znlu0wylq

November 26, 2024

[30]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
```

[31]:
```
# Load the datasets
train_data = pd.read_csv('carinsurance_train.csv')
test_data = pd.read_csv('carinsurance_test.csv')
```

[32]:
```
# Preview the data
print("Train Data:")
print(train_data.head())
print("\nTest Data:")
print(test_data.head())
```

```
Train Data:
   Id  Age          Job  Marital Education  Default  Balance  HHInsurance  \
0   1   32   management   single  tertiary        0     1218            1
1   2   32  blue-collar  married   primary        0     1156            1
2   3   29   management   single  tertiary        0      637            1
3   4   25      student   single   primary        0      373            1
4   5   30   management  married  tertiary        0     2694            0

   CarLoan Communication  LastContactDay LastContactMonth  NoOfContacts  \
0        0     telephone              28              jan             2
1        0           NaN              26              may             5
2        0      cellular               3              jun             1
3        0      cellular              11              may             2
4        0      cellular               3              jun             1

   DaysPassed  PrevAttempts  Outcome CallStart   CallEnd  CarInsurance
0          -1             0      NaN  13:45:20  13:46:30             0
1          -1             0      NaN  14:49:03  14:52:08             0
2         119             1  failure  16:30:24  16:36:04             1
3          -1             0      NaN  12:06:43  12:20:22             1
```

```
4          -1              0      NaN  14:35:44  14:38:56             0

Test Data:
     Id  Age         Job  Marital  Education  Default  Balance  HHInsurance  \
0  4001   25      admin.   single  secondary        0        1            1
1  4002   40  management  married   tertiary        0        0            1
2  4003   44  management   single   tertiary        0    -1313            1
3  4004   27    services   single  secondary        0     6279            1
4  4005   53  technician  married  secondary        0     7984            1

   CarLoan Communication  LastContactDay LastContactMonth  NoOfContacts  \
0        1           NaN              12              may            12
1        1      cellular              24              jul             1
2        1      cellular              15              may            10
3        0      cellular               9              nov             1
4        0      cellular               2              feb             1

   DaysPassed  PrevAttempts Outcome CallStart   CallEnd  CarInsurance
0          -1             0     NaN  17:17:42  17:18:06           NaN
1          -1             0     NaN  09:13:44  09:14:37           NaN
2          -1             0     NaN  15:24:07  15:25:51           NaN
3          -1             0     NaN  09:43:44  09:48:01           NaN
4          -1             0     NaN  16:31:51  16:34:22           NaN
```

[33]:
```
# Drop unnecessary columns (e.g., Id, CallStart, CallEnd)
columns_to_drop = ['Id', 'CallStart', 'CallEnd']
train_data = train_data.drop(columns=columns_to_drop, errors='ignore')
test_data = test_data.drop(columns=columns_to_drop, errors='ignore')
```

[34]:
```
# Handle missing values
train_data.fillna(train_data.median(numeric_only=True), inplace=True)
test_data.fillna(test_data.median(numeric_only=True), inplace=True)
```

[35]:
```
# Encode categorical variables using LabelEncoder
categorical_columns = ['Job', 'Marital', 'Education', 'Communication',␣
 ↪'Outcome', 'LastContactMonth']

label_encoder = LabelEncoder()
for col in categorical_columns:
    if col in train_data.columns:
        train_data[col] = label_encoder.fit_transform(train_data[col].
 ↪astype(str))
    if col in test_data.columns:
        test_data[col] = label_encoder.transform(test_data[col].astype(str))
```

[36]:
```
# Split train_data into features and target
X = train_data.drop('CarInsurance', axis=1)
```

```
y = train_data['CarInsurance']

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

[37]:
```
# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

[38]:
```
# Prepare the test data (ensure it has the same features as the training data)
X_test = test_data.drop(columns=['CarInsurance'], errors='ignore')  # Drop␣
 ↪target column from test data
X_test = scaler.transform(X_test)  # Standardize the test data
```

[39]:
```
# Build and train the Logistic Regression model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if convergence␣
 ↪warning occurs
model.fit(X_train, y_train)
```

[39]:
```
LogisticRegression(max_iter=1000)
```

[40]:
```
# Predictions
y_val_pred = model.predict(X_val)
test_predictions = model.predict(X_test)
```

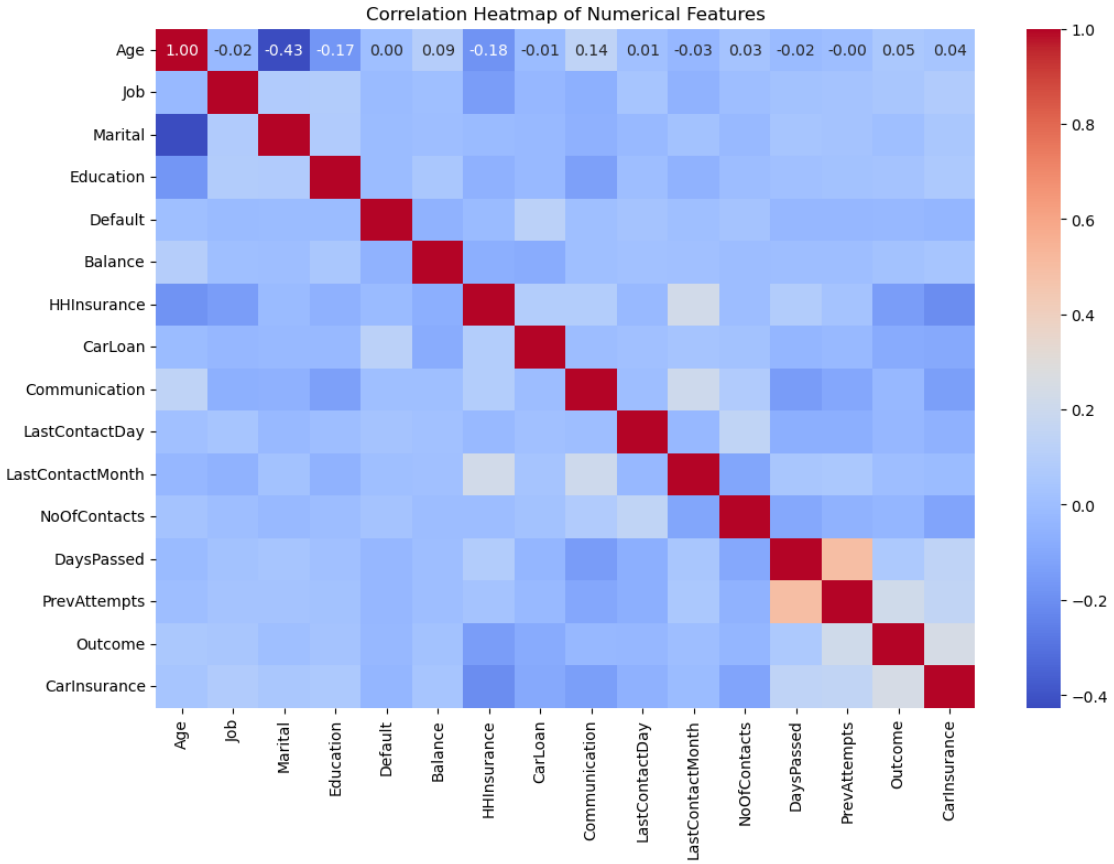
[41]:
```
# Evaluate the model
print("Validation Set Metrics:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))
```

```
Validation Set Metrics:
Accuracy: 0.675
Confusion Matrix:
 [[406  73]
 [187 134]]
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.85      0.76       479
           1       0.65      0.42      0.51       321

    accuracy                           0.68       800
   macro avg       0.67      0.63      0.63       800
weighted avg       0.67      0.68      0.66       800
```

[42]:
```
# Save the test predictions
test_data['CarInsurance'] = test_predictions
test_data.to_csv('carinsurance_predictions.csv', index=False)

