Problem Statement

Suppose we have a dataset containing information about customers of an e-commerce website, including their age, gender, location, purchase history, and whether they made a purchase in the last month. We want to identify the most important features for predicting whether a customer is likely to make a purchase in the future.

In [ ]:
```
import pandas as pd

df = pd.read_csv('ecommerce_data.csv')
```

- Prepare the Data

We need to prepare the data for feature selection. In this case, we want to perform chi-squared feature selection, which requires the data to be in the form of a contingency table. We can use the pd.crosstab function to create a contingency table:

In [ ]:
```
contingency_table = pd.crosstab(df['gender'], df['purchased'])
```

This creates a contingency table that shows the number of purchases and non-purchases for each gender. We can use this table to perform a chi-squared test of independence to determine whether gender is a significant predictor of purchase behavior.

- Perform Chi-Squared Test

We can perform a chi-squared test of independence using the chi2_contingency function from the SciPy library:

In [ ]:
```
from scipy.stats import chi2_contingency

stat, p, dof, expected = chi2_contingency(contingency_table)
```

This computes the chi-squared test statistic, the p-value, the degrees of freedom, and the expected values. We can use the p-value to determine whether gender is a significant predictor of purchase behavior. If the p-value is less than our chosen significance level (e.g., 0.05), we can reject the null hypothesis and conclude that there is a significant relationship between gender and purchase behavior.

- Perform Feature Selection

We can use the chi-squared test to perform feature selection on the entire dataset. Specifically, we can use the SelectKBest class from the scikit-learn library to select the top k features based on the chi-squared test:

In [ ]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X = df.drop('purchased', axis=1)
y = df['purchased']

selector = SelectKBest(chi2, k=3)
X_new = selector.fit_transform(X, y)
```

This selects the top 3 features based on the chi-squared test and creates a new DataFrame X_new that contains only those features. We can then use this reduced feature set to train a machine learning model.

- Model

We can use the reduced feature set X_new to train a machine learning model, such as logistic regression or decision tree:

In [ ]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_new, y)
```

This trains a logistic regression model on the reduced feature set. We can evaluate the performance of the model using cross-validation or a holdout set.

- Interpret the Results

Finally, we can interpret the results of the model to identify the most important features for predicting purchase behavior. For example, we can look at the coefficients of the logistic regression model:

In [ ]:
```
print(model.coef_)
```

This prints the coefficients of the model for each feature. The features with the largest coefficients are the most important predictors of purchase behavior.

In [ ]:
```

```

In [25]:
```
# Load your data into a Pandas DataFrame
df = pd.read_csv("insurance-data.csv")

# Split into features and target
X = df.drop(["sex","smoker","region"], axis=1)
y = df["charges"]
```

In [28]:
```
df["sex"] = df["sex"].map({"female":0,"male":1})
```

In [29]:
```
df["smoker"] = df["smoker"].map({"yes":0,"no":1})
```

In [31]:
```
df["region"].value_counts()
```

Out[31]:
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

In [32]:
```
df["region"] = df["region"].map({"southeast":0,"southwest":1,"northwest":2,"northe
```

In [33]:
```
df.head()
```

Out[33]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	0	1	16884.92400
1	18	1	33.770	1	1	0	1725.55230
2	28	1	33.000	3	1	0	4449.46200
3	33	1	22.705	0	1	2	21984.47061
4	32	1	28.880	0	1	2	3866.85520

In [34]:
```
X = df.drop(["smoker"], axis=1)
y = df["smoker"]
```

In [39]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# X is your feature matrix and y is your target vector

# create a SelectKBest object with mutual information scoring
selector = SelectKBest(mutual_info_classif, k=4)

# fit the selector to the data and transform the features
X_new = selector.fit_transform(X, y)

# get the indices of the selected features
selected_features_indices = selector.get_support(indices=True)

# get the names of the selected features
selected_features = X.columns[selected_features_indices]

# print the selected features
print("Selected features:", selected_features)
```

```
Selected features: Index(['sex', 'bmi', 'children', 'charges'], dtype='object')
```

kendalltau

In [40]:
```
df.head()
```

Out[40]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	0	1	16884.92400

	age	sex	bmi	children	smoker	region	charges
1	18	1	33.770	1	1	0	1725.55230
2	28	1	33.000	3	1	0	4449.46200
3	33	1	22.705	0	1	2	21984.47061

In [41]:
```
import pandas as pd
import numpy as np
from scipy.stats import kendalltau


# split features and target
X = df.drop(["charges"], axis=1)
y = df["charges"]
```

In [42]:
```
# compute Kendall's rank correlation coefficient for each feature
correlations = []
for feature in X.columns:
    corr, _ = kendalltau(X[feature], y)
    correlations.append(corr)

# create a dataframe with feature names and their correlations with the target
results = pd.DataFrame({'feature': X.columns, 'correlation': correlations})

# sort by absolute correlation value (ignoring the sign)
results['abs_corr'] = results['correlation'].abs()
results = results.sort_values('abs_corr', ascending=False).reset_index(drop=True)

# print the top 10 features with the highest absolute correlation value
print(results.head(10))
```

```
    feature  correlation  abs_corr
0    smoker    -0.541916  0.541916
1       age     0.475302  0.475302
2  children     0.103107  0.103107
3       bmi     0.082524  0.082524
4    region     0.015261  0.015261
5       sex     0.007751  0.007751
```

In [ ]:

CHI-square

In [45]:
```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# Load the iris dataset
iris = load_iris()

# Select the 2 best features using chi-square test
X_new = SelectKBest(chi2, k=2).fit_transform(iris.data, iris.target)

# Print the selected features
print(X_new.shape)
```

```
(150, 2)
```

In [ ]:

ANOVA

In [ ]:
```
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# Load data
data = pd.read_csv('data.csv')

# Split data into features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Apply ANOVA to select k best features
k = 5  # Number of top features to select
anova_filter = SelectKBest(f_classif, k=k)
X_filtered = anova_filter.fit_transform(X, y)

# Print selected feature names
selected_features = X.columns[anova_filter.get_support()]
print(selected_features)
```

In [ ]:

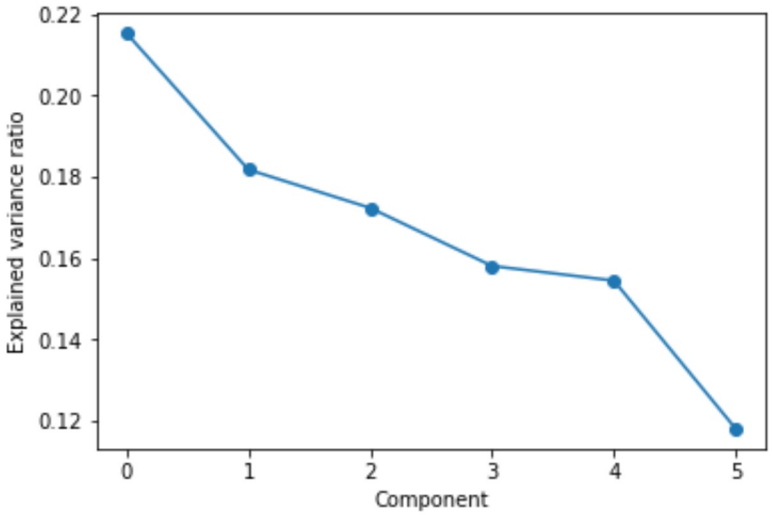
In [ ]:

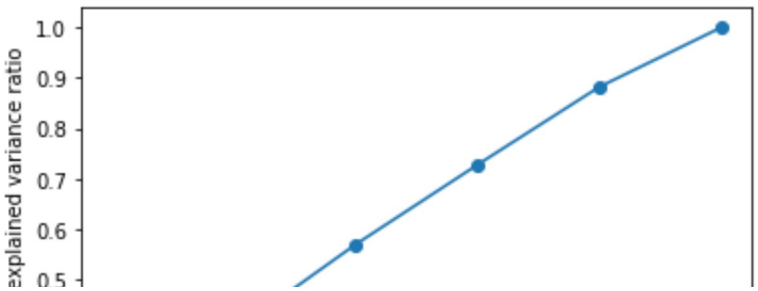
PCA

In [46]:

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt



# Split data into features and target
X = df.drop(["charges"], axis=1)
y = df["charges"]
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Determine number of components to keep
explained_var_ratio = pca.explained_variance_ratio_
cumulative_var_ratio = np.cumsum(explained_var_ratio)
n_components = np.argmax(cumulative_var_ratio >= 0.95) + 1
print(f"Number of components to keep: {n_components}")

# Plot explained variance ratio
plt.plot(explained_var_ratio, marker='o')
plt.xlabel('Component')
plt.ylabel('Explained variance ratio')
plt.show()

# Plot cumulative explained variance ratio
plt.plot(cumulative_var_ratio, marker='o')
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance ratio')
plt.show()
```

```
Number of components to keep: 6
```





In this example, we first load the data from a CSV file and split it into features (X) and target (y). We then scale the features using StandardScaler from scikit-learn's preprocessing module, which is necessary for PCA to work properly. We then create an instance of the PCA class from scikit-learn's decomposition module and fit it to the scaled features. The resulting transformed features are stored in X_pca.

To determine the number of components to keep, we examine the explained variance ratio of each component using the explained_variance*ratio* attribute of the PCA object. We also calculate the cumulative explained variance ratio using the cumsum function from NumPy. We choose to keep the minimum number of components that explains at least 95% of the variance.

Finally, we plot the explained variance ratio and the cumulative explained variance ratio using Matplotlib to visualize the results.

Note that this is just a simple example, and in practice you may want to tune the parameters of the PCA algorithm, or try other dimensionality reduction techniques to see what works best for your data. Additionally, you may want to perform additional analysis or modeling using the reduced-dimensional data after performing PCA.

In [ ]:

This code uses Matplotlib to create a line plot of the explained variance ratio of each principal component. Here's what each line does:

plt.plot(explained_var_ratio, marker='o'): This creates the actual plot, using the explained_var_ratio variable as the y-values and range(len(explained_var_ratio)) as the x-values (since each principal component corresponds to a single number in the array). The marker='o' argument specifies that we want to plot a circle marker at each data point. plt.xlabel('Component'): This sets the label for the x-axis to "Component". plt.ylabel('Explained variance ratio'): This sets the label for the y-axis to "Explained variance ratio". plt.show(): This displays the plot.

In [ ]: