Adult Income Analysis

DESCRIPTION:

This data was extracted from the 1994 Census bureau database based on the following conditions:

1- Age > 16

2- Final Weight (fnlwgt) > 1

3- Hours per week > 0

OBJECTIVE:

The prediction task is to determine whether a person makes over $50K a year.

DESCRIBING ATTRIBUTES:

1) **fnlwgt(final weight)** : *The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau.*

2) **education.num** : *This attribute orders the education's levels ascendingly from 1 to 16.*

3) **income** : *Represents two categories (whether an adult citizen gains more than 50k per year or less).*

DATA PREPROCESSING

(1)Checking and removing nulls as “?”.

(2)Checking for duplications: No duplications found.

```
[44] # Remove missing or duplicated values
     data.replace("?", np.nan, inplace=True)
     data.dropna(inplace=True)
     data.duplicated() #none
```

```
1        False
3        False
4        False
5        False
6        False
         ...
32556    False
32557    False
32558    False
32559    False
32560    False
Length: 30162, dtype: bool
```

First, we used a copy of the data to be transformed and cleaned for prediction :

-Removing unnecessary columns:

1)"education": because we have it encoded into "education.num".

2)"fnlwgt": because its a derived attribute from "race","age","sex" and "native.country".

```
[45] data_e=data.copy()
     col_to_drop=['fnlwgt', 'education'] # dropping innecessary columns
     data_e.drop(col_to_drop, inplace=True, axis=1)
```

(3)Data transformation:

converting categorical data into numerical values:

-"income" is encoded ordinally, as its values are ranked .

-the other categorical attributes are encoded using OneHotEncoder, as their values are not ranked.

```
[46] #################### Convert categorical variables into numerical variables by Label encoding ################################
     Oe = OrdinalEncoder()
     data_e[['income']] = Oe.fit_transform(data[['income']]) # encoding income


     #encoding categorical attributes
     from sklearn.preprocessing import OneHotEncoder
     categorical_cols=['marital.status','relationship','sex','workclass','occupation','race','native.country']
     OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
     OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(data_e[categorical_cols]))
     OH_cols_train.index = data_e.index #one-hot encoding removed index; put it back
     num_X_train = data_e.drop(categorical_cols, axis=1) #remove categorical columns (will replace with one-hot encoding)
     data_e = pd.concat([num_X_train, OH_cols_train], axis=1) #add one-hot encoded columns to numerical features
```

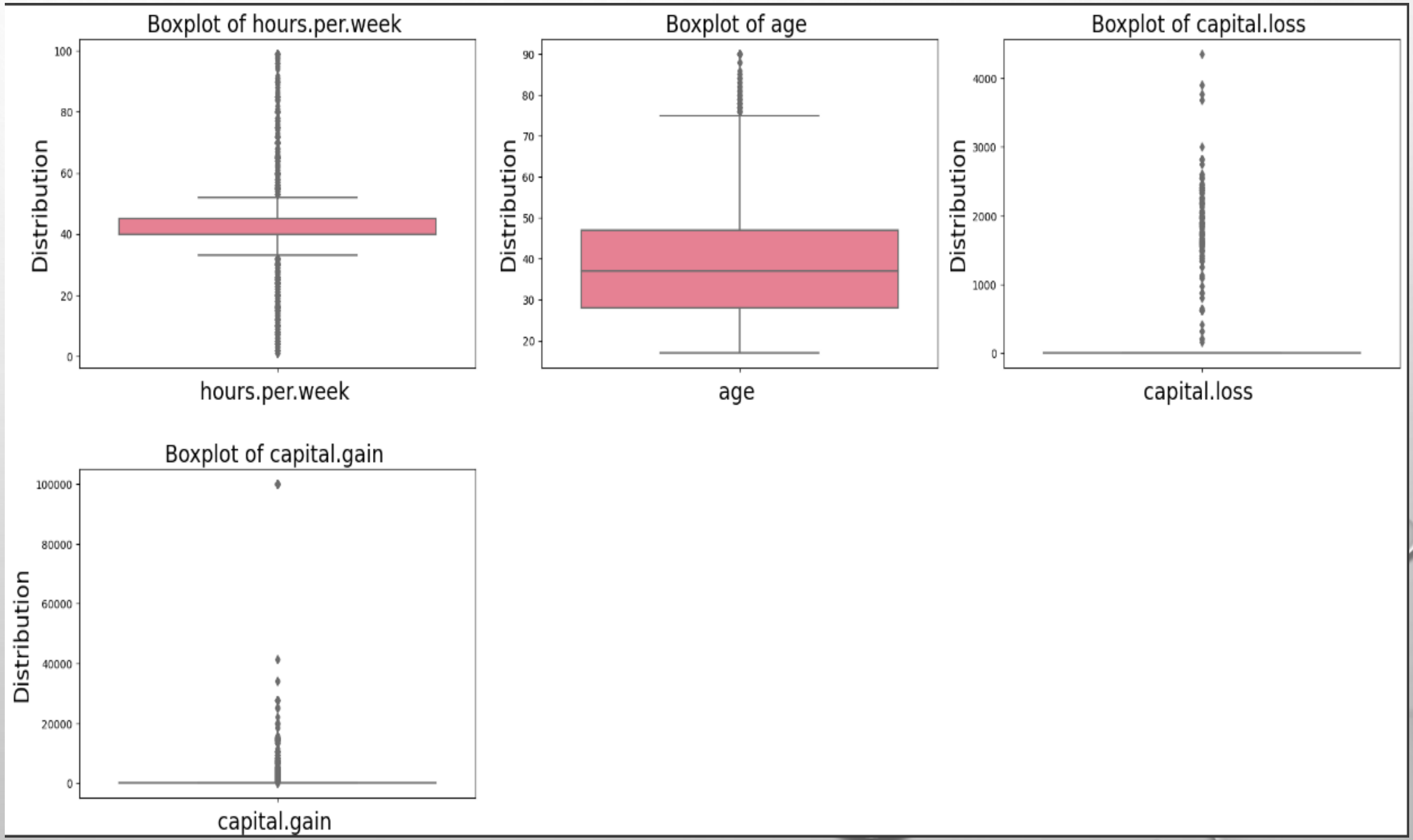
(4)Outliers detection :

-Checking for the outliers of the numerical attributes.

-Without removing the outliers as they are part of the actual data.

```
[48] ######################################## Checking for Outliers ########################
plt.figure(figsize=(20, 20))
numeric_cols = data_e[['hours.per.week','age','capital.loss','capital.gain']]

for i, column in enumerate(numeric_cols.columns):
    plt.subplot(4, 3, i + 1)
    sns.boxplot(y=column, data=data_e, palette='husl')
    plt.xlabel('{} \n'.format(column), fontsize=20)
    plt.ylabel('Distribution', fontsize=20)
    plt.title('Boxplot of {} '.format(column), fontsize=20)
plt.tight_layout()
plt.show()
```



Another reason for not removing the outliers, the attributes have a kind of good correlation with the "income".

```python
plt.figure(figsize=(7,3))
corr = data_e.iloc[:, :6].corr(numeric_only = True)
sns.heatmap(corr, annot=True)
plt.show()
```

SHOWING STATISTICS OF THE DATASET

```
# Display Summary statistics of the dataset
data.describe()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	30162.000000	3.016200e+04	30162.000000	30162.000000	30162.000000	30162.000000
mean	38.437902	1.897938e+05	10.121312	1092.007858	88.372489	40.931238
std	13.134665	1.056530e+05	2.549995	7406.346497	404.298370	11.979984
min	17.000000	1.376900e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.176272e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.784250e+05	10.000000	0.000000	0.000000	40.000000
75%	47.000000	2.376285e+05	13.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

[51]
```
####################################### Displaying Mode ####################################
coco = data[['age', 'workclass', 'fnlwgt', 'education.num', 'marital.status', 'occupation', 'relationship',
             'race', 'sex', 'capital.gain', 'capital.loss','hours.per.week', 'native.country', 'income']]
for i, co in enumerate(coco.columns):
    print("Mode value of [", co, "] is:", st.mode(data[co]))
```

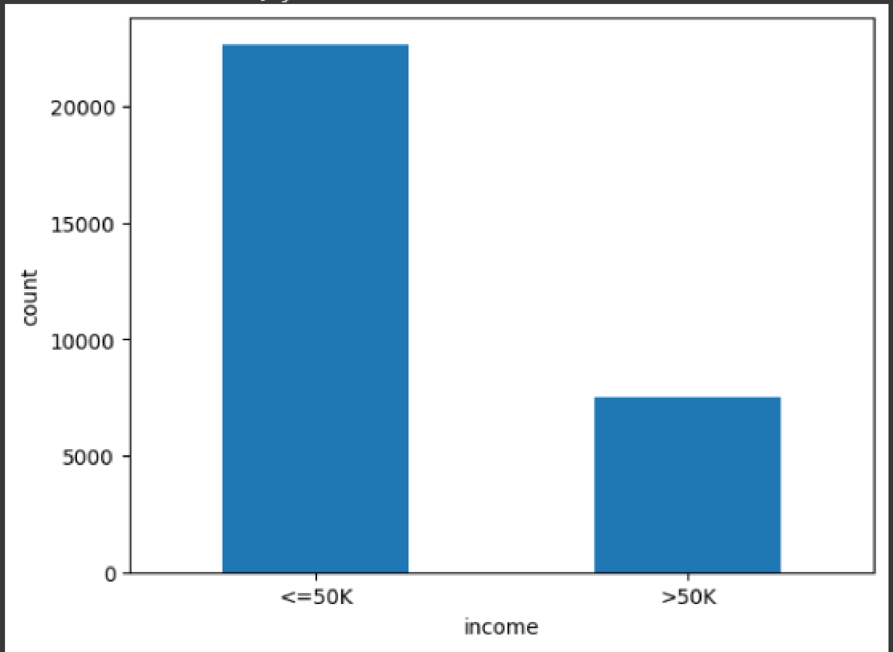
```
Mode value of [ age ] is: 36
Mode value of [ workclass ] is: Private
Mode value of [ fnlwgt ] is: 203488
Mode value of [ education.num ] is: 9
Mode value of [ marital.status ] is: Married-civ-spouse
Mode value of [ occupation ] is: Prof-specialty
Mode value of [ relationship ] is: Husband
Mode value of [ race ] is: White
Mode value of [ sex ] is: Male
Mode value of [ capital.gain ] is: 0
Mode value of [ capital.loss ] is: 0
Mode value of [ hours.per.week ] is: 40
Mode value of [ native.country ] is: United-States
Mode value of [ income ] is: <=50K
```

DATA VISUALIZATION:

(1)View the number and percentage of the people that their income is less than 50k vs greater than 50k.

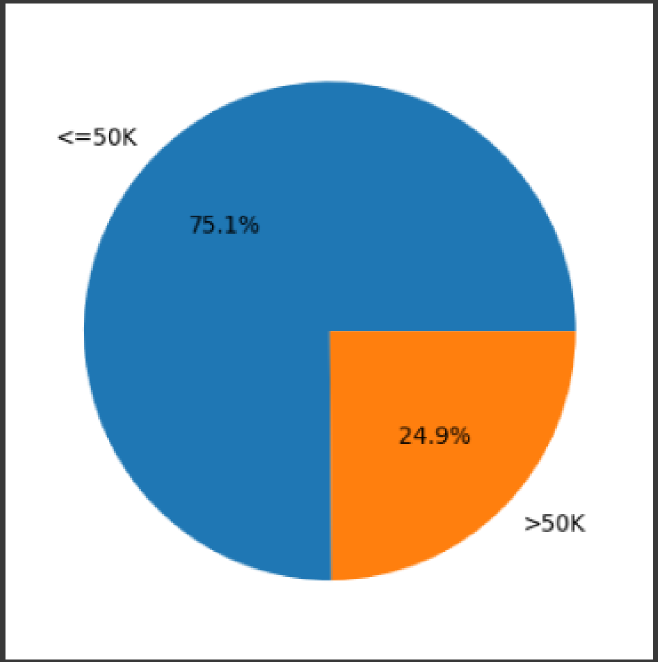
```
[52] ################################################## Visualizations ##################################
     # Bar Chart
     data['income'].value_counts().plot(kind='bar',xlabel='income',ylabel='count',rot=0)
```

<Axes: xlabel='income', ylabel='count'>




```
[53] # Pie chart
     data.groupby('income').size().plot(kind='pie',autopct='%1.1f%%')
```

<Axes: >



(2)The relation between “income” and the other categorical attributes

```python
# Grouped Countplot of each attribute with [income]
plt.figure(figsize=(30, 30))
cat_cols = data[['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.country']]

for i, column in enumerate(cat_cols.columns):
    plt.subplot(4, 3, i + 1)
    sns.countplot(y=column, hue='income', data=data, palette='husl')
    plt.xlabel(column, fontsize=20)
    plt.ylabel('Count', fontsize=20)
    plt.title('Countplot of {} by income'.format(column), fontsize=20)
plt.tight_layout()
plt.show()
```

(3)*The relation among "income" ,"education" and "age".*

Observation:

Most of the people with education (doctorate or prof-school)and age in the interval[30:80], gain more than 50k per year.

PREDICTIVE ANALYTICS:

(1)Preparation for machine learning models
by splitting data into 80% train and 20% test.

```
[56] ############################# prepare the dataset for machine learning models #####################
     X = data_e.drop(['income'], axis=1)
     X.columns = X.columns.astype(str)
     y = data_e['income']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

Decision Tree

```
################################ applying decision tree #####################################
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy (Decision Tree - test):", acc)
y_tr_pr = dt.predict(X_train)
act = accuracy_score(y_train, y_tr_pr)
print("Accuracy (Decision Tree - train):", act)
```

```
Accuracy (Decision Tree - test): 0.8130283441074092
Accuracy (Decision Tree - train): 0.9780347299929545
```

Random Forest

```
[59] ############################## applying random forest ##################################
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy (Random Forest - test):", acc)
y_tr_pr = rf.predict(X_train)
act = accuracy_score(y_train, y_tr_pr)
print("Accuracy (Random Forest - train):", act)
```

```
Accuracy (Random Forest - test): 0.8398806563898558
Accuracy (Random Forest - train): 0.9779932860872809
```

Support Vector Machine

```
[60]
############################### applying SVM ####################################
from sklearn.svm import SVC
model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy (SVM - test):", acc)
y_tr_pr = model.predict(X_train)
act = accuracy_score(y_train, y_tr_pr)
print("Accuracy (SVM - train):", act)
```

```
Accuracy (SVM - test): 0.8436930217139068
Accuracy (SVM - train): 0.8500559492726595
```

K-Nearest Neighbors

```
############################### applying KNN ####################################
model = KNeighborsClassifier(n_neighbors=7)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy (KNN - test):", acc)
y_tr_pr = model.predict(X_train)
act = accuracy_score(y_train, y_tr_pr)
print("Accuracy (KNN - train):", act)
```

```
Accuracy (KNN - test): 0.8229736449527598
Accuracy (KNN - train): 0.8670893945045381
```

Finding the optimal (K)

```
######################## getting the optimal number of neighbors for knn ############################
k_values = range(1, 30)
cv_scores = []
for k in k_values:
    model.n_neighbors = k
    data_e.columns = data_e.columns.astype(str)
    scores = cross_val_score(model, data_e, data['income'], cv=5)
    cv_scores.append(scores.mean())
optimal_k = k_values[cv_scores.index(max(cv_scores))]
print("Optimal number of neighbors:", optimal_k)
```

```
Optimal number of neighbors: 7
```

Conclusion of the predictive techniques

1) *Support Vector Machine: has the highest accuracy, but it's time consuming when applying on very large data (our BEST model).*

2) *Random Forest: has the second highest accuracy, but it comes with a bit of overfitting as the training accuracy is 97.8%.*

3) *K-Nearest Neighbors: has the third highest accuracy, with almost no overfitting.*

4) *Decision Tree: has the lowest accuracy with a big overfitting (training accuracy is 97.8%).*

Thank you