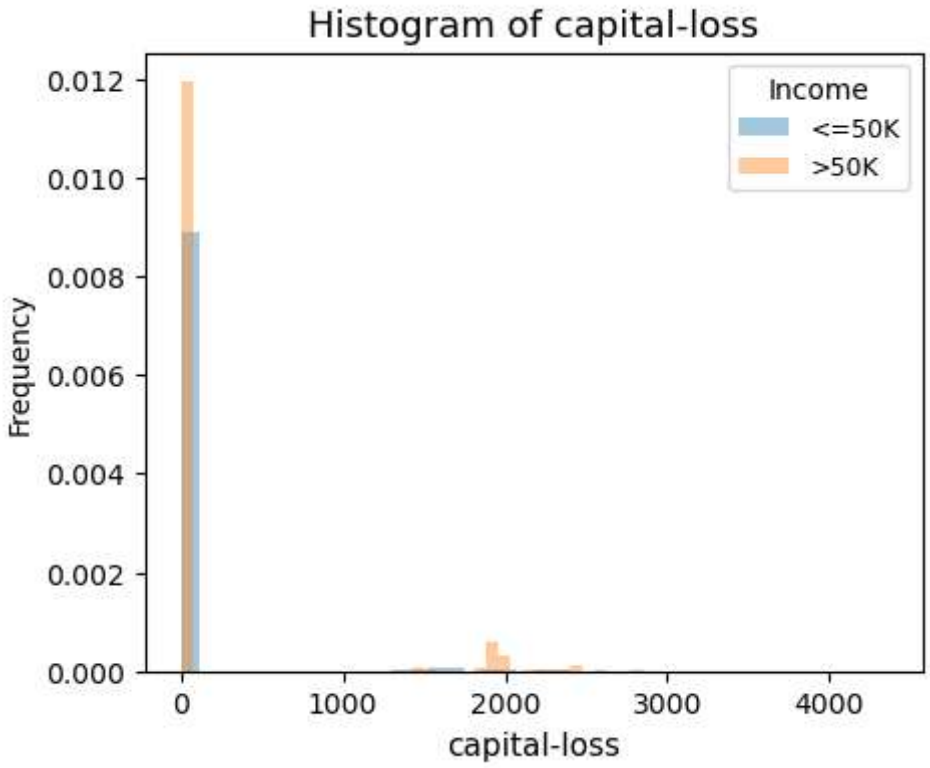
Additionally, some variables such as capital-gain and capital-loss are extremely skewed, with most values concentrated near zero. We are going to the compare the distribution of the numeric features differ between the two target classes income <=50K and >50K.

