SALARY PREDICTION ENGINE USING PYTHON AND ML.ipynb

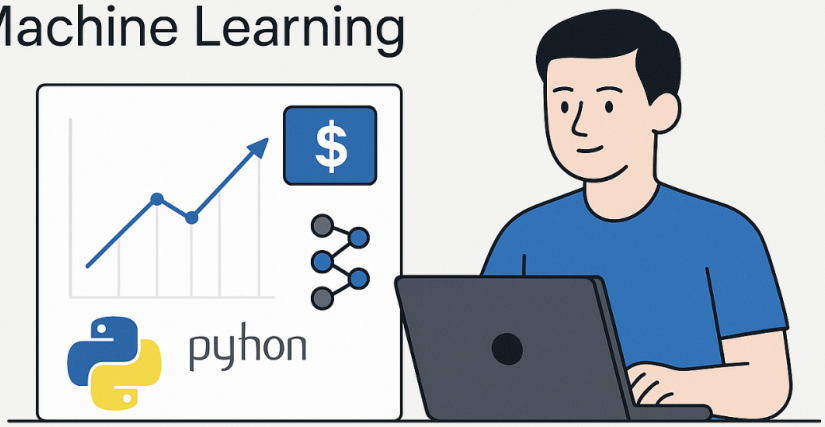
File Edit View Insert Runtime Tools Help

Share Gemini

Commands + Code + Text Run all

RAM Disk

Building a Salary Prediction Engine using Python and Machine Learning



Libraries and Modules required : ---

```
[47] # Libraries for Data Manipulation
     import pandas as pd
     import numpy as np

     # Libraries for Data Visualizations
     import matplotlib.pyplot as plt
     import seaborn as sns

     # Ignoring in-built warning
     import warnings
     warnings.filterwarnings('ignore')

     # Libraries for Pre-Processing
     from sklearn.preprocessing import LabelEncoder, StandardScaler

     # Libraries of data treatment before model building
     from sklearn.model_selection import train_test_split, RandomizedSearchCV

     # Libraries for model building
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.svm import SVC

     # Libaries for model evaluation
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

     # Saving and Loading files
     import pickle
```

Loading Dataset : ---

```
[48] data = pd.read_csv('adult 3.csv')
```

```
[49] data.sample(7)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
44714	42	Private	198341	Masters	14	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	0	1902	55	India	>50K
44920	36	Private	266461	HS-grad	9	Never-married	Transport-moving	Own-child	Black	Male	0	0	48	United-States	<=50K
29246	33	Private	192663	HS-grad	9	Separated	Sales	Not-in-family	White	Female	0	0	35	United-States	<=50K
42611	35	Self-emp-not-inc	160192	Some-college	10	Never-married	Craft-repair	Own-child	White	Male	0	0	40	United-States	<=50K
47915	29	Private	82393	HS-grad	9	Married-civ-spouse	Other-service	Own-child	Asian-Pac-Islander	Male	0	0	25	Philippines	<=50K
44354	30	Private	104223	HS-grad	9	Never-married	Other-service	Unmarried	Black	Female	0	0	32	United-States	<=50K
13268	49	Private	222829	Masters	14	Married-civ-	Prof-specialty	Husband	White	Male	0	0	42	United-States	>50K

spouse

Basic EDA Check : —

```
[50] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48842 non-null  int64 
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64 
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64 
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64 
 11  capital-loss     48842 non-null  int64 
 12  hours-per-week   48842 non-null  int64 
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
[51] data.isnull().sum()
```

	0
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

Duplicated Values

```
[52] data.duplicated().sum()
```

```
np.int64(52)
```

```
[53] # Removing Duplicated rows

data.drop_duplicates()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

48790 rows × 15 columns

Let's Check the Value Counts of all columns

```
[54] # As we can see some '?' values in various columns let's treat them

     categorical_columns = data.select_dtypes(include=object)

     for i in categorical_columns:
       print(f"\n**Column name - {i}**")
       print(data[i].value_counts())
```

```
**Column name - workclass**
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

**Column name - education**
education
HS-grad         15784
Some-college    10878
Bachelors        8025
Masters          2657
Assoc-voc        2061
11th             1812
Assoc-acdm       1601
10th             1389
7th-8th           955
Prof-school       834
9th               756
12th              657
Doctorate         594
5th-6th           509
1st-4th           247
Preschool          83
Name: count, dtype: int64

**Column name - marital-status**
marital-status
Married-civ-spouse       22379
Never-married            16117
Divorced                  6633
Separated                 1530
Widowed                   1518
Married-spouse-absent      628
Married-AF-spouse           37
Name: count, dtype: int64

**Column name - occupation**
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
```

Replacing the Irrelevant Values

```
[55] # workclass, occupation and native-country columns have the '?' missing values and let's replace it

     data['workclass'] = data['workclass'].replace('?','Not-Specified')
     data['occupation'] = data['occupation'].replace('?','Not-Defined')
     data['native-country'] = data['native-country'].replace('?','Others')
```

Creating a copy of Dataset

```
[56] data1 = data.copy()
     data1.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	Not-Specified	103497	Some-college	10	Never-married	Not-Defined	Own-child	White	Female	0	0	30	United-States	<=50K

Next steps: Generate code with data1 | View recommended plots | New interactive sheet

Feature Selection

```
[57] # this feature selection is by considering the real life parameters that affect the salary of an individual

     df = data1[['age','workclass','education','occupation','hours-per-week','native-country','income']]
     df.head()
```

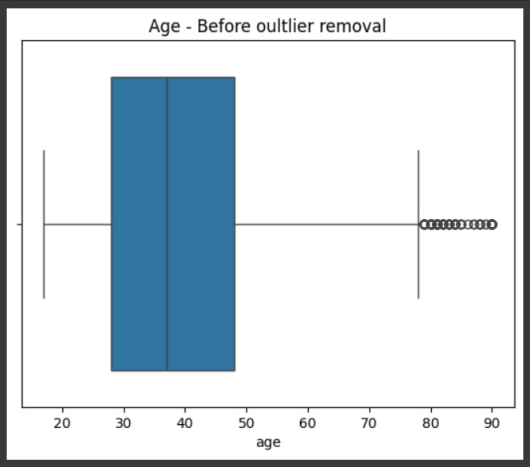
	age	workclass	education	occupation	hours-per-week	native-country	income
0	25	Private	11th	Machine-op-inspct	40	United-States	<=50K
1	38	Private	HS-grad	Farming-fishing	50	United-States	<=50K
2	28	Local-gov	Assoc-acdm	Protective-serv	40	United-States	>50K
3	44	Private	Some-college	Machine-op-inspct	40	United-States	>50K
4	18	Not-Specified	Some-college	Not-Defined	30	United-States	<=50K

Next steps: Generate code with df | View recommended plots | New interactive sheet

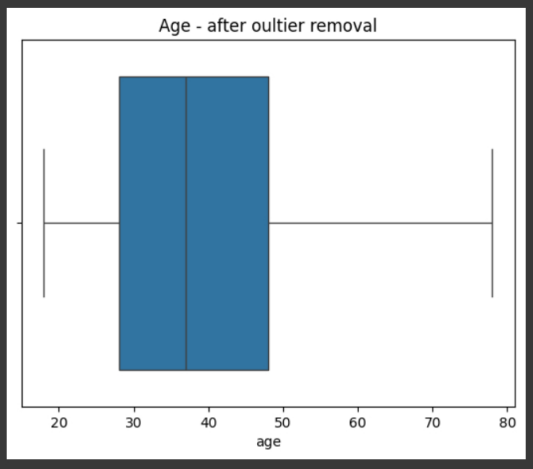
Outlier Analysis of the columns

age column

```
[58] # Let's see age column first

     sns.boxplot(data=df,x='age')
     plt.title("Age - Before oultlier removal")
     plt.show()
```

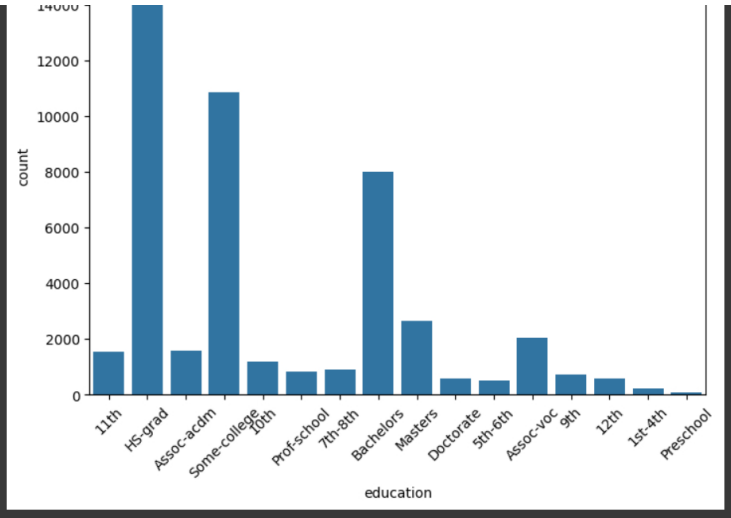



```
[59] # as the age column consist outlier new range is 18 to 78

     df = df[(df['age']>17)&(df['age']<79)]

     # let's visualize now
     sns.boxplot(data=df, x='age')
     plt.title("Age - after oultier removal")
     plt.show()
```



education column

```
[60] plt.figure(figsize=(8,6))
     sns.countplot(data=df,x='education')
     plt.xticks(rotation = 45)
     plt.show()
```




```
[61] # as some values from education column is irrelevant and share very less proportions
     # therefore removing them
     df = df[~df['education'].isin(['Preschool','1st-4th','7th-8th','5th-6th'])]
     df['education'].value_counts()
```

education	count
HS-grad	15701
Some-college	10848
Bachelors	8000
Masters	2644
Assoc-voc	2055
Assoc-acdm	1596
11th	1539
10th	1183
Prof-school	823
9th	718
Doctorate	590
12th	588

dtype: int64

occupation

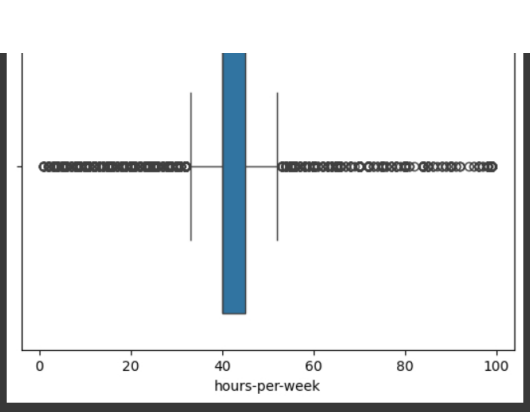
```
[62] df['occupation'].value_counts()
```

occupation	count
Prof-specialty	6111
Exec-managerial	6012
Craft-repair	5811
Adm-clerical	5519
Sales	5280
Other-service	4404
Machine-op-inspct	2744
Not-Defined	2494
Transport-moving	2207
Handlers-cleaners	1856
Tech-support	1437
Farming-fishing	1254
Protective-serv	965
Priv-house-serv	176
Armed-Forces	15

dtype: int64

hours-per-week

```
[63] sns.boxplot(data=df, x='hours-per-week')
     plt.show()
```



Splitting into Features and Target

```
[64] x = df.drop('income',axis=1)
     y = df.income

     print(f"Shape of features(x) : {x.shape}")
     print(f"Shape of target(y) : {y.shape}")
```

```
Shape of features(x) : (46285, 6)
Shape of target(y) : (46285,)
```

Label Encoding

```
[65] le = LabelEncoder()

     # categorical and numerical columns
     categorical_df = x.select_dtypes(include = object)
     numerical_df = x.select_dtypes(include = int)

     # encoding of categorical_columns using loop
     for i in categorical_df:
       x[i] = le.fit_transform(df[i])
       print(le.classes_)
```

```
['Federal-gov' 'Local-gov' 'Never-worked' 'Not-Specified' 'Private'
 'Self-emp-inc' 'Self-emp-not-inc' 'State-gov' 'Without-pay']
['10th' '11th' '12th' '9th' 'Assoc-acdm' 'Assoc-voc' 'Bachelors'
 'Doctorate' 'HS-grad' 'Masters' 'Prof-school' 'Some-college']
['Adm-clerical' 'Armed-Forces' 'Craft-repair' 'Exec-managerial'
 'Farming-fishing' 'Handlers-cleaners' 'Machine-op-inspct' 'Not-Defined'
 'Other-service' 'Priv-house-serv' 'Prof-specialty' 'Protective-serv'
 'Sales' 'Tech-support' 'Transport-moving']
['Cambodia' 'Canada' 'China' 'Columbia' 'Cuba' 'Dominican-Republic'
 'Ecuador' 'El-Salvador' 'England' 'France' 'Germany' 'Greece' 'Guatemala'
 'Haiti' 'Holand-Netherlands' 'Honduras' 'Hong' 'Hungary' 'India' 'Iran'
 'Ireland' 'Italy' 'Jamaica' 'Japan' 'Laos' 'Mexico' 'Nicaragua' 'Others'
 'Outlying-US(Guam-USVI-etc)' 'Peru' 'Philippines' 'Poland' 'Portugal'
 'Puerto-Rico' 'Scotland' 'South' 'Taiwan' 'Thailand' 'Trinadad&Tobago'
 'United-States' 'Vietnam' 'Yugoslavia']
```

Splitting into Training and Testing set

```
[66] x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.20, random_state=42, stratify=y)

     # shape
     print(f"Shape of x_train : {x_train.shape}")
     print(f"Shape of y_train : {y_train.shape}")
     print(f"Shape of x_test : {x_test.shape}")
     print(f"Shape of y_test : {y_test.shape}")
```

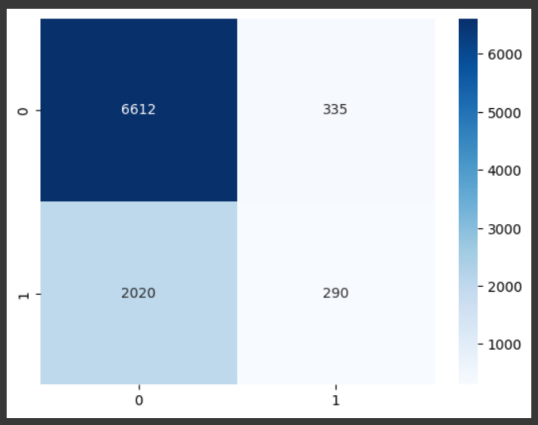
```
Shape of x_train : (37028, 6)
Shape of y_train : (37028,)
Shape of x_test : (9257, 6)
Shape of y_test : (9257,)
```

Trail - Model Building and Evaluation

```
[67] # using Logistic Regression
     LR = LogisticRegression()

     # Fitting the model into training data
     model = LR.fit(x_train,y_train)

     # predicting the labels using testing data
     y_pred = model.predict(x_test)


     # model evaluation
     accuracy = accuracy_score(y_test,y_pred)
     print(f"Accuracy of LogisticRegression model : {accuracy:.4f}\n")

     # Confusion Matrix
     print("Confusion matrix for Logistic Regression Model\n")
     cm = confusion_matrix(y_test, y_pred)
     sns.heatmap(data=cm, cmap='Blues', annot=True, fmt='2g')
     plt.show()
```

Accuracy of LogisticRegression model : 0.7456

Confusion matrix for Logistic Regression Model



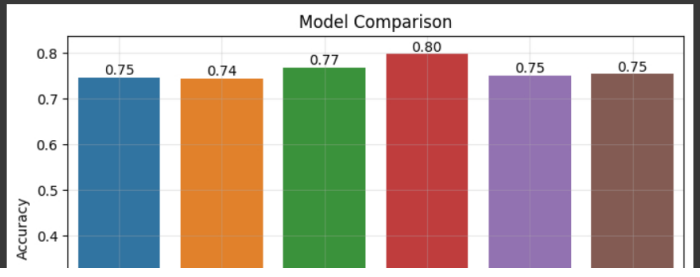
Trying Multimodels to Find best Fit Model

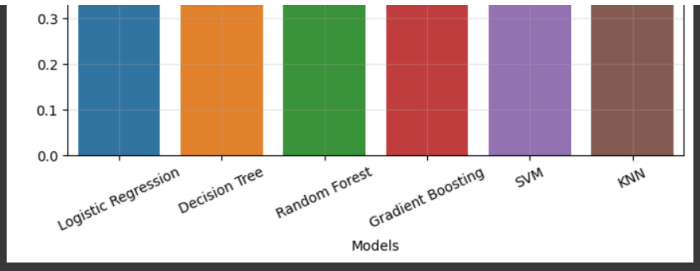
[68]
```
# Models : ---

models = {
    'Logistic Regression' : LogisticRegression(),
    'Decision Tree' : DecisionTreeClassifier(),
    'Random Forest' : RandomForestClassifier(),
    'Gradient Boosting' : GradientBoostingClassifier(),
    'SVM' : SVC(),
    'KNN' : KNeighborsClassifier()
    }

results = {}

# training and evalution of models
for name,model in models.items():
    test_model = model.fit(x_train,y_train)
    pred = test_model.predict(x_test)
    acc_score = accuracy_score(y_test, pred)
    results[name] = acc_score
    print(f"{name} Model : {acc_score:.2f}\n")

# finding best model among all
best_model_name = max(results, key=results.get)
best_model = models[best_model_name]
print(f"Best model -->> {best_model_name} with accuracy of {results[best_model_name]:.2f}")

# Saving the best model
with open('best_model.pkl','wb') as file:
    pickle.dump(best_model,file)
print("\nModel saved Successfully , saved file name : best_model.pkl")
```

```
Logistic Regression Model : 0.75

Decision Tree Model : 0.74

Random Forest Model : 0.77

Gradient Boosting Model : 0.80

SVM Model : 0.75

KNN Model : 0.75

Best model -->> Gradient Boosting with accuracy of 0.80

Model saved Successfully , saved file name : best_model.pkl
```

Let's Visualize the Accuracy of all models

[69]
```
plt.figure(figsize=(8,5))
ax = sns.barplot(data = results, x = results.keys(),y= results.values(),hue=results.keys())
for i in ax.containers:
  plt.bar_label(i,fmt='%.2f')
plt.xticks(rotation = 25)
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.title("Model Comparison")
plt.grid(alpha=0.30)
plt.show()
```





App Interface for Model Deployment

```
%%writefile app.py
import streamlit as st
import pickle
import pandas as pd
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

# loading saved model
with open('best_model.pkl','rb') as file:
 loaded_model = pickle.load(file)

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

st.title("👷 EMPLOYEES SALARY PREDICTION APP")
st.markdown("Develop a predictive model to classify whether an employee's salary exceeds $50,000 or not based on their input features.")

# sidebar inputs
st.sidebar.header("ENTER EMPLOYEE DETAILS")


age = st.sidebar.slider("AGE", 18,78,30)
workclass = st.sidebar.selectbox("WORKCLASS",['Federal-gov', 'Local-gov', 'Never-worked',
            'Not-Specified', 'Private','Self-emp-inc', 'Self-emp-not-inc', 'State-gov',
            'Without-pay'])
education = st.sidebar.selectbox("EDUCATION LEVEL", ['10th', '11th', '12th', '9th',
            'Assoc-acdm', 'Assoc-voc', 'Bachelors','Doctorate', 'HS-grad', 'Masters',
            'Prof-school', 'Some-college'])
occupation = st.sidebar.selectbox("JOB ROLE", ['Adm-clerical','Armed-Forces',
            'Craft-repair', 'Exec-managerial','Farming-fishing', 'Handlers-cleaners',
            'Machine-op-inspct', 'Not-Defined', 'Other-service', 'Priv-house-serv',
            'Prof-specialty', 'Protective-serv','Sales', 'Tech-support', 'Transport-moving'
            ])
hours_per_week = st.sidebar.slider("HOURS-PER-WEEK", 1,80,45)
native_country = st.sidebar.selectbox("NATIVE COUNTRY", ['Cambodia', 'Canada', 'China',
                 'Columbia', 'Cuba', 'Dominican-Republic',
                 'Ecuador', 'El-Salvador', 'England', 'France','Germany','Greece','Guatemala'
                 'Haiti','Holand-Netherlands','Honduras','Hong','Hungary','India','Iran',
                 'Ireland','Italy','Jamaica','Japan','Laos','Mexico','Nicaragua','Others',
                 'Outlying-US(Guam-USVI-etc)','Peru','Philippines','Poland','Portugal',
                 'Puerto-Rico','Scotland','South','Taiwan','Thailand','Trinadad&Tobago',
                 'United-States','Vietnam','Yugoslavia'])


# Build input DataFrame (⚠️ must match preprocessing of your training data)
input_df = pd.DataFrame({
    'age': [age],
    'workclass' : [workclass],
    'education': [education],
    'occupation': [occupation],
    'hours-per-week': [hours_per_week],
    'native-country' : [native_country]
})

st.write("### 🔎 INPUT DATA")
st.write(input_df)

# Predict button
if st.button("PREDICT SALARY CLASS"):
  for i in input_df.columns:
    if input_df[i].dtypes==object:
      input_df[i] = le.fit_transform(input_df[i])
  prediction = loaded_model.predict(input_df)
  st.success(f"✅ PREDICTION : {prediction}")

# Batch prediction
st.markdown("---")
st.markdown("## 📁 BATCH PREDICTION")
uploaded_file = st.file_uploader("Upload a CSV file for batch prediction", type="csv")

if uploaded_file is not None:
  batch_data = pd.read_csv(uploaded_file)
  st.write("Upload data preview :", batch_data.head())
  batch_pred = loaded_model.predict(batch_data)
  batch_data['Predicted Classes'] = batch_pred
  st.write("✅ Predictions:")
  st.write(batch_data.head())
  csv = batch_data.to_csv(index=False).encode('utf-8')
  st.download_button("Download Predictions CSV", csv, file_name='predicted_classes.csv', mime='text/csv')
```

```
Overwriting app.py
```

Using Python Wrapper and Auth Tokens by ngrok

```
[71] ! pip install streamlit pyngrok
```

```
Requirement already satisfied: streamlit in /usr/local/lib/python3.11/dist-packages (1.47.0)
Requirement already satisfied: pyngrok in /usr/local/lib/python3.11/dist-packages (7.2.12)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (1.9.0)
Requirement already satisfied: cachetools<7,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (8.2.1)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.0.2)
Requirement already satisfied: packaging<26,>=20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (25.0)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (11.2.1)
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.29.5)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (8.5.0)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (4.14.1)
Requirement already satisfied: watchdog<7,>=2.1.5 in /usr/local/lib/python3.11/dist-packages (from streamlit) (6.0.0)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from streamlit) (3.1.44)
Requirement already satisfied: pydeck<1,>=0.8.0b4 in /usr/local/lib/python3.11/dist-packages (from streamlit) (0.9.1)
Requirement already satisfied: tornado!=6.5.0,<7,>=6.0.3 in /usr/local/lib/python3.11/dist-packages (from streamlit) (6.4.2)
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.11/dist-packages (from pyngrok) (6.0.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (4.24.1)
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (1.47.1)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.12)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2025.7.14)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit) (5.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->altair<6,>=4.0->streamlit) (3.0.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (2025.4.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.26.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1.4.0->streamlit) (1.17.0)
```

```
[72] ! ngrok authtoken 2y3JiQbQ6peTqC6nsSTsFox8AS5_6St8TfKuJQTfi9JUQLRJi
```

```
Authtoken saved to configuration file: /root/.config/ngrok/ngrok.yml
```

Deployment of Model using Streamlit

```
[73] import os
     import threading

     def run_streamlit():
       os.system("streamlit run app.py -- server.port 8501")

     thread = threading.Thread(target = run_streamlit)
     thread.start()
```

App Run

```
from pyngrok import ngrok
import time

# wait for few seconds to make streamlit run
time.sleep(3)

# create a tunnel to streamlit port 8501
public_url = ngrok.connect(8501)
print("Your Streamlit App Server is Live;", public_url)
```

```
Your Streamlit App Server is Live; NgrokTunnel: "https://19d4c364e0b6.ngrok-free.app" -> "http://localhost:8501"
```

```
[75] # to kill the running services.

     # ngrok.kill()
```

```
[75] Start coding or generate with AI.
```