print("Predictions saved to 'carinsurance_predictions.csv'")
```

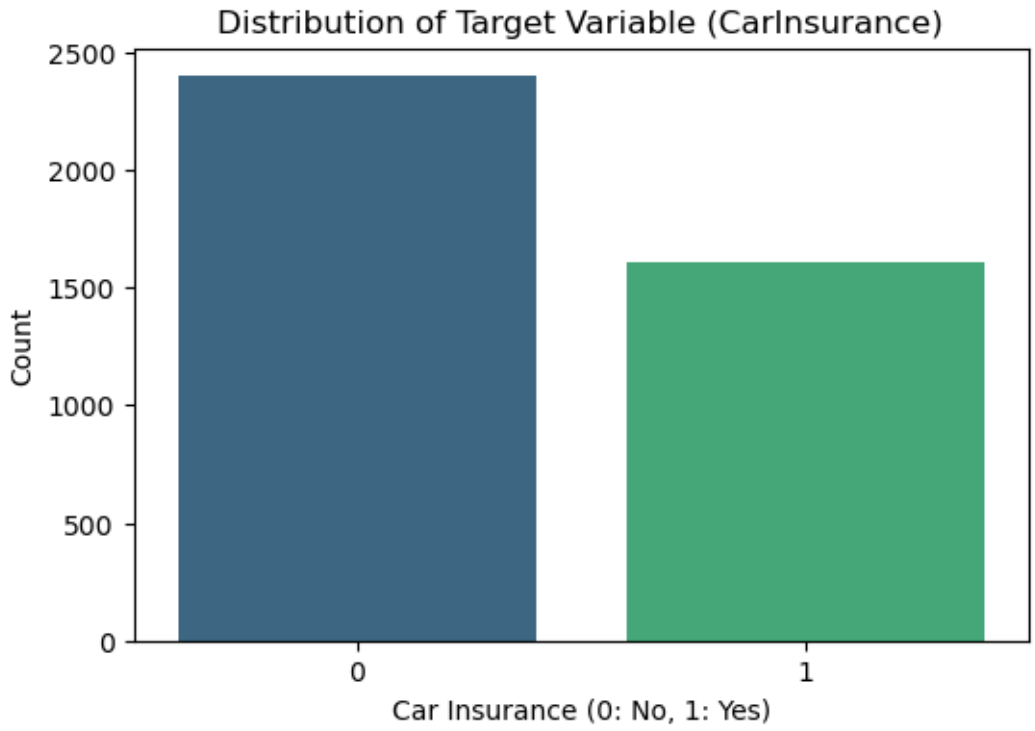
```
Predictions saved to 'carinsurance_predictions.csv'
```

[43]:
```
# Correlation heatmap for numerical features
plt.figure(figsize=(12, 8))
sns.heatmap(train_data.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



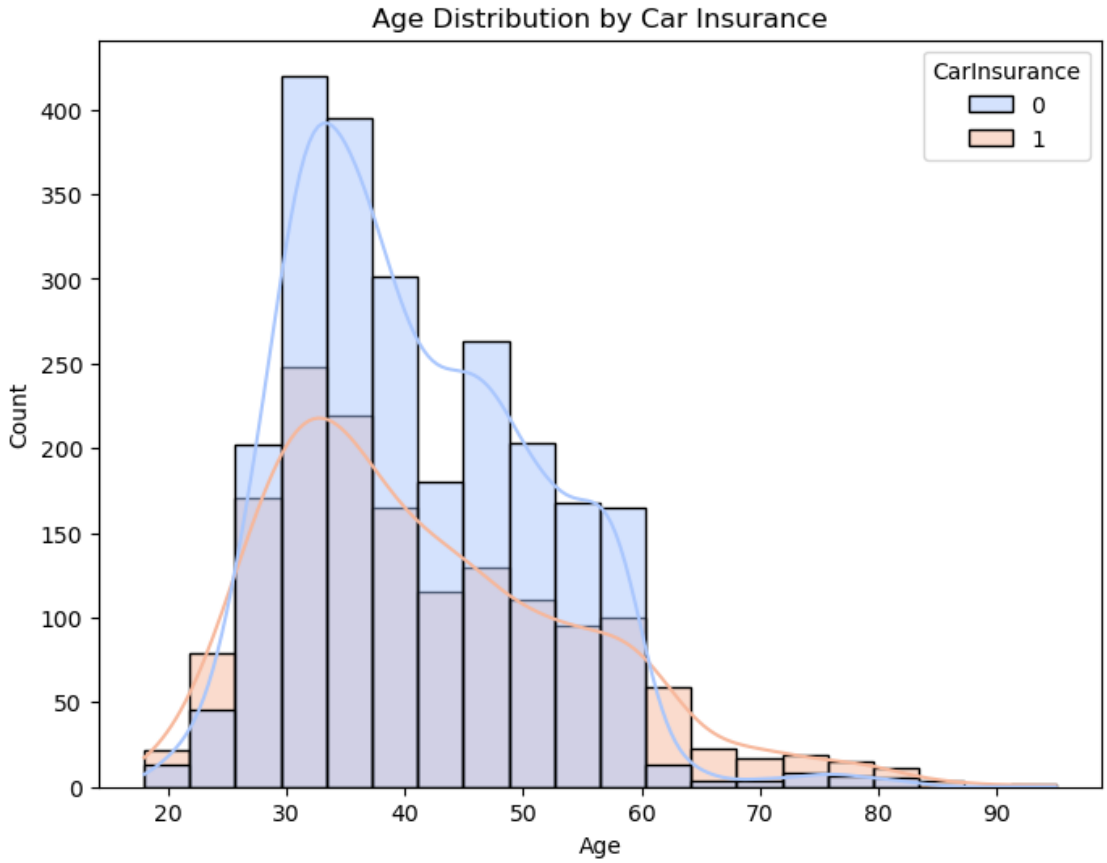
[44]:
```
# Distribution of target variable in training data
plt.figure(figsize=(6, 4))
sns.countplot(x='CarInsurance', data=train_data, palette='viridis')
plt.title("Distribution of Target Variable (CarInsurance)")
```

```
plt.xlabel("Car Insurance (0: No, 1: Yes)")
plt.ylabel("Count")
plt.show()
```



[45]:
```
# Age distribution by CarInsurance in training data
plt.figure(figsize=(8, 6))
sns.histplot(data=train_data, x="Age", hue="CarInsurance", kde=True,␣
 ↪palette="coolwarm", bins=20)
plt.title("Age Distribution by Car Insurance")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

```
D:\python\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

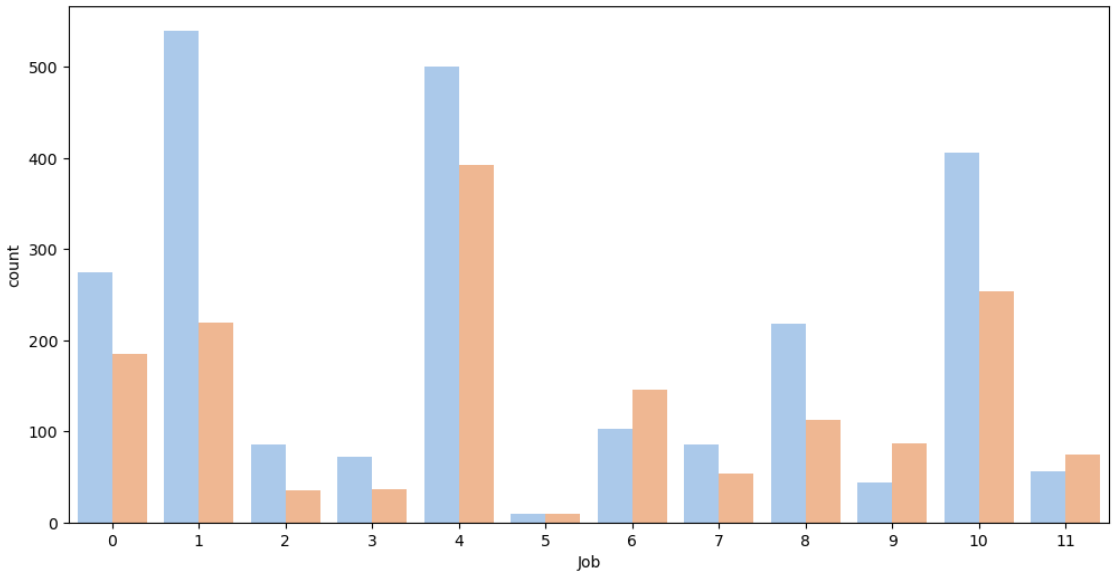


[46]:
```
# Job category vs CarInsurance in training data
plt.figure(figsize=(12, 6))
sns.countplot(data=train_data, x='Job', hue='CarInsurance', palette='pastel')
plt.title("Job Category vs Car Insurance")
plt.xlabel("Job")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

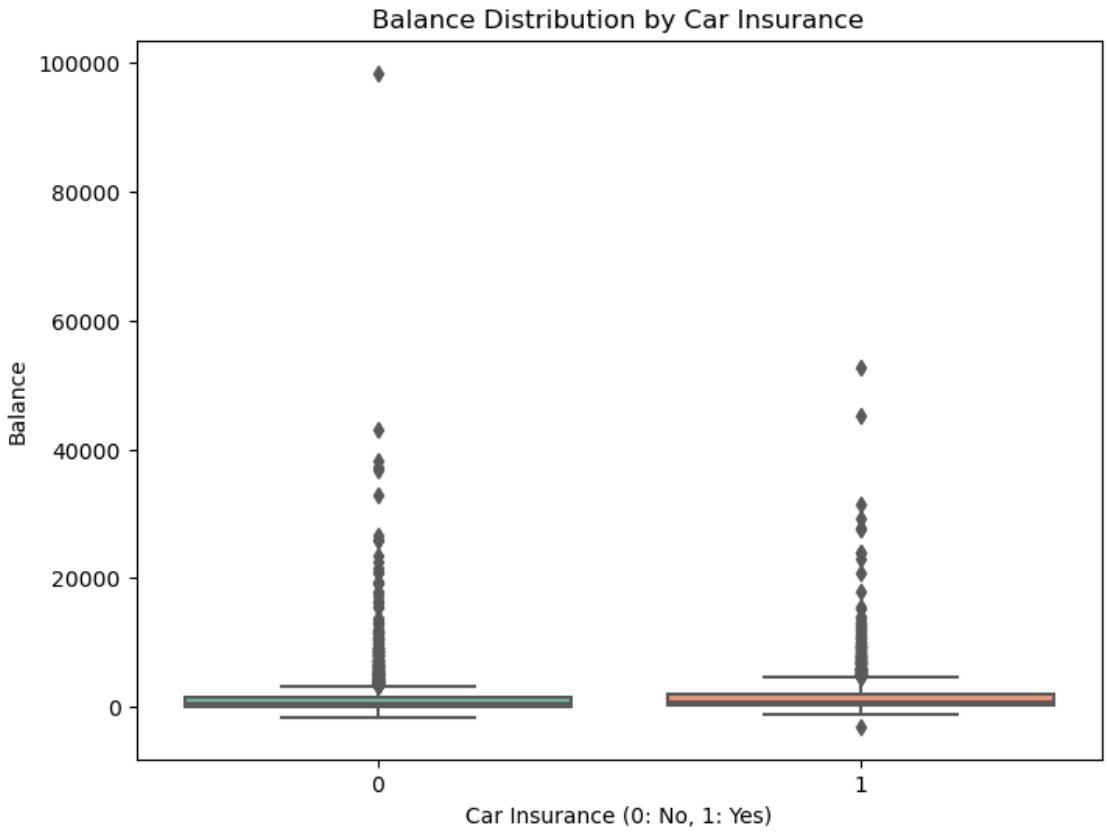
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[46], line 3
      1 # Job category vs CarInsurance in training data
      2 plt.figure(figsize=(12, 6))
----> 3 sns.countplot(data=train_data, x='Job', hue='CarInsurance',␣
 ↪palette='pastel')
      4 plt.title("Job Category vs Car Insurance")
      5 plt.xlabel("Job")
```

```
File D:\python\Lib\site-packages\seaborn\categorical.py:2955, in countplot(data
 ↪x, y, hue, order, hue_order, orient, color, palette, saturation, width, dodge
 ↪ax, **kwargs)
   2952 if ax is None:
   2953     ax = plt.gca()
-> 2955 plotter.plot(ax, kwargs)
   2956 return ax

File D:\python\Lib\site-packages\seaborn\categorical.py:1587, in _BarPlotter.
 ↪plot(self, ax, bar_kws)
   1585 """Make the plot."""
   1586 self.draw_bars(ax, bar_kws)
-> 1587 self.annotate_axes(ax)
   1588 if self.orient == "h":
   1589     ax.invert_yaxis()

File D:\python\Lib\site-packages\seaborn\categorical.py:767, in
 ↪_CategoricalPlotter.annotate_axes(self, ax)
    764     ax.set_ylim(-.5, len(self.plot_data) - .5, auto=None)
    766 if self.hue_names is not None:
--> 767     ax.legend(loc="best", title=self.hue_title)

File D:\python\Lib\site-packages\matplotlib\axes\_axes.py:322, in Axes.
 ↪legend(self, *args, **kwargs)
    204 @_docstring.dedent_interpd
    205 def legend(self, *args, **kwargs):
    206     """
    207     Place a legend on the Axes.
    208
   (…)
    320     .. plot:: gallery/text_labels_and_annotations/legend.py
    321     """
--> 322     handles, labels, kwargs = mlegend._parse_legend_args([self], *args,
 ↪**kwargs)
    323     self.legend_ = mlegend.Legend(self, handles, labels, **kwargs)
    324     self.legend_._remove_method = self._remove_legend

File D:\python\Lib\site-packages\matplotlib\legend.py:1361, in
 ↪_parse_legend_args(axs, handles, labels, *args, **kwargs)
   1357     handles = [handle for handle, label
   1358                in zip(_get_legend_handles(axs, handlers), labels)]
   1360 elif len(args) == 0:  # 0 args: automatically detect labels and handles
-> 1361     handles, labels = _get_legend_handles_labels(axs, handlers)
   1362     if not handles:
   1363         log.warning(
   1364             "No artists with labels found to put in legend.  Note that
   1365             "artists whose label start with an underscore are ignored "
   1366             "when legend() is called with no argument.")
```

```
File D:\python\Lib\site-packages\matplotlib\legend.py:1291, in␣
 ↪_get_legend_handles_labels(axs, legend_handler_map)
   1289 for handle in _get_legend_handles(axs, legend_handler_map):
   1290     label = handle.get_label()
-> 1291     if label and not label.startswith('_'):
   1292         handles.append(handle)
   1293         labels.append(label)

AttributeError: 'numpy.int64' object has no attribute 'startswith'
```



[47]:
```
# Balance distribution by CarInsurance in training data
plt.figure(figsize=(8, 6))
sns.boxplot(data=train_data, x='CarInsurance', y='Balance', palette="Set2")
plt.title("Balance Distribution by Car Insurance")
plt.xlabel("Car Insurance (0: No, 1: Yes)")
plt.ylabel("Balance")
plt.show()
```

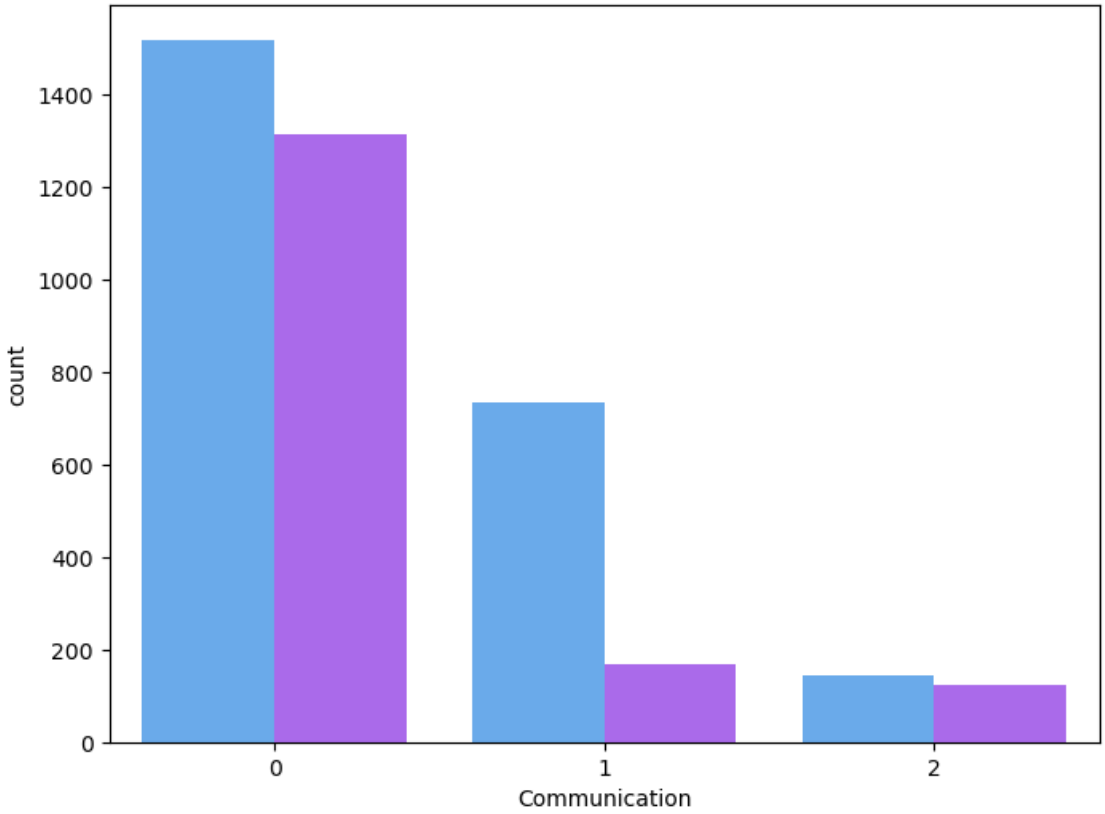


[48]:
```
# Contact method distribution in training data
plt.figure(figsize=(8, 6))
sns.countplot(data=train_data, x='Communication', hue='CarInsurance',␣
 ↪palette='cool')
plt.title("Contact Method vs Car Insurance")
plt.xlabel("Communication Method")
plt.ylabel("Count")
plt.show()
```

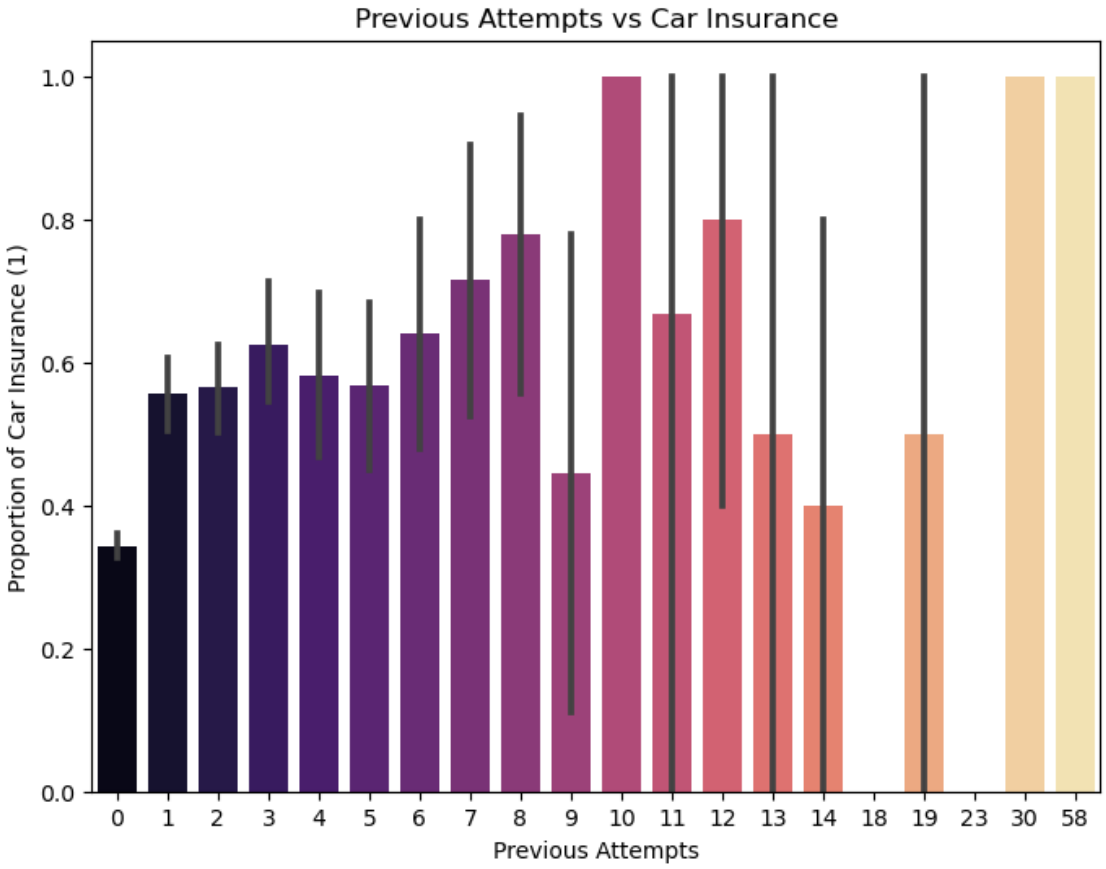
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[48], line 3
      1 # Contact method distribution in training data
      2 plt.figure(figsize=(8, 6))
----> 3 sns.countplot(data=train_data, x='Communication', hue='CarInsurance',␣
 ↪palette='cool')
      4 plt.title("Contact Method vs Car Insurance")
      5 plt.xlabel("Communication Method")
```

```
File D:\python\Lib\site-packages\seaborn\categorical.py:2955, in countplot(data
 ↪x, y, hue, order, hue_order, orient, color, palette, saturation, width, dodge
 ↪ax, **kwargs)
   2952 if ax is None:
   2953     ax = plt.gca()
-> 2955 plotter.plot(ax, kwargs)
   2956 return ax

File D:\python\Lib\site-packages\seaborn\categorical.py:1587, in _BarPlotter.
 ↪plot(self, ax, bar_kws)
   1585 """Make the plot."""
   1586 self.draw_bars(ax, bar_kws)
-> 1587 self.annotate_axes(ax)
   1588 if self.orient == "h":
   1589     ax.invert_yaxis()

File D:\python\Lib\site-packages\seaborn\categorical.py:767, in
 ↪_CategoricalPlotter.annotate_axes(self, ax)
    764     ax.set_ylim(-.5, len(self.plot_data) - .5, auto=None)
    766 if self.hue_names is not None:
--> 767     ax.legend(loc="best", title=self.hue_title)

File D:\python\Lib\site-packages\matplotlib\axes\_axes.py:322, in Axes.
 ↪legend(self, *args, **kwargs)
    204 @_docstring.dedent_interpd
    205 def legend(self, *args, **kwargs):
    206     """
    207     Place a legend on the Axes.
    208
   (…)
    320     .. plot:: gallery/text_labels_and_annotations/legend.py
    321     """
--> 322     handles, labels, kwargs = mlegend._parse_legend_args([self], *args,
 ↪**kwargs)
    323     self.legend_ = mlegend.Legend(self, handles, labels, **kwargs)
    324     self.legend_._remove_method = self._remove_legend

File D:\python\Lib\site-packages\matplotlib\legend.py:1361, in
 ↪_parse_legend_args(axs, handles, labels, *args, **kwargs)
   1357     handles = [handle for handle, label
   1358                in zip(_get_legend_handles(axs, handlers), labels)]
   1360 elif len(args) == 0:  # 0 args: automatically detect labels and handles
-> 1361     handles, labels = _get_legend_handles_labels(axs, handlers)
   1362     if not handles:
   1363         log.warning(
   1364             "No artists with labels found to put in legend.  Note that
   1365             "artists whose label start with an underscore are ignored "
   1366             "when legend() is called with no argument.")
```

```
File D:\python\Lib\site-packages\matplotlib\legend.py:1291, in␣
 ↪_get_legend_handles_labels(axs, legend_handler_map)
   1289 for handle in _get_legend_handles(axs, legend_handler_map):
   1290     label = handle.get_label()
-> 1291     if label and not label.startswith('_'):
   1292         handles.append(handle)
   1293         labels.append(label)

AttributeError: 'numpy.int64' object has no attribute 'startswith'
```



[49]:
```
# Previous attempts vs CarInsurance in training data
plt.figure(figsize=(8, 6))
sns.barplot(data=train_data, x='PrevAttempts', y='CarInsurance',␣
 ↪palette='magma')
plt.title("Previous Attempts vs Car Insurance")
plt.xlabel("Previous Attempts")
plt.ylabel("Proportion of Car Insurance (1)")
plt.show()
```


Previous Attempts vs Car Insurance
Proportion of Car Insurance (1)
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 18 19 23 30 58
Previous Attempts

[ ]: