Career Path Prediction and Guidance System

CODE:

```
!pip install streamlit

!pip install pyngrok


# STEP 1

#Importing required libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model_selection import train_test_split


df = pd.read_csv("/content/PS2_Dataset.csv")


categorical_cols = df.select_dtypes(include='object').columns.tolist()

numerical_cols = df.select_dtypes(include='int64').columns.tolist()


#Encoding categorical columns

label_encoders = {}

df_encoded = df.copy()

for col in categorical_cols:

   le = LabelEncoder()

   df_encoded[col] = le.fit_transform(df_encoded[col])

   label_encoders[col] = le


# Prepare features and target

X = df_encoded.drop("Suggested Job Role", axis=1)

y = df_encoded["Suggested Job Role"]


# Train-test split

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

```
# Scaling numerical features
scaler = StandardScaler()
X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])

print("✅ Data loaded and preprocessed successfully.")
# STEP 2: Deep Learning Model with TensorFlow/Keras
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Get number of output classes
num_classes = len(np.unique(y))

# Build the model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')  # Multiclass output
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=30, batch_size=32, verbose=1)
```

```
# Evaluate on test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"\n✅ Test Accuracy: {test_accuracy:.4f}")

# STEP 3: Prediction + Recommendation Logic

# Inverse transform job role labels
job_role_encoder = label_encoders["Suggested Job Role"]

# Function to predict career path
def predict_career(input_data):
    """
    input_data: dict with keys matching X.columns
    returns: predicted job role and suggested learning path
    """
    # Convert to DataFrame
    input_df = pd.DataFrame([input_data])

    # Label encode all categorical fields using saved encoders
    for col in categorical_cols:
        if col != "Suggested Job Role":  # Avoid target
            le = label_encoders[col]
            # Convert single-element lists to strings
            input_df[col] = input_df[col].apply(lambda x: x[0] if isinstance(x, list) else x)
            # Handle unseen labels by assigning a default value or ignoring them
            try:
                input_df[col] = le.transform(input_df[col])
            except ValueError:
                # Option 1: Assign a default value (e.g., -1)
                input_df[col] = -1
                # Option 2: Ignore the column (remove it from input_df)
            # del input_df[col]
            # Option 3: If possible, retrain your model with more data
            # to include the unseen values in the encoder's vocabulary

    # Scale numerical columns
    input_df[numerical_cols] = scaler.transform(input_df[numerical_cols])

    # Predict
    pred = model.predict(input_df)
    pred_class = np.argmax(pred, axis=1)[0]
    job_role = job_role_encoder.inverse_transform([pred_class])[0]

    # ... (rest of the function remains the same)

    # Example Recommendation Logic (basic)
    recommendation = {
        "Data Scientist": ["Learn Python", "Master Pandas/Numpy", "Study ML algorithms", "Practice Kaggle problems"],
        "Software Developer": ["Build projects on GitHub", "Learn DSA", "Explore full-stack dev", "Internship"],
        "Web Developer": ["Learn HTML/CSS/JS", "React or Angular", "Deploy projects", "Freelance for experience"],
        "UX Designer": ["Study UI/UX design", "Work on Figma", "Take design thinking courses", "Build a portfolio"],
        # Add more if needed
    }

    action_plan = recommendation.get(job_role, ["Explore industry trends", "Build relevant skills", "Network", "Seek mentorship"])

    return job_role, action_plan

# EXAMPLE USAGE:
```

```
sample_input = {
    'Logical quotient rating': 7,
    'hackathons': 2,
    'coding skills rating': 7,
    'public speaking points': 6,
    'self-learning capability?': ['yes'],
    'Extra-courses did': ['yes'],
    'certifications': ['yes'],
    'workshops': ['yes'],
    'reading and writing skills': ['excellent'],
    'memory capability score': ['high'],
    'Interested subjects': ['Computer Architecture'],
    'interested career area ': ['Software Engineering'],
    'Type of company want to settle in?': ['Startup'],
    'Taken inputs from seniors or elders': ['yes'],
    'Interested Type of Books': ['Science & Technology'],
    'Management or Technical': ['Technical'],
    'hard/smart worker': ['smart worker'],
    'worked in teams ever?': ['yes'],
    'Introvert': ['no']
}

predicted_job, recommendations = predict_career(sample_input)
print("🎯 Predicted Job Role:", predicted_job)
print("📘 Suggested Learning Path:")
for step in recommendations:
    print("-", step)

model.save('career_model.h5')
import pickle
with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
from google.colab import files
files.download('career_model.h5')
files.download('label_encoders.pkl')
files.download('scaler.pkl')
app_code = '''
import streamlit as st
import numpy as np
import pandas as pd
import pickle
from tensorflow.keras.models import load_model

# Load saved model and encoders
model = load_model("career_model.h5")
with open("label_encoders.pkl", "rb") as f:
    label_encoders = pickle.load(f)
with open("scaler.pkl", "rb") as f:
    scaler = pickle.load(f)

categorical_cols = [
    'self-learning capability?', 'Extra-courses did', 'certifications',
    'workshops', 'reading and writing skills', 'memory capability score',
    'Interested subjects', 'interested career area ',
    'Type of company want to settle in?', 'Taken inputs from seniors or elders',
    'Interested Type of Books', 'Management or Technical',
    'hard/smart worker', 'worked in teams ever?', 'Introvert'
]
```

```
numerical_cols = ['Logical quotient rating', 'hackathons', 'coding skills rating', 'public speaking points']

job_role_encoder = label_encoders["Suggested Job Role"]

def predict_career(input_data):
    input_df = pd.DataFrame([input_data])
    for col in categorical_cols:
        le = label_encoders[col]
        input_df[col] = input_df[col].apply(lambda x: x[0] if isinstance(x, list) else x)
        try:
            input_df[col] = le.transform(input_df[col])
        except ValueError:
            input_df[col] = -1

    input_df[numerical_cols] = scaler.transform(input_df[numerical_cols])
    pred = model.predict(input_df)
    pred_class = np.argmax(pred, axis=1)[0]
    job_role = job_role_encoder.inverse_transform([pred_class])[0]

    recommendation = {
        "Data Scientist": ["Learn Python", "Master Pandas/Numpy", "Study ML algorithms", "Practice Kaggle problems"],
        "Software Developer": ["Build projects on GitHub", "Learn DSA", "Explore full-stack dev", "Internship"],
        "Web Developer": ["Learn HTML/CSS/JS", "React or Angular", "Deploy projects", "Freelance for experience"],
        "UX Designer": ["Study UI/UX design", "Work on Figma", "Take design thinking courses", "Build a portfolio"]
    }

    action_plan = recommendation.get(job_role, ["Explore industry trends", "Build relevant skills", "Network", "Seek mentorship"])
    return job_role, action_plan


# Streamlit UI

st.set_page_config(page_title="Career Path Predictor", page_icon="🎓", layout="centered",
initial_sidebar_state="collapsed")


st.markdown("<h1 style='text-align: center;'>🎓 Career Path Prediction and Guidance
System</h1>", unsafe_allow_html=True)

st.markdown("### Enter your information below:")


input_data = {}


# Categorical fields

for col in categorical_cols:

    unique_vals = label_encoders[col].classes_.tolist()

    input_data[col] = st.selectbox(col, unique_vals)


# Numerical fields

for col in numerical_cols:

    input_data[col] = st.slider(col, min_value=0, max_value=10, value=5)


if st.button("Predict Career Path"):

    job, plan = predict_career(input_data)

    st.success(f"🎯 Predicted Job Role: {job}")

    st.markdown("### 🧭 Suggested Learning Path:")

    for step in plan:

        st.markdown(f"- {step}")

'''


with open("app.py", "w") as f:

    f.write(app_code)
```

```
!ngrok config add-authtoken 2wtolL8Ga89HvQ3RDUhLYE2EEAE_4ZCz3NVSnBZp6Uw9KSZKC
from pyngrok import ngrok

# Kill existing tunnels
ngrok.kill()

# Start ngrok tunnel with protocol specified
public_url = ngrok.connect(8501, "http")
print("✅ Streamlit app is live at:", public_url)
!streamlit run app.py &>/content/logs.txt &
```