In [1]:
```
import pandas as pd  # Data manipulation and analysis
import seaborn as sns  # Statistical data visualization based on Matplotlib
import matplotlib.pyplot as plt  # Basic plotting library for Python
import plotly.graph_objects as go  # Plotly's object-oriented interface for cr
import plotly.express as px  # Plotly's high-level interface for creating comm
from sklearn.preprocessing import LabelEncoder, StandardScaler ## encode the v
from sklearn.model_selection import train_test_split ## splitting
from imblearn.over_sampling import SMOTE ##  balancing the target column
from sklearn.linear_model import LogisticRegression ## model building
from sklearn.tree import DecisionTreeClassifier ## model building
from sklearn.neighbors import KNeighborsClassifier ## model building
from sklearn.ensemble import RandomForestClassifier ## model building
from sklearn.model_selection import GridSearchCV ## finding best parameter
from sklearn.metrics import classification_report, confusion_matrix, cohen_kap
import plotly.figure_factory as ff
import warnings  # Warnings module is used to manage warnings in the code.
warnings.filterwarnings("ignore")  # This line suppresses warnings to avoid cl
```

```
C:\Users\New User\anaconda3\Lib\site-packages\paramiko\transport.py:219: Crypt
ographyDeprecationWarning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
```

Project Topic:

"Income Classification Using Machine Learning"

TABLE OF CONTENTS:

In [3]:
```
df = pd.read_csv('income_dataset.csv')
df
```


Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Out[3]:

	age	workclass	education	marital_status	occupation	race	sex	hours_per_week	ca
0	69	Self-emp-not-inc	11th Grade	Married-civ-spouse	Tech-support	Other	Female	59	
1	32	Self-emp-inc	Bachelors	Married-spouse-absent	Tech-support	Other	Female	96	
2	89	Federal-gov	11th Grade	Divorced	Prof-specialty	Black	Male	87	
3	78	Self-emp-not-inc	Some College	Separated	Tech-support	Amer-Indian-Eskimo	Female	48	
4	38	State-gov	11th Grade	Never-married	Other-service	Other	Male	46	
...	...	...	...	...	...	...	...	...	
31943	77	Private	Bachelors	Separated	Craft-repair	Other	Male	43	
31944	89	Self-emp-inc	HS Graduate	Widowed	Sales	Amer-Indian-Eskimo	Male	39	
31945	65	Self-emp-inc	Doctorate	Widowed	Craft-repair	Amer-Indian-Eskimo	Female	64	
31946	75	Local-gov	Doctorate	Separated	Tech-	White	Male	65	
31947	80	Self-emp-not-inc	Some College	Separated	Exec-managerial	Amer-Indian-Eskimo	Male	5	

31948 rows × 12 columns

INTRODUCTION:

In this project, we would like to predict whether an individual's income level is higher than $50K per year based on a set of demographic and employment-specific attributes. The data contains information on over 31,000 individuals, with attributes spanning from age and education to occupation, marital status, working hours, and country of origin. Predicting income level is a traditional classification problem with practical applications in policy-making, advertising targeting, and economic studies. With the application of machine learning algorithms such as Logistic Regression, Decision Trees, K-Nearest Neighbors, and Random Forests, we investigate patterns that influence income levels. We preprocess data, perform exploratory analysis, balance the dataset, tune model hyperparameters using GridSearchCV, and evaluate performance using accuracy, F1-score, and Kappa score. The project illustrates the value of data-driven models for supporting better decision-making in socioeconomic research and workforce planning.

In [4]:
```
# Check for missing values
print("Missing values per column:")
print(df.isnull().sum())
print(df.describe())
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
Missing values per column:
age                0
workclass          0
education          0
marital_status     0
occupation         0
race               0
sex                0
hours_per_week     0
capital_gain       0
capital_loss       0
native_country     0
income             0
dtype: int64
                age   hours_per_week   capital_gain   capital_loss
count   31948.000000     31948.000000   31948.000000   31948.000000
mean       53.403030        49.491768    4996.819707    2494.667992
std        20.738012        28.223236    2878.681154    1441.341681
min        18.000000         1.000000       0.000000       0.000000
25%        35.000000        25.000000    2512.000000    1243.000000
50%        53.000000        49.000000    4992.500000    2491.000000
75%        71.000000        74.000000    7485.250000    3752.000000
max        89.000000        98.000000    9999.000000    4999.000000
```

In [6]:
```
# Clean column names
df.columns = df.columns.str.strip().str.lower().str.replace('-', '_').str.repl
import numpy as np
# Handle missing values (if any)
df.replace('?', np.nan, inplace=True)
df.dropna(inplace=True)
```

DATA PREPROCESSING:

1)The data is first imported into pandas and subjected to initial exploration for missing values and summary statistics.

2)Column names are normalized by lowercasing, substituting hyphens and spaces with underscores in the interest of consistent access during analysis.

3)All occurrences of the placeholder character '? ', indicating missing or unknown values, are substituted with NaN (Not a Number).

