customer-segmentation

October 18, 2023

1 Customer Segmentation

1.1 Table of Contents

1. Functions

2. Preprocess Data

   2.1 Load Data

   2.2 NaN Values (i.e., " ?")

   2.3 Preprocessing Steps

3. EDA

4. Clustering

   4.1 k-Means

   4.2 Normalized k-Means

   4.3 DBSCAN

5. Visualization

   5.1 PCA

   5.2 t-SNE

   5.3 3D Animation

6. Evaluation

7. What makes a cluster unique?

   7.1 Variance

   7.2 Feature Importance

1.2 1. Functions

Back to Table of Contents

[2]:
```
# Data handling
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Clustering
from sklearn.cluster import KMeans, DBSCAN
from sklearn import preprocessing
from sklearn.metrics import silhouette_score

# Dimensionality reduction
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

# Visualization
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
from matplotlib import animation

%matplotlib inline

def load_preprocess_data():
    """ Load and preprocess data
    """

    # Load data
    df = pd.read_csv("data.csv")

    # remove empty values
    df = df.loc[df.TotalCharges!=" ", :]
    df.TotalCharges = df.TotalCharges.astype(float)

    # Label data correctly
    replace_cols = [ 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
                    'TechSupport','StreamingTV', 'StreamingMovies', 'Partner',␣
 ↪'Dependents',
                   'PhoneService', 'MultipleLines', 'PaperlessBilling', 'Churn']
    for i in replace_cols :
        df.loc[:, i]  = df.loc[:, i].replace({'No internet service' : 'No', 'No␣
 ↪phone service':'No'})
        df.loc[:, i]  = df.loc[:, i].map({'No':0, 'Yes':1})
    df.gender = df.gender.map({"Female":0, "Male":1})

    # One-hot encoding of variables
    others_categorical = ['Contract', 'PaymentMethod', 'InternetService']
```

```
    for i in others_categorical:
        df = df.join(pd.get_dummies(df[i], prefix=i))
    df.drop(others_categorical, axis=1, inplace=True)

    # Calculate number of services
    services = ['PhoneService', 'MultipleLines', 'OnlineSecurity',
                'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
                'StreamingMovies', 'InternetService_DSL', 'InternetService_Fiber␣
↪optic',
                'InternetService_No']
    df['nr_services'] = df.apply(lambda row: sum([row[x] for x in services[:
↪-1]]), 1)

    return df.drop('customerID', 1)

def plot_corr(df):
    corr = df.corr()
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    f, ax = plt.subplots(figsize=(11, 9))
    cmap = sns.diverging_palette(220, 10, as_cmap=True)
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})

def plot_tsne(tnse_data, kmeans_labels):
    df_tsne = pd.DataFrame(tsne_data).rename({0: 'x', 1: 'y'}, axis=1)
    df_tsne['z'] = kmeans_labels
    sns.scatterplot(x=df_tsne.x, y=df_tsne.y, hue=df_tsne.z, palette="Set2")
    plt.show()

def prepare_pca(n_components, data, kmeans_labels):
    names = ['x', 'y', 'z']
    matrix = PCA(n_components=n_components).fit_transform(data)
    df_matrix = pd.DataFrame(matrix)
    df_matrix.rename({i:names[i] for i in range(n_components)}, axis=1,␣
↪inplace=True)
    df_matrix['labels'] = kmeans_labels

    return df_matrix

def prepare_tsne(n_components, data, kmeans_labels):
    names = ['x', 'y', 'z']
    matrix = TSNE(n_components=n_components).fit_transform(data)
    df_matrix = pd.DataFrame(matrix)
    df_matrix.rename({i:names[i] for i in range(n_components)}, axis=1,␣
↪inplace=True)
    df_matrix['labels'] = kmeans_labels
```

```
    return df_matrix

def plot_3d(df, name='labels'):
    iris = px.data.iris()
    fig = px.scatter_3d(df, x='x', y='y', z='z',
                  color=name, opacity=0.5)


    fig.update_traces(marker=dict(size=3))
    fig.show()

def plot_animation(df, label_column, name):
    def update(num):
        ax.view_init(200, num)

    N=360
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(tsne_3d_df['x'], tsne_3d_df['y'], tsne_3d_df['z'],␣
 ↪c=tsne_3d_df[label_column],
               s=6, depthshade=True, cmap='Paired')
    ax.set_zlim(-15, 25)
    ax.set_xlim(-20, 20)
    plt.tight_layout()
    ani = animation.FuncAnimation(fig, update, N, blit=False, interval=50)
    ani.save('{}.gif'.format(name), writer='imagemagick')
    plt.show()
```

1.3 2. Preprocess Data

Back to Table of Contents

Demographic * Gender * SeniorCitizen * Partner * Dependents * Tenure

Services * PhoneService * MultipleLines * InternetService * OnlineSecurity * OnlineBackup * DeviceProtection * TechSupport * StreamingTV * StreamingMovies

Customer account information * Contract * PaperlessBilling * PaymentMethod * MonthlyCharges * TotalCharges

Target * Churn

No = 0 Yes = 1

Female = 0 Male = 1

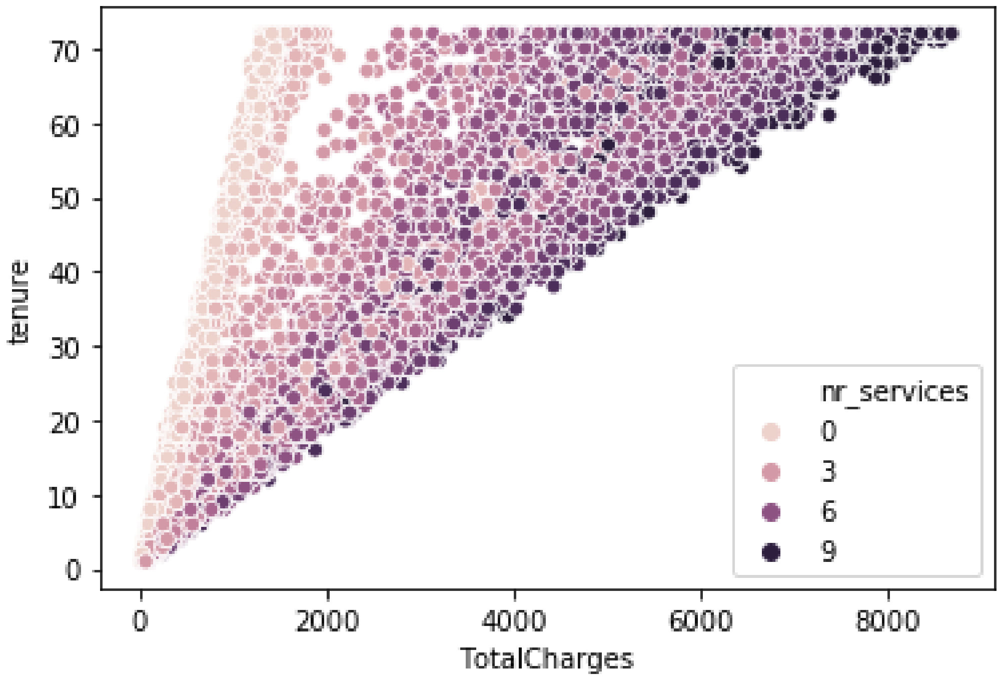
[3]:
```
df = load_preprocess_data()
```

1.4 3. EDA

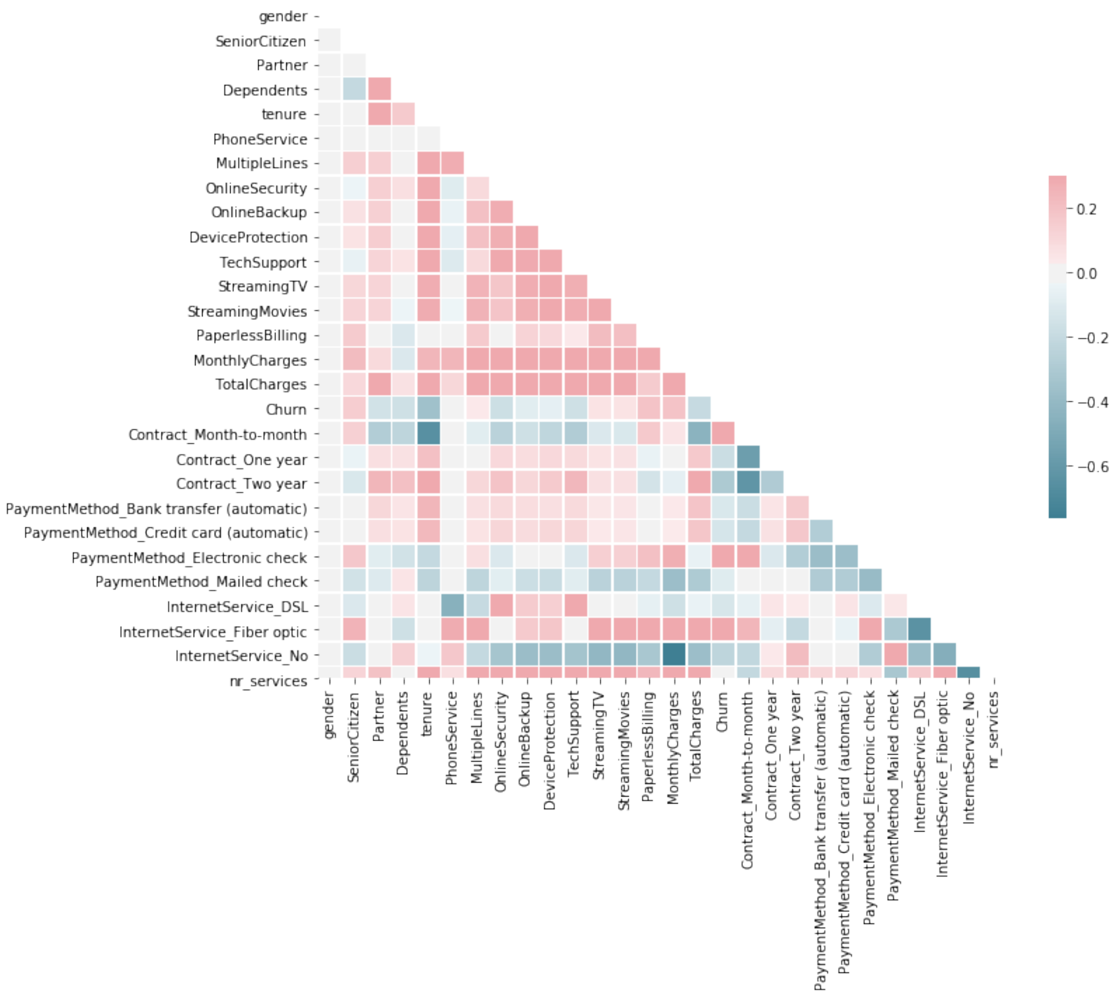
Back to Table of Contents

[10]:
```
sns.scatterplot(df.TotalCharges, df.tenure, df.nr_services)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e91517c4e0>
```