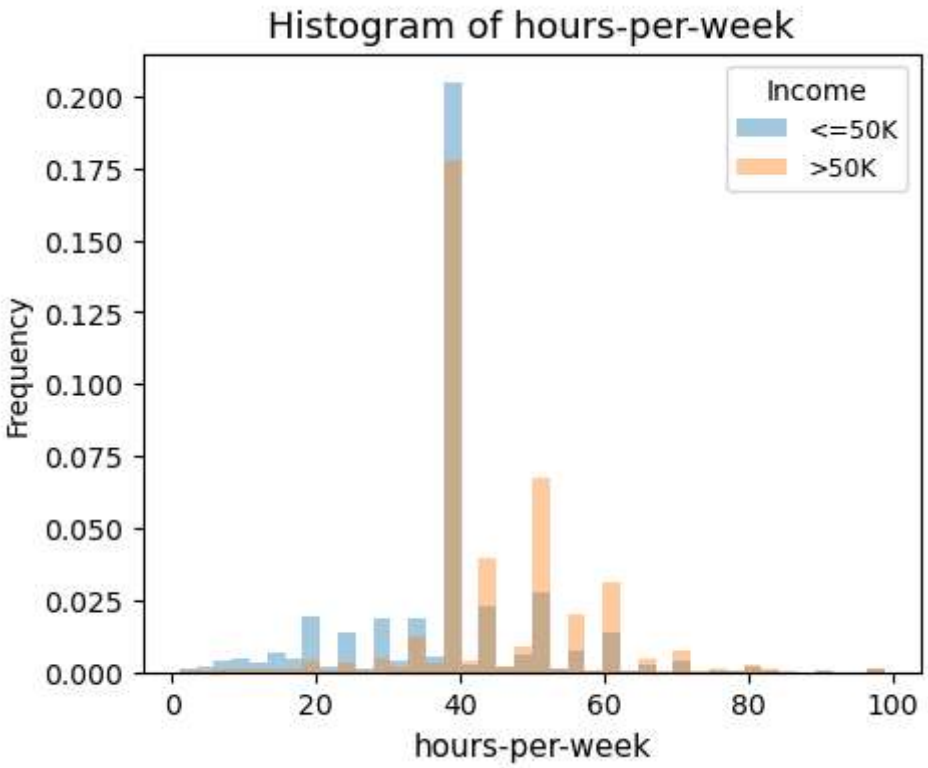
In [10]:
```
numeric_col_list = ['age',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week']

for feat in numeric_col_list:
    plt.figure(figsize=(5, 4))
    train_df.groupby("income")[feat].plot.hist(bins=40, alpha=0.4, legend=True, den
    plt.xlabel(feat, fontsize=11)
    plt.title(f"Histogram of {feat}", fontsize=13)
    plt.legend(title="Income", fontsize=9, title_fontsize=10, loc="upper right")


    plt.show()
```


Histogram of age
Income
<=50K
>50K
Frequency
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
30
40
50
60
70
80
90
age


Histogram of education-num
Income
<=50K
>50K
Frequency
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12
14
16
education-num


Histogram of capital-gain
Income
<=50K
>50K
Frequency
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
20000
40000
60000
80000
100000
capital-gain


Histogram of capital-loss
Income
<=50K
>50K
Frequency
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
1000
2000
3000
4000
capital-loss



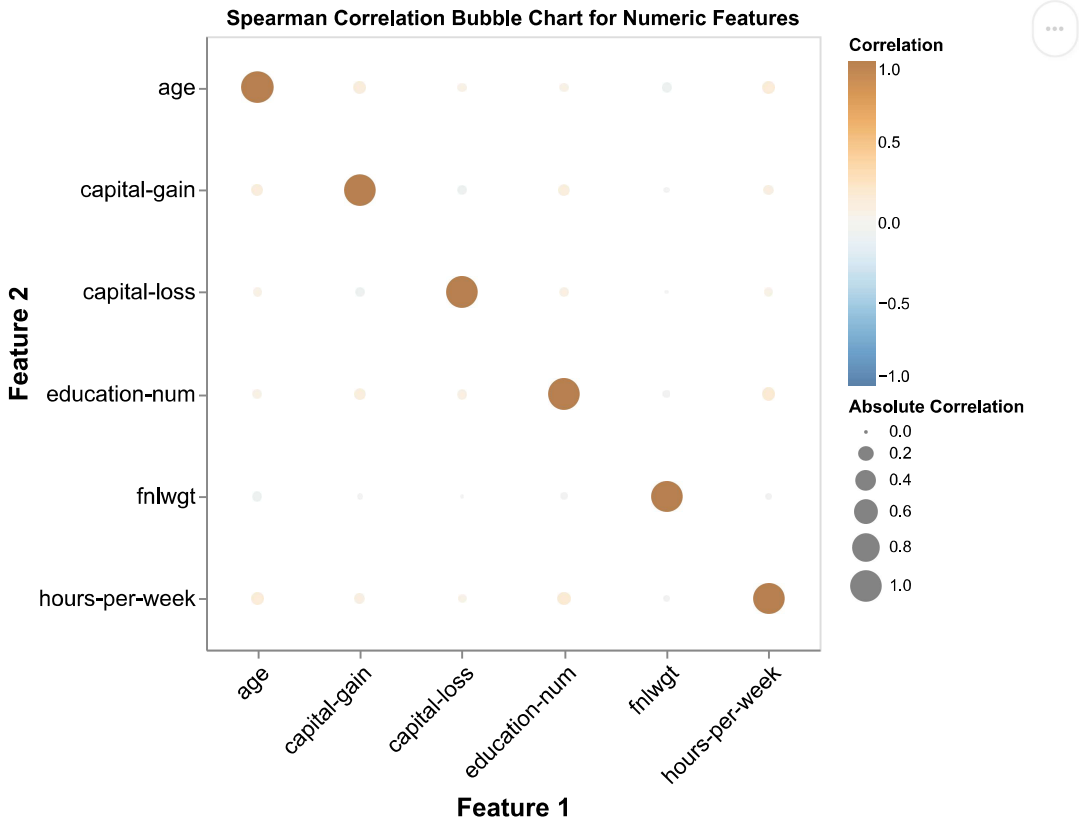
Insights for Histogram of Numeric Features:

- Age: The two income groups differ clearly in their age distributions. Individuals earning >50K tend to be older, most commonly between ages 40 and 55, while the <=50K group is more spread out and skewed younger. This suggests that higher income may be linked to greater work experience.
- Education-num: Individuals earning >50K generally have higher education levels, indicating that formal education is an important factor associated with income.
- Capital-gain: The distribution is strongly right-skewed, with most values at zero for both groups. Non-zero values appear more often in the >50K group, and there is an extreme outlier near 100,000.
- Capital-loss: The pattern is similar to capital-gain, with almost all values at zero and slightly more non-zero observations in the >50K group.
- Hours-per-week: Individuals earning >50K tend to work longer hours, showing a clear shift toward extended working time.

In [11]:
```
corr_df = train_df.corr(method='spearman', numeric_only=True).stack().reset_index()
corr_df.columns = ['Feature 1', 'Feature 2', 'Correlation']
corr_df['Absolute Correlation'] = corr_df['Correlation'].abs()

alt.Chart(corr_df).mark_circle().encode(
    x=alt.X('Feature 1:N', title='Feature 1',axis=alt.Axis(labelAngle=-45)),
    y=alt.Y('Feature 2:N', title='Feature 2'),
```

```
    size=alt.Size('Absolute Correlation:Q').scale(domain=(0, 1)),
    color=alt.Color('Correlation:Q').scale(
        scheme='blueorange',
        domain=(-1, 1)
    ),
    tooltip=[
        alt.Tooltip('Feature 1'),
        alt.Tooltip('Feature 2'),
        alt.Tooltip('Correlation', format='.3f', title='Spearman Correlation'),
        alt.Tooltip('Absolute Correlation', format='.3f', title='Absolute Correlati
    ]
).properties(
    width=380,
    height=380,
    title='Spearman Correlation Bubble Chart for Numeric Features'
).configure_axis(
    labelFontSize=14,
    titleFontSize=16
)
```

Out[11]:



In the Spearman Correlation Bubble Chart for numeric features, we can see that none of the numeric variables are strongly correlated with each other. The highest correlation is between education level and hours per week, at approximately 0.14, which is still very weak. Therefore, multicollinearity is not a concern for these features.

