In [98]:
```
import pandas as pd

df = pd.read_csv(r"C:\Users\User\Downloads\archive\adult.csv")
df_viz = df.copy()

df.info()
df.head()

df.isnull().sum()
df.dropna(inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [100…
```
for col in df.select_dtypes(include=["object"]).columns:
    df[col] = df[col].fillna(df[col].mode().iloc[0])

df.loc[:, df.select_dtypes(include=["number"]).columns] = df.select_dtypes(include=

print(df.isnull().sum())

print(df.select_dtypes(include=["object"]).columns)
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
Index(['workclass', 'education', 'marital.status', 'occupation',
       'relationship', 'race', 'sex', 'native.country', 'income'],
      dtype='object')
```

In [102…

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df["income"] = le.fit_transform(df["income"])
df["sex"] = le.fit_transform(df["sex"])

df = pd.get_dummies(df, columns=["workclass", "occupation", "education", "marital.s

df.head()
df.dtypes

df["hours.per.week"] = df["hours.per.week"].apply(lambda x: 80 if x > 80 else x)

print(df.describe())
print(df.nunique())

from sklearn.preprocessing import StandardScaler

# Instantiate the scaler
scaler = StandardScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(df[['capital.gain', 'capital.loss', 'hours.per.w

# Convert it back to a DataFrame if needed
scaled_df = pd.DataFrame(scaled_data, columns=['capital.gain', 'capital.loss', 'hou
```

```
                age        fnlwgt  education.num           sex  capital.gain  \
count  32561.000000  3.256100e+04   32561.000000  32561.000000  32561.000000
mean      38.581647  1.897784e+05      10.080679      0.669205   1077.648844
std       13.640433  1.055500e+05       2.572720      0.470506   7385.292085
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.178270e+05       9.000000      0.000000      0.000000
50%       37.000000  1.783560e+05      10.000000      1.000000      0.000000
75%       48.000000  2.370510e+05      12.000000      1.000000      0.000000
max       90.000000  1.484705e+06      16.000000      1.000000  99999.000000

       capital.loss  hours.per.week        income  workclass_Federal-gov  \
count  32561.000000    32561.000000  32561.000000           32561.000000
mean      87.303830       40.357329      0.240810               0.029483
std      402.960219       12.034700      0.427581               0.169159
min        0.000000        1.000000      0.000000               0.000000
25%        0.000000       40.000000      0.000000               0.000000
50%        0.000000       40.000000      0.000000               0.000000
75%        0.000000       45.000000      0.000000               0.000000
max     4356.000000       80.000000      1.000000               1.000000

       workclass_Local-gov  ...  native.country_Portugal  \
count         32561.000000  ...             32561.000000
mean              0.064279  ...                 0.001136
std               0.245254  ...                 0.033691
min               0.000000  ...                 0.000000
25%               0.000000  ...                 0.000000
50%               0.000000  ...                 0.000000
75%               0.000000  ...                 0.000000
max               1.000000  ...                 1.000000

       native.country_Puerto-Rico  native.country_Scotland  \
count                32561.000000             32561.000000
mean                     0.003501                 0.000369
std                      0.059068                 0.019194
min                      0.000000                 0.000000
25%                      0.000000                 0.000000
50%                      0.000000                 0.000000
75%                      0.000000                 0.000000
max                      1.000000                 1.000000

       native.country_South  native.country_Taiwan  native.country_Thailand  \
count          32561.000000           32561.000000             32561.000000
mean               0.002457               0.001566                 0.000553
std                0.049507               0.039546                 0.023506
min                0.000000               0.000000                 0.000000
25%                0.000000               0.000000                 0.000000
50%                0.000000               0.000000                 0.000000
75%                0.000000               0.000000                 0.000000
max                1.000000               1.000000                 1.000000

       native.country_Trinadad&Tobago  native.country_United-States  \
count                    32561.000000                  32561.000000
mean                         0.000584                      0.895857
std                          0.024149                      0.305451
min                          0.000000                      0.000000
25%                          0.000000                      1.000000
```

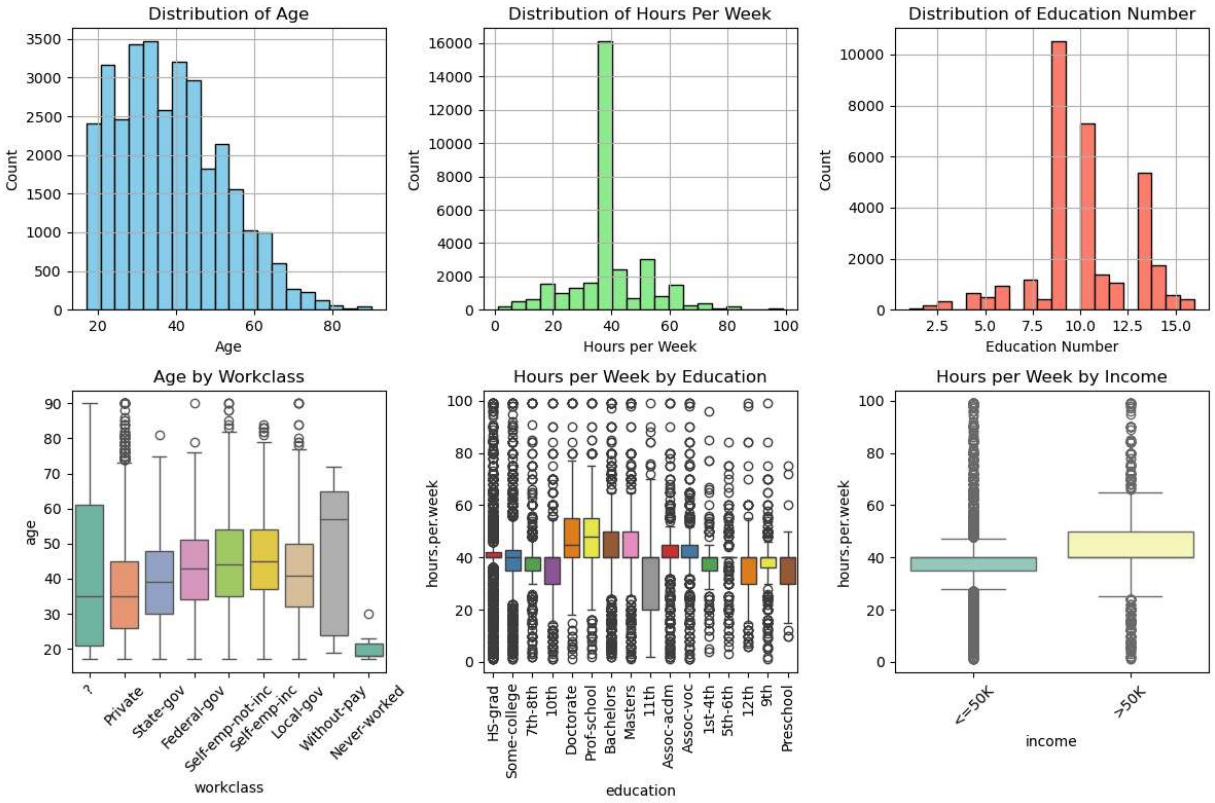
```
50%                          0.000000                     1.000000
75%                          0.000000                     1.000000
max                          1.000000                     1.000000

       native.country_Vietnam  native.country_Yugoslavia
count            32561.000000               32561.000000
mean                 0.002058                   0.000491
std                  0.045316                   0.022162
min                  0.000000                   0.000000
25%                  0.000000                   0.000000
50%                  0.000000                   0.000000
75%                  0.000000                   0.000000
max                  1.000000                   1.000000

[8 rows x 101 columns]
age                                73
fnlwgt                          21648
education.num                      16
sex                                 2
capital.gain                      119
                                  ...
native.country_Thailand             2
native.country_Trinadad&Tobago      2
native.country_United-States        2
native.country_Vietnam              2
native.country_Yugoslavia           2
Length: 101, dtype: int64
```