[11]:
```
plot_corr(df)
```



1.5 4. Clustering

Back to Table of Contents

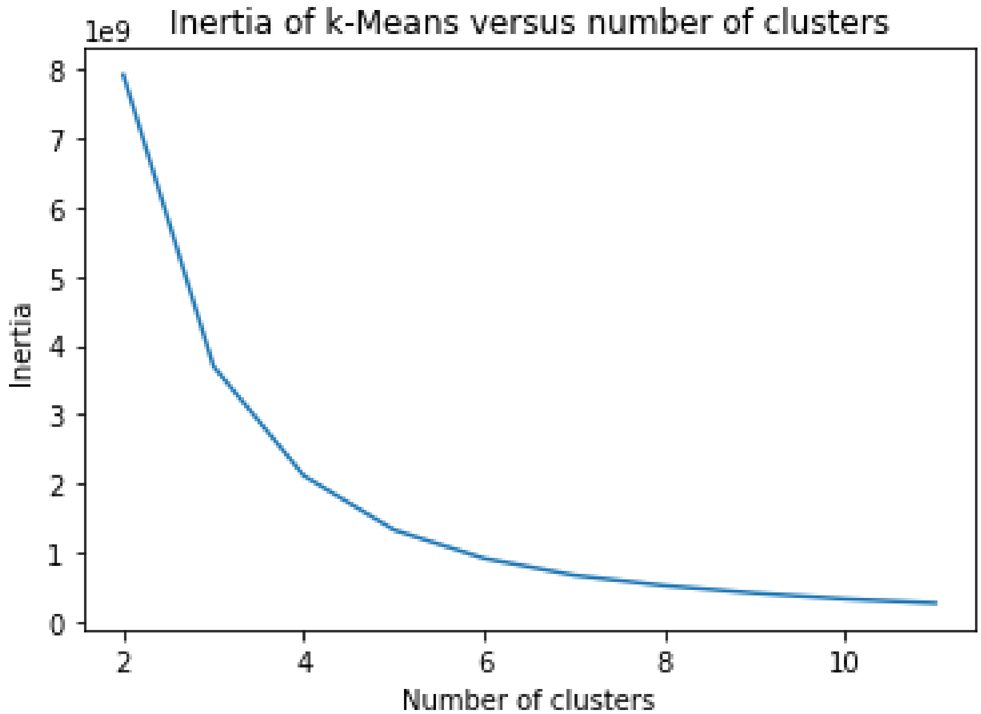
[4]:
```
df = df.drop(["Churn"], 1)
```

1.5.1 4.1. k-Means

Back to Table of Contents

[13]:
```
scores = [KMeans(n_clusters=i+2).fit(df).inertia_ for i in range(10)]
sns.lineplot(np.arange(2, 12), scores)
plt.xlabel('Number of clusters')
plt.ylabel("Inertia")
plt.title("Inertia of k-Means versus number of clusters")
```

[13]:
```
Text(0.5, 1.0, 'Inertia of k-Means versus number of clusters')
```



[14]:
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(df)
```

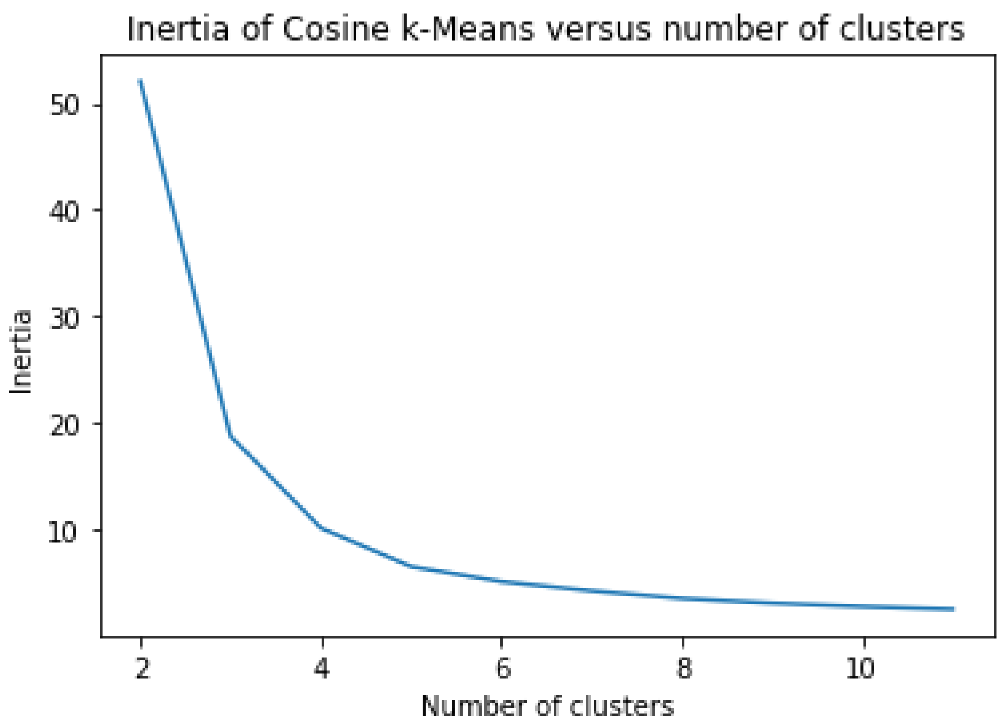
[14]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

1.5.2 4.2. Normalized k-Means

Back to Table of Contents

[15]:
```
normalized_vectors = preprocessing.normalize(df)
scores = [KMeans(n_clusters=i+2).fit(normalized_vectors).inertia_ for i in␣
 ↪range(10)]
sns.lineplot(np.arange(2, 12), scores)
plt.xlabel('Number of clusters')
plt.ylabel("Inertia")
plt.title("Inertia of Cosine k-Means versus number of clusters")
```

[15]:
```
Text(0.5, 1.0, 'Inertia of Cosine k-Means versus number of clusters')
```



[16]:
```
normalized_kmeans = KMeans(n_clusters=4)
normalized_kmeans.fit(normalized_vectors)
```

[16]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

1.5.3 4.3. DBSCAN

Back to Table of Contents

[18]:
```
min_samples = df.shape[1]+1 #  Rule of thumb; number of dimensions D in the
  ↪data set, as minPts   D + 1
dbscan = DBSCAN(eps=3.5, min_samples=min_samples).fit(df)
```

1.6 5. Visualization

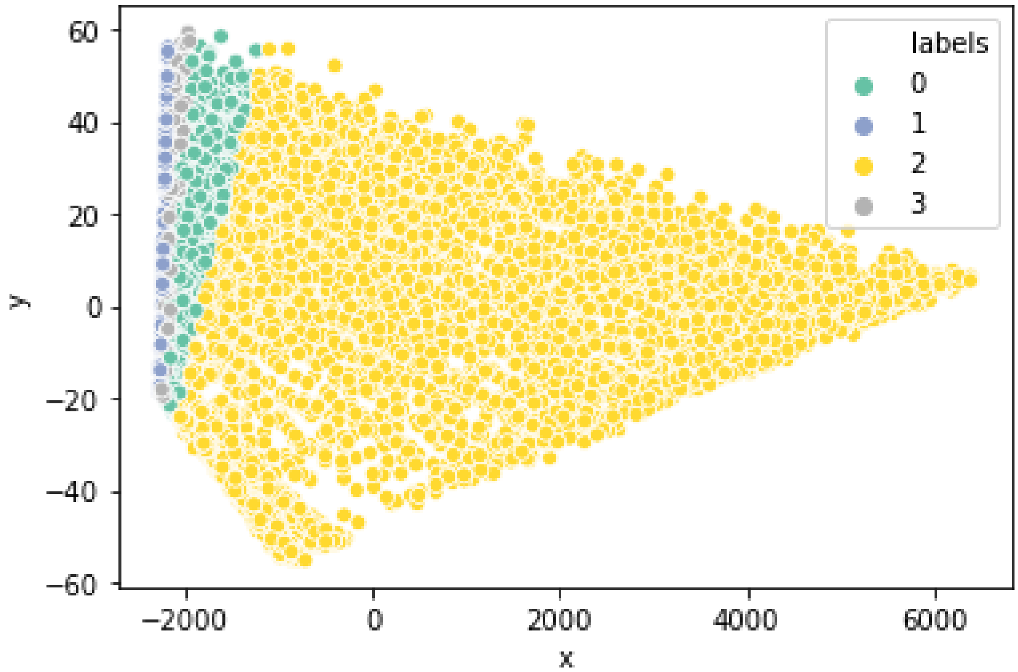
Back to Table of Contents

1.6.1 5.1. PCA

Back to Table of Contents

[112]:
```
pca_df = prepare_pca(3, df, normalized_kmeans.labels_)
sns.scatterplot(x=pca_df.x, y=pca_df.y, hue=pca_df.labels, palette="Set2")
```

[112]: <matplotlib.axes._subplots.AxesSubplot at 0x2292d211e48>



[108]:
```
pca_df = prepare_pca(3, df, normalized_kmeans.labels_)
plot_3d(pca_df)
```

1.6.2 5.2. t-SNE

Back to Table of Contents

[21]:
```
tsne_3d_df = prepare_tsne(3, df, kmeans.labels_)
```

[22]:
```
plot_3d(tsne_3d_df)
```

[307]:
```
tsne_3d_df['normalized_kmeans'] = normalized_kmeans.labels_
plot_3d(tsne_3d_df, name='normalized_kmeans')
```

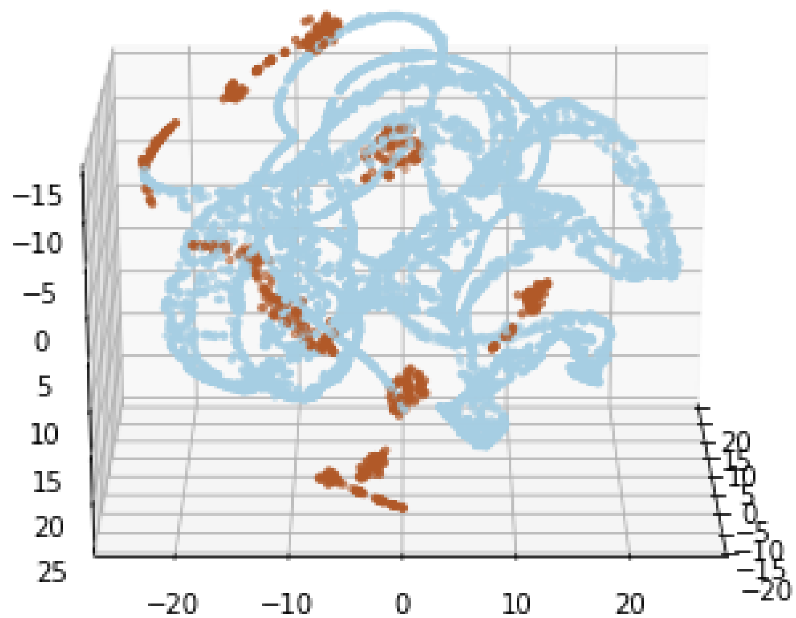
[67]:
```
min_samples = clustering.shape[1]+1 #  Rule of thumb; number of dimensions D in␣
 ↪the data set, as minPts   D + 1
dbscan = DBSCAN(eps=3.5, min_samples=min_samples).fit(clustering)
# dbscan = DBSCAN(eps=50, min_samples=min_samples).fit(clustering)
tsne_3d_df['dbscan'] = [str(label) for label in dbscan.labels_]
plot_3d(tsne_3d_df, name='dbscan')
```

1.6.3 5.3. 3D Animation

Back to Table of Contents

[321]:
```
tsne_3d_df.dbscan = tsne_3d_df.dbscan.astype(int)
plot_animation(tsne_3d_df, 'normalized_kmeans', 'normalized_kmeans_new')
```

MovieWriter imagemagick unavailable; trying to use <class 'matplotlib.animation.PillowWriter'> instead.



1.7 6. Evaluation

Back to Table of Contents

[7]:
```
kmeans = KMeans(n_clusters=4).fit(df)

normalized_vectors = preprocessing.normalize(df)
normalized_kmeans = KMeans(n_clusters=4).fit(normalized_vectors)

min_samples = df.shape[1]+1 #  Rule of thumb; number of dimensions D in the␣
 ↪data set, as minPts   D + 1
dbscan = DBSCAN(eps=3.5, min_samples=min_samples).fit(df)
```

[105]:
```
print('kmeans: {}'.format(silhouette_score(df, kmeans.labels_,␣
 ↪metric='euclidean')))
```

```
print('Cosine kmeans: {}'.format(silhouette_score(normalized_vectors,␣
 ↪normalized_kmeans.labels_, metric='cosine')))
print('DBSCAN: {}'.format(silhouette_score(df, dbscan.labels_,␣
 ↪metric='cosine')))
```

```
kmeans: 0.6018318118002677
Cosine kmeans: 0.8633823077551214
DBSCAN: 0.8302013261718773
```

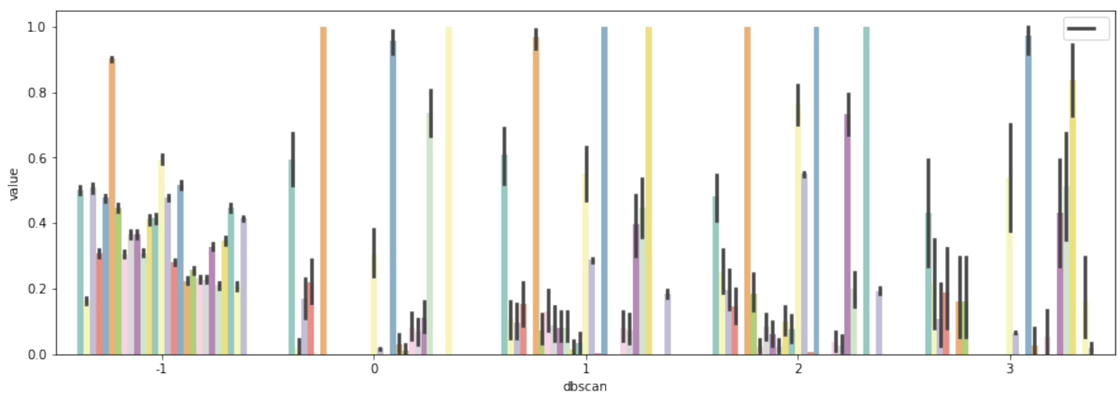
1.8 7. What makes a cluster unique?

Back to Table of Contents

One way to see the differences between clusters is to take the average value of each cluster and visualize it.

[9]:
```
# Setting all variables between 0 and 1 in order to better visualize the results
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df))
df_scaled.columns = df.columns
df_scaled['dbscan'] = dbscan.labels_
```

[17]:
```
# df = load_preprocess_data()
df['dbscan'] = dbscan.labels_
tidy = df_scaled.melt(id_vars='dbscan')
fig, ax = plt.subplots(figsize=(15, 5))
sns.barplot(x='dbscan', y='value', hue='variable', data=tidy, palette='Set3')
plt.legend([''])
# plt.savefig("mess.jpg", dpi=300)
plt.savefig("dbscan_mess.jpg", dpi=300)
```



The problem with this approach is that we simply have too many variables. Not all of them are likely to be important when creating the clusters. Instead, I will select the most important columns based on the following approach:

1.8.1 7.1. Variance

Back to Table of Contents

What I essentially do is group datapoints by cluster and take the average. Then, I calculate the standard deviation between those values for each variable. Variables with a high standard deviation indicate that there are large differences between clusters and that the variable might be important.

[11]:
```
df_mean = df_scaled.loc[df_scaled.dbscan!=-1, :].groupby('dbscan').mean().
↪reset_index()
```

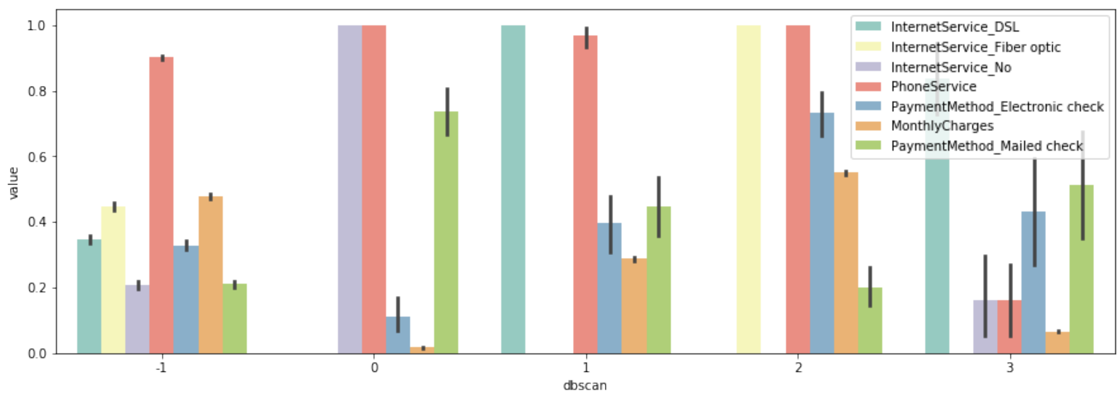
[12]:
```
df_mean
```

[12]:
```
   dbscan    gender  SeniorCitizen   Partner  Dependents  tenure  \
0       0  0.593750       0.018750  0.168750    0.218750     0.0
1       1  0.609756       0.105691  0.097561    0.154472     0.0
2       2  0.480447       0.251397  0.195531    0.145251     0.0
3       3  0.432432       0.216216  0.108108    0.189189     0.0

   PhoneService  MultipleLines  OnlineSecurity  OnlineBackup  …  \
0      1.000000       0.000000        0.000000      0.000000  …
1      0.967480       0.073171        0.130081      0.089431  …
2      1.000000       0.184358        0.022346      0.083799  …
3      0.162162       0.162162        0.000000      0.000000  …

   Contract_One year  Contract_Two year  \
0           0.031250             0.0125
1           0.000000             0.0000
2           0.000000             0.0000
3           0.027027             0.0000

   PaymentMethod_Bank transfer (automatic)  \
0                                 0.081250
1                                 0.081301
2                                 0.039106
3                                 0.054054

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
0                               0.068750                        0.112500
1                               0.073171                        0.398374
2                               0.027933                        0.731844
3                               0.000000                        0.432432

   PaymentMethod_Mailed check  InternetService_DSL  \
0                    0.737500             0.000000
1                    0.447154             1.000000
2                    0.201117             0.000000
3                    0.513514             0.837838
```

```
   InternetService_Fiber optic  InternetService_No  nr_services
0                          0.0            1.000000     0.000000
1                          0.0            0.000000     0.183943
2                          1.0            0.000000     0.194134
3                          0.0            0.162162     0.020270

[4 rows x 28 columns]
```

[15]:

```python
# Setting all variables between 0 and 1 in order to better visualize the results
# df = load_preprocess_data().drop("Churn", 1)
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df))
df_scaled.columns = df.columns
df_scaled['dbscan'] = dbscan.labels_

# Calculate variables with largest differences (by standard deviation)
# The higher the standard deviation in a variable based on average values for
↪each cluster
# The more likely that the variable is important when creating the cluster
df_mean = df_scaled.loc[df_scaled.dbscan!=-1, :].groupby('dbscan').mean().
↪reset_index()
results = pd.DataFrame(columns=['Variable', 'Std'])
for column in df_mean.columns[1:]:
    results.loc[len(results), :] = [column, np.std(df_mean[column])]
selected_columns = list(results.sort_values('Std', ascending=False).head(7).
↪Variable.values) + ['dbscan']

# Plot data
tidy = df_scaled[selected_columns].melt(id_vars='dbscan')
fig, ax = plt.subplots(figsize=(15, 5))
sns.barplot(x='dbscan', y='value', hue='variable', data=tidy, palette='Set3')
plt.legend(loc='upper right')
plt.savefig("dbscan_results.jpg", dpi=300)
```

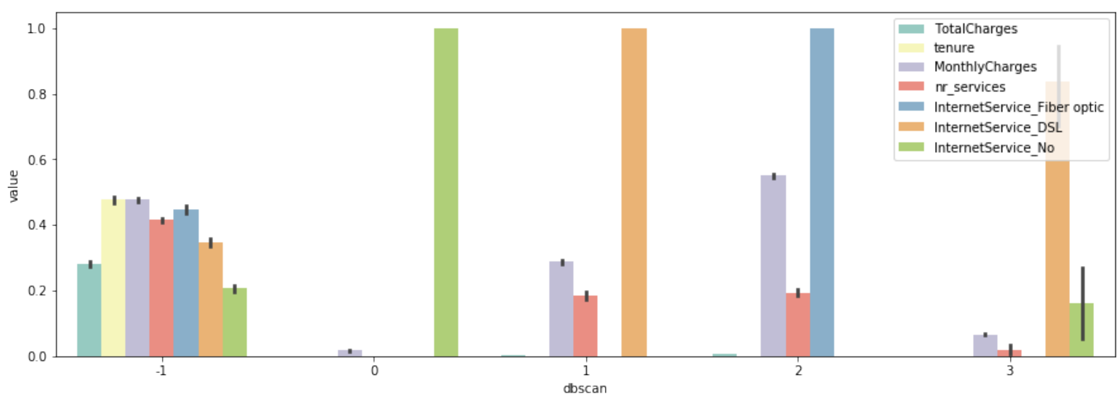


1.8.2 7.2. Feature Importance

Back to Table of Contents

[18]:
```
from sklearn.ensemble import RandomForestClassifier
y = df.iloc[:,-1]
X = df.iloc[:,:-1]
clf = RandomForestClassifier(n_estimators=100).fit(X, y)
selected_columns = list(pd.DataFrame(np.array([clf.feature_importances_, X.
↪columns]).T, columns=['Importance', 'Feature'])
           .sort_values("Importance", ascending=False)
           .head(7)
           .Feature
           .values)
```

[20]:
```
# Plot data
tidy = df_scaled[selected_columns+['dbscan']].melt(id_vars='dbscan')
fig, ax = plt.subplots(figsize=(15, 5))
sns.barplot(x='dbscan', y='value', hue='variable', data=tidy, palette='Set3')
plt.legend(loc='upper right')
plt.savefig('randomforest.jpg', dpi=300)
```



[ ]: