Part B

Q1.

1.1 At the above BPTT function, it is observably that 'self,ForwardPass(sample, slide)' includes two parameter 'sample' and 'slide', the reasons will list below:

1. Processing sequence data: BPTT is designed for processing sequence data, especially in recurrent neural networks (RNN). In this case, 'sample' stands for a sample in a sequence, such as a time series of data or a sentence.
2. Processing time step: The slide parameter represents the time step or a specific moment in the sequence. In sequence processing the network needs to process a portion of the input data at each time step. For example, when working with a piece of text, each "slide" might represent a word in the sentence.

In conclusion, the two parameters 'sample' and 'slide' in self.ForwardPass(sample, slide) enable the algorithm to process the input correctly at each point in time in the d sequence and maintain continuity and dependency in time, which is critical for processing time series data or any form of sequence data.

Reference:

1. https://www.wrike.com/project-management-guide/faq/what-is-forward-pass-in-project-management/
2. https://datascience.stackexchange.com/questions/116973/whats-a-forward-pass-in-a-neural-net

1.2 In the provided K-Means algorithm Python code, the formation and refinement of clusters are explained through the following steps:

1. **Initialization of Centroids (Lines 11-13):**
   - The `fit` function starts by selecting the first `k` points from the dataset as the initial centroids.

2. **Iteration Process (Lines 15-50):**
   - The algorithm performs a loop for a maximum number of iterations, controlled by the `max_iter` parameter(in line8, 11, 21). Each iteration updates the cluster assignments and the positions of the centroids.

3. **Assigning Points to the Nearest Centroid (Lines 22-29):**
   - For each point (`featureset`) in the dataset, the Euclidean distance to each centroid is calculated. The point is then assigned to the cluster of the closest centroid.

4. **Updating Centroid Positions (Lines 33-35):**
   - For each cluster, the average position of all its points is calculated, and this average is set as the new position of the centroid.

5. **Checking for Convergence (Lines 37-46):**
   - The algorithm checks if the movement of all centroids is less than a set tolerance (`tol`). If the movement of all centroids is below this threshold, it indicates that the algorithm has converged, and the iteration ends.

6. **Termination and Returning Centroids (Lines 48-49):**
   - Once the algorithm converges or reaches the maximum number of iterations, the iteration stops, and the current positions of the centroids are returned.

This process effectively involves iteratively reassigning data points to their nearest centroids and updating the positions of centroids based on these assignments. This repeated process ensures that over time, clusters gradually stabilize and ultimately converge to a relatively stable state, forming the final clusters.

Reference:

1. https://en.wikipedia.org/wiki/K-means_clustering
2. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
3. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
4. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5. https://www.stata.com/manuals13/rmaximize.pdf

Q2.

2.1 Choosing the Appropriate Prior Distribution:

The appropriate prior distribution for a model depends on the prior knowledge about the parameters and the problem domain. For instance, if you know a parameter is positive and likely to be in a specific range, a uniform or log-normal prior might be appropriate. If there's no specific prior knowledge, a non-informative or weakly informative prior, such as a normal distribution with a large variance, could be used to reflect this uncertainty. In a Bayesian neural network, priors over weights could be chosen based on the expected size of the weights or based on regularization considerations—smaller weights might suggest a prior that favors smaller absolute values, like a Gaussian centered at zero.

2.2 Types of Priors in the Given Equation:

The equation shows two priors that are derived from the inverse gamma distribution, which is a type of conjugate prior for the variance in a Gaussian distribution. Conjugate priors are often used in Bayesian statistics because they simplify the posterior updating process. The

reason for using two different priors (parameters v1 and v2 for the inverse gamma distributions) in the Gaussian likelihood for regression problems is to separately model the uncertainty in the scale of the data (the variance of the Gaussian likelihood) and the scale of the weights (from the regularization term). These separate priors allow for more flexibility in modeling; the data variance and weight variance can be independently controlled and updated based on observed data.

2.3 Suitability of Priors for Multinomial Likelihood:
The given priors are not suitable for a multinomial likelihood because a multinomial distribution models discrete outcomes and typically requires different types of priors, like the Dirichlet distribution, which is the conjugate prior for the parameters of a multinomial distribution. If one were to use these priors for a multinomial likelihood, they would need to change to reflect the discrete nature of the data and the different parameter space—for example, using a Dirichlet prior instead of an inverse gamma, to model the probabilities associated with the different possible outcomes of the multinomial distribution. This change is necessary because the support and nature of the parameters (probabilities in multinomial vs. variances in Gaussian) are fundamentally different.

PART C OPTION 1

Part A

In [4]:
```
import pandas as pd

data = pd.read_csv('dataset.csv')

data.replace({'?': None, '$': None}, inplace=True)
data.dropna(inplace=True)

correlation_matrix = data.corr()
print(correlation_matrix)
```

	f1	f2	f3	f4	f5	f6	f7	\
f1	1.000000	0.024178	0.026541	-0.025051	-0.008979	0.031344	0.025752	
f2	0.024178	1.000000	-0.022585	-0.102970	-0.004265	0.120531	0.027509	
f3	0.026541	-0.022585	1.000000	-0.034716	0.021287	-0.006907	-0.005938	
f4	-0.025051	-0.102970	-0.034716	1.000000	0.047946	-0.699204	0.292030	
f5	-0.008979	-0.004265	0.021287	0.047946	1.000000	-0.033049	-0.014263	
f6	0.031344	0.120531	-0.006907	-0.699204	-0.033049	1.000000	0.009467	
f7	0.025752	0.027509	-0.005938	0.292030	-0.014263	0.009467	1.000000	
f8	-0.029742	-0.121417	0.020273	0.302272	0.001408	-0.599658	-0.053808	
f9	0.044816	0.112239	-0.046483	-0.055234	-0.000653	0.576507	0.455235	
f10	-0.007634	-0.008145	-0.022678	-0.226376	0.016959	0.118209	-0.842103	
f11	0.057505	-0.027015	-0.021438	0.002516	0.025575	0.019936	-0.015426	
f12	0.019111	0.101742	0.007091	-0.321061	-0.008118	0.407687	0.075312	
f13	-0.014891	0.003094	0.051952	-0.025602	-0.010357	0.057332	-0.079692	
f14	-0.007312	-0.006081	-0.051062	0.108898	0.035246	0.169213	-0.621953	
f15	0.006163	-0.058493	-0.007918	-0.006491	0.050536	0.027631	-0.013857	
f16	0.017289	-0.012710	0.019812	-0.103604	-0.017921	-0.387987	0.165368	
f17	-0.009738	0.047059	-0.001671	-0.191294	0.008545	0.411145	-0.332460	
f18	-0.063372	-0.007230	-0.017055	0.001795	0.019621	-0.007303	-0.011469	
f19	-0.002234	-0.050260	-0.009334	0.037216	-0.010262	0.018309	0.052455	
f20	0.018069	0.080218	-0.034082	-0.409114	0.009046	0.552347	-0.579042	
f21	-0.017463	-0.097170	-0.033516	0.571391	0.024123	-0.345657	0.051155	
f22	0.004772	-0.038985	-0.009580	-0.041336	-0.004397	-0.151122	-0.160618	
f23	0.008686	0.038552	0.016600	-0.021287	-0.012129	-0.019353	0.001645	
f24	-0.053350	-0.108050	0.031963	0.137354	0.014752	-0.433902	-0.605263	
f25	-0.014831	-0.070345	0.047242	-0.133689	-0.027092	-0.359910	0.021722	
f26	-0.023408	-0.025874	-0.017016	0.510890	0.031412	-0.089290	0.112017	
f27	0.014925	-0.014340	0.006747	-0.033239	-0.014426	0.009167	-0.046281	
f28	-0.046011	-0.124488	-0.007485	0.758448	0.032273	-0.501331	0.076632	
f29	-0.022659	-0.051485	0.050156	-0.062255	-0.029227	-0.067606	0.280334	
f30	-0.021592	-0.051265	-0.001757	0.300410	-0.003007	0.145092	0.360535	
target	-0.029275	0.074944	-0.021559	-0.038915	-0.031068	0.098351	0.013298	

	f8	f9	f10	...	f22	f23	f24	\
f1	-0.029742	0.044816	-0.007634	...	0.004772	0.008686	-0.053350	
f2	-0.121417	0.112239	-0.008145	...	-0.038985	0.038552	-0.108050	
f3	0.020273	-0.046483	-0.022678	...	-0.009580	0.016600	0.031963	
f4	0.302272	-0.055234	-0.226376	...	-0.041336	-0.021287	0.137354	
f5	0.001408	-0.000653	0.016959	...	-0.004397	-0.012129	0.014752	
f6	-0.599658	0.576507	0.118209	...	-0.151122	-0.019353	-0.433902	
f7	-0.053808	0.455235	-0.842103	...	-0.160618	0.001645	-0.605263	

f8	1.000000	-0.733879	0.066348	...	0.662554	0.044240	0.461074
f9	-0.733879	1.000000	-0.110719	...	-0.188413	-0.019804	-0.883449
f10	0.066348	-0.110719	1.000000	...	0.516596	0.014121	0.195963
f11	-0.052518	0.040609	0.021028	...	-0.026897	-0.009053	-0.018769
f12	-0.898579	0.438881	-0.309563	...	-0.858143	-0.043998	-0.211232
f13	-0.054597	0.015026	0.071799	...	-0.031532	0.018019	0.033172
f14	-0.175887	0.214647	0.849696	...	0.256153	-0.019427	0.047354
f15	0.008237	0.001454	0.022818	...	0.015546	-0.007325	0.008333
f16	0.534632	-0.217653	0.011876	...	0.696010	0.078292	-0.246865
f17	-0.721759	0.168313	0.033032	...	-0.817382	-0.068477	0.252698
f18	0.042982	-0.014933	0.031221	...	0.057559	-0.000869	0.001057
f19	0.708777	-0.167900	0.279161	...	0.860774	0.042117	-0.051610
f20	-0.588693	0.456947	0.756988	...	0.054204	-0.013325	-0.236078
f21	0.694663	-0.152292	0.271715	...	0.703200	0.019192	0.068545
f22	0.662554	-0.188413	0.516596	...	1.000000	0.063843	-0.112955
f23	0.044240	-0.019804	0.014121	...	0.063843	1.000000	-0.016664
f24	0.461074	-0.883449	0.195963	...	-0.112955	-0.016664	1.000000
f25	0.861051	-0.719886	-0.098985	...	0.564010	0.057124	0.346269
f26	-0.400300	0.116624	-0.357880	...	-0.820138	-0.075273	0.271854
f27	-0.083301	0.022729	0.022507	...	-0.060902	0.001461	-0.003416
f28	0.450423	-0.424876	-0.235732	...	-0.174654	-0.038062	0.594182
f29	0.464902	-0.547663	-0.575020	...	-0.156438	-0.002346	0.446892
f30	0.249397	-0.085459	-0.403717	...	-0.171429	-0.037887	0.225745
target	-0.263550	0.232841	0.027917	...	-0.112013	-0.005433	-0.185685

	f25	f26	f27	f28	f29	f30	target
f1	-0.014831	-0.023408	0.014925	-0.046011	-0.022659	-0.021592	-0.029275
f2	-0.070345	-0.025874	-0.014340	-0.124488	-0.051485	-0.051265	0.074944
f3	0.047242	-0.017016	0.006747	-0.007485	0.050156	-0.001757	-0.021559
f4	-0.133689	0.510890	-0.033239	0.758448	-0.062255	0.300410	-0.038915
f5	-0.027092	0.031412	-0.014426	0.032273	-0.029227	-0.003007	-0.031068
f6	-0.359910	-0.089290	0.009167	-0.501331	-0.067606	0.145092	0.098351
f7	0.021722	0.112017	-0.046281	0.076632	0.280334	0.360535	0.013298
f8	0.861051	-0.400300	-0.083301	0.450423	0.464902	0.249397	-0.263550
f9	-0.719886	0.116624	0.022729	-0.424876	-0.547663	-0.085459	0.232841
f10	-0.098985	-0.357880	0.022507	-0.235732	-0.575020	-0.403717	0.027917
f11	-0.061222	0.028194	0.003884	-0.015077	-0.050322	-0.022267	-0.017477
f12	-0.658421	0.485987	0.090803	-0.346706	-0.163737	-0.229005	0.211565
f13	-0.069752	0.044345	-0.005787	0.004713	-0.038054	0.008078	0.020607
f14	-0.505201	0.065391	0.020350	0.010560	-0.699009	-0.162252	0.089042
f15	-0.000284	-0.000226	-0.012133	0.014637	0.008230	0.033450	-0.047126
f16	0.671408	-0.829065	-0.023541	-0.402962	-0.031436	-0.484739	-0.057260

f17	-0.685310	0.708205	0.070774	0.066024	-0.074923	0.094895	0.121584
f18	0.033469	-0.041340	0.020484	0.001145	-0.002363	0.002775	0.015937
f19	0.585611	-0.591085	-0.100719	0.130312	0.196150	0.344720	-0.186492
f20	-0.620590	-0.097716	0.063440	-0.528086	-0.768182	-0.459459	0.192935
f21	0.309380	-0.189355	-0.090447	0.501772	-0.009493	0.374931	-0.152291
f22	0.564010	-0.820138	-0.060902	-0.174654	-0.156438	-0.171429	-0.112013
f23	0.057124	-0.075273	0.001461	-0.038062	-0.002346	-0.037887	-0.005433
f24	0.346269	0.271854	-0.003416	0.594182	0.446892	0.225745	-0.185685
f25	1.000000	-0.605009	-0.067597	0.098985	0.635272	0.121089	-0.249061
f26	-0.605009	1.000000	0.025318	0.622743	0.091655	0.437745	0.052959
f27	-0.067597	0.025318	1.000000	-0.056367	-0.064486	-0.085022	0.019441
f28	0.098985	0.622743	-0.056367	1.000000	0.450617	0.715962	-0.172652
f29	0.635272	0.091655	-0.064486	0.450617	1.000000	0.676683	-0.237301
f30	0.121089	0.437745	-0.085022	0.715962	0.676683	1.000000	-0.174820
target	-0.249061	0.052959	0.019441	-0.172652	-0.237301	-0.174820	1.000000

[31 rows x 31 columns]

In [7

```
print(data.isnull().sum())
```

```
f1        0
f2        0
f3        0
f4        0
f5        0
f6        0
f7        0
f8        0
f9        0
f10       0
f11       0
f12       0
f13       0
f14       0
f15       0
f16       0
f17       0
f18       0
f19       0
f20       0
f21       0
f22       0
f23       0
f24       0
f25       0
f26       0
f27       0
f28       0
f29       0
f30       0
target    0
dtype: int64
```

In [14]

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

results_df = pd.DataFrame({
    'Metric': ['Training Accuracy', 'Test Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Value': [train_accuracy, test_accuracy, precision, recall, f1]
})

print(results_df)
```

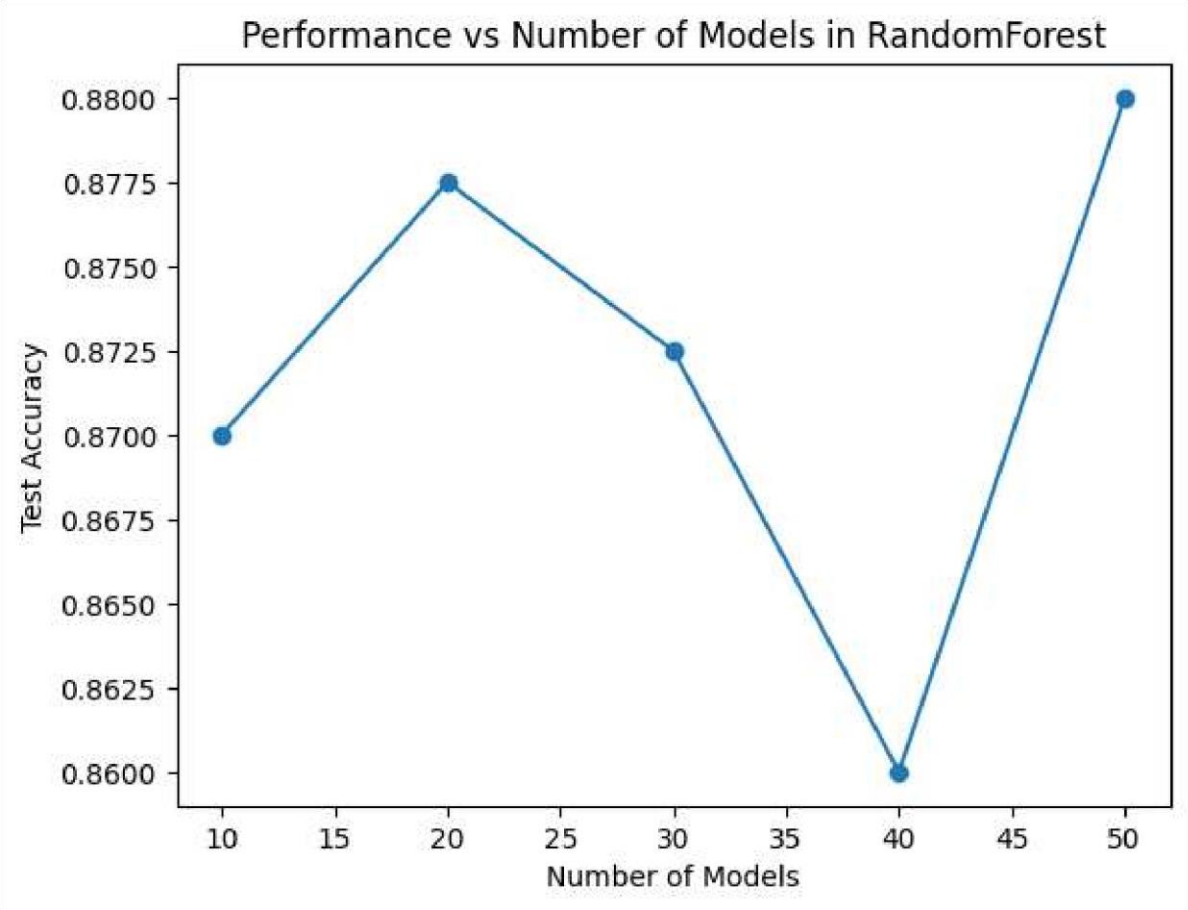
```
             Metric     Value
0  Training Accuracy  1.000000
1      Test Accuracy  0.870000
2          Precision  0.900000
3             Recall  0.653226
4           F1 Score  0.757009
```

In [10]:
```
import matplotlib.pyplot as plt

num_models = [10, 20, 30, 40, 50]
accuracy_scores = []

for n in num_models:
    model = RandomForestClassifier(n_estimators=n)
    model.fit(X_train, y_train)
    accuracy = accuracy_score(y_test, model.predict(X_test))
    accuracy_scores.append(accuracy)

plt.plot(num_models, accuracy_scores, marker='o')
plt.xlabel('Number of Models')
plt.ylabel('Test Accuracy')
plt.title('Performance vs Number of Models in RandomForest')
plt.show()
```



When the classifier reaches 50, the performance of the model is optimal, and the accuracy of the training set reaches 0.88, resul
performance

Part B

In [11]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = explained_variance_ratio.cumsum()

plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.xlabel('Number of Dimensions')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA: Cumulative Explained Variance')
plt.show()
```