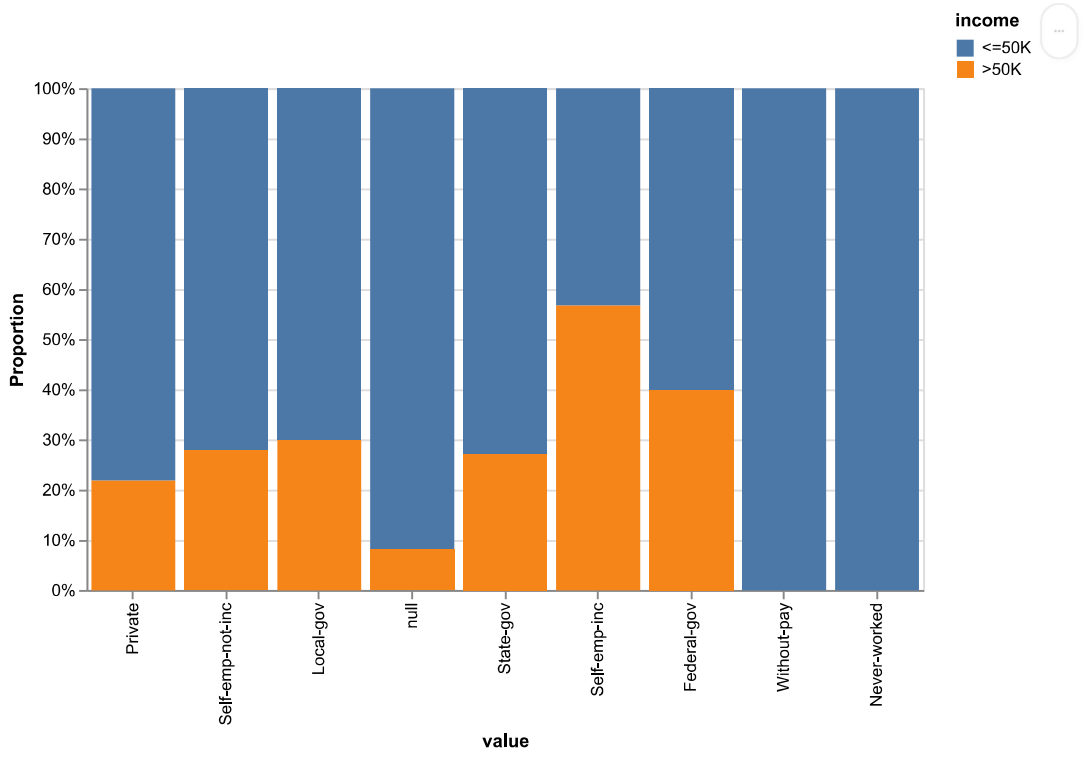
Categorical Features

For categorical features, we produced normalized stacked bar charts to visualize how proportions of each income group differ across categories.

In [13]:
```
categorical_cols = train_df.select_dtypes(exclude=["number"]).columns.drop("income"
```

In [15]:
```
radio = alt.selection_point(fields=["feature"],
    bind=alt.binding_radio(options=categorical_cols, name="Categorical Feature: "),
    value=categorical_cols[0])

df_long = train_df[categorical_cols + ["income"]].melt(id_vars="income", var_name="

categories_chart = (alt.Chart(df_long).transform_filter(radio).mark_bar().encode(
        x=alt.X("value:N", sort="-y"),
        y=alt.Y("count():Q", stack="normalize", title="Proportion"),
        color="income:N",
        tooltip=["feature", "value", "income", "count()"]).properties(width=500, he

radio_bar = alt.Chart(pd.DataFrame({"feature": categorical_cols})).mark_point(opaci

radio_bar & categories_chart
```

Out[15]:



Categorical Feature: ◉ workclass ○ education ○ marital-status ○ occupation ○ relationship ○ race ○ sex ○ native-country

This interactive chart shows the distribution of income levels across different categorical features. By selecting a feature from the radio buttons, the chart displays the proportion of individuals earning <=50K and >50K within each category of that feature. This can be used to compare how income outcomes vary across target groups.

For example, when selecting workclass, we can see that individuals in “Self-emp-inc” have a noticeably higher proportion of >50K earners compared with categories like “Private” or “Without-pay.” This helps identify which categories are more associated with higher or lower income.

Insights for Histogram of Categorical Features

- Workclass: Some categories like Self-emp-inc and Federal-gov show a noticeably larger share of individuals earning >50K. In contrast, groups like Private and Never-worked are totally composed of <=50K earners.
- Education: Higher degrees (e.g., Masters, Doctorate) are strongly associated with higher income, with a much greater proportion of >50K individuals. Meanwhile, categories such as HS-grad and Some-college are still dominated by the <=50K group.
- Other categorical features: Variables including marital-status, occupation, and relationship also show clear differences in the distribution of income levels across categories. This uneven spread suggests that many of these categorical predictors carry useful information for distinguishing between the two income groups.

Analysis

To build a classification model, all columns in the dataset were used except final weight (irrelevant data), education-num (redundant information with the education column), and race (for ethical considerations). All columns are imputed and then transformed according to their data types. We compare the performance of SVM RBF, logistic regression, and random forest, using their default parameters. We then select the top performing model for hyperparameter and decision threshold tuning with cross validation. We evaluate the models using F1 score due to the class imbalance.

In [16]:
```
X_train = train_df.drop(columns="income")
y_train = train_df["income"]
X_test = test_df.drop(columns="income")
y_test = test_df["income"]
```

In [17]:
```
X_train.head()
```

Out[17]:

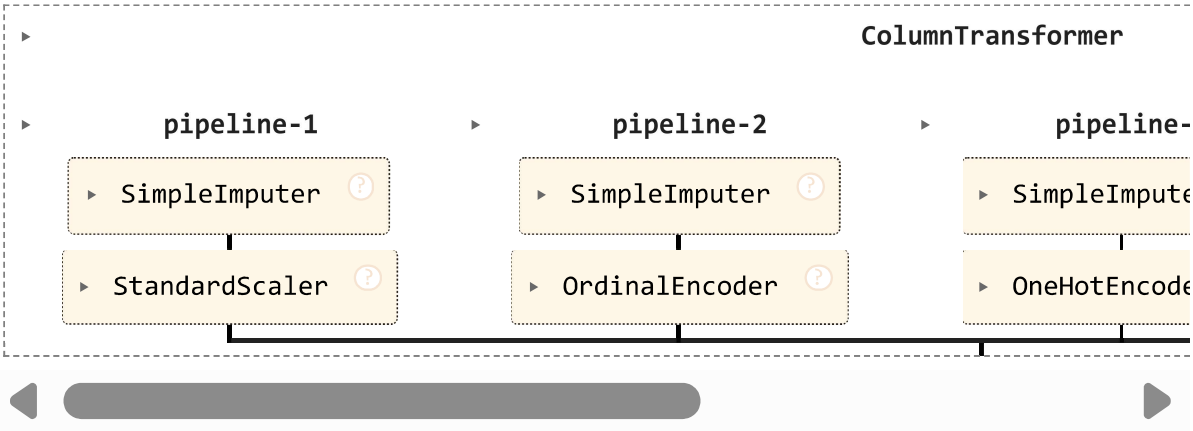
	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship
4044	19	Private	246652	Some-college	10	Never-married	Sales	Own-child
34649	20	NaN	323309	HS-grad	9	Never-married	NaN	Own-child
48375	51	Self-emp-inc	318351	HS-grad	9	Divorced	Adm-clerical	Not-in-family
37576	18	Self-emp-not-inc	161245	12th	8	Never-married	Farming-fishing	Own-child
8211	73	Private	301210	1st-4th	2	Married-civ-spouse	Transport-moving	Husband

In [18]:

```
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer

numeric_features = ['age', 'capital-gain', 'capital-loss', 'hours-per-week']
categorical_features = ['workclass', 'marital-status', 'occupation', 'relationship'
ordinal_features = ['education']
binary_features = ['sex']
drop_features = ['fnlwgt', 'education-num', 'race']
order = ['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th', '10th',
         '11th', '12th', "HS-grad", "Prof-school", "Assoc-voc", "Assoc-acdm",
         "Some-college", "Bachelors", 'Masters', 'Doctorate']

ordinal_transformer = make_pipeline(
    SimpleImputer(strategy='most_frequent'),
    OrdinalEncoder(categories=[order], dtype = int, handle_unknown='use_encoded_val
)
binary_transformer = make_pipeline(
    SimpleImputer(strategy='most_frequent'),
    OneHotEncoder(drop="if_binary", dtype=int)
)
numeric_transformer = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler()
)
categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore", sparse_output=False)
)


preprocessor = make_column_transformer(
```

```
    (numeric_transformer, numeric_features),
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
    ('drop', drop_features)
)

preprocessor
```

Out[18]:



In [19]:
```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import make_scorer, f1_score, precision_recall_curve
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
f1_scorer = make_scorer(f1_score, pos_label='>50K')

# dummy baseline. Predicting the majority class (negative label in our case) will g
# So we force the DummyClassifier to predict the positive label every time which gi
dummy = DummyClassifier(strategy='constant', constant='>50K')
dummy_pipe = make_pipeline(preprocessor, dummy)
cross_val_score(dummy_pipe, X_train, y_train, cv=5, scoring=f1_scorer).mean()
```

Out[19]:
```
np.float64(0.38732043556536)
```

In [20]:
```
# logistic regression
logReg = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=45
logReg_pipe = make_pipeline(preprocessor, logReg)
cross_val_score(logReg_pipe, X_train, y_train, cv=5, scoring=f1_scorer).mean()
```

Out[20]:
```
np.float64(0.6743110188770938)
```

In [21]:
```
# RBF SVM
svm = SVC(kernel='rbf', class_weight='balanced', random_state=456)
svm_pipe = make_pipeline(preprocessor, svm)
cross_val_score(svm_pipe, X_train, y_train, cv=5, scoring=f1_scorer).mean()
```

Out[21]:
```
np.float64(0.6555552295116688)
```

In [22]:
```
# random forest
rf = RandomForestClassifier(class_weight='balanced', random_state=456)
```

```
rf_pipe = make_pipeline(preprocessor, rf)
cross_val_score(rf_pipe, X_train, y_train, cv=5, scoring=f1_scorer).mean()
```

Out[22]: np.float64(0.6570910765711588)

In [23]:
```
from scipy.stats import loguniform
from sklearn.model_selection import RandomizedSearchCV


# C tuning for best performing model (logistic regression)
param_dist = {
    "logisticregression__C": loguniform(0.01, 10)
}

random_search = RandomizedSearchCV(
    logReg_pipe,
    param_distributions = param_dist,
    n_iter=50,
    n_jobs=-1,
    return_train_score=True,
    scoring=f1_scorer,
    cv=5,
    random_state=114514
)

random_search.fit(X_train, y_train)
print(random_search.best_params_)
print(random_search.best_score_)
```

```
{'logisticregression__C': np.float64(0.31457503004671256)}
0.675070022702436
```

In [24]:
```
# threshold tuning
best_model = random_search.best_estimator_
pred_y = cross_val_predict(
    best_model,
    X_train,
    y_train,
    cv=5,
    method="predict_proba"
)[:, 1]
pre, rec, thr = precision_recall_curve(y_train, pred_y, pos_label='>50K')
f1 = 2 * (pre * rec) / (pre + rec)
best_thr = thr[f1.argmax()]
best_thr
```

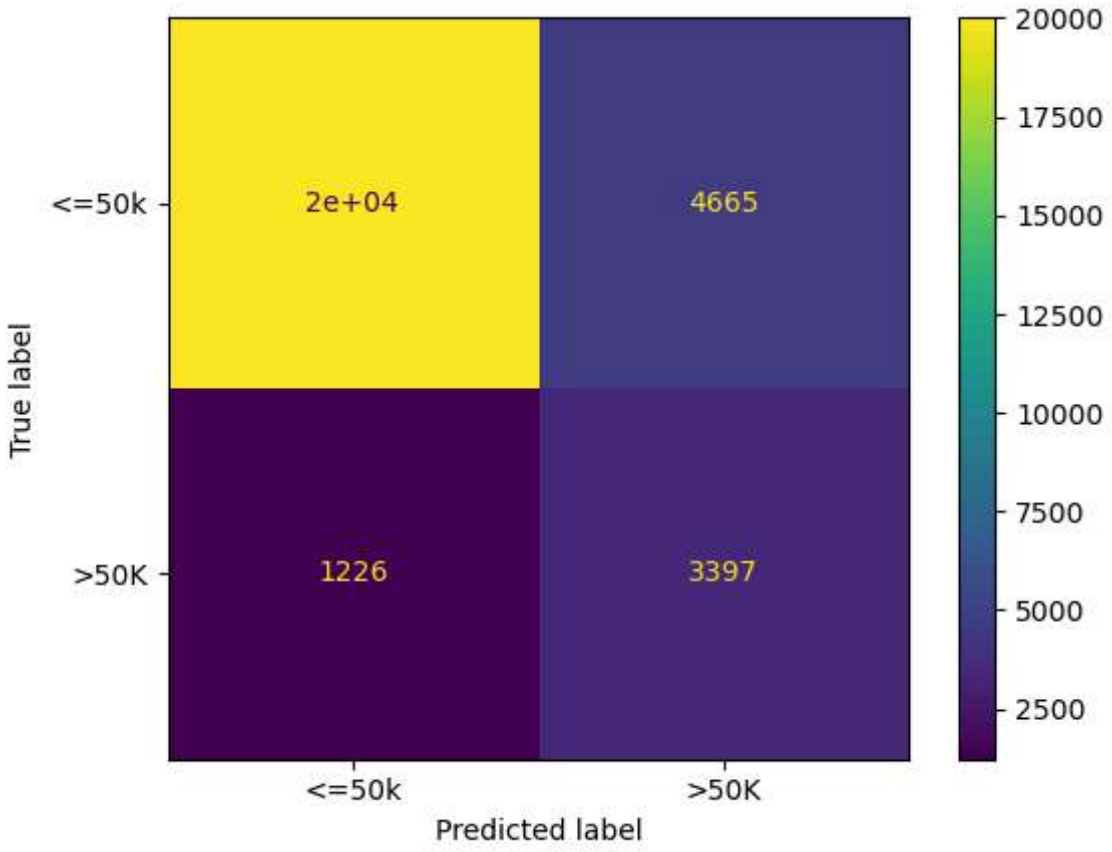
Out[24]: np.float64(0.6394673986593543)

In [25]:
```
from sklearn.metrics import classification_report

best_model.fit(X_train, y_train)
y_test_pred = best_model.predict_proba(X_test)
pred_result = (y_test_pred[:,1] >= best_thr)
y_test_binary = (y_test == '>50K').astype(int)
print(classification_report(y_test_binary, pred_result))
```

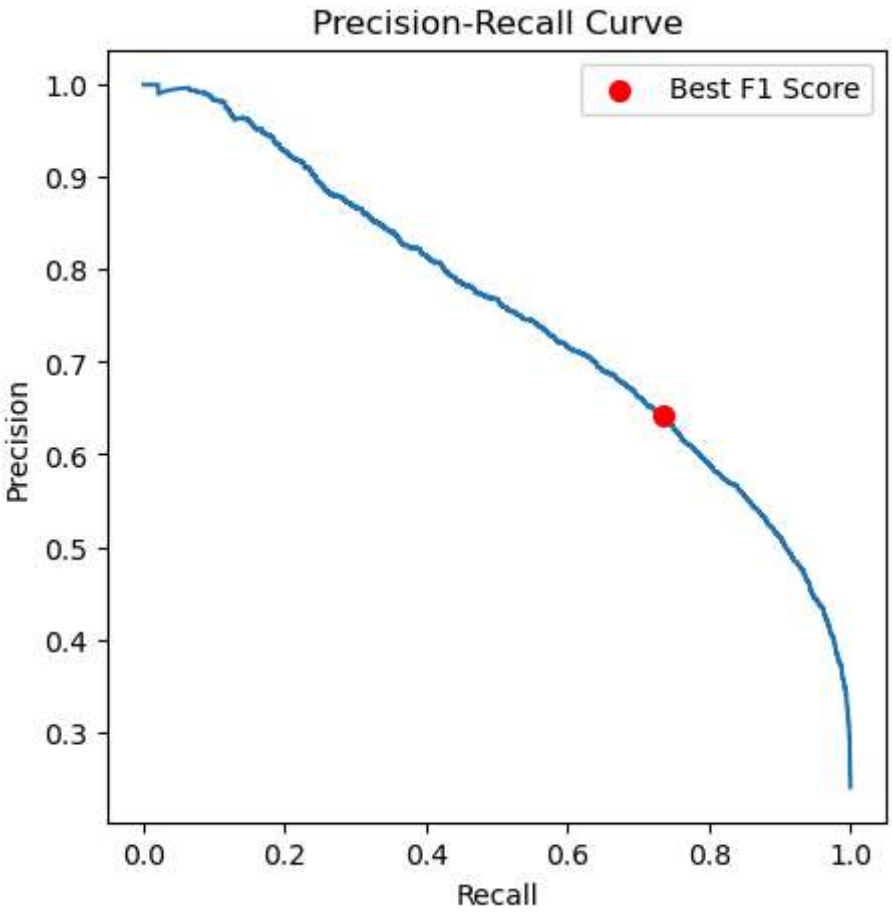
	precision	recall	f1-score	support
0	0.94	0.81	0.87	24683
1	0.42	0.73	0.54	4623
accuracy			0.80	29306
macro avg	0.68	0.77	0.70	29306
weighted avg	0.86	0.80	0.82	29306

In [29]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test_binary, pred_result,display_labels=[
```

Out[29]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x266b5664bf0>



In [30]:
```
plt.figure(figsize=(5, 5))
plt.plot(rec, pre)
plt.scatter(rec[f1.argmax()], pre[f1.argmax()], color='red', label='Best F1 Score',
plt.title(f'Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.show()
```



Results & Discussion

The initial experiment revealed that the logistic regression model has the highest base F1 score of ~0.674. After tuning the hyperparameter C and the decision threshold, we evaluated our final model on the test set. Our model achieved a final accuracy of 0.80 and a weighted average F1 score of 0.83.

Regarding our class specific results, we can see our models perform well on the majority class (income <= 50K), with a precision and recall of 0.94 and 0.81 respectively. On the minority class (income > 50K), our model has a recall of 0.74, meaning it catches 74% of the minority class. We only have a 0.42 precision on the minority class, which implies that there are many false positives. In conclusion, our model is tuned to be sensitive to catch many cases where one's income is above 50K, but also creates many false alarms.

Future Works

To improve the model, we propose some solutions to be done in further works. First, our decision threshold is tuned to maximize F1 score. Instead of this, we could select a threshold manually, depending on if we want to focus more on recall or precision. Instead of using

class_weight='balanced' to give the minority class a heavier weight, another common and often better approach is to oversample the minority class, creating synthetic data until the classes are balanced. Finally, with the suboptimal results of both the recall and precision of the minority class, we suspect our models are not complex enough to accurately model the data. Other models such as XGBoost or neural networks could be tested to see if there is an improvement.

References

Becker, B., & Kohavi, R. (1996). Adult [Data set]. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20

Statistics Canada. (2025, October 9). Distributions of household economic accounts for income, consumption, saving and wealth of Canadian households, second quarter 2025. https://www150.statcan.gc.ca/n1/daily-quotidien/251009/dq251009a-eng.html

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., … Sievert, S. (2018). Altair: Interactive statistical visualizations for Python. Journal of Open Source Software, 3(32), 1057. https://doi.org/10.21105/joss.01057

Connolly, M., & Haeck, C. (2024). Intergenerational income mobility trends in Canada. Canadian Journal of Economics / Revue canadienne d'économique, 57(1), 5–26. https://doi.org/10.1111/caje.12699