Problem1

1. **MLE**

$$P(X) = \left(\frac{\#class\_counts}{\#total\_samples}\right)$$

$$P(Y|X) = \left(\frac{(\#class\_counts|X = x)}{(\#total\_samples|X = x)}\right)$$

2. **K2**, $\alpha = 1$

$$P(X) = \left(\frac{\alpha + \#class\_counts}{\sum \alpha + \#total\_samples}\right)$$

$$P(Y|X) = \left(\frac{\alpha + (\#class\_counts|X = x)}{\sum \alpha + (\#total\_samples|X = x)}\right)$$

3. **Bde**, |D'| = 6, uniform probability

$$P(X) = \left(\frac{3 + \#class\_counts}{|D'| + \#total\_samples}\right)$$

$$P(Y|X) = \left(\frac{2 + (\#class\_counts|X = x)}{|D'| + (\#total\_samples|X = x)}\right)$$

(MLE):

MLE for P(X)
P(X=T) = (10 + 20 + 30) / (10 + 20 + 30 + 40 + 50) = 60/150 = 0.4

P(X=F) = (40 + 50) / 150 = 0.6

MLE for P(Y|X)
P(Y=R|X=T) = (10 + 20) / (10 + 20 + 30) = 30/60 = 0.5

P(Y=B|X=T) = 30/60 = 0.5

P(Y=R|X=F) = 40 / (40 + 50) = 0.8

P(Y=B|X=F) = 50 / 90 = 0.556

MLE for P(Z|Y)
P(Z=T|Y=R) = 10 / (10 + 20 + 40) = 10/70 = 0.143

P(Z=F|Y=R) = (20 + 40) / 70 = 0.857

P(Z=T|Y=B) = (30 + 50) / 80 = 1

P(Z=F|Y=B) = 0 / 80 = 0

2. K2 appraoch

P(X=T) =(60 +1) / (150 +2) =0.4013

P(X=F) = 91/ 152 = 0.5987

K2 for P(Y|X)

Y/X	T	F
R	30 +1 =31	40+1 =41
B	30 +1 =31	50+1 = 51

P(Y=R|X=T) = 30 +1/ 60 +2 = 31/62 = 0.5

P(Y=B|X=T) = 31/62 = 0.5

P(Y=R|X=F) = 41 / 92 = 0.446

P(Y=B|X=F) = 51 / 92 = 0.554

K2for P(Z|Y)

Z/Y	R	B
T	10 +1 =31	80+1 =81
F	60 +1 =61	0+1 = 1

P(Z=T|Y=R) = 11 / 72 = 0.153

P(Z=F|Y=R) = 61 / 72 = 0.847

P(Z=T|Y=B) = 81 / 82 = 0.988

P(Z=F|Y=B) = 1 / 82 = 0.012

3. BDE
so, for |D’| = 12 ,

X {T.F} = {0.5,0.5} (uniform dist prior)
XαT = P(X=T) * |D’| = 0.5 *12 =6
XαF = P(X=F) * |D’| = 0.5 *12 =6

Y {R.B} = {0.5,0.5}
αR = P(Y=R) * |D’| = 0.5 *12 =6
αB = P(Y=B) * |D’| = 0.5 *12 =6

Z {T.F} = {0.5,0.5}
ZαT = P(Z=T) * |D’| = 0.5 *12 =6
ZαF = P(Z=F) * |D’| = 0.5 *12 =6

P(X=T) =(60 +6) / (150 +6+6) = 66/162 =0.407

P(X=F) = 90 +6/ 150 +6+6= 96/162 =0.593

Y/X	T	F
R	30 +6 =36	40+6 =46
B	30 +6 =36	50+6 = 56
total #	72	102

P(Y=R|X=T) = 36/ 72 = = 0.5

P(Y=B|X=T) = 36/ 72 = 0.5

P(Y=R|X=F) = 46 / 102 = 0.451

P(Y=B|X=F) = 51 / 102 = 0.5

Z/Y	R	B
T	10 +6 =16	80+6 =86
F	60 +6 =66	0+6 = 6
total #	82	92

P(Z=T|Y=R) = 16 / 82 = 0.195

P(Z=F|Y=R) = 66 / 82 = 0.805

P(Z=T|Y=B) = 86 / 92 = 0.9348

P(Z=F|Y=B) = 6 / 92 = 0.065

Problem 2 .

Problem 2

[16]
```
# step 1.  Prepare dataset for pgmpy
# Load the dataset into IPython notebook or script. Discretize continuous values. Use median
# value to separate continuous variables into ‘high’ and ‘low’ categories.
import pandas as pd

data = pd.read_csv("/content/auto-mpg.csv")

# Calculate median for continuous columns
selected_cols = ['mpg', 'displacement', 'horsepower', 'weight', 'acceleration']
medians = data[selected_cols].median()

# Classify values as "low" or "high" based on the median
for col in selected_cols:
    median = medians[col]
    data[col] = data[col].apply(lambda x: 'low' if x < median else 'high')

print(data)
```

```
    mpg  cylinders displacement horsepower weight acceleration  model year  \
0   low          8         high       high   high          low          70
1   low          8         high       high   high          low          70
2   low          8         high       high   high          low          70
3   low          8         high       high   high          low          70
4   low          8         high       high   high          low          70
..  ...        ...          ...        ...    ...          ...         ...
94  high         4          low        low    low          low          74
95  high         4          low       high    low         high          74
96  high         4          low        low    low         high          74
97  high         4          low        low    low         high          74
98  high         4          low        low    low         high          75

    origin                   car name
0        1  chevrolet chevelle malibu
1        1          buick skylark 320
2        1         plymouth satellite
3        1              amc rebel sst
4        1                ford torino
..     ...                        ...
94       2               fiat 124 tc
95       3               honda civic
96       3                    subaru
97       2                 fiat x1.9
98       3            toyota corolla

[99 rows x 9 columns]
```

```
!pip install pgmpy
```

```
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-ru
```

[17]

```
#step 2 a. [15 pts] Perform structure learning using conditional independence tests (PC algorithm)
from pgmpy.estimators import PC

# Structure learning using the PC algorithm
pc_estimator = PC(data)
estimated_model_pc = pc_estimator.estimate()
print("PC Algorithm Estimated Model:")
print(estimated_model_pc.edges())

# Building skeleton and deriving PDAG
estimator = PC(data)
skeleton, separating_sets = estimator.build_skeleton(significance_level=0.01)
print("Undirected edges: ", skeleton.edges())

pdag = estimator.skeleton_to_pdag(skeleton, separating_sets)
print("PDAG edges:       ", pdag.edges())

model = pdag.to_dag()
print("DAG edges:        ", model.edges())
```

```
Working for n conditional variables: 4:  80%  4/5 [00:01<00:00,  1.74it/s]
PC Algorithm Estimated Model:
[('acceleration', 'cylinders'), ('mpg', 'displacement'), ('displacement', 'cylinders'), ('origin', 'displacement')]
Working for n conditional variables: 4:  80%  4/5 [00:01<00:00,  2.76it/s]
Undirected edges:  [('mpg', 'displacement'), ('cylinders', 'displacement'), ('cylinders', 'acceleration'), ('displacement', 'origin')]
PDAG edges:        [('acceleration', 'cylinders'), ('mpg', 'displacement'), ('displacement', 'cylinders'), ('origin', 'displacement')]
DAG edges:         [('acceleration', 'cylinders'), ('mpg', 'displacement'), ('displacement', 'cylinders'), ('origin', 'displacement')]
```

```
[27] # Structure learning using Hill Climb Search with different scores
     # BIC score
     from pgmpy.estimators import HillClimbSearch, BicScore, K2Score, BDeuScore

     # Initialize Hill Climb Search with BIC score
     hillClimb_bic = HillClimbSearch(data)
     estimated_model_bic = hillClimb_bic.estimate(scoring_method=BicScore(data))

     # Initialize Hill Climb Search with BDeu score
     hillClimb_bdeu = HillClimbSearch(data)
     estimated_model_bdeu = hillClimb_bdeu.estimate(scoring_method=BDeuScore(data, equivalent_sample_size=5))

     # Initialize Hill Climb Search with K2 score
     hillClimb_k2 = HillClimbSearch(data)
     estimated_model_k2 = hillClimb_k2.estimate(scoring_method=K2Score(data))

     # Print the estimated model obtained using BIC score
     print("Hill Climb Search BIC Score for our Estimated Model:")
     print(estimated_model_bic.edges())

     # Print the estimated model obtained using BDeu score
     print("Hill Climb Search BDeu Score for our Estimated Model:")
     print(estimated_model_bdeu.edges())

     # Print the estimated model obtained using K2 score
     print("Hill Climb Search K2 Score for our Estimated Model:")
     print(estimated_model_k2.edges())
```

```
Output-
Hill Climb Search BIC Score for our Estimated Model:
[('mpg', 'weight'), ('mpg', 'horsepower'), ('cylinders', 'displacement'),
('cylinders', 'mpg'), ('cylinders', 'acceleration'), ('displacement',
'origin'), ('displacement', 'model year')]

Hill Climb Search BDeu Score for our Estimated Model:
[('mpg', 'horsepower'), ('mpg', 'model year'), ('cylinders', 'mpg'),
('cylinders', 'weight'), ('cylinders', 'acceleration'), ('cylinders',
'model year'), ('displacement', 'cylinders'), ('displacement', 'origin'),
('weight', 'mpg'), ('acceleration', 'horsepower'), ('origin', 'weight'),
('car name', 'displacement')]

Hill Climb Search K2 Score for our Estimated Model:
[('mpg', 'weight'), ('mpg', 'horsepower'), ('mpg', 'car name'), ('mpg',
'model year'), ('cylinders', 'displacement'), ('cylinders', 'mpg'),
('cylinders', 'acceleration'), ('cylinders', 'car name'), ('cylinders',
'model year'), ('displacement', 'origin'), ('displacement', 'car name'),
('displacement', 'model year'), ('displacement', 'weight'), ('horsepower',
'car name'), ('horsepower', 'model year'), ('weight', 'car name'),
('weight', 'model year'), ('acceleration', 'car name'), ('acceleration',
'model year'), ('acceleration', 'horsepower'), ('model year', 'car name'),
('origin', 'car name'), ('origin', 'model year')]
```

```python
# algorithm. Print out CPDs and local independencies of the network.
# b. [22.5 pts]  parameter estimation using the Expectation Maximization
# algorithm. Print out CPDs and local independencies of the network.
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator, ExpectationMaximization

bayesian_model_bic = BayesianNetwork(estimated_model_bic.edges())
nodes_list_bic = list(bayesian_model_bic.nodes)
mle_estimator_bic = MaximumLikelihoodEstimator(bayesian_model_bic, df)
cpds_mle_bic = {variable: mle_estimator_bic.estimate_cpd(variable) for variable in nodes_list_bic}

# Maximum Likelihood Estimation (MLE) Conditional Probability Distributions (CPDs)
print("Maximum Likelihood Estimation (MLE) CPDs:")
for cpd in cpds_mle_bic.values():
    print(cpd)

print("\nLocal Independencies of the Bayesian Network (BIC Score):")
local_independencies_bic = bayesian_model_bic.local_independencies(nodes_list_bic)
for assertion in local_independencies_bic.get_assertions():
    print(assertion)
em_estimator_bic = ExpectationMaximization(bayesian_model_bic, df)
# Conditional Probability Distributions (CPDs) obtained from Expectation Maximization (EM)
print("\nExpectation Maximization (EM) CPDs:")
for cpd in em_estimator_bic.get_parameters():
    print(cpd)

print("\nLocal Independencies of the Bayesian Network (EM):")
local_independencies_em = bayesian_model_bic.local_independencies(nodes_list_bic)
for assertion in local_independencies_em.get_assertions():
    print(assertion)
```

Output-

Maximum Likelihood Estimation (MLE) CPDs:

cylinders	cylinders(3)	cylinders(4)	cylinders(6)	cylinders(8)
mpg(high)	0.0	0.9791666666666666	0.25	0.0
mpg(low)	1.0	0.020833333333333332	0.75	1.0

mpg	mpg(high)	mpg(low)
weight(high)	0.06	0.9591836734693877
weight(low)	0.94	0.04081632653061224

mpg	mpg(high)	mpg(low)

horsepower(high)	0.06	0.9591836734693877
horsepower(low)	0.94	0.04081632653061224

cylinders(3)	0.010101
cylinders(4)	0.484848
cylinders(6)	0.121212
cylinders(8)	0.383838

cylinders	cylinders(3)	cylinders(4)	cylinders(6)	cylinders(8)
displacement(high)	0.0	0.0	1.0	1.0
displacement(low)	1.0	1.0	0.0	0.0

cylinders	cylinders(3)	...	cylinders(6)	cylinders(8)
acceleration(high)	0.0	...	0.8333333333333334	0.07894736842105263
acceleration(low)	1.0	...	0.16666666666666666	0.9210526315789473

displacement	displacement(high)	displacement(low)
origin(1)	1.0	0.24489795918367346
origin(2)	0.0	0.3877551020408163
origin(3)	0.0	0.3673469387755102

displacement	displacement(high)	displacement(low)
model year(70)	0.44	0.12244897959183673
model year(71)	0.3	0.24489795918367346

model year(72)	0.26	0.30612244897959184
model year(74)	0.0	0.30612244897959184
model year(75)	0.0	0.02040816326530612

Local Independencies of the Bayesian Network (BIC Score):
(mpg ⊥ origin, model year, displacement, acceleration | cylinders)
(weight ⊥ origin, model year, acceleration, cylinders, displacement, horsepower | mpg)
(horsepower ⊥ origin, model year, weight, acceleration, cylinders, displacement | mpg)
(displacement ⊥ mpg, weight, acceleration, horsepower | cylinders)
(acceleration ⊥ origin, model year, weight, mpg, displacement, horsepower | cylinders)
(origin ⊥ model year, weight, acceleration, cylinders, mpg, horsepower | displacement)
(model year ⊥ origin, weight, acceleration, cylinders, mpg, horsepower | displacement)

Expectation Maximization (EM) CPDs:

0%
0/100 [00:00<?, ?it/s]

displacement	displacement(high)	displacement(low)
origin(1)	1.0	0.24489795918367346
origin(2)	0.0	0.3877551020408163
origin(3)	0.0	0.3673469387755102

cylinders(3)	0.010101
cylinders(4)	0.484848
cylinders(6)	0.121212
cylinders(8)	0.383838

displacement	displacement(high)	displacement(low)
model year(70)	0.44	0.12244897959183673
model year(71)	0.3	0.24489795918367346

model year(72)	0.26	0.30612244897959184
model year(74)	0.0	0.30612244897959184
model year(75)	0.0	0.02040816326530612

mpg	mpg(high)	mpg(low)
weight(high)	0.06	0.9591836734693877
weight(low)	0.94	0.04081632653061224

cylinders	cylinders(3)	...	cylinders(6)	cylinders(8)
acceleration(high)	0.0	...	0.8333333333333334	0.07894736842105263
acceleration(low)	1.0	...	0.16666666666666666	0.9210526315789473

cylinders	cylinders(3)	cylinders(4)	cylinders(6)	cylinders(8)
mpg(high)	0.0	0.9791666666666666	0.25	0.0
mpg(low)	1.0	0.020833333333333332	0.75	1.0

cylinders	cylinders(3)	cylinders(4)	cylinders(6)	cylinders(8)
displacement(high)	0.0	0.0	1.0	1.0
displacement(low)	1.0	1.0	0.0	0.0

mpg	mpg(high)	mpg(low)
horsepower(high)	0.06	0.9591836734693877
horsepower(low)	0.94	0.04081632653061224

Local Independencies of the Bayesian Network (EM):

(mpg $\perp$ origin, model year, displacement, acceleration | cylinders)
(weight $\perp$ origin, model year, acceleration, cylinders, displacement, horsepower | mpg)
(horsepower $\perp$ origin, model year, weight, acceleration, cylinders, displacement | mpg)
(displacement $\perp$ mpg, weight, acceleration, horsepower | cylinders)

(acceleration $\perp$ origin, model year, weight, mpg, displacement, horsepower | cylinders)
(origin $\perp$ model year, weight, acceleration, cylinders, mpg, horsepower | displacement)
(model year $\perp$ origin, weight, acceleration, cylinders, mpg, horsepower | displacement)

Summary-

We started by preparing our dataset, where we categorized continuous variables into "low" and "high" based on their median values. This helped us make our data more manageable for analysis.

Next, we performed structure learning using the PC algorithm to estimate our model's structure. This involved understanding conditional dependencies and deriving the skeleton and PDAG (Partially Directed Acyclic Graph) of our model.

After that, we used the Hill Climb Search algorithm with different scoring methods like BIC, BDeu, and K2 to estimate the structure of our model. Each scoring method gave us insights into the dependencies and relationships within our dataset.

For our final analysis, we examined the Maximum Likelihood Estimation (MLE) and Expectation Maximization (EM) results. MLE provided us with conditional probability distributions (CPDs) based on our model, while EM refined these estimates iteratively.

In comparing the scores, the choice of scoring method significantly influenced the complexity and accuracy of our learned model. The BIC score produced a more compact model with fewer dependencies, while the BDeu and K2 scores captured more relationships.