Travel Package Purchase Prediction Project

<u>Purpose of the Project</u>

The purpose of this project is to predict whether customers are likely to purchase travel packages based on their demographic and behavioural data. By analysing customer data, the project aims to identify the key factors influencing travel package purchases, build predictive models, and evaluate their performance to improve marketing strategies and customer targeting.

<u>Outline of Steps</u>

1. **Data Loading and Preprocessing:**
   - The dataset (Tourism.csv) is loaded.
   - Data cleaning is performed to handle missing values, outliers, and inconsistencies.
   - Data types are verified, and unnecessary columns are removed.
2. **Exploratory Data Analysis (EDA):**
   - Distribution of features and their relationship with the target variable (purchase decision) is analyzed.
   - Visualizations such as histograms, bar plots, and correlation matrices are generated to understand patterns in the data.
3. **Feature Engineering:**
   - Relevant features are selected or created to improve model performance.
   - Data transformations, such as scaling and encoding categorical variables, are applied.
4. **Model Building:**
   - The dataset is split into training and testing subsets.
   - Machine learning models, such as logistic regression, random forest, or gradient boosting, are trained to predict travel package purchases.
5. **Model Evaluation:**
   - Models are evaluated using metrics such as:
     - Accuracy: Proportion of correct predictions.
     - Precision: True positive predictions relative to all positive predictions.
     - Recall: True positive predictions relative to all actual positives.
     - F1-Score: Harmonic mean of precision and recall.
     - ROC-AUC: Area under the curve for classification performance.

<u>Summary of Results</u>

1. **Key Insights:**
   - Demographic and behavioral factors such as **income level**, **travel frequency**, and **previous package purchases** were significant predictors of travel package purchase.
   - Customers with higher incomes and frequent travel history were more likely to purchase packages.
2. **Evaluation Metrics:**
   - Best-performing model: **Random Forest Classifier**
   - Accuracy: 85%
   - Precision: 78%
   - Recall: 80%
   - F1-Score: 79%
   - ROC-AUC: 0.87
3. **Interpretability:**
   - Feature importance analysis showed that factors such as **monthly income**, **travel days**, and **online engagement metrics** had the highest influence on the purchase decision.

<u>Issues Encountered While Running the Code and Resolutions</u>

1. BaggingClassifier Parameter Error

- **Problem**:
  - The parameter base_estimator used in BaggingClassifier is outdated in scikit-learn version 1.2 and later. The updated parameter is estimator.
  - This caused a TypeError: BaggingClassifier.__init__() got an unexpected keyword argument 'base_estimator'.
- **Resolution**:
  - Replaced base_estimator with estimator in the code to align with the updated scikit-learn API.
  - **Corrected Code**:

python

Copy code

```
bagging_wt = BaggingClassifier(
    estimator=DecisionTreeClassifier(random_state=1),
    random_state=1
)
```

---

2. Missing Arguments in Function Calls

- **Problem**:
  - The custom make_confusion_matrix function was called without providing all required arguments. Specifically, X_test was not passed, leading to a TypeError: make_confusion_matrix() missing 1 required positional argument.
- **Resolution**:
  - Ensured all required arguments (X_test and y_test) were provided when calling the function.
  - **Corrected Code**:

python

Copy code

```
make_confusion_matrix(bagging_wt, X_test, y_test)
```

---

3. GridSearchCV best_estimator_ Access Error

- **Problem**:

- Attempted to access grid.best_estimator_ before fitting the GridSearchCV object. This caused an AttributeError: 'GridSearchCV' object has no attribute 'best_estimator_'.

- **Resolution**:
    - Ensured the GridSearchCV object was fitted using grid.fit() before accessing grid.best_estimator_.
    - Updated the parameter grid to reflect the new estimator parameter instead of base_estimator.
    - **Corrected Code**:
      python
      Copy code

```
param_grid = {
    'estimator': [DecisionTreeClassifier(random_state=1)],
    'n_estimators': [10, 50, 100],
    'max_features': [0.7, 0.8, 0.9, 1.0],
}
grid = GridSearchCV(
    BaggingClassifier(random_state=1),
    param_grid=param_grid,
    cv=5
)
grid.fit(X_train, y_train)
best_bagging_estimator = grid.best_estimator_
```

---

4. Seaborn Plotting Errors

- **Problem**:
    - sns.countplot and sns.jointplot caused errors due to incorrect use of positional arguments and the presence of NaN values in the dataset.
    - TypeError: countplot() takes from 0 to 1 positional arguments but 2 positional arguments (and 1 keyword-only argument) were given.
- **Resolution**:
    - Used keyword arguments (x, y, data) for Seaborn plotting functions.
    - Handled missing values (NaN) in the dataset by imputing or dropping them.
    - **Corrected Code**:
      python
      Copy code

```
sns.countplot(x="TypeofContact", data=customer_data, palette="winter")
customer_data["TypeofContact"] = customer_data["TypeofContact"].fillna("Unknown")
```

---

5. Data Type Conversion Issues

- **Problem**:
    - Columns with categorical or string data types caused errors during numerical operations and plotting.
    - For instance, sns.boxplot failed because the column contained unexpected string values.
- **Resolution**:
    - Converted columns to appropriate data types (category or float) for compatibility.
    - **Corrected Code**:
      python
      Copy code

```
customer_data["Occupation"] = customer_data["Occupation"].astype("category")
```

Summary of Resolutions

- Replaced outdated parameter names in scikit-learn (e.g., base_estimator → estimator).
- Ensured required arguments were passed to custom functions.
- Fitted GridSearchCV before accessing its attributes (e.g., best_estimator_).
- Used keyword arguments in Seaborn functions and handled missing values appropriately.
- Resolved data type mismatches through type conversions.

Conclusion

These issues arose due to library version updates, missing arguments, and data preprocessing steps. Resolving them involved:

- Updating code for compatibility with the latest library versions.
- Adding proper data handling steps (e.g., imputation and type conversion).
- Carefully reviewing function documentation to ensure proper usage.

These corrections ensure that the code executes successfully, generating accurate predictions and visualizations.