UGL MINI PROJECT

Online Shoppers Intention Prediction Dataset - Based on given class of online browsing for online shopping by(of different clusters in time whether customer will buy or not) Dataset consists of 10 numerical and 8 categorical attributes. 'Revenue' attribute can be used as the class label. The dataset consists of 12,330 sessions. The dataset was formed so that each session would belong to different user in a period of 1 year to avoid any tendency to a specific campaign, special day, user profile or period. Informative Features: Administrative, Administrative_Duration, Informational, Informational_Duration, ProductRelated, ProductRelated_Duration, BounceRates, ExitRates, PageValues, SpecialDay, Month, OperatingSystems, Browser, Region, TrafficType, VisitorType, Weekend. The "Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product Related Duration" represent the number of different types of pages visited by the visitor in that session and total time spent in each of these page categories. The values of these features are derived from the URL information of the pages visited by the user and updated in real time when a user takes an action, e.g. moving from one page to another. The "Bounce Rate", "Exit Rate" and "Page Value" features represent the metrics measured by "Google Analytics" for each page in this e-commerce site. The value of "Bounce Rate" feature for a web page refers to the percentage of visitors who enter the site from that page and then "bounce" out of the site without triggering any other requests to the server during that session. The value of "Exit Rate" feature for a specific web page is calculated as the percentage of all pageviews to the page that were the last in the session. The "Page Value" feature represents the average value for a web page that a user visited before completing an e-commerce transaction. The "Special Day" feature indicates the closeness of the site visiting session to a special day (e.g. Mother's Day, Valentine's Day) in which the sessions are more likely to be finalized with transaction. The value of this attribute is determined by the dynamics of e-commerce such as the duration between ordering and delivery for a certain period prior to a special day and activity such as eg. Mother's Day is given a value of 1 for 2 days before and after Mother's Day, 0 for all other days. The "Month" attribute represents the month of the year in which the session was carried out. The "Operating Systems", "Browser", "Region", "Traffic Type", "Visitor Type" and "Weekend" are other characteristics that describe a session. The "Visitor Type" attribute takes three values: "New Visitor", "Returning Visitor" and "Other". The "Weekend" attribute is a binary (0 or 1) variable indicating whether the session was on a weekend. The "Revenue" attribute is the class label. The dataset was created by collecting data over a period of one year. The authors have shown that this dataset is suitable for binary classification tasks. Since the "Revenue" attribute is a binary variable, it is a typical classification task. The dataset contains 10 numerical and 8 categorical attributes. The "Revenue" attribute can be used as the class label. "Administrative", "Informational", "ProductRelated" and "OperatingSystems" attributes have missing values. There are no duplicate records. The dataset is imbalanced, as only 15% of the records belong to the class 1 (purchase) and the remaining 85% belong to the class 0 (no purchase). The class of this feature can be predicted using the other 17 features. Since the class is a binary variable, it is a typical classification task

```
# Product Related Duration vs Revenue
sns.boxplot(data=df, x='ProductRelated_Duration', y='Revenue')
plt.title('Product Related Duration vs Revenue')
plt.show()
```

```
# Administrative vs Revenue
sns.boxplot(data=df, x='Administrative', y='Revenue')
plt.title('Administrative vs Revenue')
plt.show()
```

```
df.Administrative.value_counts()
```

```
Administrative
0     3530
1     1114
2      856
3      592
4      421
5      326
6      239
7      181
8      135
9      114
10      87
11      75
12      57
13      51
14      41
15      31
16      26
17      18
18      14
19      13
20      10
21       8
22       7
23       6
24       5
25       4
26       3
27       2
Name: Administrative, dtype: int64
```

```
df[['Region']].value_counts()
```

```
Region
1     4780
3     2403
4     1182
2     1136
6      948
5      761
7      730
8      601
9      507
Name: count, dtype: int64
```

```
# plot customer frequency by region
plt.figure(figure_size=(8,6))
sns.countplot(data=df, x='Region')
plt.title('Distribution of users(Customers)')
plt.xlabel('Region Codes')
plt.ylabel('Count')
plt.show()
```

```
# bar plot for traffic type and revenue
plt.figure(figure_size=(8,6))
sns.countplot(data=df, x='TrafficType', hue='Revenue')
plt.title('Traffic Type vs Revenue')
plt.xlabel('Traffic Type')
plt.ylabel('Count')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

Multivariate Analysis

```
# month vs pageviews w.r.t. revenue
plt.figure(figure_size=(8,6))
sns.violinplot(data=df, x='Month', y='PageValues', hue='Revenue', palette='Paired')
plt.title('month vs pageviews w.r.t. revenue')
plt.xlabel('Month')
plt.ylabel('PageValues')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

```
# month vs bounceRates w.r.t. Revenue
plt.figure(figure_size=(8,6))
sns.violinplot(data=df, x='Month', y='BounceRates', hue='Revenue', palette='Paired')
plt.title('month vs bounceRates w.r.t. Revenue')
plt.xlabel('Month')
plt.ylabel('BounceRates')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

```
# visitor type vs bounceRates w.r.t. revenue
plt.figure(figure_size=(8,6))
sns.violinplot(data=df, x='VisitorType', y='BounceRates', hue='Revenue', palette='Paired')
plt.title('visitor type vs bounceRates w.r.t. revenue')
plt.xlabel('VisitorType')
plt.ylabel('BounceRates')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

```
# visitor type vs exit rates w.r.t. revenue
plt.figure(figure_size=(8,6))
sns.violinplot(data=df, x='VisitorType', y='ExitRates', hue='Revenue', palette='Paired')
plt.title('visitor type vs exit rates w.r.t. revenue')
plt.xlabel('VisitorType')
plt.ylabel('ExitRates')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

```
# region vs exit rates w.r.t. revenue
plt.figure(figure_size=(8,6))
sns.violinplot(data=df, x='Region', y='ExitRates', hue='Revenue', palette='Paired')
plt.title('region vs exit rates w.r.t. revenue')
plt.xlabel('Region')
plt.ylabel('ExitRates')
plt.legend(title='Revenue', labels=['False', 'True'])
plt.show()
```

```
# correlation plot
plt.figure(figure_size=(8,6))
sns.heatmap(data=df.corr(), annot=True, cmap='Spectral')
plt.title('correlation plot')
plt.show
```

Perform required encoding techniques

```
# Month,VisitorType,Weekend,Revenue
df['Month'].value_counts()
df['VisitorType'].value_counts()
df['Weekend'].value_counts()
df['Revenue'].value_counts()
```

```
Month
Nov    2907
May    2732
Mar    1727
Dec    1232
Oct     549
Feb     448
Jul     432
Aug     432
Sep     364
Apr     326
June    288
Name: count, dtype: int64
VisitorType
Returning_Visitor    10551
New_Visitor           1694
Other                   85
Name: count, dtype: int64
Weekend
False    9462
True     2868
Name: count, dtype: int64
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

```
df.sample(2)
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ExitRates	ProductRelated	ProductRelated_Duration	BounceRates	PageValues	Region	TrafficType	Month	OperatingSystems	Browser	VisitorType	Weekend	Revenue
8	0	0.000000	0	0.000000	0.05	1	0.000000	0.200000	0.0	1	9	Mar	2	2	Returning_Visitor	True	True
9	0	0.000000	0	0.000000	0.05	2	64.333333	0.000000	0.0	1	2	Mar	4	6	Returning_Visitor	False	True

DETECTING AND REMOVAL OF OUTLIERS

```
# plot boxplot for numerical features
df.boxplot(figsize=(20,30))
```

Administrative
Administrative_Duration
Informational
Informational_Duration
ExitRates
ProductRelated
ProductRelated_Duration
BounceRates
PageValues
Region
TrafficType
Month
OperatingSystems
Browser
VisitorType
Weekend
Revenue
0
5
10
15
20
25
0
1000
2000
3000
4000
5000
6000
7000
8000
0
200
400
600
800
1000
0
1000
2000
3000
4000
0
200
400
600
800
1000
0
1000
2000
3000
4000
5000
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

```
[ 2 2 2
```

```
# plot the dendrogram
plt.figure(figure=(10,7))
dendrogram = sch.dendrogram(
    Z,
    truncate_mode='lastp',
    p=30,
    show_leaf_counts=True,
    leaf_rotation=90,
    leaf_font_size=12,
    show_contracted=True,
    orientation='top',
    color_threshold=None,
    above_threshold_color='b'
)
# Add a horizontal line at the optimal distance threshold
plt.axhline(y=optimal_distance, c='r', linestyle='--', lw=2, label='Optimal distance')
# Add labels and title
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.legend()
plt.show()
```

```
# Calculate silhouette scores for different numbers of clusters
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
silhouette_scores = []
calinski_scores = []
davies_bouldin_scores = []
for k in range(2, 11):
    cluster_labels = fcluster(Z, k, criterion='maxclust')
    silhouette_avg = silhouette_score(scaled_features, cluster_labels)
    calinski_score = calinski_harabasz_score(scaled_features, cluster_labels)
    davies_bouldin_score = davies_bouldin_score(scaled_features, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    calinski_scores.append(calinski_score)
    davies_bouldin_scores.append(davies_bouldin_score)
    print(f"{k}\t{silhouette_avg:.3f}\t{calinski_score:.3f}\t{davies_bouldin_score:.3f}")
```

```
2	0.333	1.000	0.000
3	0.333	1.000	0.000
4	0.333	1.000	0.000
5	0.333	1.000	0.000
6	0.333	1.000	0.000
7	0.333	1.000	0.000
8	0.333	1.000	0.000
9	0.333	1.000	0.000
10	0.333	1.000	0.000
```

```
# Plot the optimal number of clusters based on different metrics
plt.figure(figure=(15,5))
plt.subplot(1, 3, 1)
plt.plot(range(2, 11), silhouette_scores, 'o-')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis')
plt.subplot(1, 3, 2)
plt.plot(range(2, 11), calinski_scores, 'o-')
plt.xlabel('Number of clusters')
plt.ylabel('Calinski-Harabasz Score')
plt.title('Calinski-Harabasz Analysis')
plt.subplot(1, 3, 3)
plt.plot(range(2, 11), davies_bouldin_scores, 'o-')
plt.xlabel('Number of clusters')
plt.ylabel('Davies-Bouldin Score')
plt.title('Davies-Bouldin Analysis')
plt.tight_layout()
plt.show()
```

```
# Choose optimal number of clusters (based on silhouette score)
optimal_k = silhouette_scores.index(max(silhouette_scores)) + 2
print(f"Optimal number of clusters: {optimal_k}")
```

```
Optimal number of clusters: 2
```

```
# Apply hierarchical clustering with optimal number of clusters
final_clusters = fcluster(Z, optimal_k, criterion='maxclust')
# Add cluster labels to the original data
df['cluster'] = final_clusters
# Display cluster sizes
print("Cluster sizes:")
print(df['cluster'].value_counts())
```

```
Cluster sizes:
cluster
2    500
1    500
Name: count, dtype: int64
```

```
# Create a pairplot with cluster colors
sns.pairplot(df, hue='cluster', palette='viridis')
plt.suptitle('Pairplot of Features by Cluster', y=1.02)
plt.show
```


3. Principal Component Analysis (PCA)

```
# Import PCA and prepare data
from sklearn.decomposition import PCA
# Initialize PCA with all components
pca = PCA()
# Fit PCA on the scaled features
pca_features = pca.fit_transform(scaled_features)
# Print explained variance ratio for each component
print("Explained variance ratio:")
print(pca.explained_variance_ratio_)
```

```
Explained variance ratio:
[0.20000000 0.20000000 0.20000000 0.20000000 0.20000000]
```

```
# Calculate cumulative explained variance
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
print("Cumulative explained variance:")
print(cumulative_variance)
```

```
Cumulative explained variance:
[0.20000000 0.40000000 0.60000000 0.80000000 1.00000000]
```

```
# Create scree plot
plt.figure(figure=(10,6))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('Proportion of Variance Explained')
plt.grid(True)
plt.show()
```



```
# Create a scree plot
plt.figure(figure=(10,6))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('Proportion of Variance Explained')
plt.grid(True)
plt.show()
```



```
# Calculate and print the cumulative explained variance
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
print("Cumulative explained variance:")
print(cumulative_variance)
```

```
Cumulative explained variance:
[0.20000000 0.40000000 0.60000000 0.80000000 1.00000000]
```

```
# Create a dendrogram for hierarchical clustering
plt.figure(figure=(10,7))
dendrogram = sch.dendrogram(
    Z,
    truncate_mode='lastp',
    p=30,
    show_leaf_counts=True,
    leaf_rotation=90,
    leaf_font_size=12,
    show_contracted=True,
    orientation='top',
    color_threshold=None,
    above_threshold_color='b'
)
# Add a horizontal line at the optimal distance threshold
plt.axhline(y=optimal_distance, c='r', linestyle='--', lw=2, label='Optimal distance')
# Add labels and title
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.legend()
plt.show()
```

```
# Calculate the optimal number of components (explaining 95% of variance)
optimal_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Optimal number of components: {optimal_components}")
```

```
Optimal number of components: 1
```

```
# Create a biplot for the first two principal components
plt.figure(figure=(10,8))
for i, cluster in enumerate(np.unique(df['cluster'])):
    mask = df['cluster'] == cluster
    plt.scatter(pca_features[mask, 0], pca_features[mask, 1], c=f'C{i}', label=f'Cluster {cluster}')
# Add feature vectors
for i, feature in enumerate(df.columns[:-1]):
    plt.arrow(0, 0, pca.components_[0, i], pca.components_[1, i], head_width=0.05, head_length=0.05)
    plt.text(pca.components_[0, i], pca.components_[1, i], feature)
plt.title('Biplot')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Create a dendrogram for hierarchical clustering
plt.figure(figure=(10,7))
dendrogram = sch.dendrogram(
    Z,
    truncate_mode='lastp',
    p=30,
    show_leaf_counts=True,
    leaf_rotation=90,
    leaf_font_size=12,
    show_contracted=True,
    orientation='top',
    color_threshold=None,
    above_threshold_color='b'
)
# Add a horizontal line at the optimal distance threshold
plt.axhline(y=optimal_distance, c='r', linestyle='--', lw=2, label='Optimal distance')
# Add labels and title
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.legend()
plt.show()
```

```
# Create a pairplot with cluster colors
sns.pairplot(df, hue='cluster', palette='viridis')
plt.suptitle('Pairplot of Features by Cluster', y=1.02)
plt.show
```

In [15]:
```
# Scatter plot for Annual Income vs Spending Score
# Color by cluster and add cluster centers
plt.figure(figure=None, facecolor='w', edgecolor='k')
sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', hue='Cluster', palette='tab10', data=df, s=50, alpha=0.6)
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', marker='X', label='Centroids')

# Add the cluster centroids to the plot
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.title('K-means Clustering (Top 5)', fontsize=14)
plt.xlabel('Annual Income (k$)', fontsize=12)
plt.ylabel('Spending Score (1-100)', fontsize=12)
plt.grid(True)
plt.show()
```

In [16]:
```
# Save the plot to a file
```