Census Project

December 5, 2024

1 Census Data Model

1.0.1 Author: Riley Walburger

1.0.2 11/30/2024

[30]:
```
import pandas as pd

# Load the dataset
data = pd.read_csv("C:/Users/walbr/Desktop/U of U/1- Fall Semester 2024/Data Mining/Project/census.csv")

df_model = data

# Display the first few rows of the dataset
data.head()
```

[30]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	y
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

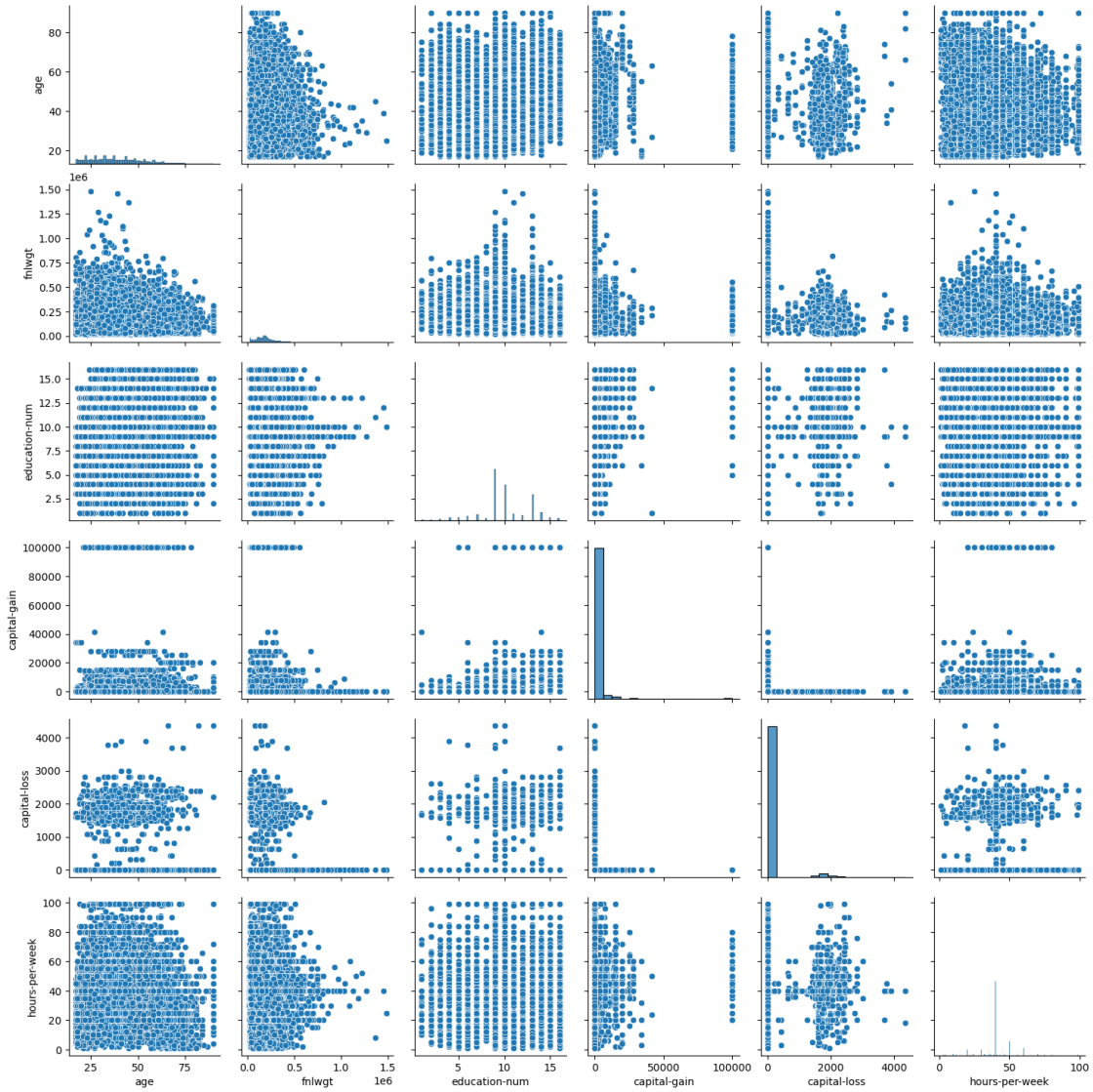
[31]:
```
# Print the column names
print(data.columns)
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'y'],
      dtype='object')
```

[32]:
```
# Count occurrences of each unique value in column 'y'
value_counts_y = data['y'].value_counts()

# Calculate percentages
percentages_y = (value_counts_y / value_counts_y.sum()) * 100

# Combine counts and percentages into a single DataFrame
counts_and_percentages = pd.DataFrame({
    'Count': value_counts_y,
    'Percentage': percentages_y
})

print("Counts and Percentages of each element in column 'y':")
print(counts_and_percentages)
```

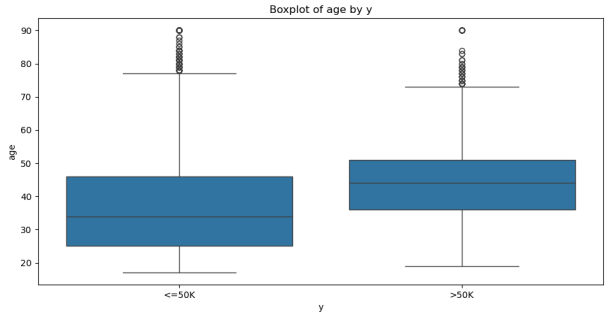
```
Counts and Percentages of each element in column 'y':
       Count  Percentage
y
<=50K  24720   75.919044
>50K    7841   24.080956
```

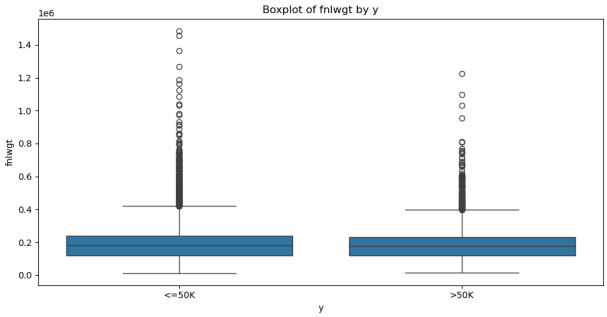
[33]:
```
import seaborn as sns
import matplotlib.pyplot as plt


# Select numeric columns for the pairs plot
numeric_data = data.select_dtypes(include=['number'])

# Create the pairs plot
sns.pairplot(numeric_data)

# Show the plot
plt.show()
```

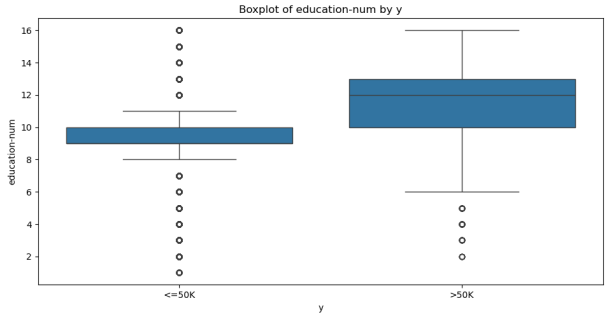


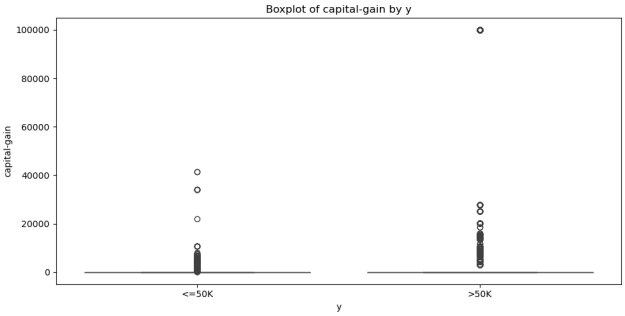
[34]:
```
data['y'] = data['y'].astype('category')

# Add the 'y' column to the numeric data
numeric_data['y'] = data['y']

# Create the pair plot with 'y' as hue
sns.pairplot(numeric_data, hue='y', diag_kind='kde', palette='Set2')

# Show the plot
plt.show()
```

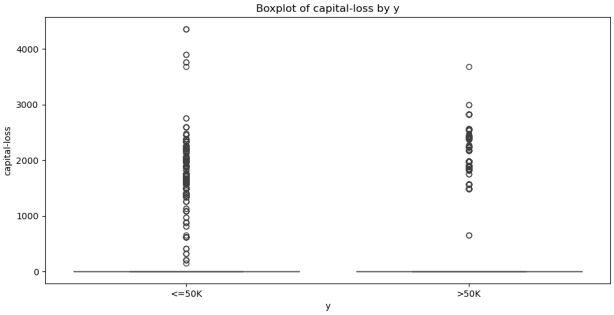
[26]:
```
# Set up the plotting grid
numeric_columns = data.select_dtypes(include=['number']).columns
num_plots = len(numeric_columns)
fig, axes = plt.subplots(nrows=num_plots, figsize=(10, 5 * num_plots))

# Create boxplots for each numeric column grouped by 'y'
for i, column in enumerate(numeric_columns):
    sns.boxplot(data=data, x='y', y=column, ax=axes[i])
    axes[i].set_title(f'Boxplot of {column} by y')
    axes[i].set_xlabel('y')
    axes[i].set_ylabel(column)

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```

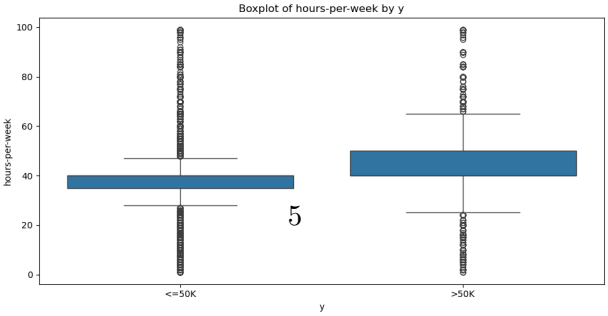

Boxplot of age by y
age
y
<=50K
>50K


Boxplot of fnlwgt by y
1e6
fnlwgt
y
<=50K
>50K


Boxplot of education-num by y
education-num
y
<=50K
>50K


Boxplot of capital-gain by y
capital-gain
y
<=50K
>50K


Boxplot of capital-loss by y
capital-loss
y
<=50K
>50K


Boxplot of hours-per-week by y
hours-per-week
y
<=50K
>50K

1.0.3 Some Key Notes

- There seems to be a group of really high capita gain that are all above 50K.
- Education and age Levels seem to be higher inside of the >50k group.

1.0.4 Next Steps

Will be useful to make basic linear regression and decision tree to see how each individual group relates to the y column.

1.1 Linear Regression

[37]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Encode categorical features using LabelEncoder or one-hot encoding
categorical_cols = df_model.select_dtypes(include=['object']).columns

# Use LabelEncoder for simplicity; you can use OneHotEncoder if needed
encoder = LabelEncoder()
for col in categorical_cols:
    df_model[col] = encoder.fit_transform(df_model[col])

# If the target variable is named 'y_1', rename it to 'y'
df_model.rename(columns={'y_1': 'y'}, inplace=True)

# Split the data into features (X) and target (y)
X = df_model.drop('y', axis=1)  # Drop the target column
y = df_model['y']  # Target variable

# Standardize the features for better performance in logistic regression
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,␣
 ↪random_state=42)

# Create and train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Extract feature names and coefficients
```

```python
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_[0]
}).sort_values(by='Coefficient', ascending=False)

# Print the top 20 coefficients
print(coefficients.head(20))

# Evaluate the model
y_pred = model.predict(X_test)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
           Feature  Coefficient
10    capital-gain     2.340856
4    education-num     0.847136
0              age     0.470903
9              sex     0.431766
12  hours-per-week     0.383201
11    capital-loss     0.262446
8             race     0.096510
2           fnlwgt     0.057607
6       occupation     0.057277
13  native-country     0.039984
3        education     0.037932
1        workclass    -0.036762
7     relationship    -0.161715
5   marital-status    -0.355023

Accuracy: 0.8258777766403931

Classification Report:
              precision    recall  f1-score   support

       <=50K       0.85      0.94      0.89      7455
        >50K       0.71      0.45      0.55      2314

    accuracy                           0.83      9769
   macro avg       0.78      0.70      0.72      9769
weighted avg       0.81      0.83      0.81      9769
```

There seems to be some interesting things that we couldn't see very well from the graphs such as the correlation between exec managers and y. Along with being married. Both of these seem to be highly correlated.

1.2 Readying Data

[38]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer

df = df_model

# Handle categorical features using label encoding
categorical_cols = ['workclass', 'education', 'marital-status', 'occupation',
                    'relationship', 'race', 'sex', 'native-country', 'y']

# Apply Label Encoding for categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

# Handle missing values (if any) using SimpleImputer
imputer = SimpleImputer(strategy='most_frequent')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Split the data into features (X) and target (y)
X = df.drop('y', axis=1)  # Features
y = df['y']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

1.2.1 Basic Decision Tree

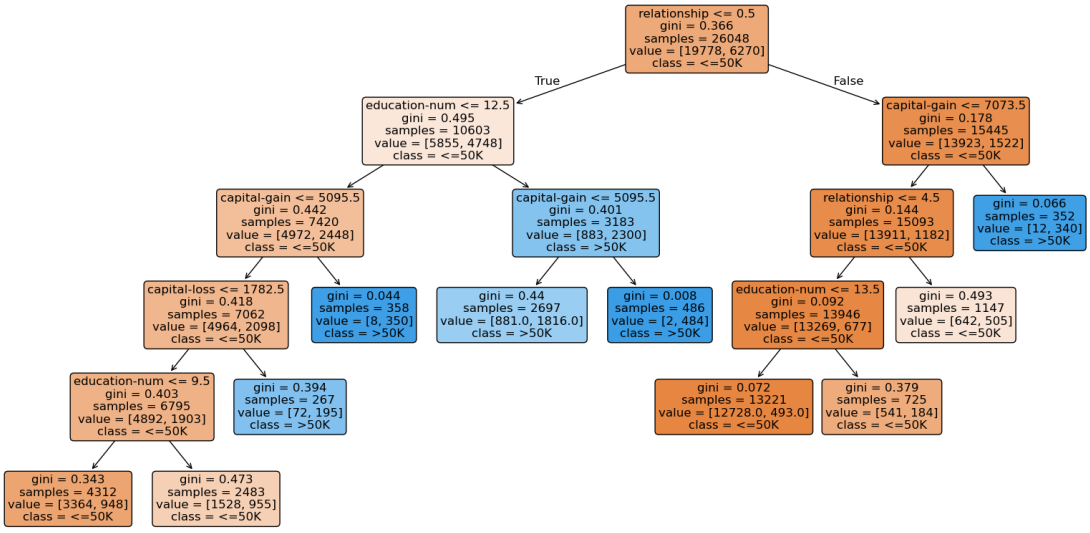
Limited Tree to see Important Factors

[39]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, precision_score, recall_score,␣
 ↪accuracy_score

# Initialize the DecisionTreeClassifier
dt = DecisionTreeClassifier(max_leaf_nodes=10, random_state=10)

# Perform 5-fold cross-validation with multiple metrics
scoring = ['accuracy', 'precision', 'recall']
cv_results = cross_validate(dt, X_train, y_train, cv=5, scoring=scoring)

# Output the cross-validation scores for each metric and their means
print("Cross-validation accuracy scores:", cv_results['test_accuracy'])
```

```
print("Mean accuracy:", cv_results['test_accuracy'].mean())

print("Cross-validation precision scores:", cv_results['test_precision'])
print("Mean precision:", cv_results['test_precision'].mean())

print("Cross-validation recall scores:", cv_results['test_recall'])
print("Mean recall:", cv_results['test_recall'].mean())
```

```
Cross-validation accuracy scores: [0.83973129 0.84779271 0.84203455 0.84162027
0.84545978]
Mean accuracy: 0.8433277190039827
Cross-validation precision scores: [0.76418663 0.74288725 0.76506765 0.75505351
0.78027466]
Mean precision: 0.7614939395720418
Cross-validation recall scores: [0.48325359 0.56220096 0.49601276 0.50637959
0.4984051 ]
Mean recall: 0.5092503987240828
```

[40]:
```
from sklearn.tree import plot_tree

# Train the Decision Tree on the training data
dt.fit(X_train, y_train)

# Plot the trained decision tree
plt.figure(figsize=(20, 10))
plot_tree(dt, filled=True, feature_names=X.columns, class_names=['<=50K',␣
↪'>50K'], rounded=True, fontsize=12)
plt.show()
```



1.3 Key Notes

Some of the important things to the model are things that we have already found to be important such as relationship status, education and capital gain. These seem to be the factors with the most information gain here as well. ## Next Steps

Some basic numbers that we should be able to beat are 80% as I am able to get this through both logistic regression and decision trees. However I need believe this could be best done through an XG Boost model so I am going to start creating those models and then

[ ]:
```
#pip install xgboost
```

1.4 XGBoost Using Default Hyperparameters

[41]:
```
import xgboost as xgb
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, precision_score, recall_score,␣
 ↪accuracy_score
import numpy as np

# Initialize XGBoost model
xgb_model = xgb.XGBClassifier()

# Perform 5-fold cross-validation with multiple metrics
scoring = ['accuracy', 'precision', 'recall']
cv_results = cross_validate(xgb_model, X_train, y_train, cv=5, scoring=scoring)

# Output the cross-validation scores for each metric and their means
print("Cross-validation accuracy scores:", cv_results['test_accuracy'])
print("Mean accuracy:", cv_results['test_accuracy'].mean())

print("Cross-validation precision scores:", cv_results['test_precision'])
print("Mean precision:", cv_results['test_precision'].mean())

print("Cross-validation recall scores:", cv_results['test_recall'])
print("Mean recall:", cv_results['test_recall'].mean())
```

```
Cross-validation accuracy scores: [0.86007678 0.86737044 0.86890595 0.87406412
0.86888078]
Mean accuracy: 0.8678596140077947
Cross-validation precision scores: [0.74881517 0.74714662 0.77799416 0.78314394
0.76410731]
Mean precision: 0.764241438180356
Cross-validation recall scores: [0.62998405 0.67862839 0.63716108 0.65948963
0.65869219]
Mean recall: 0.6527910685805423
```

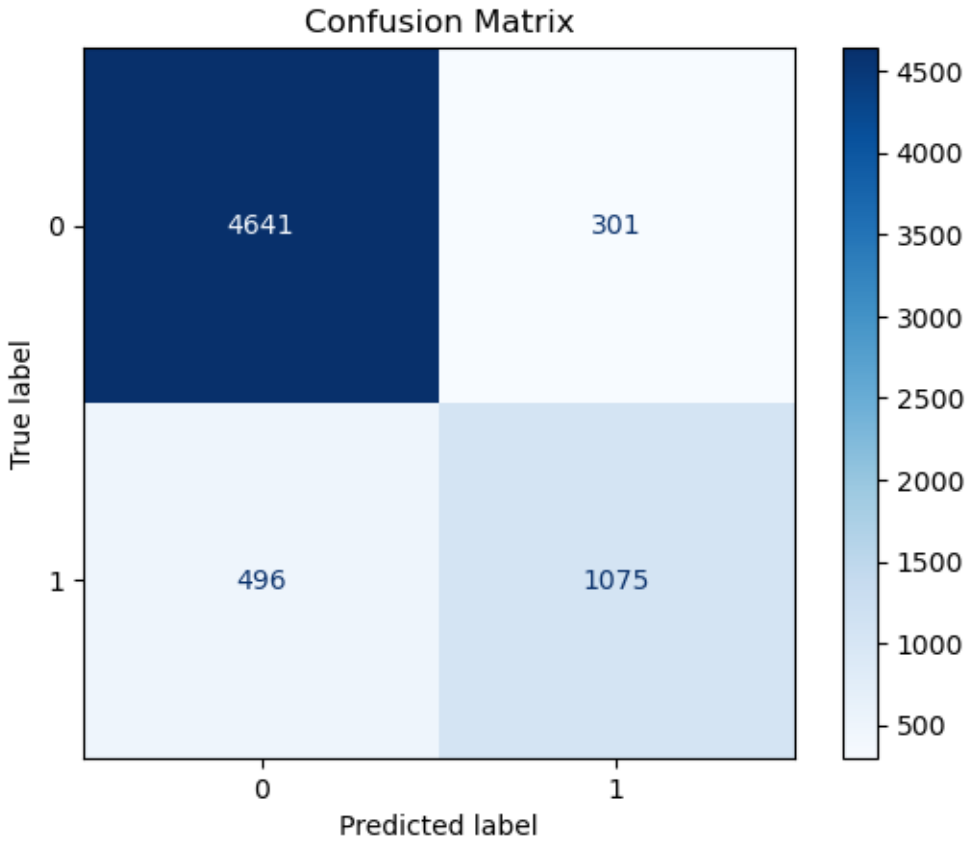
[42]:
```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Define the parameter grid for the randomized search
param_dist = {
    'max_depth': [5, 10, 15],              # Depth of the tree
    'n_estimators': [100, 500],            # Number of boosting rounds
    'learning_rate': [0.01, 0.05, 0.1],    # Learning rate
    'subsample': [0.8, 1],                 # Fraction of samples to use per tree
    'colsample_bytree': [0.8, 1],          # Fraction of features to use per␣
 ↪tree
    'gamma': [0, 0.1],                     # Regularization term
    'min_child_weight': [1, 5]             # Minimum weight of child nodes
}

# Initialize the XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='binary:logistic',␣
 ↪eval_metric='logloss')

# Set up RandomizedSearchCV with 5-fold cross-validation and 10 iterations
random_search = RandomizedSearchCV(estimator=xgb_model,␣
 ↪param_distributions=param_dist,
                                   n_iter=10, cv=5, verbose=1, n_jobs=-1,␣
 ↪scoring='accuracy', random_state=42)

# Perform the randomized search
random_search.fit(X_train, y_train)

# Output the best parameters and best score
print("Best parameters found: ", random_search.best_params_)
print("Best cross-validation score: ", random_search.best_score_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters found:  {'subsample': 1, 'n_estimators': 500,
'min_child_weight': 1, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0,
'colsample_bytree': 1}
Best cross-validation score:  0.869971012078976
```

[43]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Use the best model from RandomizedSearchCV to make predictions
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



Best Hyperparameters and Model Performance: After performing the randomized search, the best combination of hyperparameters found were:

max_depth: 5 n_estimators: 500 learning_rate: 0.1 subsample: 1 colsample_bytree: 1 gamma: 0 min_child_weight: 1 These parameters led to a best cross-validation accuracy score of 0.8701, meaning that the model correctly predicted approximately 87% of the instances in the cross-validation process.

Conclusion: By using RandomizedSearchCV with a carefully selected parameter grid, we efficiently found a set of hyperparameters that optimized the performance of the XGBoost model. This model shows strong performance with an accuracy of 87% on the training data, which suggests that it has learned well from the patterns in the data. Further validation on unseen data would be necessary to assess its generalizability.

1.5 Reflections

1. Introduction and Initial Insights This project explored a dataset of census information to investigate patterns and relationships, particularly in predicting whether an individual earns more or less than $50,000 annually. The project involved preprocessing the dataset, exploratory data analysis, and experimenting with multiple machine learning models to understand the factors influencing income levels.

One key observation during the exploratory analysis was the disproportionate distribution of income levels: around 76% of individuals earned <=$50K, while only 24% earned >$50K. This imbalance required careful consideration during model building to ensure accurate performance across both income groups.

2. Data Exploration The initial data exploration revealed some expected and surprising patterns:

Capital Gain: Individuals with significantly high capital gains predominantly belonged to the >$50K group. Education: Higher education levels, particularly degrees like Master's or Doctorate, correlated strongly with higher income. Age: Older individuals tended to fall into the >$50K category. Marital Status and Occupation: Married individuals and those in executive or professional roles had a clear positive correlation with higher income. Visualizations, including pair plots and boxplots, helped uncover these relationships. Boxplots of numeric features (e.g., education level, age, and hours worked per week) stratified by income category provided additional clarity.

3. Model Development 3.1 Logistic Regression The first model employed logistic regression. Despite the simplicity of this approach, it provided valuable insights:

Coefficients confirmed the importance of features like occupation (executive roles), marital status (married individuals), and education level. The model achieved an accuracy of ~85%, with precision and recall metrics indicating better performance on the <=$50K group. However, the logistic regression model faced limitations, particularly with recall on the >$50K group, suggesting room for improvement.

3.2 Decision Tree A basic decision tree model with limited complexity (max 10 leaf nodes) was employed to understand the most critical factors contributing to income.

Key Insights: The model identified marital status, education, and capital gain as the most important features. Performance: Cross-validation yielded an accuracy of ~84%, similar to logistic regression but with slightly better recall on the >$50K group. 3.3 XGBoost Given the need for a more robust model, I explored XGBoost, a popular gradient boosting algorithm:

Default Hyperparameters: Using default settings, XGBoost significantly outperformed previous models, achieving an average accuracy of ~87% and better recall for the >$50K group. Hyperparameter Tuning: A randomized search optimized parameters like tree depth, learning rate, and number of estimators. This further improved accuracy and generalizability. 4. Key Takeaways Feature Importance: Across all models, marital status, education, and capital gain consistently emerged as top predictors of income. This aligns with expectations but also highlights the potential of advanced features (e.g., work hours and specific occupations) to improve predictions further.

Class Imbalance: Balancing the skewed income distribution remains a challenge. Advanced techniques like oversampling or cost-sensitive algorithms may be beneficial in future iterations.

Model Evolution: While logistic regression provided a baseline, advanced tree-based methods like XGBoost demonstrated their superiority in handling complex, non-linear relationships in the data.

5. Next Steps Model Optimization: Implement strategies like SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance. Advanced Algorithms: Experiment with ensemble methods like Random Forest and CatBoost for further performance improvements. Feature Engineering: Explore interactions between features (e.g., education and occupation) to enhance the predictive power.

Conclusion This project has been an exciting journey into census data analysis and machine learning modeling. It reinforced the importance of data preprocessing, thoughtful model selection, and iterative refinement to achieve meaningful insights and robust predictions.