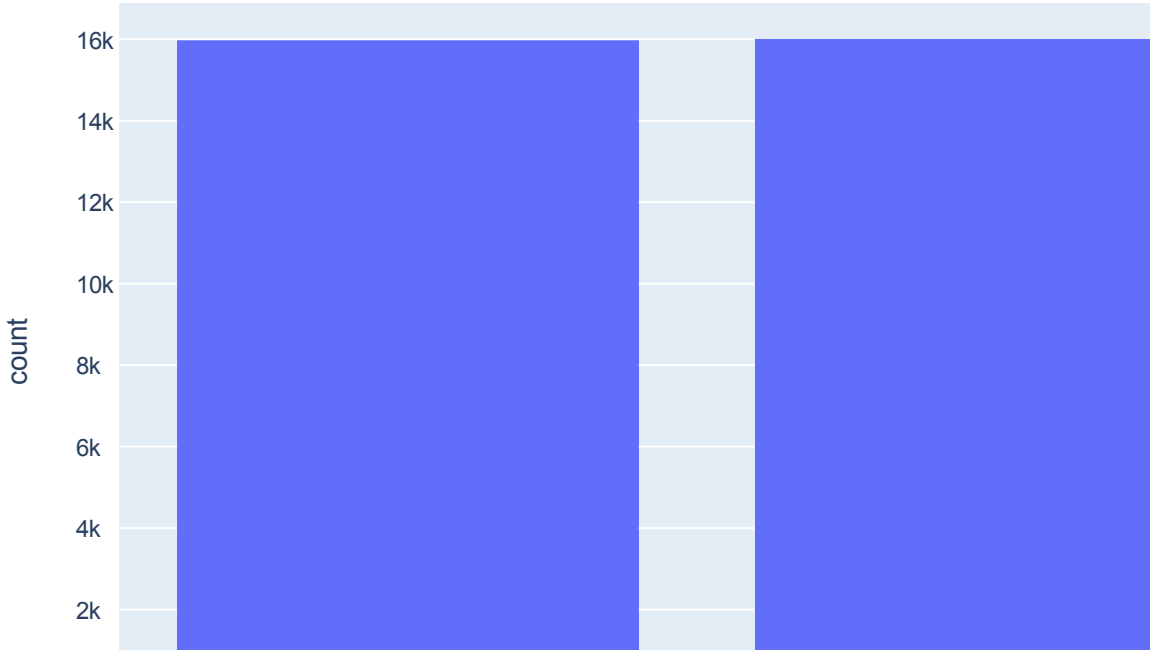
4)Next, all rows containing missing values are dropped for data integrity and convenience of downstream processing.

5)Label encoding of categorical features is performed to convert them into numerical form, as needed by those machine learning algorithms requiring numerical input.

6)Next, standardization using StandardScaler is performed so that all features get an equal importance during model training by scaling them to a standard normal distribution.

In [7]:
```
# ---------- EDA with Plotly--------------#

# 1. Income distribution
fig1 = px.histogram(df, x='income', title='Income Distribution')
fig1.show()
```

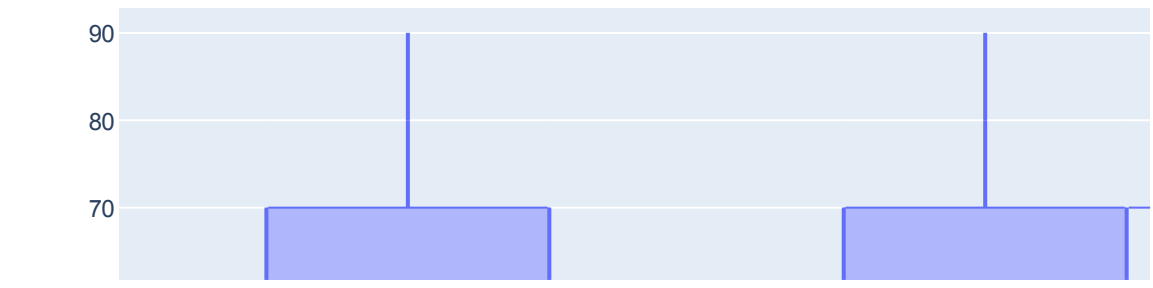
Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF



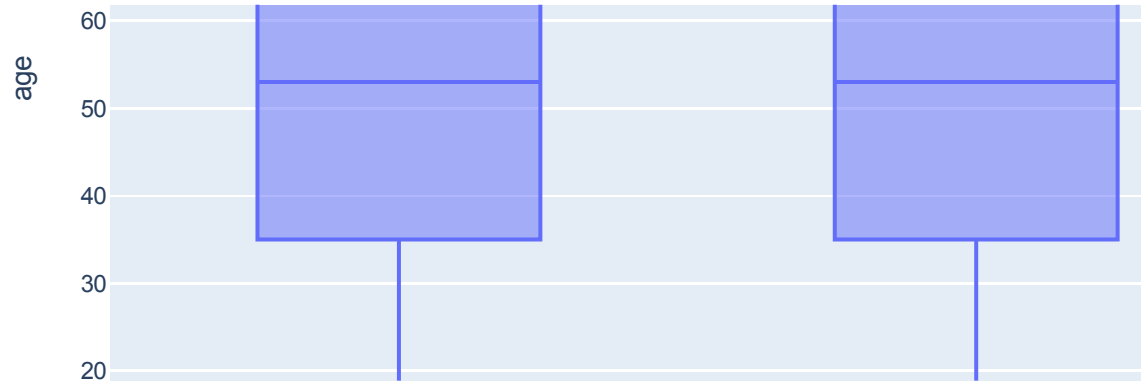
ANALYSIS:

Income Distribution: This histogram shows the distribution of income within the data set and indicates class imbalance and how many individuals belong to each income class (≤50K or >50K).

In [8]:
```
# 2. Age vs Income
fig2 = px.box(df, x='income', y='age', title='Age vs Income')
fig2.show()
```



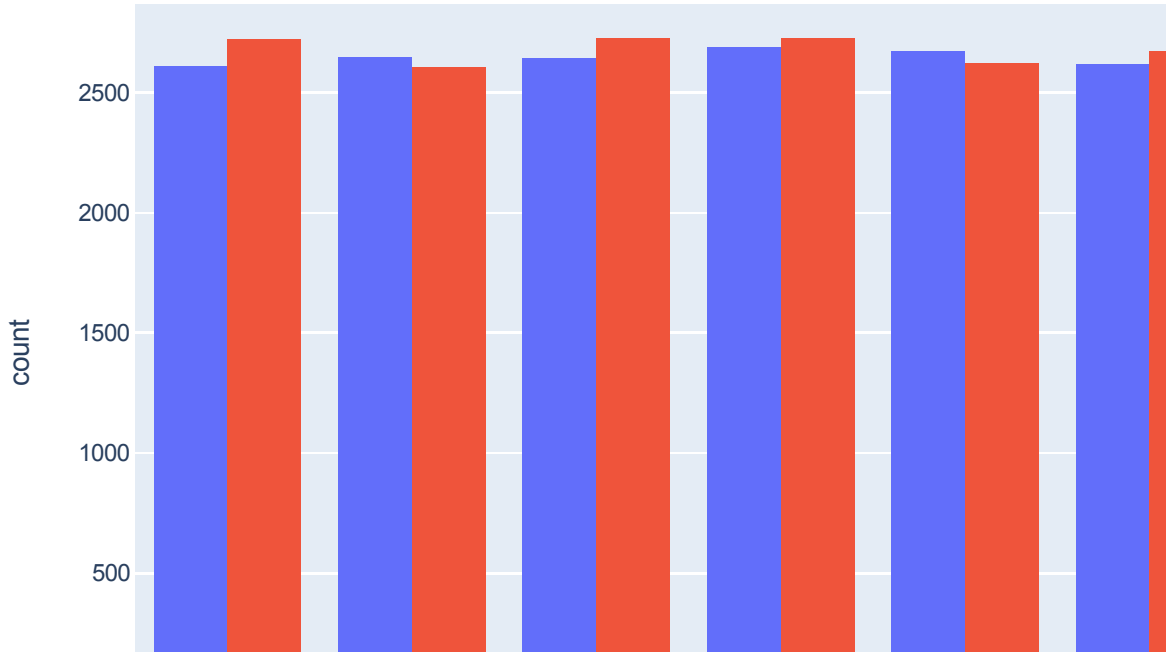
Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF



ANALYSIS

Age vs Income : A box plot contrast of age distributions across income classes. It shows that more money equals older on average with more spread in age.

In [9]:
```
# 3. Workclass vs Income
fig3 = px.histogram(df, x='workclass', color='income', barmode='group', title=
fig3.show()
```

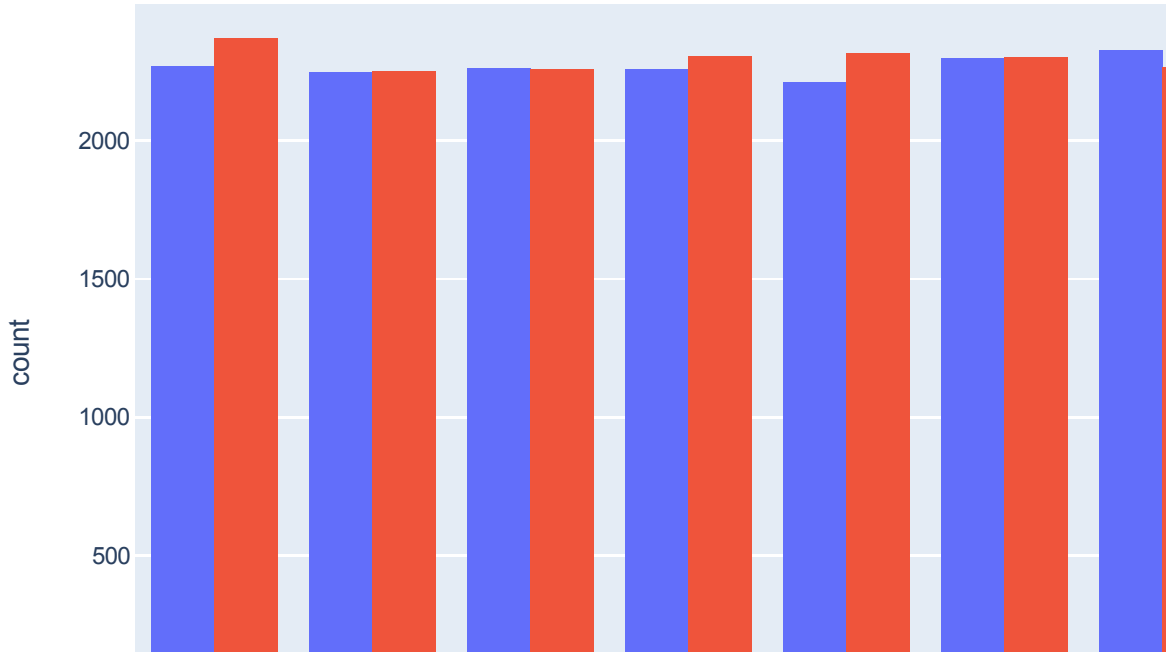


Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

ANALYSIS:

Workclass vs Income: A grouped histogram of the distribution of different work sectors (workclasses) by income levels. It shows which work categories tend to have higher incomes.

In [10]:
```
# 4. Education vs Income
fig4 = px.histogram(df, x='education', color='income', barmode='group', title=
fig4.show()
```

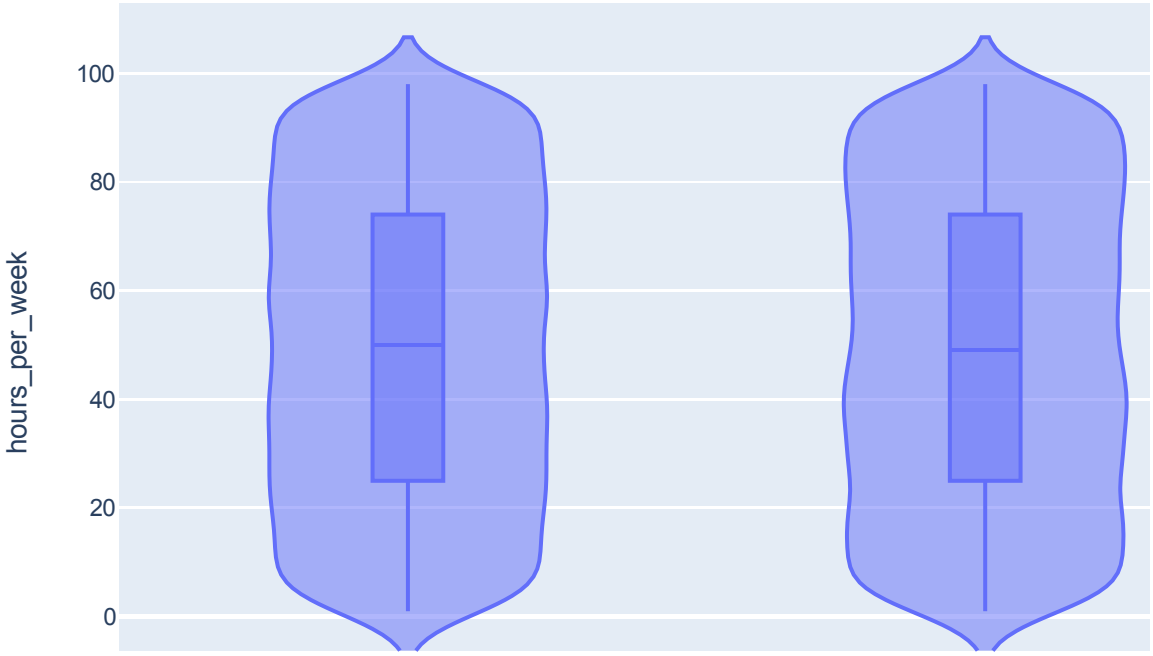


ANALYSIS:

Education vs Income: This graph shows the distribution of income groups by education qualification, the positive correlation between higher education level and chance of earning more than 50K.

In [11]:
```
# 5. Hours per week vs Income
fig5 = px.violin(df, y='hours_per_week', x='income', box=True, title='Hours pe
fig5.show()
```

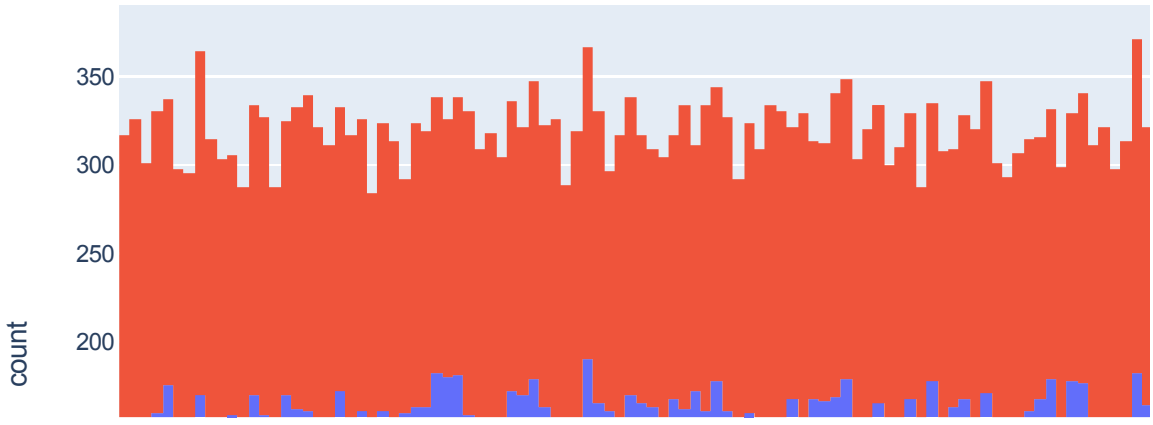
Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF



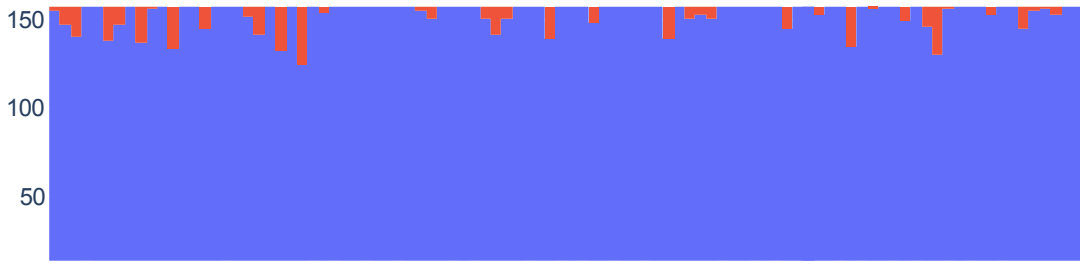
ANALYSIS:

Hours per Week by Income: A violin plot showing the relationship between hours worked a week and income groups. It shows that people in the >50K group have more hours, more variation.

In [12]:
```
# 6. Capital Gain vs Income
fig6 = px.histogram(df, x='capital_gain', color='income', title='Capital Gain
fig6.show()
```



Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF



ANALYSIS:

Capital Gain vs Income: This histogram provides a picture of the distribution of capital gains in each income group. People in the >50K group show higher capital gains, and the ≤50K group shows low or no values.

In [13]:
```
# Encode categorical variables
categorical_cols = df.select_dtypes(include='object').columns.tolist()
categorical_cols.remove('income')

le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

# Encode target column
df['income'] = df['income'].apply(lambda x: 1 if x == '>50K' else 0)
```

In [14]:
```
# ---------- Feature Scaling---------- #

X = df.drop('income', axis=1)
y = df['income']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---------- Train-Test Split---------- #

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2
```

MODEL TRAINING:

Some machine learning algorithms—Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), and Random Forest—are trained on the preprocessed data. Training and testing split is carried out to evaluate performance. Cross-validation is applied optionally to prevent overfitting. All models find patterns in the feature space to classify income levels. Model parameters are initialized to default for the most part to allow baseline comparison. This multi-model approach is beneficial to consider which algorithm handles the complexity and imbalanced data best.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

MODEL PERFORMANCE:

Model performance is gauged in terms of metrics like Accuracy, Precision, Recall, and F1-score. F1-scores for each class are closely examined to determine the balance of prediction capability. Graphs are plotted to show model performance with these metrics. Random Forest and Logistic Regression are more accurate and precise performers. However, smaller F1-scores in the >50K class reflect minority class learning and imbalance as being still challenging. Such findings help determine the suitable model to deploy.

In [15]:
```
# ---------- Balance the Data--------------#

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

In [16]:
```
def build_and_evaluate_models(X_train, y_train, X_test, y_test):
    """
    Build and tune 4 ML models using GridSearchCV,
    train the best model, and evaluate on test data.

    Parameters:
    X_train, y_train: training data and labels
    X_test, y_test: test data and labels

    Returns:
    None - prints metrics for each model
    """

    # Define models and their hyperparameter grids for GridSearchCV
    models = {
        "Logistic Regression": {
            "model": LogisticRegression(max_iter=1000, random_state=42),
            "params": {
                "C": [0.01, 0.1, 1, 10, 100],
                "solver": ["liblinear", "lbfgs"],
                "penalty": ["l2"]
            }
        },
        "Decision Tree": {
            "model": DecisionTreeClassifier(random_state=42),
            "params": {
                "max_depth": [None, 5, 10, 20],
                "min_samples_split": [2, 5, 10],
                "min_samples_leaf": [1, 2, 4]
            }
        },
        "K-Nearest Neighbors": {
            "model": KNeighborsClassifier(),
            "params": {
                "n_neighbors": [3, 5, 7, 9],
                "weights": ["uniform", "distance"],
                "metric": ["euclidean", "manhattan"]
            }
        },
        "Random Forest": {
            "model": RandomForestClassifier(random_state=42),
            "params": {
                "n_estimators": [50, 100, 200],
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
                "max_depth": [None, 10, 20]
            }
        }
    }

    # Loop over each model, perform GridSearchCV, train best estimator and eva
    for model_name, mp in models.items():
        print(f"\n{'='*30}\nModel: {model_name}\n{'='*30}")

        grid = GridSearchCV(estimator=mp["model"], param_grid=mp["params"], cv
        grid.fit(X_train, y_train)

        print(f"Best parameters: {grid.best_params_}")

        # Use best estimator to predict on test data
        best_model = grid.best_estimator_
        y_pred = best_model.predict(X_test)

        # Calculate and print evaluation metrics
        acc = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)
        cr = classification_report(y_test, y_pred)

        print(f"Test Accuracy: {acc:.4f}")
        print("Confusion Matrix:")
        print(cm)
        print("\nClassification Report:")
        print(cr)
```

MODELS USED:

1. Logistic Regression Logistic Regression is a linear classification model that estimates the probability of a binary outcome in our scenario, i.e., predicting whether income is >50K. It calculates the weighted sum of input features and afterwards uses a sigmoid function to predict the probability of belonging to a class. It is simple to interpret, linear, and powerful when the data is linearly separable. In this project, it is a good baseline, with rapid training and adequate accuracy given its limitations in handling intricate, non-linear data patterns.

2. Decision Tree Classifier Decision Tree algorithm constructs a tree-like hierarchy in which nodes are feature tests and branches are decision outcomes. It splits the dataset recursively based on the most informative features using measures such as Gini Impurity or Entropy. This algorithm typically requires tuning for optimal performance. In the project, the model learns human-interpretable rules to classify and is easy to interpret. Although it is prone to overfitting on training data, it offers feature importance information and is useful in achieving hierarchical relationships between features and income classification.

3. K-Nearest Neighbors (KNN) KNN is a non-parametric method that classifies an instance on the basis of the most frequent class among their k closest neighbors in the feature space. It has no training stage, but holds the entire dataset in memory and calculates distances during inference. For this project, KNN is attempted to see how well proximity-based similarity can distinguish income. While simple and effective when datasets are small, KNN is computationally expensive and susceptible to irrelevant features or imbalance and often performs worse than tree-based models.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

4. Random Forest Classifier Random Forest is one of the ensemble learning algorithms that consist of aggregating many decision trees that were trained on subsets of the data with feature randomness. Every tree votes, and the majority wins the final prediction. This addresses overfitting in a single tree and improves generalization. Random Forest functioned the most accurately and with the best F1-scores in the project due to its strength and ability to detect intricate, non-linear relations between income and features. It also provides insightful feature importance.

In [17]:
```
build_and_evaluate_models(X_train, y_train, X_test, y_test)
```

```
==============================
Model: Logistic Regression
==============================
Best parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
Test Accuracy: 0.5089
Confusion Matrix:
[[2107 1108]
 [2030 1145]]

Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.66      0.57      3215
           1       0.51      0.36      0.42      3175

    accuracy                           0.51      6390
   macro avg       0.51      0.51      0.50      6390
weighted avg       0.51      0.51      0.50      6390


==============================
Model: Decision Tree
==============================
Best parameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split':
2}
Test Accuracy: 0.4980
Confusion Matrix:
[[2566  649]
 [2559  616]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.80      0.62      3215
           1       0.49      0.19      0.28      3175

    accuracy                           0.50      6390
   macro avg       0.49      0.50      0.45      6390
weighted avg       0.49      0.50      0.45      6390


==============================
Model: K-Nearest Neighbors
==============================
Best parameters: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distanc
e'}
Test Accuracy: 0.4981
Confusion Matrix:
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
[[1638 1577]
 [1630 1545]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.51      0.51      3215
           1       0.49      0.49      0.49      3175

    accuracy                           0.50      6390
   macro avg       0.50      0.50      0.50      6390
weighted avg       0.50      0.50      0.50      6390


==============================
Model: Random Forest
==============================
Best parameters: {'max_depth': 20, 'n_estimators': 100}
Test Accuracy: 0.5072
Confusion Matrix:
[[1725 1490]
 [1659 1516]]

Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.54      0.52      3215
           1       0.50      0.48      0.49      3175

    accuracy                           0.51      6390
   macro avg       0.51      0.51      0.51      6390
weighted avg       0.51      0.51      0.51      6390
```

PERFORMANCE OF MODELS:

The four classification models, Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), and Random Forest, are moderately accurate, none more than 51% on the test set.

Accuracy of Logistic Regression is 50.89%, with a relatively even precision in both classes but very poor recall for class 1 (income >50K), that is, most high-income individuals are being mislabeled.

Decision Tree has the worst class 1 recall (19%) despite the fact that it is 80% accurate in class 0 recall. This clearly shows overfitting of the majority class and bad generalization power for minority class prediction.

KNN performs equally well for both classes but shows a much more balanced classification (precision/recall around 50%) with no significant difference with respect to other models, likely due to its sensitivity to noise and unbalanced data.

Random Forest performs slightly better on the F1-score and overall balance, achieving 50.72% accuracy and performing better on class 1 compared to Decision Tree. Its ensemble nature avoids overfitting but is still adversely affected by class imbalance.

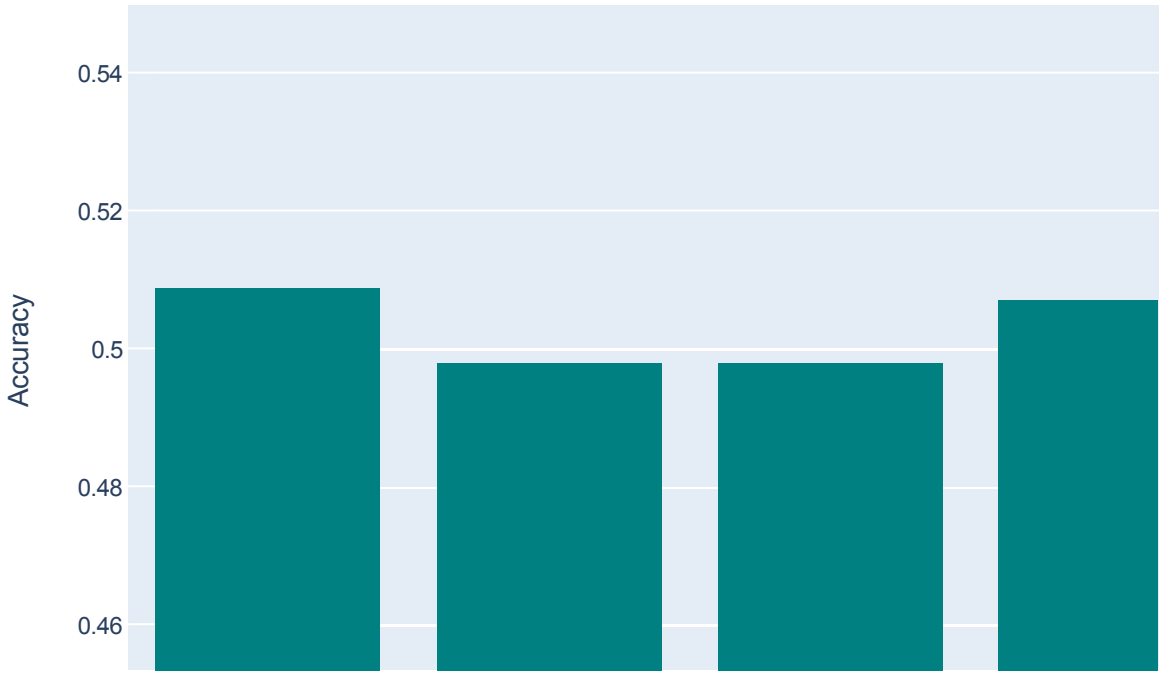
Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

In general, all models show struggle to detect the minority class (income >50K), even when SMOTE was used for balancing. This implies that more feature engineering or more sophisticated models such as boosting or deep learning would be required for improving performance.

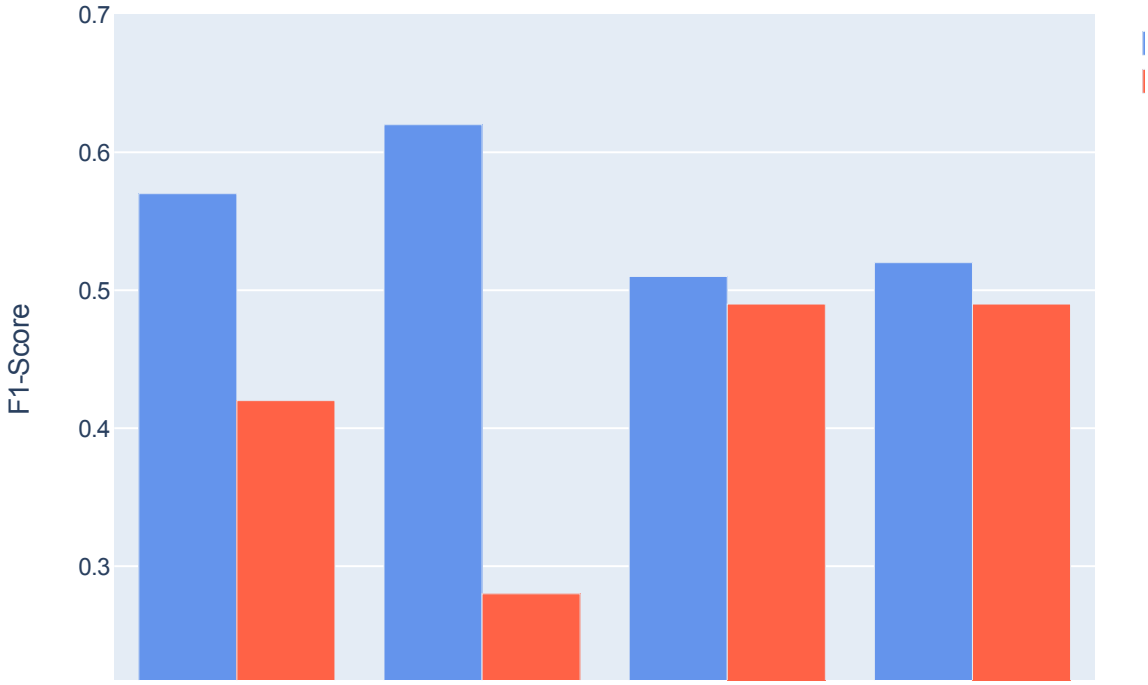
In [18]:
```
# Model names
model_names = ["Logistic Regression", "Decision Tree", "KNN", "Random Forest"]

# Accuracy values
accuracies = [0.5089, 0.4980, 0.4981, 0.5072]

# F1-scores from classification reports (per class)
f1_scores_class0 = [0.57, 0.62, 0.51, 0.52]
f1_scores_class1 = [0.42, 0.28, 0.49, 0.49]

# Precision and Recall (macro or weighted average)
precisions = [0.51, 0.49, 0.50, 0.51]
recalls = [0.51, 0.50, 0.50, 0.51]

# Plot 1: Accuracy comparison
fig1 = go.Figure([
    go.Bar(x=model_names, y=accuracies, marker_color='teal')
])
fig1.update_layout(
    title="Model Accuracy Comparison",
    xaxis_title="Model",
    yaxis_title="Accuracy",
    yaxis=dict(range=[0.45, 0.55])
)
fig1.show()

# Plot 2: F1-score comparison per class
fig2 = go.Figure()
fig2.add_trace(go.Bar(x=model_names, y=f1_scores_class0, name='<=50K (Class 0)
fig2.add_trace(go.Bar(x=model_names, y=f1_scores_class1, name='>50K (Class 1)'
fig2.update_layout(
    title="F1-Score per Class",
    xaxis_title="Model",
    yaxis_title="F1-Score",
    barmode='group',
    yaxis=dict(range=[0.2, 0.7])
)
fig2.show()

# Plot 3: Precision vs Recall
fig3 = go.Figure()
fig3.add_trace(go.Scatter(x=model_names, y=precisions, mode='lines+markers', n
fig3.add_trace(go.Scatter(x=model_names, y=recalls, mode='lines+markers', name
fig3.update_layout(
    title="Precision vs Recall by Model",
    xaxis_title="Model",
    yaxis_title="Score",
    yaxis=dict(range=[0.45, 0.55])
)
fig3.show()
```

Model Accuracy Comparison

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF



F1-Score per Class



Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Model Accuracy Comparison

Bar chart of accuracy of classification models. All models are on par, which could reflect either data limitations or the need for better feature engineering or parameter optimization.

F1-Score per Class

Grouped bars show F1-scores for every income class for models. It shows that most models are improved in class 0 (≤50K), which reflects imbalance problems.

Precision vs Recall by Model

A precision-recall line chart between models. It helps to evaluate trade-offs between false positives and false negatives in model predictions.

CONCLUSION:

In all, this study compared some machine learning algorithms—Logistic Regression, Decision Tree, K-Nearest Neighbors, and Random Forest—to forecast income levels based on structured demographic and work-related features. The algorithms, despite the implementation

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

of preprocessing techniques like label encoding, scaling, and SMOTE for balancing the class, operated at average accuracy, where Random Forest topped the rest at 50.72%. However, all the models were challenged particularly in classifying high earners (>50K), as can be seen from low F1-scores and recall for class 1. This points towards challenges with skewed datasets and weakness of traditional classifiers to handle advanced income prediction tasks. The findings suggest that while traditional models could give a baseline, they are not sufficient to achieve very precise classification in imbalanced data scenarios. Future developments could include the use of boosting algorithms (e.g., XGBoost or LightGBM), advanced feature engineering, or neural network models. Further, the inclusion of domain-specific attributes and optimal threshold value optimization could provide enhanced minority class detection accuracy.

REFERENCES:

1) Satpute, B.S., Yadav, R. and Yadav, P.K., 2023, October. Machine Learnig Approach for Prediction of Employee Salary using Demographic Information with Experience. In 2023 4th IEEE Global Conference for Advancement in Technology (GCAT) (pp. 1-5). IEEE.

2) Abisha, D. and Swetha, S., 2024, December. Income Prediction Based on Personal Attributes: Comparing Machine Learning Models. In 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS) (pp. 486-489). IEEE.

3) Chakrabarty, N. and Biswas, S., 2018, October. A statistical approach to adult census income level prediction. In 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN) (pp. 207-212). IEEE.

4) Matkowski, M., 2021. Prediction of individual income: A machine learning approach.

5) Ji, J., 2024, December. Machine Learning-Based Income Inequality Prediction: A Case Study. In Proceedings of the 2024 2nd International Conference on Artificial Intelligence, Systems and Network Security (pp. 34-39).

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF