Code Editor Used Google Colab

#01 Calculation Of Graduation Year

```
import pandas as pd


#dataset loading
file_path = "/content/Final Lead Data.xlsx"
df = pd.read_excel(file_path)


#Checking initial data
print("Initial columns:", df.columns.tolist())
df.head()
#Parse the 'Created' column as defined format
df['Created'] = pd.to_datetime(df['Created'], format='%m/%d/%Y %I:%M:%S %p',
errors='coerce')
#Extracting admission year and calculating graduation year
df['Admission_Year'] = df['Created'].dt.year
df['Year_of_Graduation'] = df['Admission_Year'] + 4
#Preparing output DataFrame
output_df = df[['ID', 'First Name', 'Email', 'Year_of_Graduation']]
output_df = output_df.sort_values(by='ID')
#Saving to Excel
output_path = "/content/Clean-Data-yr-of-graduation.xlsx"
output_df.to_excel(output_path, index=False)
print(f"✅ Output saved to: {output_path}")
```

#02 Prediction Of Placement Status

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
import os

# Loading data
train_path = '/content/01 Train Data.xlsx'
test_path = '/content/02 Test Data.xlsx'

train_df = pd.read_excel(train_path)
test_df = pd.read_excel(test_path)

# Mapping target column
train_df["Placement Status"] = train_df["Placement Status"].map({"Placed": 1, "Not placed": 0})
target_column = "Placement Status"

# Dropping non-feature columns
drop_columns = [
    'First Name', 'Email ID', 'Attendee #', 'College Name',
    'Specify in "Others" (how did you come to know about this event)',
    target_column
]

X = train_df.drop(columns=drop_columns)
y = train_df[target_column]
X_test = test_df.drop(columns=drop_columns[:-1])  # test data, no target
```

```
# Getting all columns in training data
train_columns = X.columns

# Ensuring test data has the same columns as training data
# Adding missing columns to test data and fill with a default value (e.g., 0)
for col in train_columns:
    if col not in X_test.columns:
        X_test[col] = 0  # Filling with 0 or another appropriate default value

# Removing columns that are completely NaN in train
cols_all_nan = X.columns[X.isna().all()].tolist()
X = X.drop(columns=cols_all_nan)
#Ensuring X_test has the same columns as X after dropping NaN columns:
X_test = X_test.drop(columns=cols_all_nan, errors='ignore')  # Ignore if cols_all_nan not in X_test

# Impute
imputer = SimpleImputer(strategy="most_frequent")
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

X_test = X_test[X.columns]
X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

train_df = train_df.dropna(subset=[target_column])

# Redefining X and y after dropping NaN rows
X = train_df.drop(columns=drop_columns)
y = train_df[target_column]

#Removing columns with all NaN values from X and X_test after dropping NaN rows:
```

```
X = X.drop(columns=[col for col in X.columns if X[col].isnull().all()])

X_test = X_test.drop(columns=[col for col in X_test.columns if X_test[col].isnull().all()], errors='ignore')


# Ensuring X and X_test have the same columns again after potentially dropping more NaN columns:

X_test = X_test[X.columns]


# Encoding categorical variables, handling unseen labels

encoders = {}

for column in X.columns:

  if X[column].dtype == object or isinstance(X[column][0], str):

    le = LabelEncoder()

    # Fit on combined unique values from both train and test

    all_values = pd.concat([X[column], X_test[column]]).unique()

    le.fit(all_values.astype(str))


    X[column] = le.transform(X[column].astype(str))

    # Handle unseen labels in test data during transform

    X_test[column] = X_test[column].astype(str).map(lambda s: le.transform([s])[0] if s in le.classes_ else -1) #-1 represents unseen label


    encoders[column] = le


# Training model

model = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)

model.fit(X, y)


# Evaluating accuracy

cv_scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

print(f"Cross-validated accuracy: {cv_scores.mean():.4f}")
```

```
# Making predictions
predictions = model.predict(X_test)
predicted_labels = ["Yes" if pred == 1 else "No" for pred in predictions]

# Creating the directory
output_dir = os.path.dirname(output_path)
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Saving output
output_df.to_excel(output_path, index=False)

output_df = test_df[["First Name", "Email ID"]].copy()
output_df["Predicted Placement"] = predicted_labels
output_path = "/mnt/data/Predicted_Placement_Output.xlsx"
output_df.to_excel(output_path, index=False)

print(f"Predictions saved to: {output_path}")
```