In [55]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the figure size
plt.figure(figsize=(12, 8))

# Plot 1: Histogram for 'age'
plt.subplot(2, 3, 1)  # (rows, columns, index)
df_viz['age'].hist(bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')

# Plot 2: Histogram for 'hours-per-week'
plt.subplot(2, 3, 2)
df_viz['hours.per.week'].hist(bins=20, color='lightgreen', edgecolor='black')
plt.title('Distribution of Hours Per Week')
plt.xlabel('Hours per Week')
plt.ylabel('Count')

# Plot 3: Histogram for 'education-num' (or any other interesting variable)
plt.subplot(2, 3, 3)
df_viz['education.num'].hist(bins=20, color='salmon', edgecolor='black')  # replace
plt.title('Distribution of Education Number')
plt.xlabel('Education Number')
plt.ylabel('Count')

# Plot 4: Boxplot of 'age' by 'workclass'
```

```
plt.subplot(2, 3, 4)
sns.boxplot(x='workclass', y='age', data=df_viz, palette='Set2')
plt.xticks(rotation=45)
plt.title('Age by Workclass')

# Plot 5: Boxplot of 'hours-per-week' by 'education'
plt.subplot(2, 3, 5)
sns.boxplot(x='education', y='hours.per.week', data=df_viz, palette='Set1')
plt.xticks(rotation=90)
plt.title('Hours per Week by Education')

# Plot 6: Boxplot of 'hours-per-week' by 'income' (or 'sex')
plt.subplot(2, 3, 6)
sns.boxplot(x='income', y='hours.per.week', data=df_viz, palette='Set3')  # Change
plt.xticks(rotation=45)
plt.title('Hours per Week by Income')

# Tight layout for better spacing
plt.tight_layout()
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_14432\3476556393.py:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='workclass', y='age', data=df_viz, palette='Set2')
C:\Users\User\AppData\Local\Temp\ipykernel_14432\3476556393.py:36: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='education', y='hours.per.week', data=df_viz, palette='Set1')
C:\Users\User\AppData\Local\Temp\ipykernel_14432\3476556393.py:42: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='income', y='hours.per.week', data=df_viz, palette='Set3')  # Change
to 'sex' if needed
```



In [57]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df["income"] = le.fit_transform(df["income"])
df["sex"] = le.fit_transform(df["sex"])

df = pd.get_dummies(df, columns=["workclass", "occupation", "education", "marital.s

print(df.head())
print(df.dtypes)
```

```
   age  fnlwgt  education.num  sex  capital.gain  capital.loss  \
0   90   77053              9    0             0          4356
1   82  132870              9    0             0          4356
2   66  186061             10    0             0          4356
3   54  140359              4    0             0          3900
4   41  264663             10    0             0          3900

   hours.per.week  income  workclass_Federal-gov  workclass_Local-gov  ...  \
0              40       0                      0                    0  ...
1              18       0                      0                    0  ...
2              40       0                      0                    0  ...
3              40       0                      0                    0  ...
4              40       0                      0                    0  ...

   native.country_Portugal  native.country_Puerto-Rico  \
0                        0                           0
1                        0                           0
2                        0                           0
3                        0                           0
4                        0                           0

   native.country_Scotland  native.country_South  native.country_Taiwan  \
0                        0                     0                      0
1                        0                     0                      0
2                        0                     0                      0
3                        0                     0                      0
4                        0                     0                      0

   native.country_Thailand  native.country_Trinadad&Tobago  \
0                        0                               0
1                        0                               0
2                        0                               0
3                        0                               0
4                        0                               0

   native.country_United-States  native.country_Vietnam  \
0                             1                       0
1                             1                       0
2                             1                       0
3                             1                       0
4                             1                       0

   native.country_Yugoslavia
0                           0
1                           0
2                           0
3                           0
4                           0

[5 rows x 101 columns]
age                             int64
fnlwgt                          int64
education.num                   int64
sex                             int32
capital.gain                    int64
                                ...
```

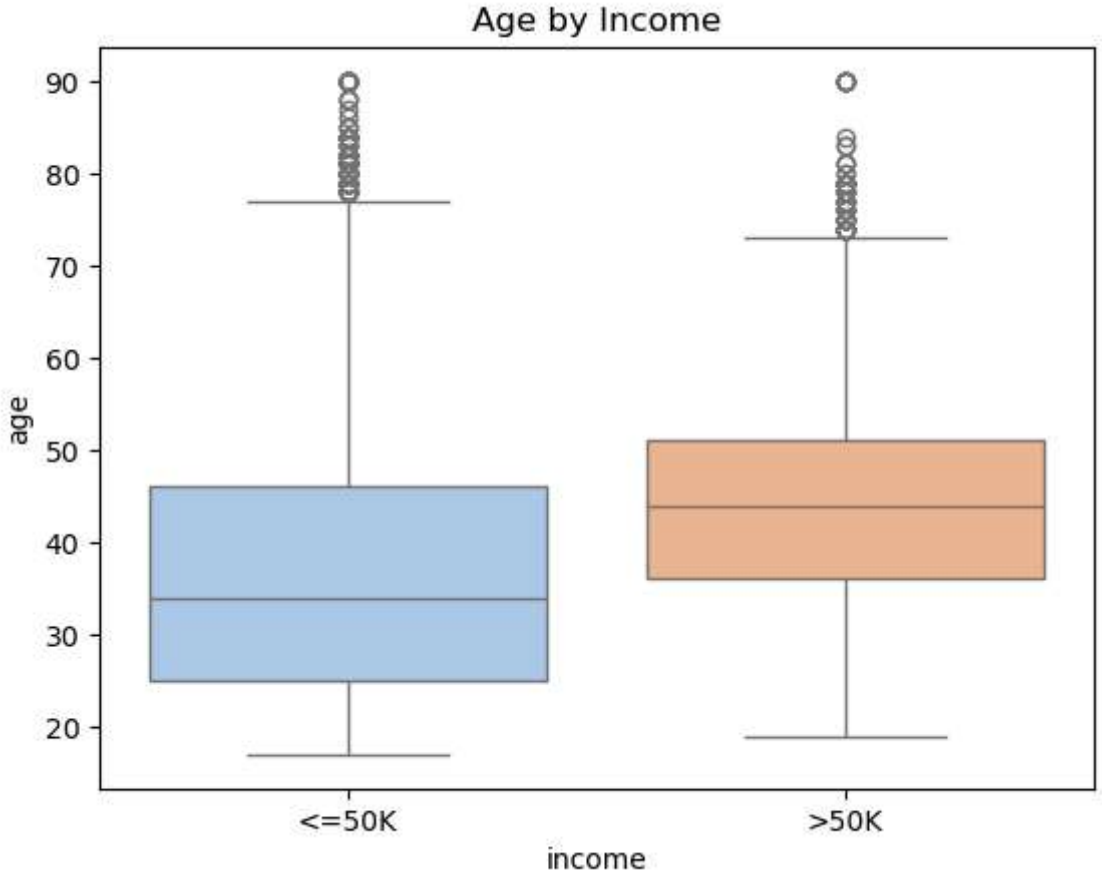
```
native.country_Thailand           int32
native.country_Trinadad&Tobago    int32
native.country_United-States      int32
native.country_Vietnam            int32
native.country_Yugoslavia         int32
Length: 101, dtype: object
```

In [59]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers_before = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()
print(outliers_before.sort_values(ascending=False))
```

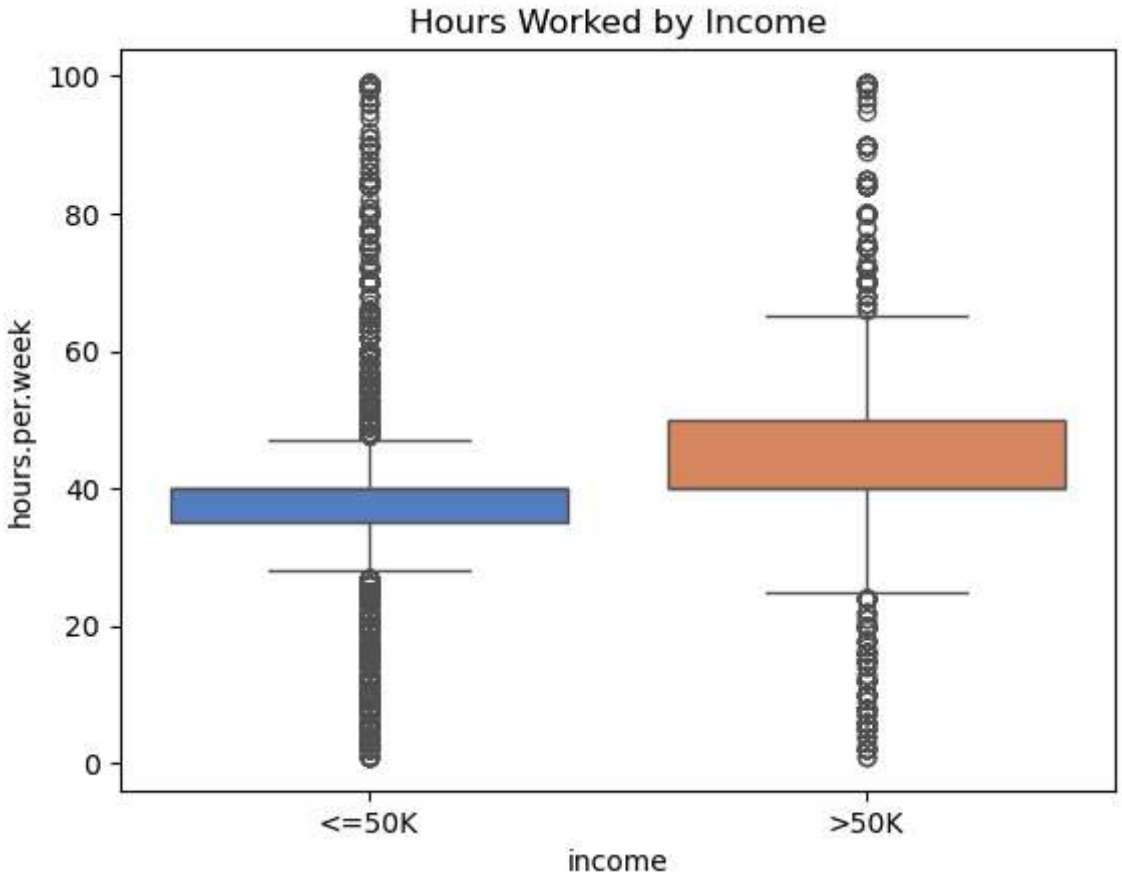
```
hours.per.week                    9008
income                            7841
education_Some-college            7291
education_Bachelors               5355
relationship_Own-child            5068
                                  ...
marital.status_Never-married         0
education_HS-grad                    0
sex                                  0
marital.status_Married-civ-spouse    0
relationship_Not-in-family           0
Length: 101, dtype: int64
```

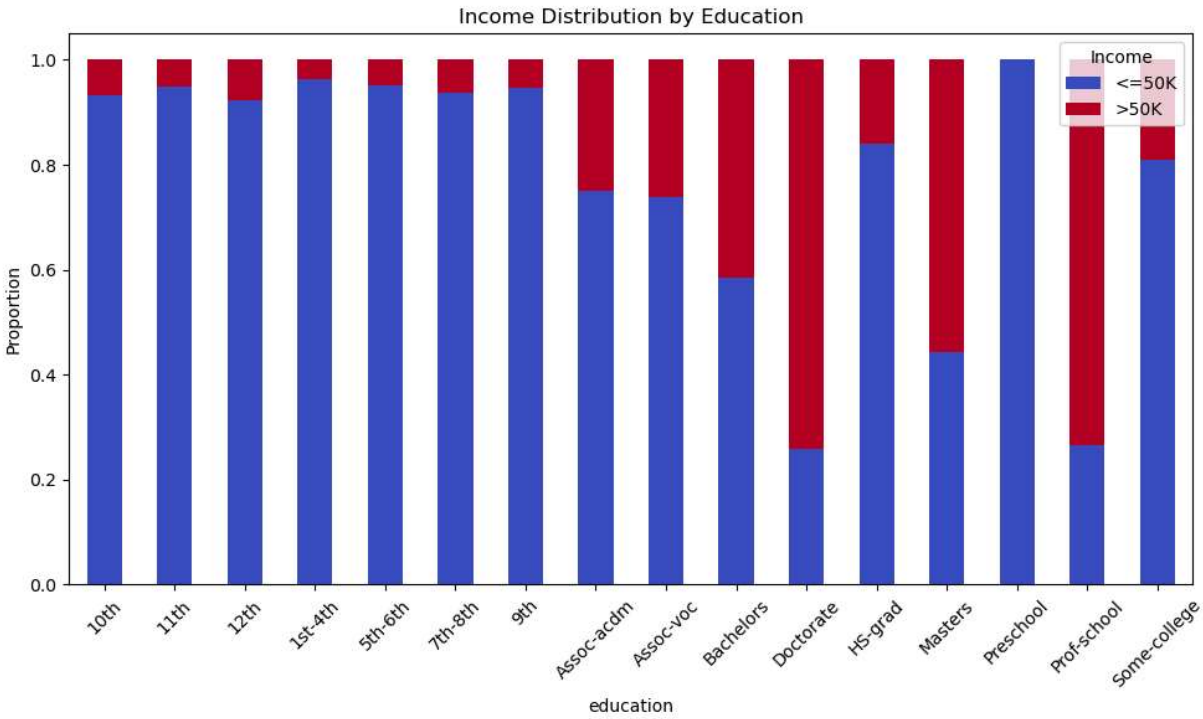
In [61]:
```
sns.boxplot(x='income', y='age', data=df_viz, palette='pastel')
plt.title('Age by Income')
plt.show()

sns.boxplot(x='income', y='hours.per.week', data=df_viz, palette='muted')
plt.title('Hours Worked by Income')
plt.show()

# First, calculate the income distribution for each education level
edu_income = df_viz.groupby(['education', 'income']).size().unstack()

# Normalize to get proportions
edu_income_prop = edu_income.div(edu_income.sum(axis=1), axis=0)

# Plot
edu_income_prop.plot(kind='bar', stacked=True, colormap='coolwarm', figsize=(10,6))
plt.title('Income Distribution by Education')
plt.ylabel('Proportion')
plt.xticks(rotation=45)
plt.legend(title='Income')
plt.tight_layout()
plt.show()

sns.boxplot(x='workclass', y='hours.per.week', hue='income', data=df_viz)
plt.xticks(rotation=45)
plt.title('Hours Worked by Workclass and Income')
plt.show()
```

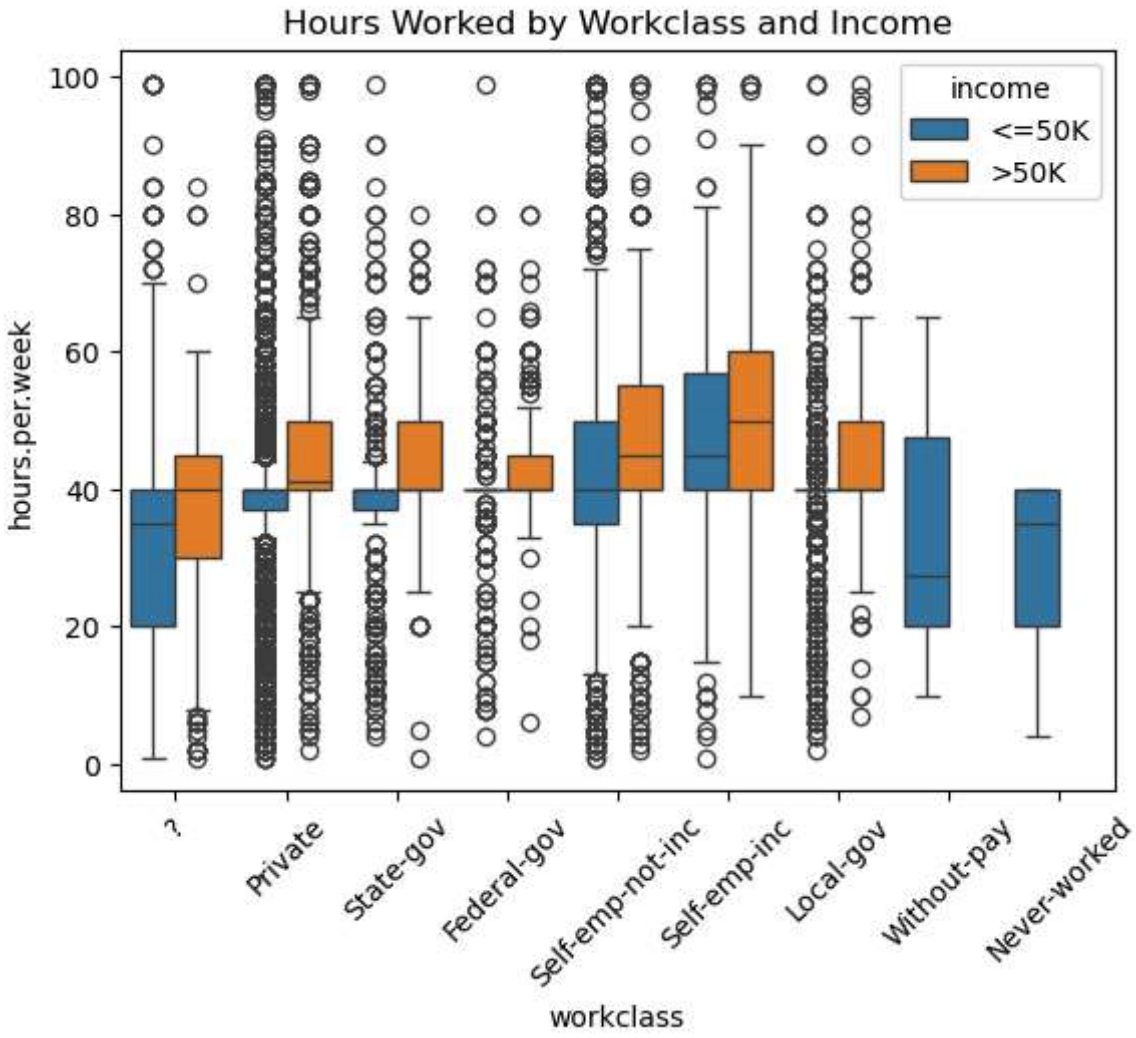
```
C:\Users\User\AppData\Local\Temp\ipykernel_14432\1390696377.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='income', y='age', data=df_viz, palette='pastel')
```




```
C:\Users\User\AppData\Local\Temp\ipykernel_14432\1390696377.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='income', y='hours.per.week', data=df_viz, palette='muted')
```

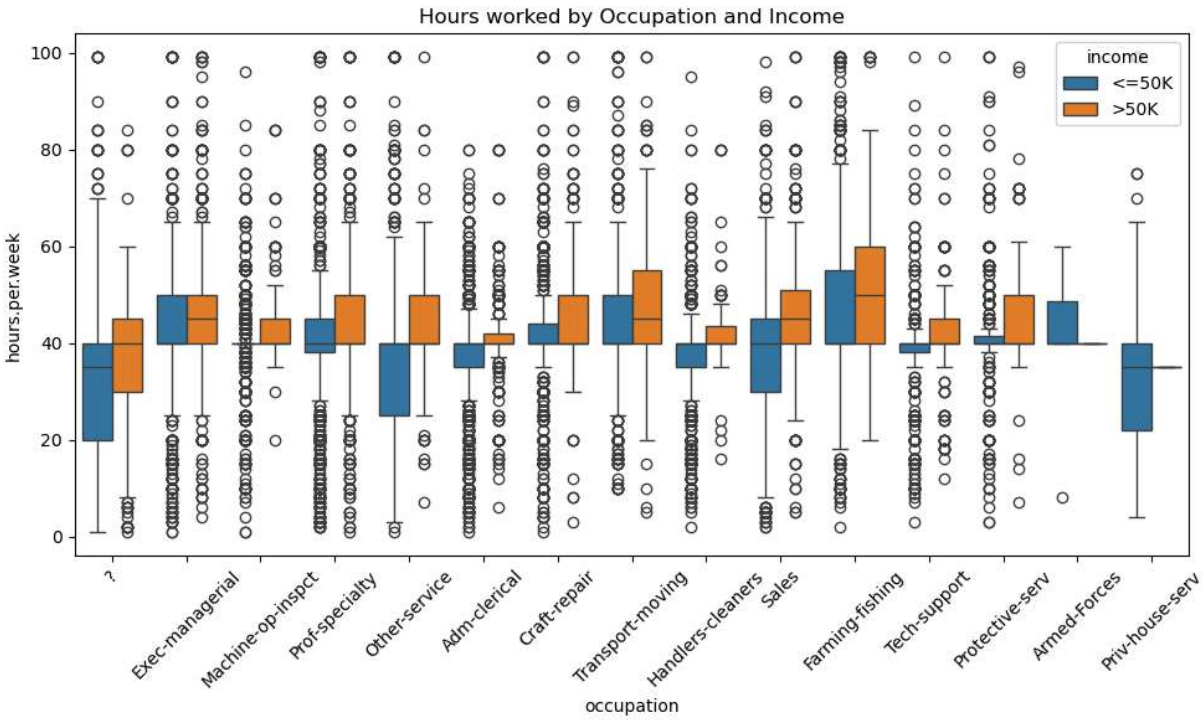

Hours Worked by Income
hours.per.week
100
80
60
40
20
0
<=50K
>50K
income

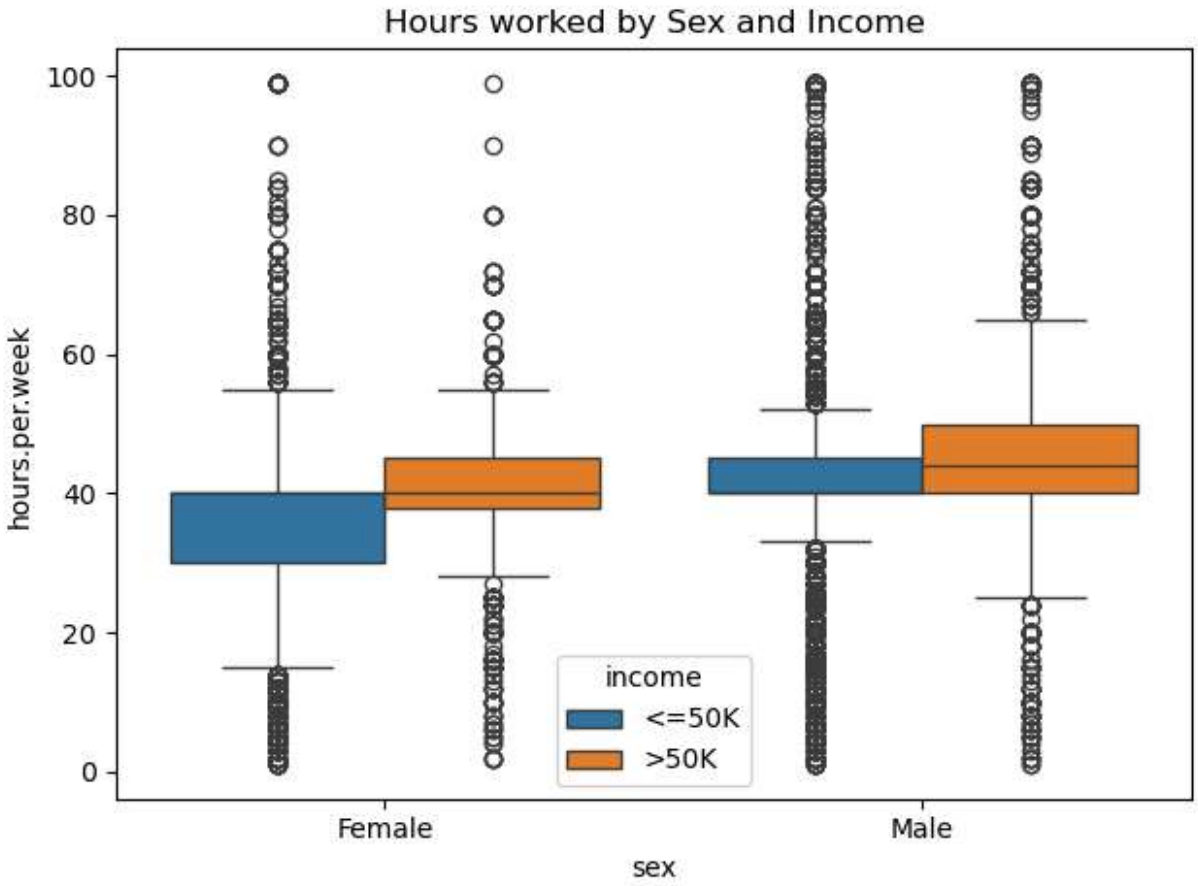

Income Distribution by Education
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
Income
<=50K
>50K
10th
11th
12th
1st-4th
5th-6th
7th-8th
9th
Assoc-acdm
Assoc-voc
Bachelors
Doctorate
HS-grad
Masters
Preschool
Prof-school
Some-college
education

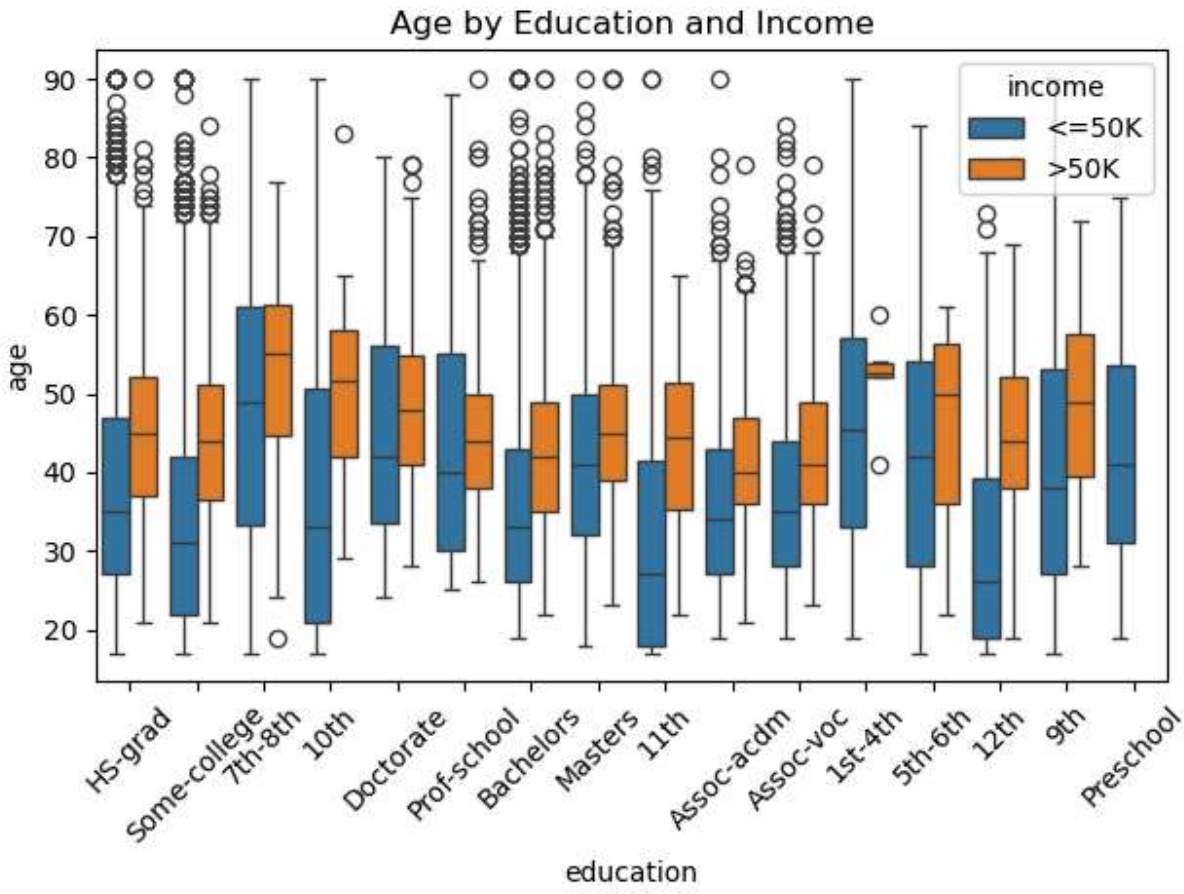


In [63]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set up plot size for all
plt.figure(figsize=(10,6))

# 1. Occupation by income
sns.boxplot(x='occupation', y='hours.per.week', hue='income', data=df_viz)
plt.xticks(rotation=45)
plt.title('Hours worked by Occupation and Income')
plt.tight_layout()
plt.show()

# 2. Sex by income
sns.boxplot(x='sex', y='hours.per.week', hue='income', data=df_viz)
plt.title('Hours worked by Sex and Income')
plt.tight_layout()
plt.show()

# 3. Education by income
sns.boxplot(x='education', y='age', hue='income', data=df_viz)
plt.xticks(rotation=45)
plt.title('Age by Education and Income')
```

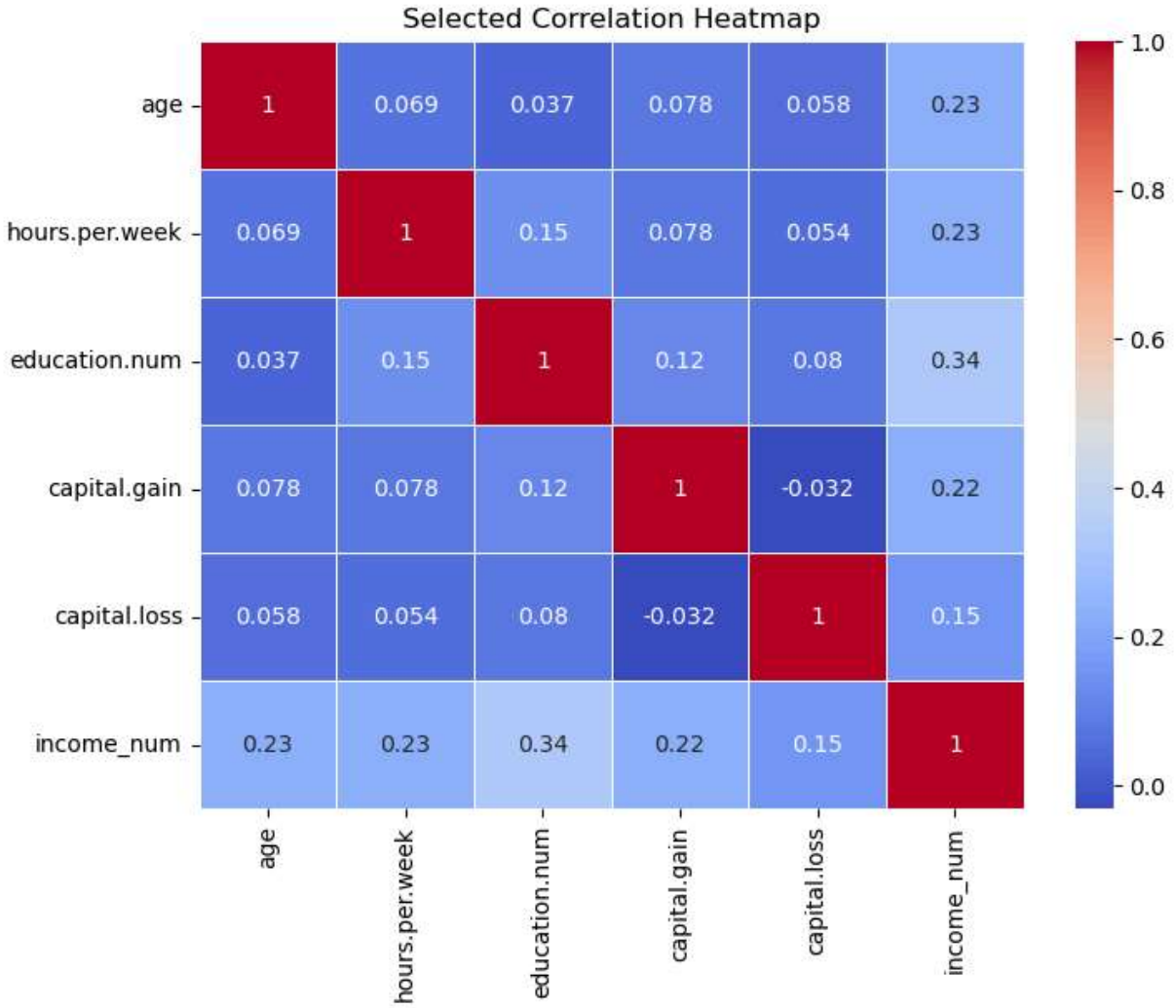
```
plt.tight_layout()
plt.show()
```







In [65]:
```
# Create a numeric version of the income column
df_viz['income_num'] = df_viz['income'].map({'<=50K': 0, '>50K': 1})

# Select specific numeric columns to reduce clutter
selected_cols = ['age', 'hours.per.week', 'education.num', 'capital.gain', 'capital

# Then plot
corr_matrix = df_viz[selected_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Selected Correlation Heatmap')
plt.show()
```



In [67]:
```
from sklearn.model_selection import train_test_split

# Assuming "income" is your target variable
X = df.drop(columns=["income"])
y = df["income"]

# Split data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Training set: {X_train.shape}, Testing set: {X_test.shape}")
```

```
Training set: (26048, 100), Testing set: (6513, 100)
```

In [69]:
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    scale_pos_weight=1.5,  # Adjust for class imbalance
    max_depth=6,
    n_estimators=200,  # More trees for better performance
    learning_rate=0.05,  # Smaller step size for better precision
    subsample=0.8,  # Avoid overfitting
    colsample_bytree=0.8,
```

```
    random_state=42
)

xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

from sklearn.metrics import classification_report
print("🔷 XGBoost Model Performance:")
print(classification_report(y_test, y_pred_xgb))
```

```
🔷 XGBoost Model Performance:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      4976
           1       0.72      0.73      0.73      1537

    accuracy                           0.87      6513
   macro avg       0.82      0.82      0.82      6513
weighted avg       0.87      0.87      0.87      6513
```

In [71]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

# Define the model with base parameters
xgb_base = XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    use_label_encoder=False,
    random_state=42
)

# Define the parameter grid
param_grid = {
    "n_estimators": [100, 200, 300],  # Number of trees
    "max_depth": [4, 6, 8],  # Tree depth
    "learning_rate": [0.01, 0.05, 0.1],  # Step size
    "subsample": [0.7, 0.8, 0.9],  # % of data used in training
    "colsample_bytree": [0.7, 0.8, 0.9],  # % of features used
    "scale_pos_weight": [1, 1.5, 2]  # Handle class imbalance
}

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(
    xgb_base, param_grid, cv=3, scoring="precision", verbose=2, n_jobs=-1
)
grid_search.fit(X_train, y_train)

# Get the best model
best_xgb = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

# Make predictions
y_pred_xgb = best_xgb.predict(X_test)
```

```
# Print classification report
print("🔷 Fine-Tuned XGBoost Performance:")
print(classification_report(y_test, y_pred_xgb))
```

```
Fitting 3 folds for each of 729 candidates, totalling 2187 fits
C:\Users\User\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [00:
31:17] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best Parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.01, 'max_depth': 4, 'n
_estimators': 100, 'scale_pos_weight': 1, 'subsample': 0.7}
🔷 Fine-Tuned XGBoost Performance:
              precision    recall  f1-score   support

           0       0.82      1.00      0.90      4976
           1       0.96      0.28      0.43      1537

    accuracy                           0.83      6513
   macro avg       0.89      0.64      0.66      6513
weighted avg       0.85      0.83      0.79      6513
```

In [81]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Example: assuming 'income' is your target column
X = df.drop('income', axis=1)  # All columns except target
y = df['income']               # The target column

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

print("Random Forest model Performance:")
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      4976
           1       0.71      0.61      0.65      1537

    accuracy                           0.85      6513
   macro avg       0.80      0.77      0.78      6513
weighted avg       0.84      0.85      0.84      6513
```

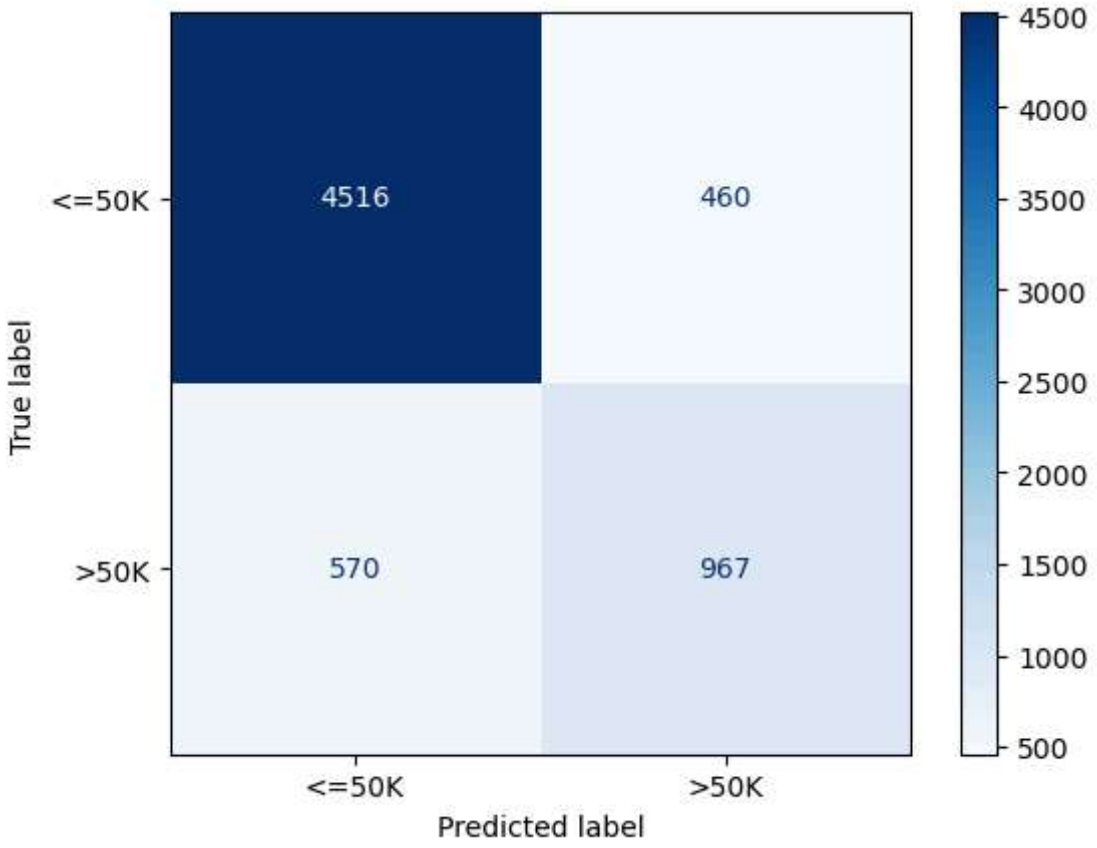
In [84]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define parameter grid
param_grid = {
    "n_estimators": [50, 100, 200],
```

```
    "max_depth": [10, 20, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Apply GridSearchCV
grid_search = GridSearchCV(rf, param_grid, cv=3, scoring="f1_macro", n_jobs=-1, ver
grid_search.fit(X_train_balanced, y_train_balanced)

# Print best parameters
print("Best Parameters:", grid_search.best_params_)

# Train model with best parameters
best_rf = grid_search.best_estimator_
y_pred_best = best_rf.predict(X_test)

# Evaluate new model
from sklearn.metrics import classification_report
print("Tuned Model Performance:")
print(classification_report(y_test, y_pred_best))
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n
_estimators': 100}
Tuned Model Performance:
              precision    recall  f1-score   support

           0       0.91      0.86      0.89      4976
           1       0.62      0.73      0.67      1537

    accuracy                           0.83      6513
   macro avg       0.77      0.80      0.78      6513
weighted avg       0.84      0.83      0.84      6513
```

In [85]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Assuming y_test and y_pred_balanced from your fine-tuned model
cm = confusion_matrix(y_test, y_pred_balanced)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["<=50K", ">50K"]
disp.plot(cmap='Blues')
```

Out[85]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16c0ad02300>
```


4516
460
570
967
<=50K
>50K
True label
<=50K
>50K
Predicted label
4500
4000
3500
3000
2500
2000
1500
1000
500

In [ ]: