```
import pandas as pd
```

```
data=pd.read_csv("adult 3.csv")
```

```
data.head(10)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United States
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United States
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United States
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United States
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United States
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United States
6	29	?	227026	HS-grad	9	Never-married	?	Unmarried	Black	Male	0	0	40	United States
7	63	Self-emp-	104626	Prof-	15	Married-civ-	Prof-	Husband	White	Male	3103	0	32	United

```
data['fnlwgt'].min()
```

```
np.int64(12285)
```

```
data.tail(3)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	na co
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	U
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	U

```
data.shape
```

```
(48842, 15)
```

```
#null values
data.isna().sum() #mean mdeian mode arbitrary
```

```
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

```
print(data.workclass.value_counts())
```

```
workclass
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
?                    2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: count, dtype: int64
```

```
data.workclass.replace({'?':'Others'},inplace=True)
print(data['workclass'].value_counts())
```

```
workclass
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
Others               2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: count, dtype: int64
C:\Users\afroz\AppData\Local\Temp\ipykernel_6892\4184710730.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data.workclass.replace({'?':'Others'},inplace=True)
```

```
print(data['occupation'].value_counts())
```

```
occupation
Prof-specialty       6172
Craft-repair         6112
Exec-managerial      6086
Adm-clerical         5611
Sales                5504
Other-service        4923
Machine-op-inspct    3022
?                    2809
Transport-moving     2355
Handlers-cleaners    2072
Farming-fishing      1490
Tech-support         1446
Protective-serv       983
Priv-house-serv       242
Armed-Forces           15
Name: count, dtype: int64
```

```
data.occupation.replace({'?':'Others'},inplace=True)
print(data['occupation'].value_counts())
```

```
occupation
Prof-specialty       6172
Craft-repair         6112
Exec-managerial      6086
Adm-clerical         5611
Sales                5504
Other-service        4923
Machine-op-inspct    3022
Others               2809
Transport-moving     2355
Handlers-cleaners    2072
Farming-fishing      1490
Tech-support         1446
Protective-serv       983
Priv-house-serv       242
Armed-Forces           15
Name: count, dtype: int64
C:\Users\afroz\AppData\Local\Temp\ipykernel_6892\1148816719.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me
```

```
    data.occupation.replace({'?':'Others'},inplace=True)
```

```
data=data[data['workclass']!='Without-pay']
data=data[data['workclass']!='Never-worked']
print(data['workclass'].value_counts())
```

```
workclass
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
Others               2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Name: count, dtype: int64
```

```
print(data.relationship.value_counts())
```

```
relationship
Husband           19708
Not-in-family     12582
Own-child          7566
Unmarried          5123
Wife               2327
Other-relative     1505
Name: count, dtype: int64
```

```
print(data.gender.value_counts())
```

```
gender
Male      32629
Female    16182
Name: count, dtype: int64
```

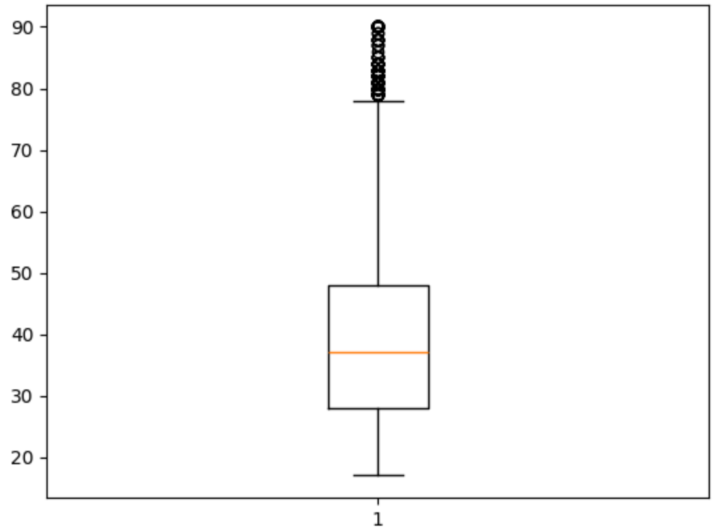
```
data['marital-status'].value_counts()
```

```
marital-status
2    21256
4    15695
0     6464
5     1443
6     1275
3      550
1       37
Name: count, dtype: int64
```

```
data.shape
```

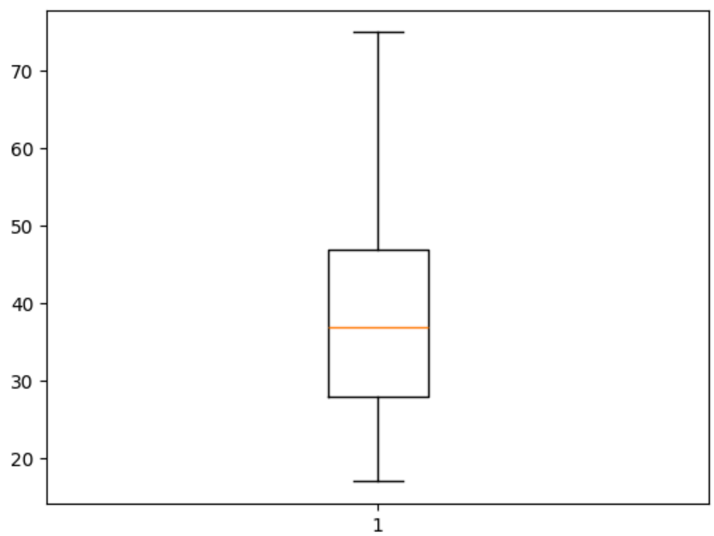
```
(48811, 15)
```

```
#outlier detection
import matplotlib.pyplot as plt   #visualization
plt.boxplot(data['age'])
plt.show()
```




```
data=data[(data['age']<=75)&(data['age']>=17)]
```

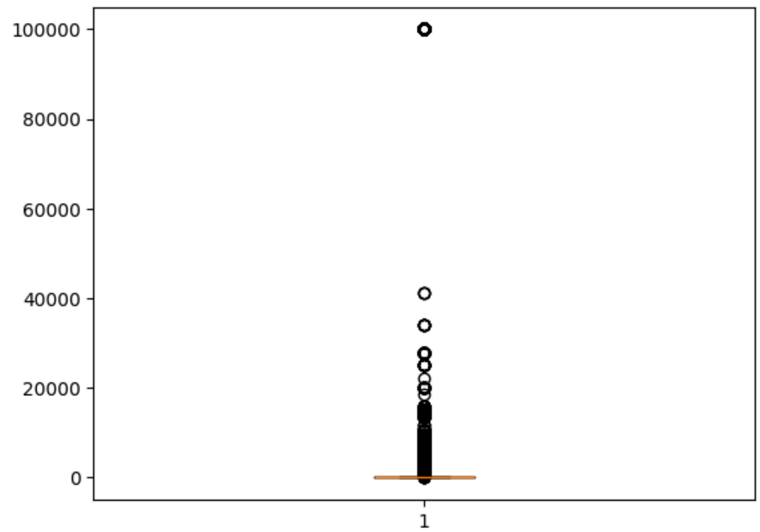
```
plt.boxplot(data['age'])
plt.show()
```



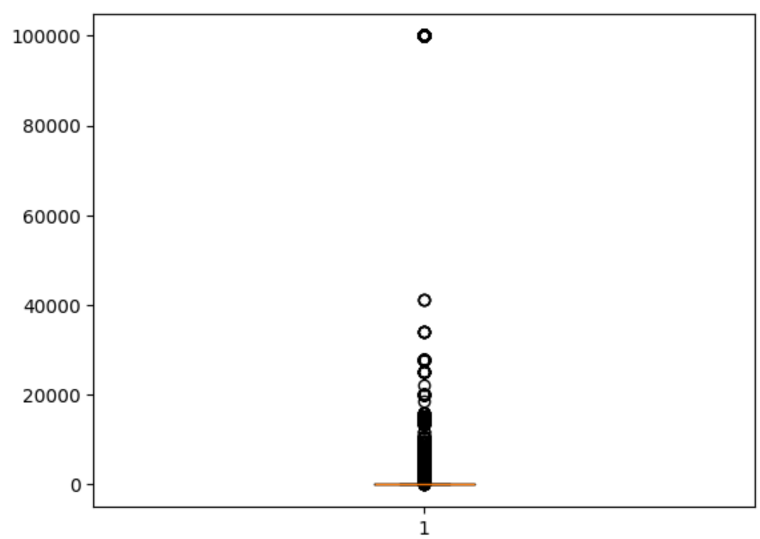

```
data.shape
```

```
(48438, 15)
```

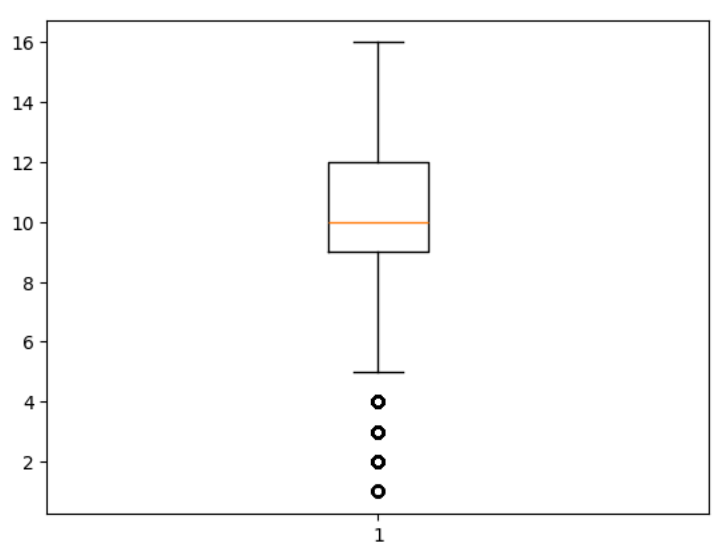
```
plt.boxplot(data['capital-gain'])
plt.show()
```




```
plt.boxplot(data['capital-gain'])
plt.show()
```

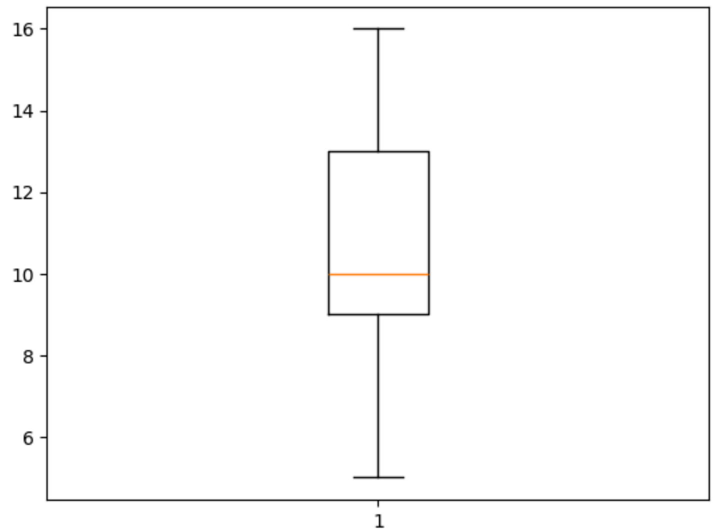



```
plt.boxplot(data['educational-num'])
plt.show()
```

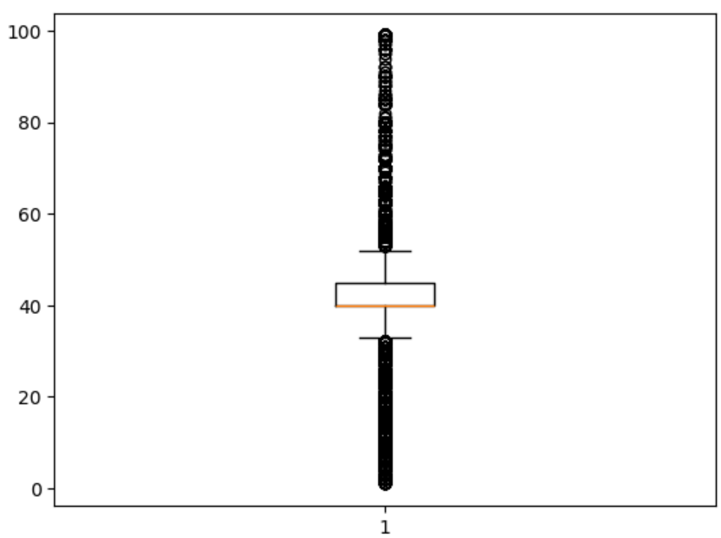


```
data=data[(data['educational-num']<=16)&(data['educational-num']>=5)]
```

```
plt.boxplot(data['educational-num'])
plt.show()
```




```
plt.boxplot(data['hours-per-week'])
plt.show()
```




```
data.shape
```

```
(46720, 15)
```

```
data=data.drop(columns=['education']) #redundant features removal
```

```
data
```

	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	incor
0	25	Private	226802	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=5(
1	38	Private	89814	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=5(
2	28	Local-gov	336951	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>5(
3	44	Private	160323	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>5(
4	18	Others	103497	10	Never-married	Others	Own-child	White	Female	0	0	30	United-States	<=5(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	

```
from sklearn.preprocessing import LabelEncoder   #import libarary
encoder=LabelEncoder()                          #create object
data['workclass']=encoder.fit_transform(data['workclass']) #7 categories   0,1, 2, 3, 4, 5, 6,
data['marital-status']=encoder.fit_transform(data['marital-status'])   #3 categories 0, 1, 2
data['occupation']=encoder.fit_transform(data['occupation'])
data['relationship']=encoder.fit_transform(data['relationship'])      #5 categories  0, 1, 2, 3, 4
data['race']=encoder.fit_transform(data['race'])
data['gender']=encoder.fit_transform(data['gender'])    #2 catogories     0, 1
data['native-country']=encoder.fit_transform(data['native-country'])
```

```
data
```

	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	incom
0	25	3	226802	7	4	6	3	2	1	0	0	40	39	<=50I
1	38	3	89814	9	2	4	0	4	1	0	0	50	39	<=50I
2	28	1	336951	12	2	11	0	4	1	0	0	40	39	>50I
3	44	3	160323	10	2	6	0	2	1	7688	0	40	39	>50I
4	18	2	103497	10	4	8	3	4	0	0	0	30	39	<=50I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
48837	27	3	257302	12	2	13	5	4	0	0	0	38	39	<=50I
48838	40	3	154374	9	2	6	0	4	1	0	0	40	39	>50I
48839	58	3	151910	9	6	0	4	4	0	0	0	40	39	<=50I
48840	22	3	201490	9	4	0	3	4	1	0	0	20	39	<=50I
48841	52	4	287927	9	2	3	5	4	0	15024	0	40	39	>50I

```
data['race'].value_counts()
```

```
race
4    39974
2     4500
1     1450
0      451
3      345
Name: count, dtype: int64
```

```
x=data.drop(columns=['income'])
y=data['income']
x
```

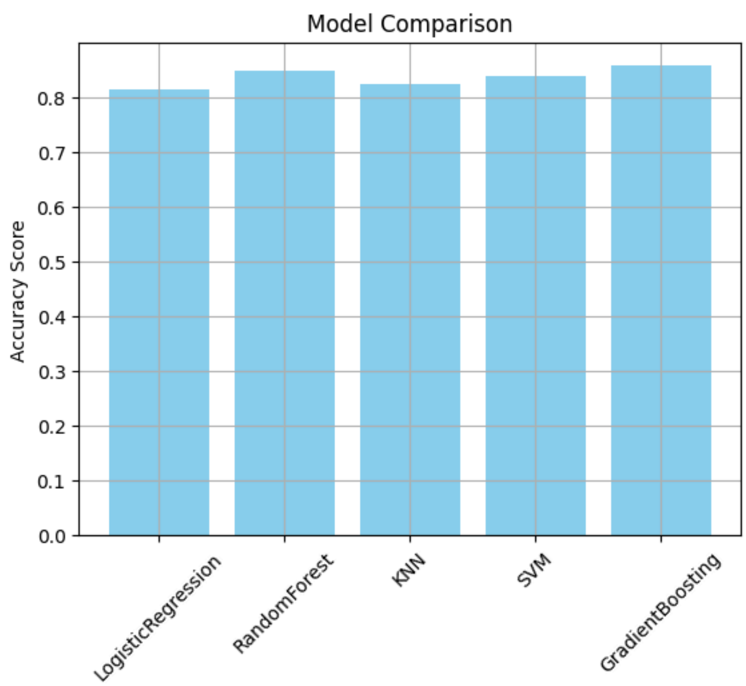
	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country
0	25	3	226802	7	4	6	3	2	1	0	0	40	39
1	38	3	89814	9	2	4	0	4	1	0	0	50	39
2	28	1	336951	12	2	11	0	4	1	0	0	40	39
3	44	3	160323	10	2	6	0	2	1	7688	0	40	39
4	18	2	103497	10	4	8	3	4	0	0	0	30	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	3	257302	12	2	13	5	4	0	0	0	38	39
48838	40	3	154374	9	2	6	0	4	1	0	0	40	39
48839	58	3	151910	9	6	0	4	4	0	0	0	40	39
48840	22	3	201490	9	4	0	3	4	1	0	0	20	39
48841	52	4	287927	9	2	3	5	4	0	15024	0	40	39

```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, OneHotEncoder

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

models = {
    "LogisticRegression": LogisticRegression(),
    "RandomForest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "GradientBoosting": GradientBoostingClassifier()
}

results = {}

for name, model in models.items():
    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('model', model)
    ])

    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
```

```
LogisticRegression Accuracy: 0.8149
              precision    recall  f1-score   support

       <=50K       0.84      0.93      0.88      7010
        >50K       0.69      0.46      0.55      2334

    accuracy                           0.81      9344
   macro avg       0.77      0.70      0.72      9344
weighted avg       0.80      0.81      0.80      9344

RandomForest Accuracy: 0.8488
              precision    recall  f1-score   support

       <=50K       0.88      0.93      0.90      7010
        >50K       0.74      0.61      0.67      2334

    accuracy                           0.85      9344
   macro avg       0.81      0.77      0.79      9344
weighted avg       0.84      0.85      0.84      9344
```

```
KNN Accuracy: 0.8245
              precision    recall  f1-score   support

       <=50K       0.87      0.90      0.88      7010
        >50K       0.67      0.60      0.63      2334

    accuracy                           0.82      9344
   macro avg       0.77      0.75      0.76      9344
weighted avg       0.82      0.82      0.82      9344

SVM Accuracy: 0.8396
              precision    recall  f1-score   support

       <=50K       0.86      0.94      0.90      7010
        >50K       0.75      0.54      0.63      2334

    accuracy                           0.84      9344
   macro avg       0.80      0.74      0.76      9344
weighted avg       0.83      0.84      0.83      9344

GradientBoosting Accuracy: 0.8571
              precision    recall  f1-score   support

       <=50K       0.88      0.94      0.91      7010
        >50K       0.78      0.60      0.68      2334

    accuracy                           0.86      9344
   macro avg       0.83      0.77      0.79      9344
weighted avg       0.85      0.86      0.85      9344
```

```
import matplotlib.pyplot as plt
plt.bar(results.keys(), results.values(), color='skyblue')
plt.ylabel('Accuracy Score')
plt.title('Model Comparison')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```




```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
# Define models
models = {
    "LogisticRegression": LogisticRegression(max_iter=1000),
    "RandomForest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "GradientBoosting": GradientBoostingClassifier()
}

results = {}

# Train and evaluate
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    results[name] = acc
    print(f"{name}: {acc:.4f}")

# Get best model
best_model_name = max(results, key=results.get)
best_model = models[best_model_name]
print(f"\n✅ Best model: {best_model_name} with accuracy {results[best_model_name]:.4f}")

# Save the best model
joblib.dump(best_model, "best_model.pkl")
print("✅ Saved best model as best_model.pkl")
```

```
C:\Users\afroz\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logistic.py:465: ConvergenceWarning: lbfg
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression: 0.7958
RandomForest: 0.8489
KNN: 0.7704
SVM: 0.7884
GradientBoosting: 0.8571

✅ Best model: GradientBoosting with accuracy 0.8571
✅ Saved best model as best_model.pkl
```

```
%%writefile app.py
import streamlit as st
import pandas as pd
import joblib

# Load the trained model
model = joblib.load("best_model.pkl")

st.set_page_config(page_title="Employee Salary Classification", page_icon="💼", layout="centered")

st.title("💼 Employee Salary Classification App")
st.markdown("Predict whether an employee earns >50K or ≤50K based on input features.")

# Sidebar inputs (these must match your training feature columns)
st.sidebar.header("Input Employee Details")

# ✨ Replace these fields with your dataset's actual input columns
age = st.sidebar.slider("Age", 18, 65, 30)
education = st.sidebar.selectbox("Education Level", [
    "Bachelors", "Masters", "PhD", "HS-grad", "Assoc", "Some-college"
])
occupation = st.sidebar.selectbox("Job Role", [
    "Tech-support", "Craft-repair", "Other-service", "Sales",
    "Exec-managerial", "Prof-specialty", "Handlers-cleaners", "Machine-op-inspct",
    "Adm-clerical", "Farming-fishing", "Transport-moving", "Priv-house-serv",
    "Protective-serv", "Armed-Forces"
])
hours_per_week = st.sidebar.slider("Hours per week", 1, 80, 40)
experience = st.sidebar.slider("Years of Experience", 0, 40, 5)
```

```
    # Build input DataFrame (⚠️ must match preprocessing of your training data)
    input_df = pd.DataFrame({
        'age': [age],
```